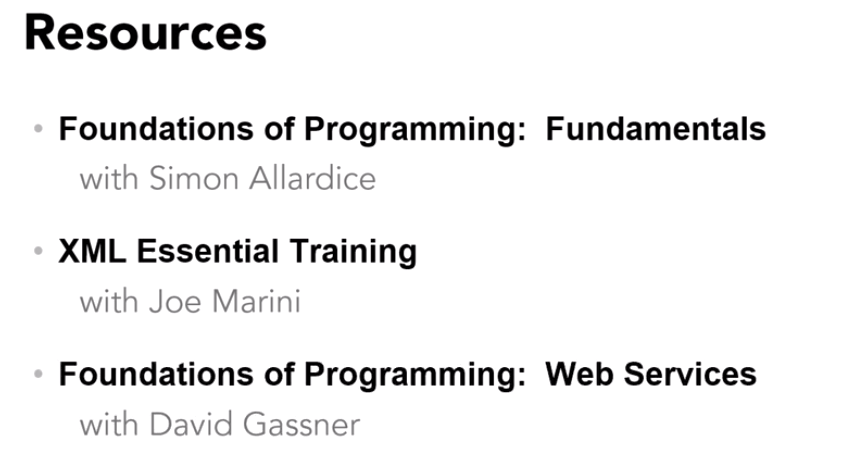
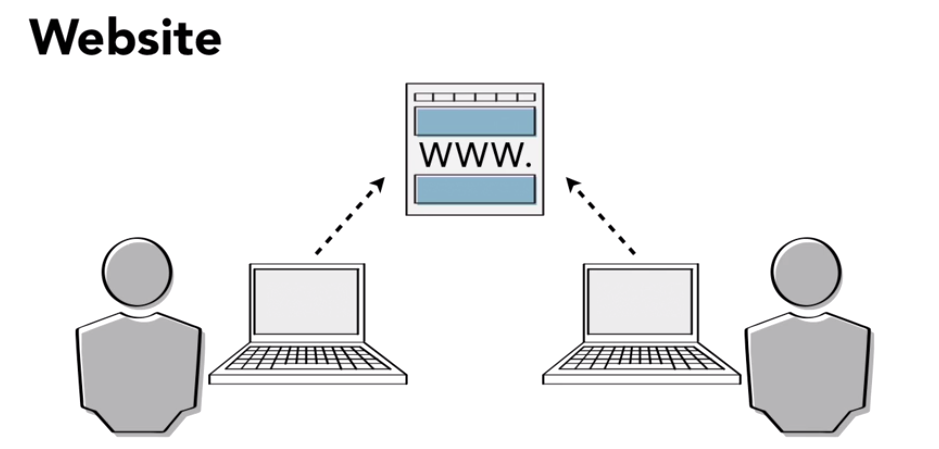
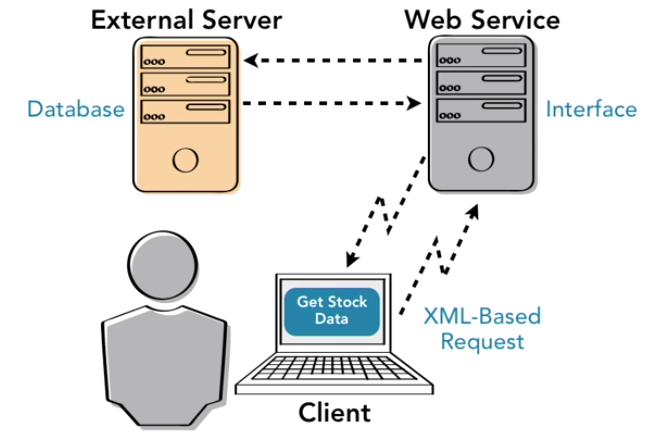
Web services are designed to work with other programs and allow computers to remotely call functions.



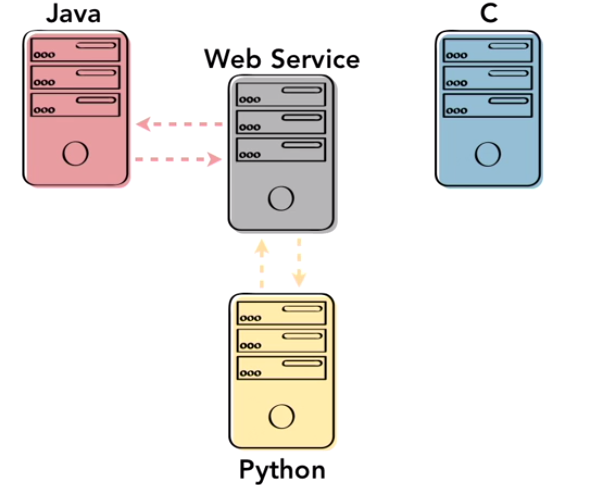


A website can be thought of as a communication between two humans, where the content is arranged in such a way that each human can read and understand it



A web service, on the other hand, is the communication between two computers. Instead of text and graphics, it transmits code requests that interact directly with another computer, often without any human intervention. Computers can use web services to communicate directly with one another at a low level. Sending and receiving requests for raw data or sharing functions and methods. In this sense, a web service is the backend part of the world wide web.

 A web service allows us to leave key functions on the source server instead of pulling them down to the local browsers. This makes it much easier to update and modify them from a centralized location.



Web services bridge the gap of understanding the code written in different languages by providing compatibility across multiple languages and operating systems. Rather than requiring the user to know the details and precise code structure of every program they access; a web service provides a set of standard information in universally readable format. Programmers often call this the black box method of programming. For maximum compatibility and flexibility, users should be able to effectively interact with the program with as little information as possible.

