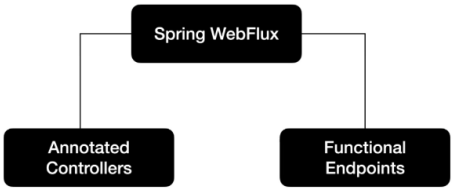
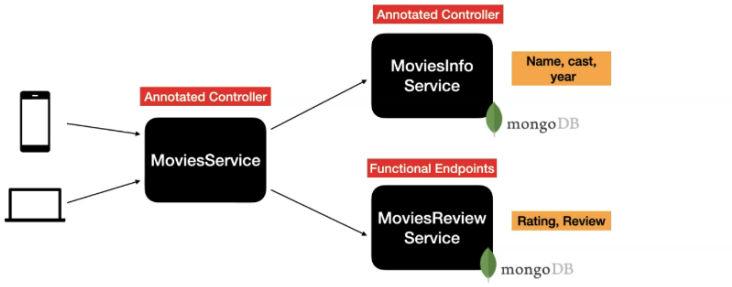
**This document contains Flux endpoints with get,post,put and delete and their unit testing**

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

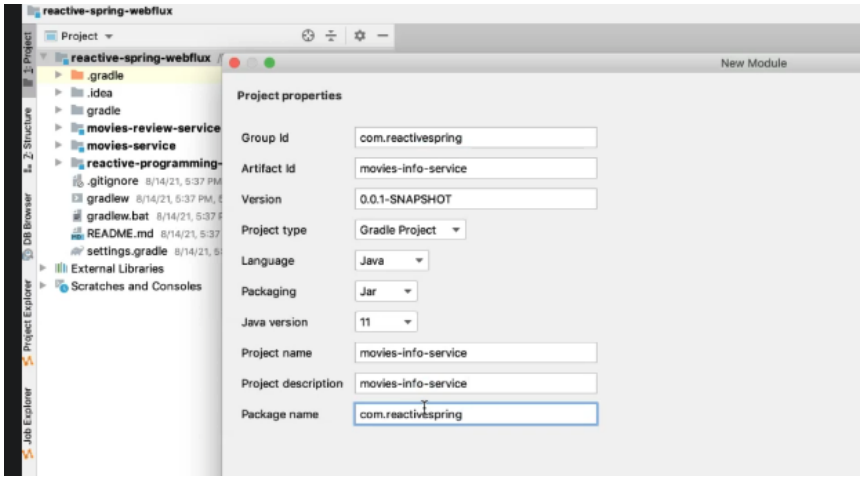
## Spring WebFlux

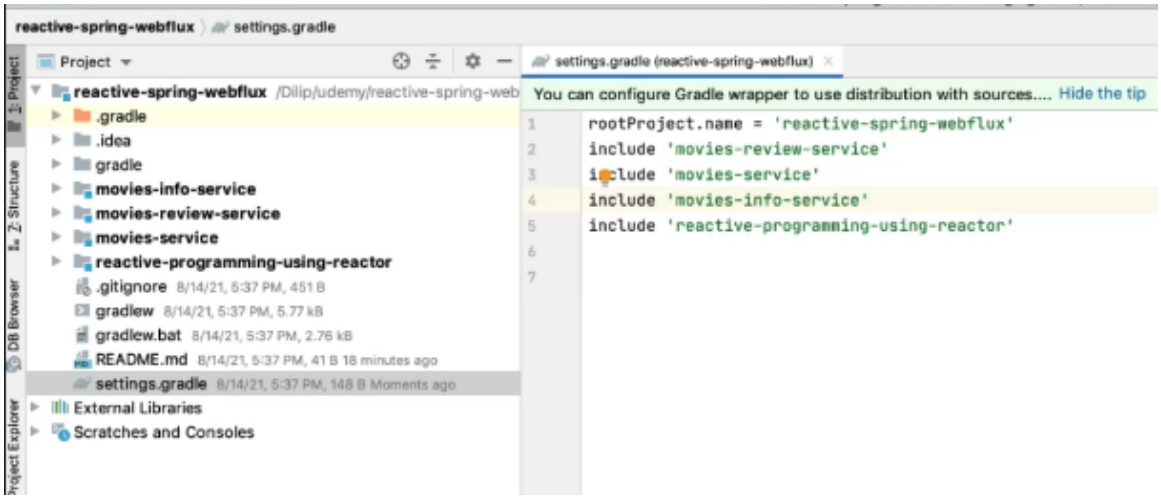



## Movies Application using MicroServices Pattern




Create new module

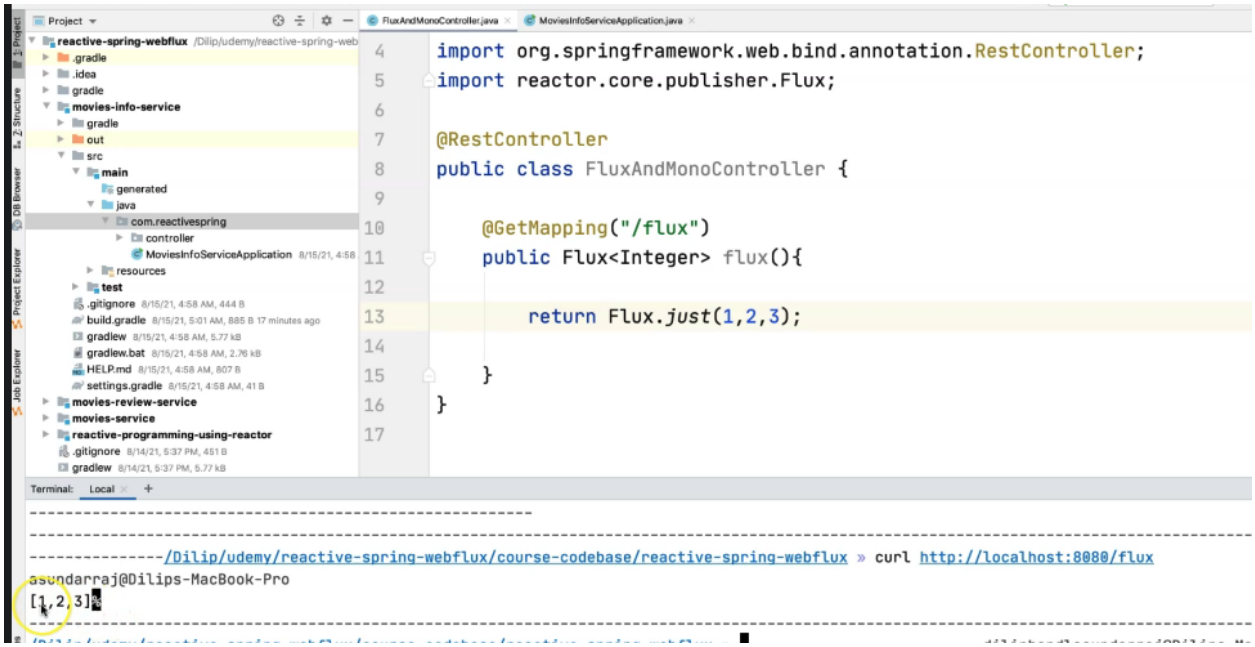




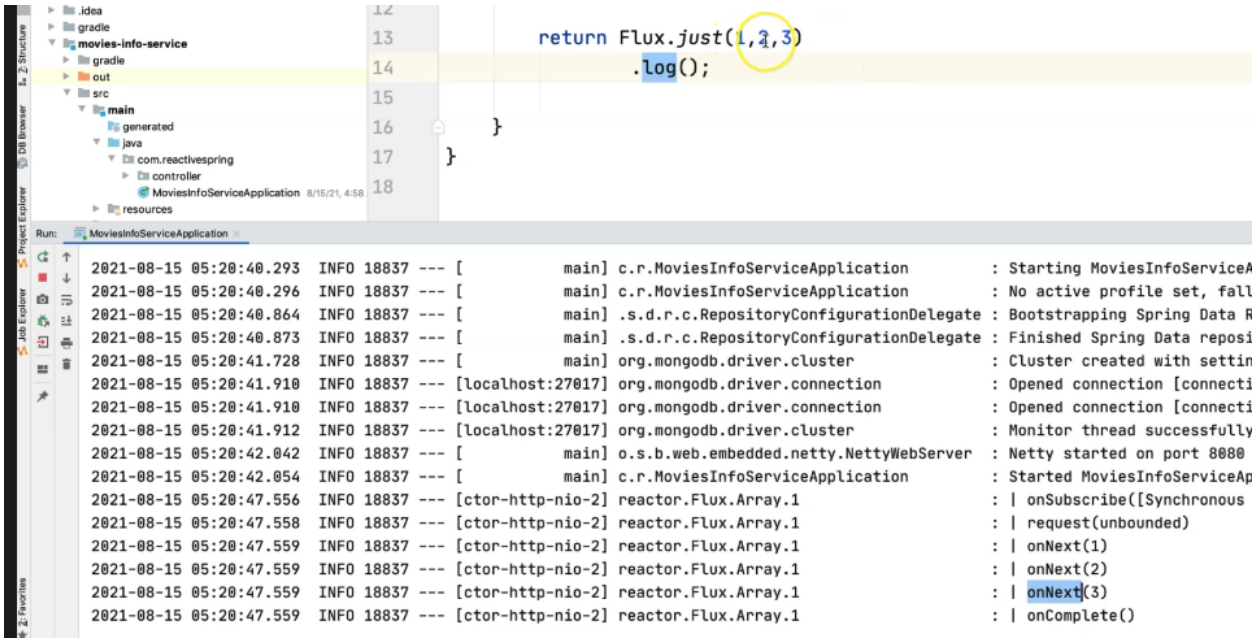
https://github.com/dilipsundarraj1/reactive-spring-webflux - readme file contain how to install mongo db.

And start the app it works

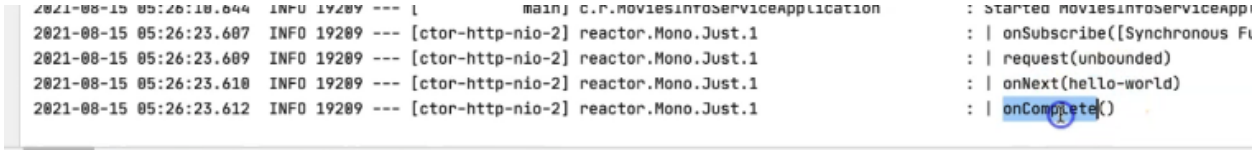



Create a controller that returns flux.

