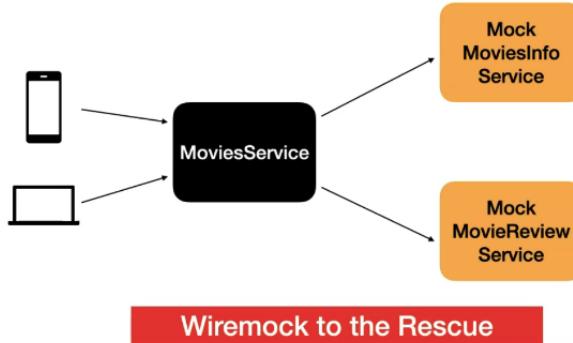


This document contains continuation wiremock, retrying http calls , server sent events (SSE).



## Benefits of WireMock

- Easy to test the success scenarios (2xx)
  - Test the contract
  - Serialization/Deserialization
- Easy to simulate error scenarios
  - 4xx
  - 5xx
- SocketTimeout Exceptions and more..,

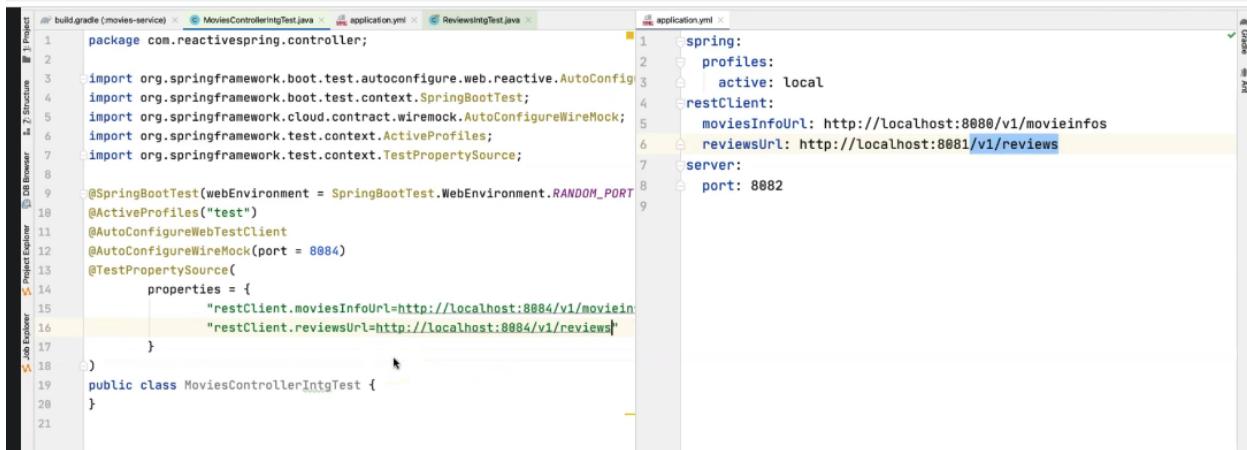
```
30
31
32 // wiremock
33 testImplementation 'org.springframework.cloud:spring-cloud-starter-contract-stub-runner:3.0.3'
34 }
```



```
1 package com.reactivespring.controller;
2
3 import org.springframework.boot.test.autoconfigure.web.reactive.AutoConfigureWebTestClient;
4 import org.springframework.boot.test.context.SpringBootTest;
5 import org.springframework.cloud.contract.wiremock.AutoConfigureWireMock;
6 import org.springframework.test.context.ActiveProfiles;
7
8 @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
9 @ActiveProfiles("test")
10 @AutoConfigureWebTestClient
11 @AutoConfigureWireMock(port = 8084)
12 public class MoviesControllerIntgTest {
13 }
14
```

The line number 11 `@AutoConfigureWireMock` will spin up the http server and give a response to us.

Because our wiremock server is on 8084 we have to introduce test properties notice the difference between left values and right values ( we are overriding)



```
1 package com.reactivespring.controller;
2
3 import org.springframework.boot.test.autoconfigure.web.reactive.AutoConfigureWebTestClient;
4 import org.springframework.boot.test.context.SpringBootTest;
5 import org.springframework.cloud.contract.wiremock.AutoConfigureWireMock;
6 import org.springframework.test.context.ActiveProfiles;
7 import org.springframework.test.context.TestPropertySource;
8
9 @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
10 @ActiveProfiles("test")
11 @AutoConfigureWebTestClient
12 @AutoConfigureWireMock(port = 8084)
13 @TestPropertySource(
14     properties = {
15         "restClient.moviesInfoUrl=http://localhost:8084/v1/movieinfos"
16         "restClient.reviewsUrl=http://localhost:8084/v1/reviews"
17     }
18 )
19 public class MoviesControllerIntgTest {
20 }
21
```

application.yml:

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

Compare wiremock syntax below

The screenshot shows a Java code editor with a test class named `MoviesControllerIntgTest`. The code uses the `WebTestClient` from the `com.reactive.spring.controller` package to interact with a service application. It includes stubbing logic for a movie ID and assertions for the response status.

```
24
25 public class MoviesControllerIntgTest {
26
27     @Autowired
28     WebTestClient webTestClient;
29
30
31     @Test
32     void retrieveMovieById() {
33         //given
34         var movieId = "abc";
35         stubFor(get(urlEqualTo(testUrl: "/v1/movieinfos" + "/" + movieId))
36             .willReturn(aResponse()
37                 .withHeader(key: "Content-Type", ...values: "application/json")
38                 .withBodyFile("movieinfo.json")));
39
40         //when
41         webTestClient
42             .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
43             .uri(uri: "/v1/movies/{id}", movieId) capture of ?
44             .exchange() WebTestClient.ResponseSpec
45             .expectStatus().isOk()
46     }
47 }
```

The code editor interface includes toolbars for Build, Run, DB Execution Console, TODO, Messages, and Terminal, along with a Project Explorer and a right-hand pane for the Event Log.

