

This document contains Flux endpoints with get,post,put and delete and their unit testing

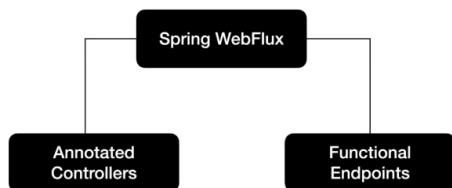
Web on Reactive stack :

<https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html>

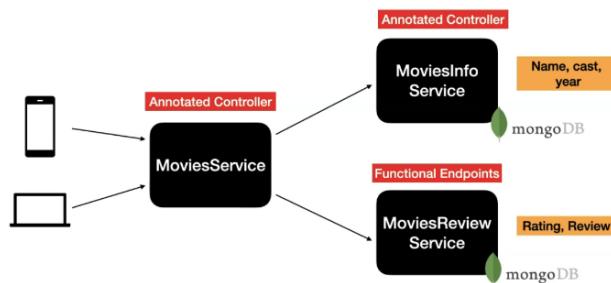
<https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html#webflux-controller>

<https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html#webflux-fn>

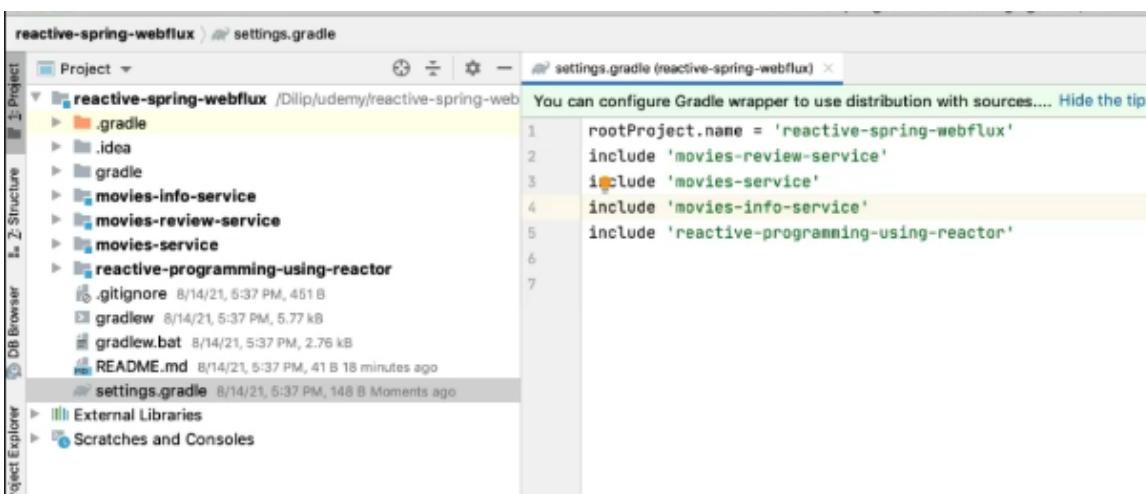
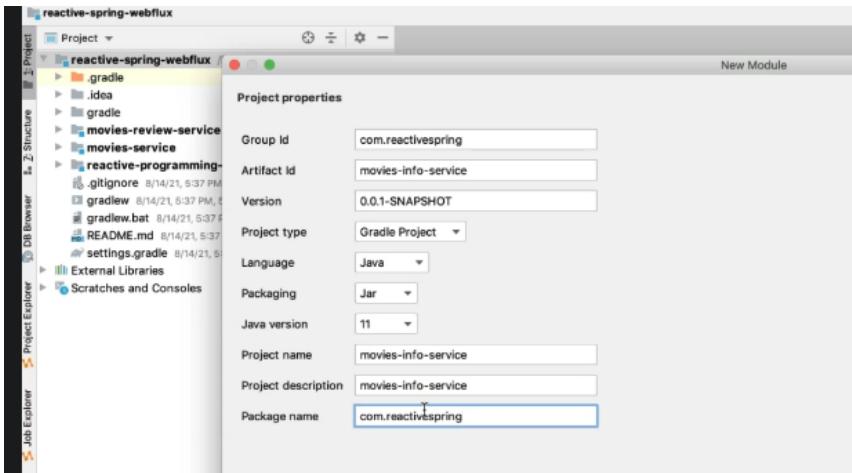
Spring WebFlux



Movies Application using MicroServices Pattern



Create new module



```

19 }
20
21 > dependencies {
22
23     implementation 'org.springframework.boot:spring-boot-starter-data-mongodb-reactive'
24     implementation 'org.springframework.boot:spring-boot-starter-validation'
25     implementation 'org.springframework.boot:spring-boot-starter-webflux'
26
27     compileOnly 'org.projectlombok:lombok'
28     annotationProcessor 'org.projectlombok:lombok'
29
30     testImplementation 'org.springframework.boot:spring-boot-starter-test'
31     testImplementation 'de.flapdoodle.embed:de.flapdoodle.embed.mongo'
32     testImplementation 'io.projectreactor:reactor-test'
33 }

```

<https://github.com/dilipsundarraj1/reactive-spring-webflux> - readme file contain how to install mongo db.

And start the app it works

```

import org.springframework.web.bind.annotation.RestController;
import reactor.core.publisher.Flux;

@RestController
public class FluxAndMonoController {

    @GetMapping("/flux")
    public Flux<Integer> flux(){
        return Flux.just(1,2,3);
    }
}

```

Create a controller that returns flux.

Normally we return a response entity but we are returning flux. What is the difference non-blocking /asynchronous

```

return Flux.just(1,2,3)
.log();

```

```

2021-08-15 05:20:40.293 INFO 18837 --- [main] c.r.MoviesInfoServiceApplication : Starting MoviesInfoServiceApplication
2021-08-15 05:20:40.296 INFO 18837 --- [main] c.r.MoviesInfoServiceApplication : No active profile set, falling back to default profiles: default
2021-08-15 05:20:40.864 INFO 18837 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data R
2021-08-15 05:20:40.873 INFO 18837 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data reposi
2021-08-15 05:20:41.728 INFO 18837 --- [main] org.mongodb.driver.cluster : Cluster created with settin
2021-08-15 05:20:41.910 INFO 18837 --- [localhost:27017] org.mongodb.driver.connection : Opened connection [connecti
2021-08-15 05:20:41.910 INFO 18837 --- [localhost:27017] org.mongodb.driver.connection : Opened connection [connecti
2021-08-15 05:20:41.912 INFO 18837 --- [localhost:27017] org.mongodb.driver.cluster : Monitor thread successfully
2021-08-15 05:20:42.042 INFO 18837 --- [main] o.s.b.web.embedded.netty.NettyWebServer : Netty started on port 8080
2021-08-15 05:20:42.054 INFO 18837 --- [main] c.r.MoviesInfoServiceApplication : Started MoviesInfoServiceAp
2021-08-15 05:20:47.556 INFO 18837 --- [ctor-http-nio-2] reactor.Flux.Array.1 : | onSubscribe{[Synchronous
2021-08-15 05:20:47.558 INFO 18837 --- [ctor-http-nio-2] reactor.Flux.Array.1 : | request(unbounded)
2021-08-15 05:20:47.559 INFO 18837 --- [ctor-http-nio-2] reactor.Flux.Array.1 : | onNext(1)
2021-08-15 05:20:47.559 INFO 18837 --- [ctor-http-nio-2] reactor.Flux.Array.1 : | onNext(2)
2021-08-15 05:20:47.559 INFO 18837 --- [ctor-http-nio-2] reactor.Flux.Array.1 : | onNext(3)
2021-08-15 05:20:47.559 INFO 18837 --- [ctor-http-nio-2] reactor.Flux.Array.1 : | onComplete()

```

Same server thread being used. Ctor-http-nio-2 as there is not much traffic.

