

SERVICIOS REST

JERSEY JAX-RS



YO

```
{  
    name : 'Ricardo Borillo',  
    company : 'Universitat Jaume I',  
    mail : 'borillo@uji.es',  
    social : {  
        twitter : '@borillo',  
        blog : 'xml-utils.com',  
        linkedin : 'linkedin.com/in/borillo'  
    }  
}
```

YO



ÍNDICE



- Servicios web
- HTTP y REST
- Uso
- Jersey JAX-RS
- Testing
- El futuro de JAX-RS

SERVICIOS WEB

WIKIPEDIA:

"Un servicio web es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet."

DISTINTAS APROXIMACIONES:

SOAP. Formato XML e independiente del protocolo.
Implementado en Metro (JAX-WS)

"SOAP (Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML"

DISTINTAS APROXIMACIONES:

REST. Cualquier formato sobre HTTP. Implementado en Jersey (JAX-RS)

"Cualquier interfaz web simple sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes"

LA GRAN VENTAJA DE REST APROVECHA AL MÁXIMO LA INFRAESTRUCTURA DE HTTP

Simplicidad, escalabilidad, cacheo, seguridad, ...

REST != RPC

Evitar cosas como:

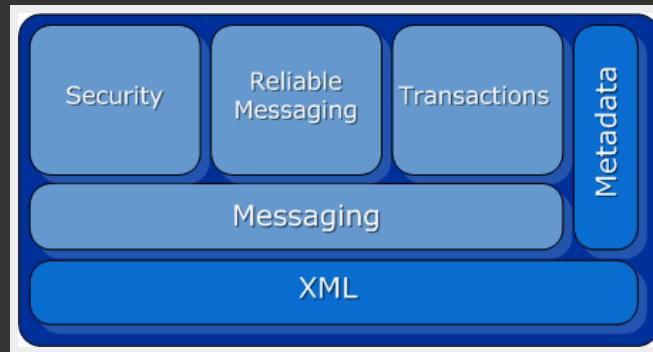
```
/getUsuario  
/getAllAsuarios  
/modificaCuentaById
```

En su lugar utilizamos nombres que definen recursos:

```
/usuarios  
/usuarios/1  
/usuarios/1/facturas
```

NO USAMOS SOAP PORQUE:

- Complejo y difícil de mantener (WSDL)
- Necesidad de un framework también en el cliente
- Problemas de interoperabilidad
- Problemas para ser usado desde móviles o JavaScript
- No aprovecha al 100% HTTP
- Muchísimos estándares a conocer: ws-*



WS -



<http://www.loudthinking.com/arc/000585.html>

flickr™



Google™

REST



HTTP Y REST

HTTP: ACCEDIENDO A CONTENIDOS EN LA WEB

A través de un navegador web:

- Consulta de páginas
- Envío de formularios
- Subir ficheros al servidor

HTTP: ACCEDIENDO A CONTENIDOS EN LA WEB

Desde línea de comandos

```
curl -XGET http://www.google.es/
```

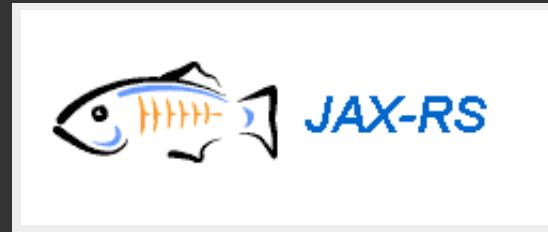
Desde algún lenguaje de programación como Java

```
DefaultHttpClient httpclient = new DefaultHttpClient();
HttpGet httpGet = new HttpGet("http://www.google.es/");
HttpResponse response = httpclient.execute(httpGet);
```

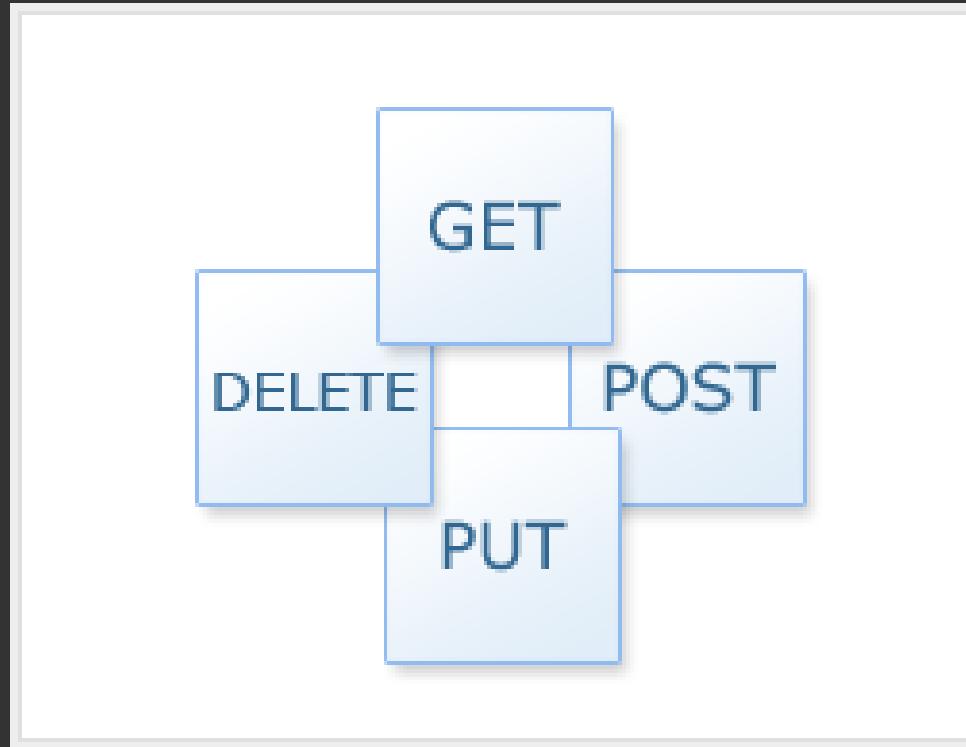
HTTP: ACCEDIENDO A CONTENIDOS EN LA WEB

O con el API cliente de Jersey JAX-RS:

```
Client client = Client.create();
WebResource webResource = client.resource("http://www.google.es/");
ClientResponse response = webResource.accept("text/html").get(ClientResponse.class);
if (response.getStatus() == 200) {
    System.out.println(response.getEntity(String.class));
}
```

