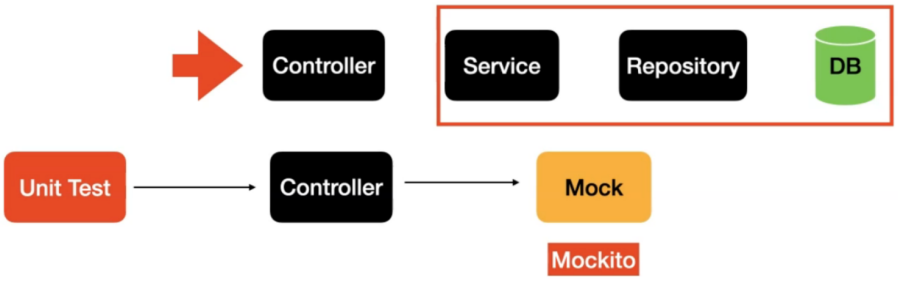
This document contains : Reactive unit testing, Bean Validations, using ResponseEntity in webflux, Custom Query in Reactive Mongo Repository, How netty works with spring webflux and Functional web module in webflux.

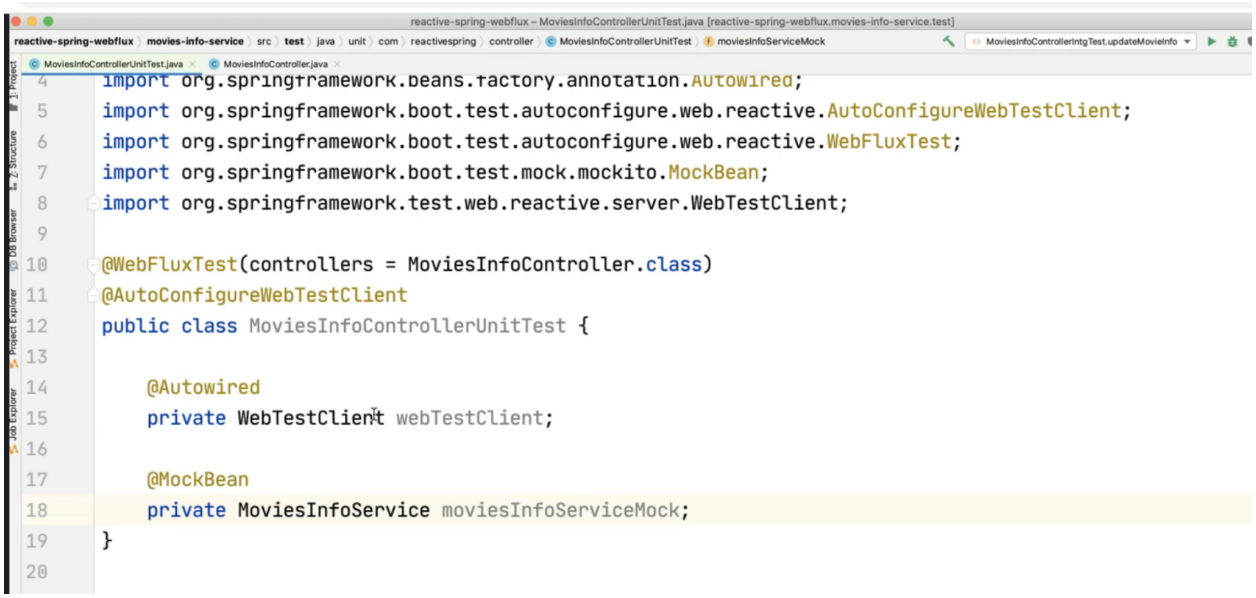
Unit Tests

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



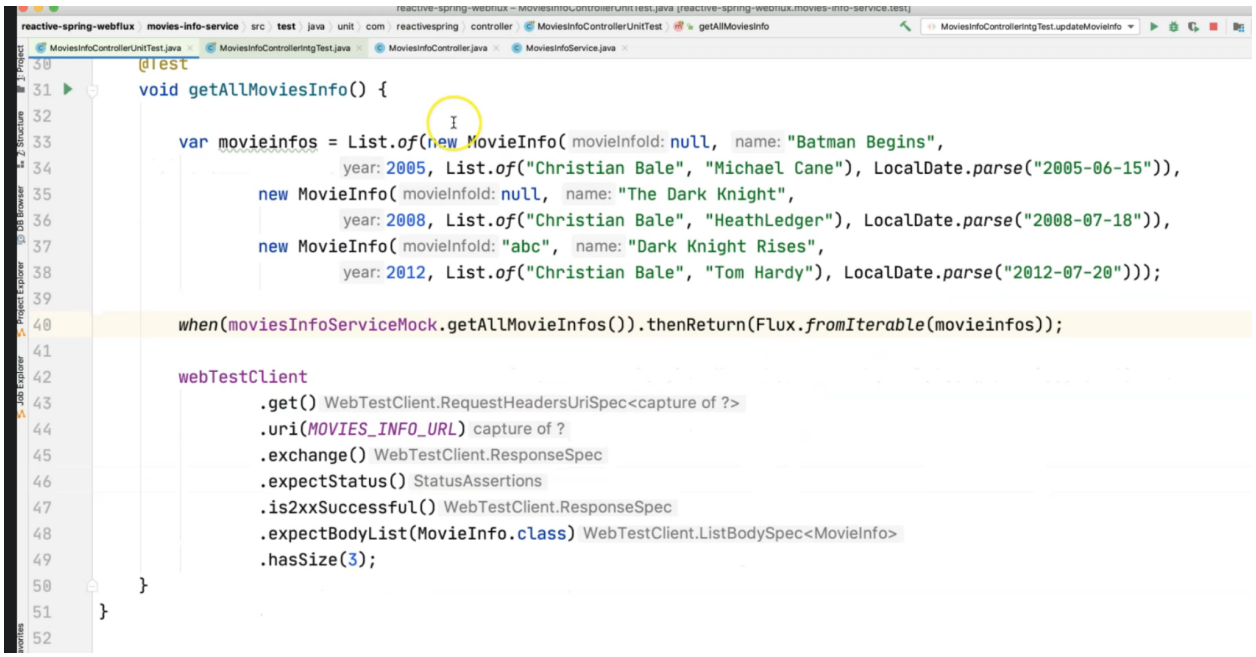
Benefits of Unit Tests

- Unit Tests are faster compared to Integration Tests
- Unit Tests are handy for performing Bean Validations



```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.reactive.AutoConfigureWebTestClient;
import org.springframework.boot.test.autoconfigure.web.reactive.WebFluxTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.test.web.reactive.server.WebTestClient;

@WebFluxTest(controllers = MoviesInfoController.class)
@AutoConfigureWebTestClient
public class MoviesInfoControllerUnitTest {

    @Autowired
    private WebTestClient webTestClient;

    @MockBean
    private MoviesInfoService moviesInfoServiceMock;
}
```

Get call unit testing



```java
    @Test
    void getAllMoviesInfo() {

        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")));

        when(moviesInfoServiceMock.getAllMovieInfos()).thenReturn(Flux.fromIterable(movieinfos));

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

Assignment: Unit Test for getMovieInfoById

15 minutes to complete | 201 student solutions

Write the Unit Test for getMovieInfoById

Post call unit testing

```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(moviesInfoServiceMock.addMovieInfo(isA(MovieInfo.class))).thenReturn(
            Mono.just(new MovieInfo( movieInfoId: "mockId",  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 -> {
```

```java
            .isCreated() WebTestClient.ResponseSpec
            .expectBody(MovieInfo.class) WebTestClient.BodySpec<MovieInfo, capture of ?>
            .consumeWith(movieInfoEntityExchangeResult -> {

                var savedMovieInfo = movieInfoEntityExchangeResult.getResponseBody();
                assert savedMovieInfo!=null;
                assert savedMovieInfo.getMovieInfoId()!=null;
                assertEquals( expected: "mockId",savedMovieInfo.getMovieInfoId());
            });
```

Put method unit test case.

