1. Goals

Understand WHY we need a Client, how to implement one and how to use it in complex scenarios against the API.

2. Lesson Notes

Why do we need a Client for the API?

A client consolidates the low level logic into a single point and provides a simple, clean way to use the API.

The client is especially useful for more complex scenarios.

A Simple Client

Starting with an empty Spring bean, we'll simply pull the logic from the previous tests into their own methods / operations.

For example - the *create* operation:

- POST the new Resource
- extract the URI of the newly created Resource from the *Location* header out of the Response
- do a GET on that URI and return the actual, new Resource

Dealing with Paths

Hardcoding the URL of the API is something we did to keep things simple, but not something we want to do long-term.

In this lesson, we are using a simple *UmPaths* class to read the URL from a properties file and get rid of the hardcoded values.

The properties file that now holds this info by default is *web-local.properties* (along with the alternative *web-dev.properties*, *web-cargo.properties*). These hold the following properties to create the URL that the live tests go to:

```
http.protocol=http
http.host=localhost
http.port=8082
http.sec.path=/um-webapp
```

And that's about it - a simple and flexibile solution that now allows us to run the live tests against any environment where our API is deployed, not just *localhost*.

To do that - change the properties file that the live tests use, here's a quick example:

```
- DwebTarget=dev
```

This is going to of course mean that the *web-dev.properties* file is used instead of the default *web-local.properties*.