Ejercicios de MVC en Spring

Índice

La aplicación "AmigosSpring" es una especie de red social muy simple en la que los usuarios pueden hacer login en el sistema y ver el perfil propio y el de otros. El núcleo de la aplicación ya está desarrollado, y el objetivo de la sesión es ponerle un interfaz web usando Spring MVC.

1. Configurar el proyecto para Spring MVC (0.5 puntos)

El proyecto está en las plantillas de la sesión. Es un proyecto Maven, así que lo puedes importar como "Existing Maven Projects".

Antes de probar la aplicación, **crea la base de datos del script "amigosSpring.sql"** incluido en la carpeta "database" del proyecto.

Tras desplegar el proyecto, comprueba que todo funciona con la página "testDAO.jsp". Tendrás que pasarle como parámetro "login" el login de un usuario de la base de datos. Por ejemplo, accede a `http://localhost:8080/AmigosSpring/testDAO.jsp?login=jsf` y comprueba que aparecen los datos de la usuaria "jsf".

Nótese que para simplificar, la aplicación no tiene capa de negocio. La capa de presentación va a acceder directamente a los DAOs, lo que no es en general una buena práctica.

Las dependencias de Spring 3 ya se han incluido en el pom.xml del proyecto, pero tendrás que hacer las siguientes tareas para configurarlo para Spring MVC:

- Échale un vistazo al XML de configuración de beans para la capa de datos, `src/main/webapp/WEB-INF/daos.xml`. Fijate en que está referenciado en el web.xml, como siempre hacemos para arrancar Spring. (Vale, en este punto no es que hayas tenido que hacer mucho, pero al menos recuerda que estos ficheros son necesarios, para cuando hagas tus propios proyectos).
- Modifica el web.xml para crear el servlet "dispatcher". Asócialo con las peticiones de tipo *.do, como hemos hecho en los apuntes.
- Crea el fichero XML de configuración de beans para la capa web, WEB-INF/dispatcher-servlet.xml (New > Spring Bean configuration file). En el segundo paso del asistente necesitamos el "context" y también el "mvc". Puedes tomar como modelo el "dispatcher-servlet.xml" de los apuntes. Pero cuidado, tendrás que cambiar el atributo "package" de la etiqueta component-scan por lo que consideres necesario (fijate en el package en que está el LoginController, que es donde vamos a colocar el resto de Controllers de Spring).
- Crea dentro del dispatcher-servlet.xml el bean "viewResolver", que asociará nombres lógicos de vistas con JSPs. Toma como modelo el de los apuntes, pero ten en cuenta que aquí los JSP no están dentro de ningún directorio especial (directamente están en webapp) por lo que la propiedad "prefix" sobra.

Comprueba que la configuración es correcta entrando en la aplicación (por ejemplo como

"jsf" con password "jsf"). . Una vez hecho login, debería aparecer la página del usuario actual, ya que el controlador que maneja el caso de uso "login" ya está implementado en la clase `es.ua.jtech.amigosspring.presentacion.LoginController`

2. MVC sin procesamiento de datos de entrada (1 punto)

Implementar la capa web para el caso de uso "ver amigo". Escribiendo un identificador de usuario y pulsando el botón "ver amigo" de la parte izquierda de la pantalla, veremos sus datos en un JSP ya incluido en la plantilla, "usuario.jsp".

Pasos a seguir para crear el controller:

1. Crear en el paquete `es.ua.jtech.amigosspring.presentacion` la clase `VerAmigoController`. Añádele la anotación `@Controller` para convertirlo en controlador de Spring.
2. La clase necesitará de un objeto de tipo `IUsuariosDAO` para hacer su trabajo, que debes inyectar con `@Autowired`
3. Crea un método java para procesar la petición:
   - Anota el método con `@RequestMapping` para asociarlo a la URL "verAmigo", que es a la que llama el formulario.
   - La petición disparada por el formulario tiene un único parámetro HTTP, de nombre "amigo", que es el del campo de texto donde se escribe el login del usuario a ver. Usa la anotación `@RequestParam` para asociar este parámetro HTTP con un parámetro del método java
   - El método tendrá un parámetro de tipo `Model` es para poder colocar el objeto Usuario. La vista, que es la página usuario.jsp, espera un Usuario bajo el nombre "amigo". (fíjate en que se usan expresiones del tipo ${amigo.edad})
   - Finalmente recuerda que el método java debe devolver un String que es el nombre lógico de la vista (en nuestro caso, el nombre real menos el .jsp)
   - Si el usuario no existe (si el DAO devuelve null), debes saltar a otra vista distinta. Crea una página noexiste.jsp que simplemente diga "el usuario no existe" y devuelve esta vista si el usuario buscado no existe.