```java
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(moviesInfoServiceMock.updateMovieInfo(isA(MovieInfo.class), isA(String.class))).thenReturn(
            Mono.just(new MovieInfo(movieInfoId,  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;
```

```
98
99          //when
100         webTestClient
101                 .put() WebTestClient.RequestBodyUriSpec
102                 .uri( uri: MOVIES_INFO_URL+"/{id}", movieInfoId) WebTestClient.RequestBodySpec
103                 .bodyValue(movieInfo) WebTestClient.RequestHeadersSpec<capture of ?>
104                 .exchange() WebTestClient.ResponseSpec
105                 .expectStatus() StatusAssertions
106                 .is2xxSuccessful() WebTestClient.ResponseSpec
107                 .expectBody(MovieInfo.class) WebTestClient.BodySpec<MovieInfo, capture of ?>
108                 .consumeWith(movieInfoEntityExchangeResult -> {
109                     var updatedMovieInfo = movieInfoEntityExchangeResult.getResponseBody();
110                     assert updatedMovieInfo!=null;
111                     assert updatedMovieInfo.getMovieInfoId()!=null;
112                     assertEquals( expected: "Dark Knight Rises1", updatedMovieInfo.getName());
113                 });
114
```

Bean validation:

```
implementation 'org.springframework.boot:spring-boot-starter-validation'
```

The above dependency is important for bean validation.

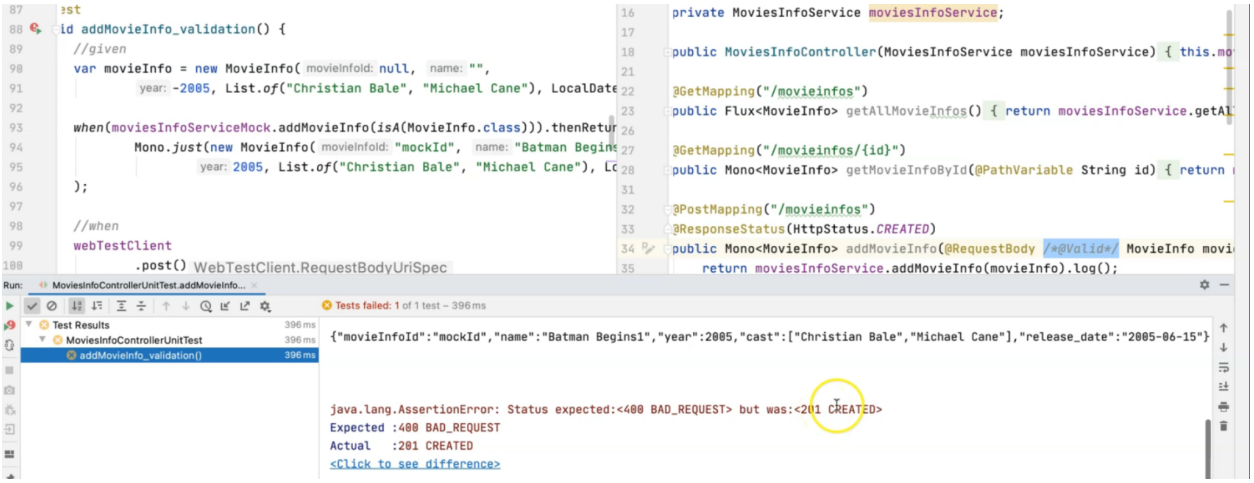
```
17  @AllArgsConstructor
18  @Document
19  public class MovieInfo {
20
21      @Id
22      private String movieInfoId;
23      @NotBlank(message = "movieInfo.name must be present")
24      private String name;
25      @NotNull
26      @Positive(message = "movieInfo.year must be a Positive value")
27      private Integer year;
28      private List<String> cast;
29      private LocalDate release_date;
30  }
31
```

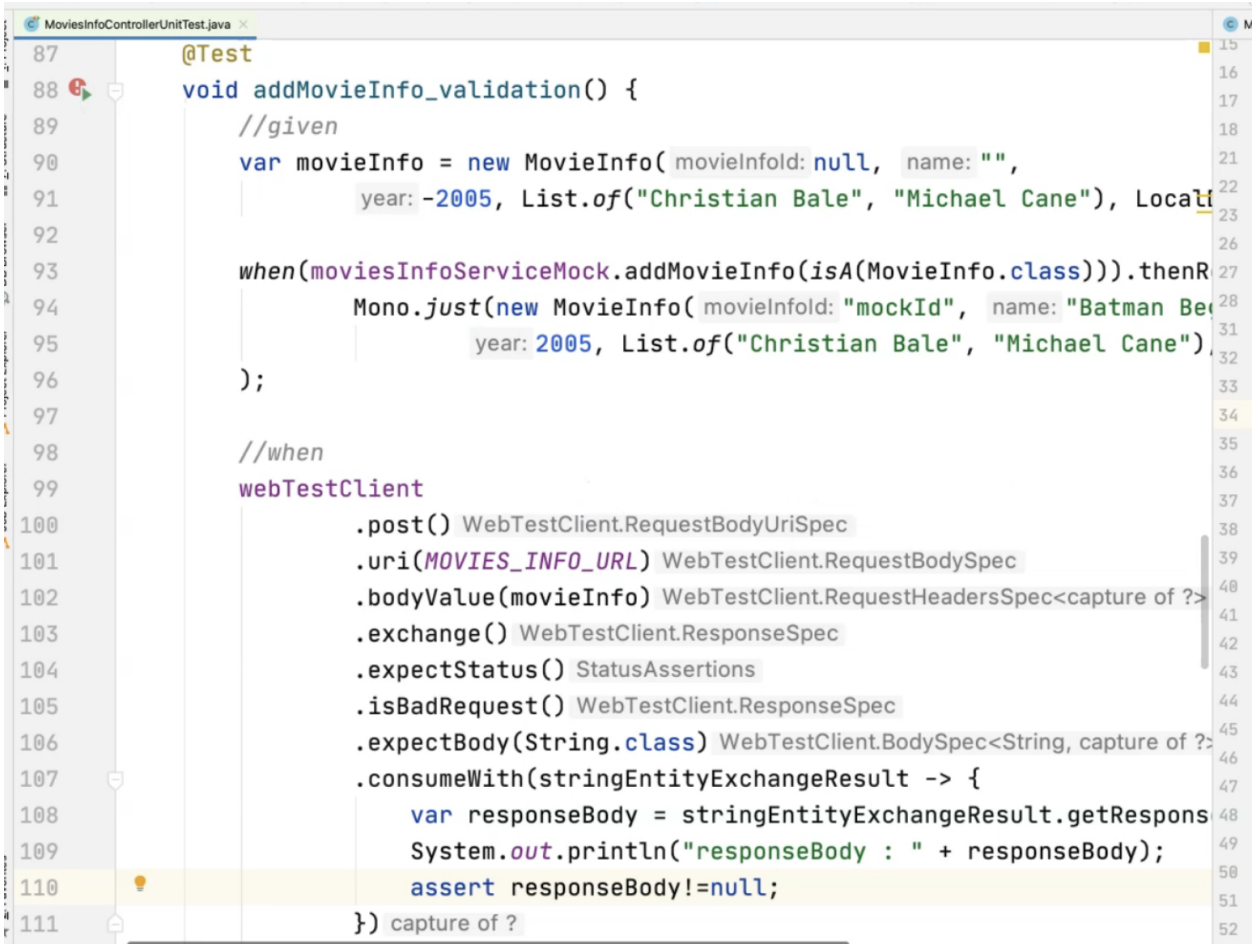
In order to validate these wherever we are passing this object we have to add @Valid annotation.

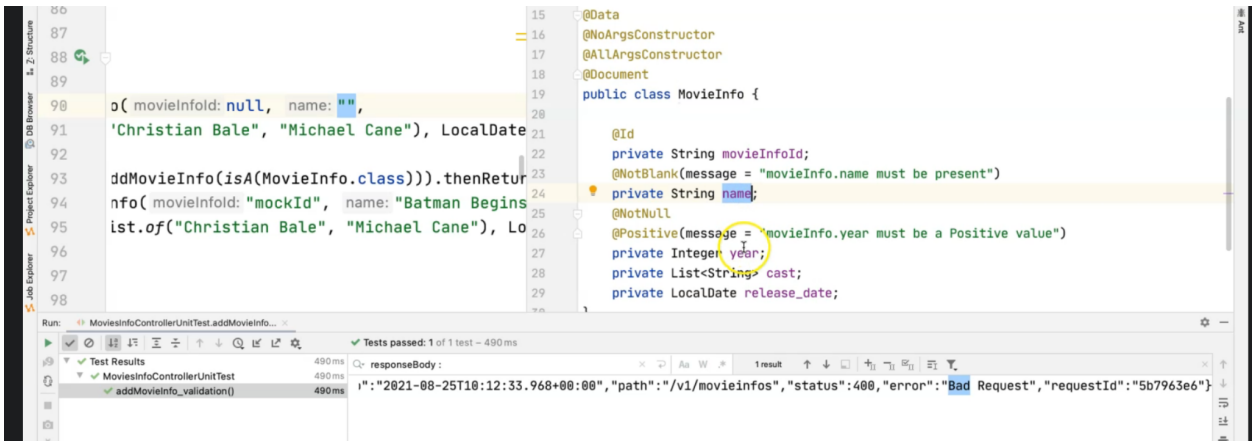
Line number 34.