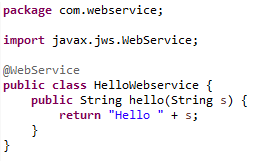
The ultimate universal language for modern computers is something called XML, or extensible mark-up language. This format is essentially nothing more than plain text with a series of very simple tags creating a hierarchy. Every computer can interpret text. So, XML is universally compatible while still being flexible enough to support complex transmissions of information. Web services communicate through the web services description language or **WSDL** which is based in the XML format.

Using this format, it's easy to transmit all the information required to use a program in a format understandable by any computer.

**Creating a Simple Web service in Java:**

**@webservice** directly before your class designation. When I put this in I will have to import it, so I'll just import Javax.jws and that means that Java will now investigate this annotation package and understand that everything that follows is a web service and should be treated in a certain way

Ex:

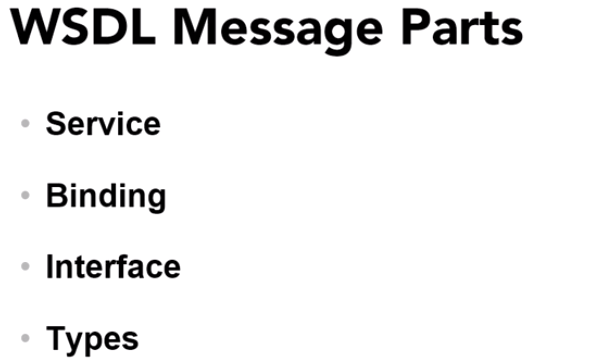


Once the basic code is complete, you can build, package, and deploy your service.  A web service must run on a server, and then be accessed by a client



Web Services Description Language or WSDL, is an XML based language used by web services to describe the functionality they offer. This can be thought of as the machine-based equivalent to the class and method descriptions of normal code. A WSDL description of a web service provides all the information that a computer needs to know how to call the web service, what parameters to pass, and what data structures they expect to get back.

All the end user really needs to know is, what type of data is expected for the inputs and the outputs. The most current version of WSDL is version 2.0



**Service**: Service part of the message contains direct references to the endpoints, their bindings, and their locations. The location is usually defined by a simple HTTP URL string, which could be in an external website or a port on your computer.

**Binding**: Specifies the interface and tells the reader what type of SOAP binding style and transport will be used. Such as RPC, or document style encodings. It also contains one or more operations tag that define the operations that can be used with the SOAP.

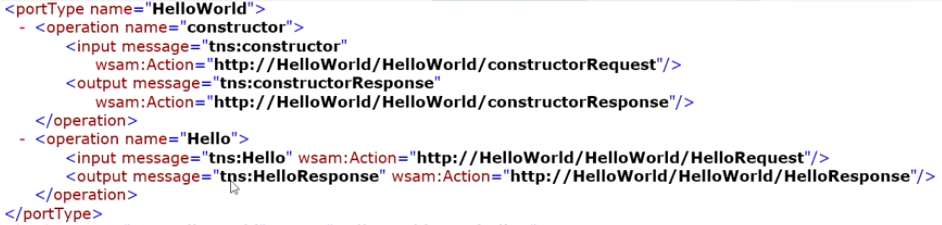
**Interface**: Defines the web service itself, including all the methods. As with the bindings, each method of the web service is represented by an operation tag. Inside that tag there is an input and an output tag. These indicate to the user what data structures are accepted and returned by that method.

**Types:** Describes the data itself.

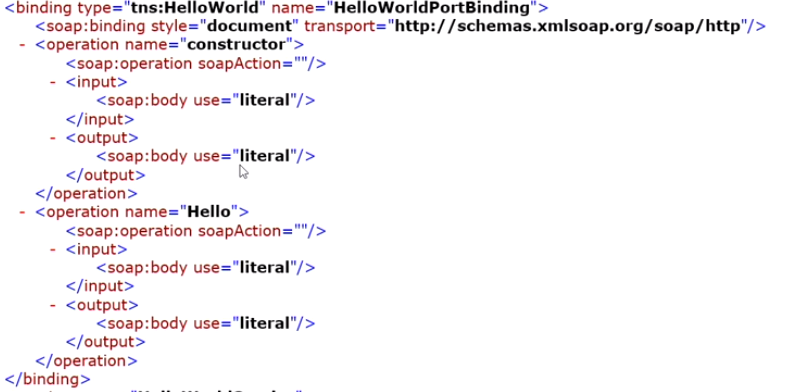
**Root tag** of soap xml is **definitions.** It contains information about where and how each of the main components of your message are defined. Most of these referenced URLs serve as something of a universal standard for defining tags like XSD, SOAP and so on.

**Message Tag:** Each one of the message tags defines one possible transfer of data. There are two per method. One for the input, and one for the response.

**Port Type**: The port type tag looks like the messages tags in form, but, has a slightly different function. This constructs an HTTP URL of the package class and method identifiers for each method, to be used by the accessing program.



