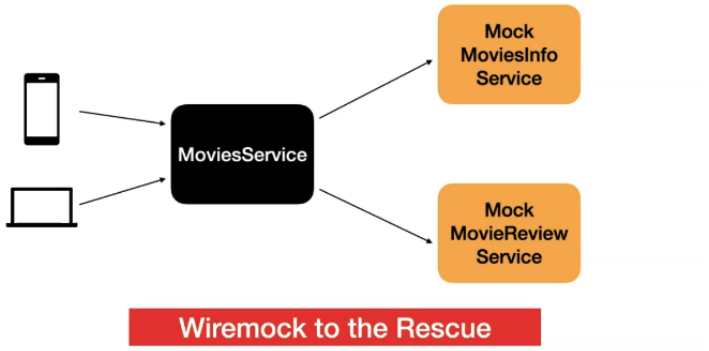
This document contains continuation wiremock, retrying http calls , server sent events (SSE).



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

```java
package com.reactivespring.controller;

import org.springframework.boot.test.autoconfigure.web.reactive.AutoConfigureWebTestClient;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.cloud.contract.wiremock.AutoConfigureWireMock;
import org.springframework.test.context.ActiveProfiles;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
@AutoConfigureWebTestClient
@AutoConfigureWireMock(port = 8084)
public class MoviesControllerIntgTest {
}
```

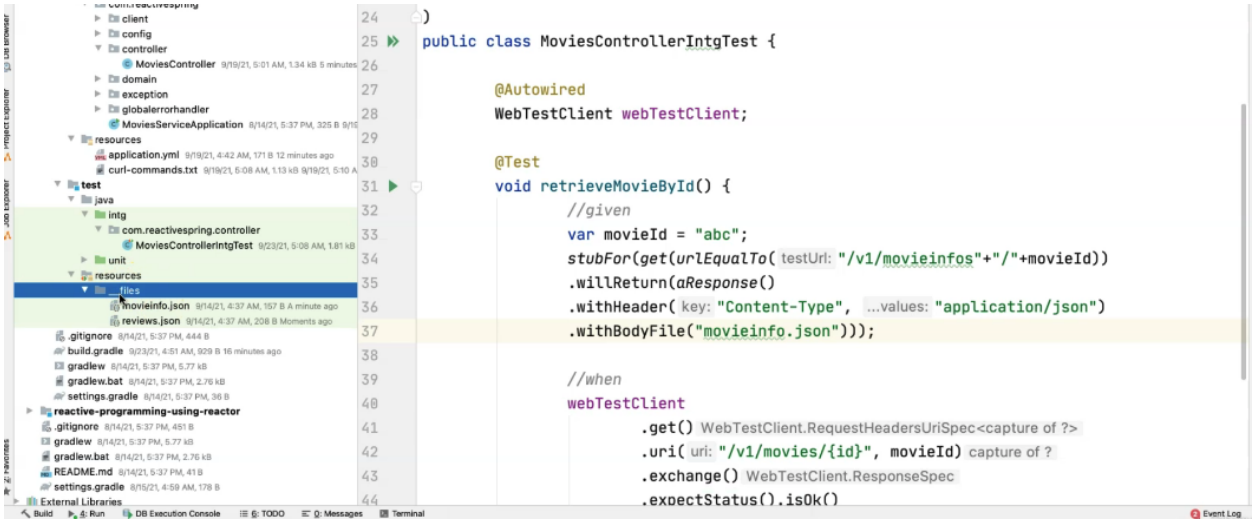
The line number 11 @AutoConfigureWireMock will spin up the http server and give a response to us.

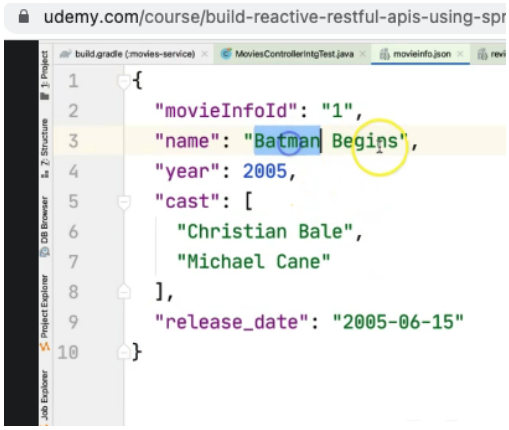
Because our wiremock server is on 8084 we have to introduce test properties notice the difference between left values and right values (we are overriding)

```java
package com.reactivespring.controller;

import org.springframework.boot.test.autoconfigure.web.reactive.AutoConfig
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.cloud.contract.wiremock.AutoConfigureWireMock;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.TestPropertySource;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT
@ActiveProfiles("test")
@AutoConfigureWebTestClient
@AutoConfigureWireMock(port = 8084)
@TestPropertySource(
        properties = {
                "restClient.moviesInfoUrl=http://localhost:8084/v1/movieIn
                "restClient.reviewsUrl=http://localhost:8084/v1/reviews"
        }
)
public class MoviesControllerIntgTest {
}
```