The screenshot shows a browser window displaying a JSON object representing a movie. The object includes fields like `movieInfoId`, `name`, `year`, `cast`, and `release_date`. The value for `name` is highlighted with a yellow circle.

```
1 {
2     "movieInfoId": "1",
3     "name": "Batman Begins",
4     "year": 2005,
5     "cast": [
6         "Christian Bale",
7         "Michael Caine"
8     ],
9     "release_date": "2005-06-15"
10 }
```

```

15 //given
16 var movieId = "abc";
17 stubFor(get(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId))
18     .willReturn(aResponse()
19         .withHeader( key: "Content-Type", ...values: "application/json")
20         .withBodyFile("movieinfo.json")));
21
22 stubFor(get(urlPathEqualTo( testUrl: "v1/reviews"))
23     .willReturn(aResponse()
24         .withHeader( key: "Content-Type", ...values: "application/json")
25         .withBodyFile("reviews.json")));
26
27 //when
28 webTestClient
29     .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
30     .uri( uri: "/v1/movies/{id}", movieId) capture of ?
31     .exchange() WebTestClient.ResponseSpec
32     .expectStatus().isOk()
33     .expectBody(Movie.class) WebTestClient.BodySpec<Movie, capture of ?>
34     .consumeWith(movieEntityExchangeResult -> {
35         var movie = movieEntityExchangeResult.getResponseBody();
36         assert Objects.requireNonNull(movie).getReviewList().size() ==2;
37         assertEquals( expected: "Batman Begins", movie.getMovieInfo().getName());
38     });

```

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

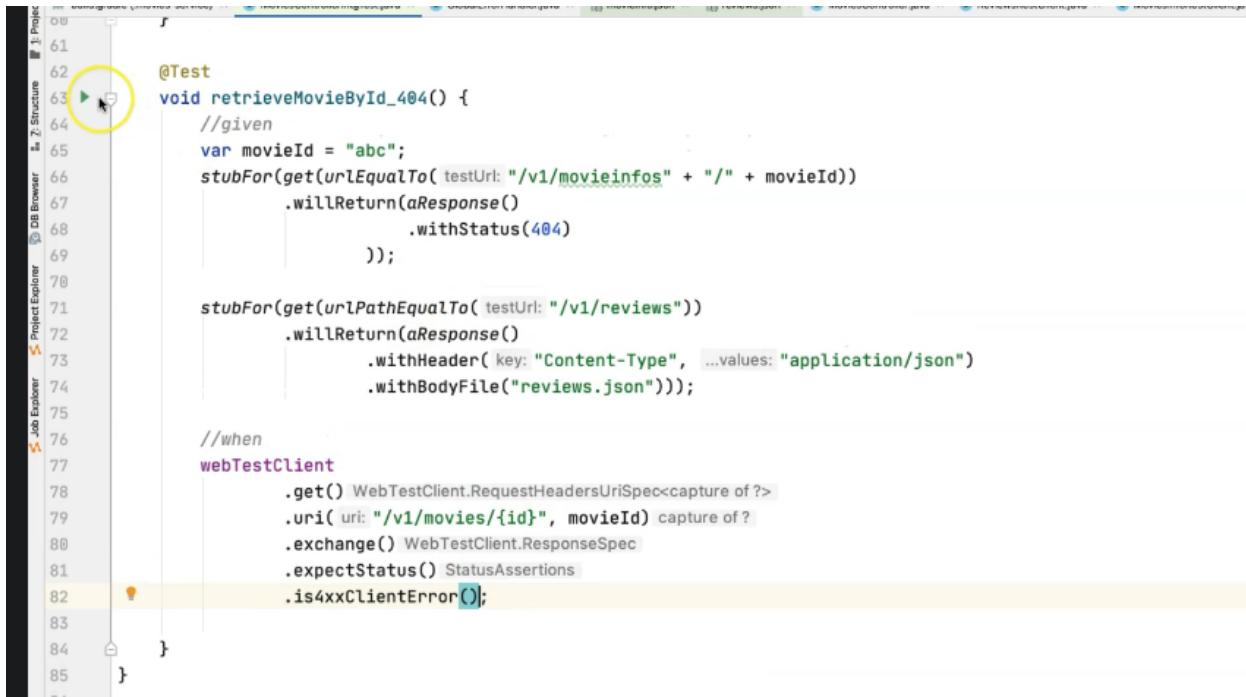
Wiremock will give lot of logs to understand

```

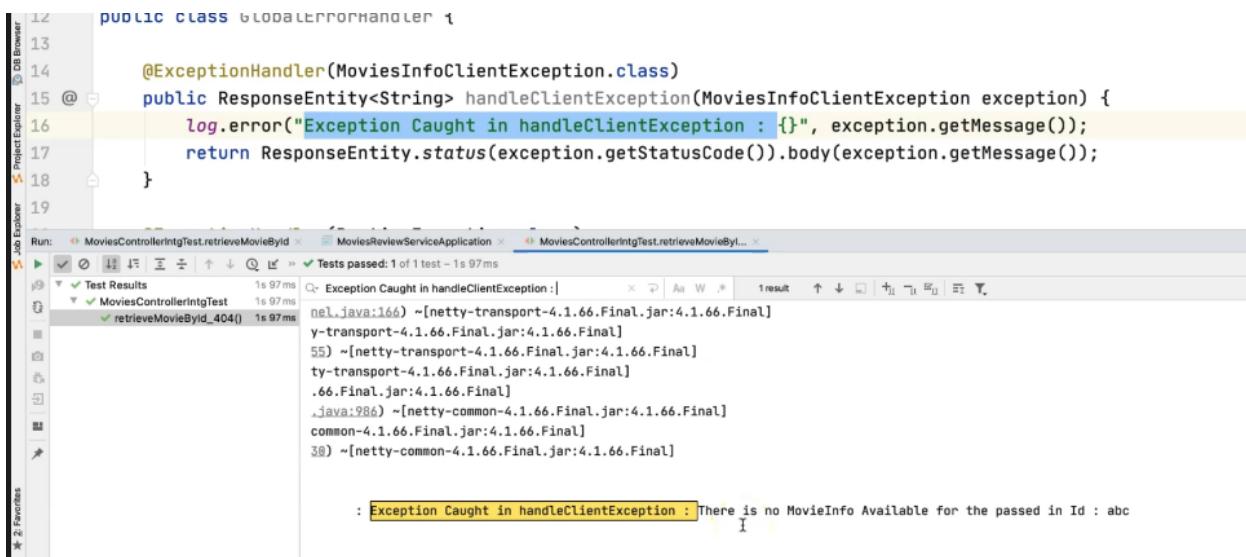
Matched response definition:
{
    "status" : 200,
    "bodyFileName" : "reviews.json",
    "headers" : {
        "Content-Type" : "application/json"
    }
}

Response:
HTTP/1.1 200