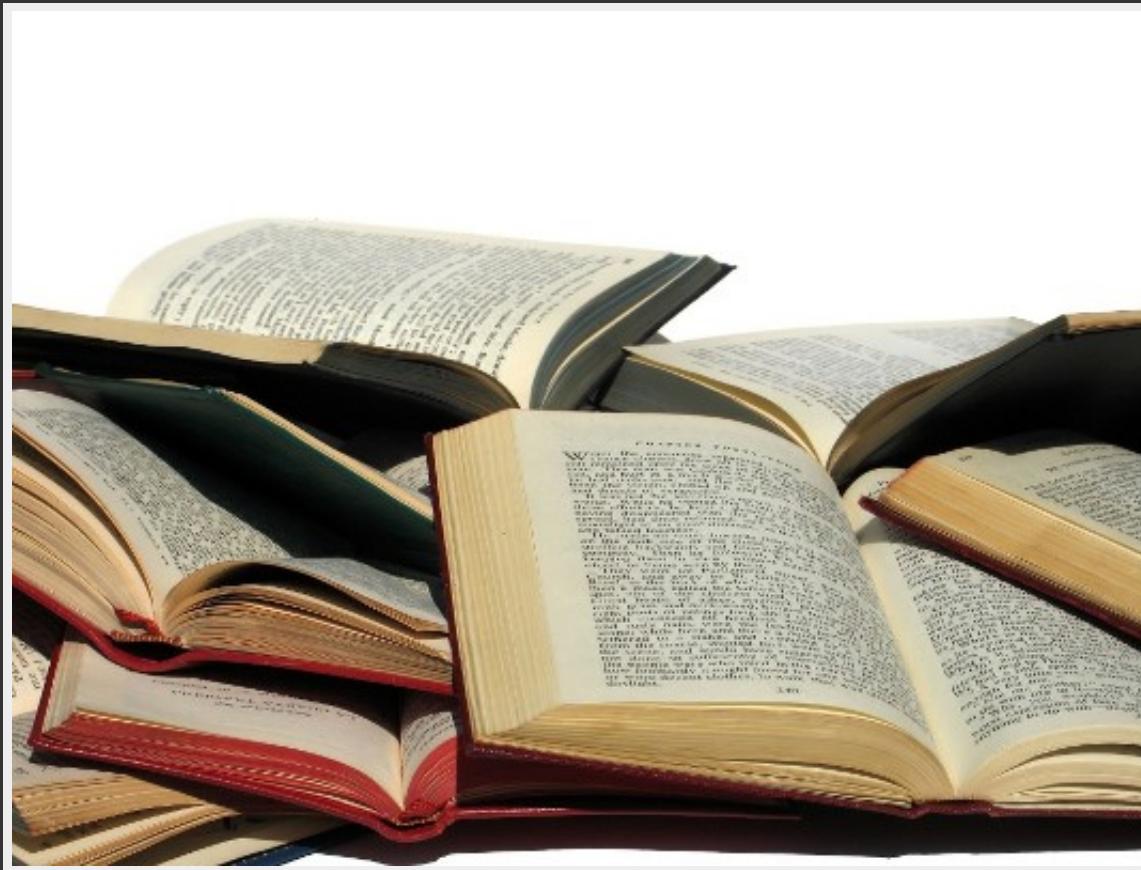


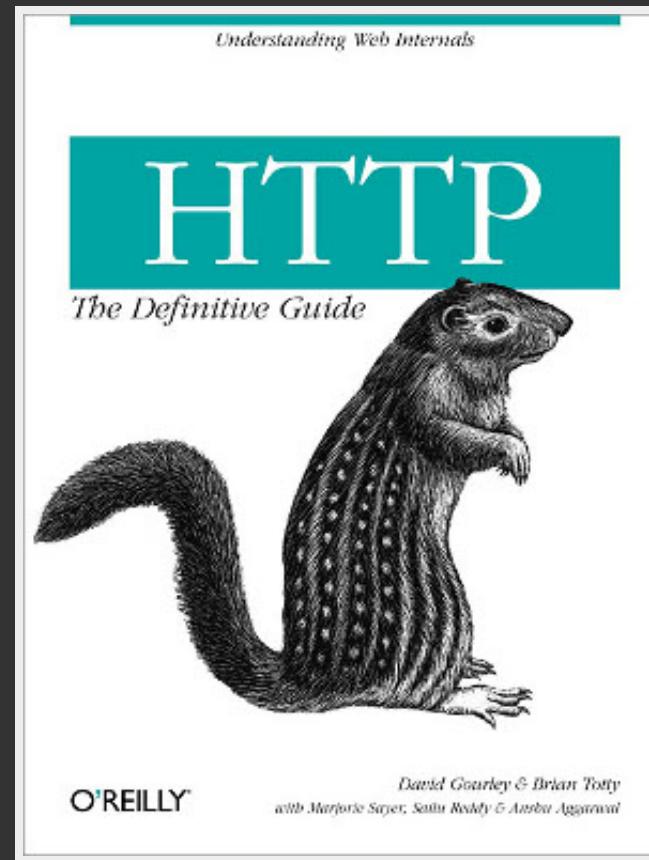
REST APROVECHA LA ARQUITECTURA DE LA WEB:



REST ESTÁ ORIENTADO A RECURSOS. ESTOS SON NOMBRES, NO VERBOS. ACCIONES:

- **GET**: Recuperación de un recurso
- **POST**: Creación de un nuevo recurso
- **PUT**: Modificación de un recurso o creación si ya conocemos la clave
- **DELETE**: Eliminación de un recurso
- **HEAD**: Envío sólo de cabeceras. Muy utilizada para comprobar el Last-Modified.