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

Compare wiremock syntax below

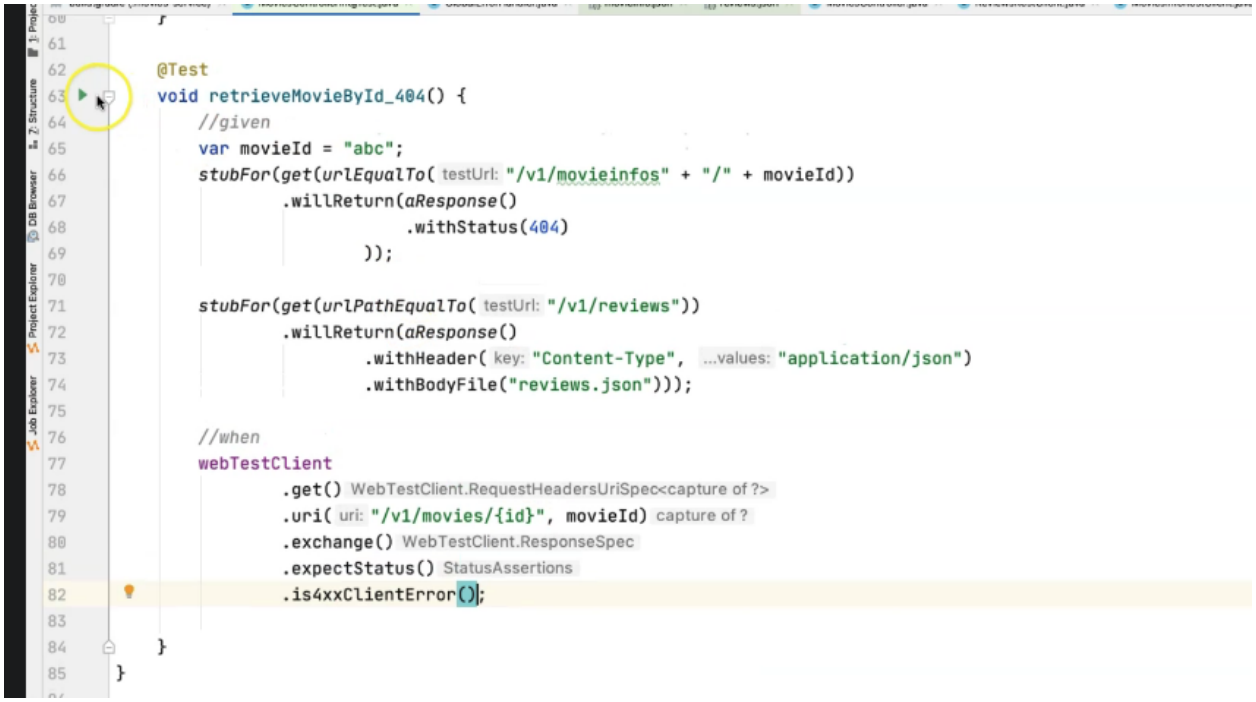




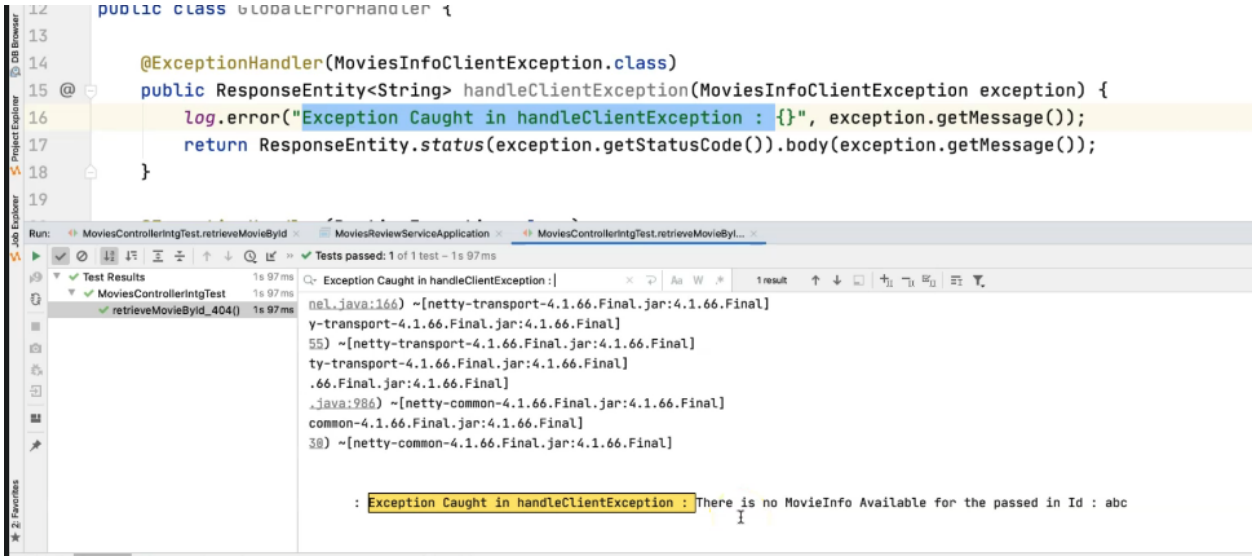
```
//given
var movieId = "abc";
stubFor(get(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId))
        .willReturn(aResponse()
                .withHeader( key: "Content-Type",  ...values: "application/json")
                .withBodyFile("movieinfo.json")));

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().isOk()
        .expectBody(Movie.class) WebTestClient.BodySpec<Movie, capture of ?>
        .consumeWith(movieEntityExchangeResult -> {
            var movie = movieEntityExchangeResult.getResponseBody();
            assert Objects.requireNonNull(movie).getReviewList().size() ==2;
            assertEquals( expected: "Batman Begins", movie.getMovieInfo().getName());
        });
```