```



```
61
62
63 @Test
64 void retrieveMovieById_404() {
65     //given
66     var movieId = "abc";
67     stubFor(get(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId))
68             .willReturn(aResponse()
69                         .withStatus(404)
70             ));
71
72     stubFor(get(urlPathEqualTo( testUrl: "/v1/reviews"))
73             .willReturn(aResponse()
74                         .withHeader( key: "Content-Type", ...values: "application/json")
75                         .withBodyFile("reviews.json")));
76
77     //when
78     webTestClient
79         .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
80         .uri( url: "/v1/movies/{id}", movieId) capture of ?
81         .exchange() WebTestClient.ResponseSpec
82         .expectStatus() StatusAssertions
83         .is4xxClientError();
84
85 }
```



```
12 public class GlobalErrorHandler {
13
14     @ExceptionHandler(MoviesInfoClientException.class)
15     public ResponseEntity<String> handleClientException(MoviesInfoClientException exception) {
16         log.error("Exception Caught in handleClientException : {}", exception.getMessage());
17         return ResponseEntity.status(exception.getStatusCode()).body(exception.getMessage());
18     }
19 }
```

Run: MoviesControllerIntgTest.retrieveMovieById > MoviesReviewServiceApplication > MoviesControllerIntgTest.retrieveMovieById\_404()

Test	Time	Result
Test Results	1s 97ms	Exception Caught in handleClientException : [redacted]
MoviesControllerIntgTest	1s 97ms	[redacted]
retrieveMovieById_404()	1s 97ms	[redacted]



```
80
81
82     .exchange() WebTestClient.ResponseSpec
83     .expectStatus() StatusAssertions
84     .is4xxClientError() WebTestClient.ResponseSpec
85     .expectBody(String.class) WebTestClient.BodySpec<String, capture of ?>
86         .isEqualTo("There is no MovieInfo Available for the passed in Id : abc");
87
88 }
```

Add last two lines to make our test more meaningful.

```
88     @Test
89     void retrieveMovieById_reviews_404() {
90         //given
91         var movieId = "abc";
92         stubFor(get(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId))
93             .willReturn(aResponse()
94                 .withHeader( key: "Content-Type", ...values: "application/json")
95                 .withBodyFile("movieinfo.json")
96             ));
97
98         stubFor(get(urlPathEqualTo( testUrl: "/v1/reviews"))
99             .willReturn(aResponse()
100                .withStatus(404)));
101
102         //when
103         webTestClient
104             .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
105             .uri( uri: "/v1/movies/{id}", movieId) capture of ?
106             .exchange() WebTestClient.ResponseSpec
107             .expectStatus().isOk()
108             .expectBody(Movie.class) WebTestClient.BodySpec<Movie, capture of ?>
109             .consumeWith(movieEntityExchangeResult -> {
110                 var movie = movieEntityExchangeResult.getResponseBody();
111                 assert Objects.requireNonNull(movie).getReviewList().size() == 0;
112                 assertEquals( expected: "Batman Begins", movie.getMovieInfo().getName());
113             });
114
115     }

```

```
88     @Test
89     void retrieveMovieById_5XX() {
90         //given
91         var movieId = "abc";
92         stubFor(get(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId))
93             .willReturn(aResponse()
94                 .withStatus(500)
95                 .withBody("MovieInfo Service Unavailable"));
96
97         stubFor(get(urlPathEqualTo( testUrl: "/v1/reviews"))
98             .willReturn(aResponse()
99                 .withHeader( key: "Content-Type", ...values: "application/json")
100                .withBodyFile("reviews.json")));
101
102         //when
103         webTestClient
104             .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
105             .uri( uri: "/v1/movies/{id}", movieId) capture of ?
106             .exchange() WebTestClient.ResponseSpec
107             .expectStatus() StatusAssertions
108             .is5xxServerError() WebTestClient.ResponseSpec
109             .expectBody(String.class) WebTestClient.BodySpec<String, capture of ?>
110             .isEqualTo("MovieInfo Service Unavailable");
111     }

```

Test failed

```

java.lang.AssertionError: Response body expected:<MovieInfo Service Unavailable> but was:<
Expected :MovieInfo Service Unavailable
Actual   :Server Exception in MoviesInfoService MovieInfo Service Unavailable
<Click to see difference>

```

As below code execute change the value as per Actual.

