Aplicacions Distribuïdes (AD)

Práctica 4: Desarrollo de servicios web con REST

Creación de servicios web basados en REST

En esta práctica vamos a crear servicios Web RESTful con *Netbeans*.

Lo primero es crear una aplicación web que contenga el servicio REST. Tiene que ser una nueva aplicación web, no se pueden reutilizar las aplicaciones web de las prácticas 2 y 3 porque nos darían problemas a la hora de hacer la integración. Podéis llamar a dicha aplicación `RestAD`.

Una vez creada, hay que crear el servicio REST. Utilizaremos la opción `RESTful Web Services from Patterns`. Crearemos un servicio de tipo recurso genérico, que responderá a métodos HTTP siguiendo las directrices de REST. El tipo MIME soportado será `text/html`. Otros tipos MIME posibles son `application/html`, `application/json` y `text/plain`.

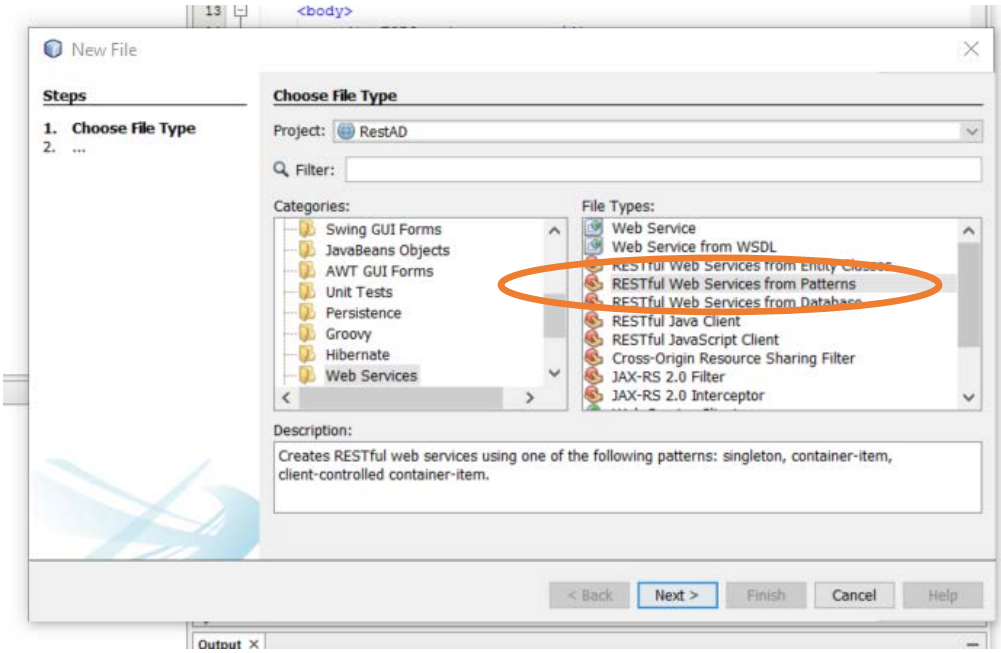


Figura 1. Creación servicio RESTful.