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

Wiremock will give lot of logs to understand

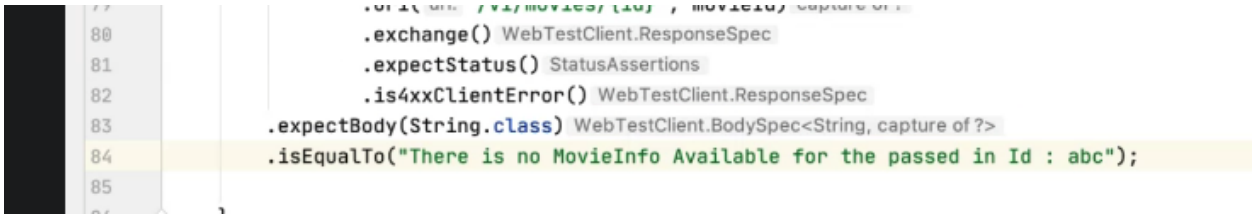


```
61
62      @Test
63      void retrieveMovieById_404() {
64          //given
65          var movieId = "abc";
66          stubFor(get(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId))
67                  .willReturn(aResponse()
68                          .withStatus(404)
69                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



```
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());
    }
```

```
Run: MoviesControllerIntgTest.retrieveMovieById  MoviesReviewServiceApplication  MoviesControllerIntgTest.retrieveMovieByI...
Tests passed: 1 of 1 test – 1s 97ms
Test Results 1s 97ms
MoviesControllerIntgTest 1s 97ms
retrieveMovieById_404() 1s 97ms
Exception Caught in handleClientException :
1 result
nel.java:166) ~[netty-transport-4.1.66.Final.jar:4.1.66.Final]
y-transport-4.1.66.Final.jar:4.1.66.Final]
55) ~[netty-transport-4.1.66.Final.jar:4.1.66.Final]
ty-transport-4.1.66.Final.jar:4.1.66.Final]
.66.Final.jar:4.1.66.Final]
.java:986) ~[netty-common-4.1.66.Final.jar:4.1.66.Final]
common-4.1.66.Final.jar:4.1.66.Final]
30) ~[netty-common-4.1.66.Final.jar:4.1.66.Final]

: Exception Caught in handleClientException : There is no MovieInfo Available for the passed in Id : abc
```



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

Add last two lines to make our test more meaningful.

```java
@Test
void retrieveMovieById_reviews_404() {
    //given
    var movieId = "abc";
    stubFor(get(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId))
            .willReturn(aResponse()
                    .withHeader( key: "Content-Type",  ...values: "application/json")
                    .withBodyFile("movieinfo.json")
            ));

    stubFor(get(urlPathEqualTo( testUrl: "/v1/reviews"))
            .willReturn(aResponse()
                    .withStatus(404)));

    //when
    webTestClient
            .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
            .uri( uri: "/v1/movies/{id}", movieId) capture of ?
            .exchange() WebTestClient.ResponseSpec
            .expectStatus().isOk()
            .expectBody(Movie.class) WebTestClient.BodySpec<Movie, capture of ?>
            .consumeWith(movieEntityExchangeResult -> {
                var movie = movieEntityExchangeResult.getResponseBody();
                assert Objects.requireNonNull(movie).getReviewList().size() == 0;
                assertEquals( expected: "Batman Begins", movie.getMovieInfo().getName());
            });;

}
```

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

```java
@Test
void retrieveMovieById_5XX() {
    //given
    var movieId = "abc";
    stubFor(get(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId))
            .willReturn(aResponse()
                    .withStatus(500)
                    .withBody("MovieInfo Service Unavailable")

            ));