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




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


# 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

```java
@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 ir
2021-08-16 05:00:12.803  INFO 40888 --- [ctor-http-nio-2] reactor.Flux.Interval.1                : onSubscribe(FluxInterval.IntervalRunnat
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)
```

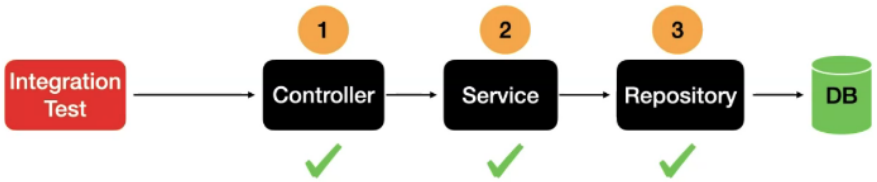
Notice the parallel thread started.

# 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

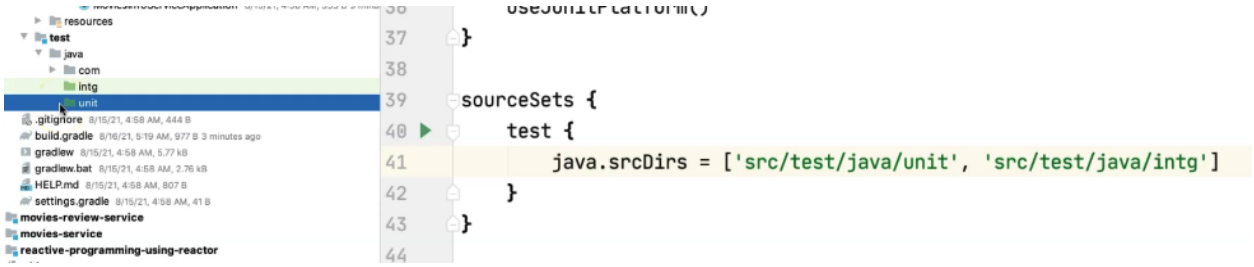



# Unit Tests

- 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.

