Vamos a construir nuestro primer ejemplo de Java EE MicroServices usando Payara como servidor de aplicaciones . Payara este basado en Glassfish y esta cogiendo tracción en este mundo nuevo de los microservicios. El primer paso que tenemos que hacer es descargarnos la versión de Payara para microservices Básicamente lo que estaremos descargando es un simple fichero jar. Este servidor es capaz de ejecutar cualquier aplicación en formato war . Vamos a construir nuestro primer **Microservicio sobre Java EE.**

```
package com.arquitecturajava.recursos.ejemplo1;

import java.util.ArrayList;
import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

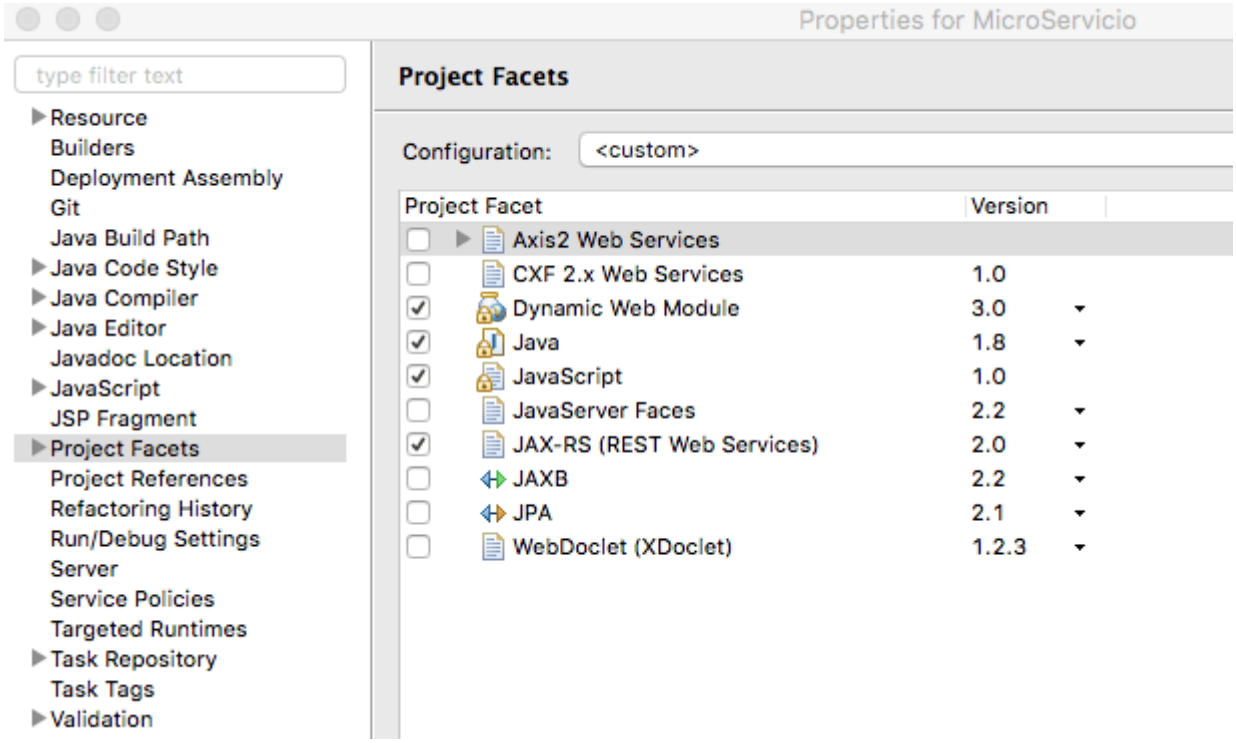
import com.arquitecturajava.Libro;

@Path(&quot;libros&quot;)
public class RecursoLibros {
        @GET
        @Produces(&quot;application/json&quot;)
        public List&lt;Libro&gt;  getLibros() {
                Libro l1= new Libro(&quot;El juego de
ender&quot;,500);
                Libro l2= new Libro(&quot;Dune&quot;,100);
                List&lt;Libro&gt; misLibros= new
ArrayList&lt;Libro&gt;();
                misLibros.add(l1);
```

```
                misLibros.add(l2);
                return misLibros;
        }
}
```

Creando Java EE MicroServices

Hemos usado la anotación @Path para dar de alta la ruta “libros” y la anotación @GET para que responda a la petición GET habitual devolviendo un listado de libros. La anotación @Produces nos devuelve los datos en formato JSON. El uso de estas anotaciones nos generará errores a nivel de la aplicación web en Eclipse ya que no las encuentra. Para solventarlo modificaremos las propiedades del proyecto y añadiremos soporte para el estandar de JAX-RS en su versión 2.



