Creating a Question Service part 3

👉 Testing and Scaling the Question Service

In this section, we will test our Question Service application and explore how to create multiple instances of the same service. One of the core advantages of microservices architecture is that we can scale each service individually based on demand, unlike monolithic applications where we must scale the entire application at once.

Here's the complete implementation of our Question Service that we'll be testing:

Controller:

```java
package com.telusko.questionservice.controller;

import com.telusko.questionservice.model.Question;
import com.telusko.questionservice.model.QuestionWrapper;
import com.telusko.questionservice.model.Response;
import com.telusko.questionservice.service.QuestionService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("question")
public class QuestionController {

    @Autowired
    QuestionService questionService;

    @Autowired
    Environment environment;

    @GetMapping("allQuestions")
    public ResponseEntity<List<Question>> getAllQuestions(){
        return questionService.getAllQuestions();
    }

    @GetMapping("category/{category}")
    public ResponseEntity<List<Question>> getQuestionsByCategory(@PathVariable String category){
        return questionService.getQuestionsByCategory(category);
    }

    @PostMapping("add")
    public ResponseEntity<String> addQuestion(@RequestBody Question question){
        return  questionService.addQuestion(question);
    }

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

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

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

Service:

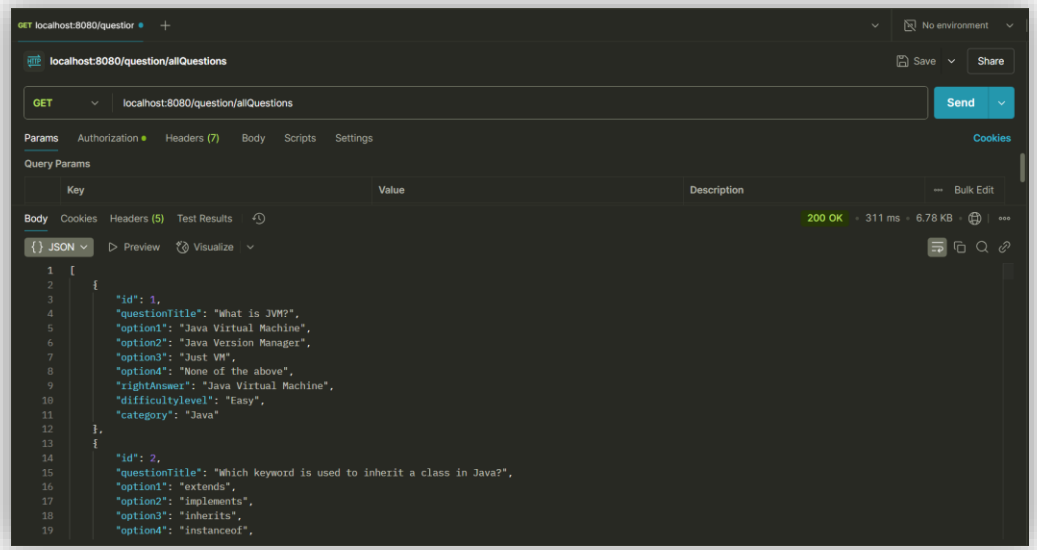
```java
package com.telusko.questionservice.service;

import com.telusko.questionservice.dao.QuestionDao;
import com.telusko.questionservice.model.Question;
import com.telusko.questionservice.model.QuestionWrapper;
import com.telusko.questionservice.model.Response;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
public class QuestionService {
    @Autowired
    QuestionDao questionDao;

    public ResponseEntity<List<Question>> getAllQuestions() {
        try {
            return new ResponseEntity<>(questionDao.findAll(), HttpStatus.OK);
        }catch (Exception e){
            e.printStackTrace();
        }
        return new ResponseEntity<>(new ArrayList<>(), HttpStatus.BAD_REQUEST);
    }

    public ResponseEntity<List<Question>> getQuestionsByCategory(String category) {
        try {
            return new ResponseEntity<>(questionDao.findByCategory(category),HttpStatus.OK);
        }catch (Exception e){
            e.printStackTrace();
        }
        return new ResponseEntity<>(new ArrayList<>(), HttpStatus.BAD_REQUEST);
    }

    public ResponseEntity<String> addQuestion(Question question) {
        questionDao.save(question);
        return new ResponseEntity<>("success",HttpStatus.CREATED);
    }