```
31
32      @PostMapping("/movieinfos")
33      @ResponseStatus(HttpStatus.CREATED)
34      public Mono<MovieInfo> addMovieInfo(@RequestBody @Valid MovieInfo movieInfo){
35          return moviesInfoService.addMovieInfo(movieInfo).log();
36
37      }
```

Without this annotation the validation wont happen. Refer below screenshot







Response body is not telling name is missing and year is negative.

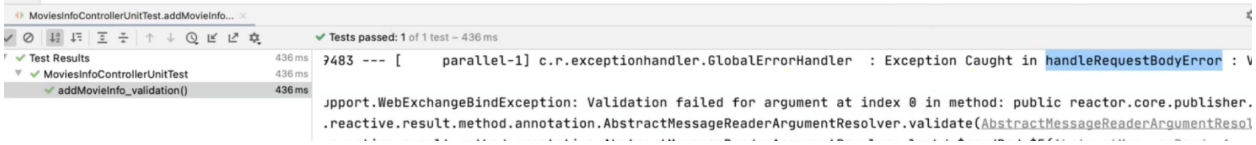
udemy.com/course/build-reactive-restful-apis-using-spring-boot-webflux/learn/lecture/28111458#overview

```java
@ControllerAdvice
@Slf4j
public class GlobalErrorHandler {

    @ExceptionHandler(WebExchangeBindException.class)
    public ResponseEntity<String> handleRequestBodyError(WebExchangeBindException ex) {
        log.error("Exception Caught in handleRequestBodyError : {} ", ex.getMessage(), ex);
        var error = ex.getBindingResult().getAllErrors().stream()
                .map(DefaultMessageSourceResolvable::getDefaultMessage)
                .sorted()
                .collect(Collectors.joining(",", ","));
        log.error("Error is : {} ", error);
        return  ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);

    }
}
```

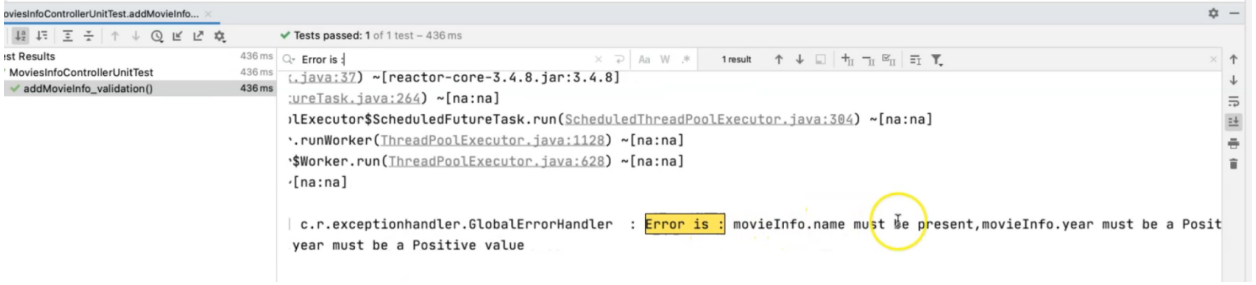
```java
@Test
void addMovieInfo_validation() {
    //given
    var movieInfo = new MovieInfo( movieInfoId: null,  name: "",
            year: -2005, List.of("Christian Bale", "Michael Cane"), LocalDate.parse("2005-06-15"));

    //when
    webTestClient
            .post() WebTestClient.RequestBodyUriSpec
            .uri(MOVIES_INFO_URL) WebTestClient.RequestBodySpec
            .bodyValue(movieInfo) WebTestClient.RequestHeadersSpec<capture of ?>
            .exchange() WebTestClient.ResponseSpec
            .expectStatus() StatusAssertions
            .isBadRequest() WebTestClient.ResponseSpec
            .expectBody(String.class) WebTestClient.BodySpec<String, capture of ?>
            .consumeWith(stringEntityExchangeResult -> {
                var responseBody = stringEntityExchangeResult.getResponseBody();
                System.out.println("responseBody : " + responseBody);
                assert responseBody!=null;
            }) capture of ?
            /*.expectBody(MovieInfo.class)
            .consumeWith(movieInfoEntityExchangeResult -> {
```

Notice the log messages

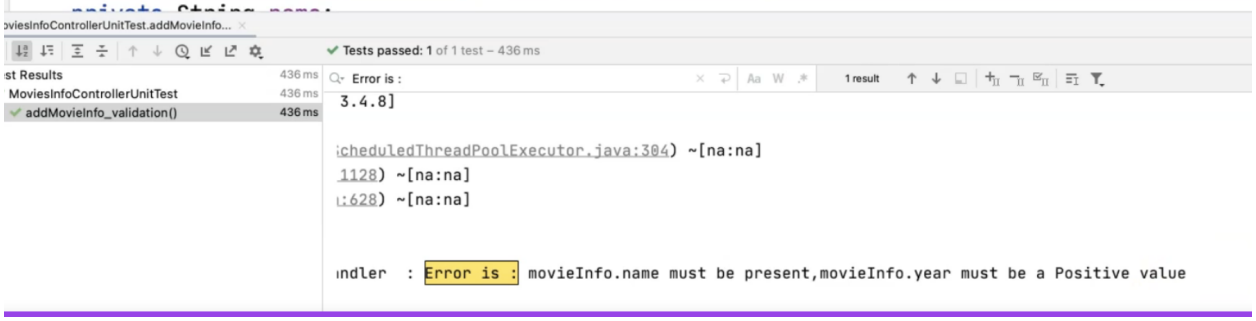
```
@ExceptionHandler(WebExchangeBindException.class)
public ResponseEntity<String> handleRequestBodyError(WebExchangeBindException ex) {
    log.error("Exception Caught in handleRequestBodyError : {} ", ex.getMessage(), ex);
    var error = ex.getBindingResult().getAllErrors().stream() Stream<ObjectError>
            .map(DefaultMessageSourceResolvable::getDefaultMessage) Stream<String>
            .sorted()
            .collect(Collectors.joining( delimiter: ","));
    log.error("Error is : {} ", error);
    return  ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);

}
```




```
        .collect(Collectors.joining( delimiter: ","));
    log.error("Error is : {} ", error);
    return  ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);

}
```




```
@Id
private String movieInfoId;
@NotBlank(message = "movieInfo.name must be present")
```



Now correct the response body.

```
        .expectStatus() StatusAssertions
        .isBadRequest() WebTestClient.ResponseSpec
        .expectBody(String.class) WebTestClient.BodySpec<String, capture of ?>
        .consumeWith(stringEntityExchangeResult -> {
            var responseBody = stringEntityExchangeResult.getResponseBody();
            System.out.println("responseBody : " + responseBody);
            var expectedErrorMessage = "movieInfo.name must be present,movieInfo.year must be a Positive value";
            assert responseBody!=null;
            assertEquals(expectedErrorMessage, responseBody);
        }) capture of ?
        /*.expectBody(MovieInfo.class)
        .consumeWith(movieInfoEntityExchangeResult -> {
```

Validation on list object.