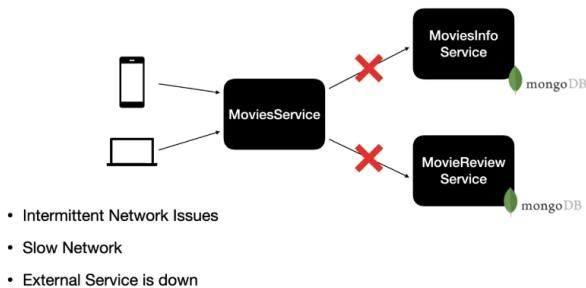
```

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

```

Retrying http calls

## Movies Application using MicroServices Pattern



## Handle Network Errors

- Retrying Failed Calls
  - Retry the failed call N number of times before giving up
  - Retry the failed call with a backoff
  - Retry Specific Exceptions
  - Retry only 5xx not 4xx exceptions

```

        }

        return clientResponse.bodyToMono(String.class)
    .retryWhen(Retry retry... Mono<MovieInfo> mono.error(r
    .retry()
    .retry()
    .retry(long numRetries) Mono<MovieInfo>
    .repeat()
    .repeat(long numRepeat) Flux<MovieInfo> ClientRespon
    .repeat(BooleanSupplier... Flux<MovieInfo> tResponse.s
    .repeat(long numRepeat...) Flux<MovieInfo> tResponse.s
    .repeatWhen(Function<F... Flux<MovieInfo> nging.class)
    .repeatWhenEmpty(Funct... Mono<MovieInfo> on.error(r
    .repeatWhenEmpty(int m... Mono<MovieInfo> MoviesInfoS
    .onErrorReturn(MovieIn... Mono<MovieInfo>
    .onErrorReturn(M... Mono<MovieInfo> on.error(MovieInfoT
    .repeat()

    .re

    .log();
}

```

```

    @Test
    void retrieveMovieById_5xx() {
        //given
        var movieId = "abc";
        stubFor(get(urlEqualTo("v1/movieinfos/" + "/" + movieId))
            .willReturn(response()
                .withStatus(500)
                .withBody("MovieInfo Service Unavailable"))
        );

        /* stubFor(get(urlPathEqualTo("/v1/reviews"))
            .willReturn(response()
                .withHeader("Content-Type", "application/json")
                .withBodyFile("reviews.json"))); */

        //when
        webTestClient
            .get()
            .andExpect(headers().contentType("application/json"))
            .andExpect(status().is(500));
    }
}

var url = moviesInfoUrl.concat("/{id}");
return webClient
    .get()
    .retrieve()
    .onStatus(HttpStatus::is5xxServerError, clientResponse -> {
        log.info("Status code is : {}", clientResponse.statusCode().value());
        if(clientResponse.statusCode().equals(HttpStatus.NOT_FOUND)){
            return Mono.error(new MoviesInfoClientException(
                "There is no MovieInfo Available for the passed in Id : " + movieId));
        }
    })
    .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)
.retry()
.log();

```

Retry will happen 3 times

```

141
142
143
144     WireMock.verify(count: 4, getRequestedFor(urlEqualTo("v1/movies/abc")));
145

```

`lEqualTo("/v1/movieinfos" + "/" + movieId));

Intentionally change to 5 and verify you get below error

```

com.github.tomakehurst.wiremock.client.VerificationException: Expected exactly 5 requests matching the following pattern but received 4:
{
  "url" : "/v1/movieinfos/abc",
  "method" : "GET"
}

```

Backoff before retry attempt

```

    .bodyToMono(MovieInfo.class) Mono<MovieInfo>
    // .retry(3)
    .retryWhen(Retry)
    .log();
}
}

```

A code editor showing a dropdown menu for the `.retryWhen(Retry)` method. The menu lists various retry strategies:

- `Retry.org.springframework.retry.Retry`
- `Retry.com.sun.net.httpserver.Authenticator`
- `Retry.fixedDelay(long maxAttempts, Duration minBackoffSpec)`
- `Retry.backoff(long maxAttempts, Duration minBackoff) (reactor.util.retry)`
- `Retry.from(Function<Flux<RetrySignal>, ? extends RetrySpec> spec)`
- `Retry.indefinitely() (reactor.util.retry)`
- `Retry.max(long max) (reactor.util.retry)`
- `Retry.maxInARow(long maxInARow) (reactor.util.retry)`
- `Retry.withThrowable(Function<Flux<Throwable>, ? extends RetrySpec> spec)`

```

    "Server Exception in MoviesInfoService " + resp
})
.bodyToMono(MovieInfo.class) Mono<MovieInfo>
// .retry(3)
.retryWhen(Retry.fixedDelay(maxAttempts: 3, Duration.ofSeconds(1)))
.log();
}
}

```

Test is failed



Test took 4 seconds

```

32
33     var retrySpec = Retry.fixedDelay(maxAttempts: 3, Duration.ofSeconds(1))
34         .onRetryExhaustedThrow((retryBackoffSpec, retrySignal) ->
35             Exceptions.propagate(retrySignal.failure()));
36
37     .bodyToMono(MovieInfo.class) Mono<MovieInfo>
38     // .retry(3)
39     .retryWhen(retrySpec)
40     .log();
41 }
}

```

Introduce filter (only for 5XX errors (server errors) not for 4XX error (client side error))

```

var retrySpec = Retry.fixedDelay(maxAttempts: 3, Duration.ofSeconds(1))
    .filter(ex -> ex instanceof MoviesInfoServerErrorException)
    .onRetryExhaustedThrow((retryBackoffSpec, retrySignal) ->
        Exceptions.propagate(retrySignal.failure())));
}
}

