SDI

BELUnCO

Red Social

Universidad de Oviedo

Profesores:

Edward Rolando Núñez Valdez

Jordán Pascual Espada

Vicente García Díaz

Enrique de la Cal Marín

PABLO BARAGAÑO COTO – LUCIA DE LA GRANDA

Contenido

CASOS DE USO

1 PÚBLICO: registrarse como usuario

Para este caso de uso, hemos seguido los siguientes pasos:

- Crear un *UsersRepository* que se encarga de tres tareas. Una de ellas es buscar un Usuario por su email, otra que es buscar todos los usuarios, y la última que es buscar los usuarios cuyo nombre/email coincide con una cadena de texto específica.
- Crear un *UsersService* que usa *UsersRepository* y que tiene un método *saveUser(User):Void* que lo que hace es encriptar la password del usuario que se está registrando, y salvarlo.
- Crear un *UsersController* que usa *UsersService,* y responde a una petición que comprueba que no haya ningún error en el registro del usuario y lo registra.

2 PÚBLICO: iniciar sesión

En UsersController tenemos un método login(Model, String):String cuyo parámetro String es opcional, ya que es el que nos indica si hay algún error, para poder mostrarlos a continuación. Finalmente se pasa a la vista login, en la cual aparece el formulario de inicio de sesión. Una vez completados los datos sin ningún error se pulsaría en login y pasaría a la pantalla de la lista de usuarios.

3 USUARIO REGISTRADO: listar todos los usuarios de la aplicación

Para este caso de uso hemos usado dos métodos implementados en *UsersService*, que son:

- searchByEmailAndName(String, Pageable):Page<User> que busca por email/nombre la cadena String en la lista de usuarios
- getUsers(Pageable):Page<User> que devuelve una lista de todos los usuarios de la aplicación

El método que se usa para listar a todos los usuarios en este caso es el segundo, que lo que hace es buscar mediante el *UsersRepository* todos los usuarios de la aplicación.

El parámetro String, que es el texto que deseas buscar en la lista de usuarios es opcional, por tanto cuando ese parámetro está presente se busca en la lista, y cuando no está se lista todos los usuarios de forma normal.

Además de todo esto, se introduce el parámetro Pageable para que se pagine la lista por 5 usuarios de la aplicación.

4 USUARIO REGISTRADO: buscar entre todos los usuarios de la aplicación

Como se ha explicado al final del caso anterior, en este caso hay que usar el parámetro String opcional, de forma que se usa el motor de búsqueda por nombre/email.

5 USUARIO REGISTRADO: enviar una invitación de amistad a un usuario

Para este caso, hemos seguido los siguientes pasos:

- Crear un *PeticionRepository* que tenga tres métodos. Uno que busca los usuarios que mandan peticiones de amistad, otro que lista los que reciben peticiones, y otro que te dice si un usuario tiene peticiones recibidas.
- Crear un *PeticionService* que usa *PeticionRepository* y que se encarga de implementar los métodos anteriores.

Ahora bien, desde *UsersController* se hace una petición a un usuario con *makePeticion(Model, Long):String* que lo que hace es buscar el usuario por id, sacar el usuario actual que está en sesión, y llamar al método de *PeticionService* que crea la petición con un *save.*

6 USUARIO REGISTRADO: listar las invitaciones de amistad recibidas

Para este caso de uso hemos seguido los siguientes pasos:

- Crear un *PeticionController* que tenga un métido que lista todos las peticiones que recibió el usuario actual que está en sesión. Para ello usa un método *getFriendsList():Page<User>* que está implementado en la entidad *User*.

Para listarlos se hace de la misma forma que se hace en el caso anterior de listar los usuarios registrados.

7 USUARIO REGISTRADO: aceptar una invitación recibida

En la lista de peticiones de amistad recibidas implementada en el caso anterior hemos añadido un botín que sea "ACEPTAR PETICIÓN". Mediante ese botón hacemos que los dos usuarios sean amigos.