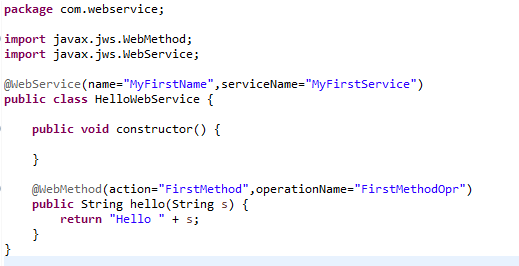
**Binding** Tag: Lists the methods once again providing the SOAP bindings for each one this time



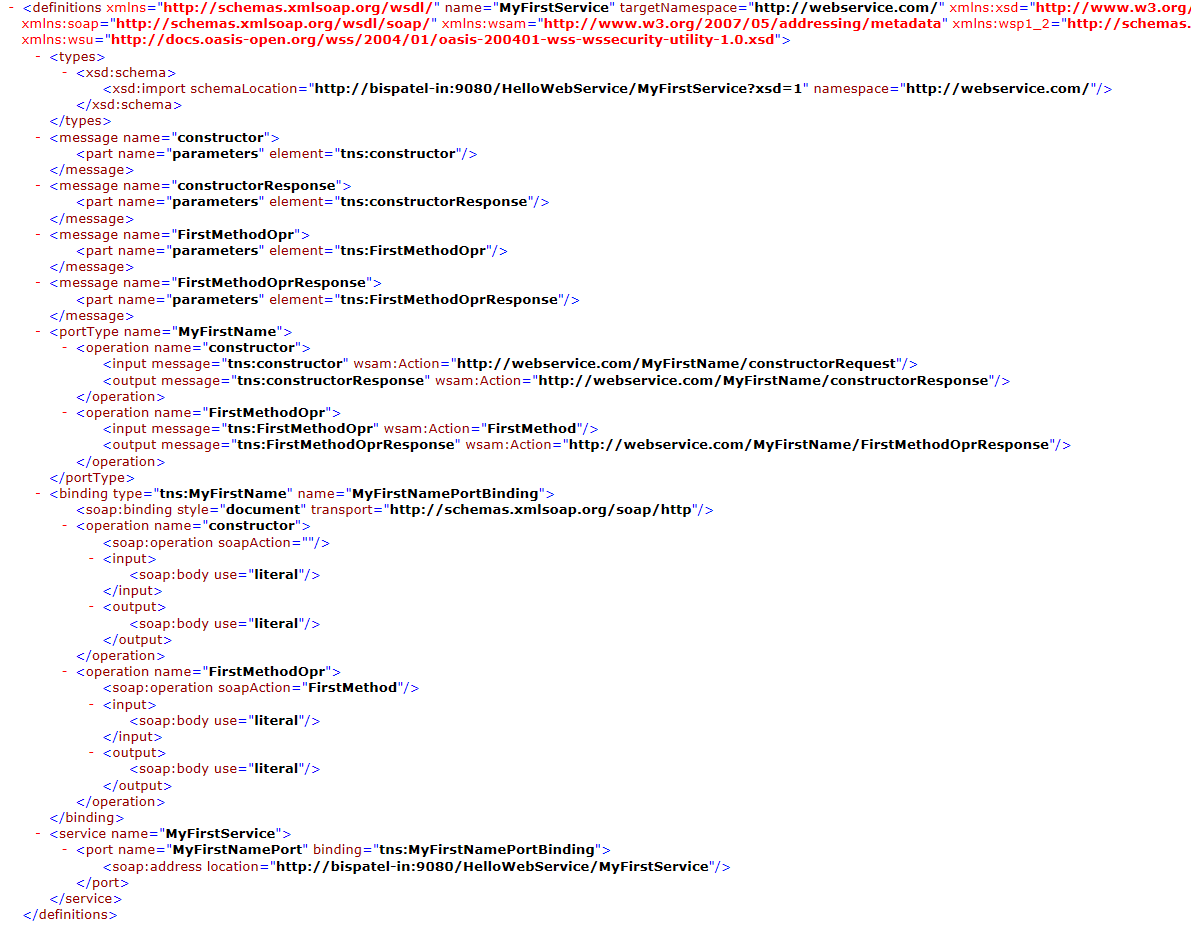
**Service** tag gives the actual URL where the service resides. Last phase uses port 8080, so the location should be just the name of your computer, the port, and then the class path to the service.



Customizing the Web Service:

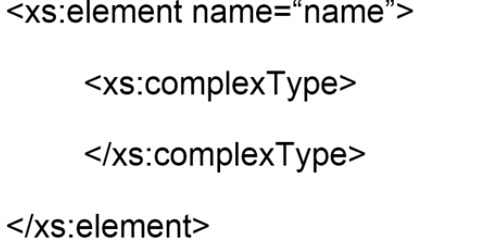


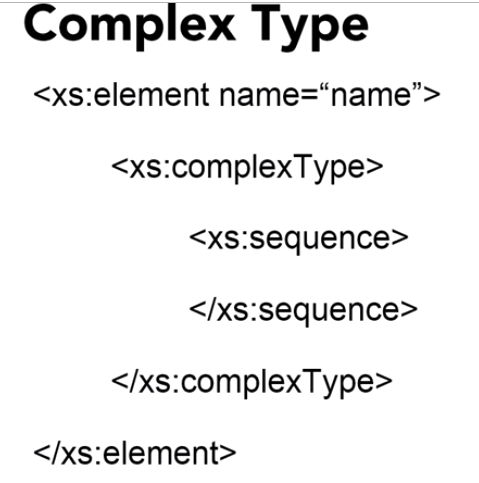
Following is the WSDL generated out of this method