    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
            .is5xxServerError() WebTestClient.ResponseSpec
            .expectBody(String.class) WebTestClient.BodySpec<String, capture of ?>
            .isEqualTo("MovieInfo Service Unavailable");

}
```

Test failed

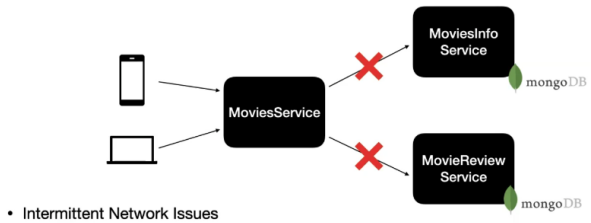
```
java.lang.AssertionError: Response body expected:<MovieInfo Service Unavailable> but was:<
Expected :MovieInfo Service Unavailable
Actual   :Server Exception in MoviesInfoService MovieInfo Service Unavailable
<Click to see difference>
```

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 MoviesInfoServerException(
                    "Server Exception in MoviesInfoService " + responseMessage)));
})
```

Retrying http calls

Movies Application using MicroServices Pattern

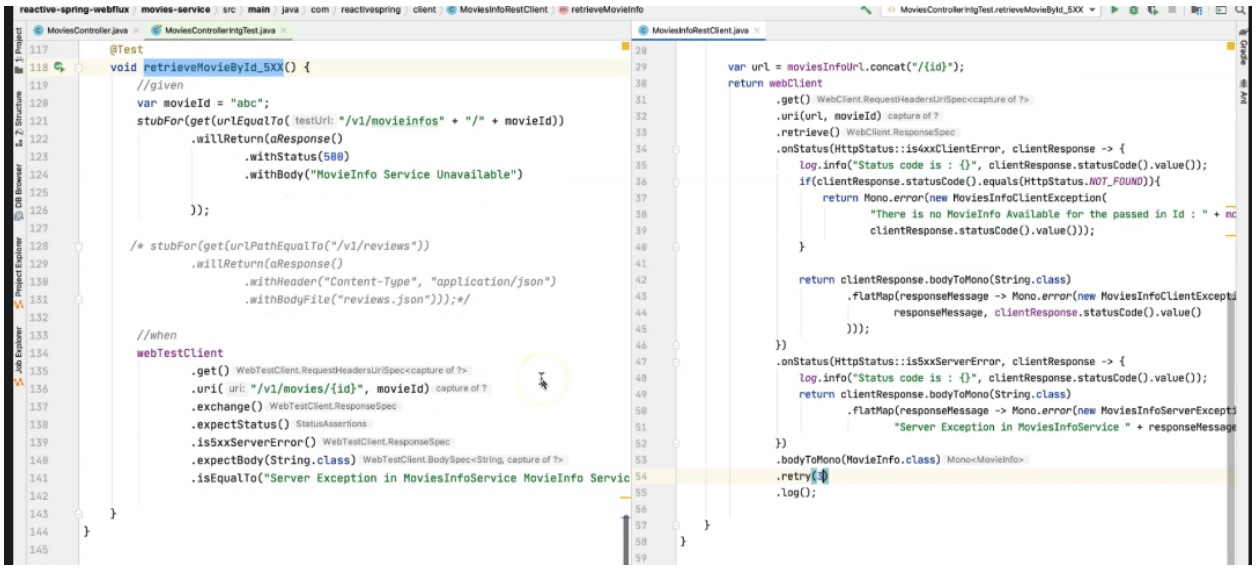


- Intermittent Network Issues
- Slow Network
- External Service is down

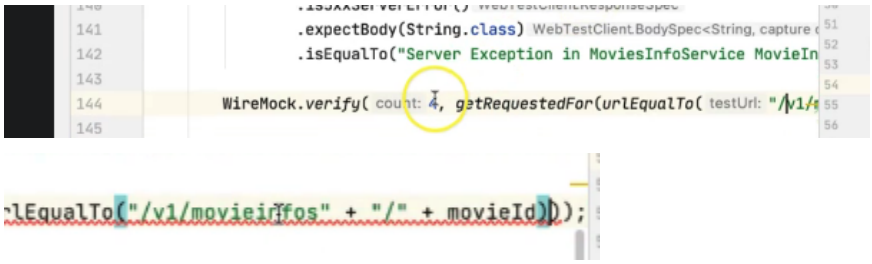
Handle Network Errors

- 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>
                retry()                  Mono<MovieInfo>
                retry(long numRetries)   Mono<MovieInfo>
                repeat()                 Flux<MovieInfo>
                repeat(long numRepeat)   Flux<MovieInfo>
                repeat(BooleanSupplie…   Flux<MovieInfo>
                repeat(long numRepeat…   Flux<MovieInfo>
                repeatWhen(Function<F…   Flux<MovieInfo>
                repeatWhenEmpty(Funct…   Mono<MovieInfo>
                repeatWhenEmpty(int m…   Mono<MovieInfo>
                onErrorReturn(MovieIn…   Mono<MovieInfo>
                .re
                .log();