Puedes probar a ver por ejemplo los usuarios `javaee` o `pepe`.

3. MVC con procesamiento de datos de entrada (1 punto)

Implementar la capa web para el caso de uso "buscar usuarios". Podemos buscar usuarios a partir de diversos criterios como edad, localidad, o sexo. El formulario de búsqueda está en la página "busqueda.jsp", ya incluída en las plantillas.

1. Definir una clase `es.ua.jtech.amigosspring.presentacion.CriteriosBusqueda` que pueda capturar los datos que se introducen en el formulario de búsqueda (con propiedades `int edadMin, int edadMax, String localidad, String sexo`)

2. Crear el *controller* de Spring
   1. Crear la clase `es.ua.jtech.amigosspring.presentacion.BuscarAmigosController`. Usando anotaciones, convertirla en controlador y mapearla con la URL "/busqueda"
   2. La clase necesitará de un `IUsuariosDAO` para hacer la búsqueda, igual que los otros Controllers.
   3. El controlador tendrá dos métodos: uno de ellos mapeado con GET y otro con POST:
      - `public String preparaForm()`: por el momento simplemente devolverá el nombre lógico de la vista con el formulario ("busqueda").
      - `public String procesaForm(CriteriosBusqueda criterios, Model modelo)`:
        - Tomando los datos del objeto "criterios", llama al método "buscar" del DAO para hacer la búsqueda.
        - Coloca el resultado en el modelo bajo el nombre "encontrados", ya que la vista JSP espera una List<Usuario> bajo ese nombre
        - Finalmente devuelve el nombre lógico de la vista que mostrará los datos ("encontrados"). Se corresponde con la página "encontrados.jsp" que ya está implementada.

3. Prueba a hacer búsquedas. **No dejes en blanco los campos de edad mínima y máxima**, porque si no dará error. Lo arreglaremos en el siguiente ejercicio.

4. Taglibs de Spring y validación básica de datos (1 punto)

Un problema de las búsquedas es que hay que escribir manualmente los valores para edad mínima y máxima. Vamos a poner unos valores por defecto sin tocar el HTML. Para ello:

1. Modifica el `preparaForm` del controller añadiéndole un parámetro de tipo `Model`. En este Model debes añadir como atributo un objeto de la clase `CriteriosBusqueda` con una edad mínima de 18 y máxima de 99.
2. Para que estos valores se muestren en el formulario hay que usar las taglibs de Spring. Cambia las etiquetas del formulario por las de Spring. Recuerda que el "modelAttribute" de la etiqueta "form:form" es el que vincula el formulario con el `CriteriosBusqueda` que has añadido al `Model` en `preparaForm`.
3. Comprueba que al acceder a la página de búsqueda aparecen los valores por defecto en el formulario.

Todavía nos queda por resolver la validación de datos. Si pruebas a introducir una edad mínima o máxima no numérica verás que se genera un error, al no ser posible vincular el campo del formulario con el dato de tipo entero. Vamos a filtrar los errores y mostrar un mensaje apropiado:

1. En el método `procesaForm` del controller define un parámetro de tipo `BindingResult` para "capturar" los errores. Tiene que estar **inmediatamente detrás**

del parámetro de tipo `CriteriosBusqueda` ya que son los errores asociados a él. Recuerda también que al usar esto debes obligatoriamente anotar el parámetro CriteriosBusqueda con @ModelAttribute

2. En el código del método, antes de llamar al DAO comprueba si el `BindingResult` contiene errores (si "hasErrors()" devuelve *true*). En ese caso, hay que ir de nuevo a la página de búsqueda.
3. Comprueba que al meter un dato no numérico en la edad se vuelve a mostrar la misma página de búsqueda
4. Nos falta todavía mostrar un mensaje de error al usuario indicando qué ha pasado. Hay que:
   - Crear el fichero .properties con los mensajes. Créalo en "src/main/resources".
   - Crear el bean "messageSource" en el dispatcher-servlet.xml, referenciando el .properties creado. Si lo has hecho en "resources" no es necesario poner el package, solo el nombre del .properties, sin la extensión.
   - Usar etiquetas `<form:errors/>` en el formulario para mostrar los errores asociados a la edad mínima y máxima. Lleva cuidado de meter las etiquetas dentro del formulario, si las pones fuera no se encontrará la referencia a los campos "edadMin" y "edadMax".
5. Comprueba que al dejar vacía la edad o introducir un valor no numérico se muestra el error en el JSP. Por el momento no vamos a validar otros errores, como por ejemplo edades negativas, eso lo haremos en sesiones posteriores.