JAVA WEB SERVICES

WS REST

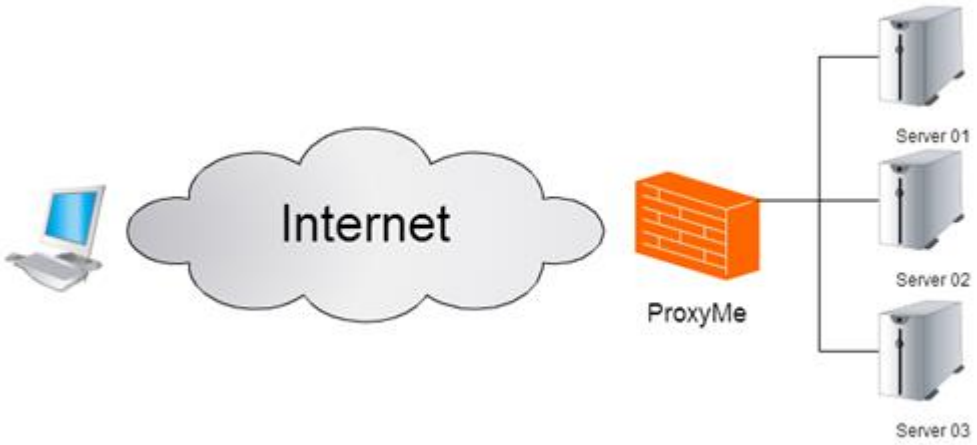
ERIC GUSTAVO CORONEL CASTILLO

www.desarrollasoftware.com

gcoronelc@gmail.com

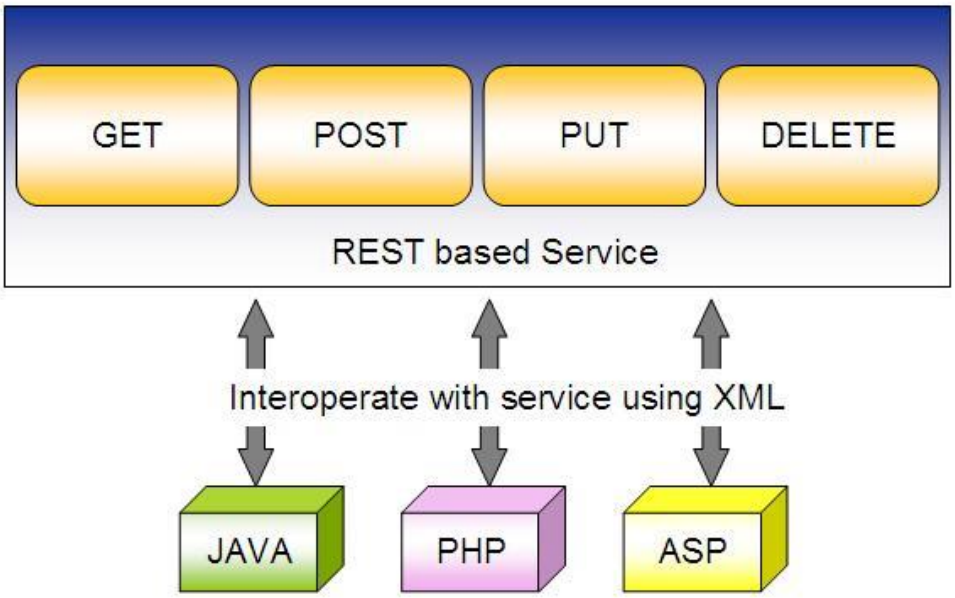
Contenido

1 INTRODUCCIÓN



Internet es una comunicación constante entre servidor y cliente, hacer que esta comunicación sea lo más eficaz posible es el principal objetivo de las tecnologías web.