}
```



Retry will happen 3 times



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();
    }
}
```

Retry com.sun.net.httpserver.Authenticator
Retry.fixedDelay(long maxAttempts, D… RetryBackoffSpec
Retry.backoff(long maxAttempts, Duration minBackoff) (reactor.u
Retry.from(Function<Flux<RetrySignal>, ? extend… Retry
Retry.indefinitely() (reactor.util.retry) RetrySpec
Retry.max(long max) (reactor.util.retry) RetrySpec
Retry.maxInARow(long maxInARow) (reactor.ut… RetrySpec
Retry.withThrowable(Function<Flux<Throwable>, ?… Retry

```
ison")                                                    "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
```

```
        .bodyToMono(MovieInfo.class) Mono<MovieIn
        // .retry(3)
        .retryWhen(retrySpec)
        .log();
```

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 MoviesInfoServerException)
        .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" + "/" +
```

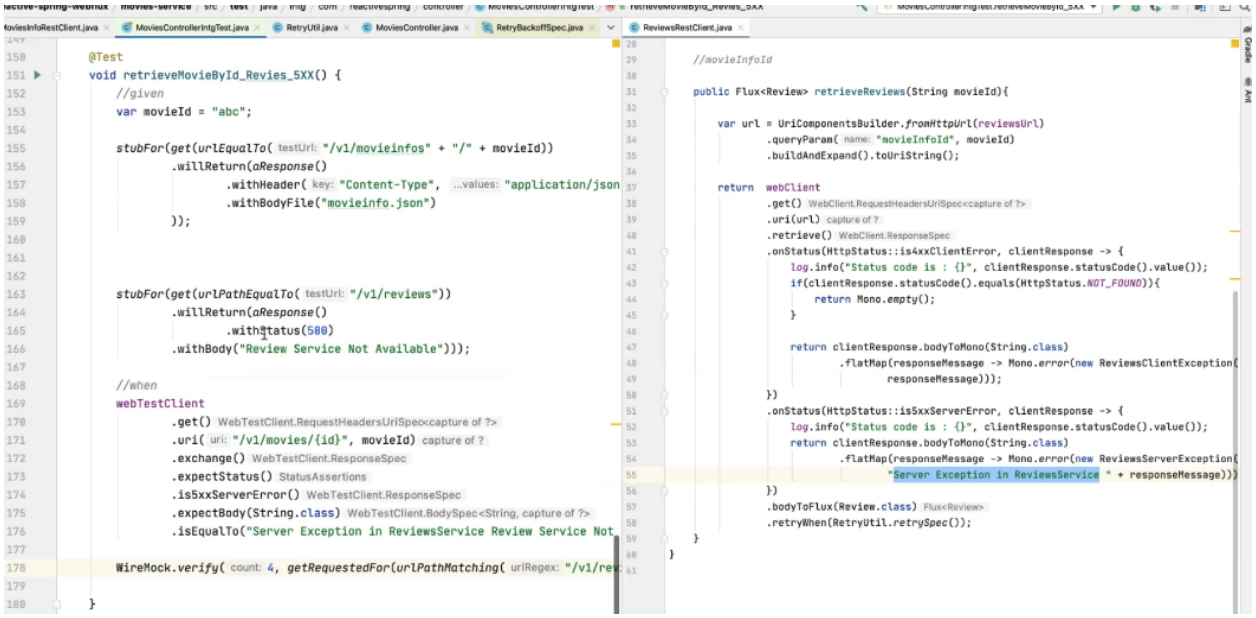
It will execute only once.

Move retry logic to util package

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

import java.time.Duration;

public class RetryUtil {

    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.



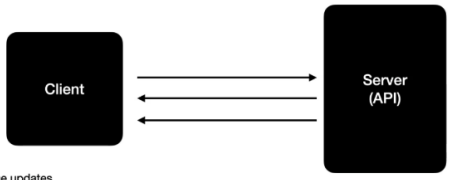
```
@Test
void retrieveMovieById_Revies_5XX() {
    //given
    var movieId = "abc";

    stubFor(get(urlEqualTo( testUrl: "/v1/movieinfos" + "/" + movieId))
            .willReturn(aResponse()
                    .withHeader( key: "Content-Type", ...values: "application/json")
                    .withBodyFile("movieinfo.json")
            ));


    stubFor(get(urlPathEqualTo( testUrl: "/v1/reviews"))
            .willReturn(aResponse()
                    .withStatus(500)
            .withBody("Review Service Not Available")));