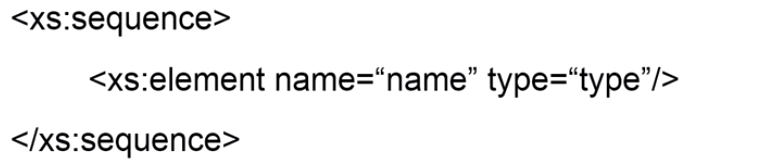


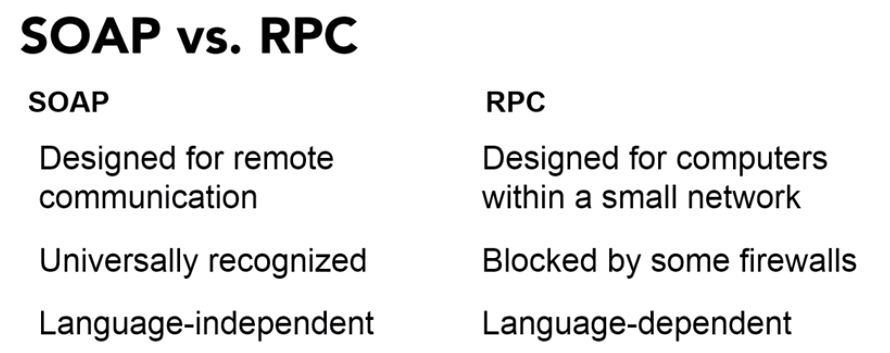
This defines the name of the element which will be used if you decide to refer this element elsewhere in your WSDL. Within that, add the tag xs: complex type.





Within the complex type tag, the innermost main tag is sequence. Inside this tag, you can write a list of all the elements that you want your complex type to use.



RPCs primarily meant to run on computers within a small well-defined network, such as a computing cluster

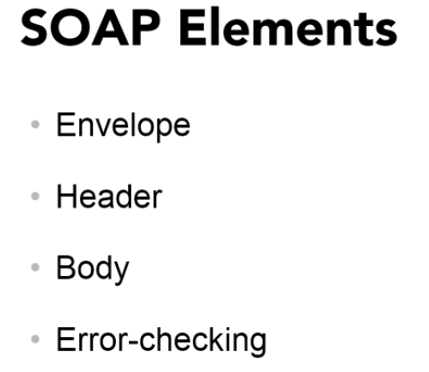
RPCs don't work as well on HTTP messages because they were not designed with compatibility and security in mind. Firewalls and PROXY servers usually block RPCs due to their security risks. And not all computers can run all RPCs without substantial modification.

To create and transmit soap messages, we will need a soap toolkit.

There are several toolkits available for soap but the most commonly used one for Java is the **Java API for XML web services, which is abbreviated as jax-ws.**

This defines the standard Java to WSDL mapping, which automatically translates Java code into a WSDL message using the SOAP protocol. In other words, **JAX-WS is the protocol that automatically generates the WSDL documents that appeared when we write the Java classes**.

The Java EE platform already includes JAX-WS and the JAX-WS API is the core that the Glassfish server is based on.

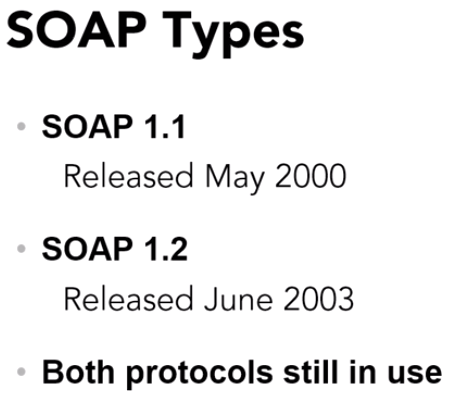


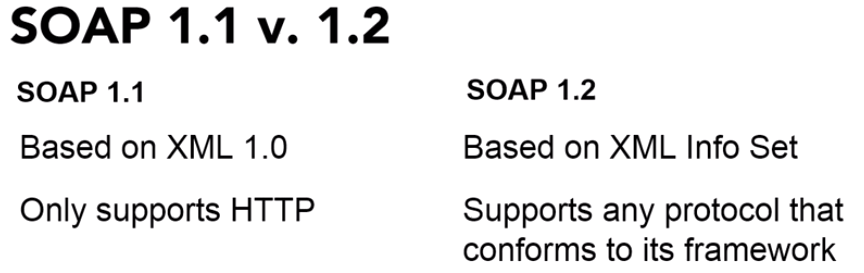
For an XML document to be used as a SOAP message, it must contain the following four elements.

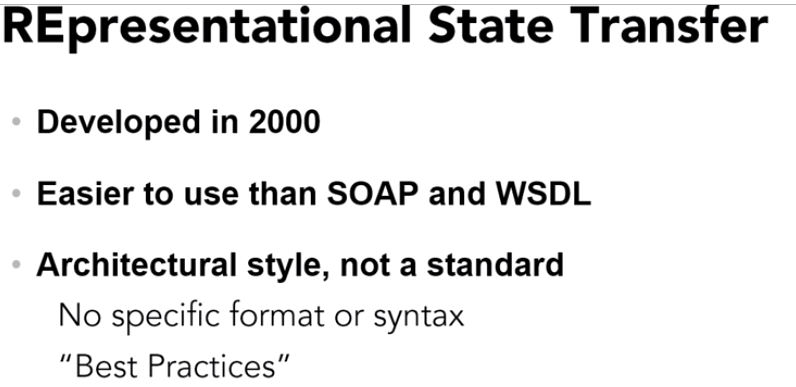
**Envelope:** This is a container for the document that identifies it as a SOAP message. This element is required to inform the recipient of how to use this document. As opposed to just an ordinary XML document

**Body:** Most important part of the message. It contains all the calls and the responses. This defines the actual function of the SOAP message

**Error-checking**: Diagnostic tool that makes it possible for the sender to see anything that goes wrong during the recipient's processing of the message.



