Red Social

DOCUMENTACIÓN SDI PRÁCTICA 1

Carlos Sanabria Miranda UO250707

Alejandro Barrera Sánchez UO251893

Tabla de contenido

Despliegue

Para arrancar la base de datos, acceder a la carpeta /hsqldb/bin y utilizar el runServer.bat (o el runServer.sh en caso de desplegarse en Mac).

A la hora de probar los tests, habrá que cambiar la línea correspondiente al Path de Firefox, para que corresponda con la ruta donde se encuentra dicho navegador en la máquina en la que se van a ejecutar.

```
static String PathFirefox = "<ruta_navegador_firefox_46.0>”
```

Descripción de los Casos de Uso

1. Público: registrarse como usuario

Se parte de una entidad User con los campos: email, nombre, apellidos, contraseña, confirmación de contraseña y role.

El role de los usuarios normales es ROLE_PUBLIC.

Se dispone de una vista “signup.html” con un formulario, que permite rellenar todos los campos anteriores (salvo el role). Dicho formulario hace una petición POST a la URL “/signup”.

En UsersController hay un método que recibe la petición GET “/signup” y devuelve la vista indicada, y otro que recibe la petición POST “/signup” y aplica el validador SignUpFormValidator, que hace varias comprobaciones de los datos, entre ellas que no exista ya un usuario con dicho email en el sistema.

Si no hubo errores al validar los datos, se le da ROLE_PUBLIC al usuario, se llama al método “addUser” del servicio “UsersService” que lo añade, y se le hace un auto inicio de sesión.

Si hubo errores, se retornará de nuevo la vista, donde serán mostrados con thymeleaf en el idioma actual.

El método “addUser” de “UsersService” encripta la contraseña dada por el usuario y llama al repositorio “UsersRepository” para guardar el usuario en la base de datos.

2. Público: iniciar sesión

Se dispone de una vista “login.html” con un formulario con los campos “username” y “password” y que hace una petición POST a la URL “/login”.

Como estamos utilizando Spring Security para gestionar el login, no tenemos que definir ningún método en el controlador “UsersController” para la petición POST anterior, pero sí que definimos un método que recibe la petición GET “/login”, que devuelve la vista “login.html” y que guarda un atributo “login” en sesión indicando que el usuario intenta acceder desde “/login”.

Dentro de la clase WebSecurityConfig definimos: que todo el mundo tiene acceso a la página de login, que si se tiene éxito al autenticarse se redirige al usuario al listado de todos los usuarios (“/user/list”) y que si los datos no son correctos, se redirige al usuario a la página de login con un parámetro de error.

```
.formLogin()
        .loginPage("/login").permitAll()
        .defaultSuccessUrl("/user/list")
        .failureUrl("/login?error")
```

3. Usuario registrado: listar todos los usuarios de la aplicación

Se dispone de una vista “user/list.html” con una tabla. La vista recorre una lista de usuarios que se le pasa a través del modelo, y por cada usuario muestra su nombre y su email en una fila de la tabla.

El fragmento “pagination.html” incluido en la vista anterior muestra enlaces a las páginas del listado.

En UsersController hay un método que recibe la petición GET “/user/list” y devuelve la vista anterior. En ese método, si no hay un parámetro “searchText” (para realizar una búsqueda) en la petición, se llama al método “getUsers” de “UsersService”, pasándole el parámetro “pageable”, que obtiene un objeto “Page<User>” con los usuarios de la página actual.

El controlador guarda en el modelo la lista de usuarios que proporciona el Page<User> y el propio Page<User>, para poder sacar información que se mostrará en los enlaces de paginación.

El método “getUsers” de “UsersService” que tiene el parámetro Pageable, llama a “UsersRepository”, pidiéndole todos los usuarios que tengan ROLE_PUBLIC (sólo nos interesa mostrar los usuarios reales, no aquellos que tengan ROLE_ADMIN).

Para que la paginación muestre 5 usuarios por página, en el método “addArgumentResolvers” de la clase “CustomConfiguration” se realiza esto:

```
resolver.setFallbackPageable(new PageRequest(0, 5));
```

Para añadir una opción de menú que permita acceder al listado, se ha incluido en el fragmento “nav.html” un dropdown-menu, con una opción que tiene un enlace a dicho listado.