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

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

        for(Integer id : questionIds){
            questions.add(questionDao.findById(id).get());
        }

        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);
    }

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

        for(Response response : responses){
            Question question = questionDao.findById(response.getId()).get();
            if(response.getResponse().equals(question.getRightAnswer()))
                right++;
        }
        return new ResponseEntity<>(right, HttpStatus.OK);
    }
}
```

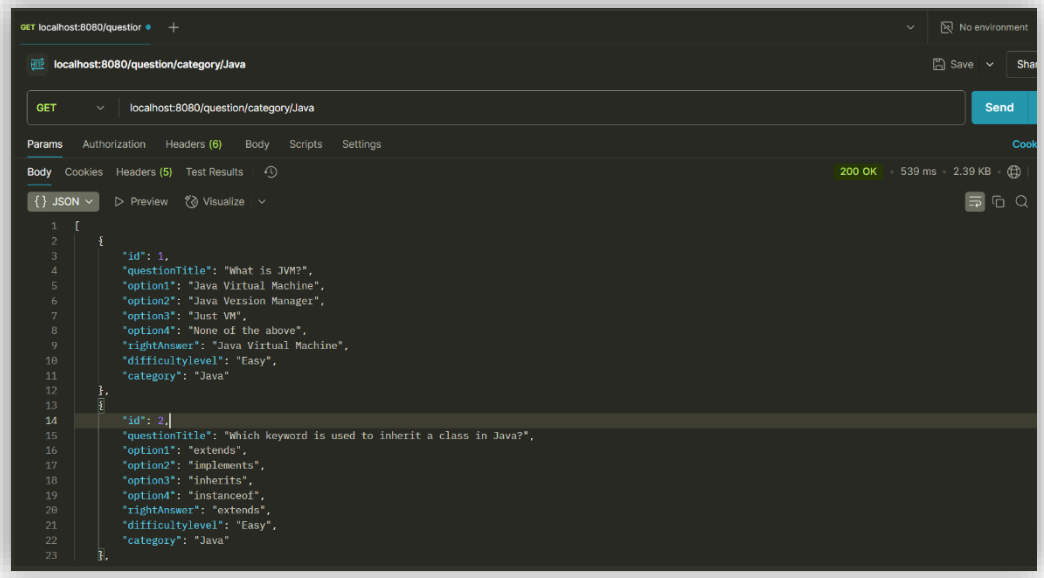
Repository:

```java
package com.telusko.questionservice.dao;

import com.telusko.questionservice.model.Question;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface QuestionDao extends JpaRepository<Question, Integer> {

    List<Question> findByCategory(String category);