```
    private List<@NotBlank(message = "movieInfo.cast must be present") String> cast;
    private LocalDate release_date;
```

.

```
    .expectBody(String.class) WebTestClient.BodySpec<String, capture of ?>
    .consumeWith(stringEntityExchangeResult -> {
        var responseBody = stringEntityExchangeResult.getResponseBody();
        System.out.println("responseBody : " + responseBody);
        var expectedErrorMessage = "movieInfo.cast must be present,movieInfo.name must be present,movieInfo.year must be a Positive value";
        assert responseBody!=null;
        assertEquals(expectedErrorMessage, responseBody);
    }) capture of ?
    ;


teMovieInfo() {
en
ovieInfoId = "abc";
```

```
MoviesInfoControllerUnitTest.addMovieInfo...
Tests failed: 1 of 1 test – 464 ms
Test Results 464 ms
MoviesInfoControllerUnitTest 464 ms
addMovieInfo_validation() 464 ms

org.opentest4j.AssertionFailedError:
Expected :movieInfo.name must be present,movieInfo.year must be a Positive value
Actual   :movieInfo.cast must be present,movieInfo.name must be present,movieInfo.year must be a Positive value
<Click to see difference>
```

ResponseEntity in webflux

Response status either we are mentioning (line number 33) or we are not mentioning at all. If not species it will take 200.

```java
@GetMapping("/movieinfos")
public Flux<MovieInfo> getAllMovieInfos() { return moviesInfoService.getAllMov:

@GetMapping("/movieinfos/{id}")
public Mono<MovieInfo> getMovieInfoById(@PathVariable String id) { return movi

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

}
```

```java
import java.util.stream.Collectors;

@ControllerAdvice
@Slf4j
public class GlobalErrorHandler {

    @ExceptionHandler(WebExchangeBindException.class)
    public ResponseEntity<String> handleRequestBodyError(WebExchangeBindException ex) {
        log.error("Exception Caught in handleRequestBodyError : {} ", ex.getMessage(), ex);
        var error = ex.getBindingResult().getAllErrors().stream() Stream<ObjectError>
                .map(DefaultMessageSourceResolvable::getDefaultMessage) Stream<String>
                .sorted()
                .collect(Collectors.joining( delimiter: ","));
        log.error("Error is : {} ", error);
        return  ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);

    }
```

In controller advice - we are defining inside the methods.

What about flux ? how to return ResponseEntity in case of flux.

https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html#webflux-ann-responseentity

> WebFlux supports using a single value reactive type to produce the `ResponseEntity` asynchronously, and/or single and multi-value reactive types for the body. This allows a variety of async responses with `ResponseEntity` as follows:
>
> - `ResponseEntity<Mono<T>>` or `ResponseEntity<Flux<T>>` make the response status and headers known immediately while the body is provided asynchronously at a later point. Use `Mono` if the body consists of 0..1 values or `Flux` if it can produce multiple values.
> - `Mono<ResponseEntity<T>>` provides all three — response status, headers, and body, asynchronously at a later point. This allows the response status and headers to vary depending on the outcome of asynchronous request handling.
> - `Mono<ResponseEntity<Mono<T>>>` or `Mono<ResponseEntity<Flux<T>>>` are yet another possible, albeit less common alternative. They provide the response status and headers asynchronously first and then the response body, also asynchronously, second.

Possible response entity in webflux.

Even though we pass not a valid id still we are seeing 200 response.

```
36
37      @Test
38      void updateMovieInfo_notfound() {
39          //given
40          var movieInfoId = "def";
41          var movieInfo = new MovieInfo( movieInfoId: null,  name: "Dark Knight Rises1",
42                  year: 2005, List.of("Christian Bale", "Michael Cane"), LocalDate.parse("2005-06-15"));
43
44          //when
45          webTestClient
46                  .put() WebTestClient.RequestBodyUriSpec
47                  .uri( uri: MOVIES_INFO_URL+"/{id}", movieInfoId) WebTestClient.RequestBodySpec
48                  .bodyValue(movieInfo) WebTestClient.RequestHeadersSpec<capture of ?>
49                  .exchange() WebTestClient.ResponseSpec
50                  .expectStatus() StatusAssertions
51                  .is2xxSuccessful();
52
53
54          //then
55      }
56  }
```

Add log and see the console.

```
5.566  INFO 3816 --- [       Thread-2] o.s.b.a.mongo.embedded.EmbeddedMongo     : 2021-08-26T04:56:45.565-0500 I COMMAND  [conn3
5.818  INFO 3816 --- [     parallel-1] reactor.Mono.FlatMap.1                   : | onSubscribe([Fuseable] MonoFlatMap.FlatMapMa
5.819  INFO 3816 --- [     parallel-1] reactor.Mono.FlatMap.1                   : | request(unbounded)
5.856  INFO 3816 --- [ntLoopGroup-3-4] reactor.Mono.FlatMap.1                   : | onComplete()
```

```
39
40      @PutMapping("/movieinfos/{id}")
41      public Mono<ResponseEntity<MovieInfo>> updateMovieInfo(@RequestBody MovieInfo updatedMovieInfo, @PathVariable Strin
42          return moviesInfoService.updateMovieInfo(updatedMovieInfo, id)
43                  .map(movieInfo -> {
44                      return ResponseEntity.ok().body(movieInfo);
45                  })
46                  .log();
47
48      }
49
```

Here line number 43 map is to convert Mono to ResponseEntity

```
40      @PutMapping("/movieinfos/{id}")
41      public Mono<ResponseEntity<MovieInfo>> updateMovieInfo(@RequestBody MovieInfo updatedMovieInfo, @PathVariable Strin
42          return moviesInfoService.updateMovieInfo(updatedMovieInfo, id)
43                  .map(movieInfo -> {
44                      return ResponseEntity.ok().body(movieInfo);
45                  })
46                  .log();
47
48      }
```

Replace with method reference and implement ResponseEntity with mono Object.

```
@PutMapping("/movieinfos/{id}")
public Mono<ResponseEntity<MovieInfo>> updateMovieInfo(@RequestBody MovieInfo updatedMovieInfo, @PathVariable Strin
    return moviesInfoService.updateMovieInfo(updatedMovieInfo, id) Mono<MovieInfo>
            .map(ResponseEntity.ok()::body) Mono<ResponseEntity<MovieInfo>>
            .switchIfEmpty(Mono.just(ResponseEntity.notFound().build()))
            .log();

}
```

```
@Test
void updateMovieInfo_notfound() {
    //given
    var movieInfoId = "def";
    var movieInfo = new MovieInfo( movieInfoId: null,
            year: 2005, List.of("Christian Bale", "M

    //when
    webTestClient
            .put() WebTestClient.RequestBodyUriSpec
            .uri( uri: MOVIES_INFO_URL+"/{id}", movieI
            .bodyValue(movieInfo) WebTestClient.Request
            .exchange() WebTestClient.ResponseSpec
            .expectStatus() StatusAssertions
            .isNotFound();


    //then
}
}
```

```
public MoviesInfoController(MoviesInfoService moviesInfoService) { this.moviesI

@GetMapping("/movieinfos")
public Flux<MovieInfo> getAllMovieInfos() { return moviesInfoService.getAllMovi

@GetMapping("/movieinfos/{id}")
public Mono<MovieInfo> getMovieInfoById(@PathVariable String id) { return movie

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

}

@PutMapping("/movieinfos/{id}")
public Mono<ResponseEntity<MovieInfo>> updateMovieInfo(@RequestBody MovieInfo u
    return moviesInfoService.updateMovieInfo(updatedMovieInfo, id) Mono<MovieInfo>
            .map(ResponseEntity.ok()::body) Mono<ResponseEntity<MovieInfo>>
            .switchIfEmpty(Mono.just(ResponseEntity.notFound().build()))
            .log();

}

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

}
```

Assignment: Implement 404 Response for the getMovieInfoById Endpoint

30 minutes to complete | 159 student solutions

Implement 404 Response for the getMovieInfoById Endpoint.

reactive-spring-webflux › movies-info-service › src › main › java › com › reactivespring › repository › MovieInfoRepository

MovieInfoRepository.java | MoviesInfoService.java

```
package com.reactivespring.repository;

import com.reactivespring.domain.MovieInfo;
import org.springframework.data.mongodb.repository.ReactiveMongoRepository;

public interface MovieInfoRepository extends ReactiveMongoRepository<MovieInfo,String> {
}
```

Custom Query in Reactive Mongo Repository.

