(/)

Guide to Spring 5 WebFlux

Last updated: October 3, 2023

Written by: baeldung (https://www.baeldung.com/author/baeldung)

HTTP Client-Side (https://www.baeldung.com/category/http)

Reactive (https://www.baeldung.com/category/reactive)

Spring Web (https://www.baeldung.com/category/spring/spring-web)

reference **Spring 5 (https://www.baeldung.com/tag/spring-5)** **WebClient (https://www.baeldung.com/tag/webclient)**

1. Overview

Spring 5 includes Spring WebFlux, which provides reactive programming support for web applications.

In this tutorial, we'll create a small reactive REST application using the reactive web components *RestController* and *WebClient*.

We'll also look at how to secure our reactive endpoints using Spring Security.

Further reading:

Spring 5 WebClient (/spring-5-webclient)

Discover Spring 5's WebClient - a new reactive RestTemplate alternative.

Read more (/spring-5-webclient) →

Handling Errors in Spring WebFlux (/spring-webflux-errors)

Have a look at different methods to gracefully handle errors in Spring Webflux.

Read more (/spring-webflux-errors) →

Introduction to the Functional Web Framework in Spring 5 (/spring-5-functional-web)

A quick and practical guide to the new Functional Web Framework in Spring 5

Read more (/spring-5-functional-web) →

2. Spring WebFlux Framework

Spring WebFlux internally uses Project Reactor (http://projectreactor.io/) and its publisher implementations, *Flux* (https://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html) and *Mono* (https://projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html).

The new framework supports two programming models:

- Annotation-based reactive components
- Functional routing and handling

We'll focus on the annotation-based reactive components, as we already explored the functional style – routing and handling (/spring-5-functional-web) in another tutorial.

3. Dependencies

Let's start with the *spring-boot-starter-webflux* dependency, which pulls in all other required dependencies:

- *spring-boot* and *spring-boot-starter* for basic Spring Boot application setup
- *spring-webflux* framework
- *reactor-core* that we need for reactive streams and also *reactor-netty*

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
    <version>3.1.2</version>
</dependency>
```

The latest spring-boot-starter-webflux (https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-webflux) can be downloaded from Maven Central.

4. Reactive REST Application

Now we'll build a very simple reactive REST *EmployeeManagement* application using Spring WebFlux:

- Use a simple domain model – *Employee* with an *id* and a *name* field
- Build a REST API with a *RestController* to publish *Employee* resources as a single resource and as a collection
- Build a client with *WebClient* to retrieve the same resource
- Create a secured reactive endpoint using *WebFlux* and Spring Security

5. Reactive *RestController*

Spring WebFlux supports annotation-based configurations in the same way as the Spring Web MVC framework.

To begin with, **on the server, we create an annotated controller that publishes a reactive stream of the *Employee* resource.**

Let's create our annotated *EmployeeController*:

```
@RestController
@RequestMapping("/employees")
public class EmployeeController {

    private final EmployeeRepository employeeRepository;

    // constructor...
}
```

EmployeeRepository can be any data repository that supports non-blocking reactive streams.

5.1. Single Resource

Then let's create an endpoint in our controller that publishes a single *Employee resource*:

```
@GetMapping("/{id}")
public Mono<Employee> getEmployeeById(@PathVariable String id) {
    return employeeRepository.findEmployeeById(id);
}
```

We wrap a single *Employee* resource in a *Mono* because we return at most one employee.

5.2. Collection Resource

We also add an endpoint that publishes the collection resource of all *Employees*:

```
@GetMapping
public Flux<Employee> getAllEmployees() {
    return employeeRepository.findAllEmployees();
}
```

For the collection resource, we use a *Flux* of type *Employee* since that's the publisher for *0*..n elements.

6. Reactive Web Client

WebClient (https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html#webflux-client), introduced in Spring 5, is a non-blocking client with support for reactive streams.

We can use *WebClient* to create a client to retrieve data from the endpoints provided by the *EmployeeController*.

Let's create a simple *EmployeeWebClient*:

```
public class EmployeeWebClient {

    WebClient client = WebClient.create("http://localhost:8080");

    // ...
}
```

Here we have created a *WebClient* using its factory method *create*. It'll point to *localhost:8080*, so we can use relative URLs for calls made by this client instance.

6.1. Retrieving a Single Resource

To retrieve a single resource of type *Mono* from endpoint */employee/{id}*:

```
Mono<Employee> employeeMono = client.get()
  .uri("/employees/{id}", "1")
  .retrieve()
  .bodyToMono(Employee.class);

employeeMono.subscribe(System.out::println);
```

6.2. Retrieving a Collection Resource

Similarly, to retrieve a collection resource of type *Flux* from endpoint */employees*:

```
Flux<Employee> employeeFlux = client.get()
  .uri("/employees")
  .retrieve()
  .bodyToFlux(Employee.class);

employeeFlux.subscribe(System.out::println);
```

We also have a detailed article on setting up and working with WebClient (/spring-5-webclient).

7. Spring WebFlux Security

We can use Spring Security to secure our reactive endpoints.

Let's suppose we have a new endpoint in our *EmployeeController*. This endpoint updates *Employee* details and sends back the updated *Employee*.

Since this allows users to change existing employees, we want to restrict this endpoint to *ADMIN* role users only.

As a result, let's add a new method to our *EmployeeController*:

```
@PostMapping("/update")
public Mono<Employee> updateEmployee(@RequestBody Employee employee) {
    return employeeRepository.updateEmployee(employee);
}
```

Now, to restrict access to this method, let's create *SecurityConfig* and define some path-based rules to allow only ADMIN users:

```
@EnableWebFluxSecurity
public class EmployeeWebSecurityConfig {

    // ...

    @Bean
    public SecurityWebFilterChain springSecurityFilterChain(
      ServerHttpSecurity http) {
        http.csrf().disable()
          .authorizeExchange()
          .pathMatchers(HttpMethod.POST, "/employees/update").hasRole("ADMIN")
          .pathMatchers("/**").permitAll()
          .and()
          .httpBasic();
        return http.build();
    }
}
```

This configuration will restrict access to the endpoint */employees/update*. Therefore, only users with a role *ADMIN* will be able to access this endpoint and update an existing *Employee*.

Finally, the annotation *@EnableWebFluxSecurity* adds Spring Security WebFlux support with some default configurations.

For more information, we also have a detailed article on configuring and working with Spring WebFlux security (/spring-security-5-reactive).

8. Conclusion

In this article, we explored how to create and work with reactive web components as supported by the Spring WebFlux framework. As an example, we built a small Reactive REST application.

Then we learned how to use *RestController* and *WebClient* to publish and consume reactive streams.

We also looked into how to create a secured reactive endpoint with the help of Spring Security.

Other than Reactive *RestController* and *WebClient*, the *WebFlux* framework also supports reactive *WebSocket* and the corresponding *WebSocketClient* for socket style streaming of Reactive Streams.

For more information, we also have a detailed article focused on working with Reactive WebSocket with Spring 5 (/spring-5-reactive-websockets).

Finally, the complete source code used in this article is available over on Github (https://github.com/eugenp/tutorials/tree/master/spring-reactive-modules/spring-reactive).

Comments are closed on this article!

(/)

COURSES

ALL COURSES (/ALL-COURSES)
ALL BULK COURSES (/ALL-BULK-COURSES)
ALL BULK TEAM COURSES (/ALL-BULK-TEAM-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
EDITORS (/EDITORS)
JOBS (/TAG/ACTIVE-JOB/)
OUR PARTNERS (/PARTNERS)
PARTNER WITH BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)