

This document contains continuation of Functional web in webflux for MovieReview Application, unit testing for functional web, bean validation in functional web, custom global exception handler in functional web. Handling resources not found in functional web , Movie Service which talks with MovieInfo and MovieReview Service and Exception service to service communication.

Post method with functional style.



```
1 @Bean
2 public RouterFunction<ServerResponse> reviewsRoute(ReviewHandler reviewHandler){
3
4     return route()
5         .GET(pattern: "/v1/helloworld", (request -> ServerResponse.ok().bodyValue("helloworld")))
6         .POST(pattern: "/v1/reviews", request -> reviewHandler.addReview(request))
7         .build();
8 }
```

Two changes here are passing the handler object as parameter in the bean method.

Handler class with component annotation



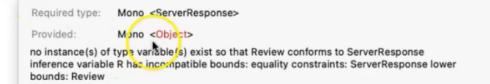
```
1 @Component
2 public class ReviewHandler {
3
4     private ReviewReactiveRepository reviewReactiveRepository;
5
6     public ReviewHandler(ReviewReactiveRepository reviewReactiveRepository) {
7         this.reviewReactiveRepository = reviewReactiveRepository;
8     }
9
10    public Mono<ServerResponse> addReview(ServerRequest request) {
11
12        request.bodyToMono(Review.class)
13            .flatMap(review -> {
14                return reviewReactiveRepository.save(review);
15            });
16    }
17 }
```

We can change it to method reference

```
21     request.bodyToMono(Review.class)
22         .flatMap(reviewReactiveRepository::save);
23 }
```

Here it is writing Mono of object instead of mono of server response.

```
17 }
18
19 @
20
21     return request.bodyToMono(Review.class)
22         .flatMap(reviewReactiveRepository::save);
23 }
24 }
25 }
```



```
0 @
1
2     public Mono<ServerResponse> addReview(ServerRequest request) {
3
4         return request.bodyToMono(Review.class)
5             .flatMap(reviewReactiveRepository::save)
6             .flatMap(savedReview -> {
7                 ServerResponse.status(HttpStatus.CREATED)
8                     .bodyValue(savedReview);
9             });
10 }
```

After replacing with method reference

```
@
1
2     public Mono<ServerResponse> addReview(ServerRequest request) {
3
4         return request.bodyToMono(Review.class)
5             .flatMap(reviewReactiveRepository::save)
6             .flatMap(ServerResponse.status(HttpStatus.CREATED)::bodyValue);
7
8 }
```

Integration test:

```

15
16     @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
17     @ActiveProfiles("test")
18     @AutoConfigureWebTestClient
19  >  public class ReviewsIntgTest {
20
21     @Autowired
22     WebTestClient webTestClient;
23
24     @Autowired
25     ReviewReactiveRepository reviewReactiveRepository;
26
27     static String REVIEWS_URL = "/v1/reviews";
28
29     @BeforeEach
30     void setUp() {
31
32         var reviewsList = List.of(
33             new Review(reviewId null, movieInfoId 1L, comment:"Awesome Movie", rating:9.0),
34             new Review(reviewId null, movieInfoId 1L, comment:"Awesome Movie1", rating:9.0),
35             new Review(reviewId null, movieInfoId 2L, comment:"Excellent Movie", rating:8.0));
36         reviewReactiveRepository.saveAll(reviewsList)
37             .blockLast();
38     }
39
40     @AfterEach
41     void tearDown() {
42         reviewReactiveRepository.deleteAll().block();
43     }
44
45
46

```

```

44
45     @Test
46     void addReview() {
47
48         //given
49         var review = new Review(reviewId null, movieInfoId 1L, comment:"Awesome Movie", rating:9.0);
50
51         //when
52
53         webTestClient
54             .post() WebTestClient.RequestBodyUriSpec
55             .uri(REVIEWS_URL) WebTestClient.RequestBodySpec
56             .bodyValue(review) WebTestClient.RequestHeadersSpec<capture of ?>
57             .exchange() WebTestClient.ResponseSpec
58             .expectStatus() StatusAssertions
59             .isCreated() WebTestClient.ResponseSpec
60             .expectBody(Review.class) WebTestClient.BodySpec<Review, capture of ?>
61             .consumeWith(movieInfoEntityExchangeResult -> {
62
63                 var savedReview = movieInfoEntityExchangeResult.getResponseBody();
64                 assert savedReview!=null;
65                 assert savedReview.getReviewId()!=null;
66             });
67

```

Get method with functional style.

A screenshot of a Java code editor showing a portion of a file named `RouterFunctionConfig.java`. The code defines a `@Bean` named `reviewsRoute` which returns a `RouterFunction<ServerResponse>`. The implementation uses the `route()` builder pattern to define two `.GET` methods: one for the path `/v1/helloworld` returning a static string, and another for the path `/v1/reviews` which calls `reviewHandler.addReview(request)` and `reviewHandler.getReviews(request)`. A yellow circle highlights the `reviewHandler.getReviews(request)` line.

```
14
15     @Bean
16     public RouterFunction<ServerResponse> reviewsRoute(ReviewHandler reviewHandler){
17
18         return route()
19             .GET(pattern: "/v1/helloworld", (request -> ServerResponse.ok().bodyValue("helloworld")))
20             .POST(pattern: "/v1/reviews", request -> reviewHandler.addReview(request))
21             .GET(pattern: "/v1/reviews", request -> reviewHandler.getReviews(request))
22             .build();
23     }

```

Line number 20

A screenshot of a Java code editor showing the `getReviews` method implementation. It takes a `ServerRequest` parameter and returns a `Mono<ServerResponse>`. Inside, it retrieves all reviews from the `reviewReactiveRepository.findAll()` and returns them as a `ServerResponse.ok().body(reviewsFlux, Review.class)`. A yellow circle highlights the `Review.class` type argument.

```
28     public Mono<ServerResponse> getReviews(ServerRequest request) {
29
30         var reviewsFlux = reviewReactiveRepository.findAll();
31         return ServerResponse.ok().body(reviewsFlux, Review.class);
32     }
33 }
```

Integration testing for the above method is assignment

Nest function used for grouping the same endpoint with different methods as shown below.

A screenshot of a Java code editor showing the `reviewsRoute` method using the `nest` function. It groups the `/v1/reviews` endpoint into a builder, defining both `.POST` and `.GET` methods. A yellow circle highlights the `ReviewHandler.getReviews(request)` line.

```
15
16     public RouterFunction<ServerResponse> reviewsRoute(ReviewHandler reviewHandler){
17
18         return route()
19             .nest(path(pattern: "/v1/reviews"), builder -> {
20                 builder.POST(pattern: "", request -> reviewHandler.addReview(request))
21                 .GET(pattern: "", request -> reviewHandler.getReviews(request));
22             })
23             .GET(pattern: "/v1/helloworld", (request -> ServerResponse.ok().bodyValue("helloworld")));
24     }
25 }
```

Run the integration test and check all are passing if not passing something wrong with our changes. That is usefulness and fastest way of doing testing.

