

```
1 package com.verizon.learning.controllers;
2
3 import java.util.Arrays;
4 import java.util.List;
5
6 import org.springframework.http.MediaType;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.PathVariable;
9 import org.springframework.web.bind.annotation.RestController;
10 import org.springframework.web.reactive.function.client.WebClient;
11
12 import com.verizon.learning.valueobjects.Customer;
13
14 import reactor.core.publisher.Flux;
15 import reactor.core.publisher.Mono;
16
17 @RestController
18 public class InvoiceRestController {
19
20     WebClient webClient = WebClient.create("http://localhost:8080");
21
22     @GetMapping(path = "/invoice-customers", produces = {MediaType.APPLICATION_JSON})
23     public Flux<Customer> handleGetInvoiceCustomers() {
24
25         // @formatter:off
26
27         Flux<Customer> customers =
28             webClient
29                 .get()
30                 .uri("/customers")
31                 .accept(MediaType.APPLICATION_JSON)
32                 .bodyToFlux(Customer.class);
33     }
34 }
```

```
10 import org.springframework.web.reactive.function.client.WebClient;
11
12 import com.verizon.learning.valueobjects.Customer;
13
14 import reactor.core.publisher.Flux;
15 import reactor.core.publisher.Mono;
16
17 @RestController
18 public class InvoiceRestController {
19
20     WebClient webClient = WebClient.create("http://localhost:8080");
21
22     private Mono<Customer> getCustomerById(String customerId) {
23
24         Mono<Customer> customer =
25             webClient
26                 .get()
27                 .uri("/customers/{customerId}", customerId)
28                 .retrieve()
29                 .bodyToMono(Customer.class);
30
31         return customer;
32     }
33
34     @GetMapping(path = "/invoice-customers", produces = {MediaType.APPLICATION_JSON})
35     public Flux<Customer> handleGetInvoiceCustomers() {
36
37         // @formatter:off
38
39         Flux<Customer> customers =
40             webClient
41                 .get()
42                 .uri("/customers")
43                 .accept(MediaType.APPLICATION_JSON)
44                 .bodyToFlux(Customer.class);
45     }
46 }
```

```
55 @GetMapping(path = "/invoice-customers/{customerId}")
56 public Mono<Customer> handleGetInvoiceCustomerById(@PathVariable String
57     customerId) {
58
59     // @formatter:off
60
61     Mono<Customer> customer =
62         webClient
63             .get()
64             .uri("/customers/{customerId}", customerId)
65             .retrieve()
66             .bodyToMono(Customer.class);
67
68     // @formatter:on
69
70     return customer;
71 }
72
73 @GetMapping(path = "/select-customers")
74 public Flux<Customer> handleGetSelectCustomers() {
75
76     List<String> ids = Arrays.asList("101", "103", "105");
77
78     Flux<Customer> customers =
79         Flux.fromIterable(ids)
80             .flatMap(x -> {
81                 handleGetInvoiceCustomerById(x);
82             })
83             .cache();
84 }
```

```
v6-sts - invoice-management/src/main/java/com/verizon/learning/controllers/InvoiceRestController.java - Spring Tool Suite 4

Package Explorer | JUnit | InvoiceRestController.java | CustomerRestController.java

InvoiceRestController.java
55 @GetMapping(path = "/invoice-customers/{customerId}")
56 public Mono<Customer> handleGetInvoiceCustomerById(@PathVariable String
57 // @formatter:off
58 //
59 //
60 // Mono<Customer> customer =
61 //     webClient
62 //         .get()
63 //         .uri("/customers/{customerId}", customerId)
64 //         .retrieve()
65 //         .bodyToMono(Customer.class);
66 //
67 //
68 //
69 //
70 // @formatter:on
71 //
72 return this.getCustomerById(customerId);
73 }

CustomerRestController.java
76 @GetMapping(path = "/select-customers")
77 public Flux<Customer> handleGetSelectCustomers() {
78     List<String> ids = Arrays.asList("101", "103", "105");
79     Flux<Customer> customers =
80         Flux.fromIterable(ids)
81             .flatMap(x -> {
82                 Mono<Customer> customer =
83                     webClient
84                         .get()
85                         .uri("/customers/{customerId}", x)
86                         .retrieve()
87                         .bodyToMono(Customer.class);
88                 return customer;
89             });
90     return customers;
91 }
```

```
v6-sts - invoice-management/src/main/java/com/verizon/learning/controllers/InvoiceRestController.java - Spring Tool Suite 4

Package Explorer | JUnit | InvoiceRestController.java | CustomerRestController.java

InvoiceRestController.java
72 return this.getCustomerById(customerId);
73 }

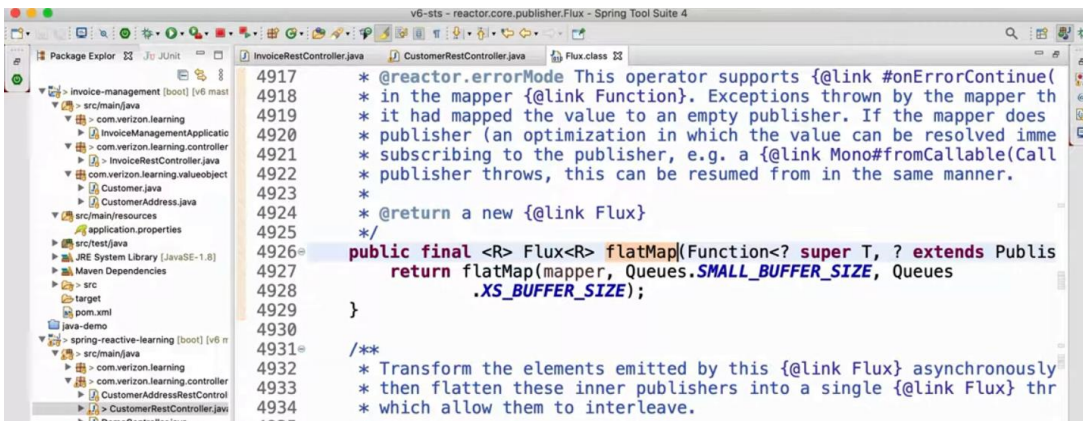
CustomerRestController.java
76 @GetMapping(path = "/select-customers")
77 public Flux<Customer> handleGetSelectCustomers() {
78     List<String> ids = Arrays.asList("101", "103", "105");
79     Flux<Customer> customers =
80         Flux.fromIterable(ids)
81             .flatMap(x -> {
82                 Mono<Customer> customer =
83                     webClient
84                         .get()
85                         .uri("/customers/{customerId}", x)
86                         .retrieve()
87                         .bodyToMono(Customer.class);
88                 return customer;
89             });
90     return customers;
91 }
```

```
v6-sts - invoice-management/src/main/java/com/verizon/learning/controllers/InvoiceRestController.java - Spring Tool Suite 4

Package Explorer | JUnit | InvoiceRestController.java | CustomerRestController.java

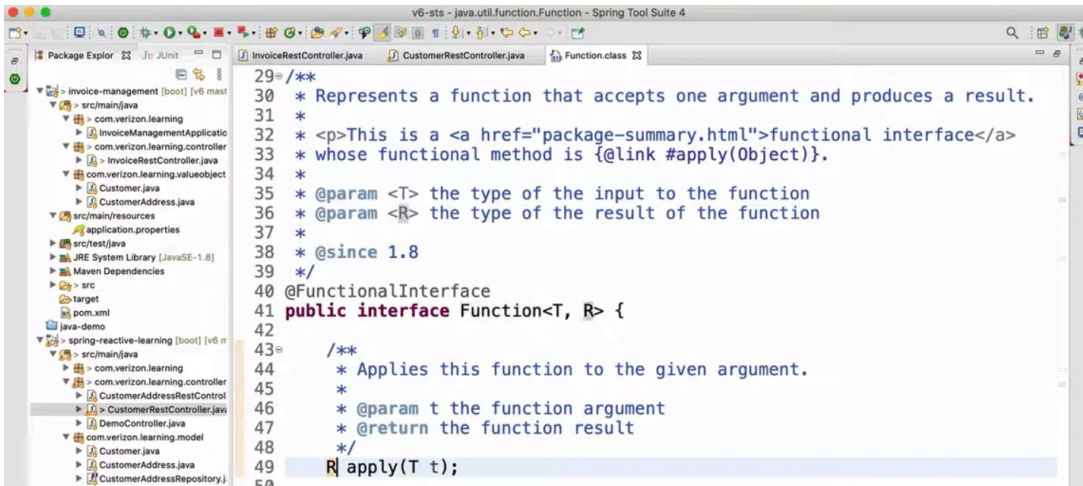
InvoiceRestController.java
72 return this.getCustomerById(customerId);
73 }

CustomerRestController.java
76 @GetMapping(path = "/select-customers")
77 public Flux<Customer> handleGetSelectCustomers() {
78     List<String> ids = Arrays.asList("101", "103", "105");
79     Flux<Customer> customers =
80         Flux.fromIterable(ids)
81             .flatMap(x -> {
82                 return this.getCustomerById(x);
83             });
84     return customers;
85 }
```

