1ª ENTREGA SDI – RED SOCIAL

Daniel Duque Barrientos – UO245553

Contenido

Implementación de los casos de uso

Caso 1: Registrarse como usuario

Implementé en este caso de uso una vista que consta con un formulario en el que hay que introducir los siguientes campos: Nombre, Email, Contraseña, Repetición contraseña.

Realicé un validador que comprueba que ninguno de los campos está vacío, que el email tiene el formato correcto (*alguien@ejemplo.com*) y que las contraseñas tienen entre 5 y 24 caracteres. También se valida que no exista el email introducido en la base de datos.

Una vez validados los campos se crea el usuario y se almacena en la base de datos y se realiza un auto inicio de sesión que introduce al usuario recién registrado en sesión.

Caso 2: Iniciar sesión

Implementé una vista con dos campos: Email y contraseña. Al introducir los datos se delega a *spring boot security* que se encarga de comprobar si existe un usuario en la base de datos con dicho email y contraseña. En caso de que no exista se muestra un mensaje en la misma vista que describe el error al usuario. Si existe un usuario con ese email y contraseña entonces se introduce a éste en sesión y se le redirige a la vista principal de la aplicación.

Caso 3: Listar todos los usuarios de la aplicación

Esta vista es la vista principal que se muestra cuando un usuario inicia sesión en el sistema. Consiste en una tabla que muestra todos los usuarios almacenados en la base de datos, visualizando sus nombres y emails. Para esto se recibe una petición *GET* en el controlador *UserController*. Dentro de la función se llama a un método del servicio *UserService* que apoyándose en el repositorio *UserRepository* obtiene una lista con todos los usuarios que hay en la base de datos. Esta lista está paginada y se muestra 5 usuarios por página. Una vez se obtiene la lista se añade un atributo al modelo para poder obtener una referencia en la vista y poder así mostrar los usuarios en una tabla.

Por último, añadí a la barra de navegación una opción para poder listar los usuarios si el usuario en sesión se encuentra en otra vista.

Dado que en el enunciado de la práctica no se especifica si se muestra o no el usuario en sesión en la lista de usuarios yo decidí no mostrarlo para permitir así que no pueda realizar peticiones de amistad a sí mismo.

Caso 4: Buscar entre todos los usuarios de la aplicación

Se añade a la vista de la lista de los usuarios de la aplicación un formulario con un campo el cuál se recibe como parámetro en la misma función que en la de listar los usuarios. Si el campo está vacío se realiza el mismo proceso que en el caso anterior. Si el campo contiene un valor se cambia la llamada del método que obtiene los usuarios

de la base de datos. Ahora el nuevo método añade a la consulta el valor del parámetro del campo para obtener la lista de usuarios que contienen en su nombre o email la cadena obtenida. Una vez obtenida la lista se realiza lo mismo que en el caso anterior. Se añade un atributo al modelo para poder referenciarlo en la vista y mostrar los usuarios en la tabla. También está paginado con 5 usuarios por página.

Caso 5: Enviar una petición de amistad a un usuario

Primero creé una nueva entidad llamada *Relationship* que almacena el usuario que realiza la petición y el usuario que la recibe. También consta de un atributo para poder saber si la petición ha sido aceptada o todavía sigue pendiente.

Para realizar la petición añadí a la vista de la lista de usuarios un botón por usuario que envía una petición *GET* a un nuevo controlador *RelationshipController* que la procesa en una función que se encarga de introducir un nuevo registro a la base de datos en la tabla *Relationship* y realiza las asociaciones pertinentes. Una vez actualizada la base de datos el botón se deshabilita para evitar que el usuario en sesión pueda volver a enviar una petición al mismo usuario.

Caso 6: Listar peticiones de amistad recibidas

Primero añadí una nueva opción a la barra de navegación para poder visualizar una lista de peticiones recibidas por el usuario en sesión.

A continuación, implementé una nueva vista que consta de una tabla que se encarga de mostrar el nombre de los usuarios que enviaron una petición de amistad al usuario en sesión. Para mostrar la tabla se realiza una petición *GET* al controlador *RelationshipController* que cuenta con una función que se encarga de obtener la lista de peticiones. Se llama a un método del servicio *RelationshipService* que con el repositorio *RelationshipRepository* obtiene de la tabla *Relationship* las filas que cumplen la siguiente condición: el usuario en sesión está en la columna de *recipient* (usuario que recibe la petición). A partir de esa lista se obtiene los usuarios que enviaron petición al usuario en sesión, se añaden como un atributo al modelo para poder referenciarlo en la vista y se muestra en la tabla. La lista con las peticiones está paginada y se muestran 5 peticiones por página.

