

Exercise 1

Create and consume a REST Service using Apache CXF and JAX-RS

Prior Knowledge

Basic understanding HTTP verbs, REST architecture
Some Java coding skill

Objectives

Understand what it takes to create REST services. Understand the REST model.
Interact with a REST service using simple web clients in Chrome, on the command line, and in Java.
See how Maven and Tomcat can be used.

Software Requirements

(see separate document for installation of these)

- Java Development Kit 7
- Apache Maven 3.0.4 or later
- Eclipse
- Tomcat 7.0.57 or later
- curl
- Google Chrome/Chromium plus Chrome Advanced REST extension



Step 1. Create a new project using Maven

Maven is a very powerful (and somewhat arcane) build tool. We are going to use Maven to create and build a simple RESTful service project.

Maven has the ability to create new projects using “archetypes”.

- a. First start a Unix Terminal window.
Now create a directory to store your code in.

```
mkdir oxsoa
```

- b. Change to that directory

```
cd oxsoa
```

- c. Test that you have maven properly installed. Execute

```
mvn -v
```

You should see something similar to this (dependent on your machine, JVM, etc)

```
Apache Maven 3.0.5
Maven home: /usr/share/maven
Java version: 1.7.0_65, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-7-openjdk-amd64/jre
Default locale: en_GB, platform encoding: UTF-8
OS name: "linux", version: "3.13.0-32-generic", arch: "amd64",
family: "unix"
```

- d. Use Maven to create a sample project:
Execute

```
mvn archetype:generate -Dfilter=org.apache.cxf.archetype:
```

This will prompt you for some choices:

Choose archetype:

- ```
1: remote -> org.apache.cxf.archetype:cxf-jaxrs-service
(Simple CXF JAX-RS webapp service using Spring
configuration)
2: remote -> org.apache.cxf.archetype:cxf-jaxws-javafirst
(Creates a project for developing a Web service starting
from Java code)
```

Choose a number or apply filter (format:

```
[groupId:]artifactId, case sensitive contains): :
```

Select 1 (*type '1' and hit enter!*)

Now it will ask which CXF **version** to use. The default is the latest (at the time of writing 3.1  
.2). *Just hit enter.*



Now it asks for a default **groupid**.

Define value for property 'groupId': :

This is a namespace. When creating this lab I chose *me.freo.rest*, so that is what you will see in screenshots, etc. Please stick to this, because the code I'm providing you with uses that.

Define value for property 'artifactId': :

This will be the name of the WAR and the overall maven artifact created. We are initially going to create a simple HelloWorld service, so type **HelloWorld**.

Define value for property 'version': 1.0-SNAPSHOT: :

Change this to **1.0** and hit Enter.

Define value for property 'package':  
me.freo.rest: :

This will default to the same namespace you chose for the groupId. That should be fine, so hit *Enter* to accept.

It will then ask you to confirm these settings. Hit *Enter* and it will go and generate the code.

You should see plenty of output explaining what is happening, and also a line showing where the resulting code was placed, e.g.:

```
[INFO] Parameter: artifactId, Value: HelloWorld
[INFO] project created from Archetype in dir: /home/ox-
soa/oxsoa/HelloWorld
```



This will have created a set of code and a tree structure for you. If you are on Linux you can use the nice **tree** command to show this:

```
ox-soa@oxsoa-2014:~/oxsoa$ tree
```

```
├── HelloWorld
│ ├── pom.xml
│ └── src
│ ├── main
│ │ ├── java
│ │ │ ├── me
│ │ │ └── freo
│ │ │ └── rest
│ │ │ ├── HelloWorld.java
│ │ │ └── JsonBean.java
│ │ └── webapp
│ │ ├── META-INF
│ │ │ └── context.xml
│ │ ├── WEB-INF
│ │ │ ├── beans.xml
│ │ │ └── web.xml
│ └── test
│ ├── java
│ │ ├── me
│ │ └── freo
│ │ └── rest
│ │ └── HelloWorldIT.java
```

15 directories, 7 files

e. You can now build this code:

```
cd ~/oxsoa/HelloWorld [on Linux/Mac]
mvn clean install
```

The first time this is run this will download a lot of stuff from the central maven repositories on the web. Depending how fast the network is, maybe a coffee is in order. You will need an active internet connection for this to work.