Put method with functional style:

A screenshot of a Java code editor showing the `PUT` method implementation. It adds a new `.PUT` method to the `nest` block for the `/v1/reviews` endpoint, which calls `reviewHandler.updateReview(request)`. A yellow circle highlights the `reviewHandler.updateReview(request)` line.

```
14
15     @Bean
16     public RouterFunction<ServerResponse> reviewsRoute(ReviewHandler reviewHandler){
17
18         return route()
19             .nest(path(pattern: "/v1/reviews"), builder -> {
20                 builder.POST(pattern: "", request -> reviewHandler.addReview(request))
21                 .GET(pattern: "", request -> reviewHandler.getReviews(request))
22                 .PUT(pattern:("/{id}"), request -> reviewHandler.updateReview(request));
23             })
24             .GET(pattern: "/v1/helloworld", (request -> ServerResponse.ok().bodyValue("helloworld")));
25     }

```

Line number 22

```

35 @ ->     public Mono<ServerResponse> updateReview(ServerRequest request) {
36
37     var reviewId = request.pathVariable("id");
38
39     var existingReview = reviewReactiveRepository.findById(reviewId);
40
41     return existingReview
42         .flatMap(review -> request.bodyToMono(Review.class)
43             .map(reqReview -> {
44                 review.setComment(reqReview.getComment());
45                 review.setRating(reqReview.getRating());
46                 return review;
47             })
48             .flatMap(reviewReactiveRepository::save)
49             .flatMap(savedReview -> ServerResponse.ok().bodyValue(savedReview))
50         );
51     }
52 }
53

```

```

Job Explic
-----[redacted]
/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i \
ps-MacBook-Pro
\ -d '{"reviewId":1, "movieInfoId":1, "comment": "Excellent Movie", "rating":8.0}' \
\ -H "Content-Type: application/json" \
\ -X POST http://localhost:8081/v1/reviews
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: 73

{"reviewId":1,"movieInfoId":1,"comment":"Excellent Movie","rating":8.0}%
-----[redacted]

```

```

-----[redacted]
/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i \
\ -d '{"reviewId":1, "movieInfoId":1, "comment": "Excellent Movie Update", "rating":8.5}' \
\ -H "Content-Type: application/json" \
\ -X PUT http://localhost:8081/v1/reviews/1
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 80

{"reviewId":1,"movieInfoId":1,"comment":"Excellent Movie Update","rating":8.5}%
-----[redacted]

```

Update reflected everywhere.

```

-----[redacted]
/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i http://localhost:8081/v1/reviews
HTTP/1.1 200 OK
transfer-encoding: chunked
Content-Type: application/json

[{"reviewId":612f53cd80720e6ae6bb2809, "movieInfoId":1, "comment": "Awesome Movie", "rating":9.0}, {"reviewId":612f566db85b70267793973f, "movieInfoId":1, "comment": "Awesome Movie", "rating":9.0}, {"reviewId":1, "movieInfoId":1, "comment": "Excellent Movie Update", "rating":8.5}]%
-----[redacted]

```

Delete method with functional style:

A screenshot of a Java code editor showing a controller class. The code defines a `route()` method that nests a route for `/v1/reviews`. It handles four methods: POST, GET, PUT, and DELETE. The DELETE method is annotated with `@DELETE("/id")` and points to a `reviewHandler.deleteReview(request)` method. Line number 23 is highlighted.

```
return route()
    .nest(path(pattern: "/v1/reviews"), builder -> {
        builder.POST(pattern: "", request -> reviewHandler.addReview(request))
            .GET(pattern: "", request -> reviewHandler.getReviews(request))
            .PUT(pattern: "/{id}", request -> reviewHandler.updateReview(request))
            .DELETE(pattern: "/{id}", request -> reviewHandler.deleteReview(request));
    })
    .GET(pattern: "/{id}", request -> reviewHandler.getReviewById(request));
```

Line number 23

A screenshot of a Java code editor showing the implementation of the `deleteReview` method. It takes a `ServerRequest` parameter. It extracts the `reviewId` from the path variables. It finds the existing review using `reviewReactiveRepository.findById(reviewId)`. Then it performs a `flatMap` operation where the review is deleted using `reviewReactiveRepository.deleteById(reviewId)`, and a `then(ServerResponse.noContent().build())` block is executed. Line number 103 is highlighted.

```
public Mono<ServerResponse> deleteReview(ServerRequest request) {
    var reviewId = request.pathVariable(name: "id");
    var existingReview = reviewReactiveRepository.findById(reviewId);
    return existingReview
        .flatMap(review -> reviewReactiveRepository.deleteById(reviewId)
            .then(ServerResponse.noContent().build()));
}
```

A screenshot of a terminal window showing a successful `DELETE` request to the `/v1/reviews/1` endpoint. The command used was `curl -i -X DELETE http://localhost:8081/v1/reviews/1`. The response status was `HTTP/1.1 204 No Content`.

```
{"reviewId": "1", "movieInfoId": 1, "comment": "Excellent Movie", "rating": 8.0}%
/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i -X DELETE http://localhost:8081/v1/reviews/1
HTTP/1.1 204 No Content
```

Based on `movieInfoId` we should retrieve all the reviews.

A screenshot of a Java code editor showing the `Review` entity definition. It uses `@Data`, `@NoArgsConstructor`, and `@AllArgsConstructor` annotations. It is annotated with `@Document` from the `reactive-mongo` library. The entity has fields: `reviewId` (with `@Id`), `movieInfoId`, `comment`, and `rating`. Line number 16 is highlighted.

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Document
public class Review {
```

A screenshot of a Java code editor showing the `ReviewReactiveRepository` interface. It extends `ReactiveMongoRepository<Review, String>`. It defines a method `Flux<Review> findReviewsByMovieInfoId(Long movieInfoId);`. Line number 10 is highlighted.

```
public interface ReviewReactiveRepository extends ReactiveMongoRepository<Review, String> {
    Flux<Review> findReviewsByMovieInfoId(Long movieInfoId);
}
```

A screenshot of a Java code editor showing a controller class. The code is as follows:

```
28
29 @
30
31     public Mono<ServerResponse> getReviews(ServerRequest request) {
32
33         var movieInfoId = request.queryParam("movieInfoId");
34
35         if(movieInfoId.isPresent()){
36             var reviewsFlux = reviewReactiveRepository.findReviewsByMovieInfoId(Long.valueOf(movieInfoId.get()));
37             return buildReviewsResponse(reviewsFlux);
38         }else {
39             var reviewsFlux = reviewReactiveRepository.findAll();
40             return buildReviewsResponse(reviewsFlux);
41         }
42     }
43
44 @
45     private Mono<ServerResponse> buildReviewsResponse(Flux<Review> reviewsFlux) {
46         return ServerResponse.ok().body(reviewsFlux, Review.class);
47     }
48
```

A screenshot of a terminal window showing the output of a curl command:

```
/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i http://localhost:8081/v1/reviews?movieInfoId=2
HTTP/1.1 200 OK
transfer-encoding: chunked
Content-Type: application/json

[{"reviewId": "2", "movieInfoId": 2, "comment": "Excellent Movie", "rating": 8.0}]
```

Assignment: Integration Test Case for the Reviews Service in the ReviewsIntgTest class

⌚ 30 minutes to complete | 🙋 103 student solutions

Write an integration test case for the below endpoints in the 1) Get All Reviews -> /v1/reviews 2) Update a Review using the PUT endpoint -> /v1/reviews/{id} 3) Delete a Review using the DELETE endpoint -> /v1/reviews/{id} 4) Get all reviews using the MovieInfoId -> /v1/reviews?movieInfoId=1

Unit test cases for functional web.

Because here no controller we need to remove it from `@WebFluxTest` `@ContextConfiguration` is important. Because it is a unit test case it is not automatically called the classes. We have to specify what classes are needed as part of unit testing.



