Creating a Question Service part 2

In Part 1, we set up our Question Service project. Now we'll implement the three essential functionalities we identified:

1. Generate quiz questions
2. Get questions by ID
3. Calculate score

1. Generate Quiz Questions

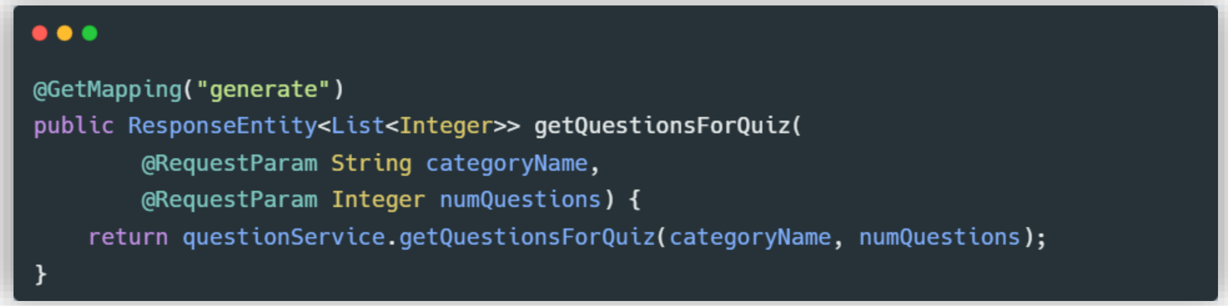
In our monolithic QuizApp, the QuizController generated quizzes directly. In our microservices architecture, the Quiz Service will request questions from our Question Service.

Instead of sending complete question objects, our Question Service will return only question IDs. This approach:

- Reduces initial data transfer
- Keeps the Question Service in control of question details
- Maintains clear separation of concerns

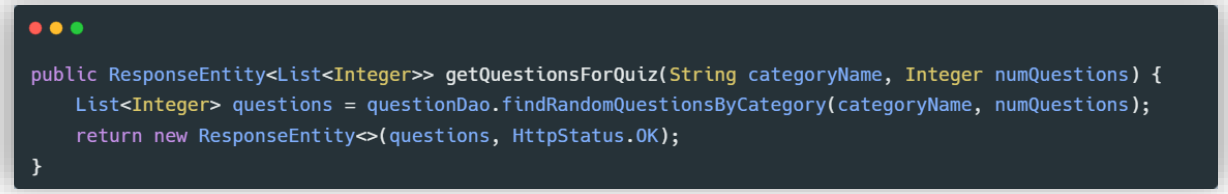
Example

Controller Layer:



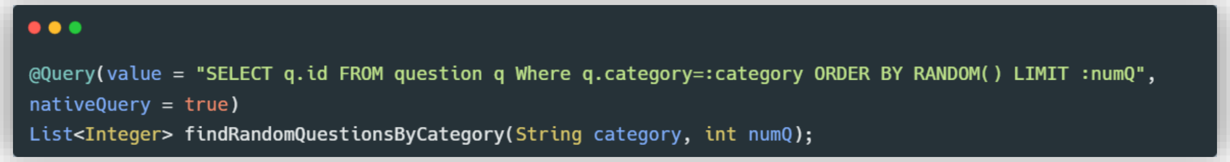
```java
@GetMapping("generate")
public ResponseEntity<List<Integer>> getQuestionsForQuiz(
        @RequestParam String categoryName,
        @RequestParam Integer numQuestions) {
    return questionService.getQuestionsForQuiz(categoryName, numQuestions);
}
```

Service Layer:



```java
public ResponseEntity<List<Integer>> getQuestionsForQuiz(String categoryName, Integer numQuestions) {
    List<Integer> questions = questionDao.findRandomQuestionsByCategory(categoryName, numQuestions);
    return new ResponseEntity<>(questions, HttpStatus.OK);
}
```

Repository Layer:



```java
@Query(value = "SELECT q.id FROM question q Where q.category=:category ORDER BY RANDOM() LIMIT :numQ",
nativeQuery = true)
List<Integer> findRandomQuestionsByCategory(String category, int numQ);
```

2. Get Questions by ID

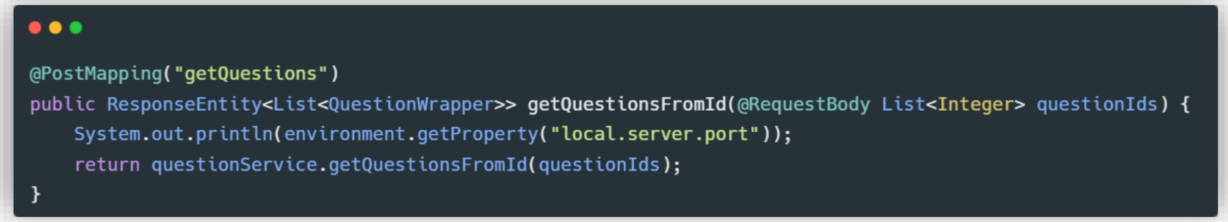
Once the Quiz Service has question IDs, it needs to retrieve the actual question content when presenting the quiz to users.

