RESTful Web Services

Introduction

- The RESTful Web Services are highly popular when implementing the Microservices architecture.
- The Representational State Transfer (REST) web services set a new style fro SOA systems.
- The term Representational State Transfer was introduced in 2000 by Roy Fielding in his doctoral dissertation.

The RESTful Services Architecture

- REST stands for Representational State Transfer. It is a web service architecture that focuses on the system resources.
- Each resource is identified by a URI (Unified Resource Identifier).
- Accessing the resources is done using HTTP. The server reply is the representation of the resource we try to access. This representation is usually a JSON document.

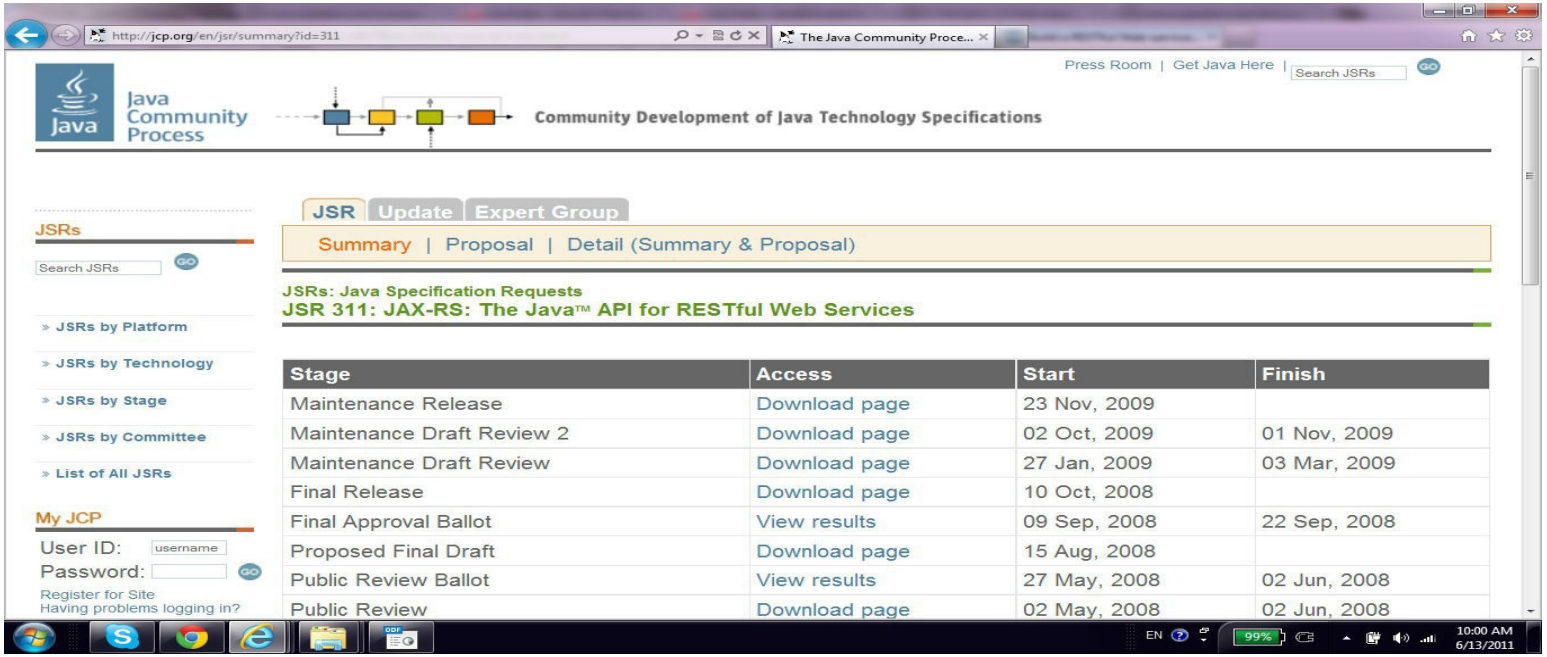
The RESTful Services Architecture

- Web Services that use the REST architecture are called RESTful Services.
- Most RESTful services are being used via a simple HTTP GET request for which they reply with a simple JSON document.