    //when
    webTestClient
            .get() WebTestClient.RequestHeadersUriSpec<capture of ?>
            .uri( uri: "/v1/movies/{id}", movieId) capture of ?
            .exchange() WebTestClient.ResponseSpec
            .expectStatus() StatusAssertions
            .is5xxServerError() WebTestClient.ResponseSpec
            .expectBody(String.class) WebTestClient.BodySpec<String, capture of ?>
            .isEqualTo("Server Exception in ReviewsService Review Service Not

    WireMock.verify( count: 4, getRequestedFor(urlPathMatching( urlRegex: "/v1/rev

}
```

```
//movieInfoId

public Flux<Review> retrieveReviews(String movieId){

    var url = UriComponentsBuilder.fromHttpUrl(reviewsUrl)
            .queryParam( name: "movieInfoId", movieId)
            .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) Flux<Review>
            .retryWhen(RetryUtil.retrySpec());
}
}
```

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

```
public class RetryUtil {

    public static Retry retrySpec(){

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

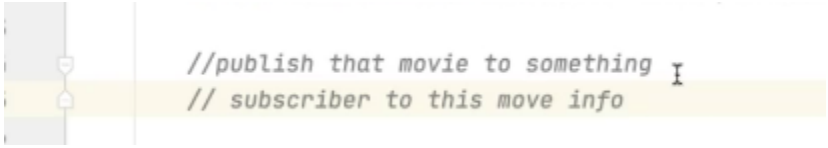
Sever Sent Events



- Uber App - Realtime updates of the driver location
- Dominos, DoorDash etc.

- Data is sent in the form of events
- Its Unidirectional once the connection is established between the client and server
- Long live client connections

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



```
//publish that movie to something
// subscriber to this move info
```

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);

//thread3, 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.

```java
@Test
void sink() {
    //given
    Sinks.Many<Integer> replaySink = Sinks.many().replay().all();


    //when

    replaySink.emitNext( t: 1, Sinks.EmitFailureHandler.FAIL_FAST);
    replaySink.emitNext( t: 2, Sinks.EmitFailureHandler.FAIL_FAST);

    //then
    Flux<Integer> integerFlux =replaySink.asFlux();
    integerFlux.subscribe((i)->{
        System.out.println("Subscriber 1 : " + i);
    });

    Flux<Integer> integerFlux1 =replaySink.asFlux();
    integerFlux1.subscribe((i)->{
        System.out.println("Subscriber 2 : " + i);
    });
```

```java
    });

    Flux<Integer> integerFlux1 =replaySink.asFlux();
    integerFlux1.subscribe((i)->{
        System.out.println("Subscriber 2 : " + i);
    });

    replaySink.tryEmitNext( t: 3);

    Flux<Integer> integerFlux2 =replaySink.asFlux();
    integerFlux2.subscribe((i)->{
        System.out.println("Subscriber 3 : " + i);
    });
}
```

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

    Sinks.Many<Integer> multicast = Sinks.many().multicast().onBackpressureBuffer();

    //when
    multicast.emitNext( t: 1, Sinks.EmitFailureHandler.FAIL_FAST);
    multicast.emitNext( t: 2, Sinks.EmitFailureHandler.FAIL_FAST);

    //then
    Flux<Integer> integerFlux =multicast.asFlux();
    integerFlux.subscribe((i)->{
        System.out.println("Subscriber 1 : " + i);
    });

    Flux<Integer> integerFlux1 =multicast.asFlux();
    integerFlux1.subscribe((i)->{
        System.out.println("Subscriber 2 : " + i);
    });
    multicast.emitNext( t: 3, Sinks.EmitFailureHandler.FAIL_FAST);
}
```

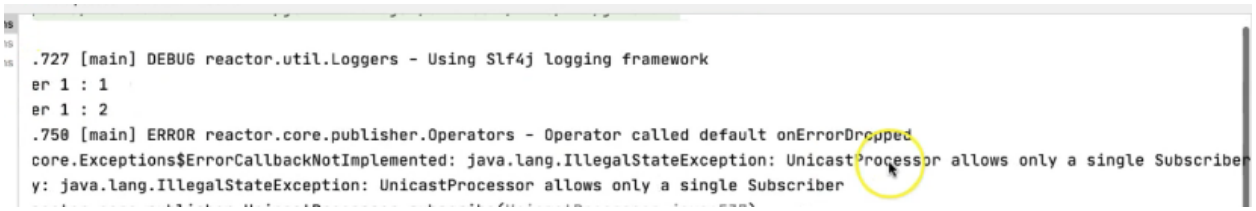
```java
    Flux<Integer> integerFlux1 =multicast.asFlux();
    integerFlux1.subscribe((i)->{
        System.out.println("Subscriber 2 : " + i);
    });
    multicast.emitNext( t: 3, Sinks.EmitFailureHandler.FAIL_FAST);
  }
}
```

```
Run: MoviesInfoServiceApplication  SinksTest.sinks_multicast
Tests passed: 1 of 1 test – 243 ms
Test Results 243 ms
SinksTest 243 ms
sinks_multicast() 243 ms
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...