We return a QuestionWrapper instead of the complete Question entity. This wrapper:

- Contains only the information needed to display questions (title and options)
- Excludes sensitive data like correct answers
- Provides a cleaner data transfer object for the client

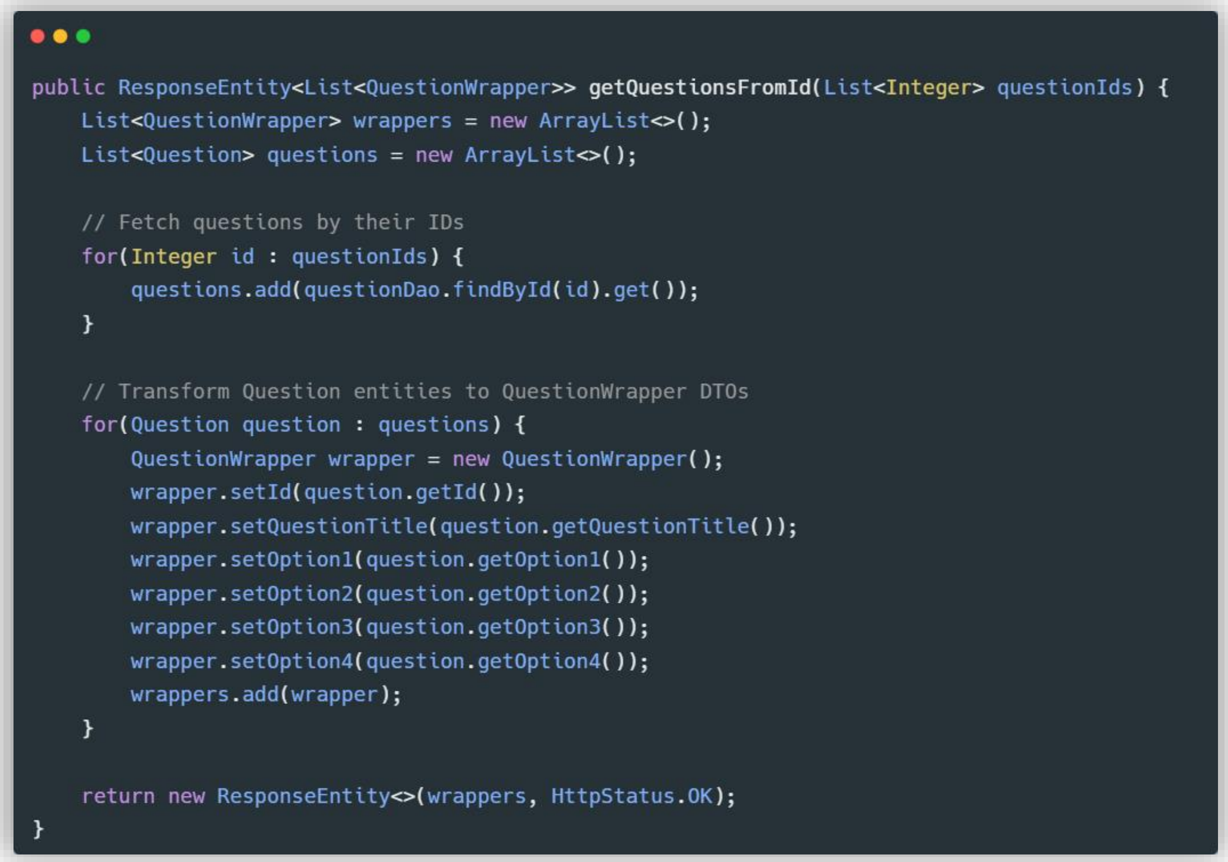
Example:

Controller Layer:



```java
@PostMapping("getQuestions")
public ResponseEntity<List<QuestionWrapper>> getQuestionsFromId(@RequestBody List<Integer> questionIds) {
    System.out.println(environment.getProperty("local.server.port"));
    return questionService.getQuestionsFromId(questionIds);
}
```