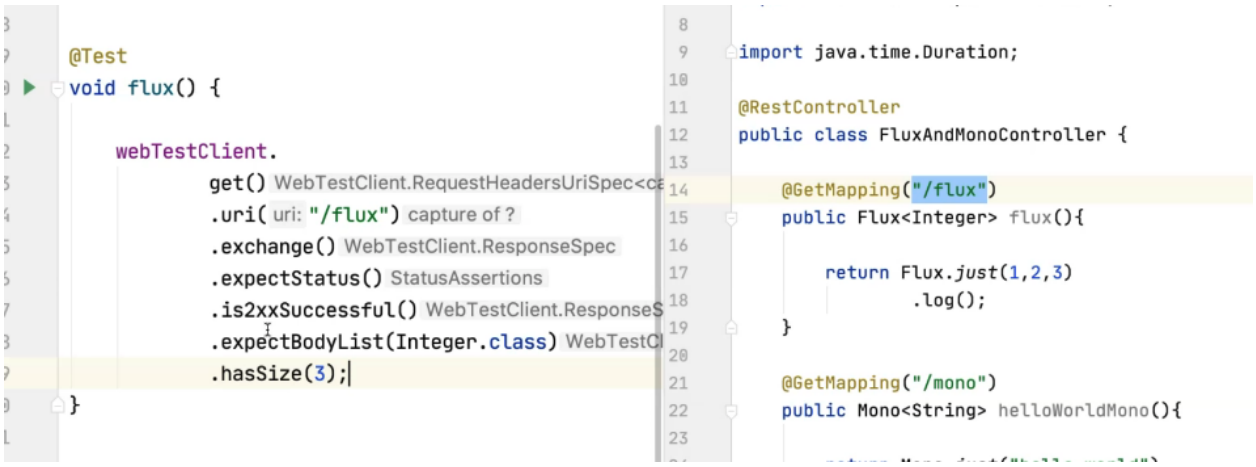


```java
package com.reactivespring.controller;

import ...

@WebFluxTest(controllers = FluxAndMonoController.class)
@AutoConfigureWebTestClient
class FluxAndMonoControllerTest {

    @Autowired
    WebTestClient webTestClient;


    @Test
    void flux() {
        //given

        //when

        //then
    }
}
```

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.



```java
@Test
void flux() {

    webTestClient.
            get() WebTestClient.RequestHeadersUriSpec<c
            .uri( uri: "/flux") capture of ?
            .exchange() WebTestClient.ResponseSpec
            .expectStatus() StatusAssertions
            .is2xxSuccessful() WebTestClient.ResponseS
            .expectBodyList(Integer.class) WebTestCl
            .hasSize(3);
}
```

```java
import java.time.Duration;

@RestController
public class FluxAndMonoController {

    @GetMapping("/flux")
    public Flux<Integer> flux(){

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

    @GetMapping("/mono")
    public Mono<String> helloWorldMono(){
```

Exchange method invokes the endpoint. Testing the size of response

```java
    @Test
    void flux_approach2() {

        var flux = webTestClient.
                get() WebTestClient.RequestHeadersUriSpec<capture of ?>
                .uri( uri: "/flux") capture of ?
                .exchange() WebTestClient.ResponseSpec
                .expectStatus() StatusAssertions
                .is2xxSuccessful() WebTestClient.ResponseSpec
                .returnResult(Integer.class) FluxExchangeResult<Integer>
                .getResponseBody();

        StepVerifier.create(flux)
                .expectNext(1,2,3)
                .verifyComplete();
    }
}
```

Testing the response body

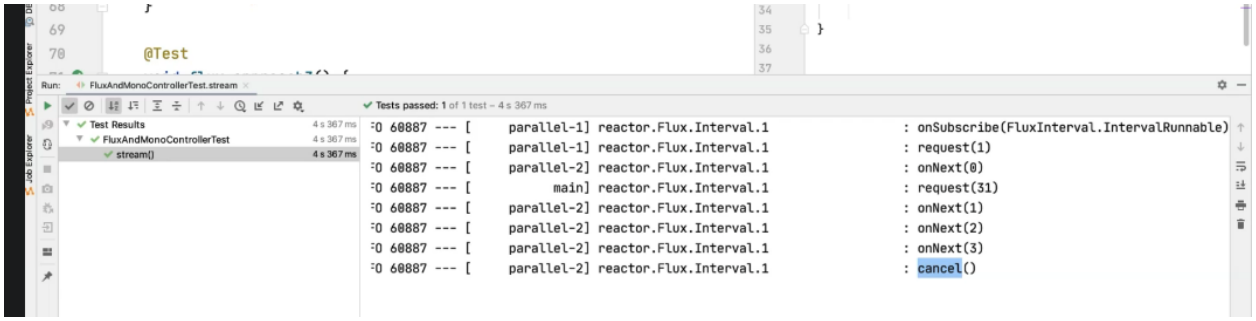
```java
        webTestClient.
                get() WebTestClient.RequestHeadersUriSpec<capture of ?>
                .uri( uri: "/flux") capture of ?
                .exchange() WebTestClient.ResponseSpec
                .expectStatus() StatusAssertions
                .is2xxSuccessful() WebTestClient.ResponseSpec
                .expectBodyList(Integer.class) WebTestClient.ListBodySpec<Integer>
                .consumeWith(listEntityExchangeResult -> {

                    var responseBody = listEntityExchangeResult.getResponseBody();
                    assert (Objects.requireNonNull(responseBody).size() == 3);

                });


    }
}
```

Approach 3

```
@Test
void stream() {

    var flux = webTestClient.
            get() WebTestClient.RequestHeadersUriSpec<capture of ?>
            .uri( uri: "/stream") capture of ?
            .exchange() WebTestClient.ResponseSpec
            .expectStatus() StatusAssertions
            .is2xxSuccessful() WebTestClient.ResponseSpec
            .returnResult(Long.class) FluxExchangeResult<Long>
            .getResponseBody();

    StepVerifier.create(flux) StepVerifier.FirstStep<Long>
            .expectNext(0L, 1L,2L,3L) StepVerifier.Step<Long>
            .thenCancel() StepVerifier
            .verify();
}
```

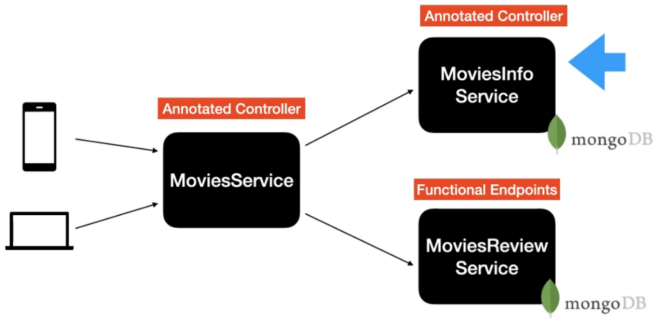
```
@GetMapping("/flux")
public Flux<Integer> flux(){

    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_E
public Flux<Long> stream(){

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

}
```