Para ello hemos creado en *UsersController* un método que añade como amigo al usuario que lanzó la petición, y elimina dicha petición por que ya está aceptada.

8 USUARIO REGISTRADO: listar los usuarios amigos

Al igual que todos los listados anteriores se hace este, creando el método que lista los usuarios en una controller llamado *FriendsController.*

9 USUARIO REGISTRADO: crear una publicación

Para llevar a cabo esta ampliación lo que hemos hecho es añadir una lista de publicaciones en la entidiad user, con un mapeo @OneToMany, creado la entidad Publicaciones, de este modo tenemos en la base de datos una lista de publicaciones asociadas a cada usuario correspondiente.

Hemos creado la vista del formulario para crear la publicación, consta de un titulo y de la descripción ó texto de la publicación.

Para gestionarlas hemos creado un nuevo controller (PublicationController.java), este recibe las peticiones post y se encarga de crear la publicación, para crearla necesitamos al autor, el titulo y la descripción, la fecha se genera automáticamente en la clase entidad. Para almacenarla en la base de datos creamos un servicio (PublicationService.java) el cual llama al repositorio (PublicationRepository.java) que haciendo un save() guarda la publicación en la base.

10 USUARIO REGISTRADO: listar mis publicaciones

En este caso lo que hacemos en una nueva vista que genere una lista con las publicaciones realizadas por el usuario en sesión. Añadimos un método al controller que responde a esta petición (`getListadoPublication()`), devuelve una lista con las publicaciones realizadas por el usuario que se encuentran en la base de datos.

11 USUARIO REGISTRADO: listar las publicaciones de un usuario amigo

Para realizar esta ampliación modificamos la vista que listaba a los amigos, añadimos un hiperenlace al nombre del usuario amigo que te redirige a la lista de las publicaciones realizadas por dicho amigo, de igual manera que en el caso anterior, se crea una vista que lista las publicaciones, en este caso en vez de por el usuario en sesión se busca por el usuario amigo, sacando de la base de datos todas sus publicaciones y añadiéndolas al modelo para poder listarlas. (`getListadoPublicacionesFriend()`).

En el caso de que el usuario, del cual intentas listar las publicaciones, no sea tu amigo, la vista se generará, no hay error de acceso, pero no se te mostrarán sus publicaciones, aunque las tenga.

12 USUARIO REGISTRADO: crear una publicación con una foto adjunta

No se ha implementado esta ampliación, y por ello, tampoco sus pruebas correspondientes.

13 PÚBLICO: iniciar sesión como administrador

Para este caso hemos tenido que crear un nuevo caso de configuración de inicio de sesión en WebSecurotyConfig, para asegurarnos de que solo damos acceso a los administradores, una vez tenemos esta nueva configuración, creamos una vista para su inicio de sesión igual a la que ya teníamos para los usuarios no administradores. Con esta nueva configuración también vetamos a los usuarios no administradores de acceder a ciertas zonas de la parte de administración, como eliminar un usuario.

También hemos implementado un servicio de roles (RolesService) para poder diferenciar los tipos de usuario de la aplicación, `ROLE_STUDENT` o `ROLE_ADMIN`.

En el caso de que un usuario sin privilegios intente logearse en esta zona de la aplicación se lanzará un mensaje de acceso denegado.

El método de logeo se gestiona en el controller (AdminController).

14 CONSOLA DE ADMINISTRACIÓN: listar todos los usuarios de la aplicación

Una vez nos logeamos satisfactoriamente como administrador podemos tener acceso a una lista completa de los usuarios de la aplicación, la vista muestra 5 usuarios por página. Para esto creamos un controller (AdminController) que se encarga de llamar al service (AdminService) para sacar la lista completa de usuarios mediante el repository (AdminRepository). Esa lista se pasa a la vista y esta se encarga de listarlos.