The screenshot shows a Java IDE interface with a code editor and a terminal window.

Code Editor:

```
19     @GetMapping("/mono")
20     public Mono<String> helloWorldMono(){
21
22         return Mono.just("hello-world")
23             .log();
24 }
```

Terminal:

```
[1,2,3]%
/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl http://localhost:8080/flux
[1,2,3]%
/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl http://localhost:8080/mono
hello-world%
```

Logs:

```
2021-08-15 05:26:18.644  INFO 19289 --- [           main] c.r.moviesintoserviceapplication      : STARTED MOVIESINTOSERVICEAPP!
2021-08-15 05:26:23.607  INFO 19289 --- [ctor-http-nio-2] reactor.Mono.Just.1          : | onSubscribe([Synchronous Fi
2021-08-15 05:26:23.609  INFO 19289 --- [ctor-http-nio-2] reactor.Mono.Just.1          : | request(unbounded)
2021-08-15 05:26:23.610  INFO 19289 --- [ctor-http-nio-2] reactor.Mono.Just.1          : | onNext(hello-world)
2021-08-15 05:26:23.612  INFO 19289 --- [ctor-http-nio-2] reactor.Mono.Just.1          : | onComplete()
```

Streaming Endpoint Using Spring Webflux

Streaming Endpoint

- Streaming Endpoint is a kind of Endpoint which continuously sends updates to the clients as the new data arrives
- This concept is similar to Server Sent Events(SSE)
- Easy to implement in Spring WebFlux
- **Examples :** Stock Tickers, Realtime updates of Sports Events

```

    @GetMapping(value = "/stream", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
    public Flux<Long> stream(){

        return Flux.interval(Duration.ofSeconds(1))
            .log();
    }

```

```

2021-08-16 04:59:57.230 INFO 40888 --- [           main] c.r.MoviesInfoServiceApplication      : Started MoviesInfoServiceApplication in
2021-08-16 05:00:12.803 INFO 40888 --- [ctor-http-nio-2] reactor.Flux.Interval.1          : onSubscribe(FluxInterval.IntervalRunnab
2021-08-16 05:00:12.805 INFO 40888 --- [ctor-http-nio-2] reactor.Flux.Interval.1          : request(1)
2021-08-16 05:00:13.808 INFO 40888 --- [parallel-1] reactor.Flux.Interval.1          : onNext(0)
2021-08-16 05:00:13.839 INFO 40888 --- [ctor-http-nio-2] reactor.Flux.Interval.1          : request(31)
2021-08-16 05:00:14.809 INFO 40888 --- [parallel-1] reactor.Flux.Interval.1          : onNext(1)
2021-08-16 05:00:15.811 INFO 40888 --- [parallel-1] reactor.Flux.Interval.1          : onNext(2)
2021-08-16 05:00:16.809 INFO 40888 --- [parallel-1] reactor.Flux.Interval.1          : onNext(3)
2021-08-16 05:00:17.810 INFO 40888 --- [parallel-1] reactor.Flux.Interval.1          : onNext(4)
2021-08-16 05:00:18.809 INFO 40888 --- [parallel-1] reactor.Flux.Interval.1          : onNext(5)
2021-08-16 05:00:19.811 INFO 40888 --- [parallel-1] reactor.Flux.Interval.1          : onNext(6)

```

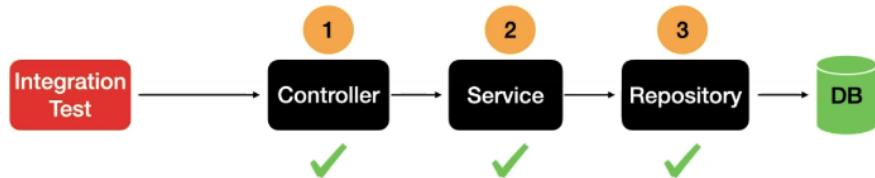
Notice the parallel thread started.

Automated Tests

- Automated Tests plays a vital role in delivering quality Software
- Two types of Automated Tests:
 - Integration Tests
 - Unit Tests

Integration Tests

- Integration test is a kind of test which actually test the application end to end



Unit Tests

- Unit test is a kind of test which tests only the class and method of interest and mocks the next layer of the code



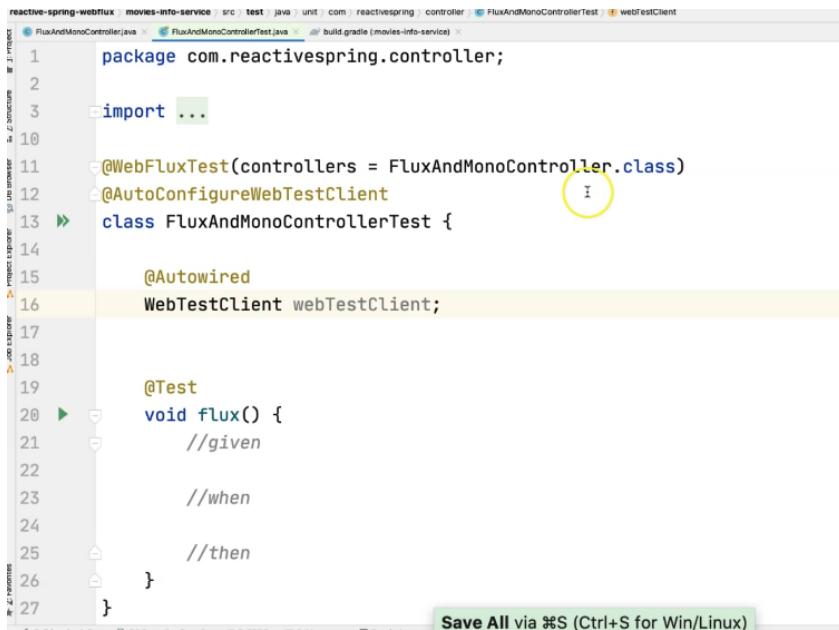
```
35 >  test {
36     useJUnitPlatform()
37 }
38
```

User Junit for testing. Spring boot uses recent versions of Junit. So it uses Junit5.

The screenshot shows a Java file structure in an IDE. The file is located in the "src/test/java/intg/unit" directory. The code contains the following JUnit5 annotations:

```
37 }
38
39 sourceSets {
40     test {
41         java.srcDirs = ['src/test/java/unit', 'src/test/java/intg']
42     }
43 }
44
```

The line "useJUnitPlatform()" is highlighted with a yellow circle.



```
reactive-spring-webflux movies-info-service src test java unit com.reactivespring.controller FluxAndMonoControllerTest webTestClient
1 package com.reactivespring.controller;
2
3 import ...
4
5 @WebFluxTest/controllers = FluxAndMonoController.class
6 @AutoConfigureWebTestClient
7 > class FluxAndMonoControllerTest {
8
9     @Autowired
10    WebTestClient webTestClient;
11
12
13     @Test
14     void flux() {
15         //given
16
17         //when
18
19         //then
20     }
21
22 }
```

Save All via %S (Ctrl+S for Win/Linux)

Webflux allows us to access all the endpoints available in that controller

Like mvc TestRestTemplate we have webTestClient automatically injected to this class.