Las opciones de la parte izquierda del navbar (las que incluyen gestión de usuarios y publicaciones) están dentro de una lista que contiene el atributo sec:authorize="isAuthenticated()" , con lo que solo se muestran para los usuarios autenticados.

4. Usuario registrado: buscar entre todos los usuarios de la aplicación

Se parte de la vista y los métodos del controlador indicados en el punto anterior.

En la vista se añade un formulario con un único campo, que es el texto a buscar, y que realiza una petición GET a la URL “/user/list”, que es recogida por el método del controlador.

Dicho método, cuando recibe una petición con el parámetro “searchText”, en lugar de llamar al método “getUsers” de “UsersService”, llama a “searchUsersByEmailAndName”, que devuelve un Page<User> con sólo aquellos usuarios con ROLE_PUBLIC que tengan el texto indicado por searchText en el email o el nombre.

Para ello, añade esto al texto pasado como parámetro: searchText = "%" + searchText + "%"; (para que no busque por coincidencia exacta) y llama a “UsersRepository”, que realiza la siguiente consulta:

```
@Query("SELECT u FROM User u WHERE u.role = ?2 AND "
                        + "(LOWER(u.name + ' ' + u.lastName) LIKE LOWER(?1) OR LOWER(u.email) LIKE LOWER(?1))")

Page<User> searchByEmailAndNameByRole(Pageable pageable, String seachtext, String role);
```

5. Usuario registrado: enviar una invitación de amistad a un usuario

Creamos el controlador InvitationController para controlar las invitaciones de amistad. Creamos también InvitationService, para la lógica de las invitaciones e InvitationRepository para la base de datos.

También creamos la entidad Invitation y las propiedades sendedInvitations y receivedInvitations en user.

El controller mapea la url "user/invitate/{id}" y le pasa al servicio el email del usuario autenticado y el id del usuario al que quiere agregar. Este se comunica con userRepository para recuperar de la BBDD los dos usuarios, si ningún usuario es null y el usuario que envía la invitación es distinto al que la recibe, crea y guarda una nueva invitación.

6. Usuario registrado: listar las invitaciones de amistad recibidas

Mapeamos en InvitationController la url "user/invitations", el controlador le pide a InvitationService la lista paginada de las invitaciones que ha recibido el usuario autenticado, el servicio pide las invitaciones recibidad paginadas a InvitationRepository.

La query usada en el repositorio es: @Query("SELECT i FROM Invitation i WHERE i.receiver.email = ?1").

Al final el controlador devuelve una vista con paginación de las invitaciones que ha recibido el usuario.

7. Usuario registrado: aceptar una invitación recibida

Mapeamos en InvitationController la url "user/accept/{id}", este se comunica con InvitationService para aceptar la invitación. El servicio hace una comprobación de seguridad de que el usuario que acepta la petición con el id indicado en la url sea realmente el usuario que recibe la invitación.

El servicio hace uso de la lógica implementada en el model para aceptar la invitación y, por último, si todo ha salido bien, pide al InvitationRepository que borre de la BBDD la invitación.

8. Usuario registrado: listar los usuarios amigos

Mapeamos en UserController la url "/user/friends", este controlador se comunica con UserService para obtener una lista paginada de amigos y este, a su vez, se comunica con el repositorio UserRepository.

La query usada es: @Query("SELECT u FROM User u JOIN u.friends f WHERE f.email = ?1")

Por último, le pasa a thymeleaf un objeto Model con la lista y la información de paginación, para que este muestre al usuario la vista pedida.

9. Usuario registrado: crear una nueva publicación

Se parte de una entidad Post con los campos: título, texto, fecha y usuario que la ha creado. La entidad User pasa a tener un conjunto de posts.

Se dispone de una vista “post/add.html” con un formulario con los campos “title” y “text”, que hace una petición POST a la URL “/post/add”.

En la clase “PostsController” hay un método que responde a GET “/post/add”, que devuelve la vista anterior, y un método que responde a POST “/post/add”, que valida los datos del formulario con “postAddFormValidator” (comprobaciones de longitud de título y texto).

Si no hubo errores al validar los datos, se obtiene el usuario en sesión y la fecha actual y se le asignan a la publicación. Seguidamente, se llama al método “addPost” de “PostsService” para que añada el nuevo post a la base de datos.

Dicho método simplemente llama a “PostsRepository” para guardar el post.