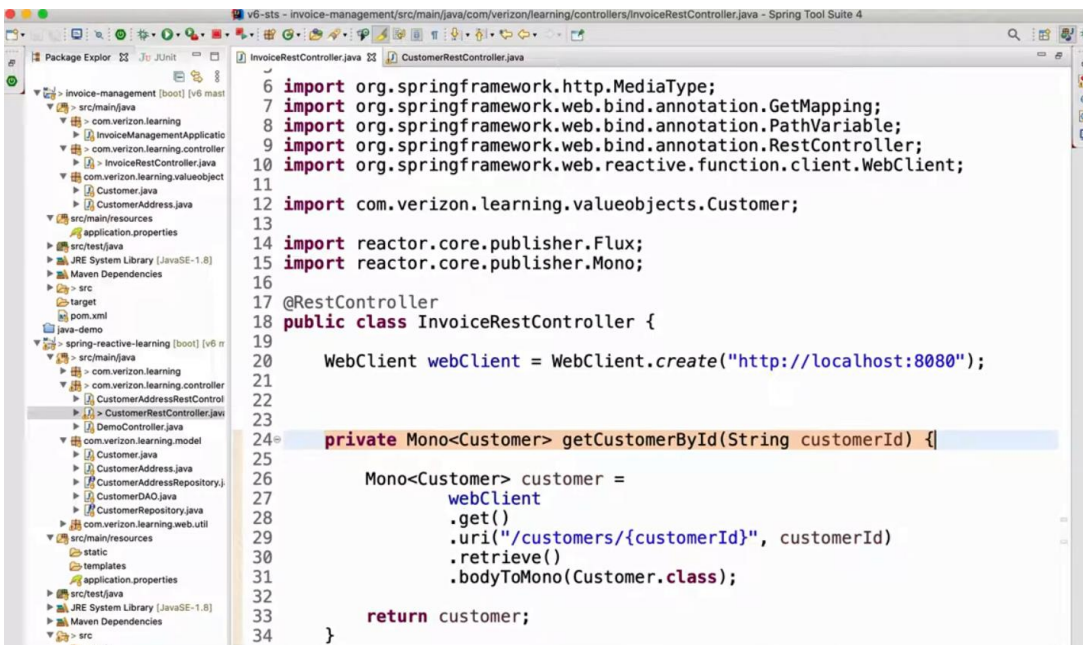



```
4917 * @reactor.errorMode This operator supports {@link #onErrorContinue(
4918 * in the mapper {@link Function}. Exceptions thrown by the mapper th
4919 * it had mapped the value to an empty publisher. If the mapper does
4920 * publisher (an optimization in which the value can be resolved imme
4921 * subscribing to the publisher, e.g. a {@link Mono#fromCallable(Call
4922 * publisher throws, this can be resumed from in the same manner.
4923 *
4924 * @return a new {@link Flux}
4925 */
4926 public final <R> Flux<R> flatMap(Function<? super T, ? extends Publis
4927     return flatMap(mapper, Queues.SMALL_BUFFER_SIZE, Queues
4928         .XS_BUFFER_SIZE);
4929 }
4930
4931 /**
4932 * Transform the elements emitted by this {@link Flux} asynchronously
4933 * then flatten these inner publishers into a single {@link Flux} thr
4934 * which allow them to interleave.
```