EJEMPLOS:

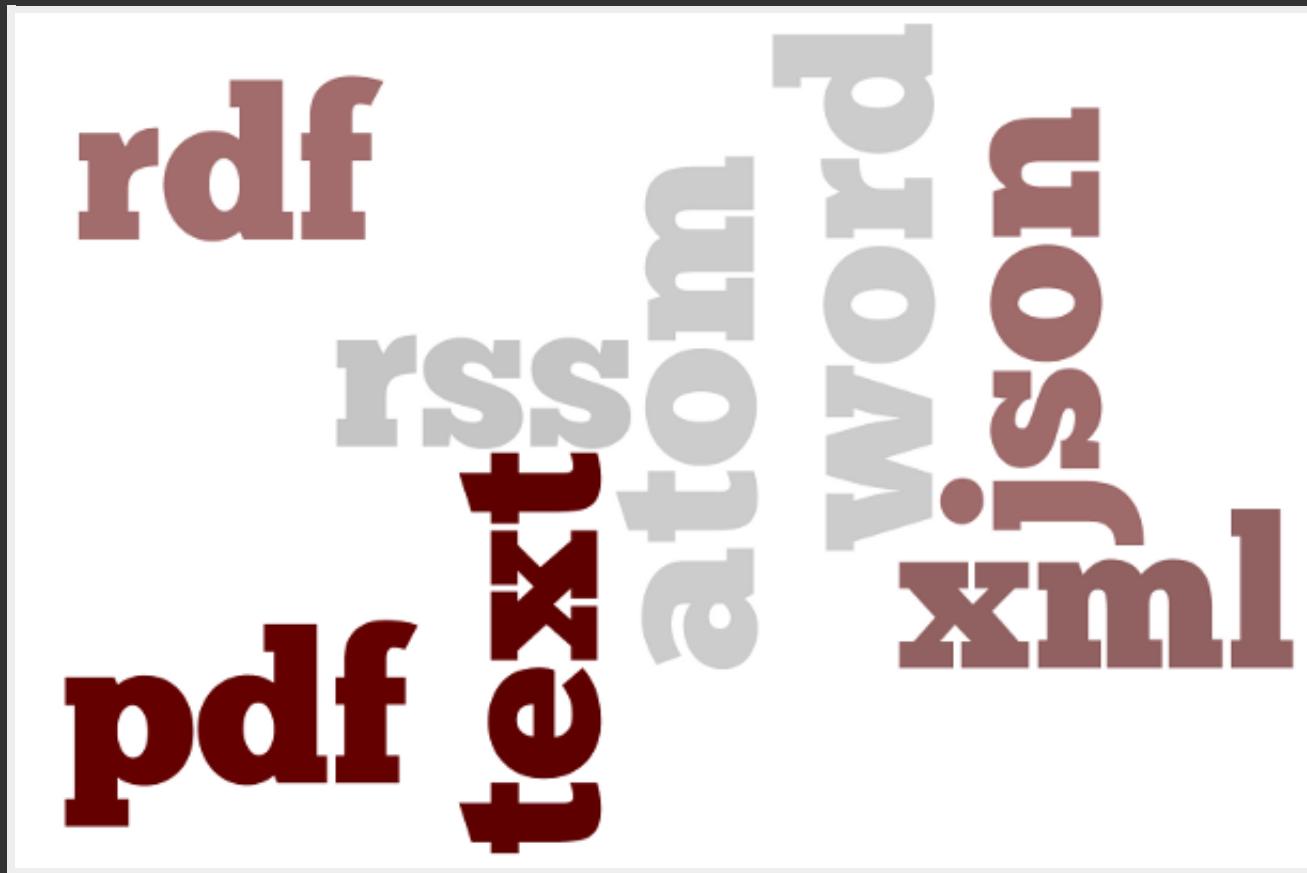
Lista todos los coches o recupera uno

```
GET /cars  
GET /cars/1234AAW
```

Añade, modifica o elimina un coche

```
POST /cars  
PUT /cars/1234AAW  
DELETE /cars/1234AAW
```

INDEPENDIENTE DEL FORMATO INTERCAMBIADO:



USO

DISEÑO DE UN API

- Nos centramos en nuestro negocio: Buen diseño del software
- Exposición de nuestro modelo como recursos
- Diseño modular
- Soporte de varios formatos y operaciones

INDEPENDENCIA DE LA CAPA CLIENTE:



JERSEY JAX-RS

¿QUÉ ES?

Jersey es la implementación Java de referencia del estándar JAX-RS para la definición de servicios REST:

<https://jersey.dev.java.net/>



¿QUÉ ES?

OBJETIVOS GENERALES:

- Definir servicios en forma de POJOs
- Mapear peticiones y respuestas HTTP a esos POJOs
- Independencias del formato de transferencia
- Independiente del contenedor
- Parte de Java EE

¿QUÉ ES?

Mapear peticiones HTTP a código Java

@GET / @POST / @PUT / @DELETE

```
@Path("users")
public class UsersResource
{
    @GET
    public List<User> getUsers() {
        ...
    }
}
```

```
GET /users
```

¿QUÉ ES?

Mapear parámetros de URL a parámetros de entrada a los métodos

@PathParam / @QueryParam

```
@GET  
@Path("/users/{userId}")  
public User getUser(  
    @PathParam("userId") String userId,  
    @QueryParam("debug") @DefaultValue("5") String debug)  
{  
}
```

```
GET /users/1421?debug=S
```

¿QUÉ ES?

Declaración del formato de los contenidos recibidos o emitidos

@Consumes / @Produces

```
@GET  
@Produces(MediaType.APPLICATION_XML)  
public List<User> getUsers() {  
}  
  
@PUT  
@Consumes(MediaType.APPLICATION_JSON)  
public void updateUser(User user) {  
}
```

MAPEO DE LA PETICIÓN HTTP:

GET /recurso/base?param=valor HTTP/1.1
Host: www.uji.es
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; es-CL; rv:1.9.2.10) Gecko/20100922
Ubuntu/10.10 (maverick) Firefox/3.6.10
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-cl,es;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive

@GET **@PathParam** **@QueryParam**

@PathParam

@Produce

MAPEO DE LA RESPUESTA HTTP:

The diagram illustrates the mapping of an HTTP response header to its content type. A red arrow labeled "Response" points from the header "Content-Type: text/html" to the word "text/html". Another red arrow labeled "@Produce" points from the same header to the word "text/html". Below the header, the content type "text/html" is shown in blue. The content itself is represented by the text "<html> ... </html>" in blue, with a red arrow labeled "Contenido de la respuesta" pointing to it.

HTTP/1.1 200 OK

Content-Type: text/html

Server: Oracle-Application-Server-10g/10.1.2.3.0

Content-Length: 25671

Date: Thu, 14 Oct 2010 17:05:03 GMT

<html>

...