Caso 7: Aceptar una petición recibida

En la vista del caso 6 añadí un botón para cada petición que permite aceptarla.

Cuando se pulsa el botón se realiza una petición *GET* al controlador *RelationshipController* que cuenta con una función que se encarga de procesar la aceptación de la petición.

Se llama a un método del servicio *RelationshipService* que con el repositorio *RelationshipRepository* se encarga de actualizar la fila correspondiente de la tabla *Relationship* modificando el valor de la columna *status.* Además, se crea una nueva entrada en la tabla ya que no solo el usuario1 es amigo del usuario2 si no que el usuario2 también es amigo del usuario1. De la forma en la que está implementado se

pueden realizar dos peticiones entre usuario1 usuario2 y usuario2 usuario1 y que en el momento en el que se acepta una de las dos peticiones ya se convierten en amigos y se eliminan las dos peticiones de la lista.

Caso 8: Listar usuarios amigos

Primero cree una nueva opción en la barra de navegación para poder acceder a la lista de usuarios amigos.

A continuación, implemente una vista similar a la implementada en el caso 3 que realiza una petición *GET* al controlador *RelationshipController* que cuenta con una función que se encarga de obtener la lista de usuarios amigos.

Se llama a un método del servicio *RelationshipService* que con el repositorio *RelationshipRepository* se obtiene de la tabla *Relationship* las filas cuyas columnas de *status* tienen el valor *FRIEND.* Se añade la lista obtenida al modelo para poder referenciarlo en la vista y poder mostrar la tabla. La lista está paginada con 5 usuarios por página.

Caso 9: Crear una nueva publicación

Primero cree una nueva entidad *Publication* para poder almacenar las publicaciones en la base de datos. Esta consta de un título, el texto, la fecha de creación y la referencia al usuario que la creó.

A continuación, implemente la vista que contiene un formulario con dos campos: Título y texto. A través de una petición *GET* al controlador PublicationController se almacena la nueva publicación en la base de datos.

Se llama a un método del servicio *PublicationService* que se encarga, mediante el repositorio *PublicationRepository,* de almacenar la publicación en la base de datos.

Caso 10: Listar mis publicaciones

Implementé una nueva vista que se encarga de visualizar todas las publicaciones realizadas por el usuario activo en ese momento. Se realiza una petición *GET* al controlador *PublicationController* que se encarga de obtener la lista de publicaciones del usuario.

Se llama a un método del servicio *PublicationService* que con el repositorio *PublicationRepository* retorna la lista de publicaciones.

A continuación, se añade la lista al modelo para que se pueda referenciar desde la vista y mostrar las publicaciones. Para cada publicación se muestra el título y la fecha de creación.

Caso 11: Listar las publicaciones de un usuario amigo

Implementé una nueva vista que se encarga de mostrar las publicaciones de un usuario seleccionado de la lista de amigos. Para ello primero modifiqué la vista de

“listar usuarios amigos” para que se muestre un botón que permita acceder a las publicaciones de dicho usuario.

A través de una petición *GET* que recibe el controlador *PublicationController* y con un parámetro (id del usuario) se encarga de obtener la lista de publicaciones. Se llama a un método del servicio *RelationshipService* que primero comprueba si el usuario activo y el usuario seleccionado son amigos, en caso contrario no se obtiene las publicaciones.

Luego se llama a un método del servicio *PublicationService* que, con el repositorio *PublicationRepository,* retorna las publicaciones para ese usuario. Al final se añade la lista al modelo para poder referenciarla en la vista. Para cada publicación se muestra el título, el texto y la fecha de creación.

Caso 12: Crear una publicación con una foto adjunta

Primero modifiqué la entidad *Publication* y añadí una nueva columna para almacenar el nombre de la imagen que contiene la publicación.

Después añadí un nuevo formulario a la vista de “crear publicación” que permite subir una foto a la aplicación y se almacena en el proyecto. Una vez almacenada la imagen para el caso “lista las publicaciones de un usuario amigo” ahora se muestra la imagen en caso de que la publicación contase con una imagen.

Caso 13: Iniciar sesión como administrador:

Creé una nueva vista similar al login del usuario pero que se accede mediante la siguiente petición: /admin/login. Esa petición la recibe el controlador *AdminController* que se encarga de comprobar si el usuario recibido es o no administrador. Para ello añadí un nuevo campo a la entidad *User* para poder saber el rol de cada usuario. En caso de que el usuario sea administrador se muestra la vista que contiene la “lista de usuarios de la aplicación”. En caso de que no sea administrador se muestra un mensaje de error.