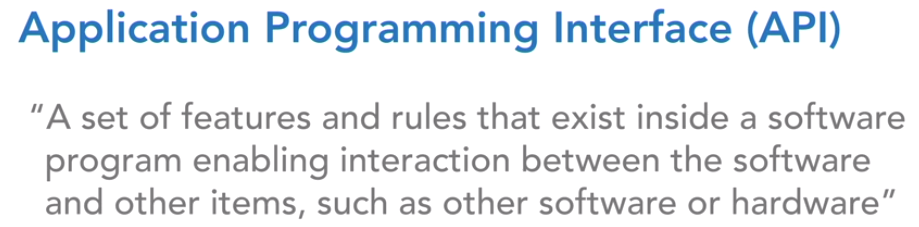




Soap and WDSL are standards. TheW3C releases details specifications for them that all messages must follow to be a part of that standard.

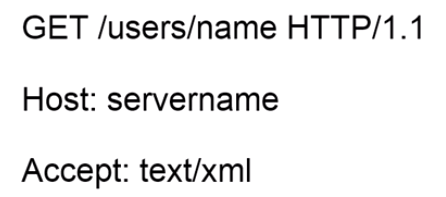
When it comes to actual programming, SOAP and WSDL focus on the methods and the objects that will be accessed. In contrast, REST focuses on the systems resources, especially the ways in which resource states are addressed and transferred via HTTP.







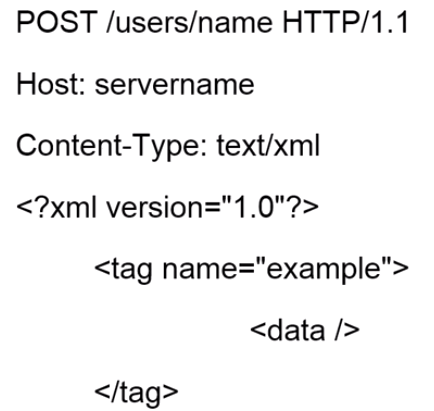
A restful web service is **stateless**. This means that any request can be run on any server and can be forwarded from one server to another as needed. Each request is designed to be complete and independent. So, the components in the intermediary servers can forward, route, and load balance without needing to hold local state data.

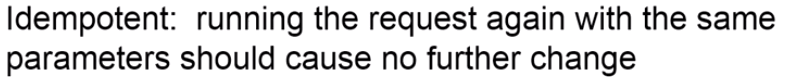


**GET**: Specifies what the server should do with the URI.

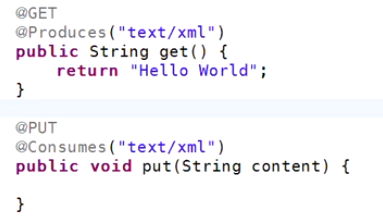
The second part is the URI itself which uses directory structure to specify where to look for the information. In this case, the name value would be the name of the person for whom you want to get the user name.

Finally, the HTTP/1.1 part tells the server what format you're using to write the message. To expand this to a full request, you would add two extra lines afterward.



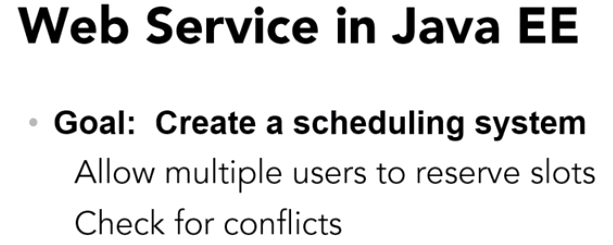


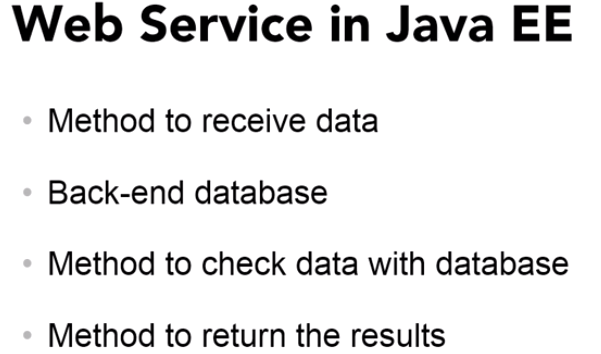
POST is not idempotent



The @Consumes and @Produces annotations specify the MIME type of the data that can be accepted and produced by this method. The acronym MIME stands for **multipurpose internet mail extension**, which has very little to do with the current usage, but it's a legacy of the original use of these data types.

**Example: Reservation of Seats**





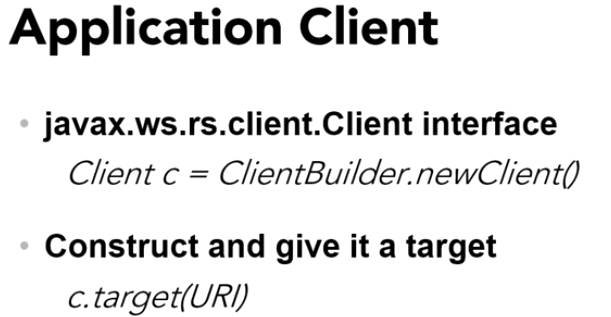


**Writing Web Service End Point:**

**@Path** annotation indicates the web service

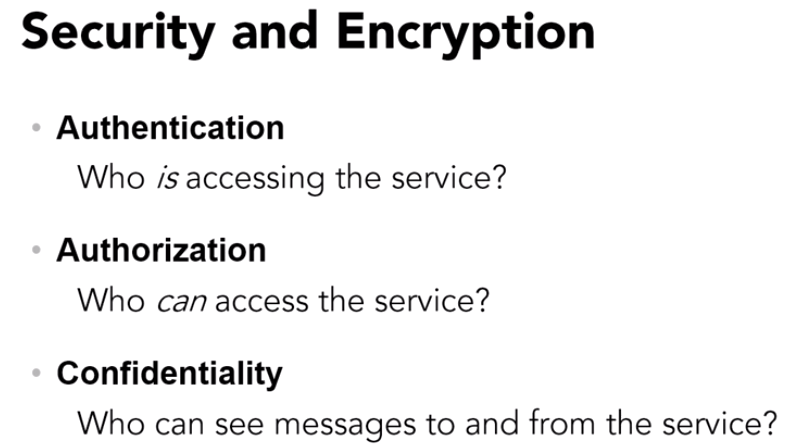
**How to develop a client that can send and receive messages from the RESTful service.**