Si hubo errores, se retornará de nuevo la vista, donde serán mostrados con thymeleaf en el idioma actual.

Se incluye un dropdown-menu “Publicaciones”, con una opción que tiene un enlace a “/post/add”.

Aparte de que las opciones de la parte izquierda del navbar sólo se muestran para usuarios en sesión, se ha añadido al dropdown-menu de las publicaciones el siguiente atributo sec:authorize="hasRole('ROLE_PUBLIC')" , para que sólo los usuarios normales tengan acceso a las opciones relacionadas con las publicaciones, y en WebSecurityConfig se ha añadido .antMatchers("/post/**").hasAnyAuthority("ROLE_PUBLIC").

(Que sólo los usuarios con ROLE_PUBLIC puedan acceder a funciones relacionadas con los posts no era imprescindible, pero hemos pensado que tenía más sentido hacerlo así, ya que los administradores sólo van a tratar con los usuarios, no con sus publicaciones.)

10. Usuario registrado: listar mis publicaciones

Se dispone de una vista “post/list.html” que recorre una lista de posts que se le pasa a través del modelo, y por cada post muestra su título y su fecha.

En la clase “PostsController” hay un método que responde a GET “/post/list”, que obtiene el usuario en sesión, obtiene sus posts llamando al método “getPostsForUser” de “PostsService” y devuelve la vista anterior pasándole los posts a través del modelo.

El método “getPostsForUser” simplemente llama a “PostsRepository” para obtener todos los posts del usuario que se le pasa como parámetro.

Se incluye una opción en el dropdown-menu “Publicaciones” que tiene un enlace a “/post/list”.

La visibilidad de dicha opción se gestiona de la misma forma que la explicada en el punto anterior.

11. Usuario registrado: listar las publicaciones de un usuario amigo

En PostController mapeamos "/post/list/{id}", este controlador se comunica con UserService para obtener los usuarios implicados en la operación y comprueba que el usuario con el id de la url sea amigo del usuario autenticado. Si no es amigo retorna al usuario la página principal.

Si es amigo, pide a PostService una lista con las publicaciones del usuario amigo. Devuelve a thymeleaf la lista a través del objeto model y crea una entrada en el log de la aplicación. Por último, devuelve la vista “post/list”.

12. Usuario registrado: crear una publicación con una foto adjunta

Modificamos en el controlador PostController el método que mapea el post de la url “/post/add”, le añadimos un nuevo parámetro @RequestParam("image") MultipartFile image. Si el nuevo parámetro no está vacio, guardamos en el servidor la imagen. Le ponemos de título el id del post.

Añadimos una nueva propiedad a la entidad post, containsImage. Para que thymeleaf sepa si tiene que mostrar una imagen o no.

En aplication.properties definimos que el tamaño máximo del archivo son 10MB.

En el html de la vista “/post/list” agregamos lo siguiente:

```
<div class="panel-body" th:if="${post.containsImage}">
    <img th:src="${'/img/' + post.id}" +  th:alt="#{post.list.image}" />
</div>
```

13. Público: iniciar sesión como administrador

Los usuarios administradores son Users que tiene como role ROLE_ADMIN.

En la clase “PostsController” hay un método que responde a GET “/admin/login”, que guarda un atributo “login” en sesión indicando que el usuario intenta acceder desde “/admin/login”, y retorna la vista que se explica a continuación.

Se dispone de una vista “admin/login.html” con un formulario con los campos “username” y “password” y que hace una petición POST a la URL “/login” (esto se debe a que, como se explica en el punto 2, estamos utilizando Spring Security para gestionar la autenticación).

Si los datos corresponden a los de un usuario existente en la aplicación (independientemente del role que tenga), se redirigirá correctamente al usuario a “/user/list” (la vista que muestra todos los usuarios), tal y como se indica en WebSecurityConfig (explicado también en el punto 2).