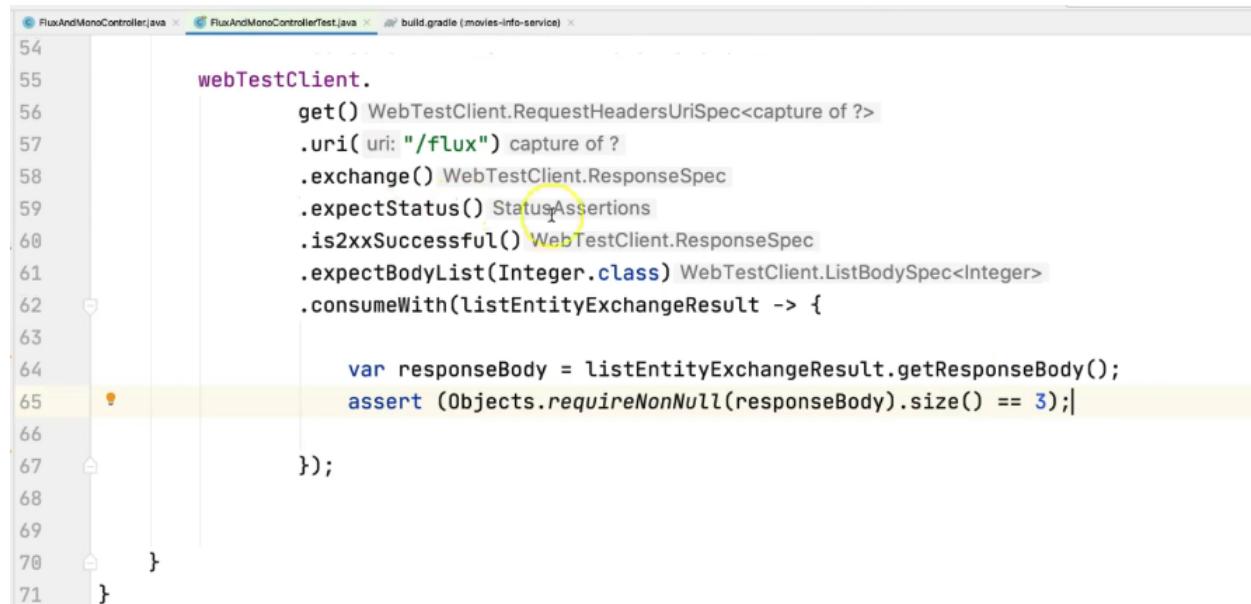
```
3
4     @Test
5     void flux() {
6
7         webTestClient.
8             get() WebTestClient.RequestHeadersUriSpec<?
9                 .uri(uri: "/flux") capture of ?
10                .exchange() WebTestClient.ResponseSpec
11                .expectStatus() StatusAssertions
12                .is2xxSuccessful() WebTestClient.ResponseSpec
13                .expectBodyList(Integer.class) WebTestClient.ResponseSpec
14                .hasSize(3);
15
16     }
17
18
19
20
21
22
23
8 import java.time.Duration;
9
10 @RestController
11 public class FluxAndMonoController {
12
13     @GetMapping("/flux")
14     public Flux<Integer> flux(){
15
16         return Flux.just(1,2,3)
17             .log();
18     }
19
20     @GetMapping("/mono")
21     public Mono<String> helloWorldMono(){
22
23 }
```

Exchange method invokes the endpoint. Testing the size of response

```

3     @Test
4     void flux_approach2() {
5
6         var flux = webTestClient.
7             get() WebTestClient.RequestHeadersUriSpec<capture of ?>
8                 .uri( uri: "/flux") capture of ?
9                 .exchange() WebTestClient.ResponseSpec
10                .expectStatus() StatusAssertions
11                .is2xxSuccessful() WebTestClient.ResponseSpec
12                .returnResult(Integer.class) FluxExchangeResult<Integer>
13                .getResponseBody();
14
15        StepVerifier.create(flux)
16            .expectNext(1,2,3)
17            .verifyComplete();
18    }
19 }
```

Testing the response body



```

54
55     webTestClient.
56         get() WebTestClient.RequestHeadersUriSpec<capture of ?>
57             .uri( uri: "/flux") capture of ?
58             .exchange() WebTestClient.ResponseSpec
59             .expectStatus() StatusAssertions
60             .is2xxSuccessful() WebTestClient.ResponseSpec
61             .expectBodyList(Integer.class) WebTestClient.ListBodySpec<Integer>
62             .consumeWith(listEntityExchangeResult -> {
63
64                 var responseBody = listEntityExchangeResult.getResponseBody();
65                 assert (Objects.requireNonNull(responseBody).size() == 3);|
66
67             });
68
69
70     }
71 }
```

Approach 3

```

51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69

    @Test
    void stream() {
        I
        var flux = webTestClient.
            get()
            .uri( "/stream" )
            .exchange()
            .expectStatus()
            .is2xxSuccessful()
            .returnResult( Long.class )
            .getResponseBody();
        StepVerifier.create( flux )
            .expectNext( 0L, 1L, 2L, 3L )
            .thenCancel()
            .verify();
    }
}

```

```

13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

    @GetMapping( "/flux" )
    public Flux< Integer > flux(){
        return Flux.just( 1, 2, 3 )
            .log();
    }

    @GetMapping( "/mono" )
    public Mono< String > helloWorldMono(){
        return Mono.just( "hello-world" )
            .log();
    }

    @GetMapping( value = "/stream", produces = MediaType.TEXT_EVENT_STREAM_TYPE )
    public Flux< Long > stream(){
        return Flux.interval( Duration.ofSeconds( 1 ) )
            .log();
    }
}

```

Testing mono endpoint

```

68
69
70

    @Test
    void stream() {
        I
        var flux = webTestClient.
            get()
            .uri( "/stream" )
            .exchange()
            .expectStatus()
            .is2xxSuccessful()
            .returnResult( Long.class )
            .getResponseBody();
        StepVerifier.create( flux )
            .expectNext( 0L, 1L, 2L, 3L )
            .thenCancel()
            .verify();
    }
}

Run: FluxAndMonoControllerTest.stream
Tests passed: 1 of 1 test - 4 s 367 ms