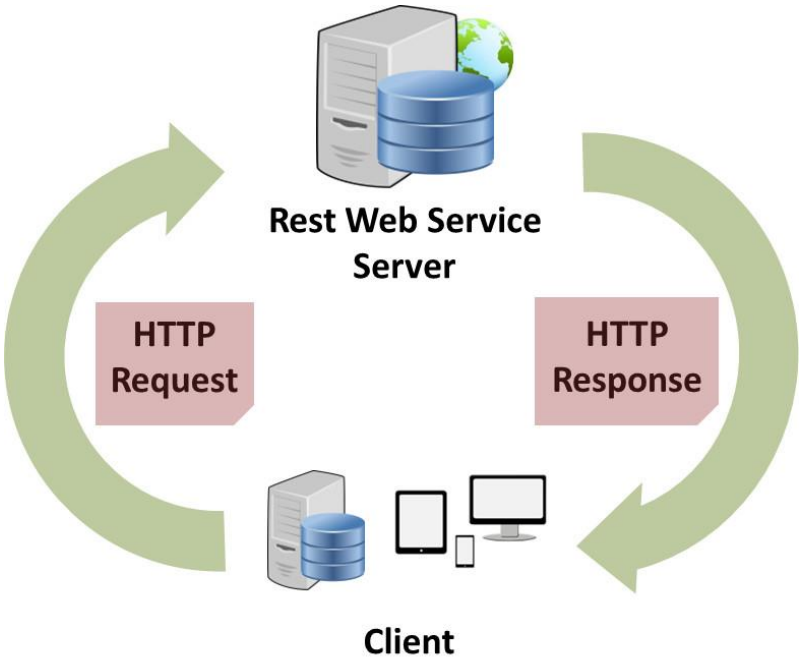
2 RESENTATIONAL STATE TRANSFER



Representational State Transfer (REST) es el estilo de arquitectura de software de la World Wide Web, esto es, bajo el protocolo HTTP.

Consiste en una serie de directrices para mejorar las comunicaciones cliente-servidor a través de la web.

3 FUNDAMENTOS



Todo lo que transportamos a través de las comunicaciones de internet es un recurso y está representado for su formato:

- image/jpeg
- video/mpeg
- text/html
- text/xml

Cada recurso debe estar representado por un identificador único y debe ser accesible, para ello debe existir una URI única para cada recurso.

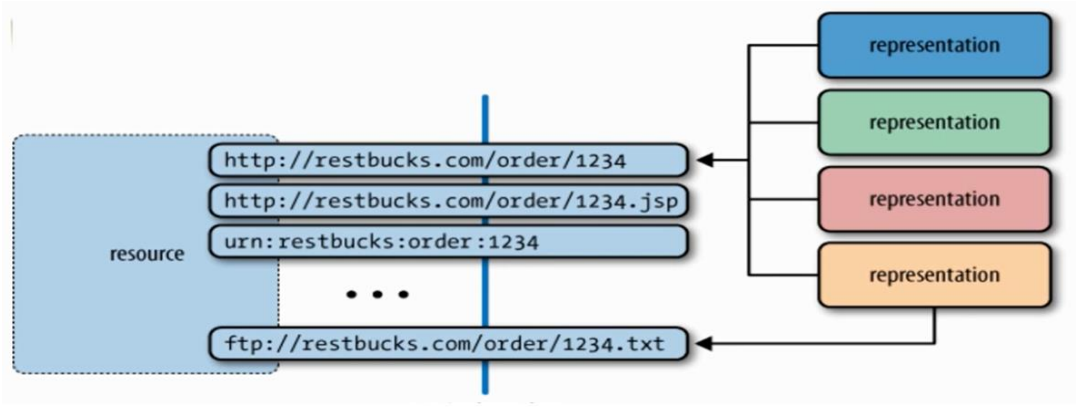
Utiliza los métodos estándares HTTP: GET, POST, DELETE, HEAD, OPTIONS, TRACE, CONNECT. Conocidos como verbos.

Los recursos pueden tener múltiples representaciones, por ejemplo, puedes solicitar los datos de un cliente en XML o JSON.

Se trata de una comunicación **SIN ESTADO**, quiere decir que cada petición es independiente, y retorna un estado final.

4 VENTAJAS

Separación del recurso y su representación. Un recurso puede ser representado en múltiples formatos.