Testing mono endpoint

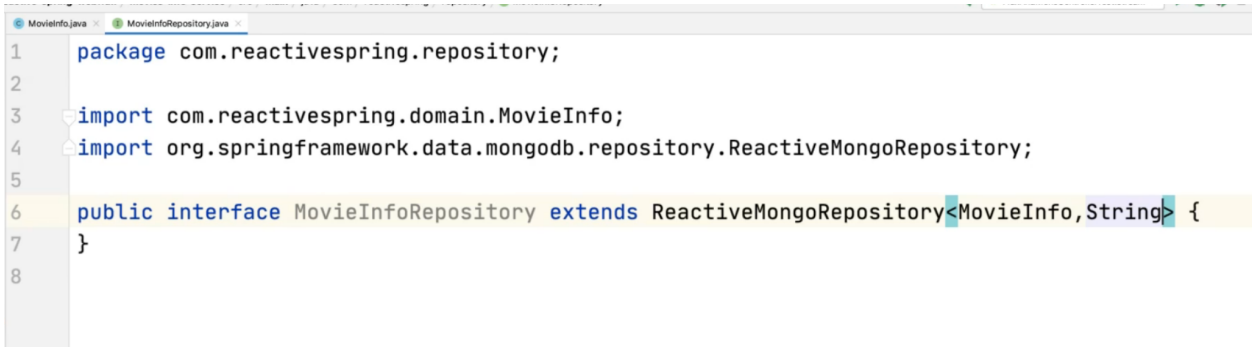


```
@Test

Run: FluxAndMonoControllerTest.stream
Tests passed: 1 of 1 test – 4 s 367 ms
Test Results 4 s 367 ms
FluxAndMonoControllerTest 4 s 367 ms
stream() 4 s 367 ms

:0 60887 --- [    parallel-1] reactor.Flux.Interval.1 : onSubscribe(FluxInterval.IntervalRunnable)
:0 60887 --- [    parallel-1] reactor.Flux.Interval.1 : request(1)
:0 60887 --- [    parallel-2] 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

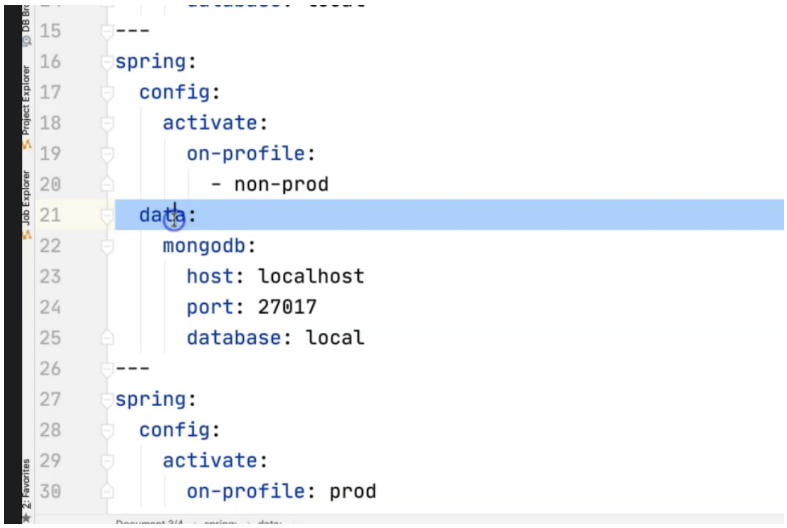


```java

@Data
@NoArgsConstructor
@AllArgsConstructor
@Document
public class MovieInfo {

    @Id
    private String movieInfoId;
    private String name;
    private Integer year;
    private List<String> cast;
    private LocalDate release_date;
}

```



MovieInfo.java | MovieInfoRepository.java | application.yml

```yaml
spring:
  profiles:
    active: local
---
spring:
  config:
    activate:
      on-profile:
        - local
  data:
    mongodb:
      host: localhost
      port: 27017
      database: local
```



```yaml
---
spring:
  config:
    activate:
      on-profile:
        - non-prod
  data:
    mongodb:
      host: localhost
      port: 27017
      database: local
---
spring:
  config:
    activate:
      on-profile: prod
```

```yaml
---
spring:
  config:
    activate:
      on-profile: prod
  data:
    mongodb:
      host: localhost
      port: 27017
      database: local
```

47. Setup the Integration Test using @DataMongoTest

```
testImplementation 'de.flapdoodle.embed:de.flapdoodle.embed.mongo'
testImplementation 'io.projectreactor:reactor-test'
```

This dependency gives embed mongo  just like h2db.

```java
package com.reactivespring.repository;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.data.mongo.DataMongoTest;
import org.springframework.test.context.ActiveProfiles;

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

@DataMongoTest
@ActiveProfiles("test")
class MovieInfoRepositoryIntgTest {

    @Autowired
    MovieInfoRepository movieInfoRepository;

}
```

@DataMonoTest 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.

```java
@BeforeEach
void setUp() {
    var movieinfos = List.of(new MovieInfo( movieInfoId: null,  name: "Batman Begins",
                    year: 2005, List.of("Christian Bale", "Michael Cane"), LocalDate.parse("2005-06-15")),
            new MovieInfo( movieInfoId: null,  name: "The Dark Knight",
                    year: 2008, List.of("Christian Bale", "HeathLedger"), 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

```java
@Test
void findAll() {
    //given

    //when
    var moviesInfoFlux = movieInfoRepository.findAll().log();

    //then
    StepVerifier.create(moviesInfoFlux)
            .expectNextCount(3)
            .verifyComplete();
```

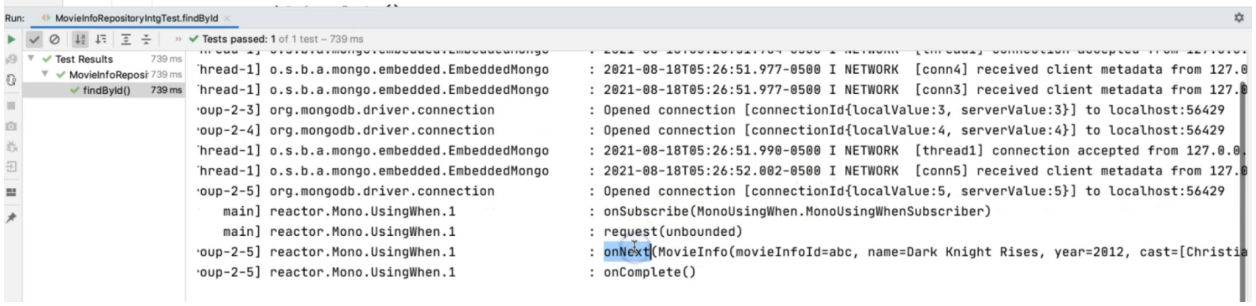
```
Run:  MovieInfoRepositoryIntgTest.findAll
Tests passed: 1 of 1 test – 683 ms
Test Re 683 ms
Mov 683 ms
f 683 ms
edded.EmbeddedMongo     : 2021-08-18T05:19:25.262-0500 I NETWORK  [thread1] connection accepted from 127.0.0.1:55502 #5 (5 connect
edded.EmbeddedMongo     : 2021-08-18T05:19:25.273-0500 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
gWhen.1                 : onSubscribe(FluxUsingWhen.UsingWhenSubscriber)
gWhen.1                 : request(unbounded)
gWhen.1                 : onNext(MovieInfo(movieInfoId=611cdead9f2149125440a784, name=Batman Begins, year=2005, cast=[Christian Ba
gWhen.1                 : onNext(MovieInfo(movieInfoId=611cdead9f2149125440a785, name=The Dark Knight, year=2008, cast=[Christian
gWhen.1                 : onNext(MovieInfo(movieInfoId=abc, name=Dark Knight Rises, year=2012, cast=[Christian Bale, Tom Hardy], r
gWhen.1                 : onComplete()
```

```
.f.embed.mongo.MongodExecutable         : start MongodConfig{version=Version{3.5.5}, timeout=de.flapdoodle.embed.mongo.config.Timed
'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 127.0.0.1:5549
```