15 CONSOLA DE ADMINISTRACIÓN: eliminar usuario

Modificamos la lista de la que hablamos en el punto 14 para añadirle un botón de eliminar usuario, montamos en botón añadiéndole código javascript para conseguir sacar el id del usuario a eliminar, actualizar la lista y deshabilitar el botón una vez lo hemos pulsado. Cuando accionamos ese botón se manda una petición al controller (AdminController) pasándole la url con el id del usuario a eliminar, este método se encarga de eliminar las relaciones de amistad que existan con ese usuario llamando al método “unlink” de la entidad User, una vez quitadas esas relaciones se elimina el usuario de la base de datos, las demás relaciones no hace falta desenlazarlas porque tenemos el tipo de borrado en cascada, por lo que se eliminarán a la vez que el usuario. Para eliminar el usuario llamamos al service (AdminService) que se encargará de llamar al repository (AdminRepository) para eliminar el usuario que se le pasa por parámetro.

Un usuario que no sea administrador no podrá acceder a estas partes de la aplicación, ya que en el securityService está implementado de tal manera que le de error de acceso.

TEST CASOS DE PRUEBA

Caso prueba RegVal()

Empezamos clicando en la opción de registrarse, y rellenamos los campos del formulario, de forma correcta (misma contraseña, etc). Finalmente se comprueba que al registrarse, aparece la opción de “Ver Usuarios”.

Caso prueba RegInval()

Empezamos clicando en la opción de registrarse, y rellenamos los campos del formulario, de forma incorrecta (distinta contraseña). Finalmente se comprueba que al registrarse, aparece el error que se comprueba.

Caso prueba InVal()

Empezamos clicando en la opción de iniciar sesión, y rellenamos los campos del formulario de inicio de sesión, de forma correcta (con un usuario correcto que ya ha sido previamente registrado). Finalmente se comprueba que al registrarse, aparece la opción de “Ver Usuarios”.

Caso prueba InInVal()

Empezamos clicando en la opción de iniciar sesión, y rellenamos los campos del formulario de inicio de sesión, de forma incorrecta (con un usuario correcto que no ha sido previamente registrado). Finalmente se comprueba que al registrarse, aparece el error que se comprueba.

Caso prueba LisUsrVal()

Empezamos clicando en la opción de iniciar sesión, y rellenamos los campos del formulario de inicio de sesión, de forma correcta (con un usuario correcto que ya ha sido previamente registrado). Finalmente se comprueba que al registrarse, aparece la opción de “Ver Usuarios”. Se obtiene el número de usuarios registrados, 5 por cada página debido a la paginación.

Finalmente se desconecta la sesión.

Caso prueba LisUsrInval()

Empezamos haciendo que se abra directamente la lista de usuarios, sin haberse previamente registrado/identificado. Por tanto, nos redirecciona a la ventana de identificación de la sesión.

Caso prueba BusUsrVal()

Empezamos clicando en la opción de iniciar sesión, y rellenamos los campos del formulario de inicio de sesión, de forma correcta (con un usuario correcto que ya ha sido previamente registrado). Comprobamos que se ha iniciado sesión correctamente ya que aparece la opción de "Ver Usuarios". A continuación se comprueba que al registrarse, se obtiene el número de usuarios registrados, 5 por cada página debido a la paginación.

Pasamos a pulsar en el motor de búsqueda, introducir un nombre que sabemos que está registrado en la web, y a buscarlo.

Finalmente se desconecta la sesión.

Caso prueba BusUsrInVal()

Empezamos haciendo que se abra directamente la lista de usuarios para buscar a uno de ellos, sin haberse previamente registrado/identificado. Por tanto, nos redirecciona a la ventana de identificación de la sesión.