Es la cabecera HTTP la que especifica la representación en que espera el recurso. En la siguiente imagen el recurso se solicita en formato XML.

```
GET /data/balance/22082014 HTTP/1.1
Host: nombrehost-mi-servidor
Accept: text/xml
```

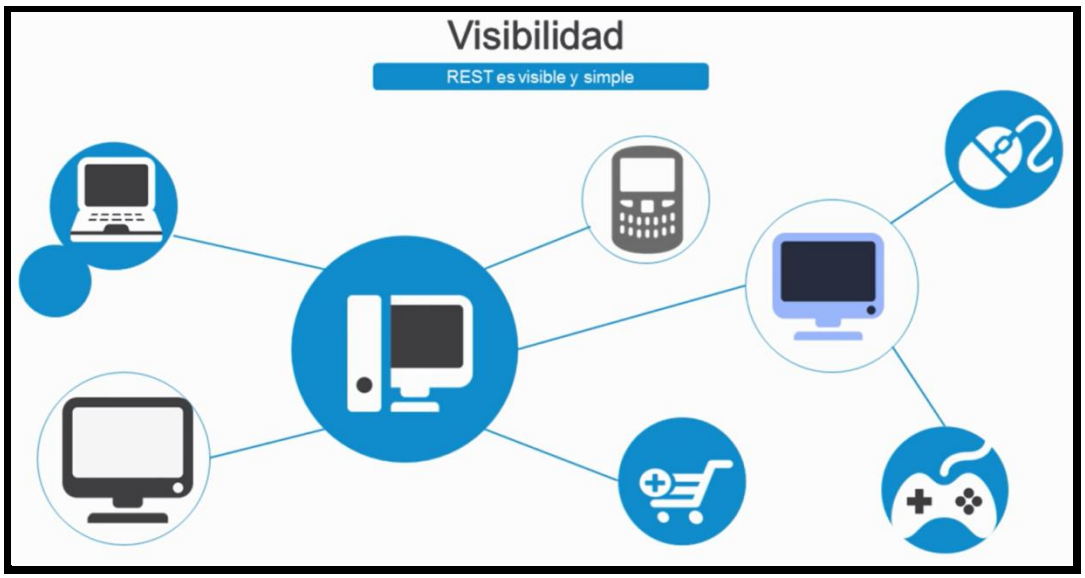
```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 140

<?xml version="1.0" encoding="utf-8"?>
<envio fecha="22082014">
<Item>Ejemplo Item</Item>
    <peso medida="Kilos">23</peso>
</envio>
```

En la siguiente imagen, el mismo recurso en formato JSON.

```
GET /data/balance/22082014 HTTP/1.1
Host: nombrehost-mi-servidor
Accept: application/json
```

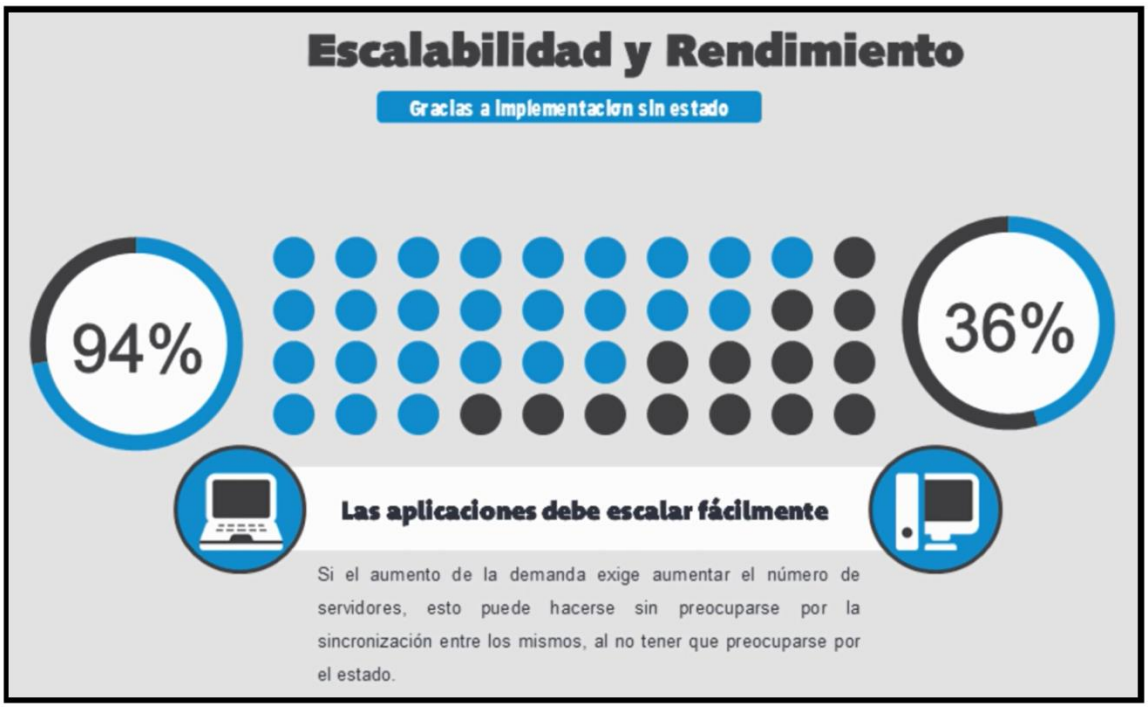
```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 120

{
 "envio": {
    "fecha": "22082014",
    "Item": "Ejemplo Item",
    "peso": {
      "-medida": "Kilos",
      "#text": "100"
    }
 }
}
```