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                })
```




```
65                .assertNext(movieInfo -> {
66                    assertEquals( expected: "Dark Knight Rises", movieInfo.getName());
67                })
68                .verifyComplete();
69        }
70    }
```

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

3. Save Method

```
11    @Test
12    void saveMovieinfo() {
13        //given
14        var movieInfo = new MovieInfo( movieInfoId: null, name: "Batman Begins1",
15                year: 2005, List.of("Christian Bale", "Michael Cane"), LocalDate.parse("2005-06-15"));
16
17        //when
18        var moviesInfoMono = movieInfoRepository.save(movieInfo).log();
19
20        //then
21        StepVerifier.create(moviesInfoMono)
22                //.expectNextCount(1)
23                .assertNext(movieInfo1 -> {
24                    assertNotNull(movieInfo1.getMovieInfoId());
25                    assertEquals( expected: "Batman Begins1", movieInfo1.getName());
26                })
27                .verifyComplete();
28    }
29 }
```

Build 4: Run DB Execution Console 6: TODO 0: Messages Terminal

```
n.1                : onSubscribe(MonoUsingWhen.MonoUsingWhenSubscriber)
n.1                : request(unbounded)
n.1                : onNext(MovieInfo(movieInfoId=611ce17a4ec4f42acf8697a5, 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 }
```

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

Delete method.

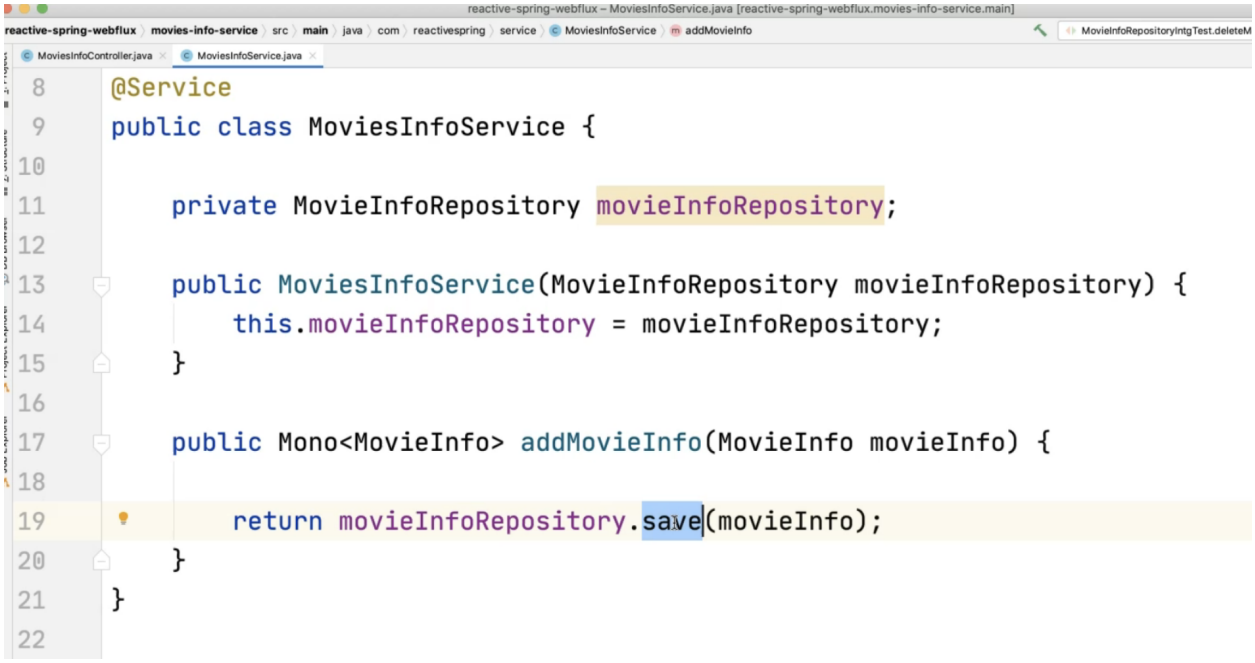
```
108
109     @Test
110     void deleteMovieInfo() {
111         //given
112
113         //when
114         movieInfoRepository.deleteById("abc").log();
115         var moviesInfoFlux = movieInfoRepository.findAll().log();
116
117         //then
118         StepVerifier.create(moviesInfoFlux)
119 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

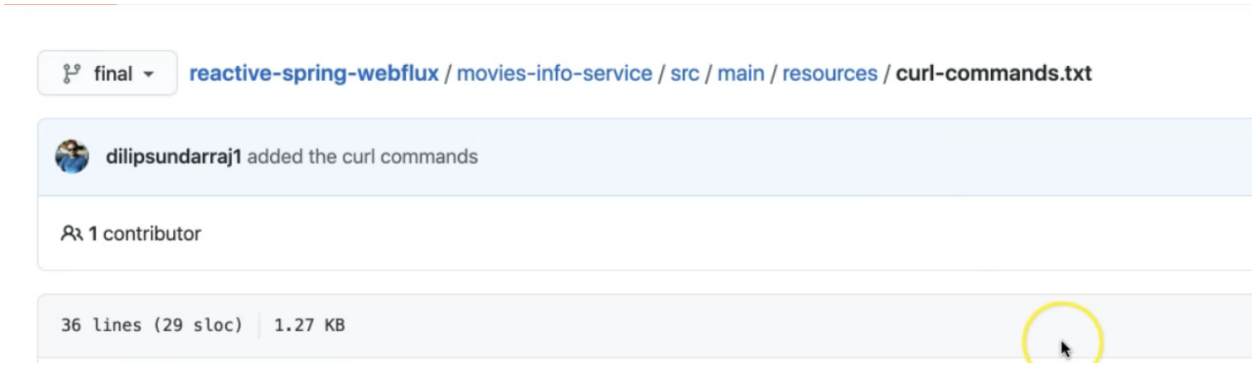

Abc got deleted. We have to replace log() with block() to block asynchronous calls during unit testing..