Application clients for RESTful web services can be developed through the JAX RS client API. However, they aren't limited to only Java services. A jaxed RS client can access any RESTful resource, regardless of language. In general, once our web service is released, other users will be creating clients to send requests to your service.





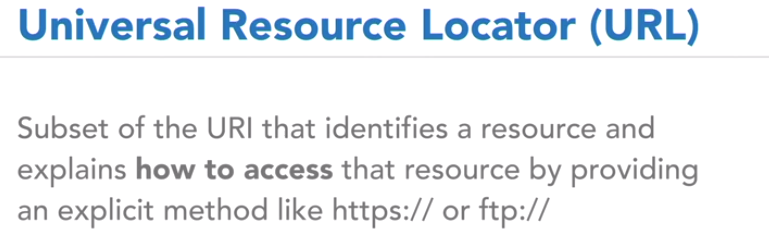




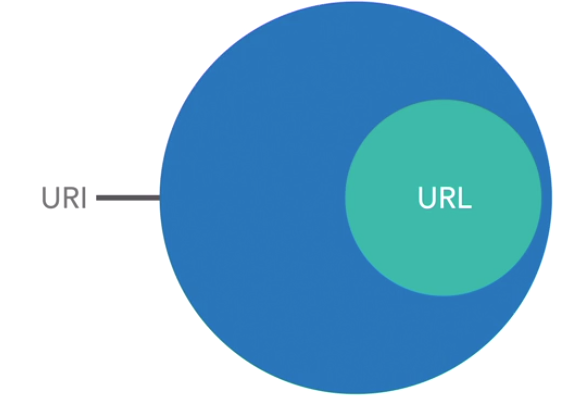
**Universal Resource Identifier (URI)**

A compact sequence of characters that identifies an abstract or physical resource that provides a simple and extensible means for identifying a resource.

The URI is the most generic method for naming and locating a web resource.

