1. Goal

The simple goal of this lesson is to introduce you to the reasons and usage of the new *WebClient*.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is : m11-lesson3-start

If you want to skip and see the complete implementation, feel free to jump ahead and import: m11-lesson3

2.1. *WebClient* vs *RestTemplate*

The *WebClient* is a reactive and non-blocking HTTP client, as opposed to the classical *RestTemplate* - which is, of course - synchronous and blocking.

This works in a very typical way - we send a request, and need to wait for the response to come back.

So, for example, when working with SSE - *RestTemplate* is not a good fit.

WebClient is non-blocking - we send the request and get a callback - when the response is ready.

Of course, consuming a typical Servlet-based endpoint still works fine.

In conclusion - the *WebClient* generally provides higher concurrency, with fewer hardware resources.

2.2. Creating the *WebClient*

Next, we'll consume an SSE endpoint using the *WebClient*.

In the last lesson, we consumed the SEE endpoint in the browser. This time, we'll consume the same SSE endpoint using the *WebClient.*

We'll start by creating a simple *WebClient*using the builder pattern. This method gives us more flexibility, as we can define filtering options and set some defaults.

For example, we can set default HTTP headers. We'll set *application/json* as the default *content-type* for our *WebClient*.

```java
@Bean
WebClient webClientBuilder() {
    return WebClient
      .builder()
      .baseUrl(BASE_URI)
      .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
      .build();
}
```

2.3. Using the *WebClient*

We'll use the WebClient to retrieve all privileges in our application. The *PrivilegesController - findAll* method produces a Privilege - every 5 seconds.

First, let's start the app so that we'll be able to hit the */privileges* endpoint.

Next, let's autowire the client in a simple test.

Now we can perform a get operation, asking for a Server-Sent Events reponse type. We'll convert the result to a *Flux* and subscribe to this *Flux*. Then, we'll process (print) the data as it comes in.

Let's sleep here to make sure the test doesn't end before we have a chance to get the privilege data event 5 seconds.

```java
@RunWith(SpringRunner.class)
@ContextConfiguration(classes = { LiveTestConfig.class }, loader = AnnotationConfigContextLoader.cla
public class WebClientLiveTest {
    @Autowired
    WebClient webClient;

    @Test
    public void retrieveAllPrivileges() throws InterruptedException {
        Flux<Privilege> result = webClient.get()
          .uri("/privileges").accept(MediaType.TEXT_EVENT_STREAM)
          .retrieve()
          .bodyToFlux(Privilege.class);

       result.subscribe(privilege -> System.out.println(privilege.toString()));

       Thread.sleep(35000);
    }
}
```

And there we have it: the tests consumes the SSE endpoint and prints out the events.

Let's now continue exploring the WebClient in the next part of the lesson.