Spring Boot Microservices Communication Example using WebClient

author: Ramesh Fadatare

MICROSERVICES SPRING BOOT

In the previous tutorial, we have seen the **Spring Boot Microservices Communication Example using RestTemplate**.

As of 5.0, the `RestTemplate` class is in maintenance mode and soon will be deprecated. So the Spring team recommended using `org.springframework.web.reactive.client.WebClient` that has a modern API and supports sync, async, and streaming scenarios.

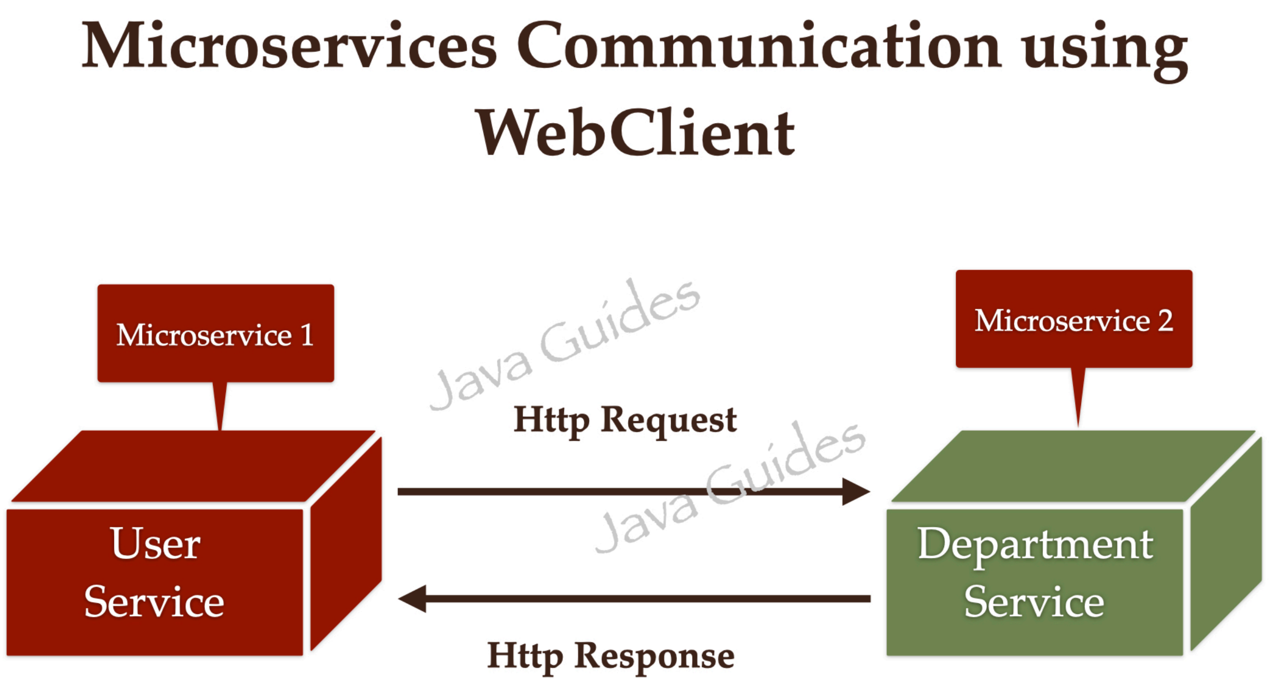
In this tutorial, we will learn how to use `WebClient` to make REST API calls (Synchronous communication) between multiple microservices.

`WebClient` is a non-blocking, reactive client to perform HTTP requests, exposing a fluent, reactive API over underlying HTTP client libraries such as Reactor Netty.

To use `WebClient` in our Spring boot project, we have to add `Spring WebFlux` dependency to the classpath.

What we will Build?

Well, we will create two microservices such as `department-service` and `user-service` and we will make a REST API call using `WebClient` from `user-service` to `department-service` to fetch a particular user department.



Prerequisites

Refer to the below tutorial to create `department-service` and `user-service` microservices.

We have created two microservices in the previous tutorial: **Spring Boot Microservices Communication Example using RestTemplate**.

Step 1: Add Spring WebFlux Dependency

Open the `pom.xml` file of the `user-service` project and add the below dependency:

```
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-webflux</artifactId>
		</dependency>
```

Step 2: Configure WebClient as Spring Bean

```
package net.javaguides.userservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.reactive.function.client.WebClient;

@SpringBootApplication
public class UserServiceApplication {

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

    @Bean
    public WebClient webClient(){
        return WebClient.builder().build();
    }
}
```

We have configured `WebClient` as Spring bean:

```
    @Bean
    public WebClient webClient(){
        return WebClient.builder().build();
    }
```

Step 3: Inject and Use WebClient to Call the REST API

Let's inject `WebClient` and use it to make a REST API call:

```
        DepartmentDto departmentDto = webClient.get()
                 .uri("http://localhost:8080/api/departments/" + user.getDepartmentId())
                         .retrieve()
                                 .bodyToMono(DepartmentDto.class)
                                         .block();
```