    @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);
}
```

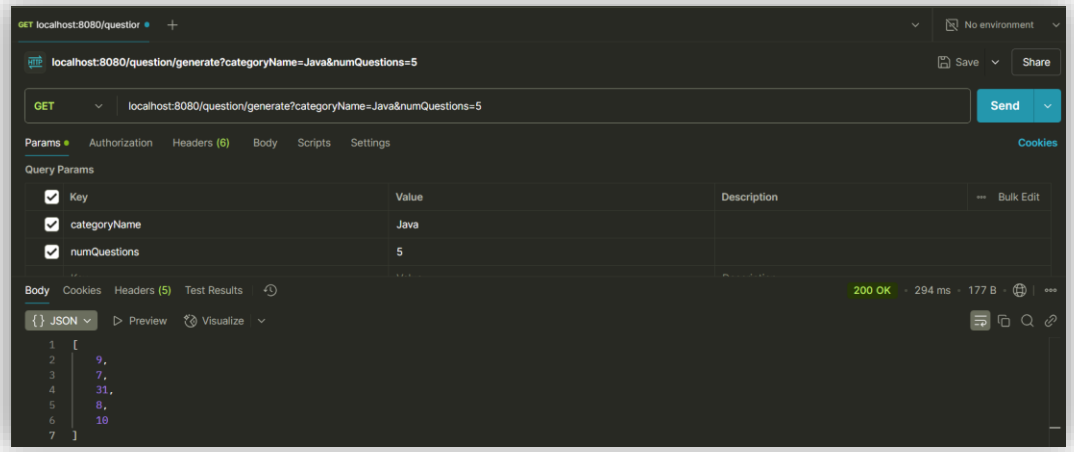
👉 Testing the Single Instance

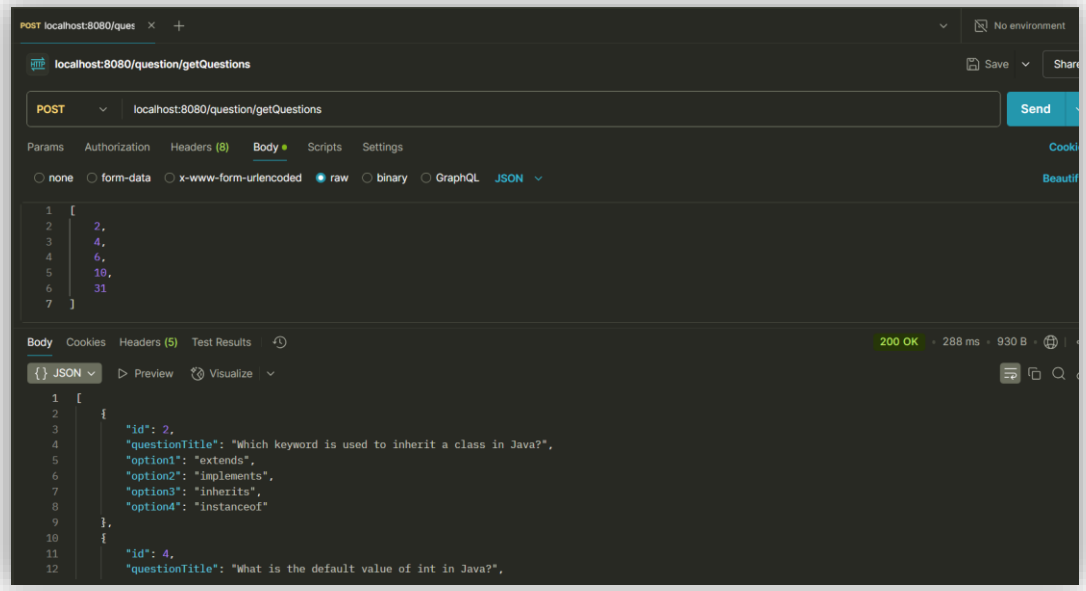
First, we'll run our Question Service on the default port 8080 and test it using Postman to ensure all endpoints are working correctly.

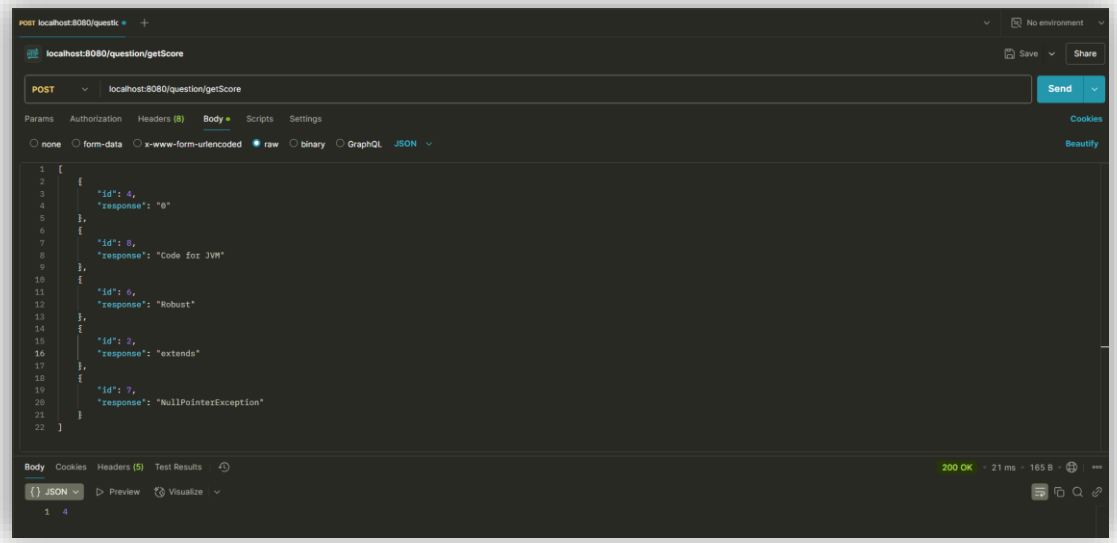





POST localhost:8080/questic
No environment
localhost:8080/question/add
Save
Share
POST
localhost:8080/question/add
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"questionTitle": "Which of the following is not a Java keyword?",
"option1": "static",
"option2": "Boolean",
"option3": "void",
"option4": "private",
"rightAnswer": "Boolean",
"difficultylevel": "Medium",
"category": "Java"
}
Body
Cookies
Headers (5)
Test Results
201 Created
685 ms
175 B
Raw
Preview
Visualize
success


GET localhost:8080/questior