This will build **and test** the sample code. Its pretty cool. It actually starts an embedded Tomcat to run the service and call unit tests against it.

f. Assuming your build worked just fine, you now have a WAR file (Web Application aRchive) that you can deploy in Tomcat. Check that there is a file:

```
~/oxsoa/HelloWorld/target/HelloWorld-1.0.war
```

g. Tomcat is already installed on your VM, in the directory  
**~/servers/tomcat/**



h. Install your webapp (do the following all on one command line!)

```
cp ~/oxsoa/HelloWorld/target/HelloWorld-1.0.war ~/servers/tomcat/webapps
```

i. Start Tomcat so it runs on the command line (so you can see the logs)

From the tomcat directory:

```
cd ~/servers/tomcat
bin/catalina.sh run
```

j. Try your REST service

Browse

<http://localhost:8080/HelloWorld-1.0/hello/echo/paul>

You should see “paul”.

You can also try this as a command line:

```
curl http://localhost:8080/HelloWorld-1.0/hello/echo/paul
-v
```

You should see something like:

```
ox-soa@oxsoa-2014:~$ curl
http://localhost:8080/HelloWorld-1.0/hello/echo/paul -v
* Hostname was NOT found in DNS cache
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /OrderService-1.0/hello/echo/paul HTTP/1.1
> User-Agent: curl/7.35.0
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 200 OK
* Server Apache-Coyote/1.1 is not blacklisted
< Server: Apache-Coyote/1.1
< Date: Sun, 23 Nov 2014 21:02:37 GMT
< Content-Type: text/plain
< Content-Length: 4
<
* Connection #0 to host localhost left intact
paul
ox-soa@oxsoa-2014:~$
```

f. You can also build the Eclipse project for this too:

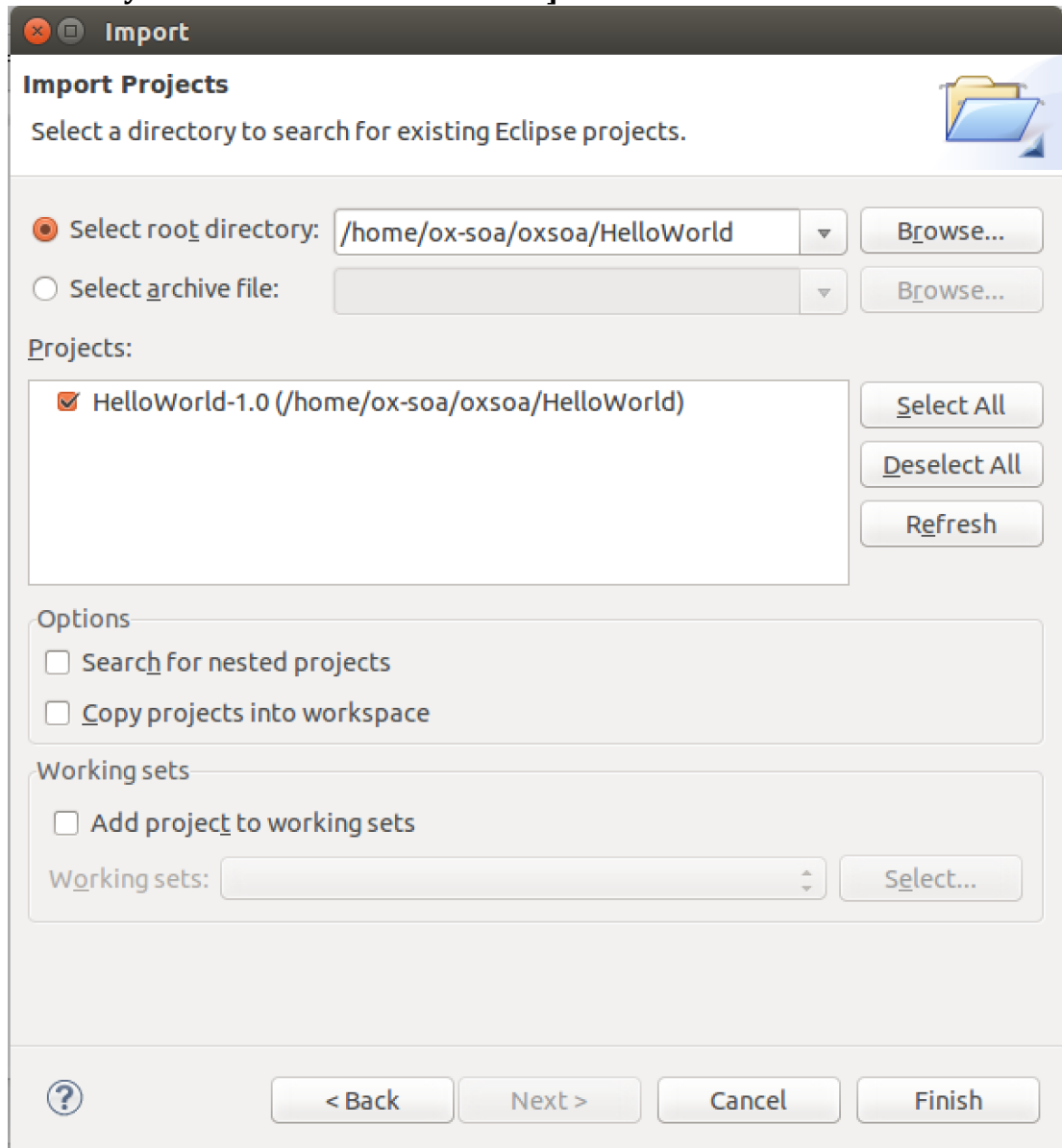
In the ~/ox-soa/HelloWorld directory type  
`mvn eclipse:clean eclipse:eclipse`