En el método que recibe la petición GET “/user/list” obtenemos el atributo “login” de la sesión, y el role del usuario en sesión. Si el atributo “login” de la sesión es “/admin/login” y el role del usuario no es ROLE_ADMIN, deslogeamos al usuario y lo redirigimos a “/admin/login” con un error debido al role del usuario.

```
if(urlLogin!= null && urlLogin.equals("/admin/login") && !role.equals("ROLE_ADMIN")) {
                securityService.logoutUserInSession();
                loggerService.errorByRoleInAdminLogin(email);
                return "redirect:/admin/login?error=role";
}
```

En GET “/login”, antes de guardar en el atributo “login” que se está en el formulario “/login”, se comprueba si se viene de “/admin/login” y si ha habido un error. En ese caso, es que se ha producido un error en “/admin/login” debido a que las credenciales no se corresponden con ningún usuario existente, por lo que se redirige al usuario a “/admin/login” con un error debido a las credenciales.

```
String urlLogin = (String) httpSession.getAttribute("login");
if(urlLogin != null && urlLogin.equals("/admin/login") && error != null) {
		loggerService.errorByCredentialsInAdminLogin();
		return "redirect:/admin/login?error=credentials";
}
```

En la vista con thymeleaf se mostrará el mensaje de error correspondiente en el idioma actual.

(Se ha incluido en la vista /login un enlace a /admin/login para facilitar su acceso)

14. Consola de administración: listar todos los usuarios de la aplicación

Se dispone de la vista “user/list.html” explicada en el punto 3.

El comportamiento es el mismo que en dicho punto 3. Como se establece en el WebSecurityConfig, tanto los usuarios normales como los administradores tienen acceso a las URL que empiezan por “/user”, lo cual incluye a la URL “/user/list” que es la que muestra la vista anterior.

```
antMatchers("/user/**").hasAnyAuthority("ROLE_PUBLIC", "ROLE_ADMIN")
```

15. Consola de administración: Consola de administración: eliminar usuario

Dentro de la tabla de la vista “user/list”, se añade una celda con el atributo sec:authorize="hasRole('ROLE_ADMIN')", por lo que el contenido de dicha celda sólo estará disponible para administradores.

Esa celda se creará para cada usuario de la tabla, y contiene un botón con un id (al que se le concatena el id del usuario), que es referenciado desde JQuery, y manda una petición GET a la URL “/user/delete/<id>” cuando se clicka el botón, donde <id> es el id del usuario de esa fila de la tabla.

En la clase “UsersController” hay un método que responde a GET “/user/delete/{id}”, que llama al método “deleteUser” de “UsersService”. Dicho método simplemente llama a “UsersRepository” para eliminar el usuario.

Los posts del usuario están definidos de la siguiente forma:

```
@OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
private Set<Post> posts;
```

Por lo que, si se elimina el usuario, se eliminan automáticamente todos sus posts. Al eliminar el usuario, dejará también de figurar como amigo de otros usuarios automáticamente.

En WebSecurityConfig se especifica que solo los administradores pueden llamar a dicha URL:

```
antMatchers("/user/delete/*").hasAuthority("ROLE_ADMIN")
```

Cuando se elimina un usuario, no se carga de nuevo la página entera, sino que se actualiza el fragmento que contiene la tabla y los enlaces de paginación:

```
$("#tableUsersAndPagination").load(urlUpdate);
```

Descripción de las Pruebas Unitarias

1.1 [RegVal] Registro de Usuario con datos válidos.

Va a la página de signup, introduce datos válidos (entre ellos un email que no existe ya en la aplicación) y comprueba que se autologea correctamente (comprueba que va a una página en la que aparece el siguiente texto: “Usuario autenticado: <email>”, donde <email> es el email del usuario que se acaba de registrar).

Finalmente, se deslogea y comprueba que le lleva a la página de login (aparece el texto “Identifícate”).

1.2 [RegInval] Registro de Usuario con datos inválidos (repetición de contraseña inválida).

Va a la página de signup, introduce datos válidos, salvo la contraseña, que al repetirla la introduce incorrectamente. Comprueba que le lleva a la página de login mostrándole el error correspondiente. Para ello, comprueba que aparece un texto con la clave “Error.signup.passwordConfirm.coincidence” en idioma Español.

2.1 [InVal] Inicio de sesión con datos válidos.

Va a la página de login, introduce datos válidos de un usuario existente en la aplicación con ROLE_PUBLIC, y comprueba que se autologea correctamente (comprueba que va a una página en la que aparece el siguiente texto: “Usuario autenticado: <email>”, donde <email> es el email del usuario que se acaba de autenticar).

Finalmente, se deslogea y comprueba que le lleva a la página de login (aparece el texto “Identifícate”).