Caso prueba InvVal()

Empezamos clicando en la opción de iniciar sesión, y rellenamos los campos del formulario de inicio de sesión, de forma correcta (con un usuario correcto que ya ha sido previamente registrado). Comprobamos que se ha iniciado sesión correctamente ya que aparece la opción de "Ver Usuarios". A continuación se comprueba que al registrarse, se obtiene el número de usuarios registrados, 5 por cada página debido a la paginación.

Buscamos el usuario que se indica en la prueba, para posteriormente mandarle una petición. Lo hacemos así para que siempre haya un usuario creado disponible para mandarle la petición, ya que el usuario siempre existe por estar creado en las pruebas anteriores.

Pasamos a pulsar en "AÑADIR AMIGO", un botón que aparece en la lista de los usuarios, para mandarle una petición.

Finalmente se desconecta la sesión.

Caso prueba InvInVal()

Empezamos clicando en la opción de iniciar sesión, y rellenamos los campos del formulario de inicio de sesión, de forma correcta (con un usuario correcto que ya ha sido previamente registrado). Comprobamos que se ha iniciado sesión correctamente ya que aparece la opción de “Ver Usuarios”. A continuación se comprueba que al registrarse, se obtiene el número de usuarios registrados, 5 por cada página debido a la paginación.

Buscamos al usuario por la misma razón que la prueba anterior.

Pasamos a pulsar en “AÑADIR AMIGO”, un botón que aparece en la lista de los usuarios, para mandarle una petición. Comprobamos que si intentamos pulsar otra vez, ya que el botón queda deshabilitado una vez se manda la petición.

Finalmente se desconecta la sesión.

Caso prueba LisInvVal()

Empezamos clicando en la opción de registrarse y rellenamos los campos del formulario de forma correcta (mismas contraseñas) y el email de forma random(ya que lo hemos decidido así). Comprobamos que se ha iniciado sesión correctamente ya que aparece la opción de “Ver Usuarios”. A continuación se comprueba que al registrarse, se obtiene el número de usuarios registrados, 5 por cada página debido a la paginación.

Pasamos a pulsar en “AÑADIR AMIGO”, un botón que aparece en la lista de los usuarios, para mandarle una petición. En este caso de prueba se manda la petición al usuario de email “1”.

Pulsamos en “Ver peticiones” para ver las peticiones recibidas, y comprobamos que tenemos una nueva petición recibida.

Finalmente se desconecta la sesión.

Caso prueba AcepInvVal()

Empezamos clicando en la opción de iniciar sesión, y rellenamos los campos del formulario de inicio de sesión, de forma correcta (con un usuario correcto que ya ha sido previamente registrado). Comprobamos que se ha iniciado sesión correctamente ya que aparece la opción de "Ver Usuarios". A continuación se comprueba que al registrarse, se obtiene el número de usuarios registrados, 5 por cada página debido a la paginación.

Pasamos a pulsar en "AÑADIR AMIGO", un botón que aparece en la lista de los usuarios, para mandarle una petición.

Pulsamos en "Ver peticiones" para ver las peticiones recibidas, y comprobamos que tenemos una petición recibida. Comprobamos también que podemos aceptarla pero tan solo una vez, ya que el botón queda inhabilitado.

Finalmente se desconecta la sesión.

Caso prueba ListAmiVal()

Empezamos como siempre iniciando sesión en la web con el usuario, en este caso, de email "2". Pulsamos en "Ver amigos", para ver los amigos que tiene el usuario, y vemos que tiene un usuario en sus amistades.

Caso prueba PubVal()

Empezamos iniciándonos en la aplicación con un usuario válido, previamente registrado. Accedemos al menú de las publicaciones, y después a la opción de añadir publicación. Añadimos una publicación de prueba rellenando el formulario y cerramos sesión.

Caso prueba ListPubVal()

Empezamos iniciándonos en la aplicación con un usuario válido, previamente registrado. Accedemos al menú de las publicaciones, y después a la opción de listar las publicaciones. Comprobamos que hay una publicación de prueba rellenada en la prueba anterior, y cerramos sesión.