```
@Test
void findByYear() {
    //given

    //when
    var moviesInfoFlux = movieInfoRepository.findByYear(
```

```
//when
var moviesInfoFlux = movieInfoRepository.findByYear(

//then
StepVerifier.create(moviesInfoFlux)
        .expectNextCount(1)
        .verifyComplete();
}
}
```

```
package com.reactivespring.repository;

import com.reactivespring.domain.MovieInfo;
import org.springframework.data.mongodb.repository.ReactiveMongoRep
import reactor.core.publisher.Flux;

public interface MovieInfoRepository extends ReactiveMongoRepositor

    Flux<MovieInfo> findByYear(Integer year);

}
```

```
ver.connection          : Opened connection [connectionId{localValue:5, serverValue:5}] to localhost:59133
mbedded.EmbeddedMongo   : 2021-08-28T05:10:48.328-0500 I COMMAND  [conn3] command test.movieInfo command: insert { insert: "movi
mbedded.EmbeddedMongo   : 2021-08-28T05:10:48.328-0500 I WRITE    [conn5] update test.movieInfo query: { _id: "abc" } planSummar
mbedded.EmbeddedMongo   : 2021-08-28T05:10:48.328-0500 I COMMAND  [conn5] command test.$cmd command: update { update: "movieInfo
mbedded.EmbeddedMongo   : 2021-08-28T05:10:48.329-0500 I COMMAND  [conn4] command test.movieInfo command: insert { insert: "movi
ingWhen.1               : onSubscribe(FluxUsingWhen.UsingWhenSubscriber)
ingWhen.1               : request(unbounded)
ingWhen.1               : onNext(MovieInfo(movieInfoId=612a0ba82c42a673d3e97a81, name=Batman Begins, year=2005, cast=[Christian
ingWhen.1               : onComplete()
```

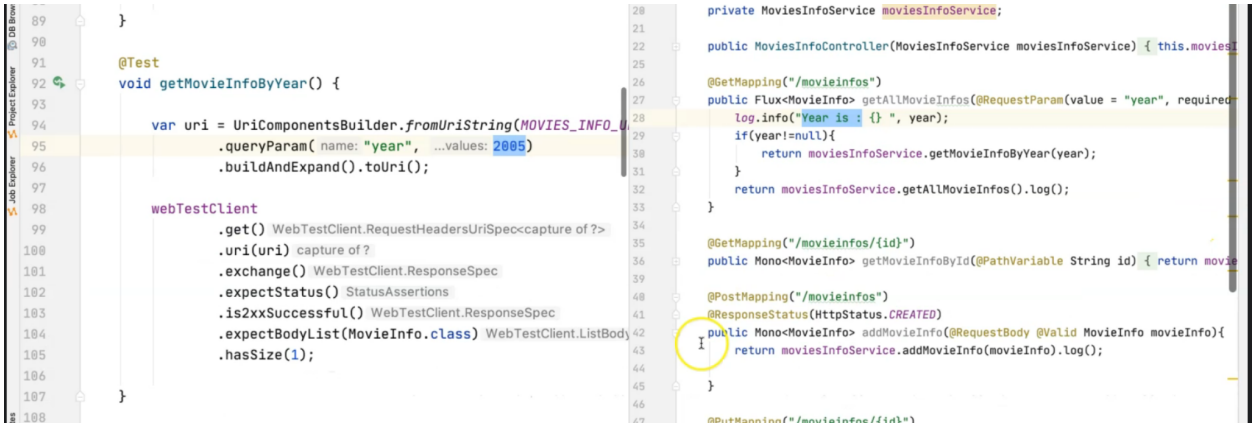
```
@GetMapping("/movieinfos")
public Flux<MovieInfo> getAllMovieInfos() { return moviesInfoService.getAllMovieInfos().log(); }

@GetMapping("/movieinfos/{id}")
```

Use the above method in the controller under get movies call

```
@GetMapping("/movieinfos")
public Flux<MovieInfo> getAllMovieInfos(@RequestParam(value = "year", required = false) Integer year){

    if(year!=null){
        return moviesInfoService.getMovieInfoByYear(year);
    }
    return moviesInfoService.getAllMovieInfos().log();
}
```

Service layer

```
    public Flux<MovieInfo> getMovieInfoByYear(Integer year) {

        return movieInfoRepository.findByYear(year);
    }
}
```



```
}

@Test
void getMovieInfoByYear() {

    var uri = UriComponentsBuilder.fromUriString(MOVIES_INFO_U
            .queryParam( name: "year",  ...values: 2005)
            .buildAndExpand().toUri();

    webTestClient
            .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
            .uri(uri) capture of ?
            .exchange() WebTestClient.ResponseSpec
            .expectStatus() StatusAssertions
            .is2xxSuccessful() WebTestClient.ResponseSpec
            .expectBodyList(MovieInfo.class) WebTestClient.ListBody
            .hasSize(1);

}
```

```
private MoviesInfoService moviesInfoService;

public MoviesInfoController(MoviesInfoService moviesInfoService) { this.moviesI

@GetMapping("/movieinfos")
public Flux<MovieInfo> getAllMovieInfos(@RequestParam(value = "year", required
    log.info("Year is : {} ", year);
    if(year!=null){
        return moviesInfoService.getMovieInfoByYear(year);
    }
    return moviesInfoService.getAllMovieInfos().log();
}

@GetMapping("/movieinfos/{id}")
public Mono<MovieInfo> getMovieInfoById(@PathVariable String id) { return movie

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

}
```

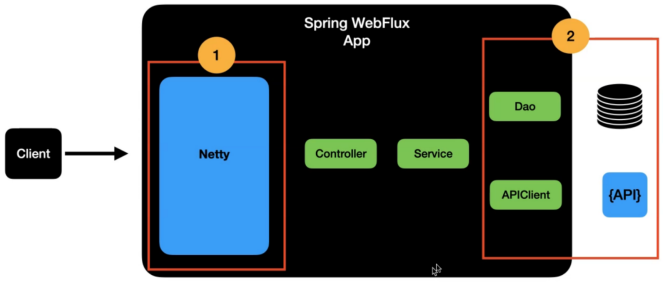
Assignment: Build a Custom Repository function to retrieve MovieInfo by name

10 minutes to complete | 145 student solutions

Build a Custom Repository function in MovieInfoRepository to retrieve MovieInfo by name

How netty works with spring webflux.

NonBlocking or Reactive API using Spring WebFlux



Here we have two Non-blocking points. Netty server and dao accessing db and api client accessing api.

