<https://javabrains.io/courses/javaee_jaxrs/lessons/Introduction>

#### Notes

## Welcome

Hello! Welcome to this course on Developing RESTful APIs with JAX-RS. I'm Koushik Kothagal, and in this course, I'll teach you how to write RESTful web service applications using JAX-RS.

To learn how to write RESTful APIs using JAX-RS, you'll need to learn 2 things: First, you'll need to learn about RESTful APIs or RESTful web services, and second, you'll need to learn how to implement them using JAX-RS.

Accordingly, this course is split into two important sections. The first few tutorials in this course introduce you to RESTful web services, explain the concepts and best practices. There are a few standard things to keep in mind when building RESTful web services. You'll learn about them first. I’ll also walk you through developing a REST API for a sample application. We’ll build a simple social media messaging application through out this course. The application will have some basic functionality of similar sites like Facebook or Google Plus.

After learning the concepts, we’ll implement some of the features of this social media messaging web service applications using Java and JAX-RS. We'll add features as we learn more about JAX-RS itself, and we'll write code as we go, to implement these features.

Restful web services have recently grown in popularity. I think now is a great time to learn about and work on Restful web services. I hope you are excited to get started!

## Web services in a nutshell

Web services are services that are exposed on the internet for programmatic access. They are online APIs that you can call from your code. To call an API when writing Java code, you add the jars or classes in your class path. Everything executes on a single machine. In the case of web services, you have different pieces of code deployed in different machines and calling methods of each other over the network.

You must have seen different Facebook or Twitter apps. Or games that can post to your Facebook wall. Even though the games are not designed by Facebook. How can they do that? They do that by calling online APIs. Companies like Facebook and Twitter publish web services that let other developers call them from their code. So, other application developers can write code to consume these services and do things on Facebook or Twitter itself.

In a way, they are similar to web pages. For example, Twitter has a web site at twitter.com. When you access it, you get HTML response that lets you read and write tweets. They have some HTML elements for data as well as styling with CSS. That's because they are meant for humans to read and consume. But, Twitter also has this different URL called api.twitter.com that does a lot of the same things as twitter.com, but it behaves a bit differently. For instance, it does not have HTML and CSS. Any data it returns is in XML or JSON format. And there are specific URLs for different operations. This is what the developers can use from their code, to read data and write data to Twitter. Since it deals with just barebones data, they can just parse the data and build up their objects and data structures. There's no need for all the fancy HTML and CSS.

This second part is what we'll be learning to build in this course. The online APIs. Of course, there are many ways to build such online APIs or web services. One way is to build them as RESTful web services. RESTful web services are a type of web services that are modern, light-weight, and use a lot of the concepts behind HTTP, the technology that drives the web. There is another type of web services that you can choose to write your services in, and that's called SOAP web services. There is a separate Java Brains course on SOAP web services [here](https://javabrains.io/courses/javaee_jaxws). This course is going to be about REST.

## Web services characteristics

When we talk about web services, there are a few characteristics about them that we have to keep in mind. First, they are web services, so the exchange of data happens over the web. Over HTTP. A client sends an HTTP request, and the server returns back an HTTP response. Similar to web sites. But like we discussed, instead of the response being complete web pages, only the data is returned, because the client is just a program, and not a human. The client could then have it’s own logic to present the data to the users in a presentable format, but the exchange between the web service client and web service server is usually just bare-bones data.

The next characteristic of web services is the protocol used. Now what’s a protocol? When a web service client makes a request to a web service endpoint, they are usually messages transmitted from one machine to another. These messages need to be in a format, a language, that both the client and the server can understand. This language, or protocol, is standardized in some web service types. For example, SOAP web services is a type of web services, where the protocol always has to follow the standard called SOAP. All SOAP web services uses this protocol to communicate with clients. It used to stand for Simple Object Access Protocol. But that name is now discontinued and we are stuck with just the acronym. There is a specific format, which is XML, and there are specific rules which detail how that XML should be. The client and the server needs to talk to each other using this SOAP protocol only!

Ok, so what’s the protocol for REST? Well, there is none! Yes, a REST client can send messages in XML. Or it can send messages in JSON format! Or text format! There are no rules. As long as the client and server understand each other, everybody is happy.

Secondly, let’s look at how the communication happens. We saw that since RESTful web services are web services, the request and response messages are almost always exchanged over HTTP. But in HTTP, there are different methods available. You would have heard of GET, POST PUT and so on. Which methods to you need to use? What’s the standard for REST?

Well, the answer is, there is none. Messages can be exchanged in any (or all) HTTP methods. There are guidelines and best practices that tells you what methods need to be used when designing the service, depending on what the request is, but there is no rule as such.

Let's look at the next characteristic: service definition. When you are coding and you need to call a method of a library class, you need to know certain things. The name of the class, the method name, the input arguments and so on. When you call web services, a similar concept applies. You need to know what the method does, what the input arguments are and what the return type is. You need to know the **service definition**. In the SOAP web services world, every web service provider publishes a formal document called WSDL that contains all the details that any client would ever need to know about the web service. This is again in XML, and it needs have a specific structure. It contains details about the methods available, the input and output types and so on. Every client has all the details they’d need.

What’s the formal document that specifies the service definition details for REST web services? I’m pretty sure you can guess the answer to this one. Yes, you are right. There is none! Most Restful web services come with an informal README document written not in XML but in readable English. Many of them don’t even have documentation. In fact, the best RESTful web services would not even need any documentation, formal or informal. We’ll talk about this topic later.

We could go on with this comparison, but let’s stop here. You get the idea. You are probably wondering if this is really true. Is it really the case? Does the RESTful web services have any rules at all? Could my grandmother have written a RESTful web service? Based on what we have seen so far, anything goes! There are no rules. In contrast, SOAP web services have strict rules for each of these characteristics we've seen. So, what’s going on here?

The reason for this kind of a difference is simple. All SOAP web services follow this thing called the SOAP specification. This specification is a set of rules that dictate what a SOAP web service should be. This was designed by a committee, and it is still maintained by the committee. The specification lays out all the rules, including the rules we just discussed. If a web service doesn’t follow even one of these rules, it is, by definition, **not** a SOAP web service. As simple as that.

RESTful web services, on the other hand, does not have any specification! It is a concept. An idea. There is no specification, and no committee to tell you what’s right and what’s wrong. The term REST was first introduced by a guy named Roy Fielding in his doctoral thesis back in the year 2000. REST stands for REpresentational State Transfer, and it is not really about web services at all!

## Representational state transfer

Representational state transfer is actually an architecture style. Say, you are working on the architecture of a new application. There are certain decisions you’ll need to make. Certain criteria you need to think about. REST consists of a coordinated set of these criteria and constraints that you can use to guide you in the application architecture. It is a set of guidelines. It is a style of architecture.

You can use this style for any application. However, if you apply this style and these guidelines when architecting a web service, you have drum roll Restful web services. There in lies the difference. Unlike SOAP web services, you don’t have any strict rulebooks to follow when it comes to Restful web services. You can have a spectrum. It is common to hear people say some web service is completely Restful, and some other web service is not fully RESTful. Really! And the goal when building RESTful web services is to make it as RESTful as practically possible.

In the next tutorials, we’ll start learning more about these constraints and understand what good RESTful web services look like and why.