The **flatMap()** takes a Function that takes one parameter and returns a value back



```
29 /**
30 * Represents a function that accepts one argument and produces a result.
31 *
32 * <p>This is a <a href="package-summary.html">functional interface</a>
33 * whose functional method is {@link #apply(Object)}.
34 *
35 * @param <T> the type of the input to the function
36 * @param <R> the type of the result of the function
37 *
38 * @since 1.8
39 */
40 @FunctionalInterface
41 public interface Function<T, R> {
42
43     /**
44      * Applies this function to the given argument.
45      *
46      * @param t the function argument
47      * @return the function result
48      */
49     R apply(T t);
50 }
```



```
6 import org.springframework.http.MediaType;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.PathVariable;
9 import org.springframework.web.bind.annotation.RestController;
10 import org.springframework.web.reactive.function.client.WebClient;
11
12 import com.verizon.learning.valueobjects.Customer;
13
14 import reactor.core.publisher.Flux;
15 import reactor.core.publisher.Mono;
16
17 @RestController
18 public class InvoiceRestController {
19
20     WebClient webClient = WebClient.create("http://localhost:8080");
21
22
23
24     private Mono<Customer> getCustomerById(String customerId) {
25
26         Mono<Customer> customer =
27             webClient
28                 .get()
29                 .uri("/customers/{customerId}", customerId)
30                 .retrieve()
31                 .bodyToMono(Customer.class);
32
33         return customer;
34     }
```