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

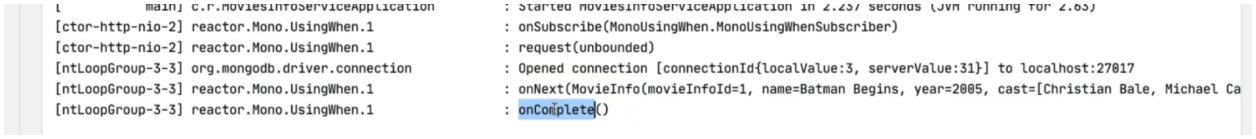
```java
@RestController
@RequestMapping("/v1")
public class MoviesInfoController {

    private MoviesInfoService moviesInfoService;

    public MoviesInfoController(MoviesInfoService moviesInfoService) {
        this.moviesInfoService = moviesInfoService;
    }

    @PostMapping("/movieinfos")
    @ResponseStatus(HttpStatus.CREATED)
    public Mono<MovieInfo> addMovieInfo(@RequestBody MovieInfo movieInfo){

        return moviesInfoService.addMovieInfo(movieInfo).log();

    }
}
```

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



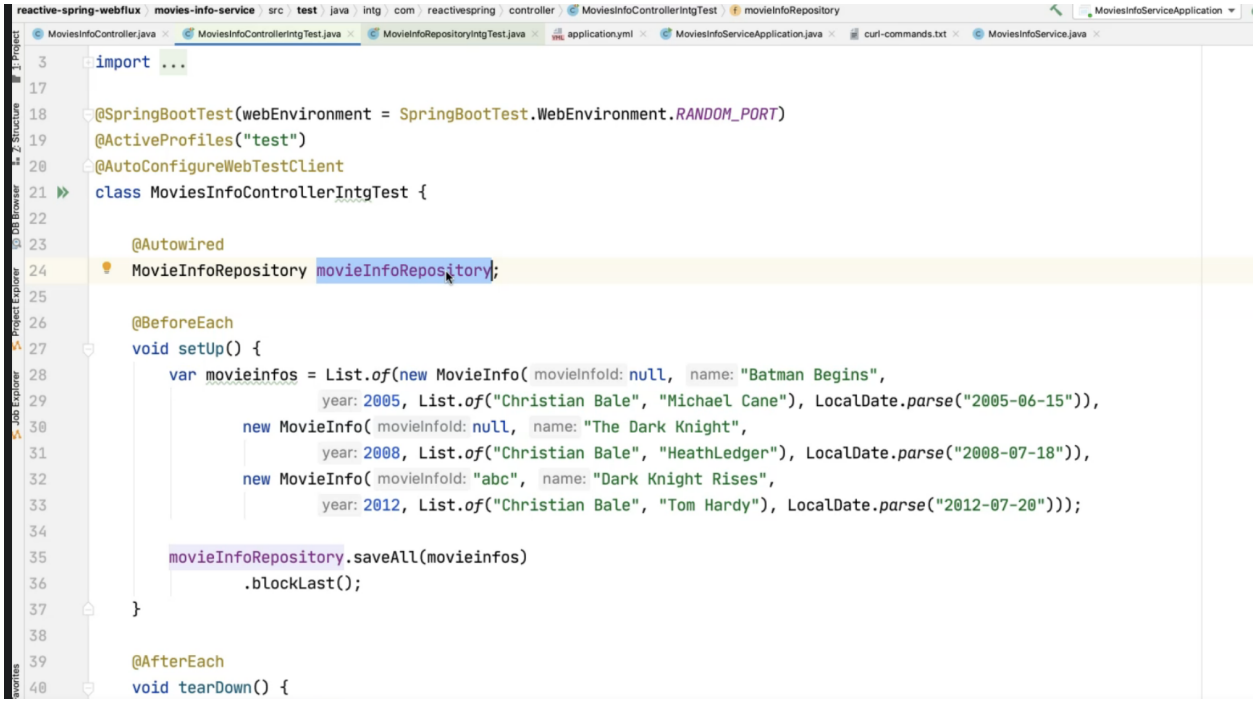

All curl commands used for testing

Passing null id

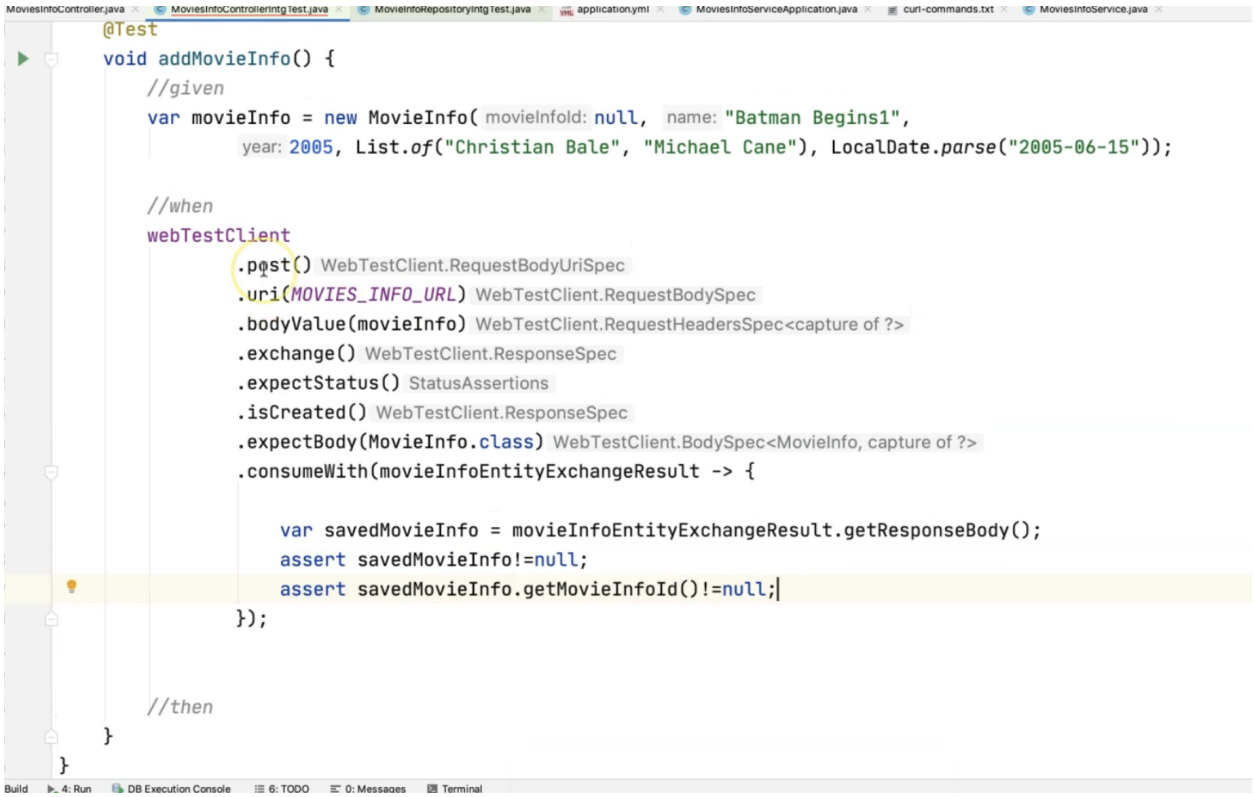
```
/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i \                    dilipbandlasundarraj@Dilips-MacBook-Pro
\ -d '{"movieInfoId":null, "name": "Dark Knight Rises", "year":2012,"cast":["Christian Bale", "Tom Hardy"],"release_date": "2012-07-20"}' \
\ -H "Content-Type: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

{"movieInfoId":"611e2b83d9fbcf17e795b287","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.




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

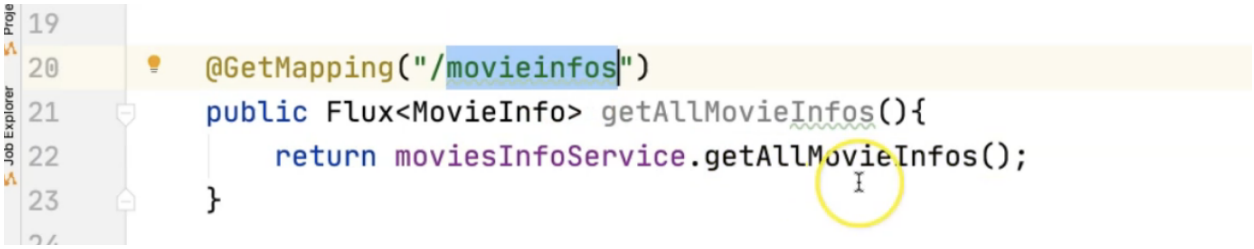
```java