05:18:50.906 [main] DEBUG reactor.util.Loggers - Using Slf4j logging framework
Subscriber 1 : 1
Subscriber 1 : 2
Subscriber 1 : 3
Subscriber 2 : 3
```

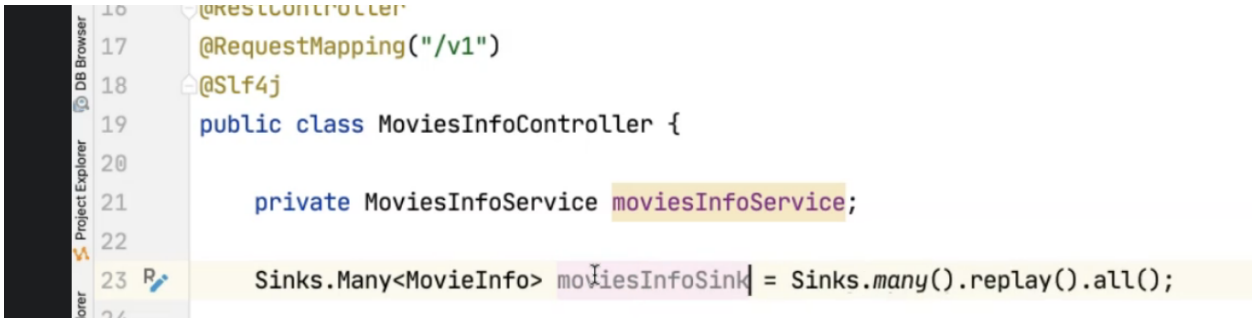
Notice the subscriber 2 has received only newly published value not old value - because of multicast.

```java
void sinks_unicast() {
    //given

    Sinks.Many<Integer> multicast = Sinks.many().unicast().onBackpressureBuffer();

    //when
    multicast.emitNext( t: 1, Sinks.EmitFailureHandler.FAIL_FAST);
    multicast.emitNext( t: 2, Sinks.EmitFailureHandler.FAIL_FAST);

    //then
    Flux<Integer> integerFlux =multicast.asFlux();
    integerFlux.subscribe((i)->{
        System.out.println("Subscriber 1 : " + i);
    });

    Flux<Integer> integerFlux1 =multicast.asFlux();
    integerFlux1.subscribe((i)->{
        System.out.println("Subscriber 2 : " + i);
    });
    multicast.emitNext( t: 3, Sinks.EmitFailureHandler.FAIL_FAST);
  }
}
```

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

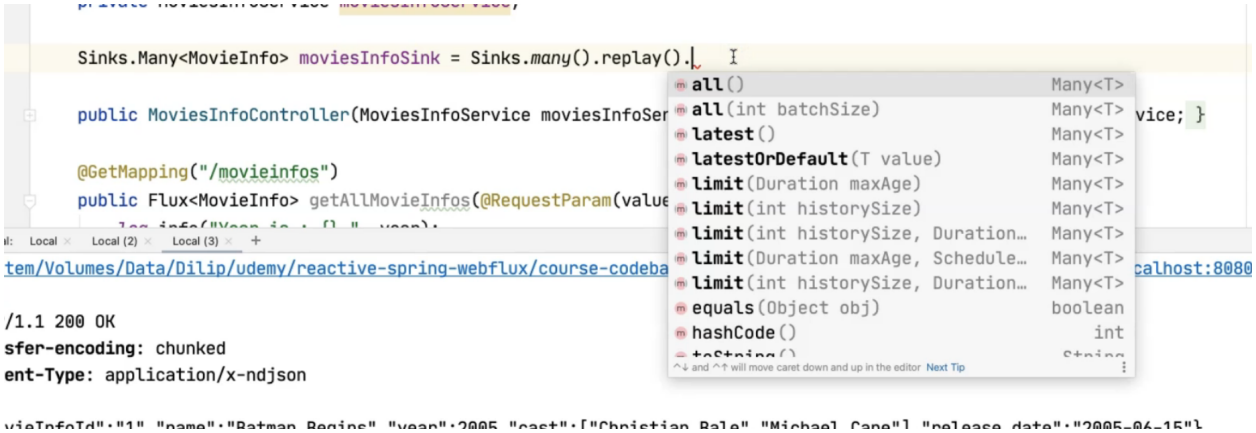


Same thing in the movie service. The post method is publisher and get methods consumer



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

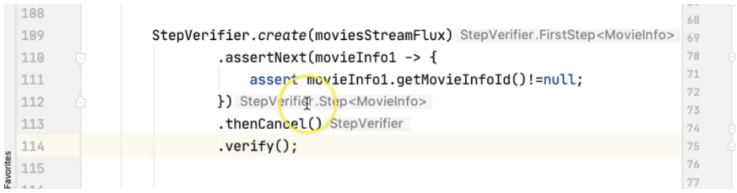