Java API for RESTful Web Services

- JSR 311 specifies Java API for RESTful web services. It aims at allowing us to develop RESTful web services in a standard way.

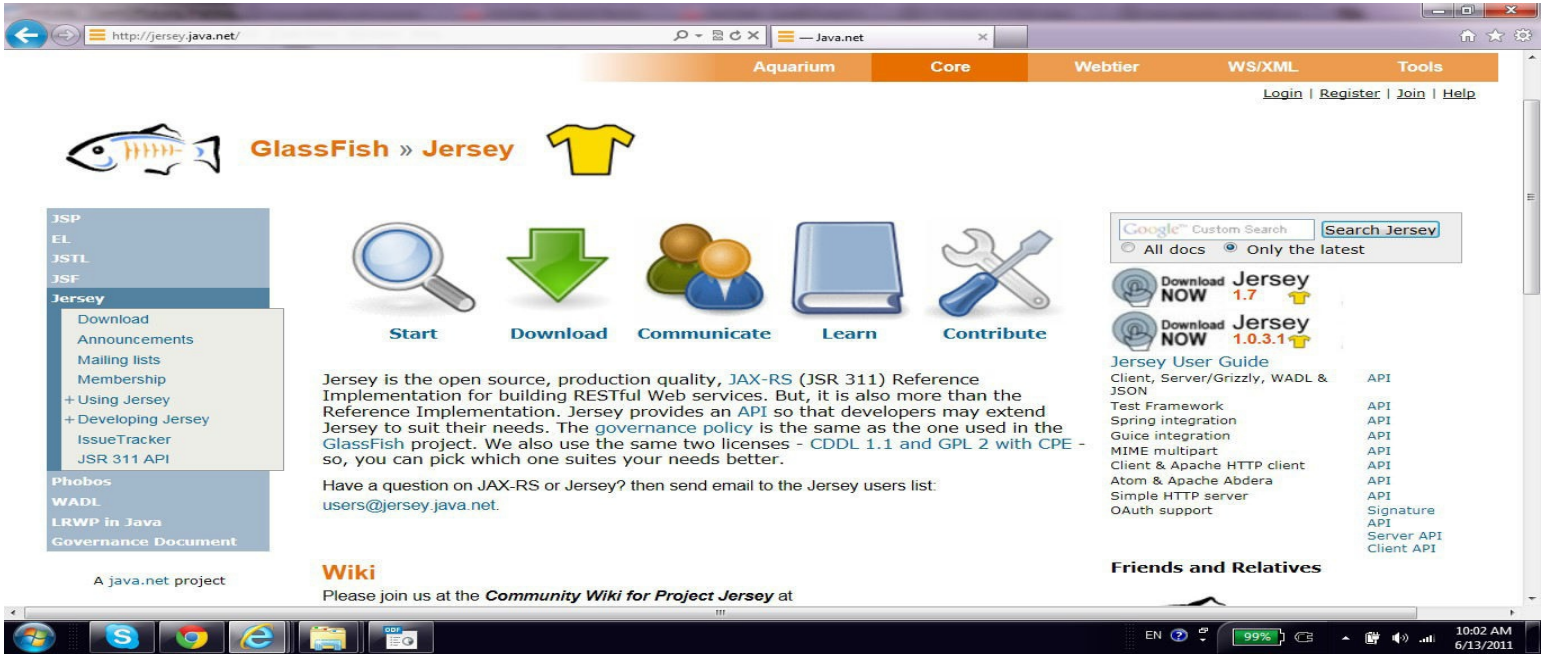
Java API for RESTful Web Services



Jersey Project

- Jersey is the reference implementation for JSR 311. We can easily integrate Jersey with Tomcat in order to develop a RESTful web service.