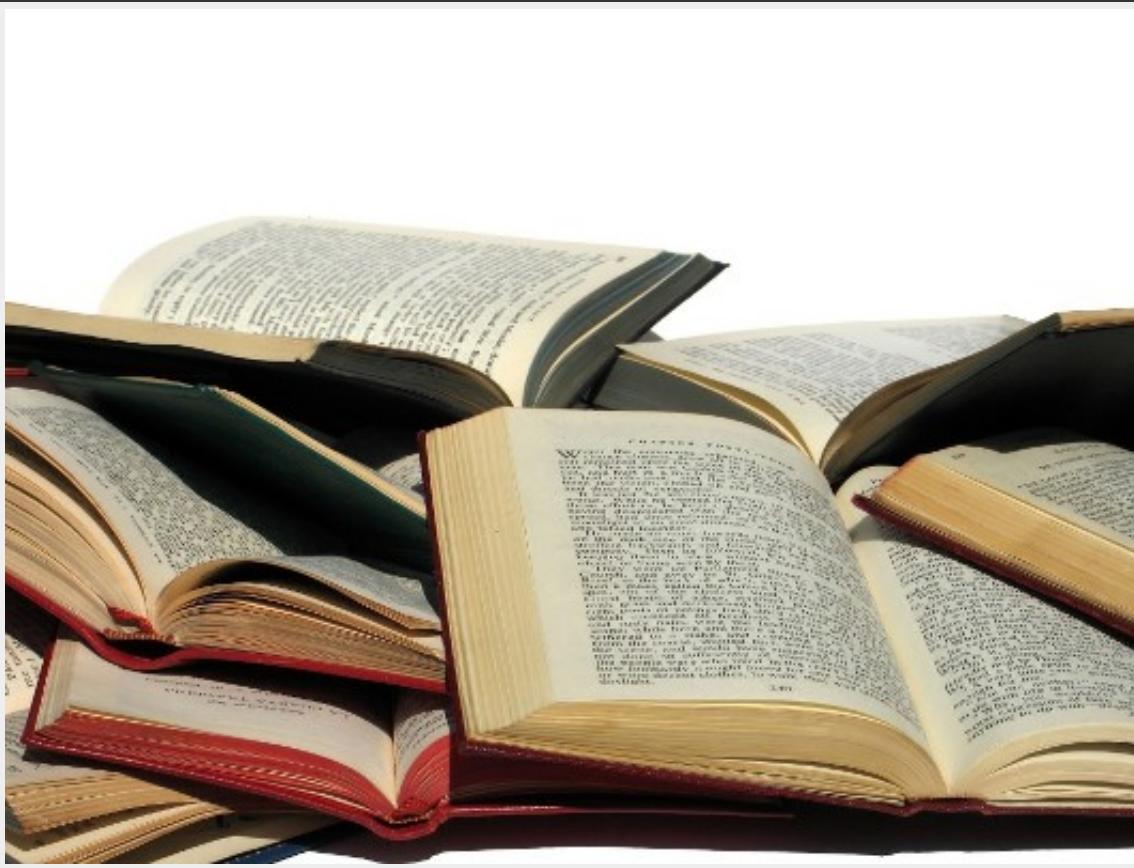
</html>

CÓDIGOS DE RESPUESTA HTTP:

- **1xx** Informativos
- **2xx** De éxito (200 OK, 201 Created, 202 Accepted, 204 No Content)
- **3xx** Redirecciones (301 Moved Permanently, 302 Found, 304 Not Modified)
- **4xx** Error en cliente (400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 405 Method Not Allowed)
- **5xx** Error en servidor (500 Internal Server Error, 501 Not Implemented, 502 Bad Gateway, 503 Service Unavailable)

GENERACIÓN DE DISTINTOS FORMATOS:

- **XML/JSON** JAXB y Jackson
- **ATOM** Apache Abdera
- **Personalizado** MessageBodyReader y MessageBodyWriter









CONFIGURACIÓN DE UNA APLICACIÓN WEB JERSEY

Añadiendo las dependencias al proyecto descargándolas desde:

<http://jersey.dev.java.net/>

Ejemplo:

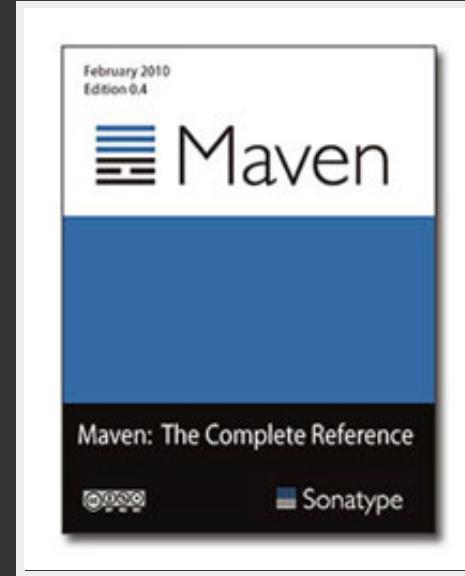
<https://github.com/borillo/java-roadshow-rest/tree/master/examples/jersey-simple>

CONFIGURACIÓN DE UNA APLICACIÓN WEB JERSEY

Usando Maven:

```
<dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-server</artifactId>
    <version>1.14</version>
</dependency>

<dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-servlet</artifactId>
    <version>1.14</version>
</dependency>
```



Ejemplo:

<https://github.com/borillo/java-roadshow-rest/tree/master/examples/jersey-maven>

CONFIGURACIÓN DE UNA APLICACIÓN WEB JERSEY

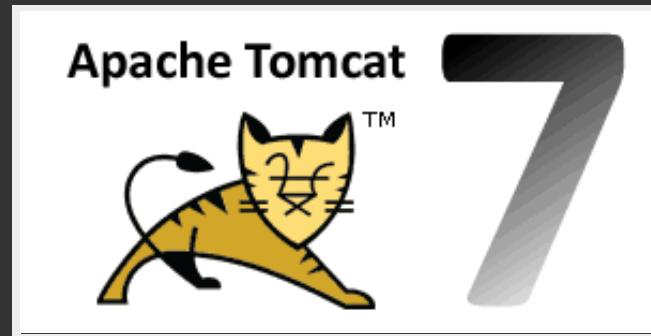
Definición del web.xml:

```
<web-app>
  <servlet>
    <servlet-name>jaxrs-servlet</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.decharlas.test.services</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>jaxrs-servlet</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

CONFIGURACIÓN DE UNA APLICACIÓN WEB JERSEY

Si usamos Servlet 3.0:

```
@ApplicationPath("rest")
public class MyApplication extends PackagesResourceConfig {
    public MyApplication() {
        super("com.decharlas.test.services");
    }
}
```



EJEMPLO BÁSICO:

```
package com.decharlas.test.services;

