* **Spring Boot WebFlux | Write Integration Test Using Mockito & Junit**

--------------------------------------------------------------------------------------------------

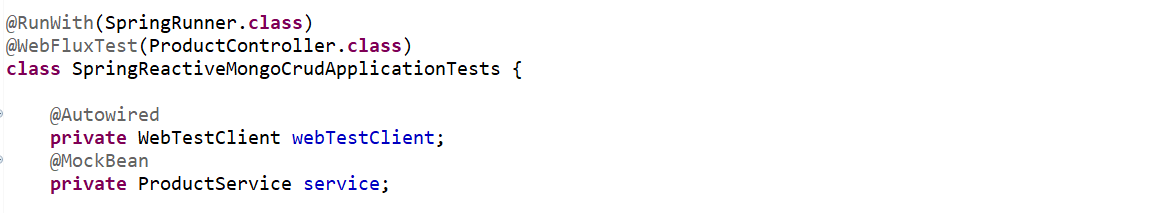
In previous tutorial, we created one crud application using Spring Data Reactive MongoDB and WebFlux. In this tutorial, we will write integration test for that crud application using JUnit and Mockito.

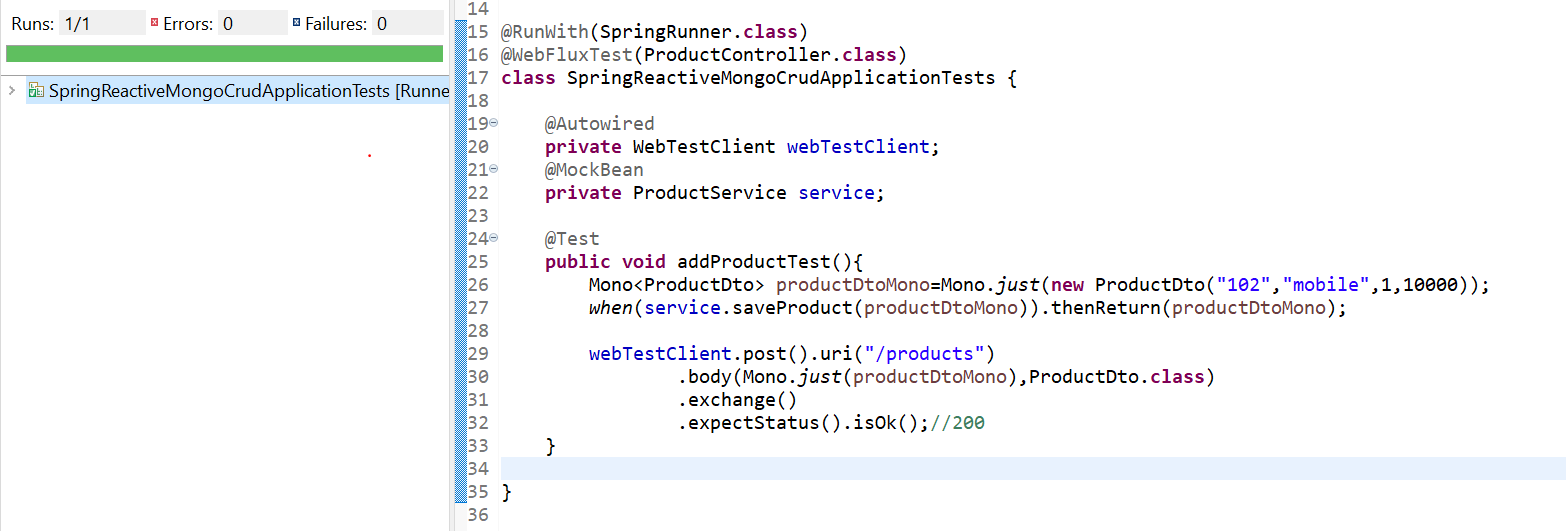
So, if you can see here, we created couple of rest endpoints in our previous tutorial. So, we are just going to write integration test for all the endpoints, almost 5 to 6 endpoints.

Now go to the test, go to the java, go to the class. So here I just need to annotate, run with, then I just need to pass spring runner.class. So, we just need to add JUnit in our class path, then just input it. Then next just annotate, add @**WebFluxTest** and just give the controller class name.