```
database: local
---
logging:
  level:
    root: debug
```

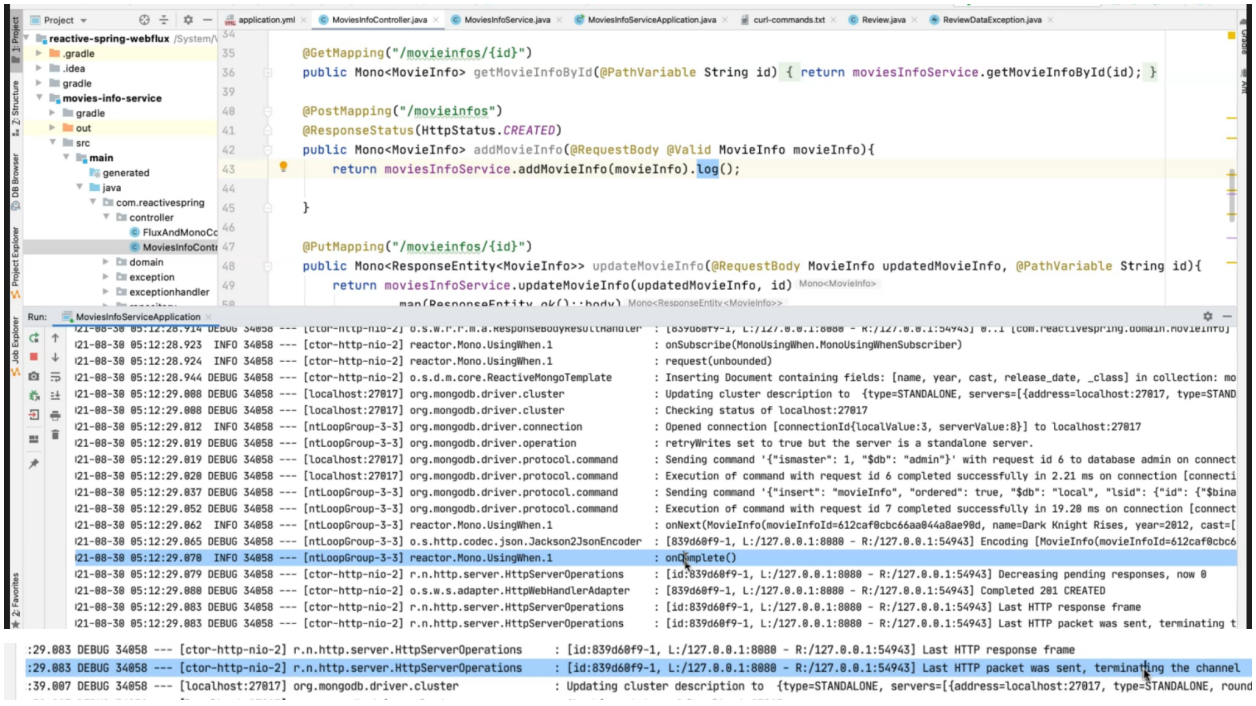
Here http is netty thread.

```
5.263 DEBUG 8287 --- [localhost:27017] org.mongodb.driver.protocol.command      : Execution of command with request id 5 completed successfully in 1.72 ms o
1.756 DEBUG 8287 --- [ctor-http-nio-2] r.n.http.server.HttpServerOperations     : [id:56fee70b, L:/127.0.0.1:8080 - R:/127.0.0.1:61537] New http connection,
1.757 DEBUG 8287 --- [ctor-http-nio-2] reactor.netty.transport.TransportConfig  : [id:56fee70b, L:/127.0.0.1:8080 - R:/127.0.0.1:61537] Initialized pipeline
1.779 DEBUG 8287 --- [ctor-http-nio-2] r.n.http.server.HttpServerOperations     : [id:56fee70b, L:/127.0.0.1:8080 - R:/127.0.0.1:61537] Increasing pending r
1.785 DEBUG 8287 --- [ctor-http-nio-2] reactor.netty.http.server.HttpServer     : [id:56fee70b-1, L:/127.0.0.1:8080 - R:/127.0.0.1:61537] Handler is being a
1.798 DEBUG 8287 --- [ctor-http-nio-2] o.s.w.s.adapter.HttpWebHandlerAdapter    : [56fee70b-1, L:/127.0.0.1:8080 - R:/127.0.0.1:61537] HTTP POST "/v1/movieI
1.814 DEBUG 8287 --- [ctor-http-nio-2] s.w.r.r.m.a.RequestMappingHandlerMapping : [56fee70b-1, L:/127.0.0.1:8080 - R:/127.0.0.1:61537] Mapped to com.reactiv
1.829 DEBUG 8287 --- [ctor-http-nio-2] .r.m.a.RequestBodyMethodArgumentResolver : [56fee70b-1, L:/127.0.0.1:8080 - R:/127.0.0.1:61537] Content-Type:applicat
1.895 DEBUG 8287 --- [ctor-http-nio-2] .r.m.a.RequestBodyMethodArgumentResolver : [56fee70b-1, L:/127.0.0.1:8080 - R:/127.0.0.1:61537] 0..1 [com.reactivespr
1.908 DEBUG 8287 --- [ctor-http-nio-2] reactor.netty.channel.FluxReceive        : [id:56fee70b-1, L:/127.0.0.1:8080 - R:/127.0.0.1:61537] FluxReceive{pendin
```

Curl

```
curl -i \
-d '{"movieInfoId":null, "name": "Dark Knight Rises", "year":2012,"cast":["Christian Bale", "Tom Hardy"],"release_date": "2012-07-
-H "Content-Type: application/json" \
-X POST http://localhost:8080/v1/movieinfos
```

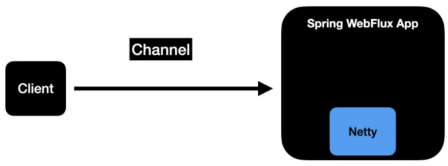
Whenever save is executed, a new thread is coming and handling till on complete.