```

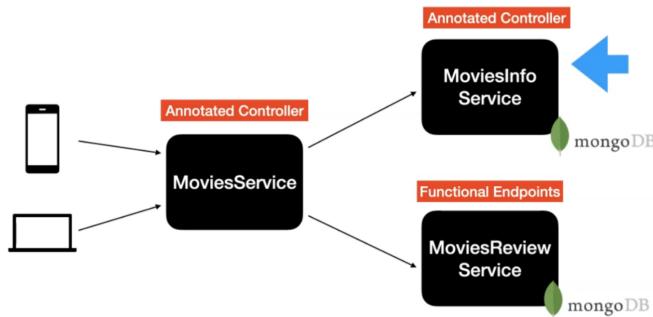
```

34
35
36
37

    0 60887 --- [ parallel-1] reactor.Flux.Interval.1 : onSubscribe(FluxInterval.IntervalRunnable)
    0 60887 --- [ parallel-1] reactor.Flux.Interval.1 : request(1)
    0 60887 --- [ parallel-1] reactor.Flux.Interval.1 : onNext(0)
    0 60887 --- [ main] reactor.Flux.Interval.1 : request(31)
    0 60887 --- [ parallel-2] reactor.Flux.Interval.1 : onNext(1)
    0 60887 --- [ parallel-2] reactor.Flux.Interval.1 : onNext(2)
    0 60887 --- [ parallel-2] reactor.Flux.Interval.1 : onNext(3)
    0 60887 --- [ parallel-2] reactor.Flux.Interval.1 : cancel()

```

Movies Application using MicroServices Pattern



```
11
12 @Data
13 @NoArgsConstructor
14 @AllArgsConstructor
15 @Document
16 public class MovieInfo {
17
18     @Id
19     private String movieInfoId;
20     private String name;
21     private Integer year;
22     private List<String> cast;
23     private LocalDate release_date;
24 }
25
```

```
MovieInfo.java x MovieInfoRepository.java x
1 package com.reactivespring.repository;
2
3 import com.reactivespring.domain.MovieInfo;
4 import org.springframework.data.mongodb.repository.ReactiveMongoRepository;
5
6 public interface MovieInfoRepository extends ReactiveMongoRepository<MovieInfo, String> {
7 }
```

A screenshot of a code editor showing the `application.yml` configuration file. The file contains YAML code defining Spring profiles and MongoDB settings. The code is as follows:

```
1 spring:
2   profiles:
3     active: local
4   ---
5 spring:
6   config:
7     activate:
8       on-profile:
9         - local
10  data:
11    mongodb:
12      host: localhost
13      port: 27017
14      database: local
```

A screenshot of a code editor showing the `application.yml` configuration file. The file contains YAML code defining Spring profiles and MongoDB settings. The code is as follows:

```
15  -----
16 spring:
17   config:
18     activate:
19       on-profile:
20         - non-prod
21   data:
22     mongodb:
23       host: localhost
24       port: 27017
25       database: local
26  -----
27 spring:
28   config:
29     activate:
30       on-profile: prod
```

```

26   ---  

27   spring:  

28     config:  

29       activate:  

30         on-profile: prod  

31     data:  

32       mongodb:  

33         host: localhost  

34         port: 27017  

35       database: local

```

47. Setup the Integration Test using `@DataMongoTest`

```

    testImplementation 'org.springframework.boot:spring-boot-starter-test'  

    testImplementation 'de.flapdoodle.embed:de.flapdoodle.embed.mongo'  

    testImplementation 'io.projectreactor:reactor-test'

```

This dependency gives embed mongo just like h2db.



```

reactive-spring-webflux movies-info-service src test java com.reactivespring.repository MovieInfoRepositoryTest movieInfoRepository  

1 package com.reactivespring.repository;  

2  

3 import org.springframework.beans.factory.annotation.Autowired;  

4 import org.springframework.boot.test.autoconfigure.data.mongo.DataMongoTest;  

5 import org.springframework.test.context.ActiveProfiles;  

6  

7 import static org.junit.jupiter.api.Assertions.*;  

8  

9 @DataMongoTest  

10 @ActiveProfiles("test")  

11 class MovieInfoRepositoryIntgTest {  

12  

13     @Autowired  

14     MovieInfoRepository movieInfoRepository;  

15  

16 }

```

`@DataMongoTest` scan the application and look for repository classes and make that class available in your test case. We do not have to instantiate the whole spring application context in order to write an integration test for the database layer. We can test the application faster than starting from scratch.

`@ActiveProfiles("test")` will give connection to embedded mongo db. If we give `@ActiveProfiles("local")` will take the values from the application.yml file.

```

    @BeforeEach
    void setUp() {
        var movieinfos = List.of(new MovieInfo(movieInfoId: null, name: "Batman Begins",
            year: 2005, List.of("Christian Bale", "Michael Caine"), LocalDate.parse("2005-06-15")),
            new MovieInfo(movieInfoId: null, name: "The Dark Knight",
            year: 2008, List.of("Christian Bale", "Heath Ledger"), LocalDate.parse("2008-07-18")),
            new MovieInfo(movieInfoId: "abc", name: "Dark Knight Rises",
            year: 2012, List.of("Christian Bale", "Tom Hardy"), LocalDate.parse("2012-07-20")));

        movieInfoRepository.saveAll(movieinfos)
            .blockLast();
    }

    @AfterEach
    void tearDown() {
        movieInfoRepository.deleteAll().block();
    }
}

```

Line 32 blockLast() (block the thread) will make the previous one complete (synchronous) otherwise our test will fail because of asynchronous nature.

1. FindAll method

```

41
42
43
44
45
46
47
48
49
50
51
52

```

```

    @Test
    void findAll() {
        //given
        //when
        var moviesInfoFlux = movieInfoRepository.findAll().log();
        //then
        StepVerifier.create(moviesInfoFlux)
            .expectNextCount(3)
            .verifyComplete();
    }

```

Run > MovieInfoRepositoryIntgTest.findAll > Test Rx 683ms > ✓ Test Rx 683ms > ✓ Mon 683ms > ✓ 683ms

```

embedded.EmbeddedMongo : 2021-08-18T05:19:25.262-0500 I NETWORK [thread1] connection accepted from 127.0.0.1:55502 #5 (5 connect
embedded.EmbeddedMongo : 2021-08-18T05:19:25.273-0500 I NETWORK [conn5] received client metadata from 127.0.0.1:55502 conn5: { c
!r.connection : Opened connection [connectionId{localValue:5, serverValue:5}] to localhost:55491
igWhen.1 : onSubscribe(FluxUsingWhen.UsingWhenSubscriber)
igWhen.1 : request(unbounded)
igWhen.1 : onNext(MovieInfo(movieInfoId=611cdead9f2149125440a784, name=Batman Begins, year=2005, cast=[Christian Ba
igWhen.1 : onNext(MovieInfo(movieInfoId=611cded9f2149125440a785, name=The Dark Knight, year=2008, cast=[Christian
igWhen.1 : onNext(MovieInfo(movieInfoId=abc, name=Dark Knight Rises, year=2012, cast=[Christian Bale, Tom Hardy], r
igWhen.1 : onComplete()

```

```

.f.embed.mongo.MongodExecutable      : start MongodConfig{version=Version{3.5.5}, timeout=de.flapdoodle.embed.mongo.config.Timeout
.g.mongodb.driver.cluster          : Cluster created with settings {hosts=[localhost:55491], mode=SINGLE, requiredClusterType=
.s.b.a.mongo.embedded.EmbeddedMongo : 2021-08-18T05:19:24.507-0500 I NETWORK [thread1] connection accepted from 127.0.0.1:55491

```

Embedded mongo db startup in the console.