No environment
localhost:8080/question/generate?categoryName=Java&numQuestions=5
Save
Share
GET
localhost:8080/question/generate?categoryName=Java&numQuestions=5
Send
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
categoryName
Java
numQuestions
5
Body
Cookies
Headers (5)
Test Results
200 OK
294 ms
177 B
{ } JSON
Preview
Visualize
[
9,
7,
31,
8,
10
]


POST localhost:8080/ques
No environment
localhost:8080/question/getQuestions
Save
POST
localhost:8080/question/getQuestions
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
[
2,
4,
6,
10,
31
]
Body
Cookies
Headers (5)
Test Results
200 OK
288 ms
930 B
{ } JSON
Preview
Visualize
[
{
"id": 2,
"questionTitle": "Which keyword is used to inherit a class in Java?",
"option1": "extends",
"option2": "implements",
"option3": "inherits",
"option4": "instanceof"
},
{
"id": 4,
"questionTitle": "What is the default value of int in Java?",

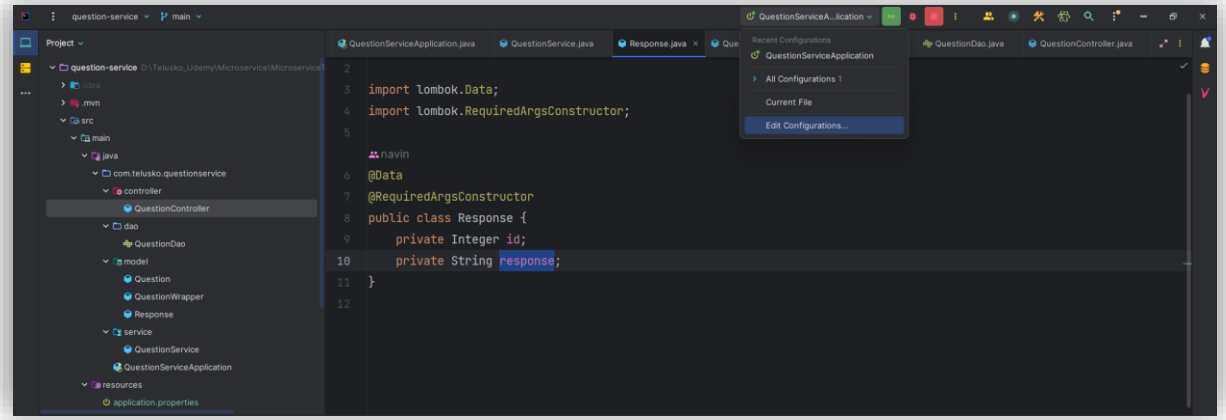


👉 Creating Multiple Instances

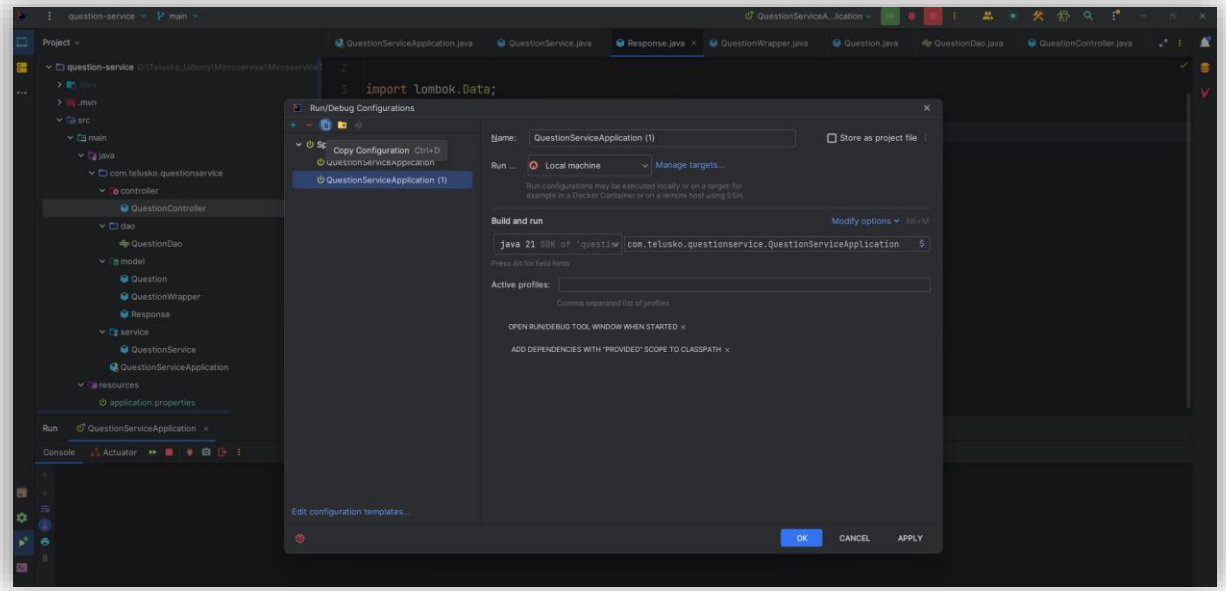
In microservices architecture, a key concept is the ability to create multiple instances of the same service for scaling and high availability. Each instance needs to run on a different port to avoid conflicts.

Steps to Create Multiple Instances in IntelliJ IDEA:

1. **Access Run Configurations:** Go to the top of IntelliJ and click "Edit Configurations"

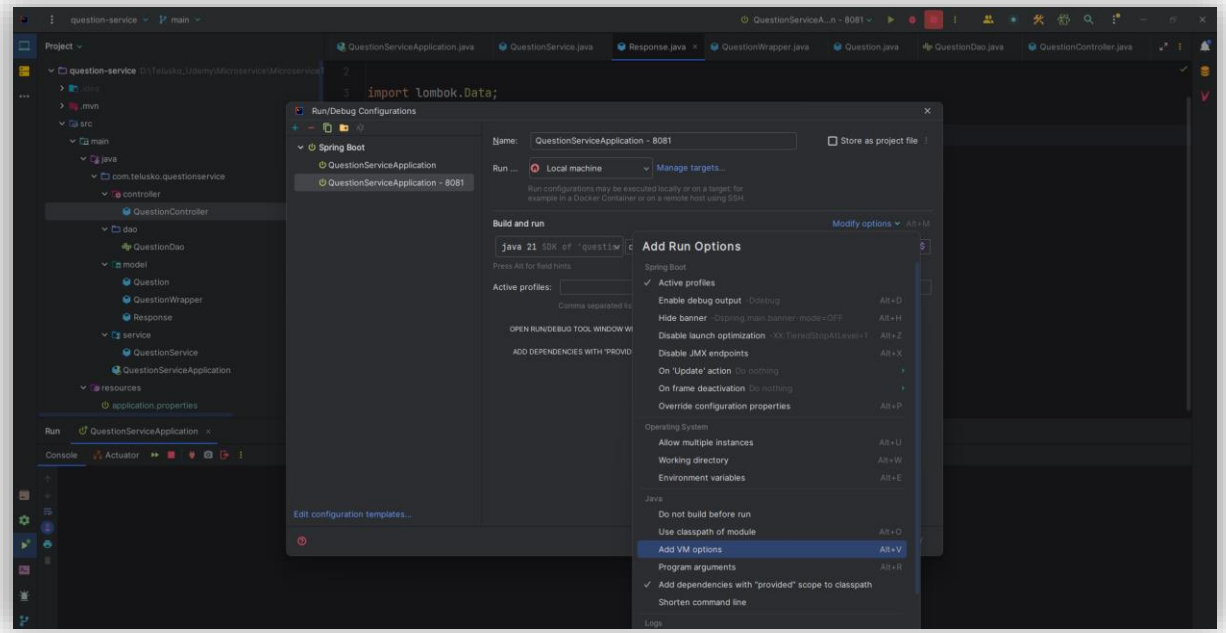


2. **Copy Existing Configuration:** Select the existing configuration and make a copy of it

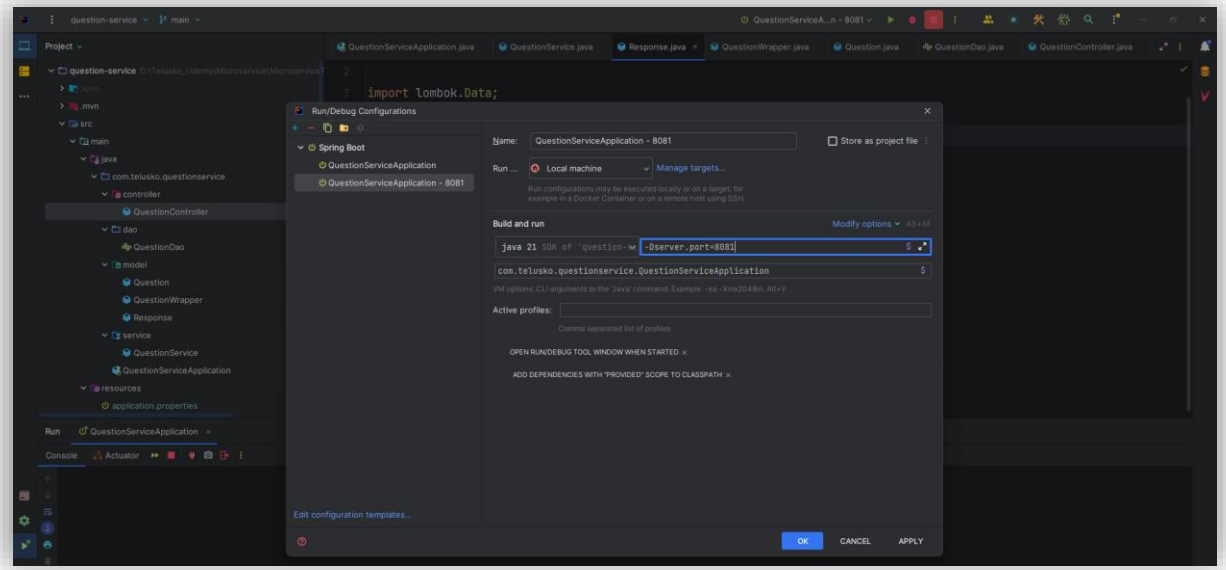


3. **Rename the New Configuration:** Give your new instance a meaningful name

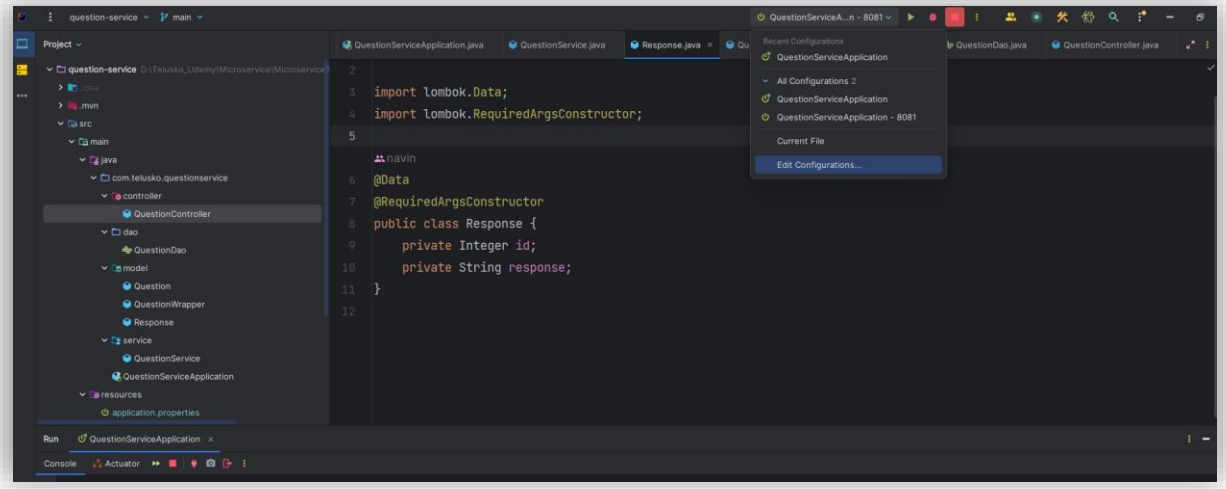
4. **Add VM Options:** Go to "Modify Options" and select "Add VM Options"