```
@Test
void getAllMovieInfos_stream() {

    var movieInfo = new MovieInfo( movieInfoId: null,  name: "Batman Begi
            year: 2005, List.of("Christian Bale", "Michael Cane"), Loc

    //when
    webTestClient
            .post() WebTestClient.RequestBodyUriSpec
            .uri(MOVIES_INFO_URL) WebTestClient.RequestBodySpec
            .bodyValue(movieInfo) WebTestClient.RequestHeadersSpec<capture
            .exchange() WebTestClient.ResponseSpec
            .expectStatus() StatusAssertions
            .isCreated() WebTestClient.ResponseSpec
```

```
            .log();
    }

    @GetMapping(value = "/movieinfos/stream", produces = MediaType.APPLICATION_NDJSON_VALUE)
    public Flux<MovieInfo> getMovieInfoById(){
        return moviesInfoSink.asFlux()
                .log();
    }

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

        //publish that movie to something
        // subscriber to this move info
```

```
            .expectStatus() StatusAssertions
            .isCreated() WebTestClient.ResponseSpec
            .expectBody(MovieInfo.class) WebTestClient.BodySpec<MovieInfo,
            .consumeWith(movieInfoEntityExchangeResult -> {

                var savedMovieInfo = movieInfoEntityExchangeResult.get
                assert savedMovieInfo!=null;
                assert savedMovieInfo.getMovieInfoId()!=null;
            });

        var moviesStreamFlux = webTestClient
```

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

    StepVerifier.create(moviesStreamFlux) StepVerifier.FirstStep<MovieInfo>
            .assertNext(movieInfo1 -> {
                assert movieInfo1.getMovieInfoId()!=null;
            }) StepVerifier.Step<MovieInfo>
```



```
    StepVerifier.create(moviesStreamFlux) StepVerifier.FirstStep<MovieInfo>
            .assertNext(movieInfo1 -> {
                assert movieInfo1.getMovieInfoId()!=null;
            }) StepVerifier.Step<MovieInfo>
            .thenCancel() StepVerifier
            .verify();
```

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( 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: "/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();

    }
}
```

```
34
35 @    public Mono<ServerResponse> addReview(ServerRequest request) {
36
37         return request.bodyToMono(Review.class)
38                 .doOnNext(this::validate)
39                 .flatMap(reviewReactiveRepository::save)
40                 .doOnNext(review -> {
41                     reviewsSink.tryEmitNext(review);
42                 })
43                 .flatMap(ServerResponse.status(HttpStatus.CREATED)::bodyValue);
44
45     }
46
```

```
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}
```

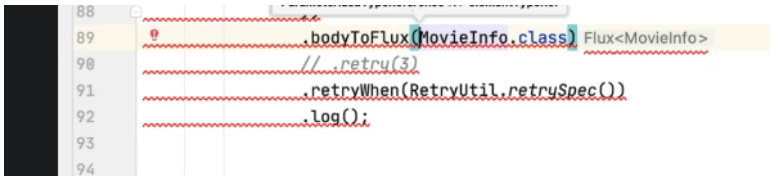
In movie service which is depending on movie info and review info services

```
39
40     @GetMapping(value = "/stream", produces = MediaType.APPLICATION_NDJSON_VALUE)
41     public Flux<MovieInfo> retrieveMovieInfos() {
42
43         return moviesInfoRestClient.retrieveMovieInfoStream();
44
45     }
46 }
47
```

```
66
67  public Flux<MovieInfo> retrieveMovieInfoStream() {
68
69      var url = moviesInfoUrl.concat("/stream");
70
71      return webClient
72              .get() WebClient.RequestHeadersUriSpec<capture of ?>
73              .uri(url) capture of ?
74              .retrieve() WebClient.ResponseSpec
75              .onStatus(HttpStatus::is4xxClientError, clientResponse -> {
76                  log.info("Status code is : {}", clientResponse.statusCode().value());
77
78                  return clientResponse.bodyToMono(String.class)
79                          .flatMap(responseMessage -> Mono.error(new MoviesInfoClientException(
80                                  responseMessage, clientResponse.statusCode().value()
81                          )));
82              })
83              .onStatus(HttpStatus::is5xxServerError, clientResponse -> {
84                  log.info("Status code is : {}", clientResponse.statusCode().value());
85                  return clientResponse.bodyToMono(String.class)
86                          .flatMap(responseMessage -> Mono.error(new MoviesInfoServerException(
87                                  "Server Exception in MoviesInfoService " + responseMessage)));
88              })
89              .bodyToFlux(MovieInfo.class) Flux<MovieInfo>
90              // .retry(3)
```

Build ▶ 4: Run 5: Debug DB Execution Console 6: TODO 0: Messages Terminal

Build completed successfully in 2 s 655 ms (20 minutes ago) 67:26 LF



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