2. findById Method

```
56     void findById() {
57         //given
58
59         //when
60         var moviesInfoMono = movieInfoRepository.findById("abc").log();
61
62         //then
63         StepVerifier.create(moviesInfoMono)
64             .expectNextCount(1)
65             .assertNext(movieInfo -> {
66                 assertEquals(expected: "Dark Knight Rises", movieInfo.getName());
67             })
68
69     }
70 }
```

Run: MovieInfoRepositoryIntgTest.findById
Tests passed: 1 of 1 test – 739 ms

Test Results
MovieInfoRepository.findById()

Logs (739 ms)
2021-08-18T05:26:51.977-0500 I NETWORK [conn4] received client metadata from 127.0.0.1:56429
: Opened connection [connectionId{localValue:3, serverValue:3}] to localhost:56429
: Opened connection [connectionId{localValue:4, serverValue:4}] to localhost:56429
: 2021-08-18T05:26:51.998-0500 I NETWORK [conn5] received client metadata from 127.0.0.1:56429
: Opened connection [connectionId{localValue:5, serverValue:5}] to localhost:56429
: onSubscribe(MonoUsingWhen.MonoUsingWhenSubscriber)
: request(unbounded)
: onNext(MovieInfo(movieInfoId=abc, name=Dark Knight Rises, year=2012, cast=[Christian Bale, Michael Caine]))
: onComplete()

Whenever we are interact with reactive repository class it will return always Flux/Mono

3. Save Method

```
1 @Test
2 void saveMovieinfo() {
3     //given
4     var movieInfo = new MovieInfo(movieInfoId: null, name: "Batman Begins1",
5         year: 2005, List.of("Christian Bale", "Michael Cane"), LocalDate.parse("2005-06-15"));
6
7     //when
8     var moviesInfoMono = movieInfoRepository.save(movieInfo).log();
9
10    //then
11    StepVerifier.create(moviesInfoMono)
12        .expectNextCount(1)
13        .assertNext(movieInfo1 -> {
14            assertNotNull(movieInfo1.getMovieInfoId());
15            assertEquals(expected: "Batman Begins1", movieInfo1.getName());
16        })
17        .verifyComplete();
18
19 }
```

Run: MovieInfoRepositoryIntgTest.saveMovieinfo
Tests passed: 1 of 1 test – 739 ms

Test Results
MovieInfoRepository.saveMovieinfo()

Logs (739 ms)
2021-08-18T05:26:51.977-0500 I NETWORK [conn4] received client metadata from 127.0.0.1:56429
: Opened connection [connectionId{localValue:3, serverValue:3}] to localhost:56429
: 2021-08-18T05:26:51.998-0500 I NETWORK [conn5] received client metadata from 127.0.0.1:56429
: Opened connection [connectionId{localValue:4, serverValue:4}] to localhost:56429
: 2021-08-18T05:26:52.002-0500 I NETWORK [conn5] received client metadata from 127.0.0.1:56429
: Opened connection [connectionId{localValue:5, serverValue:5}] to localhost:56429
: onSubscribe(MonoUsingWhen.MonoUsingWhenSubscriber)
: request(unbounded)
: onNext(MovieInfo(movieInfoId=1, name=Batman Begins1, year=2005, cast=[Christian Bale, Michael Cane]))
: onComplete()

```

    n.1      : onSubscribe(AnonymousObserver)
    n.1      : request(unbounded)
    n.1      : onNext(MovieInfo(movieInfoId=611ce17a0ec4f42acf8697a5, name=Batman Begins1, year=2005, cast=[Christian Bale, Mich
    n.1      : onComplete()

```

Update method : there is no specific function for update we use save method after data modification.

```

90      @Test
91      void updateMovieInfo() {
92          //given
93          var movieInfo = movieInfoRepository.findById("abc").block();
94          movieInfo.setYear(2021);
95
96          //when
97          var moviesInfoMono = movieInfoRepository.save(movieInfo).log();
98
99          //then
100         StepVerifier.create(moviesInfoMono)
101             //.expectNextCount(1)
102             .assertNext(movieInfo1 -> {
103                 assertNotNull(movieInfo1.getMovieInfoId());
104                 assertEquals(expected: "Batman Begins1", movieInfo1.getName());
105             })
106             .verifyComplete();
107     }
108 }
```

```

    n.1      : onSubscribe(AnonymousObserver)
    n.1      : request(unbounded)
    n.1      : onNext(MovieInfo(movieInfoId=abc, name=Dark Knight Rises, year=2021, cast=[Christian Bale, Tom Hardy], releas
    n.1      : onComplete()

```

Delete method.

```

108
109      @Test
110      void deleteMovieInfo() {
111          //given
112
113          //when
114          movieInfoRepository.deleteById("abc").log();
115          var moviesInfoFlux = movieInfoRepository.findAll().log();
116
117          //then
118          StepVerifier.create(moviesInfoFlux)
119              .expectNextCount(2)
120              .verifyComplete();
121
122      }

```

```

113     //when
114     movieInfoRepository.deleteById("abc").block();
115     var moviesInfoFlux = movieInfoRepository.findAll().log();
116
117     //then
118     StepVerifier.create(moviesInfoFlux)
119         .expectNextCount(2)
120         .verifyComplete();
121     }
122 }

```

Run: MovieInfoRepositoryIntTest.deleteMovieInfo

Test Results: Tests passed: 1 of 1 test - 598 ms

- Test Results: 598 ms
 - MovieInfoReposi: 598 ms
 - deleteMoviein: 598 ms

```

er.connection : Opened connection [connectionId{localValue:4, serverValue:4}] to localhost:54166
bedded.EmbeddedMongo : 2021-08-19T04:36:40.013-0500 I NETWORK [thread1] connection accepted from 127.0.0.1:54182 #5 (5
bedded.EmbeddedMongo : 2021-08-19T04:36:40.024-0500 I NETWORK [conn5] received client metadata from 127.0.0.1:54182 co
er.connection : Opened connection [connectionId{localValue:5, serverValue:5}] to localhost:54166
ngWhen.1 : onSubscribe(FluxUsingWhen.UsingWhenSubscriber)
ngWhen.1 : request(unbounded)
ngWhen.1 : onNext(MovieInfo(movieInfoId=611e2627eb48bc01a6ad0d1c, name=Batman Begins, year=2005, cast=[Chri
ngWhen.1 : onNext(MovieInfo(movieInfoId=611e2627eb48bc01a6ad0d1d, name=The Dark Knight, year=2008, cast=[Ch
ngWhen.1 : onComplete()

```

Abc got deleted. We have to replace log() with block() to block asynchronous calls during unit testing..