2.2 [InInVal] Inicio de sesión con datos inválidos (usuario no existente en la aplicación).

Va a la página de login, introduce datos inválidos (el usuario con esos datos no existe) y comprueba que se muestra un mensaje de error (un texto con la clave “Error.login” en el idioma Español).

3.1 [LisUsrVal] Acceso al listado de usuarios desde un usuario en sesión.

Va a la página de login, introduce datos válidos de un usuario con ROLE_PUBLIC y comprueba que inició sesión correctamente (como se indica en 2.1).

Hace click en el nav en la opción “Usuarios” y espera a que se despliegue el dropdown-menu y aparezca la opción “Ver Todos”. Hace click en esa opción para ir a la página con el listado de todos los usuarios y comprueba que aparece el texto “Todos los usuarios”.

(Se podría comprobar que aparece el texto “Todos los usuarios” directamente al iniciar sesión, ya que un inicio de sesión correcto nos redirige a la página con el listado de todos los usuarios, pero haciéndolo de esta forma nos aseguramos de que funciona correctamente la opción del nav para ir a dicha página.)

Finalmente, se deslogea y comprueba que le lleva a la página de login (aparece el texto “Identifícate”).

3.2 [LisUsrInVal] Intento de acceso con URL desde un usuario no identificado al listado de usuarios desde un usuario en sesión.

(Debe producirse un acceso no permitido a vistas privadas)

Se intenta ir directamente a la URL "/user/list" desde un usuario no logeado.

Se comprueba que se envía al usuario a la página de login (aparece el texto "Identifícate").

4.1 [BusUsrVal] Realizar una búsqueda valida en el listado de usuarios desde un usuario en sesión.

Va a la página de login, introduce datos válidos de un usuario con ROLE_PUBLIC y comprueba que inició sesión correctamente (como se indica en 2.1).

Se hace click en el campo de búsqueda, se borra su contenido y se introduce el texto "Mar". Se clicka el botón para realizar la búsqueda.

Se comprueba que aparezca sólo 2 usuarios, y que aparezcan los textos "Marta" y "María".

(El resto de usuarios no contienen la palabra "Mar" ni en su nombre ni en su email)

4.2 [BusUsrInVal] Intento de acceso con URL a la búsqueda de usuarios desde un usuario no identificado.

(Debe producirse un acceso no permitido a vistas privadas)

Se intenta ir directamente a la URL "/user/list?searchText=Mar" desde un usuario no logeado.

Se comprueba que se envía al usuario a la página de login (aparece el texto "Identifícate").

5.1 [InvVal] Enviar una invitación de amistad a un usuario de forma valida.

Primero nos logueamos con el usuario user1, después de loguearnos nos redirige automáticamente a la pagina del listado de usuarios, en ella buscamos el enlace para enviar una petición de amistad al usuario2, clickamos en el y nos desconectamos de sesión. A continuación, nos logueamos con el usuario2 y nos vamos a la vista que lista las invitaciones, comprobamos que la nueva invitación esté y nos deslogueamos.

5.2 [InvInVal] Enviar una invitación de amistad a un usuario al que ya le habíamos invitado la invitación previamente.

(No debería dejarnos enviar la invitación)

Nos logueamos con el usuario1 e intentamos enviar otra petición al usuario2 (Ya la hemos enviado previamente en la anterior prueba). Comprobamos que no hay un botón para enviar la invitación de amistad al usuario2.

6.1 [LisInvVal] Listar las invitaciones recibidas por un usuario, realizar la comprobación con una lista que al menos tenga una invitación recibida.

Nos logueamos con user1 y nos dirigimos a la vista que lista las invitaciones recibidas. Comprobamos que la vista contenga el nombre de Nombre7, el cual nos ha enviado una petición de amistad. Nos desconectamos de sesión.

7.1 [AcepInvVal] Aceptar una invitación recibida.

Nos logueamos, de nuevo, con user1, nos vamos a las invitaciones de amistad y buscamos la invitación de Nombre7 Apellido7, clickamos en el enlace. Nos redirige a la vista de amigos, por último, comprobamos que se haya agregado nuestro nuevo amigo antes de desconectarnos.

8.1 [ListAmiVal] Listar los amigos de un usuario, realizar la comprobación con una lista que al menos tenga un amigo.

