

Servicios Web

Sesión 4: Servicios Web RESTful



Puntos a tratar

- Fundamentos del estilo REST
- Crear un servicio RESTful
- Integrar con otros servicios
- Crear un cliente para servicios RESTful



Estilo REST

- Forma alternativa de crear servicios
 - Es un estilo arquitectónico
 - No un estándar
- Elemento básico: URL
 - El formato para intercambiar información es decisión del desarrollador
 - Más ligeros que SOAP
- Especialmente adecuados para AJAX
 - Acerca los servicios web a la “web”



URLs

- Cada recurso se representa por una URL
 - `http://jtech.ua.es/resources/cursos`
 - `http://jtech.ua.es/resources/cursos/1`
 - `http://jtech.ua.es/resources/cursos/2`
 - etc ...
- Operaciones sobre los recursos

GET	SELECT
POST	INSERT
PUT	UPDATE
DELETE	DELETE



Formato de los datos

- Podemos utilizar diferentes formatos

Texto plano	text/plain
HTML	text/html
XML	application/xml
JSON	application/json

- JSON
 - Lenguaje ligero de intercambio de información
 - Directamente importable en Javascript: `eval()`



SOAP vs RESTful

<i>SOAP</i>	<i>RESTful</i>
<ul style="list-style-type: none">• Contrato formal (WSDL)• Interconexión de sistemas• Información de estado• Comunicación asíncrona• Envío fiable• Seguridad• Transacciones	<ul style="list-style-type: none">• Servicios sin estado• Datos estáticos• Ancho de banda limitado• Aplicaciones AJAX• <i>Toolkit</i> de desarrollo



RESTful en Netbeans

- Necesario instalar *plug-in*
- Formas de crearlos
 - *RESTful Web Services from Patterns..*
Se crean desde cero
 - *RESTful Web Services from Entity Classes...*
Se crean de forma automática a partir de clases JPA
Cada entidad estará vinculada a una URL
Se ofrecerán operaciones para:

Obtener un objeto	GET
Publicar un nuevo objeto	POST
Modificar un objeto existente	PUT
Borrar un objeto	DELETE



Patrones

- *Singleton*
 - Se utiliza una URL única
 - Servicios sencillos (tipo *Hola Mundo*)
 - Interfaz REST para servicios SOAP existentes
- *Container-Item*
 - Colecciones de datos
 - URL contenedora y URLs para *items*
- *Client-Controlled Container-Item*
 - Variación de la anterior
 - Se añaden nuevos *items* con PUT sobre URL *item*



Ejemplo sencillo

```
static String texto="";

@GET
@ProducesMime("text/html")
public String getHtml(@QueryParam("nombre")
                     @DefaultValue("John Doe")
                     String nombre) {
    return "<p>Hola " + nombre + "</p>" + texto;
}

@PUT
@ConsumesMime("text/html")
public void putHtml(String content) {
    texto+=content;
}
```



Cliente básico

```
// Definimos la URL en la que se encuentra el
servicio URL url = new URL(
    "http://localhost:8080/HolaMundo/resources/holaMundo" );
URLConnection httpConn =
    (URLConnection)url.openConnection();

// Establecemos el método PUT
httpConn.setRequestMethod("PUT");
httpConn.setDoOutput(true);

// Enviamos texto al servicio
OutputStream os = httpConn.getOutputStream();
PrintStream ps = new PrintStream(os);
ps.println("<p>Contenido a enviado</p>");
httpConn.getInputStream();
```




Ejemplo con contenedor-item (I)

- GET sobre el contenedor

Status: 200 (OK)

Response:

Tabular View	Raw View 	Sub-Resource	Headers	Http Monitor
--------------	--	--------------	---------	--------------

```
<?xml version="1.0" encoding="UTF-8"?>
  <establecimientoes uri="http://localhost:8080/GuiaLocal/resources/establecimientoes/">
    <establecimientoRef uri="http://localhost:8080/GuiaLocal/resources/establecimientoes/1/">
      <id>1 </id>
    </establecimientoRef>
    <establecimientoRef uri="http://localhost:8080/GuiaLocal/resources/establecimientoes/2/">
      <id>2 </id>
    </establecimientoRef>
  </establecimientoes>
```



Ejemplo con contenedor-item (II)