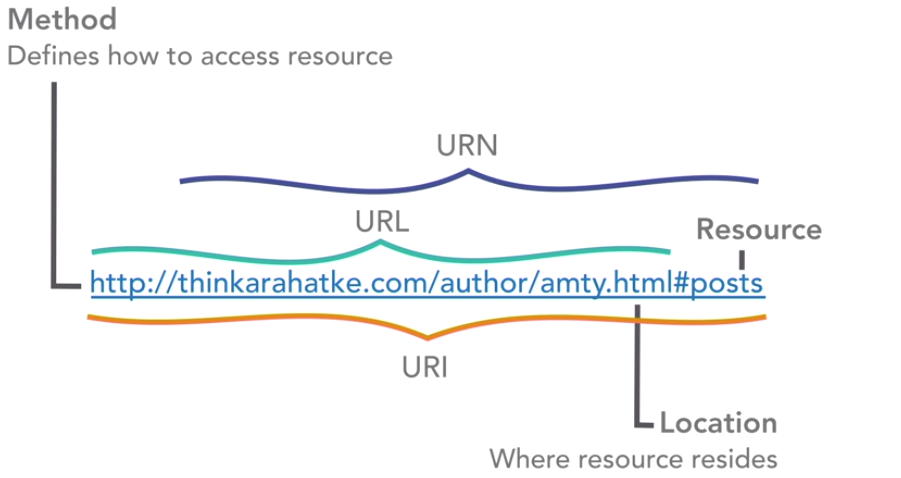


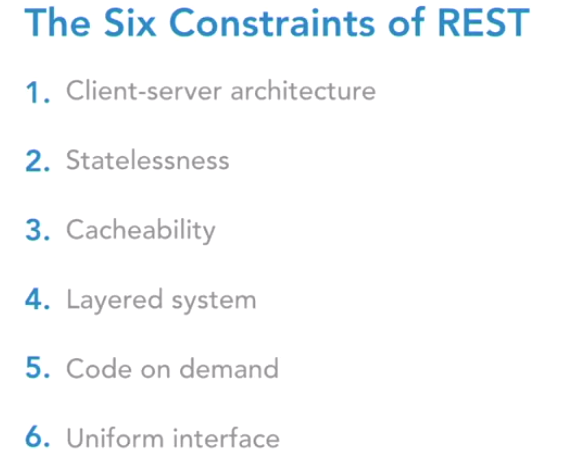


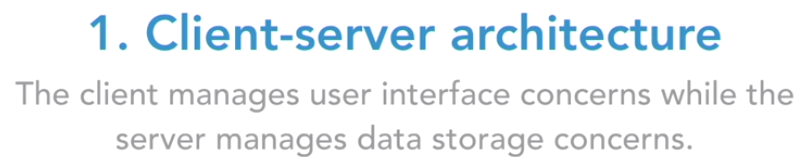


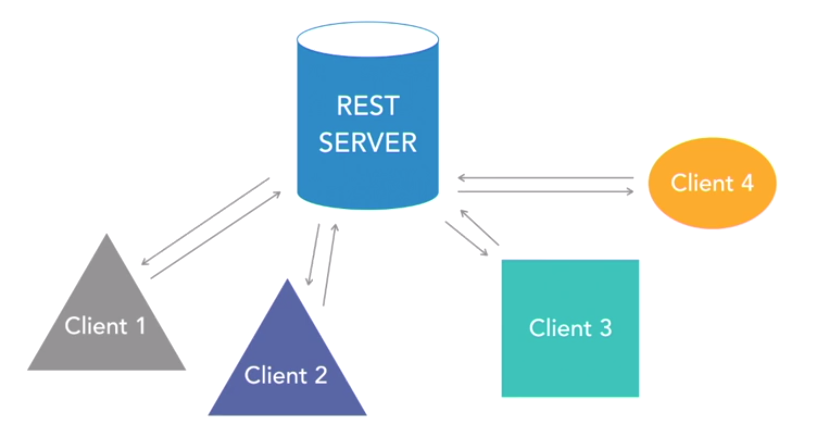
All URL are URI

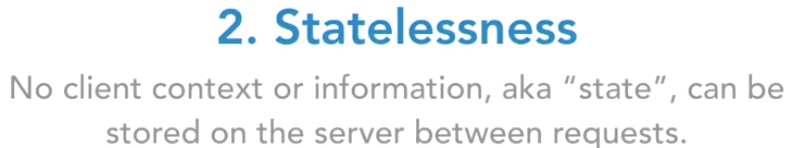
**URL URI and URN**

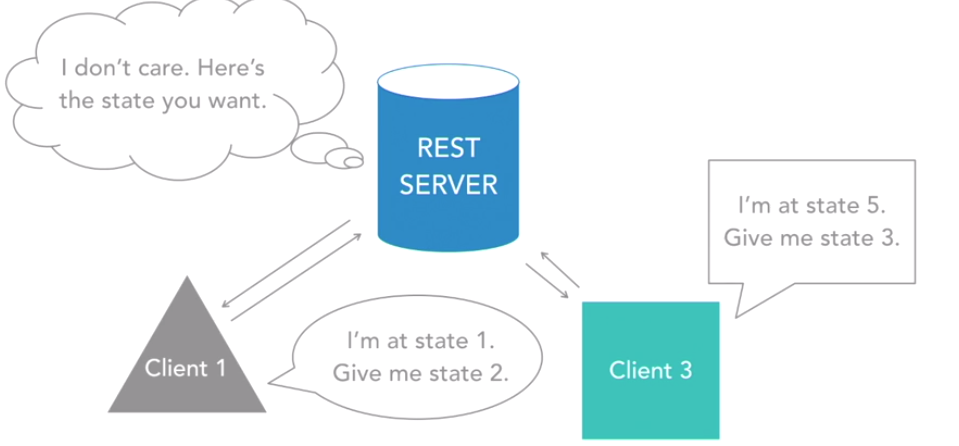


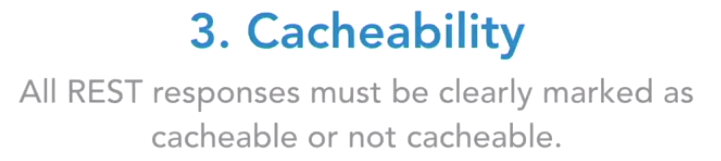


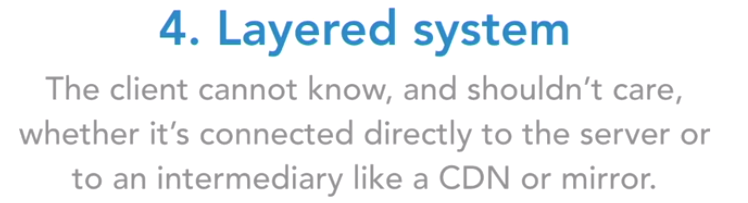


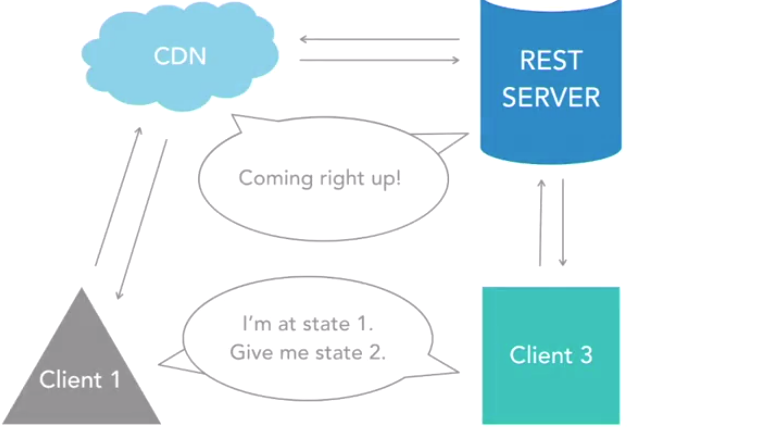


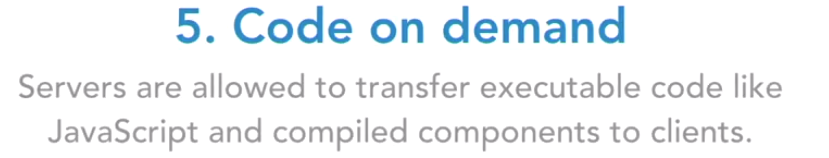


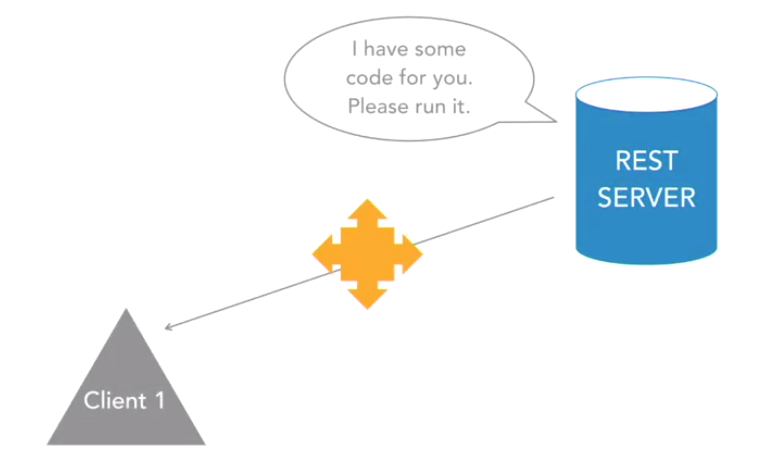












HTTP, short for HyperText Transfer Protocol, is the protocol your web browser uses to access hypertext documents on the world wide web.

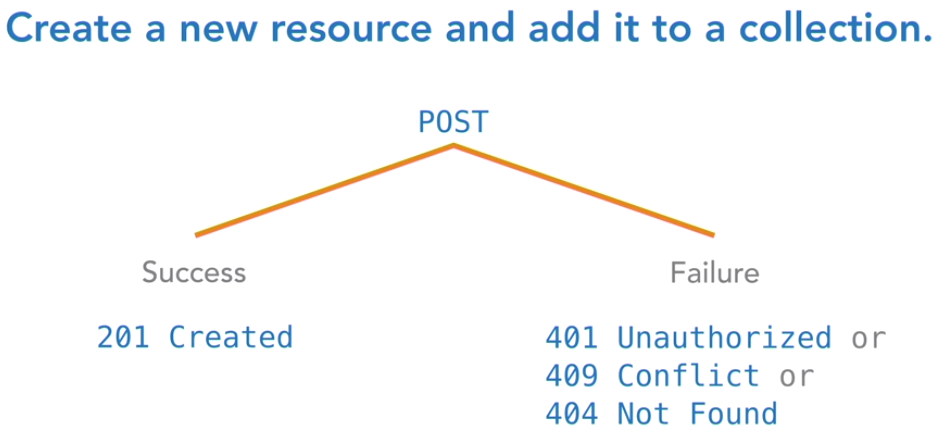
**REST and HTTP are not linked; they're just a convenient pairing. When a REST service runs on the web over HTTP to give us access to a web resource, we call it a RESTful API.**

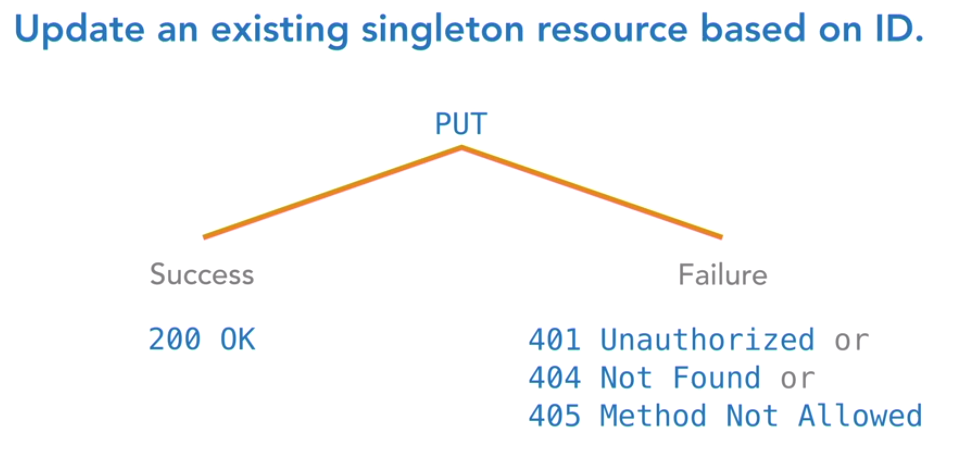