REST, está diseñado para ser visible y simple, quiere decir que cada aspecto del servicio debe ser auto-descriptivo, siguiendo el lenguaje natural HTTP.



Los servicios REST son fiables.

Los servicios REST son fáciles de escalar y mantener un rendimiento aceptable.



5 EJEMPLO BASICO

5.1 Enunciado

Implementar un servicio REST que envíe un saludo genérico.

5.2 Configuración de Dependencias

Versión de Jersey

```
<properties>
  <jersey.version>1.14</jersey.version>
</properties>
```

Configuración de Jersey

```
  <!-- Jersey -->

  <dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-server</artifactId>
    <version>${jersey.version}</version>
  </dependency>

  <dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-servlet</artifactId>
    <version>${jersey.version}</version>
  </dependency>

  <!-- Jersey Multipart -->

  <dependency>
    <groupId>com.sun.jersey.contribs</groupId>
    <artifactId>jersey-multipart</artifactId>
    <version>${jersey.version}</version>
  </dependency>

  <dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-json</artifactId>
    <version>${jersey.version}</version>
  </dependency>
```

5.3 Configuración de Servlet

```
<servlet>
  <description>JAX-RS Tools Generated - Do not modify</description>
  <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>pe.egcc.rest.server</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>
```

5.4 Clase DTO

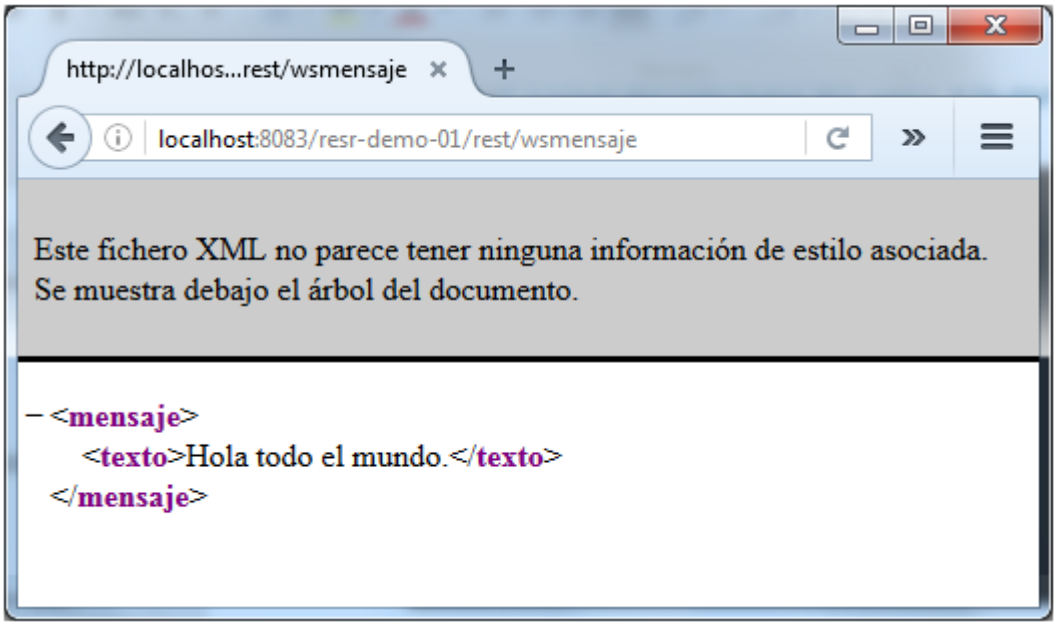
```
package pe.egcc.rest.dto;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="mensaje")
public class Mensaje {

   private String texto;

   public Mensaje() {
   }

   public Mensaje(String texto) {
      super();
      this.texto = texto;
   }

   public String getTexto() {
      return texto;
   }

   public void setTexto(String texto) {
      this.texto = texto;
   }

}
```