```



```

    //given
    var movieId = "abc";
    stubFor(get(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId))
        .willReturn(aResponse()
            .withStatus(404)
        ));

    stubFor(get(urlPathEqualTo( testUrl: "/v1/reviews"))
        .willReturn(aResponse()
            .withHeader( key: "Content-Type", ...values: "application/json")
            .withBodyFile("reviews.json")));
}

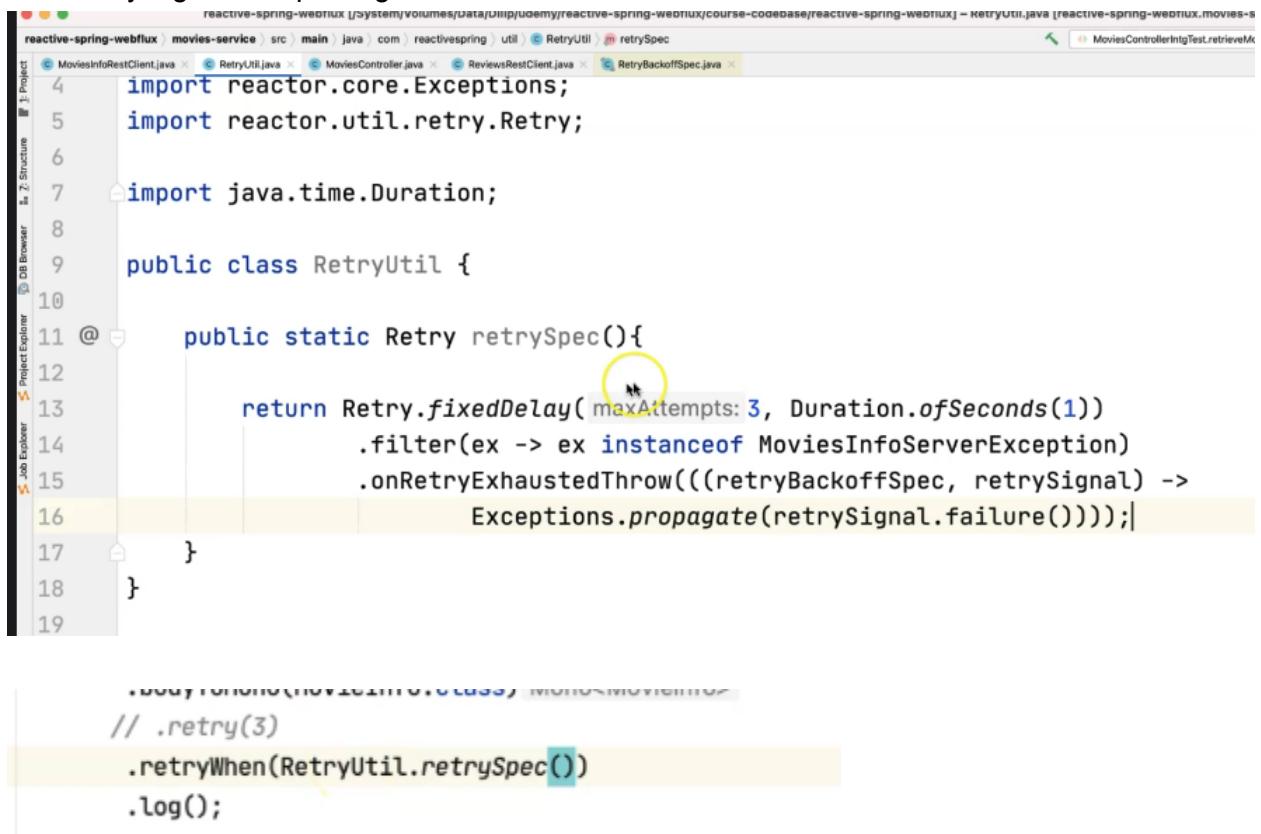
//when
webTestClient
    .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
    .uri( uri: "/v1/movies/{id}", movieId) capture of ?
    .exchange() WebTestClient.ResponseSpec
    .expectStatus() StatusAssertions
    .is4xxClientError() WebTestClient.ResponseSpec
    .expectBody(String.class) WebTestClient.BodySpec<String, capture of ?>
    .isEqualTo("There is no MovieInfo Available for the passed in Id : abc");

WireMock.verify( count: 1, getRequestedFor(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId)));
}

```

It will execute only once.

Move retry logic to util package



```

import reactor.core.Exceptions;
import reactor.util.retry.Retry;

import java.time.Duration;

public class RetryUtil {

    @SneakyThrows
    public static Retry retrySpec(){

        return Retry.fixedDelay(maxAttempts: 3, Duration.ofSeconds(1))
            .filter(ex -> ex instanceof MoviesInfoServerException)
            .onRetryExhaustedThrow(((retryBackoffSpec, retrySignal) ->
                Exceptions.propagate(retrySignal.failure())));
    }
}

```

```

// .retry(3)
.retryWhen(RetryUtil.retrySpec())
.log();

```

Same logic for review info service.