**The web platform is what makes it RESTful. In other words, if you send a request through HTTP to a REST service that meets the six constraints, that service is a RESTful API**

When explaining how REST APIs work, you'll often hear people refer to clients who consume the REST API. The question is, who are these clients? And what exactly are they consuming? As you know by now, a client in this scenario is not the human interacting with the website or app, but rather, that website or app itself. We are merely the operators of these REST clients, and the clients consume the REST API by creating and sending requests and receiving and parsing responses.

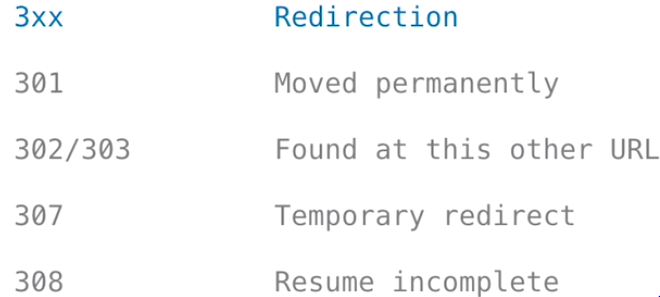














**JAX-RS 2.0 Introduction**

JAX-RS is the Java API for RESTful web services, normally shortened to just JAX-RS, and is Java's implementation of the REST architectural pattern, and provides a simple way to create RESTful APIs