```
import org.springframework.test.web.reactive.server.WebTestClient;
...
@WebFluxTest
@ContextConfiguration(classes = {ReviewRouter.class, ReviewHandler.class})
@AutoConfigureWebTestClient
public class ReviewsUnitTest {
    ...
    @MockBean
    private ReviewReactiveRepository reviewReactiveRepository;
    ...
    @Autowired
    private WebTestClient webTestClient;
}
}
```

Unit testing for save (post call)



```
@Test
void addReview() {
    //given
    var review = new Review(reviewId: null, movieInfoId: 1L, comment: "Awesome Movie", rating: 9.0);

    when(reviewReactiveRepository.save(isA(Review.class)))
        .thenReturn(Mono.just(new Review(reviewId: "abc", movieInfoId: 1L, comment: "Awesome Movie", rating: 9.0)));

    //when

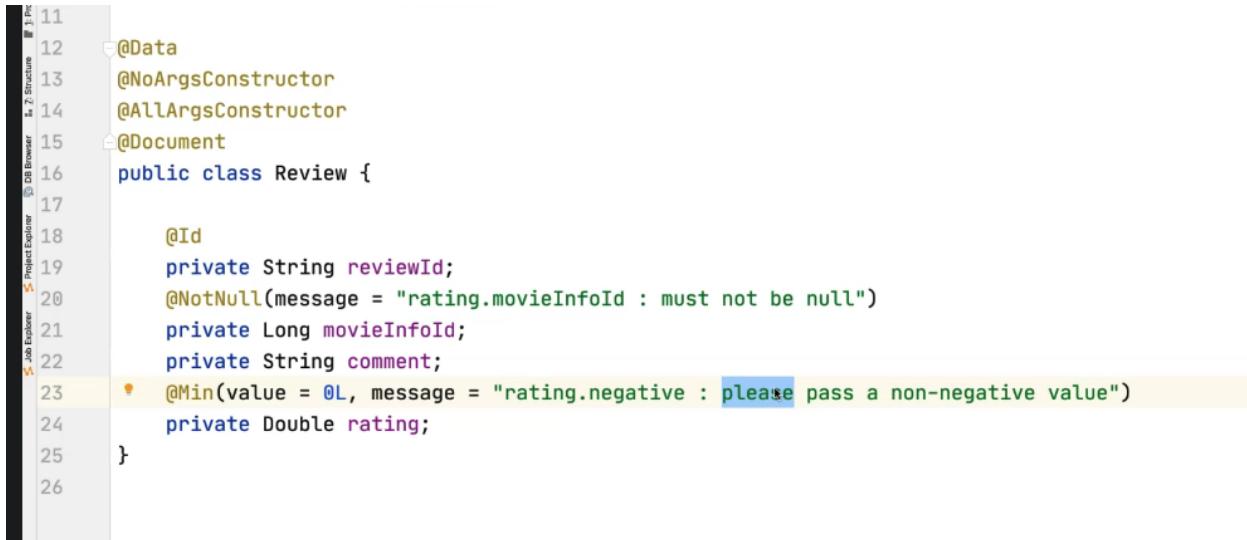
    webTestClient
        .post()
        .uri(REVIEWS_URL)
        .bodyValue(review)
        .exchange()
        .expectStatus()
        .isCreated()
        .expectBody(Review.class)
        .consumeWith(movieInfoEntityExchangeResult -> {
            var savedReview = movieInfoEntityExchangeResult.getResponseBody();
            assert savedReview!=null;
            assert savedReview.getReviewId()!=null;
        });
}
```

Assignment: Unit Test Cases for the Reviews Service in the ReviewsUnitTest class

⌚ 60 minutes to complete | 🧑 88 student solutions

Write an Unit test case for the below endpoints in the
1) Get All Reviews -> /v1/reviews
2) Update a Review using the PUT endpoint -> /v1/reviews/{id}
3) Delete a Review using the DELTE endpoint - > /v1/reviews/{id}
4) Get all reviews using the MovieInfoId -> /v1/reviews?movieInfo

Bean Validation in Functional Web.

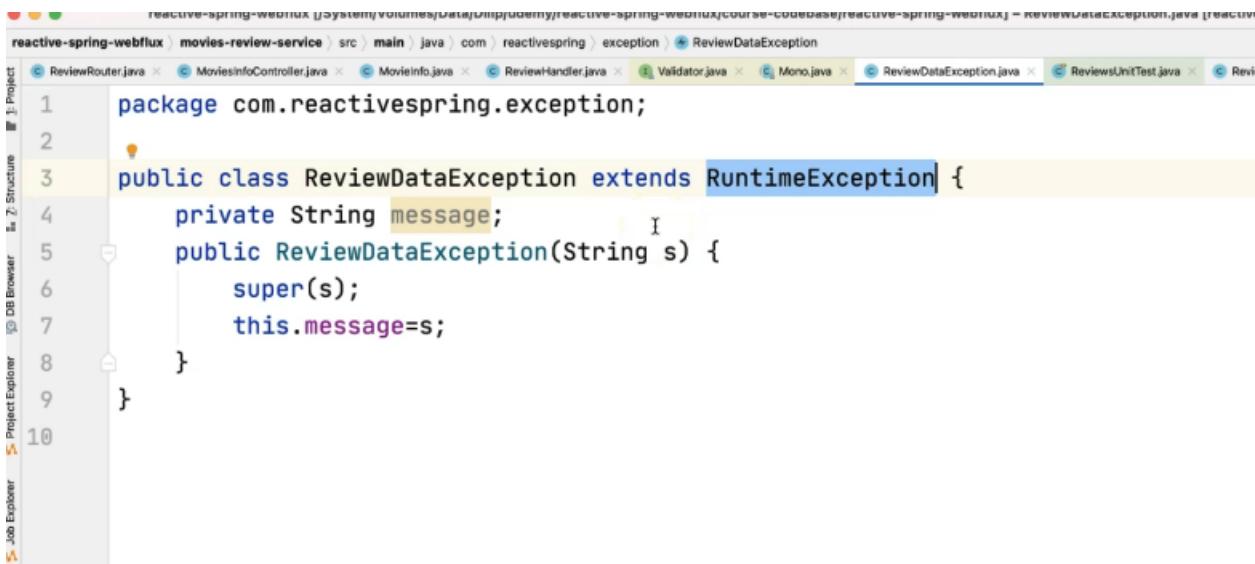


```
11
12     @Data
13     @NoArgsConstructor
14     @AllArgsConstructor
15     @Document
16     public class Review {
17
18         @Id
19         private String reviewId;
20         @NotNull(message = "rating.movieInfoId : must not be null")
21         private Long movieInfoId;
22         private String comment;
23         @Min(value = 0L, message = "rating.negative : please pass a non-negative value")
24         private Double rating;
25     }
26
```

Annotations are out of scope for functional web unlike @valid in @RequestBody

In the functional web we have to handle the below fashion.

doOnNext() is a side effect function where we can raise exceptions or do validations as shown above.