- GET sobre el *item*

```
<establecimiento uri="http://localhost:8080/GuiaLocal/resources/establecimientos/1/">
  <actividad>Restaurante</actividad>
  <direccion>Avenida de la estacion, 5, Alicante, España</direccion>
  <id>1</id>
  <nombre>Telepizza</nombre>
</establecimiento>
```

- *POST* sobre el item

Choose method to test:

Click 'Test' to continue:

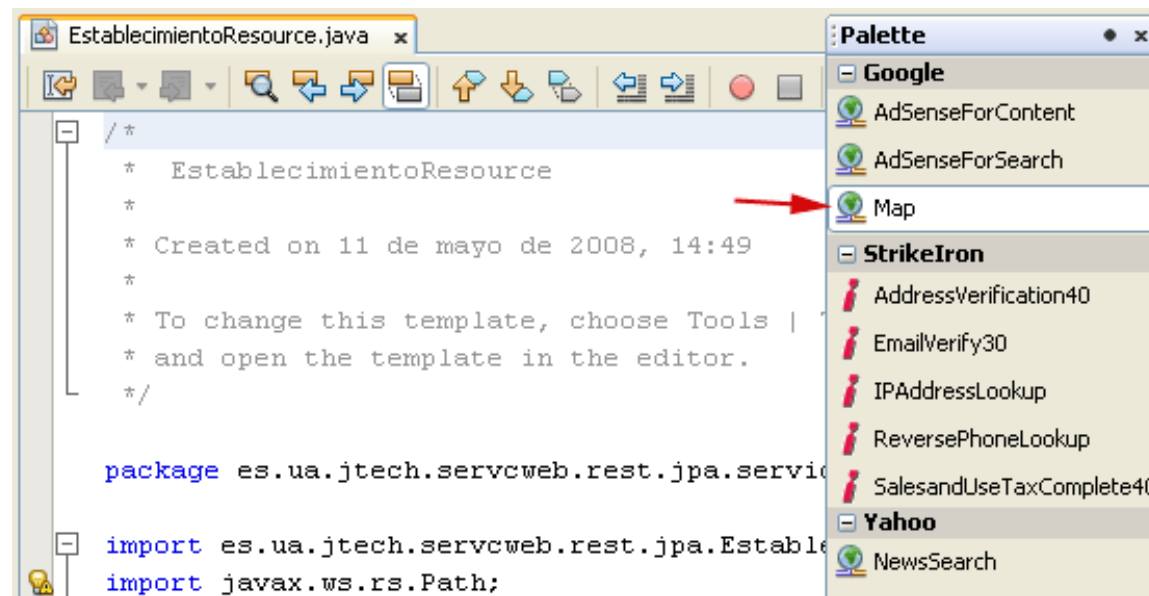
Content:

```
<establecimiento>
<actividad>Cine</actividad>
<direccion>Plaza del Carmen, 16, Alicante, España</direccion>
<id>3</id>
<nombre>Cines Astoria</nombre>
</establecimiento>
```



Integrar con otros servicios

- Paleta de servicios en Netbeans
 - Se muestra al editar nuestra clase Resource
 - Podemos arrastrar de la paleta a nuestro servicio





Ejemplo: Google Maps

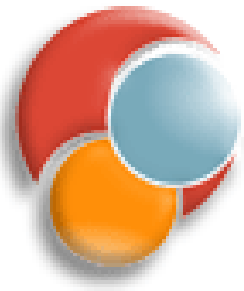
- Proporcionar posición en el mapa de cada uno de los *items*
- Relacionar propiedades del *item* con parámetros para Google Maps

```
@Path("googleMap/")
public GoogleMapResource getGoogleMap() {
    try {
        String apiKey = null;
        String address = getEntity().getDireccion(); ;
        Integer zoom = null;
        return new GoogleMapResource(apiKey, address, zoom);
    } finally {
        PersistenceService.getInstance().close();
    }
}
```



Generar *stub* para clientes

- Podemos generar una librería Javascript
 - Puede ser importada desde cualquier cliente web
 - Dará acceso a nuestro servicio
- Alternativa: *jMaki*
 - *Framework AJAX*
 - Podemos generar *widgets* jMaki para dar acceso a nuestro servicio (*tags* JSP)
 - Estos *widgets* pueden ser añadidos a la paleta de componentes *jMaki* en Netbeans
 - Será accesible mientras editemos un JSP
 - Requiere la instalación del *plugin jMaki*.



¿Preguntas...?