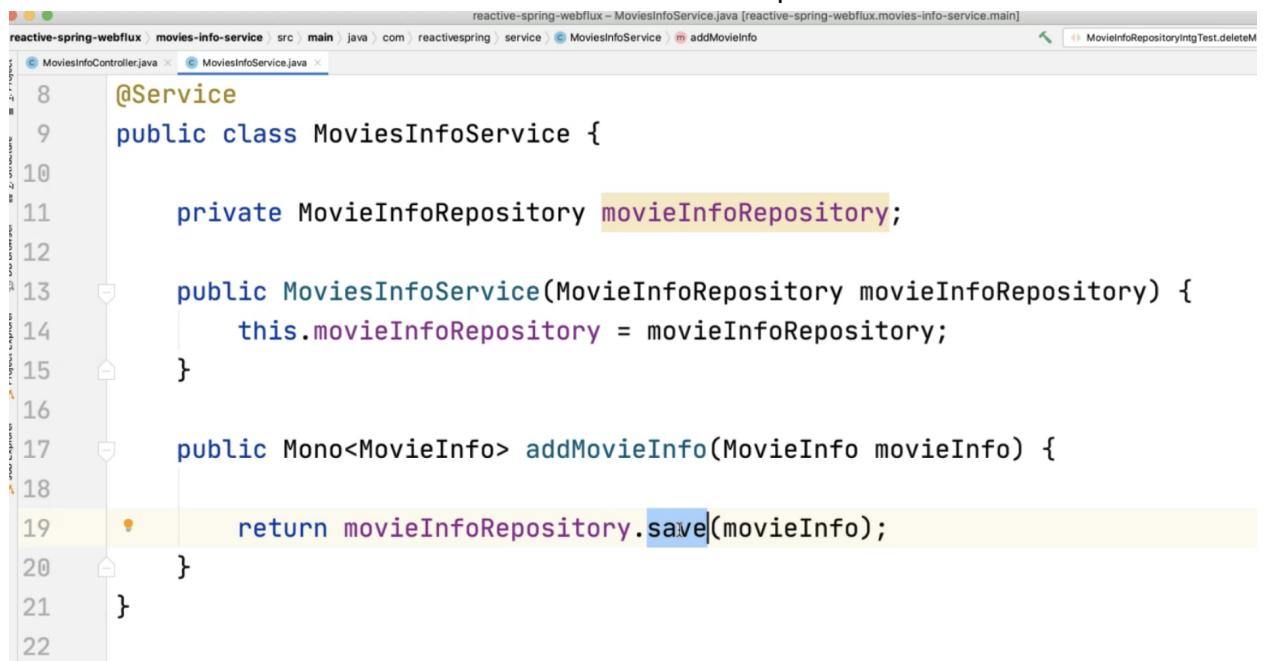
Controller Add movie. Notice service is used in constructor instead of autowire. Best practice.

```

9   @RestController
10  @RequestMapping("/v1")
11  public class MoviesInfoController {
12
13      private MoviesInfoService moviesInfoService;
14
15      public MoviesInfoController(MoviesInfoService moviesInfoService) {
16          this.moviesInfoService = moviesInfoService;
17      }
18
19      @PostMapping("/movieinfos")
20      @ResponseStatus(HttpStatus.CREATED)
21      public Mono<MovieInfo> addMovieInfo(@RequestBody MovieInfo movieInfo){
22
23          return moviesInfoService.addMovieInfo(movieInfo).log();
24      }
25  }
26
27

```

Create service method. Here also constructor used as best practice.



```
reactive-spring-webflux - movies-info-service > src > main > java > com > reactivessrping > service > MoviesInfoService > addMovieInfo

8  @Service
9
10 public class MoviesInfoService {
11
12     private MovieInfoRepository movieInfoRepository;
13
14     public MoviesInfoService(MovieInfoRepository movieInfoRepository) {
15         this.movieInfoRepository = movieInfoRepository;
16     }
17
18     public Mono<MovieInfo> addMovieInfo(MovieInfo movieInfo) {
19         return movieInfoRepository.save(movieInfo);
20     }
21 }
22
```

final → reactive-spring-webflux / movies-info-service / src / main / resources / curl-commands.txt

dilipsundarraj1 added the curl commands

1 contributor

36 lines (29 sloc) | 1.27 KB

All curl commands used for testing



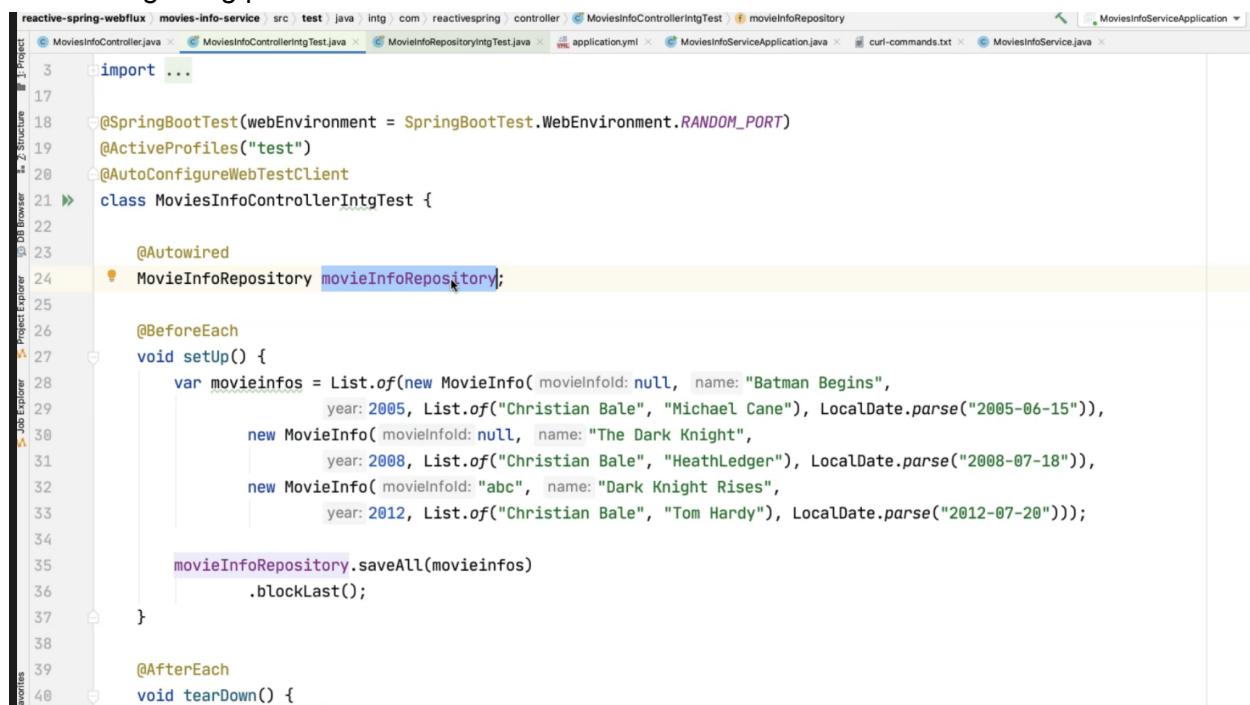
```
Terminal: Local +  
/Dilip/udemy/reactive-spring-webFlux/course-codebase/reactive-spring-webFlux > curl -i \  
-d '{"movieInfoId":1, "name": "Batman Begins", "year":2005,"cast":["Christian Bale", "Michael Cane"],"release_date": "2005-06-15"}' \  
-H "Content-Type: application/json" \  
-X POST http://localhost:8088/v1/movieinfos  
HTTP/1.1 201 Created  
Content-Type: application/json  
Content-Length: 123  
  
{"movieInfoId":1,"name":"Batman Begins","year":2005,"cast":["Christian Bale","Michael Cane"],"release_date":"2005-06-15"}  
/Dilip/udemy/reactive-spring-webFlux/course-codebase/reactive-spring-webFlux >  
L      main()  c.r.moviesinfoserviceapplication  : Started moviesinfoserviceapplication in 2.257 seconds (JVM running for 2.05)  
[ctor-http-nio-2] reactor.Mono.UsingWhen.1  : onSubscribe(MonoUsingWhen.MonoUsingWhenSubscriber)  
[ctor-http-nio-2] reactor.Mono.UsingWhen.1  : request(unbounded)  
[ntLoopGroup-3-3] org.mongodb.driver.connection  : Opened connection [connectionId{localValue:3, serverValue:31}] to localhost:27017  
[ntLoopGroup-3-3] reactor.Mono.UsingWhen.1  : onNext(MovieInfo{movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Ca  
[ntLoopGroup-3-3] reactor.Mono.UsingWhen.1  : onComplete()
```