Jersey Project



Download Jersey

- You can easily download Jersey latest version browsing at

  http://download.java.net/maven/2/com/sun/jersey/jersey-archive/1.7/jersey-archive-1.7.zip

- In order to deploy a simple Jersey based REStful

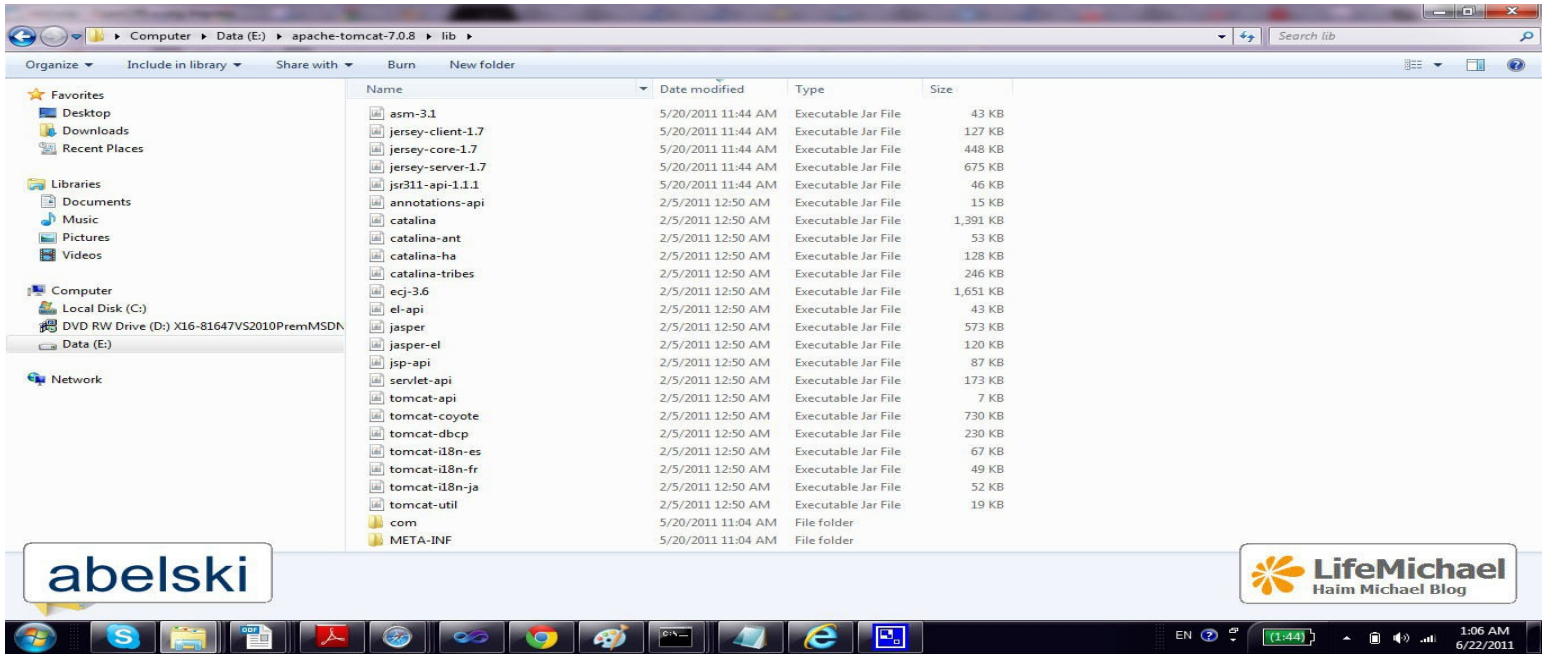
  web service we will need the following files:

```
jersey-core.jar
jersey-server.jar
jsr311-api.jar
asm.jar
```

Install Jersey

- We should first copy these jar files into the `lib` folder of our Tomcat installation.

Install Jersey



Deployment

- We should define the Jersey servlet dispatcher in our

  `web.xml` file.