@Path("users")
public class UserResource {
    @GET
    public String sayHello() {
        return "Hello world!!!";
    }
}
```

Acceso desde el navegador o línea de comandos:

```
GET -XGET http://localhost:8080/users
```

INICIO DE LA APLICACIÓN:

Eclipse "Run on server" o jetty:run en Maven:

```
27-oct-2012 20:04:31 com.sun.jersey.api.core.PackagesResourceConfig init
INFO: Scanning for root resource and provider classes in the packages:
      com.fasterxml.jackson.jaxrs.json
      com.decharlas.services
27-oct-2012 20:04:31 com.sun.jersey.api.core.ScanningResourceConfig logClasses
INFO: Root resource classes found:
      class com.decharlas.services.UserResource
27-oct-2012 20:04:31 com.sun.jersey.api.core.ScanningResourceConfig logClasses
INFO: Provider classes found:
      class com.fasterxml.jackson.jaxrs.json.JacksonJaxbJsonProvider
      class com.fasterxml.jackson.jaxrs.json.JsonParseExceptionMapper
      class com.fasterxml.jackson.jaxrs.json.JsonMappingExceptionMapper
      class com.fasterxml.jackson.jaxrs.json.JacksonJsonProvider
INFO: Initiating Jersey application, version 'Jersey: 1.14 06/29/2012 05:14 PM'
```

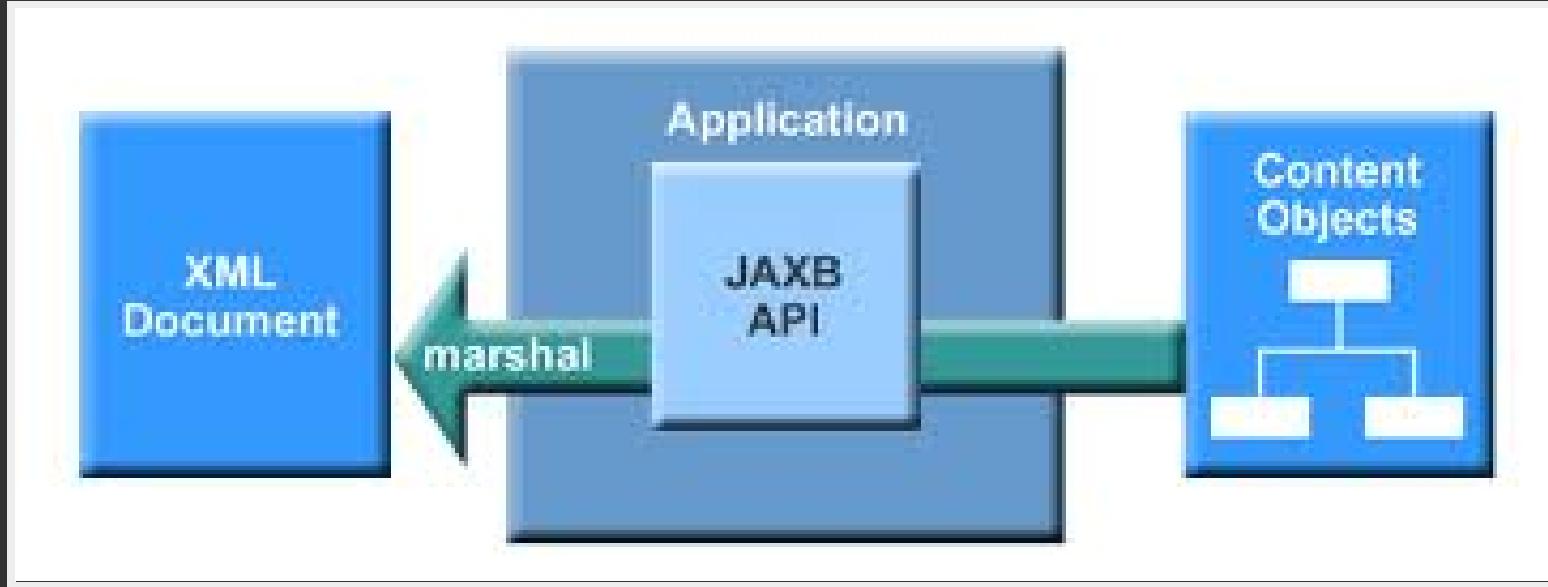


EJEMPLO JSON:

Partimos de los datos datos en forma de Java Bean:

```
@XmlElement  
public class User {  
    private Long id;  
    private String name;  
}
```

El servicio REST los serializa con JAXB o Jackson



También funciona para JSON
Natural JSON con Jackson

JSON serializado por JAXB:

```
{  
    "user": [{  
        "id": "1",  
        "name": "Pedro Perez Pardo"  
    }]  
}
```

JSON serializado por Jackson:

```
[{  
    "id": 1,  
    "name": "Pedro Perez Pardo"  
}]
```

USO DE JACKSON

CONFIGURACIÓN EXTRA DEL WEB.XML

```
<servlet>
    <servlet-name>jaxrs-servlet</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.fasterxml.jackson.jaxrs.json; ...</param-value>
    </init-param>
    ...
</servlet>
```

EJEMPLO JSON Y XML:

```
@Path("users")
public class UserResource {
    @GET
    @Produces({ "application/json", "text/xml" })
    public User getUser() {
        return new User(1, "Usuario 1");
    }
}
```

```
GET -XGET -H "Accept: application/json" http://localhost:8080/users
GET -XGET -H "Accept: text/xml" http://localhost:8080/users
```



CLIENTE DE ACCESO AL SERVICIO JSON:

```
@Path("users")
public class UserResource {
    @GET
    @Path("{id}")
    @Produces({ "application/json", "text/xml" })
    public User getUser(@PathParam("id") String id) {
        return new User(id, "Usuario 1");
    }
}
```

```
Client client = Client.create();
WebResource resource = client.resource("http://localhost:8080/jersey-maven/rest");
ClientResponse response = resource.path("/users/1").get(ClientResponse.class);
User user = response.getEntity(User.class)
```

BÚSQUEDA EN TWITTER:

Recuperación del resultado como un JSONObject:

```
Client client = Client.create();
client.addFilter(new LoggingFilter());

WebResource resource = client.resource("http://search.twitter.com/");

ClientResponse response = resource.path("search.json")
    .queryParam("q", "oracle+roadshow").get(ClientResponse.class);
JSONObject data = response.getEntity(JSONObject.class);