```
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
```

Introduce validator in the handler class

```

15     @Component
16     public class ReviewHandler {
17
18
19     @Autowired
20     private Validator validator;
21
22
23     public Mono<ServerResponse> addReview(ServerRequest request) {
24
25         return request.bodyToMono(Review.class)
26             .doOnNext(this::validate)
27             .flatMap(reviewReactiveRepository::save)
28             .flatMap(ServerResponse.status(HttpStatus.CREATED)::bodyValue);
29     }
30
31
32     private void validate(Review review) {
33
34         var constraintViolations = validator.validate(review);
35         log.info("constraintViolations : {}", constraintViolations);
36         if(constraintViolations.size() > 0) {
37             var errorMessage = constraintViolations
38                 .stream() Stream<ConstraintViolation<Review>>
39                 .map(ConstraintViolation::getMessage) Stream<String>
40                 .sorted()
41                 .collect(Collectors.joining( delimiter: ","));
42             throw new ReviewDataException(errorMessage);
43         }
44     }
45
46 }
47
48
49
50
51
52
53
54
55

```

Unit testing for bean validation

```

59     @Test
60     void addReview_validation() {
61         //given
62         var review = new Review( reviewId: null, movieInfoId: null, comment: "Awesome Movie", rating: -9.0);
63
64         when(reviewReactiveRepository.save(isA(Review.class)))
65             .thenReturn(Mono.just(new Review( reviewId: "abc", movieInfoId: 1L, comment: "Awesome Movie", rating: 9.0)));
66
67         //when
68
69         webTestClient
70             .post() WebTestClient.RequestBodyUriSpec
71             .uri(REVIEWS_URL) WebTestClient.RequestBodySpec
72             .bodyValue(review) WebTestClient.RequestHeadersSpec<capture of ?>
73             .exchange() WebTestClient.ResponseSpec
74             .expectStatus() StatusAssertions
75             .isBadRequest();
76

```

Run: MoviesReviewServiceApplication > ReviewsUnitTest.addReview_validation

Test	Time
Test Results	464 ms
ReviewsUnitTest	464 ms
addReview_validation()	464 ms

```

java.lang.AssertionError: Status expected:<400 BAD_REQUEST> but was:<500 INTERNAL_SERVER_ERROR>
expected :400 BAD_REQUEST
actual   :500 INTERNAL_SERVER_ERROR
Click to see difference

```

We are expecting a bad request but we are receiving an internal server error because it is throwing a runtime exception.

custom global exception handler in functional web.

```
11
12     @Component
13     @Slf4j
14     public class GlobalErrorHandler implements ErrorWebExceptionHandler {
15         @Override
16         public Mono<Void> handle(ServerWebExchange exchange, Throwable ex) {
17             log.error("Exception message is {}", ex.getMessage(), ex);
18             DataBufferFactory dataBufferFactory = exchange.getResponse().bufferFactory();
19             var errorMessage = dataBufferFactory.wrap(ex.getMessage().getBytes());
20
21             if(ex instanceof ReviewDataException){
22                 exchange.getResponse().setStatusCode(HttpStatus.BAD_REQUEST);
23                 return exchange.getResponse().writeWith(Mono.just(errorMessage));
24             }
25             exchange.getResponse().setStatusCode(HttpStatus.INTERNAL_SERVER_ERROR);
26             return exchange.getResponse().writeWith(Mono.just(errorMessage));
27         }
28     }
```

Add this class in the unit test class.

```
1
2     @WebFluxTest
3     @ContextConfiguration(classes = {ReviewRouter.class, ReviewHandler.class, GlobalErrorHandler.class})
4     @AutoConfigureWebTestClient
5     public class ReviewsUnitTests {
```

Refactor the unit test case

```
60
61     @Test
62     void addReview_validation() {
63         //given
64         var review = new Review(reviewId: null, movieInfoId: null, comment: "Awesome Movie", rating: -9.0);
65
66         when(reviewReactiveRepository.save(isA(Review.class)))
67             .thenReturn(Mono.just(new Review(reviewId: "abc", movieInfoId: 1L, comment: "Awesome Movie", rating: 9.0)));
68
69         //when
70
71         webTestClient
72             .post() WebTestClient.RequestBodyUriSpec
73             .uri(REVIEWS_URL) WebTestClient.RequestBodySpec
74             .bodyValue(review) WebTestClient.RequestHeadersSpec<capture of ?>
75             .exchange() WebTestClient.ResponseSpec
76             .expectStatus() StatusAssertions
77             .isBadRequest() WebTestClient.ResponseSpec
78             .expectBody(String.class) WebTestClient.BodySpec<String, capture of ?>
79             .isEqualTo("rating.movieInfoId : must not be null,rating.negative : please pass a non-negative value !");
80     }
81 }
```

Handling resources not found in functional web.

Resource NotFound(404) in Update Review using GlobalErrorHandler

```
package com.reactive.spring.exception;

public class ReviewNotFoundException extends RuntimeException{

    private String message;
    private Throwable ex;

    public ReviewNotFoundException( String message, Throwable ex) {
        super(message, ex);
        this.message = message;
        this.ex = ex;
    }

    public ReviewNotFoundException(String message) {
        super(message);
        this.message = message;
    }
}
```

Just add logic if review id not found

```
public Mono<ServerResponse> updateReview(ServerRequest request) {

    var reviewId = request.pathVariable("id");
    var existingReview = reviewReactiveRepository.findById(reviewId)
        .switchIfEmpty(Mono.error(new ReviewNotFoundException("Review not found for the given Review id " + reviewId)));

    return existingReview
        .flatMap(review -> request.bodyToMono(Review.class))
        .map(reqReview -> {
            review.setComment(reqReview.getComment());
            review.setRating(reqReview.getRating());
            return review;
        })
        .flatMap(reviewReactiveRepository::save)
        .flatMap(savedReview -> ServerResponse.ok().bodyValue(savedReview));
}
```

Add this if statement in the global exception handler

```
return exchange.getResponse().writeWith(Mono.just(errorMessage));
}

if(ex instanceof ReviewNotFoundException){
    exchange.getResponse().setStatus(HttpStatus.NOT_FOUND);
    return exchange.getResponse().writeWith(Mono.just(errorMessage));
}
exchange.getResponse().setStatus(HttpStatus.INTERNAL_SERVER_ERROR);
return exchange.getResponse().writeWith(Mono.just(errorMessage));
```

```
/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i \
\ -d '{"reviewId":1, "movieInfoId":1, "comment": "Excellent Movie Update", "rating":8.5}' \
\ -H "Content-Type: application/json" \
\ -X PUT http://localhost:8081/v1/reviews/100
HTTP/1.1 404 Not Found
content-length: 44
```

Another approach:



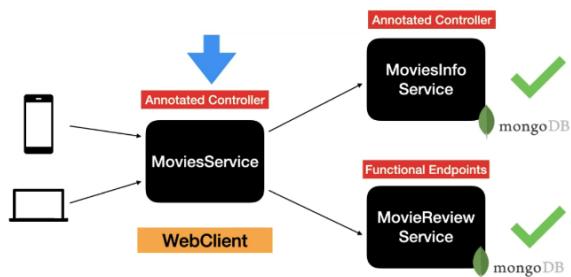
```
76
77 @ ...
78
79
80
81     var existingReview = reviewReactiveRepository.findById(reviewId);
82         //switchIfEmpty(Mono.error(new ReviewNotFoundException("Review not found for the given Review id "+ reviewId)));
83
84     return existingReview
85         .flatMap(review -> request.bodyToMono(Review.class)
86             .map(reqReview -> {
87                 review.setComment(reqReview.getComment());
88                 review.setRating(reqReview.getRating());
89                 return review;
90             })
91             .flatMap(reviewReactiveRepository::save)
92             .flatMap(savedReview -> ServerResponse.ok().bodyValue(savedReview)))
93         .switchIfEmpty(ServerResponse.notFound().build());
94 }
```

Here we gave not found at the last line. But disadvantage is no custom message.