```
:29.083 DEBUG 34058 --- [ctor-http-nio-2] r.n.http.server.HttpServerOperations     : [id:839d60f9-1, L:/127.0.0.1:8080 - R:/127.0.0.1:54943] Last HTTP response frame
:29.083 DEBUG 34058 --- [ctor-http-nio-2] r.n.http.server.HttpServerOperations     : [id:839d60f9-1, L:/127.0.0.1:8080 - R:/127.0.0.1:54943] Last HTTP packet was sent, terminating the channel
:39.007 DEBUG 34058 --- [localhost:27017] org.mongodb.driver.cluster               : Updating cluster description to  {type=STANDALONE, servers=[{address=localhost:27017, type=STANDALONE, round
```

Again the netty thread is coming to the picture to display data to the client and terminating the channel at the end. In the netty world channel is the representation of connection between client and server. Switching between threads are handled by the reactor.
The netty thread is not blocked / not waiting for db response.

Netty (WebFlux's Default Server)



- Channel represents an open connection between the client and server
- Request and Response is sent via the channel

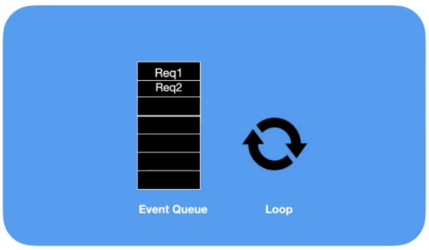
Channel

- Channel has ChannelHandlers
  - Accepting the client connection
  - Reading the data as bytes from the network to a Java Object(Transformation)
  - Writing the data back to the client
- This is all taken care for us by Spring WebFlux
- As a developer, we just focus on writing the application related code

Channel and EventLoop

- Netty , uses **EventLoop model** to handle the connections in a nonblocking fashion
- An EventLoop is powered by one single thread
  - NodeJs uses the same pattern.
  - Node js has just one thread/one eventloop to handle client requests
- Number of Eventloops to handle the request is equal to no of cores in your machine
  - **Eventloops** are part of the EventLoopGroup

EventLoop



event loop execute request available in event queue one after another.

How Channel and EventLoop linked ?

- Any time a channel is created it gets assigned to an EventLoop
- This **EventLoop** is responsible for handling the different events that occurs in the lifetime of a channel

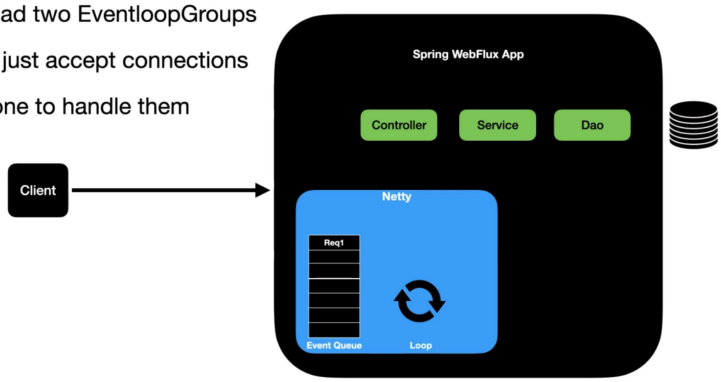
Channel Lifecycle

1. ChannelUnregistered — Channel is Created and its not registered with the Eventloop
2. ChannelRegistered — Channel is registered with the Eventloop
3. ChannelActive — Channel is active and its now possible to send and receive the data
4. ChannelInActive — Channel is not connected to the client anymore and ready to be closed

All these Lifecycle changes are treated as events

How Netty handles the request ?

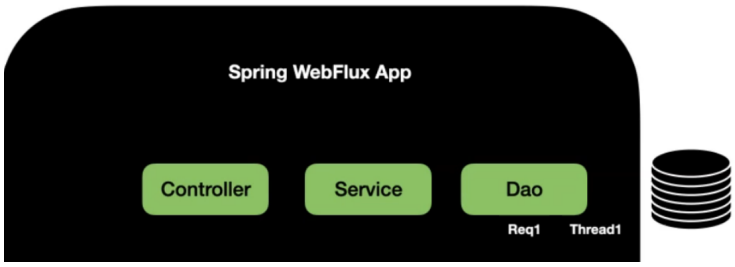
- Netty had two EventloopGroups
- One to just accept connections
- Other one to handle them



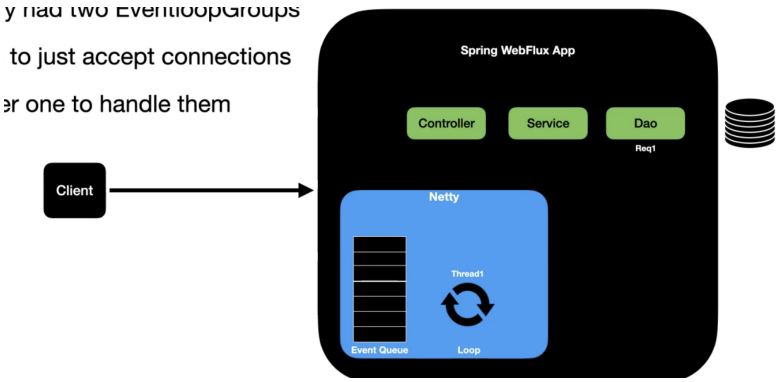
client 1 sent the request

Event loop take the request and pass it to controller , service and dao. Here dao is reactive in nature

-

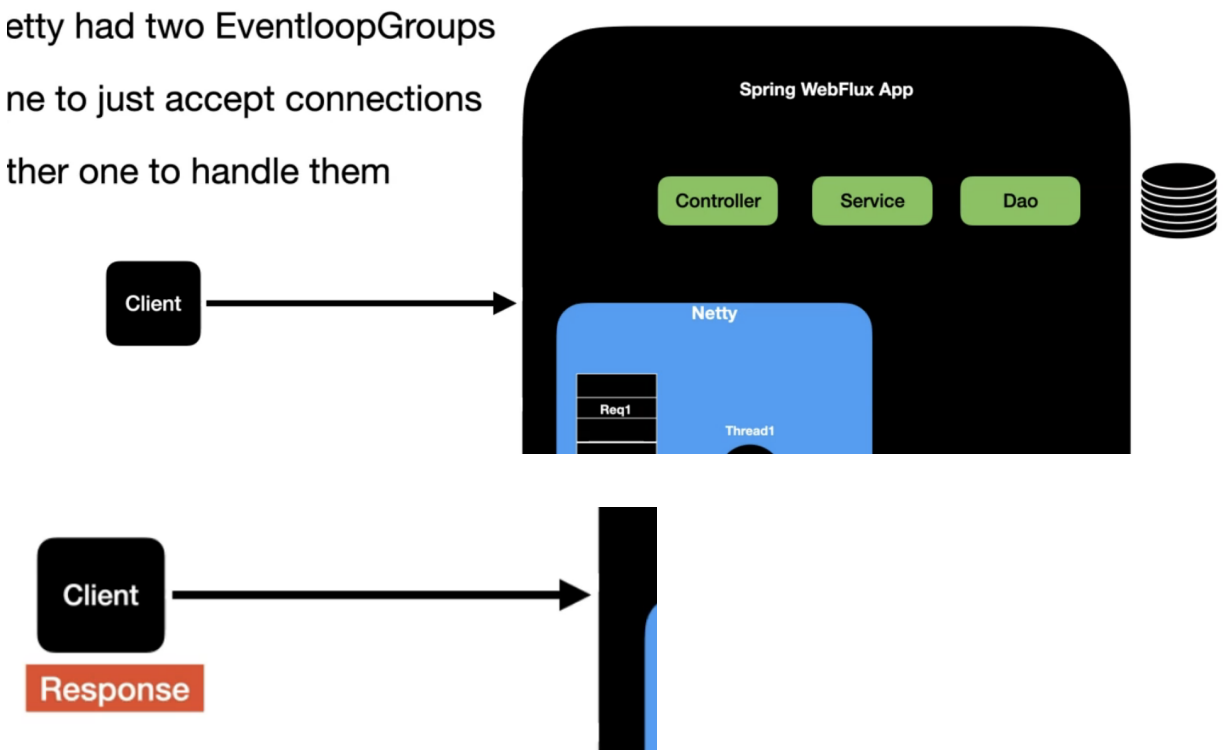


Thread1 will come back to the server once hit the DB.

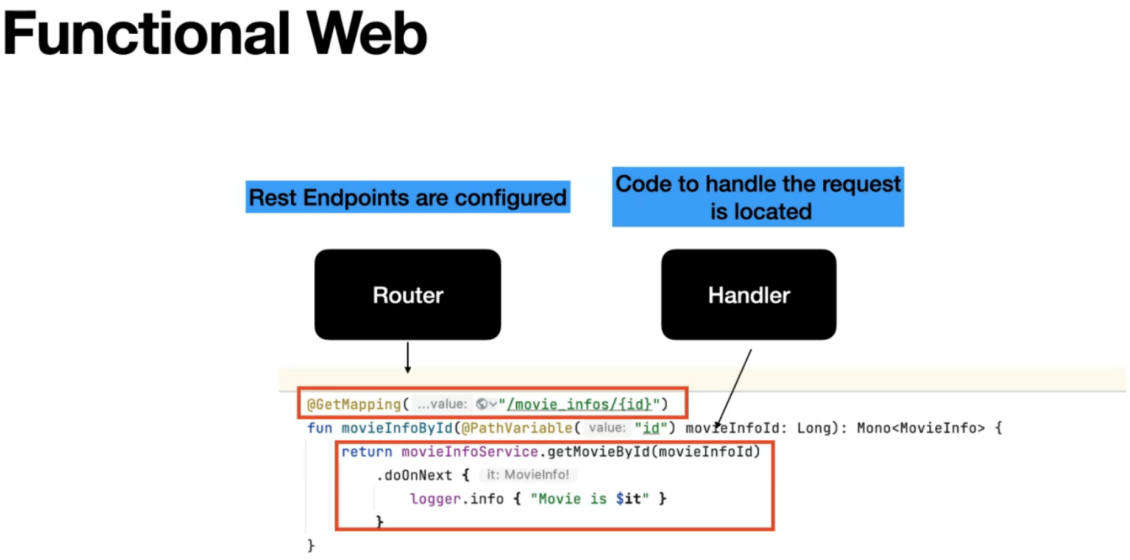


Not thread comeback to event loop in the server.

DAO layer publishes data to the event queue as part of netty and its status is write. This status will help is it send back to the client or controller again.



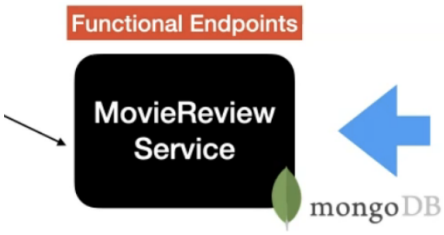
Functional web in webflux.



Is there an advantage in building RestFul APIs using Functional Web?

Functional Web

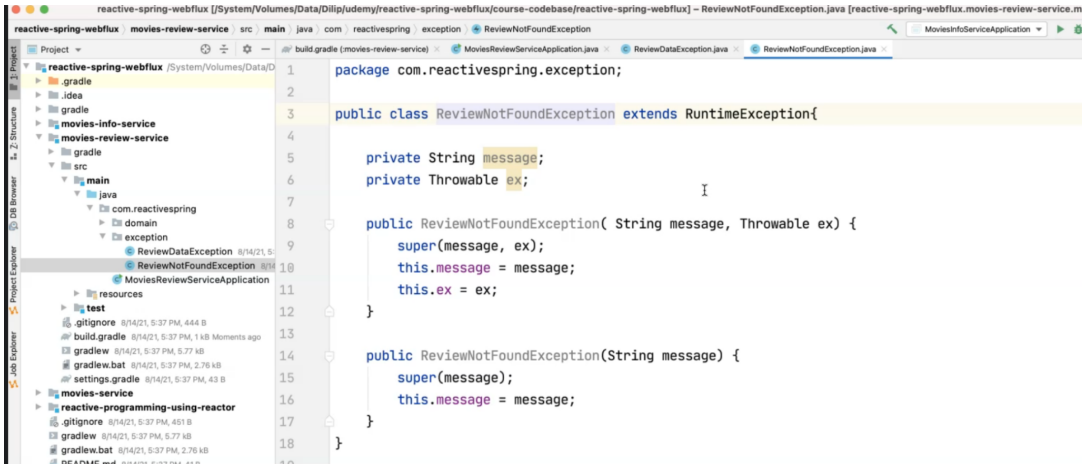
- Benefits:
  - All the RestFul APIs endpoints are configured in one single file
  - Code is lightweight compared to the Controller alternative
- Challenges:
  - Need to have knowledge about functional programming
  - Bean Validation is different in Functional Web
  - Exception handling in Functional Web is different from the Controller approach

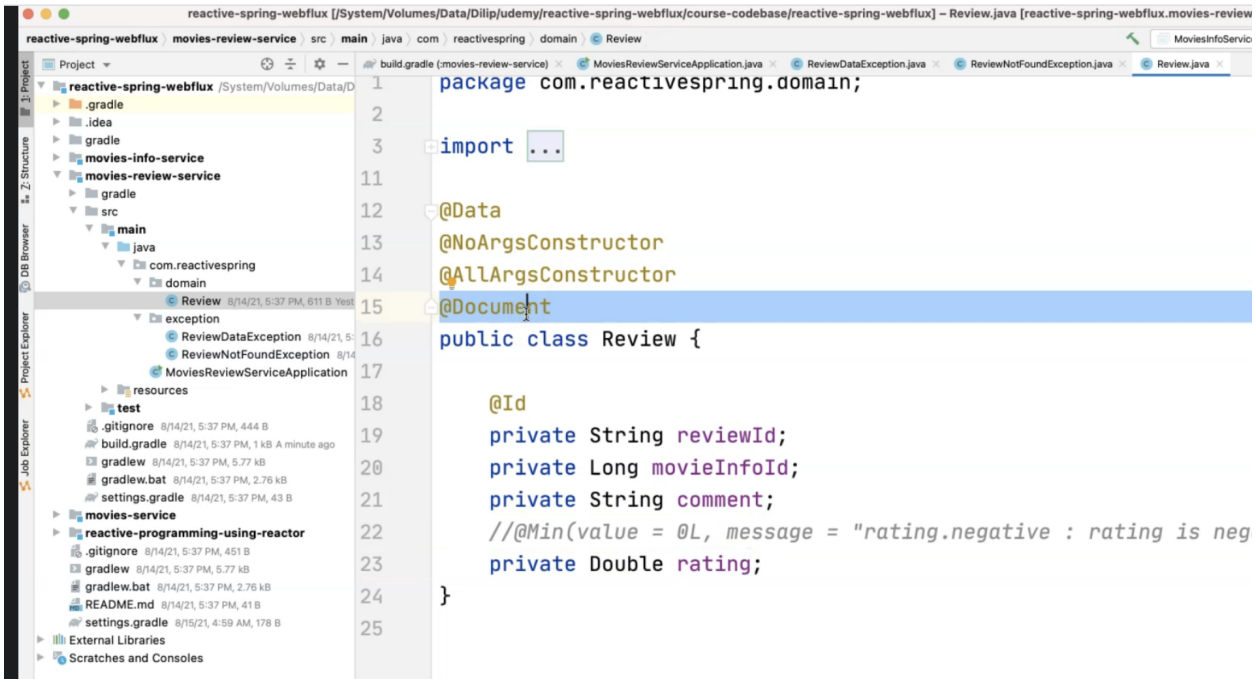