Assert.assertEquals(Status.OK.getStatusCode(), response.getStatus());
Assert.assertEquals(1, data.getInt("page"));
```

GESTIÓN DE ERRORES

Uso del intefaz ExceptionMapper para interceptar excepciones:

```
@Provider  
public class CommonExceptionMapper implements ExceptionMapper<Exception> {  
    @Override  
    public Response toResponse(Exception exception) {  
        log.error("Handled by ExceptionMapper", exception);  
        return Response.serverError().type(MediaType.TEXT_PLAIN)  
            .entity(exception.getMessage()).build();  
    }  
}
```

Los **@Provider** permiten interactuar con el ciclo petición/respuesta

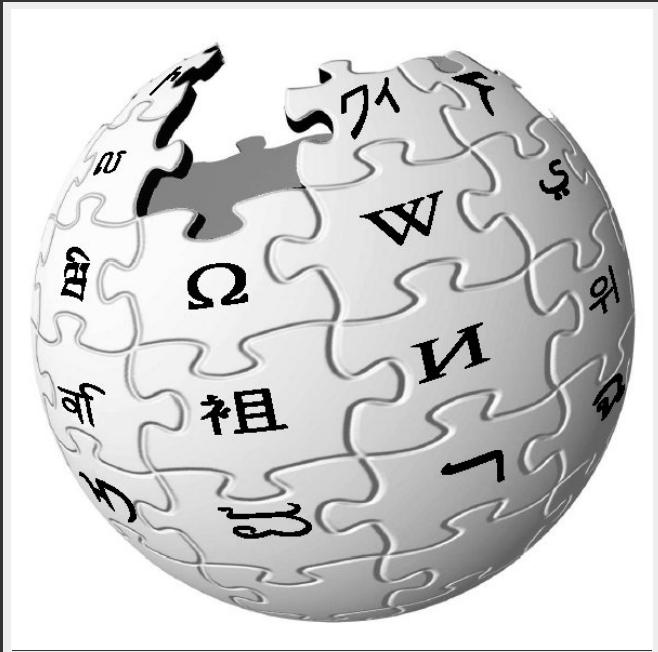


OTRAS FUNCIONALIDADES DISPONIBLES:

- Hypermedia
- Seguridad: OAuth, SSL, etc
- Logging
- Gestión de excepciones
- Soporte para Spring Framework
- API de acceso cliente
- Uploads: Jersey Multipart
- Testing: Jersey Test Framework
- Y mucho más ...

TESTING

DEFINICIÓN

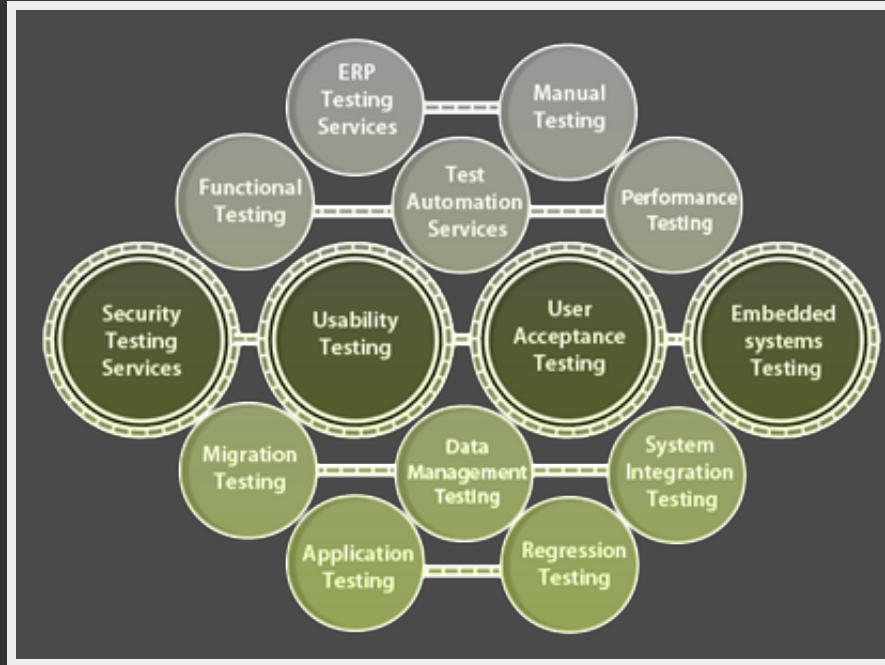


"Unit testing is a software verification and validation method in which a programmer tests if individual units of source code are fit for use. A unit is the smallest testable part of an application"

BENEFICIOS:

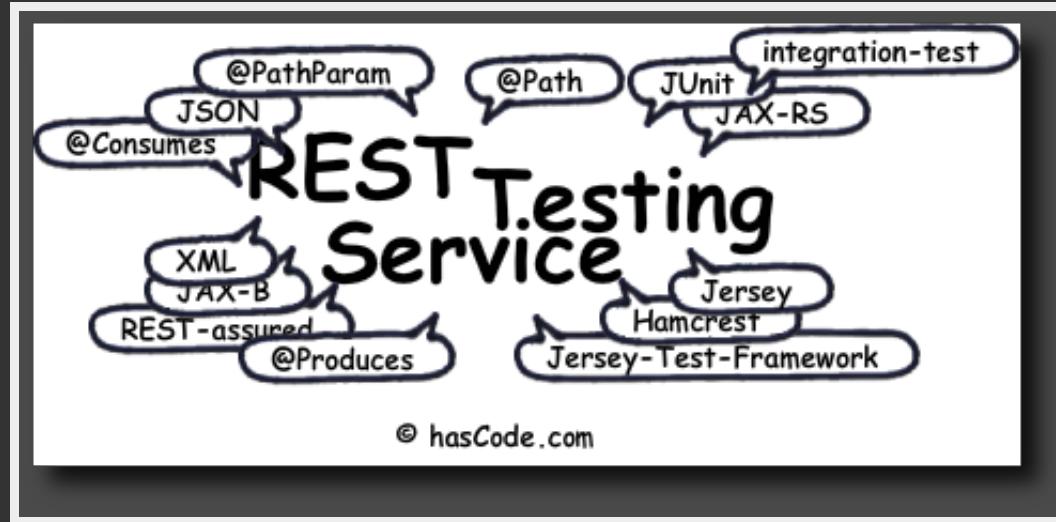
- Facilitar el cambio y el refactoring
- Diseño simple y modular del código
- Sirve como documentación
- Ahorra mucho tiempo y costes con futuros bugs





HERRAMIENTAS

- Pruebas de código: **jUnit**
- Pruebas de interfaz: **Selenium**
- Pruebas de integración: **SoapUI** o **Jersey Test Framework**
- Pruebas de estrés: **jMeter** con independencia del lenguaje en el que esté desarrollado el servicio



DEPENDENCIAS EXTRA NECESARIAS

Añadir al pom.xml las siguientes dependencias:

```
<dependency>
    <groupId>com.sun.jersey.jersey-test-framework</groupId>
    <artifactId>jersey-test-framework-core</artifactId>
    <version>1.14</version>
</dependency>

<dependency>
    <groupId>com.sun.jersey.jersey-test-framework</groupId>
    <artifactId>jersey-test-framework-grizzly</artifactId>
    <version>1.14</version>
</dependency>
```

DEFINICIÓN DE UN TEST

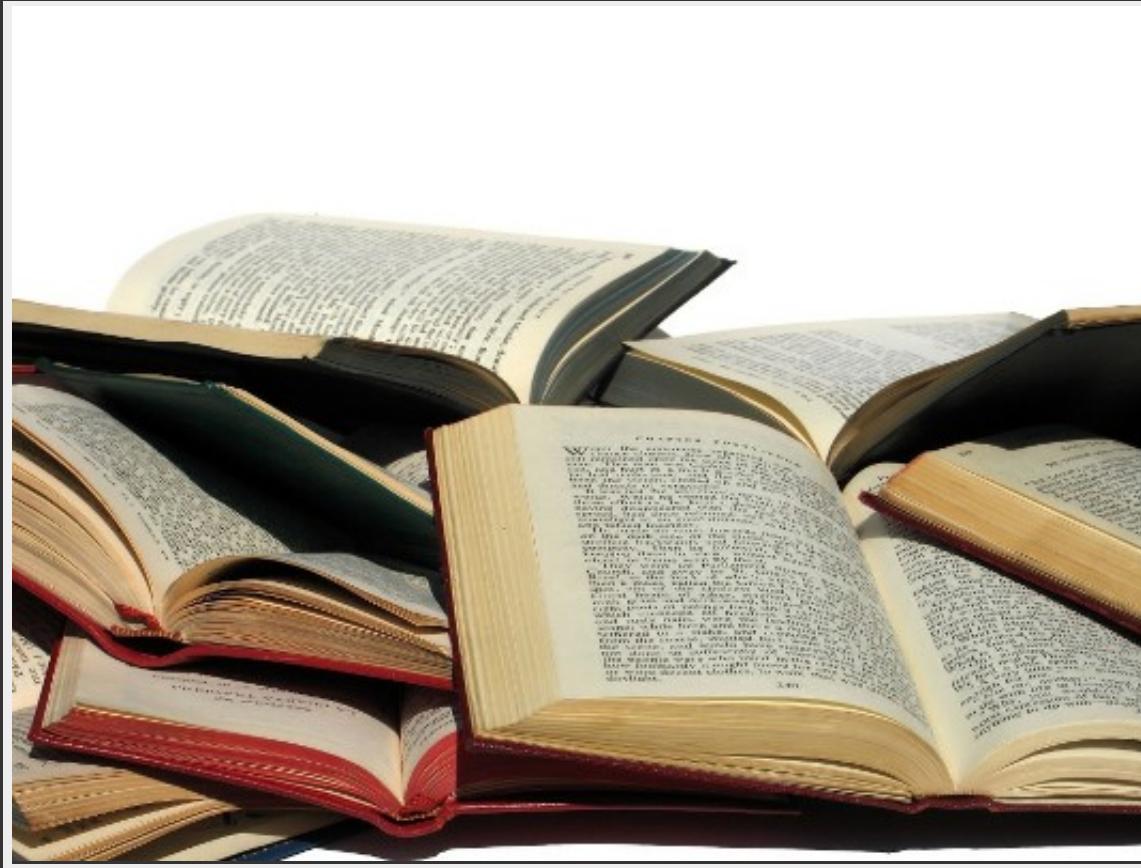
Código necesario para arrancar el contenedor Java:

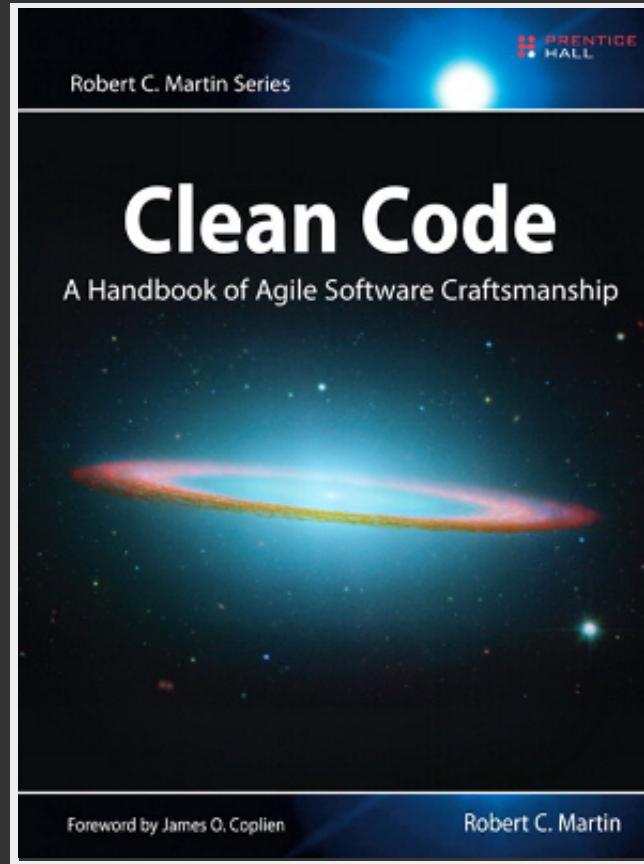
```
public class UsersResourceTest extends JerseyTest {  
    private WebResource resource;  
  
    public UsersResourceTest() {  
        super(new WebAppDescriptor.Builder("com.decharlas.services")  
            .contextParam("webAppRootKey", "jersey-maven.root")  
            .servletClass(ServletContainer.class).build());  
        this.resource = resource();  
    }  
  