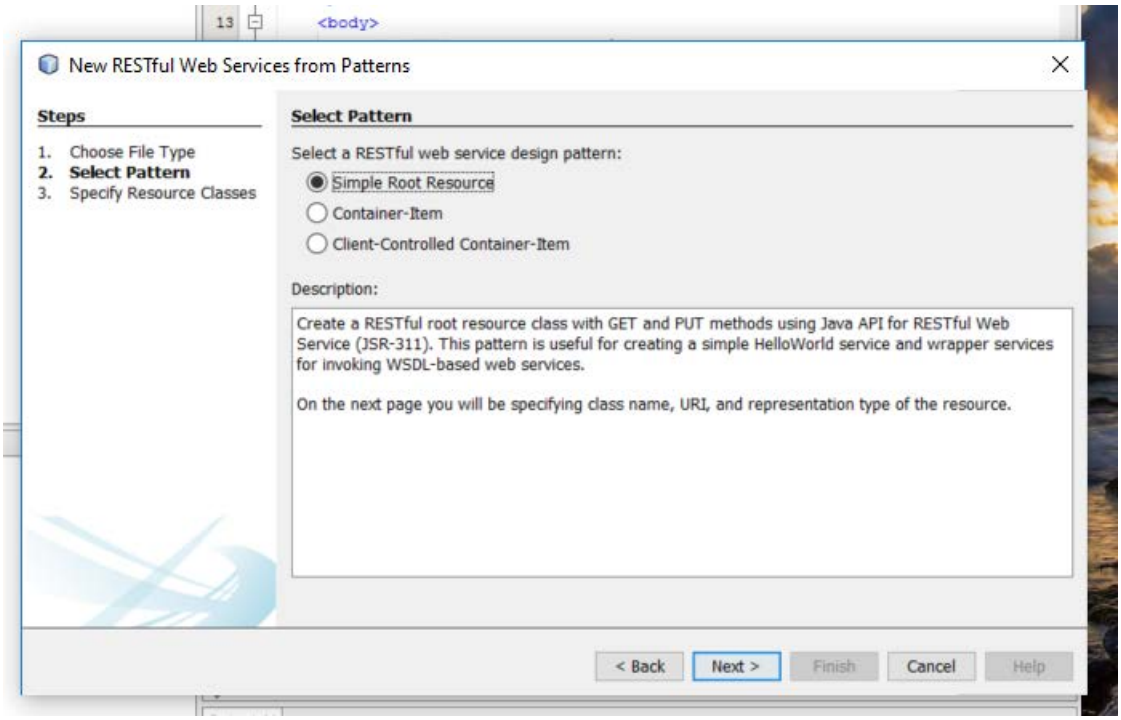


Figura 2. Selección de opción Simple Root Resource.

Definid Resource Package como `restad` y seleccionad `text/html` como tipo MIME en la pantalla que se muestra en la Figura 3.