This creates a project file that you can import into Eclipse with the right classpath, settings, etc.



Now import the project. To do this, in Eclipse:

**File -> Import -> General/Existing Projects Into Workspace->[Choose the directory where HelloWorld service is]->Finish**



Now you should have the project installed in your Eclipse and be able to edit and build it. **Take a look at the sample hello service.**

Now we are ready to build our own RESTful Service.

## Step 2. Creating the OrderService

Rather than spending a lot of time writing Java, which is not the main point of this exercise, you should focus on the REST and HTTP aspects of this. I have ready written a set of Java classes that:

- 1) Implement a simple "Order Scenario"



## 2) Serialize and Deserialize as JSON

I have also written a test case that validates a service interface.

Your aim is to create a Java Service that utilizes the existing code, and is correctly annotated using JAX-RS annotations so that it meets the test case.

Here is a rough set of documentation that explains the service interface.

| Method | URI template | Description                                                                                                                                                                                                    | Supported encoding        |
|--------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|
| GET    | /orders      | Get a list of href links to available orders<br>If no orders are on the system, return an empty list.                                                                                                          | Produces application/json |
| GET    | /orders/{id} | Get back a representation of order with identifier id.<br>If no such order is yet in the system, returns HTTP Not Found<br>If the order previously existed but has been deleted, returns HTTP Gone             | Produces application/json |
| POST   | /orders      | Passes a representation of the order and create a new entry in the order database.<br>On success returns HTTP 201 Created and an HTTP Location header containing the URI of the resulting order                | Consumes application/json |
| PUT    | /orders/{id} | Updates an existing order<br>On success return HTTP 200 OK<br>If no such order is yet in the system, returns HTTP 404 Not Found<br>If the order previously existed but has been deleted, returns HTTP 410 Gone | Consumes application/json |
| DELETE | /orders/{id} | Marks an order as deleted<br>Returns HTTP 200 OK on success<br>If no such order is yet in the system, returns HTTP Not Found<br>If the order previously existed but has been deleted, returns HTTP Gone        | No body content           |



a) Unzip the code:

```
cd ~/oxsoa/
```

```
unzip ~/Downloads/OrderService-project-code.zip
```

