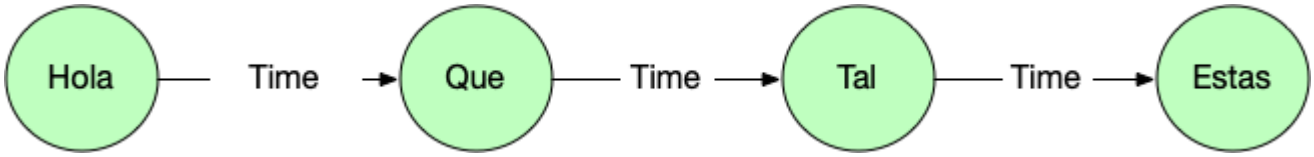
La creación de un Spring WebFlux Test es una de las cosas más habituales que tenemos que realizar cuando empezamos a construir pruebas unitarias con Spring WebFlux. La peculiaridad que estas pruebas tienen es que estamos ante programación fuertemente asíncrona de tal manera que cuando la ejecutamos existen datos que pueden llegar en un futuro cercano y no estar a nuestra disposición en el momento inicial.



Vamos a ver un par de ejemplos de estas cosas. Para ello el primer paso es construir un proyecto de Spring Boot que nos incluya las siguientes dependencias

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

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

```
                        <exclusions>
                                <exclusion>
<groupId>org.junit.vintage</groupId>
                                        <artifactId>junit-vintage-
engine</artifactId>
                                </exclusion>
                        </exclusions>
                </dependency>
                <dependency>
```

Spring WebFlux Test

Una vez tenemos las dependencias incluidas el siguiente paso es crear una clase de servicio de Spring WebFlux que devuelve Flux y Monos .

```
package com.arquitecturajava.test;

import java.time.Duration;

import org.springframework.stereotype.Service;

import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

@Service
public class ServicioSencillo {

        public Mono<String> buscarUno() {

                return Mono.just("hola");

        }
```

```
        public Flux<String> buscarTodos() {

                return Flux.just("hola","que","tal","estas");

        }
        public Flux<String> buscarTodosLento() {

                return
Flux.just("hola","que","tal","estas").delaySequence(Duration.ofSeconds
(10));

        }

}
```

Realizado este paso y sabiendo que lo que devuelven un flujo de Strings es momento de abordar la construcción de los Test.

```
package com.arquitecturajava.test;

import java.time.Duration;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;
import reactor.test.StepVerifier;

@SpringBootTest
class TestApplicationTests {
```

```
        @Autowired
        ServicioSencillo miservicio;

        @Test
        void testMono() {

                Mono<String> uno = miservicio.buscarUno();
StepVerifier.create(uno).expectNext("hola").verifyComplete();

        }

        @Test
        void testVarios() {

                Flux<String> uno = miservicio.buscarTodos();
StepVerifier.create(uno).expectNext("hola").expectNext("que").expectNe
xt("tal").expectNext("estas").verifyComplete();

        }

}
```

Estos dos primeros Test se procesan de forma bastante directa ya que usan la clase StepVerifier para verificar cada uno de los items que nos llegan . En el caso del primer método simplemente verífica un elemento ya que es una estructura Mono. Por el otro lado el otro caso es una lista de items que tendremos que verificar uno a uno hasta completar todos con verifyComplete().