```
/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i \
-d '{"reviewId":1, "movieInfoId":1, "comment": "Excellent Movie Update", "rating":8.5}' \
-H "Content-Type: application/json" \
-X PUT http://localhost:8081/v1/reviews/100
HTTP/1.1 404 Not Found
content-length: 0
```

Movie Service which talks with MovieInfo and MovieReview Service.

Movies Application using MicroServices Pattern



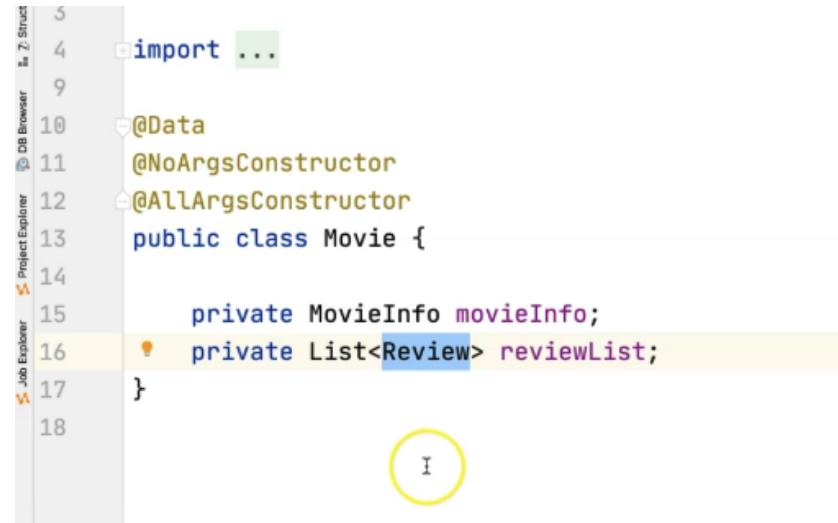
We use webclient here for connecting other services.

The application does not have dependency on DB.

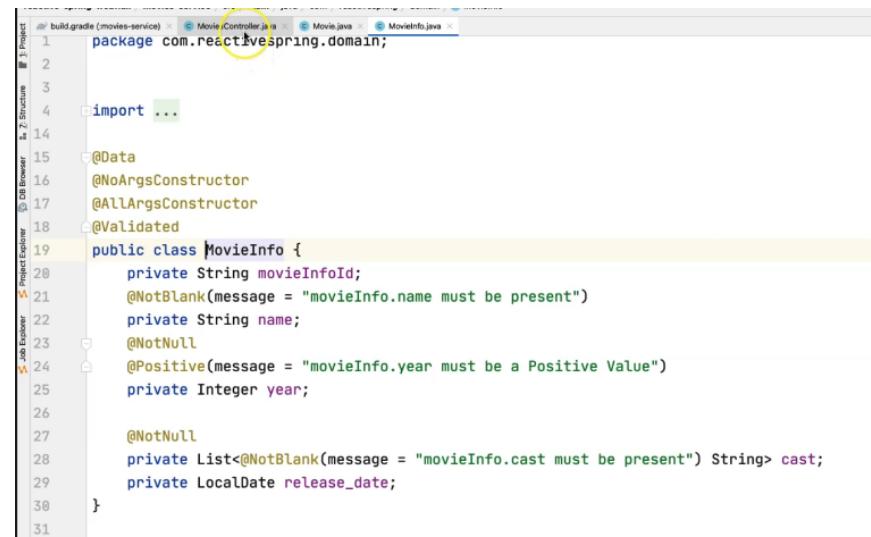


```
20
21 > dependencies {
22     implementation 'org.springframework.boot:spring-boot-starter-validation'
23     implementation 'org.springframework.boot:spring-boot-starter-webflux'
24
25     compileOnly 'org.projectlombok:lombok'
26     annotationProcessor 'org.projectlombok:lombok'
27
28     testImplementation 'org.springframework.boot:spring-boot-starter-test'
29     testImplementation 'io.projectreactor:reactor-test'
30 }
```

We are trying to get this data after connecting external service.



```
5
6 import ...
7
8 @Data
9 @NoArgsConstructor
10 @AllArgsConstructor
11 public class Movie {
12
13     private MovieInfo movieInfo;
14     private List<Review> reviewList;
15 }
```

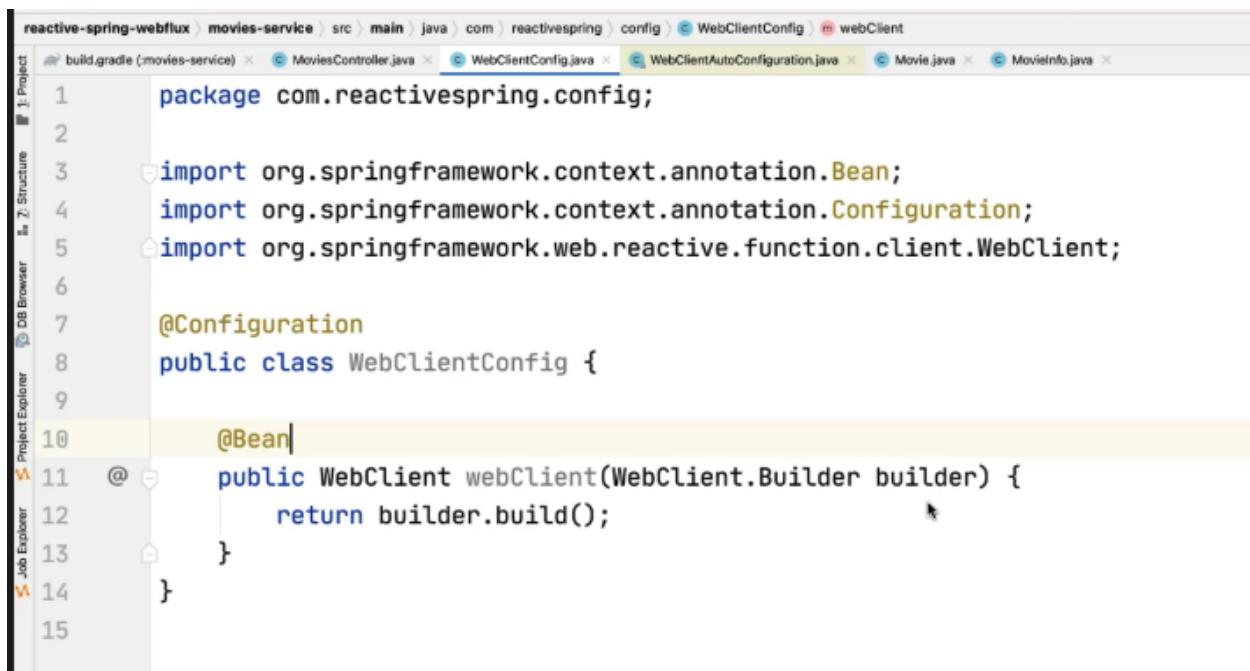


```
1 package com.reactive.spring.domain;
2
3
4 import ...
5
6 @Data
7 @NoArgsConstructor
8 @AllArgsConstructor
9 @Validated
10 public class MovieInfo {
11     private String movieInfoId;
12     @NotBlank(message = "movieInfo.name must be present")
13     private String name;
14     @NotNull
15     @Positive(message = "movieInfo.year must be a Positive Value")
16     private Integer year;
17
18     @NotNull
19     private List<@NotBlank(message = "movieInfo.cast must be present") String> cast;
20     private LocalDate releaseDate;
21 }
```

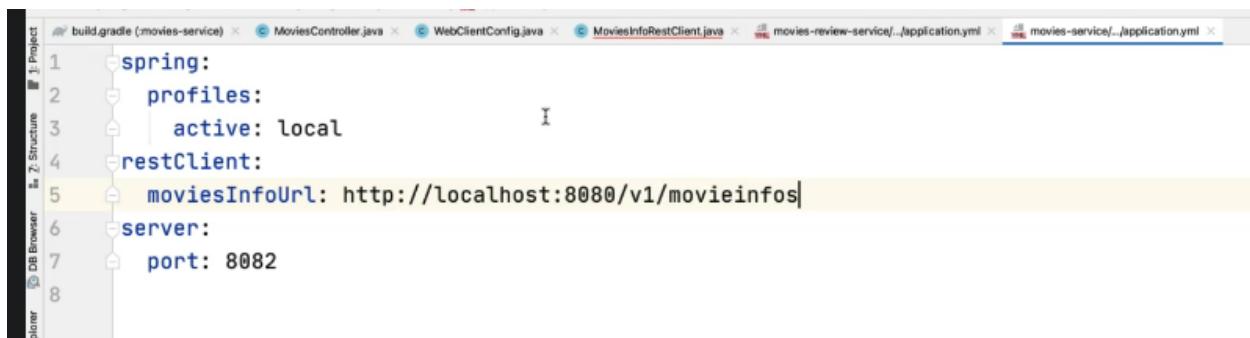
WebClient

- It is a reactive non-blocking Rest Client
- It uses a functional style API
- It allows the application to interact with other services in a non-blocking fashion
- Its auto configured in to the application by Spring Boot

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