Una vez hecho esto todas las anotaciones serán aceptadas por Eclipse. Nos queda por

mostrar el código fuente de la clase Libro.

```
ackage com.arquitecturajava;

import java.io.Serializable;

import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement(name = &amp;quot;libro&amp;quot;)
public class Libro implements Serializable {

	private String titulo;
	private int paginas;
	public String getTitulo() {
		return titulo;
	}
	public void setTitulo(String titulo) {
		this.titulo = titulo;
	}
	public int getPaginas() {
		return paginas;
	}
	public void setPaginas(int paginas) {
		this.paginas = paginas;
	}
	public Libro() {
		super();
	}
	public Libro(String titulo, int paginas) {
		super();
```

```
                this.titulo = titulo;
                this.paginas = paginas;
        }
}
```

Como podemos observar la clase incluye la anotación @XmlRootElement que permitirá que el servidor pueda serializar correctamente los diferentes libros a formato JSON. El último paso es registrar nuestro servicio.

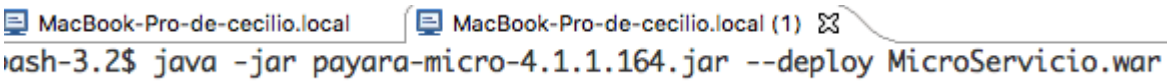
```
package com.arquitecturajava;


import java.util.Set;

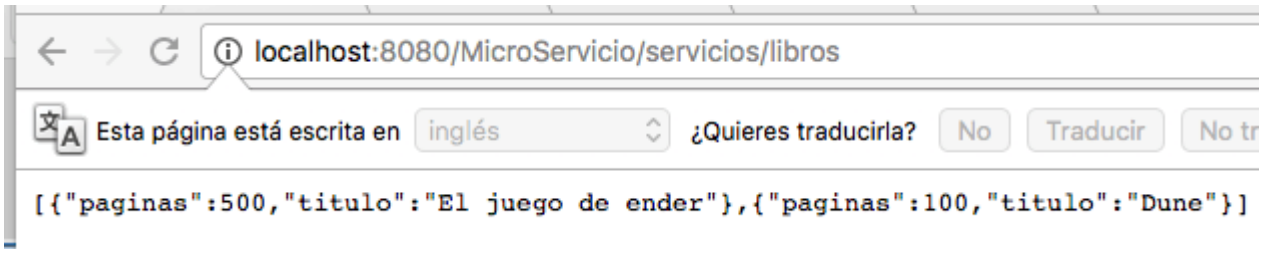
import javax.ws.rs.core.Application;
@javax.ws.rs.ApplicationPath(&amp;quot;servicios&amp;quot;)
public class MicroServicioAplicacion extends Application {
    @Override
    public Set&amp;lt;Class&amp;lt;?&amp;gt;&amp;gt; getClasses() {
        Set&amp;lt;Class&amp;lt;?&amp;gt;&amp;gt; resources = new
java.util.HashSet&amp;lt;&amp;gt;();
        addRestResourceClasses(resources);
        return resources;
    }
    private void addRestResourceClasses(Set&amp;lt;Class&amp;lt;?&amp;gt;&amp;gt;
resources) {
resources.add(com.arquitecturajava.recursos.ejemplo1.RecursoLibros.cla
ss);
```

```
    }
}
```

Ya tenemos el Microservicio creado , nos queda exportar el proyecto como war y pasarlo como parámetro a Payara server para que lo lance .



La instrucción a nivel de linea de comandos es así de sencilla , nos vale con lanzar el servidor y pasar como parámetro nuestra aplicación. Ya podemos acceder al microservicio.



Payara se esta haciendo un hueco en el mundo de Java EE MicroServices hay que estar atento a su evolución.

Otros artículos relacionados : ¿Qué es Spring Boot? , Reactive MicroServices y Arquitectura , ¿Qué es Docker y para qué sirve?