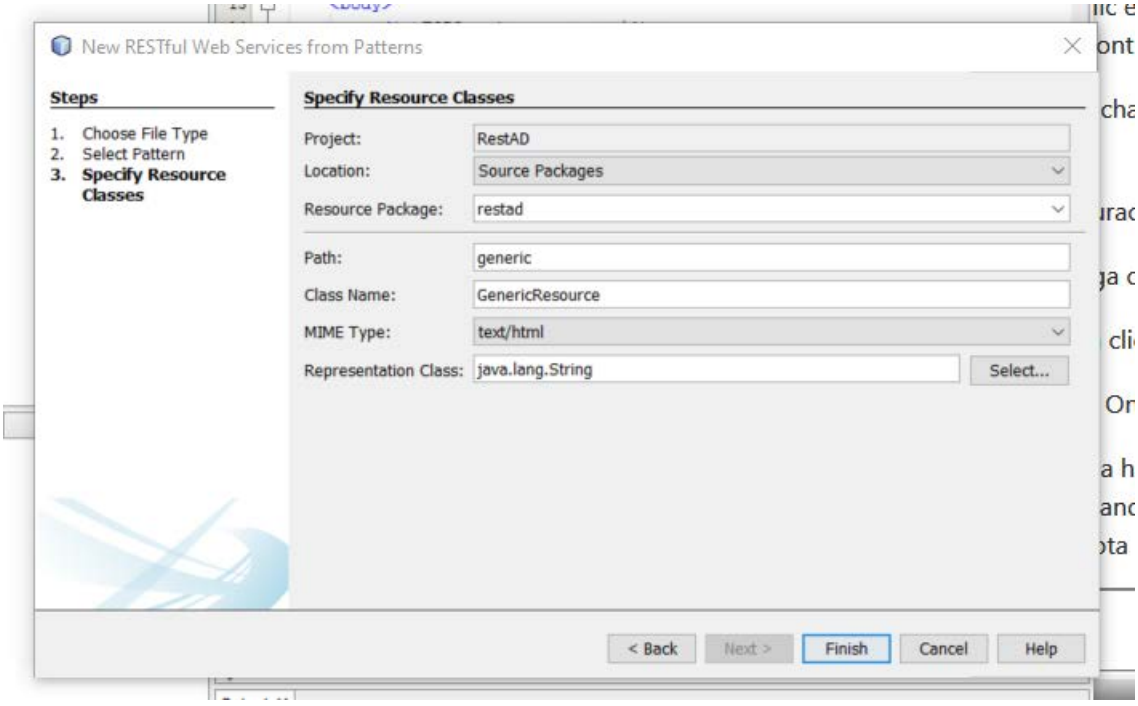


Figure 3. Creación del servicio como GenericResource.

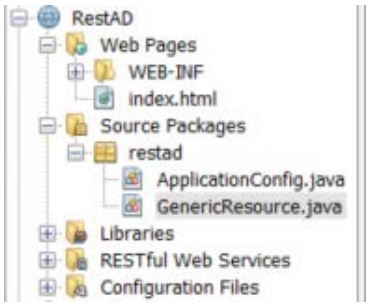


Figura 4. Ficheros generados.

Revisa el código Java de los ficheros que se han generado. Busca los métodos HTTP implementados en el fichero `GenericResource.java`.

Modifica el método `getHtml()`. Añade el código `return "<html><head/><body><h1>Hello World!</h1></body></html>";` y ejecuta la aplicación web para que se ejecute el servicio. Para acceder al método, utiliza la siguiente URL en un navegador:

`http://localhost:8080/RestAD/webresources/generic/`

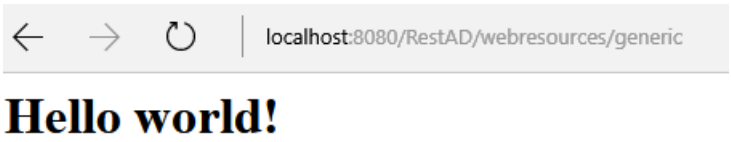


Figure 5. Ejecutando el servicio REST.

Responde a las siguientes cuestiones:

a) ¿Por qué podemos acceder la servicio desde el navegador?
b) ¿Qué método HTTP se está utilizando?
c) ¿Con qué tipo MIME?
d) ¿Dónde y cómo se indica la URL, el método HTTP y el tipo MIME del servicio REST?

Implementando el servicio

Añade el método `libres` (copia el código que hay a continuación) al fichero `GenericResource.java` para que recoja dos parámetros llamados `id` y `fecha` enviados desde un formulario html y responda con una página html.

```
/**
  * Método POST para comprobar los asientos libres dados un id y
  * una fecha
  * @param id
  * @param fecha
  * @return
  */
```

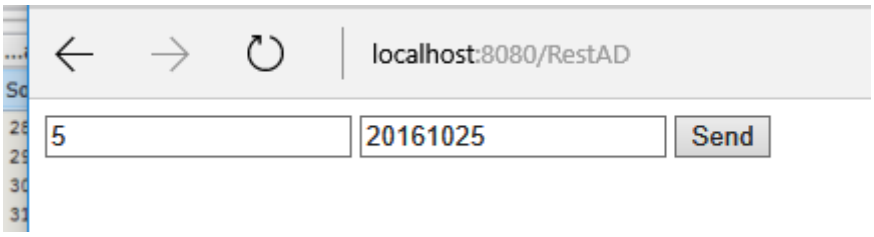
```
@Path("libres")
@POST
@Consumes("application/x-www-form-urlencoded")
@Produces("text/html")
public String libres (@FormParam("id") String id,
                      @FormParam("fecha") String fecha) {

    // Completar codigo

    return "<html><head></head> <body> Libres </body></html>";
}
```

Para solucionar los errores que os aparecerán, incluid los siguientes imports en Java:

```
import javax.ws.rs.FormParam;
import javax.ws.rs.Produces;
import javax.ws.rs.POST;
```

