Ejercicios de AJAX y REST

Índice

1. AJAX (1 punto)

Vamos a cambiar la búsqueda de usuarios para que funcione mediante AJAX. Así los resultados se mostrarán directamente en la página de búsqueda sin necesidad de cambiar de página. Seguid estos pasos:

1. Incluir la librería Jackson en el pom.xml, puedes copiar la dependencia que aparece de los apuntes
2. **Parte del servidor**: añádele un método `buscarAJAX` al BuscarAmigosController, que se ocupe de esta funcionalidad.
   - El @RequestMapping puede ser a la misma URL que el buscarAmigos "convencional", con el mismo método POST, pero debes indicar que genera JSON usando el atributo `produces`
   - El método devolverá una List<Usuario>. Debes anotar el valor de retorno con @ResponseBody para que se devuelva en la respuesta HTTP.
3. **Parte del cliente**: copia el siguiente código dentro de la página busqueda.jsp, después del formulario

```
<div id="resultados"></div>
<script type="text/javascript"
src="http://code.jquery.com/jquery.min.js"></script>
<script>
function crearTabla(obj) {
        var tabla = "<table><tr> <th>Login</th> <th>Edad</th>
<th>Localidad</th> </tr>"
        for(i=0; i<obj.length; i++) {
          tabla += "<tr> <td>" + obj[i].login + "</td> <td>" + obj[i].edad
+
             "</td> <td>" + obj[i].localidad + "</td> </tr>"
        }
        tabla += "</table>"
        alert(JSON.stringify(obj, undefined, 2))
        //hide y show solo se usan aquí para molar
        $('#resultados').hide().html(tabla).show('slow')
}

function buscarAJAX() {
        $.ajax({
                type: 'POST',
                url: 'busqueda.do',
                dataType: 'json',
                data: $('#formulario').serialize(),
                success: crearTabla
        })
}
</script>
```

4. Para que el código anterior funcione, debes ponerle al formulario un atributo id="formulario". Además, debes añadir dentro del formulario un nuevo botón para la búsqueda con AJAX. No es necesario que borres el anterior, así puedes tener los dos tipos de búsqueda:

```
<input type="button" value="buscar con AJAX" onclick="buscarAJAX()"/>
```

5. Cuando lo pruebes, primero verás que aparece en un cuadro de diálogo el JSON que nos envía el servidor (línea y luego se muestra la tabla formateada en HTML. Fijate en que aunque en el HTML no se ve el password del usuario, desde el servidor sí nos lo está enviando, lo que es un problema de seguridad. Puedes anotar la clase Usuario con la anotación Jackson `@JsonIgnoreProperties({"password", "credito"})`, para que la librería no serialice ninguno de los dos campos.
6. Una vez hecho lo anterior, puedes quitar la línea 11 (el 'alert'), solamente estaba puesta para que vieras "en crudo" los datos que envía el servidor

2. Servicios REST (1.5 puntos)

Queremos hacer una implementación REST de algunas funcionalidades de la aplicación. En concreto, queremos implementar:

- La búsqueda de usuarios
- Obtener los datos de un usuario a partir de su login ("ver amigo")
- Dar de alta un usuario

Debéis seguir estos pasos:

1. **Configuración**: usaremos otro dispatcher servlet adicional al que ya tenemos. Así podemos configurarlo por separado. En el web.xml introduce el siguiente código:

```
<servlet>
   <servlet-name>rest</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
   <servlet-name>rest</servlet-name>
   <url-pattern>/REST/*</url-pattern>
</servlet-mapping>
```

como ves, le asociamos las URL que comienzan por /REST. Ahora crearemos el fichero de configuración .xml para dicho servlet, que se llamará rest-servlet.xml y estará en WEB-INF, siguiendo la convención por defecto de Spring MVC:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.2.xsd
                http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