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

    @Autowired
    MovieInfoRepository movieInfoRepository;

    @Autowired
    WebTestClient webTestClient;

    static String MOVIES_INFO_URL = "/v1/movieinfos";
```



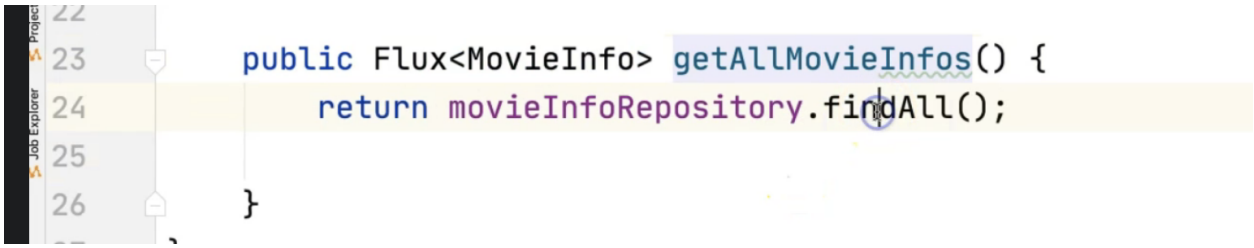
```java
    @Test
    void addMovieInfo() {
        //given
        var movieInfo = new MovieInfo( movieInfoId: null,  name: "Batman Begins1",
                year: 2005, List.of("Christian Bale", "Michael Cane"), LocalDate.parse("2005-06-15"));

        //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(movieInfoId=611e2e31c5dbd1131b67ae80, name=Batman Begins1, year=2005, ca
3] reactor.Mono.UsingWhen.1                : onComplete()
```

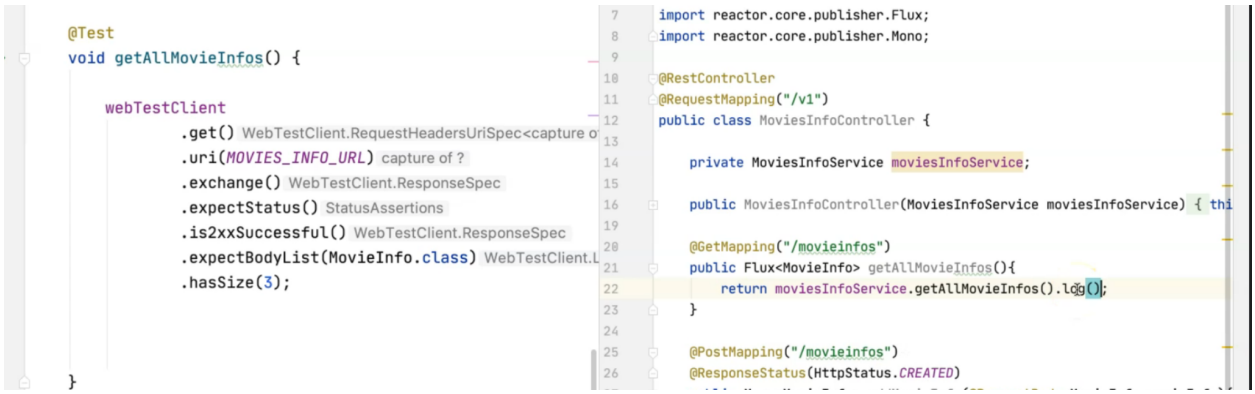
Get all movies endpoint integration testing



```
19
20      @GetMapping("/movieinfos")
21      public Flux<MovieInfo> getAllMovieInfos(){
22          return moviesInfoService.getAllMovieInfos();
23      }
```



```
22
23      public Flux<MovieInfo> getAllMovieInfos() {
24          return movieInfoRepository.findAll();
25
26      }
```



```
@Test
void getAllMovieInfos() {

    webTestClient
            .get() WebTestClient.RequestHeadersUriSpec<capture o
            .uri(MOVIES_INFO_URL) capture of ?
            .exchange() WebTestClient.ResponseSpec
            .expectStatus() StatusAssertions
            .is2xxSuccessful() WebTestClient.ResponseSpec
            .expectBodyList(MovieInfo.class) WebTestClient.L
            .hasSize(3);

}
```

```
7   import reactor.core.publisher.Flux;
8   import reactor.core.publisher.Mono;
9
10  @RestController
11  @RequestMapping("/v1")
12  public class MoviesInfoController {
13
14      private MoviesInfoService moviesInfoService;
15
16      public MoviesInfoController(MoviesInfoService moviesInfoService) { thi
19
20      @GetMapping("/movieinfos")
21      public Flux<MovieInfo> getAllMovieInfos(){
22          return moviesInfoService.getAllMovieInfos().log();
23      }
24
25      @PostMapping("/movieinfos")
26      @ResponseStatus(HttpStatus.CREATED)
```

```
2021-08-19 05:19:17.049  INFO 5531 --- [ntLoopGroup-3-5] org.mongodb.driver.connection          : Opened connection [connectionId{localValue:5, serverValue:5}] to loca
2021-08-19 05:19:17.251  INFO 5531 --- [     parallel-1] reactor.Flux.UsingWhen.1                : onSubscribe(FluxUsingWhen.UsingWhenSubscriber)
2021-08-19 05:19:17.252  INFO 5531 --- [     parallel-1] reactor.Flux.UsingWhen.1                : request(unbounded)
2021-08-19 05:19:17.296  INFO 5531 --- [ntLoopGroup-3-5] reactor.Flux.UsingWhen.1               : onNext(MovieInfo(movieInfoId=611e30253761cd1898d37178, name=The Dark
2021-08-19 05:19:17.306  INFO 5531 --- [ntLoopGroup-3-5] reactor.Flux.UsingWhen.1               : onNext(MovieInfo(movieInfoId=611e30253761cd1898d37177, name=Batman Be
2021-08-19 05:19:17.307  INFO 5531 --- [ntLoopGroup-3-5] reactor.Flux.UsingWhen.1               : onNext(MovieInfo(movieInfoId=abc, name=Dark Knight Rises, year=2012,
2021-08-19 05:19:17.307  INFO 5531 --- [ntLoopGroup-3-5] reactor.Flux.UsingWhen.1               : onComplete()
```

Get movie by id endpoint integration testing