```
reactive-spring-webflux > movies-service > src > main > java > com > reactivespring > config > WebClientConfig.java
1 package com.reactivespring.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.reactive.function.client.WebClient;
6
7 @Configuration
8 public class WebClientConfig {
9
10     @Bean
11     public WebClient webClient(WebClient.Builder builder) {
12         return builder.build();
13     }
14 }
```



```
build.gradle (.movies-service) < MoviesController.java < WebClientConfig.java < MoviesInfoRestClient.java < movies-review-service/.../application.yml < movies-service/.../application.yml
1 spring:
2   profiles:
3     active: local
4   restClient:
5     moviesInfoUrl: http://localhost:8080/v1/movieinfos
6   server:
7     port: 8082
```

The screenshot shows an IDE interface with a Java code editor. The code is part of a class named `MoviesInfoRestClient`. It uses the `WebClient` API to retrieve movie information from a URL. The code includes annotations like `@Component` and `@Value`, and methods for initializing the client and performing the retrieval.

```
reactive-spring-webflux [/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux] - MoviesInfoRestClient.java [reactive-spring-webflux movies-service src main java com reactiveSpring client MoviesInfoRestClient retrieveMovieInfo]
build.gradle (movies-service) MoviesController.java MoviesController.java WebClientConfig.java MoviesInfoRestClient.java movies-review-service./application.yml movies-service./app

@Component
public class MoviesInfoRestClient {

    private WebClient webClient;

    @Value("${restClient.moviesInfoUrl}")
    private String moviesInfoUrl;

    public MoviesInfoRestClient(WebClient webClient) {
        this.webClient = webClient;
    }

    public Mono<MovieInfo> retrieveMovieInfo(String movieId){
        I
        var url = moviesInfoUrl.concat("/{id}");
        webClient
            .get() WebClient.RequestHeadersUriSpec<capture of ?>
            .uri(url, movieId) capture of ?
            .retrieve() WebClient.ResponseSpec
            .bodyToMono(MovieInfo.class) Mono<MovieInfo>
            .log();
    }
}
```

Add return in the line number 25.

The screenshot shows the same Java code as before, but with a modification. A `return` statement has been added at line 25, immediately after the `webClient` declaration. This change is intended to fix a compilation error.

```
public Mono<MovieInfo> retrieveMovieInfo(String movieId){

    var url = moviesInfoUrl.concat("/{id}");
    return webClient
        .get() WebClient.RequestHeadersUriSpec<capture of ?>
        .uri(url, movieId) capture of ?
        .retrieve() WebClient.ResponseSpec
        .bodyToMono(MovieInfo.class) Mono<MovieInfo>
```

```
reactive-spring-webflux > movies-service > src > main > resources > application.yml
build.gradle (:movies-service) < MoviesController.java < ReviewsRestClient.java < ReviewHandler.java < MoviesInfoRestClient.java < application.yml
```

```
spring:
  profiles:
    active: local
  restClient:
    moviesInfoUrl: http://localhost:8080/v1/movieinfos
    reviewsUrl: http://localhost:8081/v1/reviews
  server:
    port: 8082
```

```
reactive-spring-webflux > movies-service > src > main > java > com > reactivespring > client > ReviewsRestClient > retrieveReviews
```

```
19 @Value("${restClient.reviewsUrl}")
20 private String reviewsUrl;
21
22 //movieInfoId
23
24 public Flux<Review> retrieveReviews(String movieId){
25
26     var url = UriComponentsBuilder.fromHttpUrl(reviewsUrl)
27         .queryParam("movieInfoId", movieId)
28         .buildAndExpand().toUriString();
29
30     return webClient
31         .get() WebClient.RequestHeadersUriSpec<capture of ?>
32         .uri(url) capture of ?
33         .retrieve() WebClient.ResponseSpec
34         .bodyToFlux(Review.class);
35 }
36
37 }
```

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "reactive-spring-webflux" and contains a module "movies-service".
- Code Editor:** The file "MoviesController.java" is open, showing Java code for a Spring WebFlux controller.
- Imports:** The code includes imports for `MoviesInfoRestClient`, `ReviewsRestClient`, `Movie`, and `Flux`.
- Annotations:** The code uses `@GetMapping` and `@PathVariable` annotations.
- Code Snippet:** The highlighted code block is:

```
private ReviewsRestClient reviewsRestClient;

public MoviesController(MoviesInfoRestClient moviesInfoRestClient, ReviewsRestClient reviewsRestClient) {
    this.moviesInfoRestClient = moviesInfoRestClient;
    this.reviewsRestClient = reviewsRestClient;
}

@GetMapping("/{id}")
public Mono<Movie> retrieveMovieById(@PathVariable("id") String movieId) {
    return moviesInfoRestClient.retrieveMovieInfo(movieId)
        .flatMap(movieInfo -> {
            var reviewsListMono = reviewsRestClient.retrieveReviews(movieId)
                .collectList();

            return reviewsListMono.map(reviews -> new Movie(movieInfo, reviews));
        });
}
```
- IDE UI:** The IDE has standard toolbars and panes for Project Explorer, DB Browser, and Help.

Exceptions in Service to Service Communication

Http Failures

- Http Failure falls in to two categories:
 - 4xx - Client Error
 - 400(Bad Request), 404 (Not Found) and etc.,
 - 5xx - Server Error
 - 500(Internal Server Error), 503 (Service Unavailable) and etc.,

```

reactive-spring-webflux [/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux] - MoviesInfoController.java [reactive-spring-webflux.movies-info-service.main]
reactive-spring-webflux movies-info-service src main java com/reactivespring/controller MoviesInfoController @GetMapping("/movieinfos/{id}")
37     return moviesInfoService.getMovieInfoById(id);
38 }
39 */
40 @GetMapping("/movieinfos/{id}")
41 public Mono<ResponseEntity<MovieInfo>> getMovieInfoById(@PathVariable String id){
42     return moviesInfoService.getMovieInfoById(id).Mono<MovieInfo>
43         .map(movieInfo1 -> ResponseEntity.ok()
44             .body(movieInfo1)).Mono<ResponseEntity<MovieInfo>>
45             .switchIfEmpty(Mono.just(ResponseEntity.notFound().build()))
46             .log();
47 }
48
49 @PostMapping("/movieinfos")
50 @ResponseStatus(HttpStatus.CREATED)
51 public Mono<MovieInfo> addMovieInfo(@RequestBody @Valid MovieInfo movieInfo){
52     return moviesInfoService.addMovieInfo(movieInfo).log();
53 }
54
55 @PutMapping("/movieinfos/{id}")
56 public Mono<ResponseEntity<MovieInfo>> updateMovieInfo(@RequestBody MovieInfo updatedMovieInfo, @PathVariable String id){
57     return moviesInfoService.updateMovieInfo(updatedMovieInfo, id).Mono<MovieInfo>
58         .map(ResponseEntity.ok():body).Mono<ResponseEntity<MovieInfo>>
59         .switchIfEmpty(Mono.error(new MovieInfoNotFoundException("MovieInfo Not Found")))
60         // .switchIfEmpty(Mono.just(ResponseEntity.notFound().build()))
61         // .switchIfEmpty(Mono.just(ResponseEntity.status(HttpStatus.NOT_FOUND).body("MovieInfo Not Found")))
62 }

```

"timestamp": "2021-09-21T01:25:15.671+00:00", "path": "/v1/movies/3", "status": 500, "error": "Internal Server Error", "requestId": "7b6ab51e-1"}
/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i http://localhost:8082/v1/movies/3
HTTP/1.1 500 Internal Server Error
Content-Type: application/json
Content-Length: 137
{"timestamp": "2021-09-21T09:51:16.757+00:00", "path": "/v1/movies/3", "status": 500, "error": "Internal Server Error", "requestId": "aaa5f982-1"}
curl -i http://localhost:8082/v1/movies/3

But we get 500 error instead of 404 (as there is no data with 3)