Service Layer:



```java
public ResponseEntity<List<QuestionWrapper>> getQuestionsFromId(List<Integer> questionIds) {
    List<QuestionWrapper> wrappers = new ArrayList<>();
    List<Question> questions = new ArrayList<>();

    // Fetch questions by their IDs
    for(Integer id : questionIds) {
        questions.add(questionDao.findById(id).get());
    }

    // Transform Question entities to QuestionWrapper DTOs
    for(Question question : questions) {
        QuestionWrapper wrapper = new QuestionWrapper();
        wrapper.setId(question.getId());
        wrapper.setQuestionTitle(question.getQuestionTitle());
        wrapper.setOption1(question.getOption1());
        wrapper.setOption2(question.getOption2());
        wrapper.setOption3(question.getOption3());
        wrapper.setOption4(question.getOption4());
        wrappers.add(wrapper);
    }

    return new ResponseEntity<>(wrappers, HttpStatus.OK);
}
```

3. Calculate Score

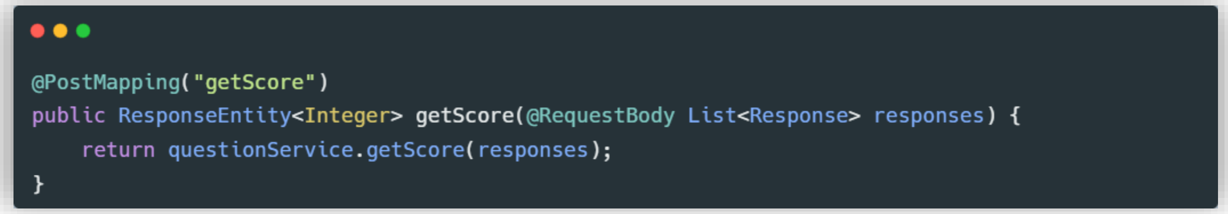
When a user completes a quiz, the Quiz Service needs to calculate the score based on their responses.

Score calculation happens in the Question Service because:

- Only the Question Service has access to the correct answers
- It centralizes validation logic in one location
- It reduces the need to expose sensitive data (correct answers) to other services

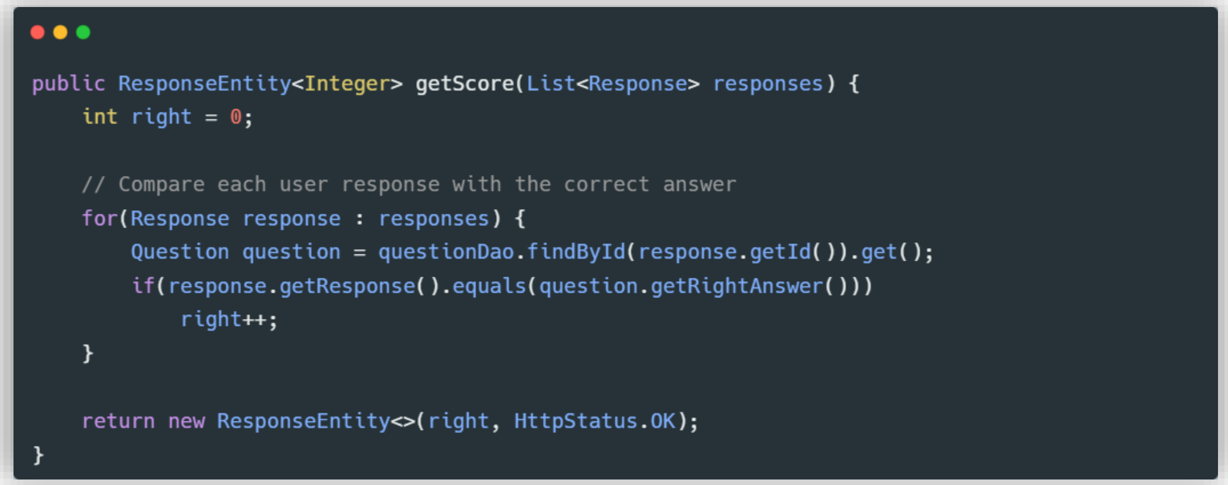
Example:

Controller Layer:



```java
@PostMapping("getScore")
public ResponseEntity<Integer> getScore(@RequestBody List<Response> responses) {
    return questionService.getScore(responses);
}
```

Service Layer:



```java
public ResponseEntity<Integer> getScore(List<Response> responses) {
    int right = 0;

    // Compare each user response with the correct answer
    for(Response response : responses) {
        Question question = questionDao.findById(response.getId()).get();
        if(response.getResponse().equals(question.getRightAnswer()))
            right++;
    }

    return new ResponseEntity<>(right, HttpStatus.OK);
}
```

Our Question Service now has three key APIs that enable it to function independently while supporting the Quiz Service:

1. **Generate** - Returns random question IDs based on category and quantity

2. **GetQuestions** - Provides question details (without answers) for specific IDs

3. **GetScore** - Calculates scores by comparing user responses with correct answers

This implementation demonstrates important microservice principles:

- Each service has clear responsibilities
- Services exchange only necessary information
- Business logic stays with the data it operates on