Iniciamos sesión con user1, nos vamos al desplegable de usuarios, después clicamos en el enlace de la lista de usuarios y comprobamos que tenemos a marta como amiga de user1. Nos desconectamos.

9.1 [PubVal] Crear una publicación con datos válidos.

Va a la página de login, introduce datos válidos de un usuario con ROLE_PUBLIC y comprueba que inició sesión correctamente (como se indica en 2.1).

Hace click en el nav en la opción “Publicaciones” y espera a que se despliegue el dropdown-menu y aparezca la opción “Nueva Publicación”. Hace click en esa opción para ir a la página con el formulario para crear una publicación y comprueba que aparece el texto “Nueva publicación”.

Rellena y envía el formulario de creación del post con los siguientes datos:

- Título: "Título nueva publicación"
- Texto: "Texto nueva publicación"

Como nos envía directamente al listado de publicaciones del usuario, en el que aparecen todas las publicaciones (no hay paginación en este listado), comprobamos que aparece el texto "Título nueva publicación", correspondiente a la nueva publicación que acabamos de crear.

Finalmente, se deslogea y comprueba que le lleva a la página de login (aparece el texto “Identifícate”).

10.1 [LisPubVal] Acceso al listado de publicaciones desde un usuario en sesión.

(El usuario con el que entramos en sesión “user2” tiene 4 publicaciones)

Va a la página de login, introduce datos válidos de un usuario con ROLE_PUBLIC y comprueba que inició sesión correctamente (como se indica en 2.1).

Hace click en el nav en la opción “Publicaciones” y espera a que se despliegue el dropdown-menu y aparezca la opción “Ver mis Publicaciones”. Hace click en esa opción para ir a la página con el listado de todas sus publicaciones y comprueba que aparece el texto “Mis publicaciones”.

Se comprueba que aparezcan 4 publicaciones.

Finalmente, se deslogea y comprueba que le lleva a la página de login (aparece el texto “Identifícate”).

11.1 [LisPubAmiVal] Listar las publicaciones de un usuario amigo

Nos identificamos como user1 y nos vamos al desplegable de usuarios, clickamos en ver amigos y comprobamos que estamos en la vista de mis amigos.

Buscamos el enlace que nos lleva a los posts de nuestra amiga Marta y clickamos.

Comprobamos que aparece el texto “Publicaciones de Marta Almonte” y que tiene 4 posts.

Nos desconectamos.

11.2 [LisPubAmiInVal] Utilizando un acceso vía URL tratar de listar las publicaciones de un usuario que no sea amigo del usuario identificado en sesión.

Nos identificamos con user1 y navegamos a la url que lista los posts del usuario4 con id 4. Comprobamos que no nos haya llevado a la vista de publicaciones, sino que nos ha llevado a la página de error. Vamos a la página de inicio haciendo click en el enlace y luego nos desconectamos.

(Tal cual está hecha la prueba, el id del usuario a eliminar está prefijado, por lo que, si se realizan varias veces los tests sin reiniciar la aplicación principal, al borrarse primero todos los usuarios de la base de datos y luego volver a añadirlos en cada ejecución de los tests, los ids de dichos usuarios no comenzarán en 1, sino que comenzarán en el último id disponible.

Por tanto, hay veces que realmente no se estará intentando acceder a los posts de un usuario existente, sino uno que no existe, pero el resultado es el mismo, nos envía a la url de error)

12.1 [PubFot1Val] Crear una publicación con datos válidos y una foto adjunta.

Iniciamos sesión con user1 y pinchamos en el desplegable publicaciones, luego en nueva publicación. Rellenamos todos los campos válidamente. El elemento de envío tiene de nombre image, le mandamos el path de nuestra foto de muestra y mandamos el formulario. Comprobamos que ahora aparece una imagen en las publicaciones y nos desconectamos.

12.1 [PubFot2Val] Crear una publicación con datos válidos y sin una foto adjunta.

Nos identificamos con user1. Hacemos click en el nav en la opción “Publicaciones” y en la opción “Nueva Publicación”. Comprobamos que aparece el texto “Nueva publicación”.

Rellena y envía el formulario de creación del post con los siguientes datos:

- Título: "Título nueva publicación 3"
- Texto: "Texto nueva publicación 3"
- Imagen: Ninguna

Nos redirige a la vista con la lista de publicaciones y comprobamos que aparece el texto "Título nueva publicación 3", que es la publicación que acabamos de crear. Finalmente, nos desconectamos.