```

MoviesInfoController.java MoviesController.java WebClient.java ReviewsUnitTest.java MoviesInfoRestClient.java MoviesInfoClientException.java MoviesInfoServerException.java ReviewsClientException.java ReviewsServerException.java
26 public Mono<MovieInfo> retrieveMovieInfo(String movieId) {
27
28     var url = moviesInfoUrl.concat("/{id}");
29     return webClient
30         .get() WebClient.RequestHeadersUriSpec<capture of ?>
31         .uri(url, movieId) capture of ?
32         .retrieve() WebClient.ResponseSpec
33         .onStatus(HttpStatus::is4xxClientError, clientResponse -> {
34             if(clientResponse.statusCode().equals(HttpStatus.NOT_FOUND)){
35                 return Mono.error(new MoviesInfoClientException(
36                     "There is no MovieInfo Available for the passed in Id : " + movieId,
37                     clientResponse.statusCode().value()));
38             }
39
40             return clientResponse.bodyToMono(String.class)
41                 .flatMap(responseMessage -> Mono.error(new MoviesInfoClientException(
42                     responseMessage, clientResponse.statusCode().value()
43                     )));
44         })
45         .bodyToMono(MovieInfo.class) Mono<MovieInfo>
46         .log();
47 }

```

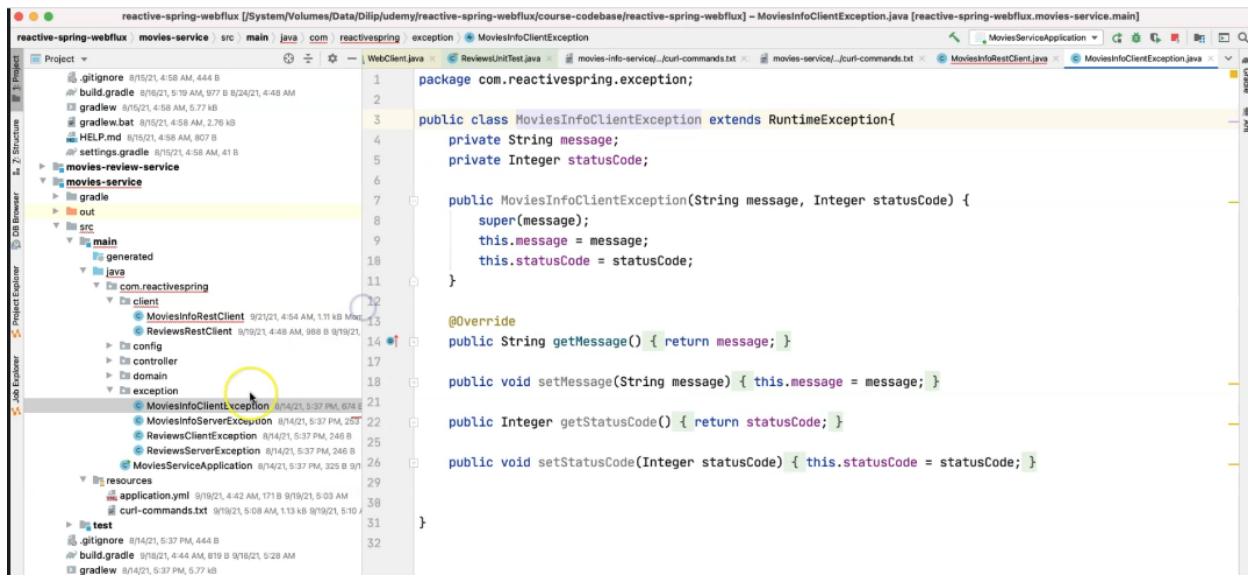
```
1 package com.reactivespring.globalerrorhandler;
2
3 import com.reactivespring.exception.MoviesInfoClientException;
4 import lombok.extern.slf4j.Slf4j;
5 import org.springframework.http.ResponseEntity;
6 import org.springframework.web.bind.annotation.ControllerAdvice;
7 import org.springframework.web.bind.annotation.ExceptionHandler;
8
9 @ControllerAdvice
10 @Slf4j
11 public class GlobalErrorHandler {
12
13     @ExceptionHandler(MoviesInfoClientException.class)
14     public ResponseEntity<String> handleClientException(MoviesInfoClientException exception) {
15         log.error("Exception Caught in handleClientException : {}", exception.getMessage());
16         return ResponseEntity.status(exception.getStatusCode()).body(exception.getMessage());
17     }
18 }
19
```

Add line number 34 in the below code.

```
26 PUBLIC Mono<MovieInfo> retrieveMovieInfo(String movieId){}
27
28     var url = moviesInfoUrl.concat("/{id}");
29     return webClient
30         .get() WebClient.RequestHeadersUriSpec<capture of ?>
31         .uri(url, movieId) capture of ?
32         .retrieve() WebClient.ResponseSpec
33         .onStatus(HttpStatus::is4xxClientError, clientResponse -> {
34             log.info("Status code is : {}", clientResponse.statusCode().value());
35             if(clientResponse.statusCode().equals(HttpStatus.NOT_FOUND)){
36                 return Mono.error(new MoviesInfoClientException(
37                     "There is no MovieInfo Available for the passed in Id : " + movieId,
38                     clientResponse.statusCode().value()));
39             }
40         })
41         .onStatus(HttpStatus::is4xxClientError, clientResponse -> {
42             log.info("Status code is : {}", clientResponse.statusCode().value());
43             if(clientResponse.statusCode().equals(HttpStatus.NOT_FOUND)){
44                 return Mono.error(new MoviesInfoClientException(
45                     "There is no MovieInfo Available for the passed in Id : " + movieId,
46                     clientResponse.statusCode().value()));
47             }
48         })
49     }
50
51     Terminal: Local
52
53     HTTP/1.1 500 Internal Server Error
54     Content-Type: application/json
55     Content-Length: 137
56
57     {"timestamp":"2021-09-21T09:51:16.757+00:00","path":"/v1/movies/3","status":500,"error":"Internal Server Error","requestId":"aaa5f982-1"}
58
59 /System/Volumes/Data/Dilip/udemy/reactive-spring-webFlux/course-codebase/reactive-spring-webFlux » curl -i http://localhost:8082/v1/movies/3
60
61 HTTP/1.1 404 Not Found
62 Content-Type: text/plain;charset=UTF-8
63 Content-Length: 56
64
65 There is no MovieInfo Available for the passed in Id : 3
```

Now we got a 404 error and with an appropriate message.
onStatus if the first argument is true then function Interface will be called.

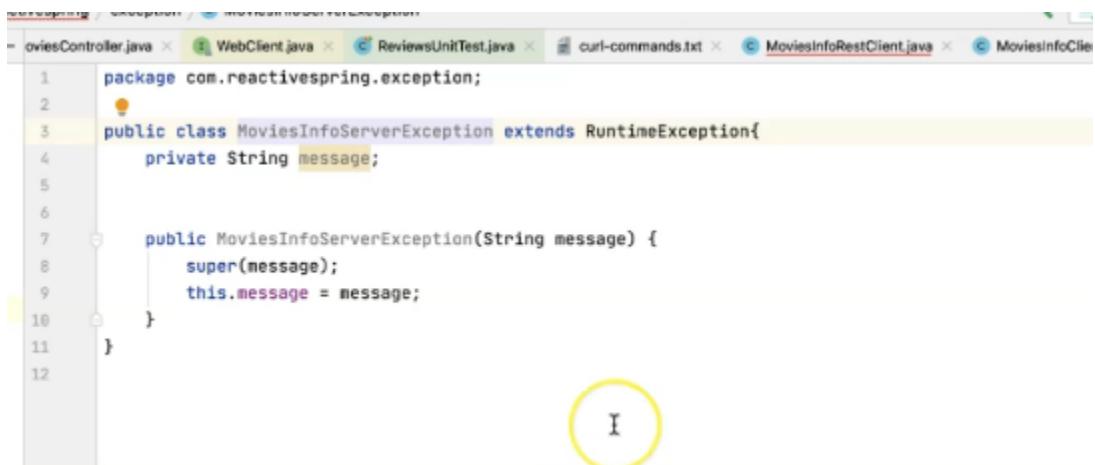
Different exception classes