Deployment

```
<servlet>
    <servlet-name>My Jersey REST Service</servlet-name>
    <servlet-class>
        com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
        <param-name>
            com.sun.jersey.config.property.packages
        </param-name>
        <param-value>
            com.abelski.samples
        </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>My Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

this param defines the package where jersey should look for the classes we defined

we should add it into our web.xml configuration file

Deployment

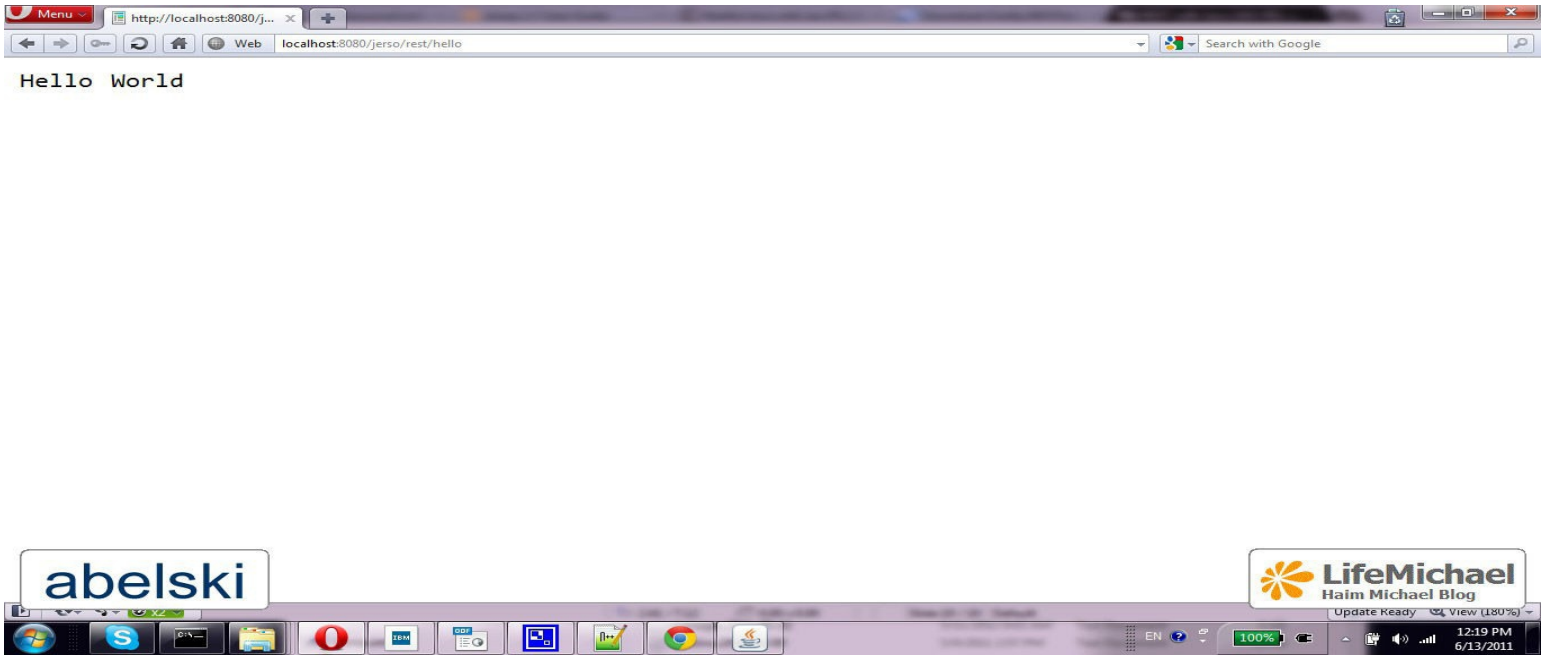
- We can develop the class that will be instantiated and function as a RESTful web service.

Deployment

```
package com.abelski.samples;