5. **Set Different Port Number:** Add the VM option **-Dserver.port=8081** to run this instance on port 8081

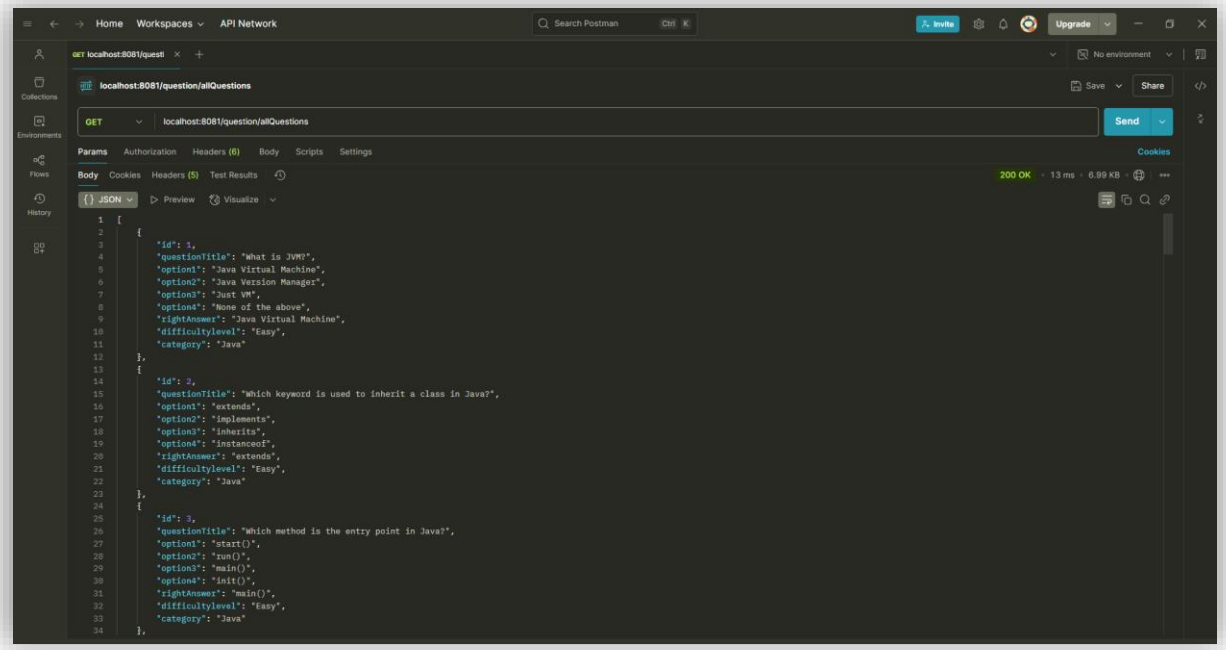


6. **Verify Configurations:** Now you should see two configuration options in the dropdown



With these configurations in place, you can now run both instances of the Question Service simultaneously. Each instance will operate independently on its assigned port, demonstrating how microservices can be scaled horizontally.

To verify that our second instance is running correctly, we can test it by sending a request to the same endpoint but on port 8081:



The successful response confirms that our second instance is functioning properly. Notice that in our controller, we included a line to print the port number, which helps us verify which instance is handling the request:

```
System.out.println(environment.getProperty("local.server.port"));
```

👉 Benefits of Running Multiple Instances

By running multiple instances of our Question Service, we've demonstrated a fundamental principle of microservices architecture: the ability to scale services independently. This approach provides:

- Increased capacity to handle more requests
- Higher availability (if one instance fails, others can still handle requests)
- Better resource utilization by scaling only the services that need it

In a production environment, these instances would typically be distributed across different servers and managed by orchestration tools, with a load balancer directing traffic between them.

The ability to scale individual services based on their specific needs is one of the primary advantages of microservices over monolithic applications, where scaling requires duplicating the entire application, regardless of which component needs additional resources.