Caso prueba ListPubAmiVal()

Empezamos iniciándonos en la aplicación con un usuario válido, previamente registrado. Accedemos a la lista de usuarios, y después a la opción de listar las publicaciones de uno de esos usuarios. Comprobamos que accede a la página que contiene la lista de publicaciones, y cerramos sesión.

Caso prueba ListPubAmiInVal()

Empezamos iniciándonos en la aplicación con un usuario válido, previamente registrado. Accedemos a la lista de publicaciones de un usuario que no es amigo del usuario actual. Comprobamos que solo sale la tabla pero sin ninguna publicación (size=1 ya que cuenta la primera fila de la tabla). Cerramos sesión.

Caso de prueba PubFot1Val

No se ha realizado la ampliación para esta prueba.

Caso de prueba PubFot2Val

No se ha realizado la ampliación para esta prueba.

Caso de prueba AdInVal

Empezamos accediendo a la url de inicio de sesión par aun administrador (http://localhost:8090/admin/login) , rellenamos el formulario del inicio con un usuario que tiene rol de administrador (lucia@gmail.com,123456). Comprobamos que nos lista a los 5 primeros usuarios existentes en la aplicación. Por último, cerramos sesión.

Caso de prueba AdInInVal

Empezamos accediendo a la url de inicio de sesión par aun administrador (http://localhost:8090/admin/login) , rellenamos el formulario del inicio con un usuario que no tiene rol de administrador (pedrito@gmail.com,123456). Comprobamos que nos salta un error de acceso denegado, pues este usuario no puede entrar a esta parte de la aplicación.

Caso de prueba AdLisUsrVal

Es igual al caso de prueba AdInVal, pues en este ya comprobábamos que listaba a los 5 primeros usuarios al estar paginado.

Caso de prueba AdBorUsrVal

Creamos un usuario que, si la base de datos es la básica, será el último que saldrá en la lista de usuarios. Accedemos a la url de inicio de sesión par aun administrador (http://localhost:8090/admin/login) , rellenamos el formulario del inicio con un usuario que tiene rol de administrador (lucia@gmail.com,123456). Comprobamos que nos lista a los 5 primeros usuarios existentes en la aplicación. Y borramos el último usuario de la lista, que será el que acabamos de crear.

El crear un nuevo usuario es una forma de evitar tener que estar creando la base de datos otra vez, es decir, lanzando el proyecto principal otra vez. Ya que si no borrásemos un usuario recién creado al pasar las pruebas por segunda vez nos daría error, ya que en la base no existe ya tal usuario.

Caso de prueba AdBorUsrInVal

Accedemos a la aplicación con un usuario sin rol de administrador, intentamos acceder a la url (http://localhost:8090/admin/2/delete) que estaría intentando borrar el usuario número 2, pero como es un usuario que no tiene derechos de administrador nos salta un error de acceso denegado.

COMENTARIOS ADICIONALES

Para alguno de los casos de prueba es posible que, si se ejecutan más de un par de veces seguidas, no funcionen correctamente, ya que en alguna de ellas se eliminan usuarios y se interactúa con la base de datos, cambiando datos que hacen que hacen fallar las pruebas.

En el caso de que de algún error, se lanza otra vez la aplicación principal, generando de nuevo la base de datos y las pruebas volverán a funcionan correctamente.

Si se modifica la base de datos, por ejemplo hacer que el usuario de email 1 y el 2 sean amigos, una de las pruebas tampoco pasaría. Ya que en ella se comprueba que no sean amigos.

También hemos querido destacar que el Firefox lo hemos añadido al proyecto de la entrega, y que, por tanto, no hace falta cambiar la URL si se cambia de ordenador.

La base de datos se entrega vacía y en el application.propierties está en modo “create”, de manera que cuando se ejecute la aplicación principal se cargaran los datos de la clase “InserSampleDataService”.