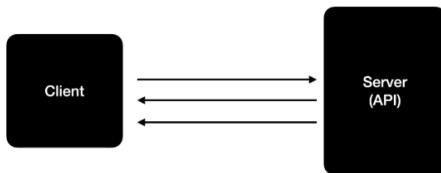
```
150
151     @Test
152     void retrieveMovieById_Reviews_5XX() {
153         //given
154         var movieId = "abc";
155
156         stubFor(get(urlEqualTo("testUrl: /v1/movieinfos" + "/" + movieId))
157                 .willReturn(aResponse()
158                         .withHeader("Content-Type", "application/json")
159                         .withBodyFile("movieinfo.json")));
160
161
162         stubFor(get(urlPathEqualTo("testUrl: /v1/reviews"))
163                 .willReturn(aResponse()
164                         .withStatus(500)
165                         .withBody("Review Service Not Available")));
166
167
168         //when
169         webTestClient
170             .get() WebClient.RequestHeadersUriSpec capture of ?
171             .uri(url: "/v1/movies/{id}", movieId) capture of ?
172             .exchange() WebTestClient.ResponseSpec
173             .expectStatus() StatusAssertions
174             .is5xxServerError() WebTestClient.ResponseSpec
175             .expectBody(String.class) WebTestClient.BodySpec<String, capture of ?>
176             .isEqualTo("Server Exception in ReviewsService Review Service Not
177
178         WireMock.verify(count: 4, getRequestedFor(urlPathMatching(urlRegex: "/v1/reviews")));
179     }
180 }
```

```
28
29
30
31     //movieInfoId
32
33     public Flux<Review> retrieveReviews(String movieId){
34
35         var url = UriComponentsBuilder.fromHttpUrl(reviewsUrl)
36             .queryParam("name", "movieInfoId", movieId)
37             .buildAndExpand().toUriString();
38
39
40         return webClient
41             .get() WebClient.RequestHeadersUriSpec<capture of ?>
42             .uri(url) capture of ?
43             .retrieve() WebClient.ResponseSpec
44             .onStatus(HttpStatus::is4xxClientError, clientResponse -> {
45                 log.info("Status code is : {}", clientResponse.statusCode().value());
46                 if(clientResponse.statusCode().equals(HttpStatus.NOT_FOUND)){
47                     return Mono.empty();
48                 }
49
50
51             return clientResponse.bodyToMono(String.class)
52                 .flatMap(responseMessage -> Mono.error(new ReviewsClientException(
53                     responseMessage)));
54
55             .onStatus(HttpStatus::is5xxServerError, clientResponse -> {
56                 log.info("Status code is : {}", clientResponse.statusCode().value());
57                 return clientResponse.bodyToMono(String.class)
58                     .flatMap(responseMessage -> Mono.error(new ReviewsServerException(
59                         "Server Exception in ReviewsService " + responseMessage)));
60
61         })
62             .bodyToFlux(Review.class) Flux<Review>
63             .retryWhen(RetryUtil.retrySpec());
64     }
65 }
```

```
ns
ns com.github.tomakehurst.wiremock.client.VerificationException: Expected exactly 4 requests matching the following pattern but received 1:
{
    "urlPathPattern" : "/v1/reviews",
    "method" : "GET"
}
```

```
9
10
11     public class RetryUtil {
12
13         public static Retry retrySpec(){
14
15             return Retry.fixedDelay(maxAttempts: 3, Duration.ofSeconds(1))
16                 .filter(ex -> ex instanceof MoviesInfoServerException ||
17                         ex instanceof ReviewsServerException)
18                 .onRetryExhaustedThrow((retryBackoffSpec, retrySignal) ->
19                     Exceptions.propagate(retrySignal.failure()));
20
21     }
22 }
```

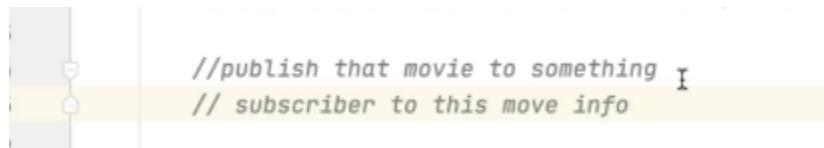
## Server Sent Events



- Uber App - Realtime updates of the driver location
- Data is sent in the form of events
- Dominos, DoorDash etc.
- Its Unidirectional once the connection is established between the client and server
- Long live client connections

The screenshot shows the Reactor 3 Reference Guide homepage. The title "Reactor 3 Reference Guide" is at the top. Below it, there's a brief introduction and credits. The main content area has a heading "Sinks" which is currently highlighted. A sub-section titled "Sinks API" is expanded, showing code examples and descriptions for different sink types like Sinks.Many, Sinks.Publisher, and Sinks.BiFunction.

<https://projectreactor.io/docs/core/release/reference/#sinks>



Sinks.One means produce 1 event. Sinks.two means produces many events

The `Sinks` builder provide a guided API to the main supported producer types. You will recognize some of the behavior found in `Flux` such as `onBackpressureBuffer`.

```
Sinks.Many<Integer> replaySink = Sinks.many().replay().all();
```

Multiple producer threads may concurrently generate data on the sink by doing the following:

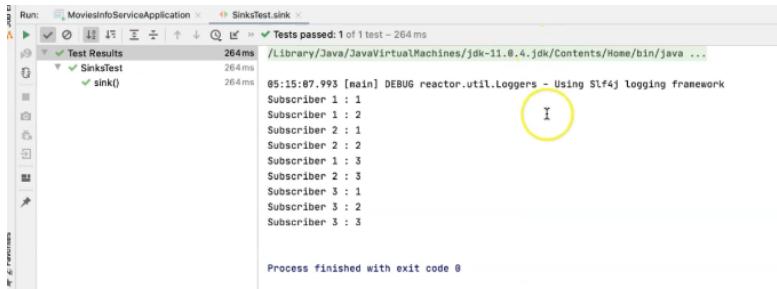
```
//thread1
sink.emitNext(1, FAIL_FAST);
//thread2, later
sink.emitNext(2, FAIL_FAST);
//threads, concurrently with thread 2
EmitResult result = sink.tryEmitNext(3); //would return FAIL_NON_SERIALIZED
```

The `Sinks.Many` can be presented to downstream consumers as a `Flux`, like in the below example:

```
Flux<Integer> fluxView = replaySink.asFlux();
fluxView
    .takeWhile(i -> i < 10)
    .log()
    .blockLast();
```