```

1 package com.reactiveSpring.exception;
2
3 public class MoviesInfoClientException extends RuntimeException{
4     private String message;
5     private Integer statusCode;
6
7     public MoviesInfoClientException(String message, Integer statusCode) {
8         super(message);
9         this.message = message;
10        this.statusCode = statusCode;
11    }
12
13    @Override
14    public String getMessage() { return message; }
15
16    public void setMessage(String message) { this.message = message; }
17
18    public Integer getStatusCode() { return statusCode; }
19
20    public void setStatusCode(Integer statusCode) { this.statusCode = statusCode; }
21
22}
23
24
25
26
27
28
29
30
31
32

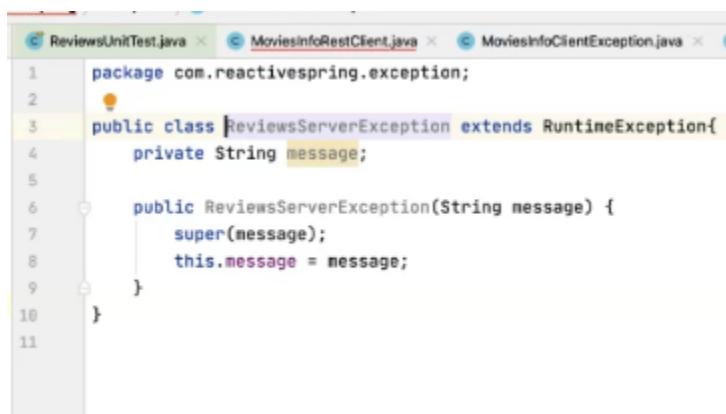
```



```

1 package com.reactiveSpring.exception;
2
3 public class MoviesInfoServerException extends RuntimeException{
4     private String message;
5
6
7     public MoviesInfoServerException(String message) {
8         super(message);
9         this.message = message;
10    }
11
12}
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

```



```

1 package com.reactiveSpring.exception;
2
3 public class ReviewsServerException extends RuntimeException{
4     private String message;
5
6
7     public ReviewsServerException(String message) {
8         super(message);
9         this.message = message;
10    }
11
12}
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

```

Stop movieInfo service

```

There is no MovieInfo Available for the passed in Id : 3
/System/Volumes/Data/Dilip/udemy/reactive-spring-webflux/course-codebase/reactive-spring-webflux » curl -i http://localhost:8082/v1/movies/2
HTTP/1.1 500 Internal Server Error
Content-Type: application/json
Content-Length: 137

{"timestamp":"2021-09-21T10:10:53.851+00:00","path":"/v1/movies/2","status":500,"error":"Internal Server Error","requestId":"e600ebc8-1"}Y

```

```

org.springframework.web.reactive.function.client.WebClientRequestException: Connection refused: localhost/127.0.0.1:8080; i
    at org.springframework.web.reactive.function.client.ExchangeFunctions$DefaultExchangeFunction.lambda$wrapException$9(E)
    Suppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException:
Error has been observed at the following site(s):
    |_ checkpoint → Request to GET http://localhost:8080/v1/movieinfos/2 [DefaultWebClient]
Stack trace:
    at org.springframework.web.reactive.function.client.ExchangeFunctions$DefaultExchangeFunction.lambda$wrapException$9(E)

```

```

        return clientResponse.bodyToMono(String.class)
            .flatMap(responseMessage -> Mono.error(new MoviesInfoClientException(
                responseMessage, clientResponse.statusCode().value()
            )));
    }
    .onStatus(HttpStatus::is5xxServerError, clientResponse -> {
        log.info("Status code is : {}", clientResponse.statusCode().value());

        return clientResponse.bodyToMono(String.class)
            .flatMap(responseMessage -> Mono.error(new MoviesInfoServerException(
                "Server Exception in MoviesInfoService " + responseMessage)));
    })
    .bodyToMono(MovieInfo.class) Mono<MovieInfo>
    .log();
}


```

```

@CONTROLLERADVICE
@Slf4j
public class GlobalErrorHandler {

    @ExceptionHandler(MoviesInfoClientException.class)
    public ResponseEntity<String> handleClientException(MoviesInfoClientException exception) {
        log.error("Exception Caught in handleClientException : {}", exception.getMessage());
        return ResponseEntity.status(exception.getStatusCode()).body(exception.getMessage());
    }

    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<String> handleRunTimeException(RuntimeException exception) {
        log.error("Exception Caught in handleClientException : {}", exception.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(exception.getMessage());
    }
}

```

It is difficult to test the above scenario. Because our server response is not 500. So in the above case it will try to capture controller advise.

The screenshot shows a Java IDE interface with several tabs at the top: MoviesController.java, MoviesInfoRestClient.java, ReviewsRestClient.java, ReviewHandler.java, ReviewsClientException.java, and GlobalErrorHandler.java. The ReviewsRestClient.java tab is active. The code in the editor is as follows:

```
var url = UriComponentsBuilder.fromHttpUrl(reviewsUrl)
    .queryParam("name", movieInfoId)
    .buildAndExpand().toUriString();

return webClient
    .get() WebClient.RequestHeadersUriSpec<capture of ?>
    .uri(url) capture of ?
    .retrieve() WebClient.ResponseSpec
    .onStatus(HttpStatus::is4xxClientError, clientResponse -> {
        log.info("Status code is : {}", clientResponse.statusCode().value());
        if(clientResponse.statusCode().equals(HttpStatus.NOT_FOUND)){
            return Mono.empty();
        }

        return clientResponse.bodyToMono(String.class)
            .flatMap(responseMessage -> Mono.error(new ReviewsClientException(
                responseMessage)));
    })
    .onStatus(HttpStatus::is5xxServerError, clientResponse -> {
        log.info("Status code is : {}", clientResponse.statusCode().value());
        return clientResponse.bodyToMono(String.class)
            .flatMap(responseMessage -> Mono.error(new ReviewsServerException(
                "Server Exception in ReviewsService " + responseMessage)));
    })
    .bodyToFlux(Review.class);
}
```

Review test client code with 4XX and 5XX error