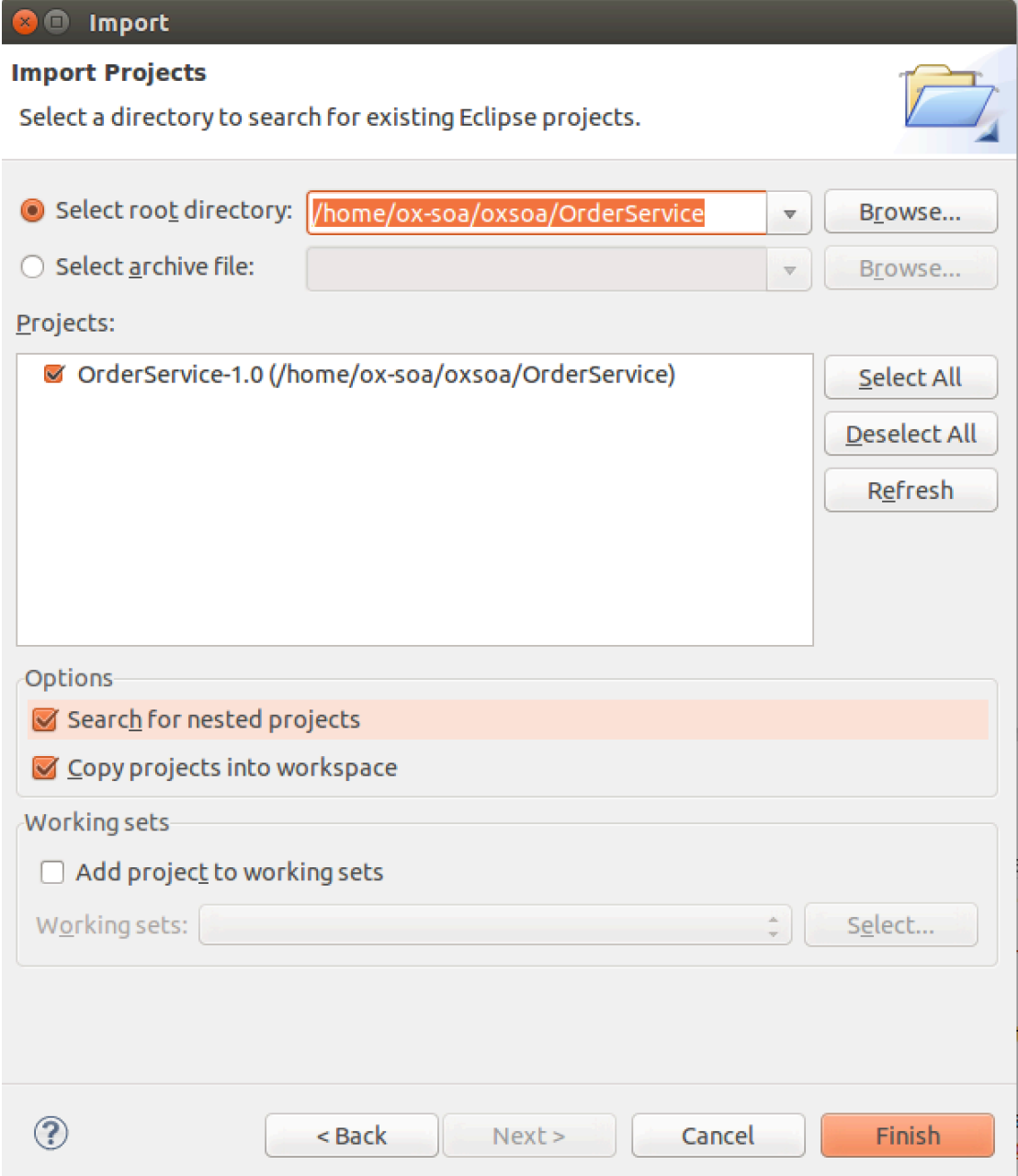
PS the file is also here: <https://github.com/pzfreo/ox-soa/blob/master/lab-exercises/code/OrderService-project-code.zip?raw=true>

b) This is an Eclipse project.

Import it into the Eclipse Workspace as you did before:







The image shows an 'Import' dialog window in Eclipse. It has a title bar with a close button, a maximize button, and the text 'Import'. The main area is titled 'Import Projects' and contains the instruction 'Select a directory to search for existing Eclipse projects.' Below this, there are two radio buttons: 'Select root directory:' (selected) and 'Select archive file:'. The 'Select root directory:' option has a text field containing '/home/ox-soa/oxsoa/OrderService' and a 'Browse...' button. The 'Select archive file:' option has an empty text field and a 'Browse...' button. Below these is a section titled 'Projects:' containing a list box with one entry: 'OrderService-1.0 (/home/ox-soa/oxsoa/OrderService)' which is checked. To the right of the list box are three buttons: 'Select All', 'Deselect All', and 'Refresh'. Below the 'Projects:' section is a section titled 'Options' containing two checked checkboxes: 'Search for nested projects' and 'Copy projects into workspace'. Below the 'Options' section is a section titled 'Working sets' containing an unchecked checkbox 'Add project to working sets' and a 'Working sets:' label followed by a dropdown menu and a 'Select...' button. At the bottom of the dialog are four buttons: a help button (question mark), '< Back', 'Next >', and 'Cancel'. The 'Finish' button is highlighted in orange.