5.5 Programando el Servicio REST

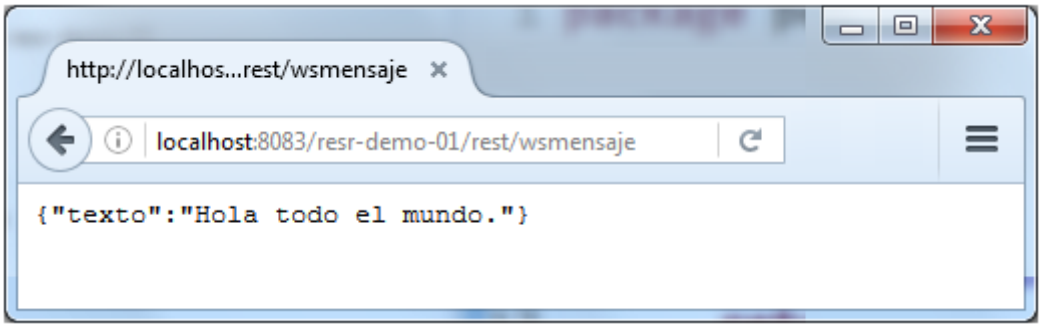
```
package pe.egcc.rest.server;

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

import pe.egcc.rest.dto.Mensaje;

@Path(value="wsmensaje")
public class WSMensaje {

   @GET
   @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
   public Mensaje getMensaje1(){
      Mensaje mensaje = new Mensaje("Hola todo el mundo.");
      return mensaje;
   }

}
```

5.6 Ejecutando el Servicio REST en el Browser

Respuesta XML:



Respuesta JSON



5.7 Realizar la Prueba con SOAPUI

Desarrolle la prueba con la aplicación SOAPUI, tanto para XML y JSON.

5.8 Crear un Cliente para Java

```
package pe.egcc.prueba;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

/**
 *
 * @author Gustavo Coronel
 * @blog gcoronelc.blogspot.com
 * @email gcoronelc@gmail.com
 */
public class Prueba01 {

  public static void main(String[] args) {
    try {

      Client client = Client.create();
      WebResource webResource =
client.resource("http://localhost:8084/DemoRESTO01/rest/wsmensaje");
      ClientResponse response =
webResource.accept("application/json").get(ClientResponse.class);

      if (response.getStatus() != 200) {
        throw new RuntimeException("Failed : HTTP error code : " +
response.getStatus());
      }

      System.out.println("Output from Server .... \n");
      String respuesta = response.getEntity(String.class);
```

```
      System.out.println(respuesta);

    } catch (Exception e) {
      System.out.println("Error");
      e.printStackTrace();
    }
  }

}
```

5.9 Crear un cliente con JQuery

```
<!DOCTYPE html>
<html>
  <head>
    <title>TODO supply a title</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="jquery/jquery.js"></script>
  </head>
  <body>
    <input type="button" id="btnBoton" value="Mostrar" />
    <script>

      $("#btnBoton").click(function () {
        $.ajax({
          url: 'http://localhost:8084/DemoRESTOO1/rest/wsmensaje', // url del
recurso
          type: "GET", // podría ser get, post, put o delete.
          headers: {"Accept":"application/json"},
          success: function (respuesta) {
            console.log( respuesta );
            alert("Hola");
            alert(respuesta.texto);
          }
        });
      });

    </script>
  </body>
</html>
```

6 INTEGRACIÓN CON BASE DE DATOS

Desarrolle un Servicio REST para realizar el CRUD de la tabla CLIENTE de la base de datos EUREKABANK.

7 EJERCICIO PROPUESTO

Desarrollar un servicio REST que permita convertir:

- De Soles a Dólares y viceversa
- De Soles a Euros y viceversa
- De Dólares a Euros y viceversa

Luego programar los clientes en:

- Java
- JQuery
- PHP
- .NET
- Android