    @Override  
    protected TestContainerFactory getTestContainerFactory() {  
        return new GrizzlyWebTestContainerFactory();  
    }  
}
```

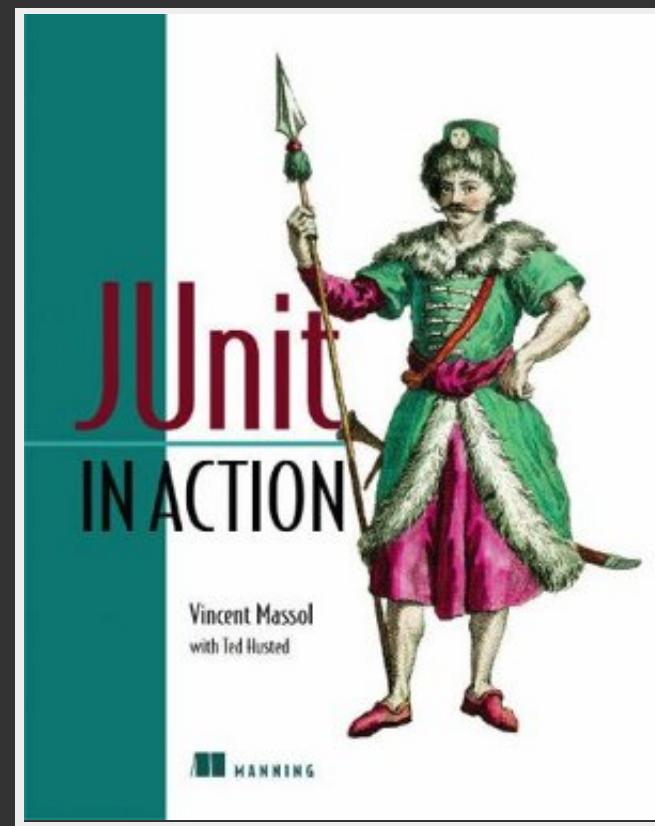
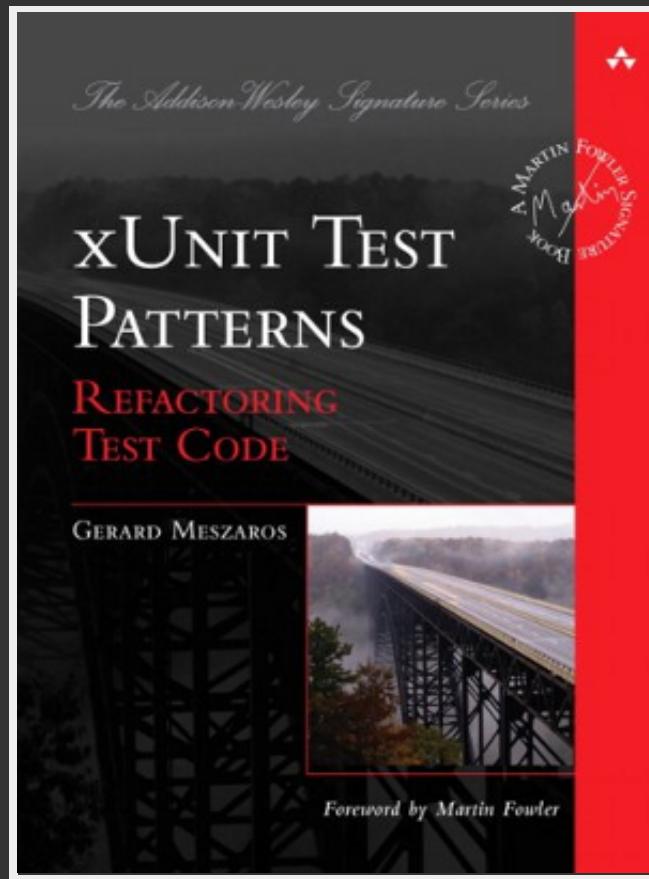
DEFINICIÓN DE UN TEST

Definición de los tests:









The Addison-Wesley Signature Series

A KENT BECK
Signature Book

GROWING OBJECT-ORIENTED SOFTWARE, GUIDED BY TESTS

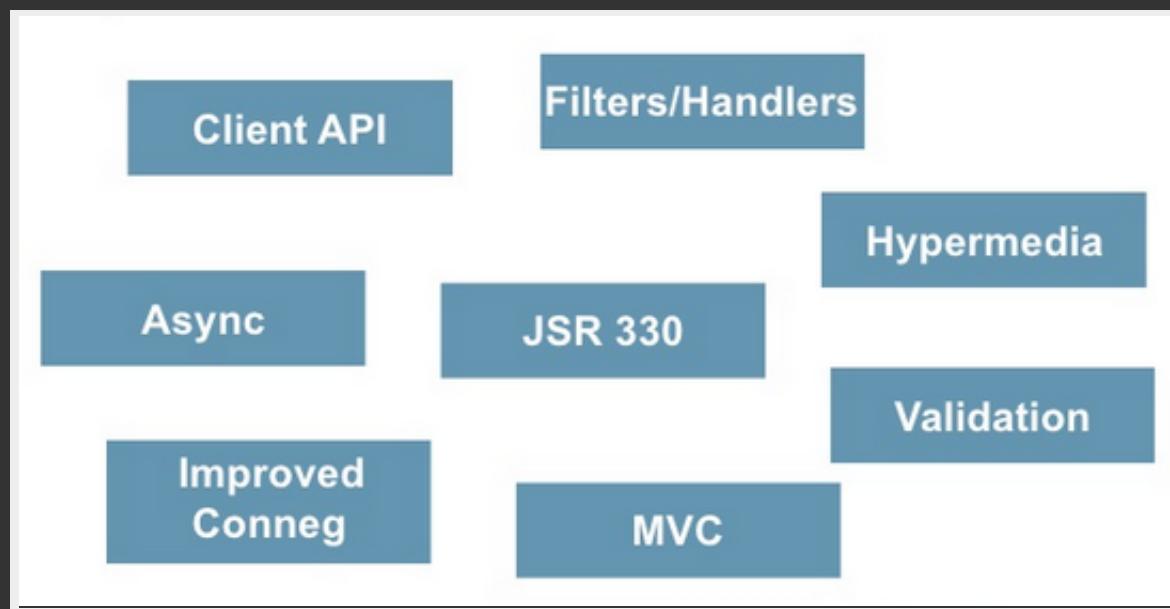
STEVE FREEMAN
NAT PRYCE



EL FUTURO DE JAX-RS

El futuro es ...

JAX-RS 2.0



¿PREGUNTAS?