```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-mongodb-reactive'
    implementation 'org.springframework.boot:spring-boot-starter-webflux'

    //validator
    implementation 'org.springframework.boot:spring-boot-starter-validation'

    //lombok
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'

    //test
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'de.flapdoodle.embed:de.flapdoodle.embed.mongo'
    testImplementation 'io.projectreactor:reactor-test'
}
```

Same dependencies like the previous project.


reactive-spring-webflux /System/Volumes/Data/D
.gradle
.idea
gradle
movies-info-service
movies-review-service
gradle
src
main
java
com.reactivespring
domain
exception
ReviewDataException 8/14/21, 5:
ReviewNotFoundException 8/14
MoviesReviewServiceApplication
resources
test
.gitignore 8/14/21, 5:37 PM, 444 B
build.gradle 8/14/21, 5:37 PM, 1 kB Moments ago
gradlew 8/14/21, 5:37 PM, 5.77 kB
gradlew.bat 8/14/21, 5:37 PM, 2.76 kB
settings.gradle 8/14/21, 5:37 PM, 43 B
movies-service
reactive-programming-using-reactor
.gitignore 8/14/21, 5:37 PM, 451 B
build.gradle (:movies-review-service)
MoviesReviewServiceApplication.java
ReviewDataException.java
1 package com.reactivespring.exception;
2
3 public class ReviewDataException extends RuntimeException {
4 private String message;
5 public ReviewDataException(String s) {
6 super(s);
7 this.message=s;
8 }
9 }
10


reactive-spring-webflux [/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux] – ReviewNotFoundException.java [reactive-spring-webflux.movies-review-service.m
reactive-spring-webflux movies-review-service src main java com reactivespring exception ReviewNotFoundException
MoviesInfoServiceApplication
build.gradle (:movies-review-service)
MoviesReviewServiceApplication.java
ReviewDataException.java
ReviewNotFoundException.java
ReviewNotFoundException 8/14
1 package com.reactivespring.exception;
2
3 public class ReviewNotFoundException extends RuntimeException{
4
5 private String message;
6 private Throwable ex;
7
8 public ReviewNotFoundException(String message, Throwable ex) {
9 super(message, ex);
10 this.message = message;
11 this.ex = ex;
12 }
13
14 public ReviewNotFoundException(String message) {
15 super(message);
16 this.message = message;
17 }
18 }


reactive-spring-webflux [/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux] – Review.java [reactive-spring-webflux.movies-review
reactive-spring-webflux movies-review-service src main java com reactivespring domain Review
MoviesInfoServic
build.gradle (:movies-review-service)
MoviesReviewServiceApplication.java
ReviewDataException.java
ReviewNotFoundException.java
Review.java
Review 8/14/21, 5:37 PM, 611 B Yest
1 package com.reactivespring.domain;
2
3 import ...
11
12 @Data
13 @NoArgsConstructor
14 @AllArgsConstructor
15 @Document
16 public class Review {
17
18 @Id
19 private String reviewId;
20 private Long movieInfoId;
21 private String comment;
22 //@Min(value = 0L, message = "rating.negative : rating is neg
23 private Double rating;
24 }
25
build.gradle 8/14/21, 5:37 PM, 1 kB A minute ago
README.md 8/14/21, 5:37 PM, 41 B
settings.gradle 8/15/21, 4:59 AM, 178 B
External Libraries
Scratches and Consoles

Create router and handler packages.
Router is configuration class which contain a bean which handle all http requests
@Configuration and @bean

Get method will take the string and handler function . handler function will take server request as input and return server responses.

```java
@Configuration
public class ReviewRouter {

    @Bean
    public RouterFunction<ServerResponse> reviewsRoute(){

        return route()
                .GET( pattern: "/v1/helloworld",(request -> ServerResponse.ok().bodyValue("helloworld")))
                .build();

    }
}
```

```
/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl http://localhost:8081/v1/helloworld
helloworld%
```