**Import**

**Import Projects**

Select a directory to search for existing Eclipse projects.

☒ Select root directory:

☐ Select archive file:

**Projects:**

- ☒ OrderService-1.0 (/home/ox-soa/oxsoa/OrderService)

**Options**

- ☒ Search for nested projects
- ☒ Copy projects into workspace

**Working sets**

☐ Add project to working sets

Working sets:



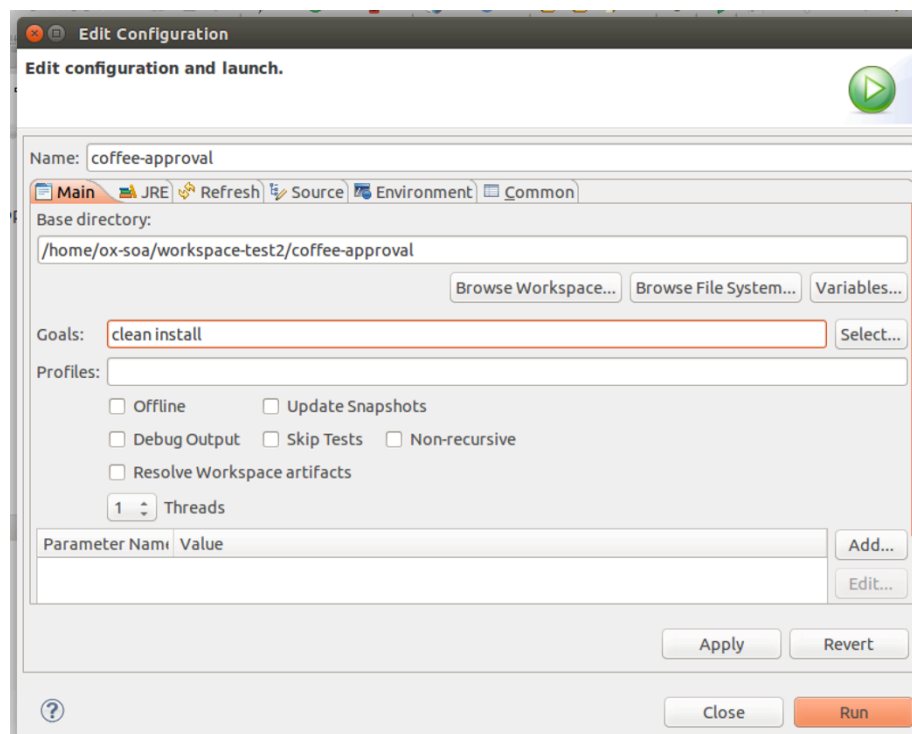
c) Build the project using **mvn clean eclipse**

*Hint: you can run mvn clean install inside Eclipse.*

Right click on the pom.xml file in your Eclipse session, **Run As -> Maven Build**.

The first time you do this, enter the **Goals** as **clean install**. Future times it will remember this. If this just runs without asking you for Goals, then the project is already set up correctly so don't worry.

The build will **FAIL**. That is on purpose. Your task is to sort it out!



d) Look at the following class in your workspace  
`me.freo.rest.OrderService`

e) Now you can incrementally add the correct methods and annotations for  
get/post/put/delete until the test case is met.



f) ONCE YOU GET YOUR BUILD WORKING:

Re-install the webapp into Tomcat. (It should hot deploy if you've left Tomcat running)

- g) Now you can test it in Chromium, using the Advanced REST Test Client.  
Start Chromium and get up a blank/empty page (Command-N or Ctrl-N)  
h) Click on the Apps link at the top of the page:

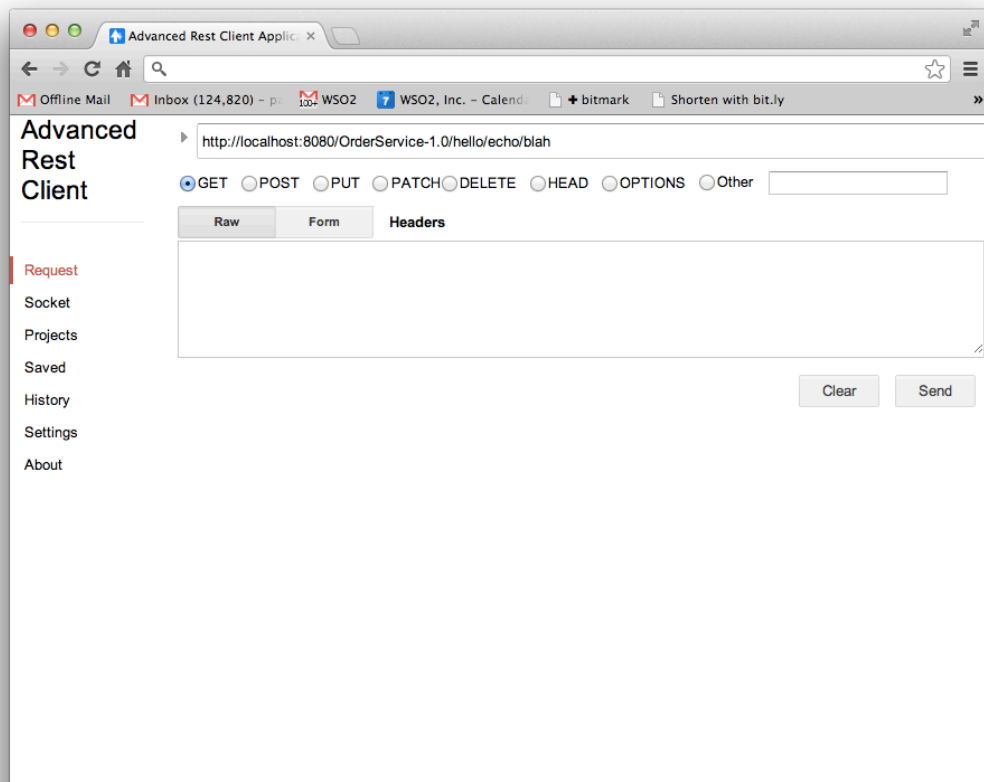


You should now see the following icon:



Advanced REST client

- i) Click on that. You should see a screen like this:



By now you should understand the REST patterns and URLs well enough to be able to test our your app. It should be available at:

<http://localhost:8080/OrderService-1.0/orders/>

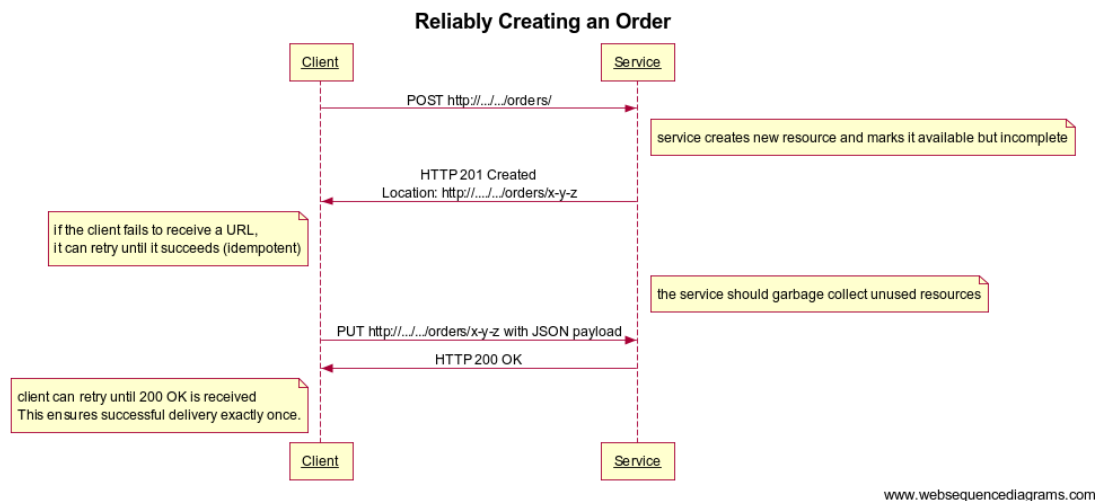
- j) Also try interacting with it using curl commands. See if you can automate posting an order.



## EXTENSION

The model we have built does not support reliability. It is impossible to know if you have successfully created a new order. To fix this, we need to implement a different pattern. In this pattern, the POST is empty, and returns a location, and only when the POST has successfully returned a Location and the client has the location, do we PUT an order into the existing location.

Here is a sequence diagram showing this model.



Implement this model by changing the code.