Añade el siguiente código al fichero index.html para que envíe los dos parámetros anteriores al servicio web REST:

```
<body>
    <form action="webresources/generic/free" method="post">
        <input type="text" name="id"/>
        <input type="text" name="fecha"/>
        <input type="submit" name="submit" value="Send"/>
    </form>
</body>
```



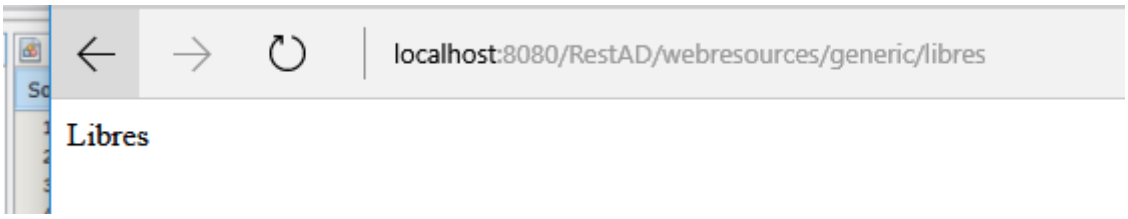


Figure 6. Ejecutando de nuevo la aplicación. Petición y respuesta del servicio.

Responde a las siguientes cuestiones:

a) ¿Cuántos elementos (packages, ficheros, carpetas, servicios, etc.) se han creado automáticamente en el servicio web?

b) ¿Para qué sirve FormParam? Busca información de cómo se recogen los parámetros en los métodos REST implementados en Java.

c) Prueba de llamar desde el navegador a `http://localhost:8080/RestAD/webresources/generic/libres`. ¿Por qué no funciona? ¿Qué error HTTP devuelve el servidor? ¿Por qué?

d) Utiliza la opción Test RESTful Web Services para monitorizar el servicio REST que has implementado. Puedes comprobar cómo es la petición, la respuesta, etc.

Desarrollo de la práctica

Servicios web con REST

Para continuar con el desarrollo de nuestra agencia de viajes, en esta práctica se van a implementar una serie de servicios web con REST que van a realizar las siguientes operaciones:

- Vuelos: Consulta de plazas libres y reserva de plazas para un vuelo.

Para ello, es necesario utilizar la tabla de vuelos de la base de datos que ya tenemos de la práctica anterior. Dicha información se detalla a continuación:

- Tabla vuelo_fecha (id_vuelo, fecha, num_plazas_ocupadas, num_plazas_max)

NOTA: La fecha se puede almacenar como un único entero en formato aaaammdd. El resto de campos también serán de tipo entero.

Operaciones a implementar en los servicios web

El servicio web que se debe implementar es VueloREST, cuyas operaciones se describen a continuación.

VueloREST tiene dos operaciones, una para consultar las plazas libres en una determinada fecha y otra para reservar una plaza en un vuelo en una fecha. A continuación se describen sus cabeceras.

Descripción de las operaciones de VueloREST:

```
/* Dados un identificador de vuelo y una fecha, retorna el número de plazas que están libres */
String consulta_libres (int id_vuelo, int fecha);

/* Dados un identificador de vuelo y una fecha, reserva una plaza si quedan plazas libres (incrementa el número de plazas ocupadas en un vuelo en una fecha.
Si es posible realizar la reserva, esta operación retorna el número de plazas ocupadas que hay en el vuelo.
Si no es posible realizar la reserva, esta operación retorna -1. */
String reserva_plaza (int id_vuelo, int fecha);
```

Para desarrollar estos servicios web, tened en cuenta el método HTML que queréis utilizar para pasarle parámetros al servicio.

Para comprobar que los servicios web funcionan correctamente, desarrollad los correspondientes clientes, que serán aplicaciones en lenguaje java como las descritas en los tutoriales realizados al principio de la práctica.

Ejercicio adicional

Implementar con REST las operaciones de Hoteles de los servicios web basados en SOAP de la práctica anterior.