Here is the complete code of the `UserServiceImpl` class for your reference:

```
package net.javaguides.userservice.service.impl;

import lombok.AllArgsConstructor;
import net.javaguides.userservice.dto.DepartmentDto;
```

```java
import net.javaguides.userservice.dto.ResponseDto;
import net.javaguides.userservice.dto.UserDto;
import net.javaguides.userservice.entity.User;
import net.javaguides.userservice.repository.UserRepository;
import net.javaguides.userservice.service.UserService;
import org.springframework.stereotype.Service;
import org.springframework.web.reactive.function.client.WebClient;

@Service
@AllArgsConstructor
public class UserServiceImpl implements UserService {

    private UserRepository userRepository;
    private RestTemplate restTemplate;
    private WebClient webClient;

    private APIClient apiClient;

    @Override
    public User saveUser(User user) {
        return userRepository.save(user);
    }

    @Override
    public ResponseDto getUser(Long userId) {

        ResponseDto responseDto = new ResponseDto();
        User user = userRepository.findById(userId).get();
        UserDto userDto = mapToUser(user);

        DepartmentDto departmentDto = webClient.get()
                 .uri("http://localhost:8080/api/departments/" + user.getDepartmentId())
                         .retrieve()
                                 .bodyToMono(DepartmentDto.class)
                                         .block();
        responseDto.setUser(userDto);
        responseDto.setDepartment(departmentDto);

        return responseDto;
    }

    private UserDto mapToUser(User user){
        UserDto userDto = new UserDto();
        userDto.setId(user.getId());
        userDto.setFirstName(user.getFirstName());
        userDto.setLastName(user.getLastName());
        userDto.setEmail(user.getEmail());
        return userDto;
    }
}
```

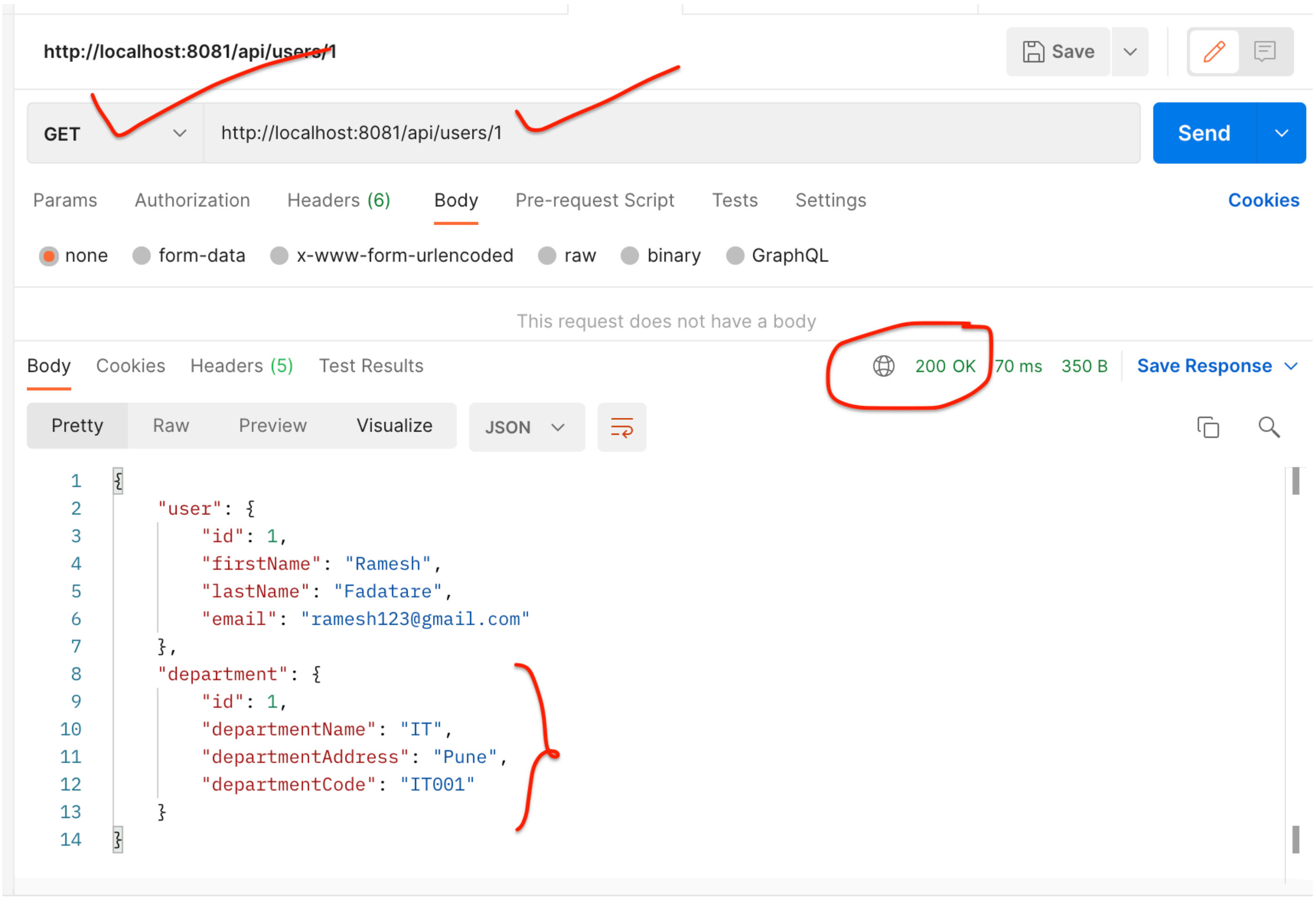
That's it. Now run both the Microservices and let's test.

Demo: Start Both Microservices

First, start the `department-service` project and then start a `user-service` project.

Once both the projects are up and running on different ports. Next, let's call the **Get User REST API** to test the `user-service` REST API call to the `department-service`.

Get User REST API:



GET http://localhost:8081/api/users/1

```
{
    "user": {
        "id": 1,
        "firstName": "Ramesh",
        "lastName": "Fadatare",
        "email": "ramesh123@gmail.com"
    },
    "department": {
        "id": 1,
        "departmentName": "IT",
        "departmentAddress": "Pune",
        "departmentCode": "IT001"
    }
}
```

Note that the response contains a Department for a User. This demonstrates that we have successfully made a REST API call from `user-service` to `department-service` using `WebClient`.

Conclusion

In this tutorial, we learned how to use `WebClient` to make REST API calls (Synchronous communication) between multiple microservices.