Passing null id

```
/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i \
\ -d '{"movieInfoId":null, "name": "Dark Knight Rises", "year":2012,"cast":["Christian Bale", "Tom Hardy"],"release_date": "2012-07-20"}' \
\ -H "Content-Type: application/json"
\ -X POST http://localhost:8080/v1/movieinfos
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: 147

{"movieInfoId":"611e2b83d9fbccf17e795b287", "name":"Dark Knight Rises", "year":2012, "cast":["Christian Bale", "Tom Hardy"], "release_date": "2012-07-20"}
```

54. Integration Test for the POST endpoint (add movie) using JUnit5 - standard way of testing is this. Testing using postman is waste of time.



```
import ...

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
@AutoConfigureWebTestClient
class MovieInfoControllerIntgTest {

    @Autowired
    MovieInfoRepository movieInfoRepository;

    @BeforeEach
    void setUp() {
        var movieinfos = List.of(new MovieInfo(movieInfoId: null, name: "Batman Begins",
                                               year: 2005, List.of("Christian Bale", "Michael Caine"), LocalDate.parse("2005-06-15")),
                               new MovieInfo(movieInfoId: null, name: "The Dark Knight",
                                               year: 2008, List.of("Christian Bale", "Heath Ledger"), LocalDate.parse("2008-07-18")),
                               new MovieInfo(movieInfoId: "abc", name: "Dark Knight Rises",
                                               year: 2012, List.of("Christian Bale", "Tom Hardy"), LocalDate.parse("2012-07-20")));

        movieInfoRepository.saveAll(movieinfos)
            .blockLast();
    }

    @AfterEach
    void tearDown() {
```

@SpringBootTest will spin up the application for testing. @ActiveProfile("test") will use embedded mongo db and @AutoConfigureWebTestClient give webTestClient to connect endpoints.

```

18
19  @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
20  @ActiveProfiles("test")
21  @AutoConfigureWebTestClient
22  class MoviesInfoControllerIntgTest {
23
24      @Autowired
25      MovieInfoRepository movieInfoRepository;
26
27      @Autowired
28      WebTestClient webTestClient;
29
30      static String MOVIES_INFO_URL = "/v1/movieinfos"

```

```

@Text
@Test
void addMovieInfo() {
    //given
    var movieInfo = new MovieInfo(movieInfoId: null, name: "Batman Begins1",
        year: 2005, List.of("Christian Bale", "Michael Caine"), LocalDate.parse("2005-06-15"));

    //when
    webTestClient
        .post() WebTestClient.RequestBodyUriSpec
        .uri(MOVIES_INFO_URL) WebTestClient.RequestBodySpec
        .bodyValue(movieInfo) WebTestClient.RequestHeadersSpec<capture of ?>
        .exchange() WebTestClient.ResponseSpec
        .expectStatus() StatusAssertions
        .isCreated() WebTestClient.ResponseSpec
        .expectBody(MovieInfo.class) WebTestClient.BodySpec<MovieInfo, capture of ?>
        .consumeWith(movieInfoEntityExchangeResult -> {

            var savedMovieInfo = movieInfoEntityExchangeResult.getResponseBody();
            assert savedMovieInfo!=null;
            assert savedMovieInfo.getMovieInfoId()!=null;
        });
    //then
}
}

```

```

5] org.mongodb.driver.connection          : Opened connection {connectionId{localValue:5, serverValue:5}} to localhost:58445
1] reactor.Mono.UsingWhen.1             : onSubscribe(MonoUsingWhen$MonoUsingWhenSubscriber)
1] reactor.Mono.UsingWhen.1             : request(unbounded)
3] reactor.Mono.UsingWhen.1             : onNext(MovieInfo{_id=611e2e31c5dbd1131b67ae80, name=Batman Begins1, year=2005, ca
3] reactor.Mono.UsingWhen.1             : onComplete()

```

Get all movies endpoint integration testing

The screenshot shows a Java code editor with two files open:

- MoviesInfoController.java:** Contains the method `getAllMovieInfos()` which returns a Flux of `MovieInfo` objects from the `moviesInfoService`. A yellow circle highlights the return statement.
- MovieInfoRepository.java:** Contains the method `findAll()` which returns a Flux of `MovieInfo` objects from the `movieInfoRepository`.
- Test class:** Contains a test method `getAllMovieInfos()` using `WebTestClient` to verify the endpoint. It checks the status, body list, and size of the response.
- Logs:** Shows the command-line output of the test execution, including logs from MongoDB and the reactor framework.

Get movie by id endpoint integration testing

The screenshot shows a Java code editor with one file open:

- MoviesInfoController.java:** Contains the method `getMovieInfoById(@PathVariable String id)` which returns a Mono of `MovieInfo` object from the `moviesInfoService`. A yellow circle highlights the return statement.

```

29
30     public Mono<MovieInfo> getMovieInfoById(String id) {
31         return movieInfoRepository.findById(id);
32     }

```

```

var movieInfoId = "abc";
webTestClient
    .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
    .uri( uri: MOVIES_INFO_URL+("/{id}", movieInfoId) capture of ?
    .exchange() WebTestClient.ResponseSpec
    .expectStatus() StatusAssertions
    .is2xxSuccessful() WebTestClient.ResponseSpec
    .expectBody(MovieInfo.class) WebTestClient.BodySpec<MovieInfo>
    .consumeWith(movieInfoEntityExchangeResult -> {
        var movieInfo = movieInfoEntityExchangeResult.getResponseBody();
        assertNotNull(movieInfo);
    });
});

Flux<MovieInfo> getAllMovieInfos(){
    return moviesInfoService.getAllMovieInfos().log();
}

ping("/movieinfos/{id}")
Mono<MovieInfo> getMovieInfoById(@PathVariable String id){
    return moviesInfoService.getMovieInfoById(id);
}

ping("/movieinfos")
seStatus(HttpStatus.CREATED)
Mono<MovieInfo> addMovieInfo(@RequestBody MovieInfo movieInfo)
    return moviesInfoService.addMovieInfo(movieInfo).log();
}

```

Json path assertion

```

91     getMovieInfoById() {
92
93         var movieInfoId = "abc";
94         webTestClient
95             .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
96             .uri( uri: MOVIES_INFO_URL+("/{id}", movieInfoId) capture of ?
97             .exchange() WebTestClient.ResponseSpec
98             .expectStatus() StatusAssertions
99             .is2xxSuccessful() WebTestClient.ResponseSpec
100            .expectBody() WebTestClient.BodyContentSpec
101            .jsonPath( expression: "$.name").isEqualTo( expectedValue: "Dark Knight")
102            /*.expectBody(MovieInfo.class)
103            .consumeWith(movieInfoEntityExchangeResult -> {
104                var movieInfo = movieInfoEntityExchangeResult.getResponseBody();
105                assertNotNull(movieInfo);
106            });*/
107
108     }

Flux<MovieInfo> getAllMovieInfos(){
    return moviesInfoService.getAllMovieInfos().log();
}

ping("/movieinfos/{id}")
Mono<MovieInfo> getMovieInfoById(@PathVariable String id){
    return moviesInfoService.getMovieInfoById(id);
}

ping("/movieinfos")
seStatus(HttpStatus.CREATED)
Mono<MovieInfo> addMovieInfo(@RequestBody MovieInfo movieInfo)
    return moviesInfoService.addMovieInfo(movieInfo).log();
}

```

update movie by id endpoint integration testing

Whenever we have an operation that is going to return a reactive type then we will use flatmap.