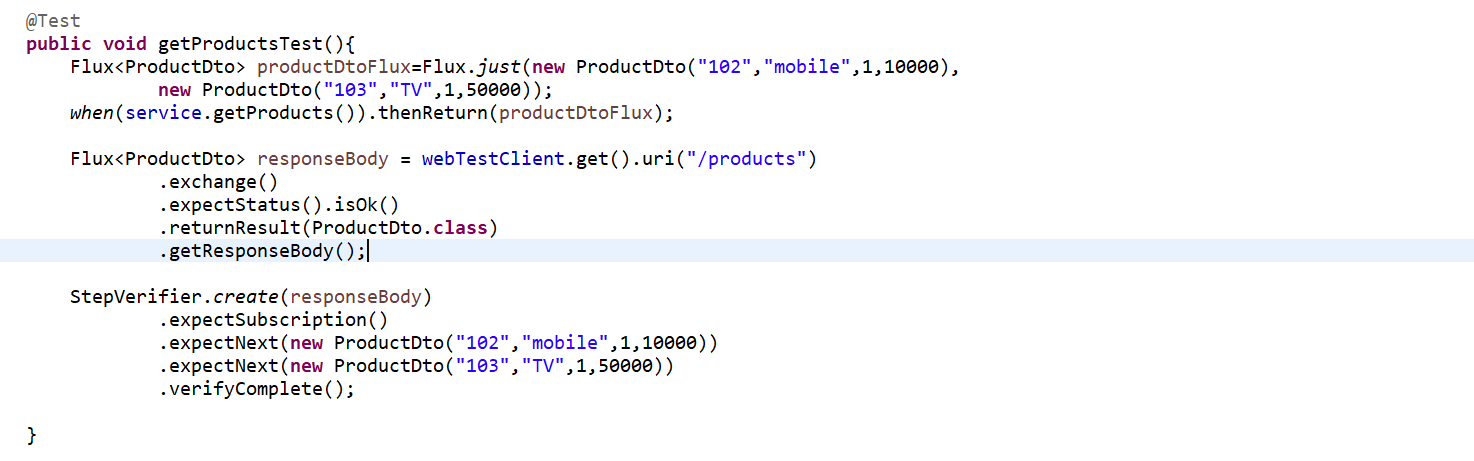
Now here, we just need to inject our web test client to call the **HTTP** method and also, we need to inject our service class to mock that layer. Because we are not going to hit our actual database, so that's the reason I just want to mock my service layer. So just inject the web test client. Inject using auto add, then we just need to add private product service, service, just annotate here, add the rate, mockbin. So that it will not create the actual object, it will give us the stop.

**App Name- spring-reactive-mongo-crud**

------------------------------------------------------------------------------------------------------------------------------------------

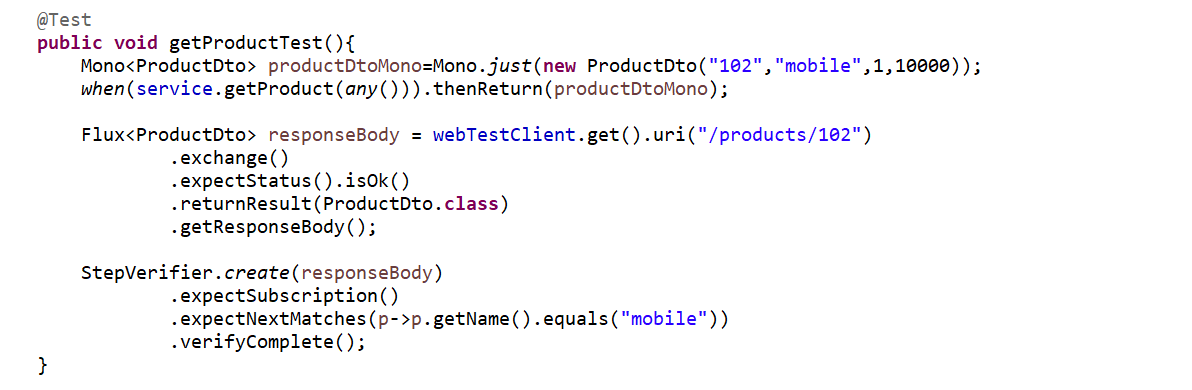


------------------------------------------------------------------------------------------------------------------------------------------

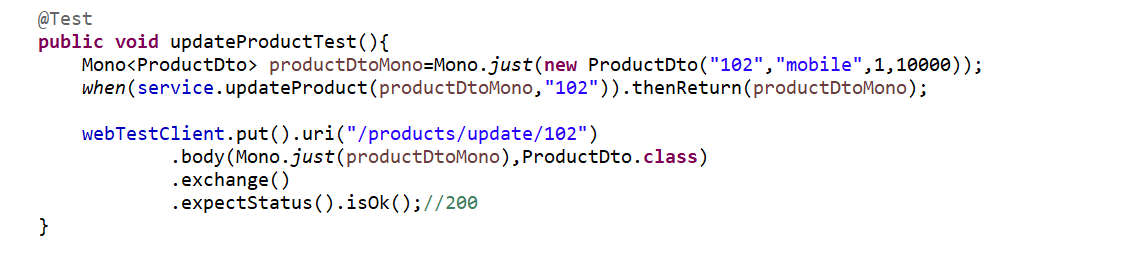


<https://www.baeldung.com/reactive-streams-step-verifier-test-publisher>

------------------------------------------------------------------------------------------------------------------------------------------