import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
@Path("/hello")
public class HelloWorld
{
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayHello()
    {
        return "Hello World";
    }
}
```

Deployment


Menu
http://localhost:8080/j...
Web
localhost:8080/jerso/rest/hello
Search with Google
Hello World
abelski
LifeMichael
Haim Michael Blog
Update Ready
View (180%)
EN
100%
12:19 PM
6/13/2011

Query String Parameters

- We can easily extract the value of a specific query string parameter by using the `@QueryParam` annotation.

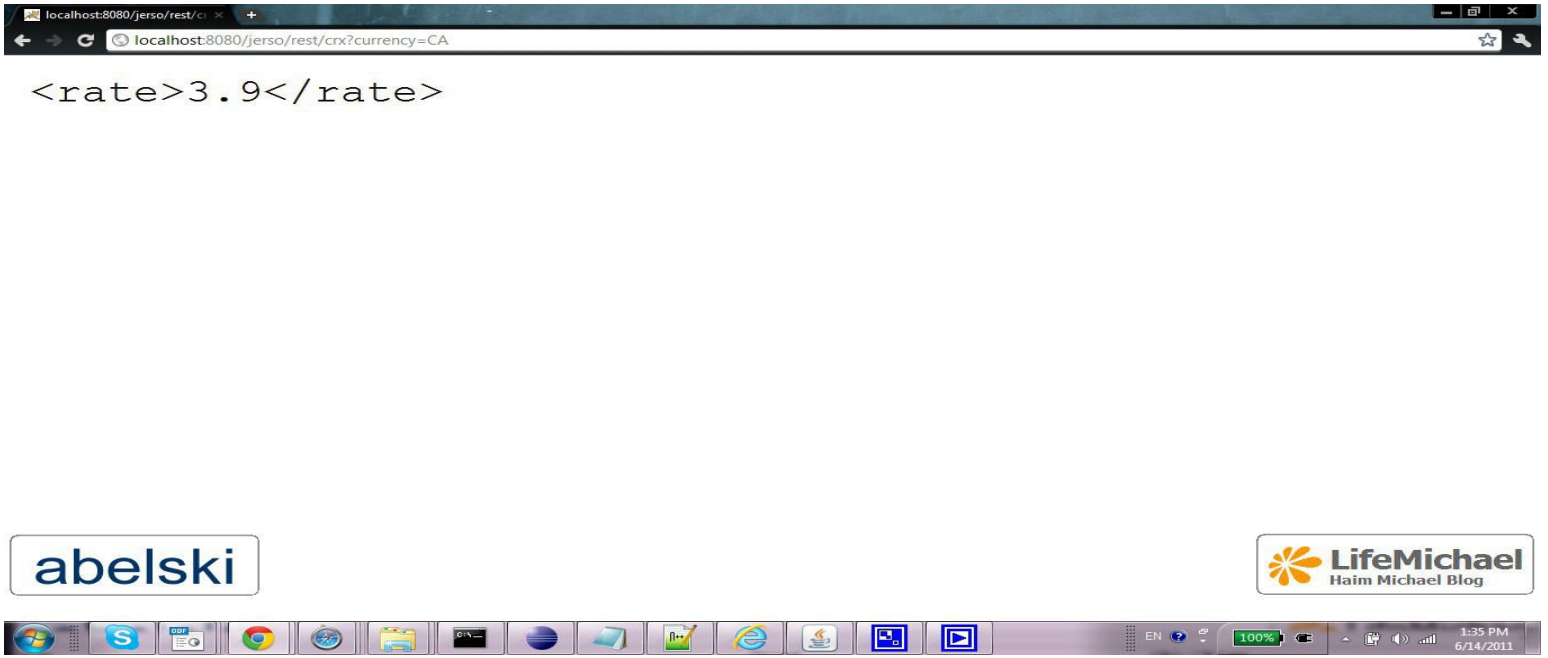
Query String Parameters

```
@Path("/crx")
public class CurrenciesExchangeRate
{
        @GET
        @Produces(MediaType.TEXT_PLAIN)
        public String getCurrencyExchangeRate(
                @DefaultValue("ILS") @QueryParam("currency")
                String currencyName)
        {
        double reply = 1;

        if(currencyName.equals("ILS")) reply = 1;
        if(currencyName.equals("USD")) reply = 3.4;
        if(currencyName.equals("EUR")) reply = 4.9;
        if(currencyName.equals("GBP")) reply = 6.5;
        if(currencyName.equals("CA")) reply = 3.9;
        return "<rate>"+reply+"</rate>";
    }
}
```

YouTube

Query String Parameters



Path Segments

- We can easily associate different methods with the various URL possibilities. In other words, we can specify different methods to be invoked in according to the URL path that was used for using the web service.

Path Segments

```
@Path("/crx")
public class CurrenciesExchangeRate
{
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getCurrencyExchangeRate(
        @DefaultValue("ILS")
        @QueryParam("currency") String currencyName)
    {
        double reply = 1;
        if(currencyName.equals("ILS")) reply = 1;
        if(currencyName.equals("USD")) reply =
        3.4; if(currencyName.equals("CA")) reply =
        3.9; return "<rate>"+reply+"</rate>";
    }
    @GET
    @Path("/count")
    @Produces(MediaType.TEXT_PLAI)
    public String getCount() {
        return "<gogo>99</gogo>";
    }
}
```

Path Segments