@Service layer code

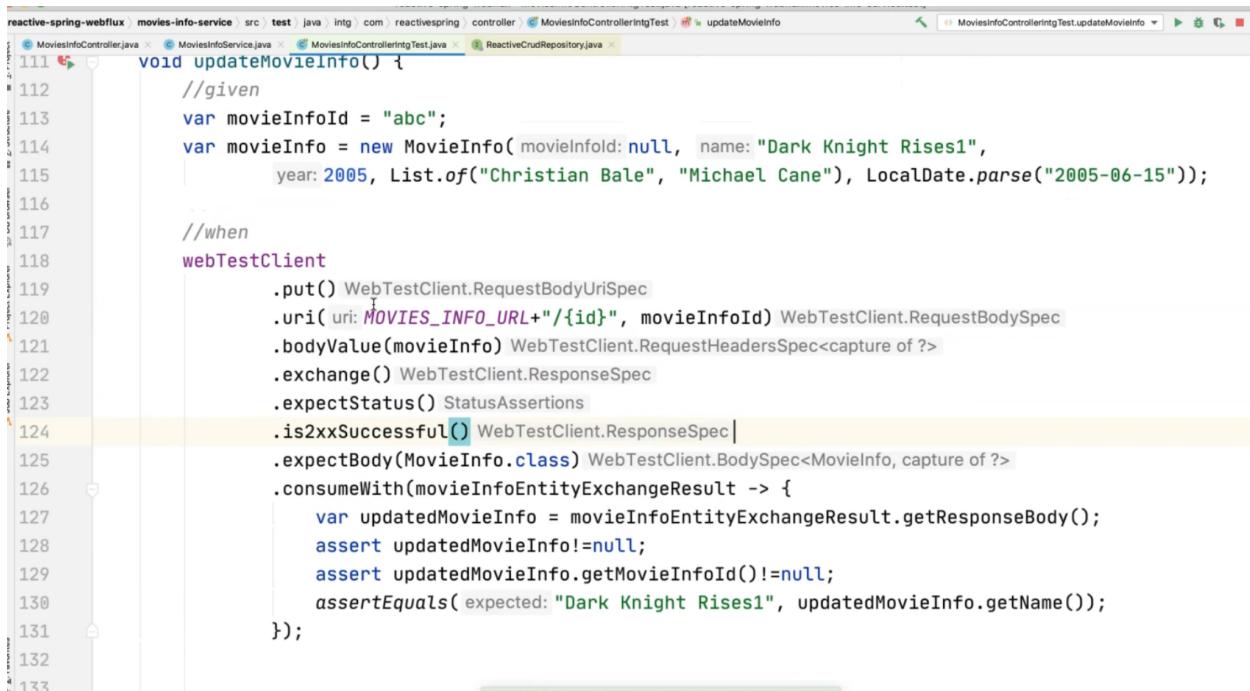
```

public Mono<MovieInfo> updateMovieInfo(MovieInfo updatedMovieInfo, String id) {
    return movieInfoRepository.findById(id)
        .flatMap(movieInfo -> {
            movieInfo.setCast(updatedMovieInfo.getCast());
            movieInfo.setName(updatedMovieInfo.getName());
            movieInfo.setRelease_date(updatedMovieInfo.getRelease_date());
            movieInfo.setYear(updatedMovieInfo.getYear());
            return movieInfoRepository.save(movieInfo);
        });
}

```

@Controller layer code

```
    @PutMapping("/movieinfos/{id}")
    public Mono<MovieInfo> updateMovieInfo(@RequestBody MovieInfo updatedMovieInfo, @PathVariable String id){
        return moviesInfoService.updateMovieInfo(updatedMovieInfo, id);
    }
```



Screenshot of an IDE showing Java code for updating a movie info endpoint. The code is annotated with numbers 111 through 133. A yellow circle highlights the path variable '{id}' in the PUT mapping. The code uses WebTestClient to perform a PUT request to the endpoint '/MOVIES_INFO_URL/{id}' with a movie info object containing a name ('Dark Knight Rises1') and year (2005). It then exchanges the response and consumes it to verify the updated movie info object.

```
111     void updateMovieInfo() {
112         //given
113         var movieInfoId = "abc";
114         var movieInfo = new MovieInfo( movieInfoId: null, name: "Dark Knight Rises1",
115             year: 2005, List.of("Christian Bale", "Michael Caine"), LocalDate.parse("2005-06-15"));
116
117         //when
118         webTestClient
119             .put() WebTestClient.RequestBodyUriSpec
120             .uri( uri: MOVIES_INFO_URL+"/"+id, movieInfoId) WebTestClient.RequestBodySpec
121             .bodyValue(movieInfo) WebTestClient.RequestHeadersSpec<capture of ?>
122             .exchange() WebTestClient.ResponseSpec
123             .expectStatus() StatusAssertions
124             .is2xxSuccessful() WebTestClient.ResponseSpec
125             .expectBody(MovieInfo.class) WebTestClient.BodySpec<MovieInfo, capture of ?>
126             .consumeWith(movieInfoEntityExchangeResult -> {
127                 var updatedMovieInfo = movieInfoEntityExchangeResult.getResponseBody();
128                 assert updatedMovieInfo!=null;
129                 assert updatedMovieInfo.getMovieInfoId()!=null;
130                 assertEquals( expected: "Dark Knight Rises1", updatedMovieInfo.getName());
131             });
132
133 }
```

Delete movie by id endpoint integration testing



Screenshot of an IDE showing Java code for deleting a movie info endpoint. The code is annotated with numbers 41 through 49. It defines a DELETE mapping for the endpoint '/movieinfos/{id}' with a response status of NO_CONTENT. The method returns a Mono<Void> and calls the deleteMovieInfo method from the moviesInfoService.

```
41
42
43     @DeleteMapping("/movieinfos/{id}")
44     @ResponseStatus(HttpStatus.NO_CONTENT)
45     public Mono<Void> deleteMovieInfo(@PathVariable String id){
46         return moviesInfoService.deleteMovieInfo(id);
47     }
48
49 }
```



Screenshot of an IDE showing Java code for deleting a movie info endpoint via repository. The code is annotated with numbers 1 through 10. It defines a DELETE mapping for the endpoint '/movieinfos/{id}' with a response status of NO_CONTENT. The method returns a Mono<Void> and calls the deleteById method from the movieInfoRepository.

```
1
2
3     public Mono<Void> deleteMovieInfo(String id) {
4
5         return movieInfoRepository.deleteById(id);
6
7     }
8
9
10 }
```