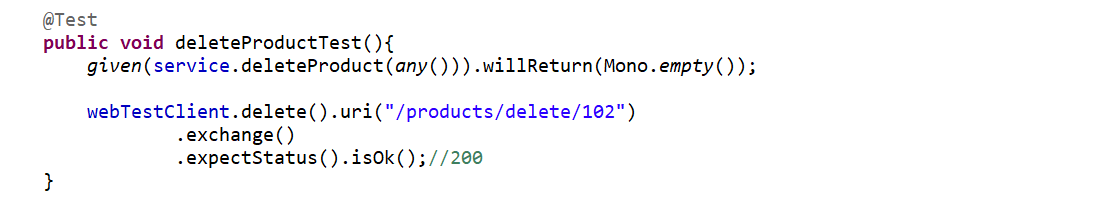


------------------------------------------------------------------------------------------------------------------------------------------



------------------------------------------------------------------------------------------------------------------------------------------

Now let's write for delete. So, if you'll go to the product controller, there is a delete mapping, and the return type is Mono void. This is what one of the interesting thing in Mockito. You need to write the test case for void return type, which is Mono void. I can directly bypass the API call. I can write, we'll return Mono dot empty.



we are just mocking the service layer. Even we are not mocking the repo. We are directly mocking the service layer.

**@RunWith(SpringRunner.class)**

**@WebFluxTest(ProductController.class)**

**class SpringReactiveMongoCrudApplicationTests {**

**@Autowired**

**private WebTestClient webTestClient;**

**@MockBean**

**private ProductService service;**

**@Test**

**public void addProductTest(){**

**Mono<ProductDto> productDtoMono=Mono.*just*(new ProductDto("102","mobile",1,10000));**

***when*(service.saveProduct(productDtoMono)).thenReturn(productDtoMono);**

**webTestClient.post().uri("/products")**

**.body(Mono.*just*(productDtoMono),ProductDto.class)**

**.exchange()**

**.expectStatus().isOk();//200**

**}**

**@Test**

**public void getProductsTest(){**

**Flux<ProductDto> productDtoFlux=Flux.*just*(new ProductDto("102","mobile",1,10000),**

**new ProductDto("103","TV",1,50000));**

***when*(service.getProducts()).thenReturn(productDtoFlux);**

**Flux<ProductDto> responseBody = webTestClient.get().uri("/products")**

**.exchange()**

**.expectStatus().isOk()**

**.returnResult(ProductDto.class)**

**.getResponseBody();**

**StepVerifier.*create*(responseBody)**

**.expectSubscription()**

**.expectNextMatches(p->p.getName().equals("mobile"))**

**.expectNext(new ProductDto("102","mobile",1,10000))**

**.expectNext(new ProductDto("103","TV",1,50000))**

**.verifyComplete();**

**}**

**@Test**

**public void getProductTest(){**

**Mono<ProductDto> productDtoMono=Mono.*just*(new ProductDto("102","mobile",1,10000));**

***when*(service.getProduct(*any*())).thenReturn(productDtoMono);**

**Flux<ProductDto> responseBody = webTestClient.get().uri("/products/102")**

**.exchange()**

**.expectStatus().isOk()**

**.returnResult(ProductDto.class)**

**.getResponseBody();**

**StepVerifier.*create*(responseBody)**

**.expectSubscription()**

**.expectNextMatches(p->p.getName().equals("mobile"))**

**.verifyComplete();**

**}**

**@Test**

**public void updateProductTest(){**

**Mono<ProductDto> productDtoMono=Mono.*just*(new ProductDto("102","mobile",1,10000));**

***when*(service.updateProduct(productDtoMono,"102")).thenReturn(productDtoMono);**

**webTestClient.put().uri("/products/update/102")**

**.body(Mono.*just*(productDtoMono),ProductDto.class)**

**.exchange()**

**.expectStatus().isOk();//200**

**}**

**@Test**

**public void deleteProductTest(){**

***given*(service.deleteProduct(*any*())).willReturn(Mono.*empty*());**

**webTestClient.delete().uri("/products/delete/102")**

**.exchange()**

**.expectStatus().isOk();//200**

**}**

**}**