```
24
25      @GetMapping("/movieinfos/{id}")
26      public Mono<MovieInfo> getMovieInfoById(@PathVariable String id){
27          return moviesInfoService.getMovieInfoById(id).log();
28      }
29
```

```
29
30      public Mono<MovieInfo> getMovieInfoById(String id) {
31          return movieInfoRepository.findById(id);
32      }
33  }
```

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

```
Flux<MovieInfo> getAllMovieInfos(){
urn moviesInfoService.getAllMovieInfos().log();


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


pping("/movieinfos")
seStatus(HttpStatus.CREATED)
Mono<MovieInfo> addMovieInfo(@RequestBody MovieInfo movieInf
```

Json path assertion

```
getMovieInfoById() {

ar movieInfoId = "abc";
ebTestClient
        .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
        .uri( uri: MOVIES_INFO_URL+"/{id}", movieInfoId) capture of ?
        .exchange() WebTestClient.ResponseSpec
        .expectStatus() StatusAssertions
        .is2xxSuccessful() WebTestClient.ResponseSpec
        .expectBody() WebTestClient.BodyContentSpec
        .jsonPath( expression: "$.name").isEqualTo( expectedValue: "Dark K
        /*.expectBody(MovieInfo.class)
        .consumeWith(movieInfoEntityExchangeResult -> {
            var movieInfo = movieInfoEntityExchangeResult.getRespon
            assertNotNull(movieInfo);
        });*/
```

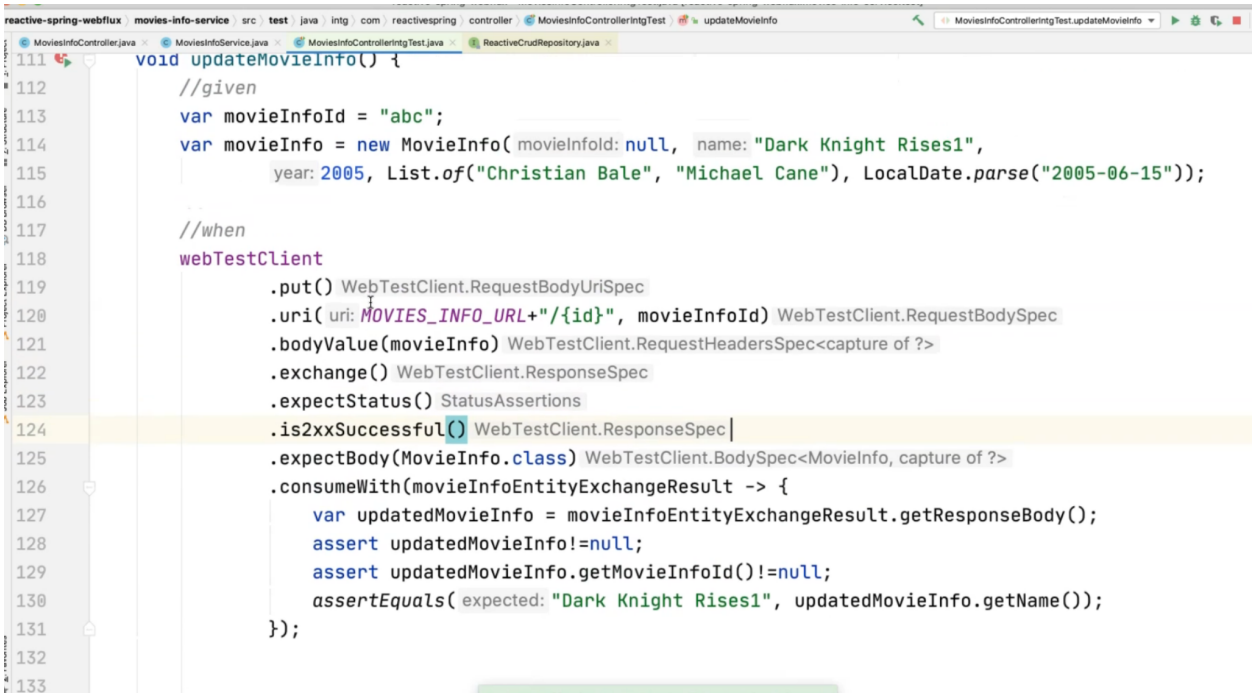
```
Flux<MovieInfo> getAllMovieInfos(){
urn moviesInfoService.getAllMovieInfos().log();


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


pping("/movieinfos")
seStatus(HttpStatus.CREATED)
Mono<MovieInfo> addMovieInfo(@RequestBody MovieInfo movieInfo
urn 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);

    }
}
```



```
    void updateMovieInfo() {
        //given
        var movieInfoId = "abc";
        var movieInfo = new MovieInfo( movieInfoId: null, name: "Dark Knight Rises1",
                year: 2005, List.of("Christian Bale", "Michael Cane"), LocalDate.parse("2005-06-15"));

        //when
        webTestClient
                .put() WebTestClient.RequestBodyUriSpec
                .uri( uri: MOVIES_INFO_URL+"/{id}", movieInfoId) WebTestClient.RequestBodySpec
                .bodyValue(movieInfo) WebTestClient.RequestHeadersSpec<capture of ?>
                .exchange() WebTestClient.ResponseSpec
                .expectStatus() StatusAssertions
                .is2xxSuccessful() WebTestClient.ResponseSpec
                .expectBody(MovieInfo.class) WebTestClient.BodySpec<MovieInfo, capture of ?>
                .consumeWith(movieInfoEntityExchangeResult -> {
                    var updatedMovieInfo = movieInfoEntityExchangeResult.getResponseBody();
                    assert updatedMovieInfo!=null;
                    assert updatedMovieInfo.getMovieInfoId()!=null;
                    assertEquals( expected: "Dark Knight Rises1", updatedMovieInfo.getName());
                });
```

Delete movie by id endpoint integration testing

```
    }

    @DeleteMapping("/movieinfos/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public Mono<Void> deleteMovieInfo(@PathVariable String id){
        return moviesInfoService.deleteMovieInfo(id);

    }
}
```

```
    public Mono<Void> deleteMovieInfo(String id) {
        return movieInfoRepository.deleteById(id);
    }
}
```