Caso 14: Listar todos los usuarios de la aplicación

Primero implementé la vista para poder mostrar los usuarios, similar a la del caso de uso 3. Para gestionar esta petición empleé el controlador *AdminController* y realicé lo mismo que en caso de uso 3.

Caso 15: Eliminar usuario

En la vista del caso de uso 14 para cada usuario se muestra un botón que permite la eliminación de este. Para ello se realiza una petición *GET* al controlador *AdminController* con un parámetro (id del usuario). Primero elimino las publicaciones del usuario, las relaciones de amistad con otro usuario y por último lo elimino de la base de datos.

Implementación de las pruebas

1.1 RegVal

Registro de un usuario con datos válidos y comprobar que se introduce en sesión y se muestra la vista principal

1.2 RegInVal

Se prueba que no se permite registrar un usuario cuando la contraseña y la repetición de la contraseña no coinciden. Se comprueba que se muestra el error en la vista.

2.1 InVal

Se comprueba que se realiza el correcto inicio de sesión con un usuario existente en la base de datos y proporcionando la contraseña correcta. Se comprueba que se muestra la vista principal una vez se realiza el login.

2.2 InInVal

Se comprueba que no se puede realizar el inicio de sesión con un usuario que no existe en la base de datos. Se comprueba que se muestra el mensaje de error.

3.1 LisUsrVal

Se comprueba que se puede acceder a la lista de usuarios desde la opción de la barra de navegación. Se comprueba que se muestra el número correcto de usuarios.

3.2 LisUsrInVal

Se comprueba que no se puede acceder a la lista de usuarios introduciendo la *URL* que redirecciona a dicha lista. Se comprueba que automáticamente el sistema nos redirecciona a la vista de inicio de sesión.

4.1 BusUsrVal

Se comprueba que se realiza la correcta búsqueda de un usuario en el buscador de la vista de la lista de usuarios. Se comprueba que la tabla cuenta con las filas correctas.

4.2 BusUsrInVal

Se comprueba que no se puede acceder a la vista que contiene el buscador. Se comprueba que automáticamente el sistema nos redirecciona a la vista de inicio de sesión.

5.1 InvVal

Se comprueba que se realiza una petición de amistad correctamente. Se comprueba que una vez realizada la petición el botón se deshabilita.

5.2 InvInVal

Se comprueba que una vez realizada la petición el botón se deshabilita.

6.1 LisInvVal

Se realiza primero una petición de amistad a un usuario. A continuación, se inicia sesión con dicho usuario. Se comprueba que accediendo al listado de peticiones de amistad se obtiene una tabla con una fila.

7.1 AcepInvVal

Se realiza primero una petición de amistad a un usuario. A continuación, se inicia sesión con dicho usuario. Se comprueba que accediendo al listado de peticiones de amistad se obtiene una tabla con una fila. Se comprueba que al aceptar la petición ésta desaparece de la tabla y se muestra un mensaje de que no hay peticiones de amistad pendientes.

8.1 ListAmiVal

Se realiza el proceso de añadir un amigo. Una vez añadido se accede a la lista de usuarios amigos y se comprueba que la tabla consta de una fila. Se comprueba con el otro usuario que también contiene una fila la tabla en la vista de los usuarios amigos.

9.1 PubVal

Creo una publicación y compruebo que se creó correctamente.

10.1 LisPubVal

Creo una publicación y compruebo que se creó correctamente y se encuentra en la vista de mis publicaciones.

11.1 LisPubAmiVal

Creo una publicación con un usuario y compruebo que se muestra correctamente en la vista que muestra las publicaciones de un amigo.

11.2 LisPubAmiInVal

Compruebo que no se puede acceder directamente mediante una URL a las publicaciones de un usuario que no es amigo.

13.1 AdInVal

Compruebo que cuando se inicia sesión en “admin/login” con un usuario administrador se loguea correctamente y muestra todos los usuarios de la aplicación.

13.2 AdInInVal

Compruebo que cuando se inicia sesión en “admin/login” con un usuario no administrador se muestra un error en la propia vista.

14.1 AdLisUsrVal

Compruebo que cuando se inicia sesión en “admin/login” con un usuario administrador se loguea correctamente y muestra todos los usuarios de la aplicación.

15.1 AdBorUsrVal

Compruebo que se puede eliminar un usuario de la lista de usuarios de la aplicación siendo el usuario administrador.

15.2 AdBorUsrInVal

Compruebo que un usuario no administrador no pueda borrar un usuario de la aplicación introduciendo directamente la URL de borrado