Spring WebFlux y programación asíncrona

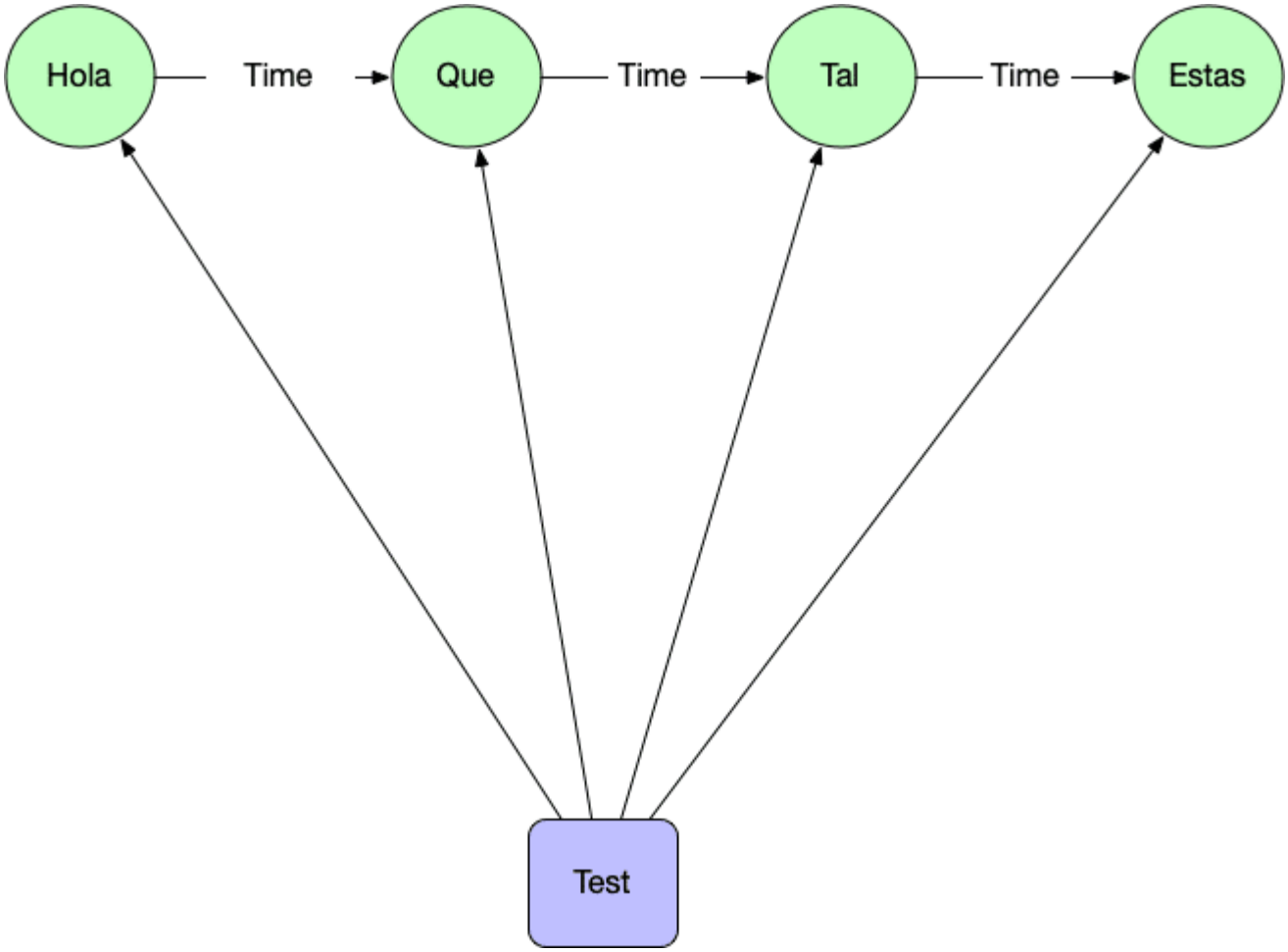
Ahora bien tenemos un problema ya que uno de los métodos devuelve una estructura Flux (BuscarTodosLento) que se ejecuta de forma claramente Asíncrona . ¿Cómo podremos

testear esto con nuestras herramientas de Testing? . Es sencillo StepVerifier nos provee de métodos como thenAwait que encajan con este tipo de programación.

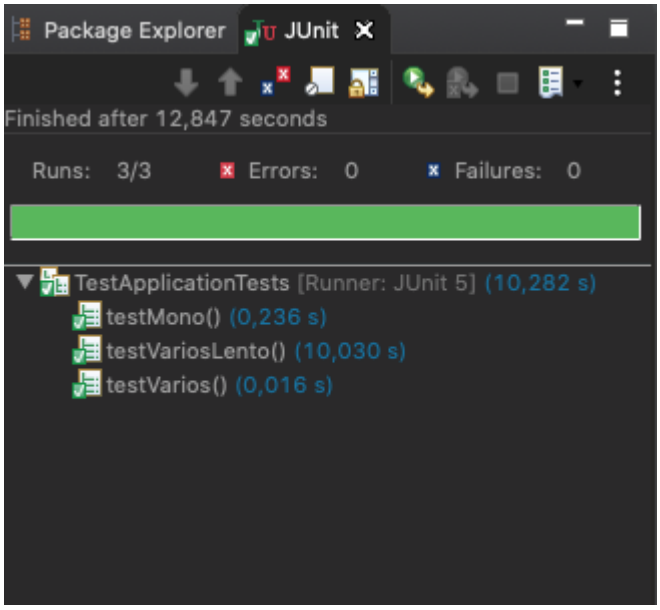
```
@Test
        void testVariosLento() {

                Flux<String> uno = miservicio.buscarTodosLento();
                StepVerifier.create(uno)
                .expectNext("hola")
                .thenAwait(Duration.ofSeconds(1))
                .expectNext("que")
                .thenAwait(Duration.ofSeconds(1))
                .expectNext("tal")
                .thenAwait(Duration.ofSeconds(1))
                .expectNext("estas")
                .thenAwait(Duration.ofSeconds(1)).verifyComplete();
        }
```

A través de este método las pruebas unitarias son capaces de esperar a que el nuevo dato nos lleve .



Según nos van llegando se verifican hasta tener una verificación total del flujo de elementos.



Conclusión

Nos damos cuenta de como el test tarda 10 segundos en ejecutarse . Hemos conseguido testear código asíncrono con Spring WebFlux Test.

Otros artículos relacionados

- Spring WebFlux
- Curso Spring WebFlux
- Spring Reactor
- Spring WebFlux