Testing WebFlux Endpoints with WebTestClient

Esteban Herrera
Author | Developer | Consultant

@eh3rrera eherrera.net

Overview

Create a WebTestClient instance

Testing annotated controllers

Testing functional endpoints

WebTestClient

Creating a WebTestClient Instance

```
client = WebTestClient.bindToController(controller).build();

client = WebTestClient.bindToRouterFunction(routerFunction).build();

client = WebTestClient.bindToApplicationContext(applicationContext)
                      .build();

client = WebTestClient.bindToServer()
                      .baseUrl("http://localhost:8080").build();
```

Creating a WebTestClient Instance

```
client = WebTestClient.bindToController(controller)
                      .configureClient().baseUrl("/products").build();

client = WebTestClient.bindToRouterFunction(routerFunction)
                      .configureClient().baseUrl("/products").build();

client = WebTestClient.bindToApplicationContext(applicationContext)
                      .configureClient().baseUrl("/products").build();
```

Testing with WebTestClient

```
client
   .get()
   .uri("/products")
   .exchange()
   .expectStatus().isOk()
   .expectHeader().contentType(MediaType.APPLICATION_JSON_UTF8)
   .expectBodyList(Product.class).isEqualTo(expectedList);
   //.expectBody(Product.class);
   //.consumeWith(result -> { /* custom assertions */ });
```

Testing with WebTestClient

```
client
  .delete()
  .uri("/products")
  .exchange()
  .expectStatus().isOk()
  .expectBody(Void.class);
```

Testing with WebTestClient

```
client
  .get()
  .uri("/products")
  .exchange()
  .expectStatus().isOk()
  .expectBody().isEmpty();
```

Demo

Annotated Controllers

- Bind to controller
  - Real server
  - Mock objects
- Bind to application context
- @WebFluxTest annotation

Demo

Functional Endpoints

- Bind to router function
- Bind to server
- Auto-configure WebTestClient

Things to Remember

WebTestClient

- Uses WebClient internally
- Adds methods to verify the response

Setup

- Controllers
- Router functions
- Application context
- Running servers

Annotations

- @SpringBootTest
- @WebFluxTest
- @AutoConfigureWebTestClient

Things to Remember

Reactive programming

Project Reactor

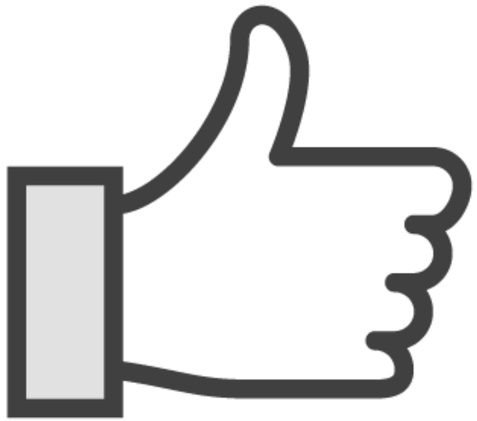
Spring WebFlux

- **Annotated controllers**
- **Functional endpoints**

WebClient

WebTestClient

Spring WebFlux Is Flexible



Server

- **Netty, Tomcat, Jetty, Undertow**

Reactive library

- **Reactor, RxJava**

Programming model

- **Controllers, functional endpoints**

Reactive Programming Model

Non-blocking

Asynchronous

Functional/Declarative

Scalability

Main Use Cases

Highly concurrent applications

- **Easy to scale**

Networks applications

- **Latency**
- **Failures**
- **Backpressure**

Server-side events

- **Real-time data**
- **Live queries**

When to Stick to Spring MVC?

Spring MVC already works for you

Reactive has a steep learning curve

Blocking libraries

Thank you