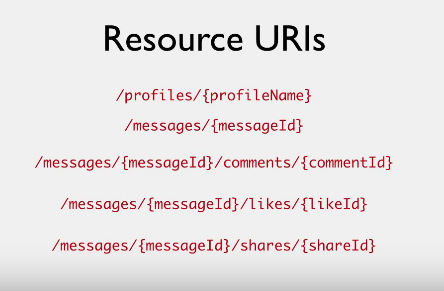
<https://javabrains.io/courses/javaee_jaxrs/lessons/RESTful-URI-types>



Instance resource URI

Notes

URI Types

We've designed URIs for messages and comments in the previous tutorial. To recap, a message is accessible at: /messages/{messageId}

And comments are accessible at: /messages/{messageId}/comments/{commentId}

These URIs are great when you want to look up a particular message or a particular comment. These are called instance resource URIs. A single instance of a message or a comment is accessible by the instance resource. Instance resource URIs typically have a unique ID of that resource to identify which instance you are interested in.

What about if you want all messages?

The answer is simple. Just access /messages

That's it! Again, this is analogous to a simple static site with HTML pages. Accessing a directory gives you all the contents in that directory. So, think of /messages as the top level directory for all messages and accessing that URI gives a list of all messages.

Similarly, if you need all comments made for message 2, the URI is: /messages/2/comments. That's the directory for all comments for message 2.

These URIs are different from the URIs we saw in the previous tutorial. They do not represent a particular resource, but rather, a collection or a list of resources. So, they are called *collection URIs*. These URIs pull up a collection of instance resources.

This also explains why the resource names are in plural. They help the client understand that they are working with a collection of resources with these URIs.

/messages returns all messages /profiles returns all profiles/messages/{messageId}/comments returns all comments for messageId/messages/{messageId}/likes returns all likes for messageId /messages/{messageId}/sharesreturns all shares for messageId

This shows you the advantage of *nesting* related resources like messages and comments, or messages and likes in this way. But this also brings up a problem. What if you need a list of *all* comments irrespective of which message they are associated with. This URI is not good enough for that. The way we have designed our URI, you have to give a message ID here. This is where a decision has to be made about how you want resources to be accessed. In this example, I choose to not provide an option for getting all comments. A comment makes sense only in the context of a message, and I don't want my clients to make requests for all comments. If I needed to provide that functionality, I would design the comments URI like /comments rather than /messages/{messageId}/comments. Again, no right or wrong. This is something you decide.

Filtering collections

When you have collection URIs like this, you'd want a way to filter the result. It's not so much a problem for the comments URI. Getting all comments for a message is mostly not going to be a big list, but imagine getting all messages. That's going to result in a lot of data, and the client who's making the request probably doesn't want all that data. We should design our API to provide a way for the client to paginate or filter the results.

One way to do that is using query params. We have so far avoided query params in our URIs, but filtering and pagination is a good scenario to use them. One standard practice to provide pagination is to have two query params: starting point and page size.

For example, consider this URI:

/messages?offset=30&limit=10

This URI fetches messages starting from message number 30 and returns the next 10 messages. The offset and limit params correspond to the start and page size values. The offset param tells you where client wants to start, and the limit tells you how many records the client wants. This, of course, assumes a specific order in which messages are returned, whether it is chronological or ID based. You as an API developer needs to make sure that the order that's returned is the same: The order in which you responded to the request for the first page should be retained in the requests for the second page.

The client who uses your API might have their own logic for displaying pagination controls on their UI. But they would use these two params to make calls to the RESTful service to get chunks of data in pages.

You can also choose to implement other kinds of filters using query params. Take an example of retrieving messages based on date. Let's say you want to provide an ability for clients to retrieve messages posted in a given year. You could have them send request like this:

/messages?year=2014

This returns all messages made in the year 2014. This is of course, something you can use together with pagination like this:

/messages?year=2014&offset=50&limit=25

Summary

There are two types of REST URIs. One is instance resource URIs that identify a specific resource, and the other is collection URI which represent a collection of resources. Collection URIs usually end in plurals, like *messages*, *comments*, *products* and so on, and are typically a portion of the resource URI. An instance resource URI identify a specific resource below a collection resource URI.

And finally, you can implement query params as a way of achieving pagination and filtering when accessing collection URIs.

I encourage you to do the exercises where you'll practice writing RESTful URIs for collection resources. I'll see you in the next video.

5th class::