        <context:component-scan
base-package="es.ua.jtech.amigosspring.rest" />
        <mvc:annotation-driven />
</beans>
```

2. **Crear el controller**: Crea el paquete `es.ua.jtech.amigosspring.rest`. Crea dentro de él la clase `AmigosRESTController`. Anótala con @Controller y mapéala con @RequestMapping a la URL "amigos"
3. Ahora "solo" te queda **implementar las tres funcionalidades**. Recuerda que:
   - Debes respetar la "filosofía" REST en cuanto al uso de los métodos HTTP. Buscar usuarios y ver un usuario usarán GET, mientras que crear usuario debería usar POST
   - Asímismo debes mapear adecuadamente las URLs. Buscar y crear usarán la misma URL que el controller "entero" (amigos), pero la URL de "ver amigo" debe acabar por el login del usuario que quieres ver (por ejemplo "amigos/jsf")
   - Buscar y ver usuario producen JSON, mientras que crear usuario consume JSON

Para probar los servicios REST puedes usar el cliente REST-shell, desarrollado por SpringSource. Si quieres más información puedes ver su repositorio en github. Ejecuta el programa "rest-shell" que está dentro del subdirectorio "bin" de "rest-shell-1.2.1.RELEASE". Es un prompt que acepta comandos que permiten hacer peticiones REST de modo relativamente sencillo. En todo momento podemos ver la ayuda general con 'help' o solo sobre un comando determinado, por ejemplo 'help get'.

Por ejemplo, para probar el "ver usuario" haríamos

```
#Establece la URL "base" por defecto para todas las peticiones a partir de
aquí
base http://localhost:8080/AmigosSpring/REST/amigos
#Para decirle al servidor que nos puede mandar JSON. Cabecera HTTP
"Accept:"
headers set --name Accept --value application/json
#hacer una petición GET a una URL
get jsf
```

Para buscar usuarios, suponiendo que ya hemos establecido la URL base y hemos fijado la cabecera Accept para aceptar JSON:

```
#podemos pasar parámetros HTTP escribiéndolos en formato JSON
#(pero no se envía JSON, se envían en la forma
param1=valor1&param2=valor&...)
get --params "{'edadMin':9, 'edadMax':90, 'sexo':'indiferente'}"
```

Para crear un nuevo usuario tendremos que hacer una petición POST y subir los datos JSON en el cuerpo de la petición.

```
#decimos que vamos a enviar JSON
headers set --name Content-Type --value application/json
base http://localhost:8080/AmigosSpring/REST/amigos
#enviamos los datos JSON (cuidado, meter todo lo siguiente en una sola
línea)
post --data "{'login':'test','password':'test','fechaNac':'1990-01-01',
    'localidad':'javaland', 'varon':'true', 'descripcion':'holaaaaaa'}"
```

Every little thing gonna be alright...

Por el momento vamos a suponer que no se produce ningún error al crear el nuevo usuario. En el último ejercicio gestionaremos alguna de las posibles excepciones y en la siguiente sesión veremos cómo validar los datos usando JSR303 (por ejemplo, que la fecha de nacimiento está en el pasado). Así que no probéis con datos erróneos ni intentéis crear un usuario con un login ya existente.

3. Gestión de errores en servicios REST (0.5 puntos)

Lo habitual en REST es devolver un código de estado HTTP si se ha producido un error y algún mensaje en el cuerpo de la respuesta con más información. Vamos a hacer esto en nuestro servicio:

- En primer lugar, cuando se quiere ver un usuario por login, si no existe se debe devolver un código de estado 404. Hazlo si no lo has hecho ya.
- Por otro lado, cuando se genera una excepción (por ejemplo al intentar crear dos usuarios con el mismo login) se ve en el cliente, lo que no es apropiado. Usa la gestión de excepciones que hemos visto en la sesión para transformar las posibles RuntimeException que devuelve el DAO en códigos de estado 400 (sí, le vamos a echar la culpa de todo al cliente, para simplificar el ejercicio). En el cuerpo de la respuesta envía el mensaje de la excepción, obtenido con getMessage().