13.1 [AdInVal] Inicio de sesión como administrador con datos válidos.

Va a la página de login del administrador. Para ello, primero va a la página de login normal, y luego clicka en el enlace a "/admin/login".

(Se podría ir directamente a la URL "/admin/login", ya que no debería de ser un enlace accesible, pero hemos decidido hacerlo así para simplificar el acceso)

Rellena y envía el formulario con los datos de un usuario con ROLE_ADMIN.

Comprueba que se autologea correctamente (comprueba que va a una página en la que aparece el siguiente texto: "Usuario autenticado: <email>", donde <email> es el email del usuario que se acaba de autenticar).

Finalmente, se deslogea y comprueba que le lleva a la página de login (aparece el texto "Identifícate").

13.2 [AdInInVal] Inicio de sesión como administrador con datos inválidos (usar los datos de un usuario que no tenga perfil administrador).

Va a la página de login del administrador (como se indica en 13.1), y rellena el formulario con los datos de "user1", el cual no tiene ROLE_ADMIN.

Se comprueba que se muestra un mensaje de error debido a que dicho usuario no tiene ROLE_ADMIN (un texto con la clave "Error.admin.login.role" en el idioma Español).

14.1 [AdLisUsrVal] Desde un usuario identificado en sesión como administrador listar a todos los usuarios de la aplicación.

Va a la página de login del administrador, introduce datos válidos de un usuario con ROLE_ADMIN y comprueba que inició sesión correctamente (como se indica en 13.1).

Una vez autenticado como administrador, el proceso es el mismo que en el punto 3.1.

15.1 [AdBorUsrVal] Desde un usuario identificado en sesión como administrador eliminar un usuario existente en la aplicación.

Va a la página de login del administrador, introduce datos válidos de un usuario con ROLE_ADMIN y comprueba que inició sesión correctamente (como se indica en 13.1).

Clicka en el botón de eliminar que está en la misma fila que el email "user5@gmail.com". Se va a actualizar el fragmento que contiene la tabla con los usuarios y los enlaces de paginación.

Se comprueba que ya no aparece el texto "user5@gmail.com".

Se deslogea y comprueba que le lleva a la página de login (aparece el texto "Identifícate").

Ahora se inicia sesión como user1, que era amigo de user5.

Hace click en el nav en la opción “Usuarios” y espera a que se despliegue el dropdown-menu y aparezca la opción “Ver Amigos”. Hace click en esa opción para ir a la página con el listado de todos sus amigos y comprueba que aparece el texto “Tus Amigos”.

Se comprueba que no aparece el email “user5@gmail.com” en el listado de sus amigos.

Finalmente, se deslogea y comprueba que le lleva a la página de login (aparece el texto “Identifícate”).

15.2 [AdBorUsrInVal] Intento de acceso vía URL al borrado de un usuario existente en la aplicación.

(Debe utilizarse un usuario identificado en sesión pero que no tenga perfil de administrador)

Va a la página de login, introduce datos válidos de un usuario con ROLE_PUBLIC y comprueba que inició sesión correctamente (como se indica en 2.1).

Se intenta ir directamente a la URL “/user/delete/2”.

Se comprueba que se envía al usuario a la página “/error” (aparece el texto “¡Se ha producido un error!”).

Se hace click en el enlace que aparece debajo del texto anterior, para ir a la página de inicio y se comprueba que aparece el texto “¡Bienvenidos a Red Social!”.

Finalmente, se deslogea y comprueba que le lleva a la página de login (aparece el texto “Identifícate”).

(Tal cual está hecha la prueba, el id del usuario a eliminar está prefijado, por lo que, si se realizan varias veces los tests sin reiniciar la aplicación principal, al borrarse primero todos los usuarios de la base de datos y luego volver a añadirlos en cada ejecución de los tests, los ids de dichos usuarios no comenzarán en 1, sino que comenzarán en el último id disponible.

Por tanto, hay veces que realmente no se estará intentando borrar un usuario existente, sino uno que no existe, pero el resultado es el mismo, ya que como no tienes permiso para acceder a la URL “/user/delete/*”, da igual si el usuario existe o no, que no te permite realizar dicho borrado.)