Replay in line number 12 means for any new subscriber to replay all the events.

```
9      @Test
10     void sink() {
11         //given
12         Sinks.Many<Integer> replaySink = Sinks.many().replay().all();
13
14         //when
15
16         replaySink.emitNext(t: 1, Sinks.EmitFailureHandler.FAIL_FAST);
17         replaySink.emitNext(t: 2, Sinks.EmitFailureHandler.FAIL_FAST);
18
19         //then
20         Flux<Integer> integerFlux = replaySink.asFlux();
21         integerFlux.subscribe((i)->{
22             System.out.println("Subscriber 1 : " + i);
23         });
24
25         Flux<Integer> integerFlux1 = replaySink.asFlux();
26         integerFlux1.subscribe((i)->{
27             System.out.println("Subscriber 2 : " + i);
28         });
29
30     });
31
32     Flux<Integer> integerFlux1 = replaySink.asFlux();
33     integerFlux1.subscribe((i)->{
34         System.out.println("Subscriber 2 : " + i);
35     });
36
37     replaySink.tryEmitNext(t: 3);
38
39     Flux<Integer> integerFlux2 = replaySink.asFlux();
40     integerFlux2.subscribe((i)->{
41         System.out.println("Subscriber 3 : " + i);
42     });
43 }
```



```

59
60 @Test
61 void sinks_multicast() {
62     //given
63
64     Sinks.Many<Integer> multicast = Sinks.many().multicast().onBackpressureBuffer();
65
66     //when
67     multicast.emitNext(t:1, Sinks.EmitFailureHandler.FAIL_FAST);
68     multicast.emitNext(t:2, Sinks.EmitFailureHandler.FAIL_FAST);
69
70     //then
71     Flux<Integer> integerFlux = multicast.asFlux();
72     integerFlux.subscribe((i)->{
73         System.out.println("Subscriber 1 : " + i);
74     });
75
76     Flux<Integer> integerFlux1 = multicast.asFlux();
77     integerFlux1.subscribe((i)->{
78         System.out.println("Subscriber 2 : " + i);
79     });
80     multicast.emitNext(t:3, Sinks.EmitFailureHandler.FAIL_FAST);
81 }

```



Notice the subscriber 2 has received only newly published value not old value - because of multicast.

```

65
66 void sinks_unicast() {
67     //given
68
69     Sinks.Many<Integer> multicast = Sinks.many().unicast().onBackpressureBuffer();
70
71     //when
72     multicast.emitNext(t:1, Sinks.EmitFailureHandler.FAIL_FAST);
73     multicast.emitNext(t:2, Sinks.EmitFailureHandler.FAIL_FAST);
74
75     //then
76     Flux<Integer> integerFlux = multicast.asFlux();
77     integerFlux.subscribe((i)->{
78         System.out.println("Subscriber 1 : " + i);
79     });
80
81     Flux<Integer> integerFlux1 = multicast.asFlux();
82     integerFlux1.subscribe((i)->{
83         System.out.println("Subscriber 2 : " + i);
84     });
85     multicast.emitNext(t:3, Sinks.EmitFailureHandler.FAIL_FAST);
86 }

```

Unitcast will take only one subscriber. If we give two subscribers we will get an error as follows.

```

15 .727 [main] DEBUG reactor.util.Loggers - Using Slf4j logging framework
16 er 1 : 1
17 er 1 : 2
18 .750 [main] ERROR reactor.core.publisher.Operators - Operator called default onErrorDropped
19 core.Exceptions$ErrorCallbackNotImplemented: java.lang.IllegalStateException: UnicastProcessor allows only a single Subscriber
20 y: java.lang.IllegalStateException: UnicastProcessor allows only a single Subscriber

```

Same thing in the movie service. The post method is publisher and get methods consumer

```

10 @RestController
11
12 @RequestMapping("/v1")
13 @Slf4j
14
15 public class MoviesInfoController {
16
17     private MoviesInfoService moviesInfoService;
18
19     Sinks.Many<MovieInfo> moviesInfoSink = Sinks.many().replay().all();
20
21 }

```

```

53     @GetMapping(value = "/movieinfos/stream", produces = MediaType.APPLICATION_NDJSON_VALUE)
54     public Flux<MovieInfo> getMovieInfoById(){
55         return moviesInfoSink.asFlux();
56     }
57
58     @PostMapping("/movieinfos")
59     @ResponseStatus(HttpStatus.CREATED)
60     public Mono<MovieInfo> addMovieInfo(@RequestBody @Valid MovieInfo movieInfo){
61         return moviesInfoService.addMovieInfo(movieInfo)
62             .doOnNext(savedInfo-> moviesInfoSink.tryEmitNext(savedInfo));
63
64     //publish that movie to something

```

This will work as streaming. Server sent event work as real time live data. Like uber car location or cricket score.

We have so many options

```

Sinks.Many<MovieInfo> moviesInfoSink = Sinks.many().replay(). I
public MoviesInfoController(MoviesInfoService moviesInfoService) {
    moviesInfoSink = moviesInfoSink.replay(1).latest();
}

```

Replay Strategies:

- all()
- all(int batchSize)
- latest()
- latestOrDefault(T value)
- limit(Duration maxAge)
- limit(int historySize)
- limit(int historySize, Duration maxAge)
- limit(Duration maxAge, Schedule<T> clock)
- limit(int historySize, Duration maxAge, Schedule<T> clock)
- equals(Object obj)
- hashCode()
- +toString()

Additional Information:

- ^↓ and ^↑ will move caret down and up in the editor
- Next Tip

```

78     @Test
79     void getAllMovieInfos_stream() {
80
81         var movieInfo = new MovieInfo( movieinfoId: null, name: "Batman Begins",
82             year: 2005, List.of("Christian Bale", "Michael Caine"), Loc
83
84         //when
85         webTestClient
86             .post() WebTestClient.RequestBodyUriSpec
87             .uri(MOVIES_INFO_URL) WebTestClient.RequestBodySpec
88             .bodyValue(movieInfo) WebTestClient.RequestHeadersSpec<capture
89             .exchange() WebTestClient.ResponseSpec
90             .expectStatus() StatusAssertions
91             .isCreated() WebTestClient.ResponseSpec
92
93         .expectStatus() StatusAssertions
94         .isCreated() WebTestClient.ResponseSpec
95         .expectBody(MovieInfo.class) WebTestClient.BodySpec<MovieInfo>
96         .consumeWith((movieInfoEntityExchangeResult -> {
97
98             var savedMovieInfo = movieInfoEntityExchangeResult.get
99             assert savedMovieInfo!=null;
100            assert savedMovieInfo.getMovieInfoId()!=null;
101        });
102
103        var moviesStreamFlux = webTestClient
104
105        var moviesStreamFlux = webTestClient
106            .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
107            .uri( uri: MOVIES_INFO_URL+ "/stream" ) capture of ?
108            .exchange() WebTestClient.ResponseSpec
109            .expectStatus() StatusAssertions
110            .is2xxSuccessful() WebTestClient.ResponseSpec
111            .returnResult(MovieInfo.class) FluxExchangeResult<MovieInfo>
112            .getResponseBody();
113
114            StepVerifier.create(moviesStreamFlux) StepVerifier.FirstStep<MovieInfo>
115                .assertNext(movieInfo1 -> {
116                    assert movieInfo1.getMovieInfoId()!=null;
117                } StepVerifier.Step<MovieInfo>
118                .thenCancel() StepVerifier
119                .verify();
120
121        StepVerifier.create(moviesStreamFlux) StepVerifier.FirstStep<MovieInfo>
122            .assertNext(movieInfo1 -> {
123                assert movieInfo1.getMovieInfoId()!=null;
124            } StepVerifier.Step<MovieInfo>
125            .thenCancel() StepVerifier
126            .verify();
127
128        .log();
129
130    }
131
132    @GetMapping(value = "/movieinfos/stream", produces = MediaType.APPLICATION_NOJSON_VALUE)
133    public Flux<MovieInfo> getMovieInfoById(){
134        return moviesInfoSink.asFlux()
135            .log();
136    }
137
138    @PostMapping("/movieinfos")
139    @ResponseStatus(HttpStatus.CREATED)
140    public Mono<MovieInfo> addMovieInfo(@RequestBody @Valid MovieInfo movieInfo){
141        return moviesInfoService.addMovieInfo(movieInfo)
142            .doOnNext(savedInfo-> moviesInfoSink.tryEmitNext(savedInfo));
143
144        //publish that movie to something
145        // subscriber to this movie info
146    }

```

Then cancel is for to cancel the stream and verify. (last two lines)

Same thing in the functional web for review info service

```

@Configuration
public class ReviewRouter {

    @Bean
    public RouterFunction<ServerResponse> reviewsRoute(ReviewHandler reviewHandler) {

        return route()
            .nest(path: "/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: "/stream", request -> reviewHandler.getReviewsStream(request));
            }
            .GET(pattern: "/v1/helloworld", (request -> ServerResponse.ok().bodyValue("helloworld")))
            /*.POST("/v1/reviews",request -> reviewHandler.addReview(request))
            .GET("/v1/reviews",request -> reviewHandler.getReviews(request))*/
            .build();
    }
}

```

```
54
55 @
56
57     public Mono<ServerResponse> addReview(ServerRequest request) {
58
59         return request.bodyToMono(Review.class)
60             .doOnNext(this::validate)
61             .flatMap(reviewReactiveRepository::save)
62             .doOnNext(review -> {
63                 reviewsSink.tryEmitNext(review);
64             })
65             .flatMap(ServerResponse.status(HttpStatus.CREATED)::bodyValue);
66
67 }
```

```
113
114     public Mono<ServerResponse> getReviewsStream(ServerRequest request) {
115
116         return ServerResponse.ok()
117             .contentType(MediaType.APPLICATION_NDJSON)
118             .body(reviewsSink.asFlux(), Review.class)
119             .log();
120
121 }
```

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

HTTP/1.1 200 OK
transfer-encoding: chunked
Content-Type: application/x-ndjson

{"reviewId": "1", "movieInfoId": 1, "comment": "Excellent Movie", "rating": 8.0}
{"reviewId": "2", "movieInfoId": 2, "comment": "Excellent Movie", "rating": 8.0}
```

In movie service which is depending on movie info and review info services

```
59
60     @GetMapping(value = "/stream", produces = MediaType.APPLICATION_NDJSON_VALUE)
61     public Flux<MovieInfo> retrieveMovieInfos() {
62
63         return moviesInfoRestClient.retrieveMovieInfoStream();
64
65     }
66
67 }
```

The screenshot shows a Java code editor with the following code:

```
public Flux<MovieInfo> retrieveMovieInfoStream() {  
    var url = moviesInfoUrl.concat("/stream");  
  
    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());  
  
            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 MoviesInfoServerErrorException(  
                    "Server Exception in MoviesInfoService " + responseMessage)));  
        })  
        .bodyToFlux(MovieInfo.class) Flux<MovieInfo>  
        // .retry(3)
```

The code editor highlights the line ".bodyToFlux(MovieInfo.class)" in green, indicating it's a valid suggestion. The line ".retry(3)" is also highlighted in green. The status bar at the bottom shows "Build completed successfully in 2 s 655 ms (20 minutes ago)".

The screenshot shows the same Java code as above, but with several lines underlined in red, indicating errors or warnings:

```
88  
89     .bodyToFlux(MovieInfo.class) Flux<MovieInfo>  
90     // .retry(3)  
91     .retryWhen(RetryUtil.retrySpec())  
92     .log();  
93  
94
```

The lines ".bodyToFlux(MovieInfo.class)", ".retry(3)", ".retryWhen(RetryUtil.retrySpec())", and ".log();" are all underlined in red.