3. A Simple User Service

We're going to be using a simple *User* API in our examples. **This API has a GET method that exposes one method *getUser* for retrieving a user using the id as a parameter.**

Let's take a look at how to make a single call to retrieve a user for a given id:

```
WebClient webClient = WebClient.create("http://localhost:8080");
```

```
public Mono<User> getUser(int id) {
    LOG.info(String.format("Calling getUser(%d)", id));

    return webClient.get()
        .uri("/user/{id}", id)
        .retrieve()
        .bodyToMono(User.class);
}
```

In the next section, we'll learn how we can call this method concurrently.

4.1. Multiple Calls to the Same Service

Let's now imagine that we want to fetch data about five users simultaneously and return the result as a list of users:

ADVERTISEMENT

```
public Flux fetchUsers(List userIds) {
    return Flux.fromIterable(userIds)
        .flatMap(this::getUser);
}
```

Let's decompose the steps to understand what we've done:

We begin by creating a Flux from our list of *userIds* using the static *fromIterable* method.

Next, we invoke *flatMap* to run the *getUser* method we created previously. This reactive operator has a concurrency level of 256 by default, meaning it executes at most 256 *getUser* calls simultaneously. This number is configurable via method parameter using an overloaded version of *flatMap*.

It's worth noting, that since operations are happening in parallel, we don't know the resulting order. If we need to maintain the input order, we can use *flatMapSequential* operator instead.

As Spring WebClient uses a non-blocking HTTP client under the hood, there is no need to define any Scheduler by the user. *WebClient* takes care of scheduling calls and publishing their results on appropriate threads internally, without blocking.

← → ↻ baeldung.com/spring-webclient-simultaneous-calls



Baeldung

Start Here

Courses ▾

Guides ▾

About ▾



4.2. Multiple Calls to Different Services Returning the Same Type

Let's now take a look at how we can call multiple services simultaneously.

In this example, we're going to create another endpoint which returns the same *User* type:

```
public Mono<User> getOtherUser(int id) {
    return webClient.get()
        .uri("/otheruser/{id}", id)
        .retrieve()
        .bodyToMono(User.class);
}
```

Now, the method to perform two or more calls in parallel becomes:

```
public Flux fetchUserAndOtherUser(int id) {
    return Flux.merge(getUser(id), getOtherUser(id));
}
```

The main difference in this example is that we've used the static method *merge* instead of the *fromIterable* method. Using the *merge* method, we can combine two or more *Fluxes* into one result.

4.3. Multiple Calls to Different Services Different Types

The probability of having two services returning the same thing is rather low. **More typically we'll have another service providing a different response type and our goal is to merge two (or more) responses.**

The *Mono* class provides the static `zip` method which lets us combine two or more results:

```
public Mono fetchUserAndItem(int userId, int itemId) {  
    Mono user = getUser(userId);  
    Mono item = getItem(itemId);  
  
    return Mono.zip(user, item, UserWithItem::new);  
}
```



The *zip* method combines the given *user* and *item Monos* into a new *Mono* with the type *UserWithItem*. This is a simple POJO object which wraps a user and item.