SDI - RED SOCIAL

24/03/2018

Documentación relativa a la implementación

Marcos Ruíz Villamea – UO196582

Raúl Gómez Pérez – UO241016

SDI - RED SOCIAL

DOCUMENTACIÓN RELATIVA A LA IMPLEMENTACIÓN

CONTENIDO

INTRODUCCIÓN

Este trabajo constaba de la implementación de una red social sencilla en la cual podemos gestionar amistades entre usuarios y tendrán la posibilidad de chatear entre ellos.

Para llevar a cabo la misma se debían utilizar las siguientes tecnologías:

Node.js para el desarrollo de la aplicación web.

API Rest con funcionalidad similar.

Cliente web utilizando jQuery.

Base de datos MongoDB para la persistencia de la aplicación.

Para el desarrollo de esta red social se han seguido las prácticas de la asignatura para la mayoría de los casos de uso a implementar y para determinadas funcionalidades o ampliaciones se ha investigado a través de la web.

CASOS DE USO

1 Público: Registrarse como usuario

En este caso de uso la aplicación debía permitir el registro de nuevos usuarios en el sistema controlando que un mismo email no pudiese ser utilizado para dos cuentas de usuario distintas.

Utilizando la vista “signup.html” y utilizando una petición POST a “/signup” desde el controlador de usuarios “rusers.js” introducimos un nombre, email, password y confirmación de password en el formulario, y completamos el registro. En caso de que el email ya haya sido introducido se devolverá al usuario a la vista de registro con un mensaje de error indicando que no ha sido posible completar el registro.

Una vez validado el usuario se crea, se almacena en base de datos y se accede a la vista de identificación.

2 Público: Iniciar sesión

En este caso de uso los usuarios registrados en la plataforma deben poder de iniciar sesión facilitando su email y contraseña a través de un formulario de login.

Se añadió un botón en la barra de navegación para mostrar el formulario de login de la vista “/signin.html”.

Utilizando la vista “signin.html” y utilizando una petición POST a “signin” desde el controlador de usuarios “rusers.js” introducimos email y password en el formulario, y completamos el login. En caso de que el login no se complete con éxito se devolverá al usuario a la pantalla de login informando del problema ocurrido. En caso de éxito, se redirigirá al usuario a la vista de los usuarios de la aplicación.

3 Usuario registrado: Listar todos los usuarios de la aplicación

Este caso de uso permite al usuario logeado en el sistema listar los usuarios del mismo mostrando su nombre y su email y paginando el resultado mostrando 5 usuarios por página, además de un botón para solicitar amistad a cada usuario listado.

Desde el controlador "rusers.js" y realizando una petición GET a "/user/list" se mostrarán en la vista "users.html" el listado de usuarios devuelto desde la base de datos.

4 Usuario registrado: Buscar entre todos los usuarios de la aplicación

En este caso de uso el usuario en sesión debe poder buscar entre los usuarios del sistema por nombre y email tanto completos como que contengan parte de la cadena buscada. El resultado de esta búsqueda será una lista de usuarios que en caso de ser superiores a cinco deberán ser mostrados paginados de cinco en cinco.

Para su implementación se ha añadido a la petición GET "/user/list" una nueva funcionalidad que en el momento en que se encuentra texto dentro del campo de búsqueda en la parte superior de la pantalla se modifica el criterio de búsqueda a la base de datos para que solo aparezcan los usuarios que cumplan ese criterio.

Cuando el usuario busca por el email, el nombre o una cadena de caracteres el sistema muestra un listado con los usuarios cuyos emails o nombres coinciden con lo buscado.

5 Usuario registrado: Enviar una invitación de amistad a un usuario

En este caso de uso el usuario deberá poder realizar una petición de amistad a un usuario del sistema mostrado en la vista de listado de usuarios.

Para su realización se ha utilizado la misma vista "users.html" a la que se realiza una petición GET a la url "/friendrequest/:id" desde el controlador "rrequests.js" donde se recoge el id del usuario que se desea añadir como amigo en la url, y posteriormente se crea una entrada en la base de datos referente a esa petición.

Solamente será posible realizar la petición en caso de que no exista una petición anterior a ese usuario, ni exista una amistad previa entre ellos. En ese caso se mostrará un mensaje de error informando al usuario.

6 Usuario registrado: Listar las invitaciones de amistad recibidas

En este caso de uso el usuario podrá visualizar las peticiones de amistad recibidas por otros usuarios a través de una opción de menú en la barra de navegación que dará acceso a la vista que muestra el listado de peticiones de amistad.

Utilizando la vista "requests.html" junto al controlador "rrequests.js" mandando una petición a GET "/requests" obtenemos las peticiones de la base de datos enviadas al usuario en sesión.

Se mostrará un listado con el nombre del usuario que ha enviado la petición y un botón para aceptar la petición de amistad.

7 Usuario registrado: Aceptar una invitación recibida

En este caso de uso se permitirá al usuario aceptar una invitación de amistad recibida por parte de otro usuario a través de un botón “aceptar”. En el momento en que el usuario acepte la petición, ésta debe desaparecer del listado.

Se han utilizado la vista “requests.html” y una petición a GET “/requests/accept/:id” en el controlador “rrequests.js” para aceptar la petición desde el listado de peticiones. En el momento en que se acepta la petición, se elimina de la base de datos, y se creará un nuevo documento contemplando la amistad entre los dos usuarios (de hecho, se crearán dos, para que la funcionalidad del resto de la aplicación sea más fácil de implementar).

Tanto en el caso de aceptar la petición de manera correcta como incorrectamente, se mostrará un mensaje al usuario informando de la acción.

8 Usuario registrado: Listar los usuarios amigos

En este caso de uso se permitirá visualizar en forma de lista todos los usuarios amigos del usuario en sesión, mostrando para cada uno el nombre y su email. El listado deberá estar paginado mostrando cinco amigos por página y se deberá acceder a él mediante una opción de menú en la barra de navegación.

Para su implementación se han utilizado la vista “friends.html” y una petición GET “/friends” en el controlador “rusers.js” donde se obtendrán los usuarios que cumplan el criterio de amistad con el usuario en sesión y se mostrarán en un listado.

S.1 Identificarse con usuario - token

En este caso se debe permitir el acceso a la API Rest mediante la utilización de un token con una caducidad limitada, el cual se utilizará para autenticar al usuario en las próximas peticiones a la API.

Para su implementación se ha utilizado un controlador “rapiusers.js” en el que existe un método POST a “/api/signin” en el cual se obtiene el email y contraseña de un objeto json enviado en el body de la petición. Posteriormente se encripta el password y se comprueba que existe un usuario en la base de datos con ese email y contraseña encriptada. En caso de que la autenticación sea correcta se devuelve un token, y en caso negativo, se enviará un mensaje de error informando al usuario.

S.2 Usuario identificado: listar todos los amigos

El servicio permite a un usuario obtener su lista de amigos. Para que esta petición funcione necesita recibir un token válido del usuario.

El servicio se ha implementado sobre una petición GET a la url “/api/users” en el cual es necesario recibir un token en la cabecera de la petición, del cual se desencriptará el email del usuario para obtener las amistades del usuario en sesión persistidos.

Finalmente se mostrará la lista de los amigos del usuario propietario del token.

S.3 Usuario identificado: Crear un mensaje

El servicio permite a un usuario crear un nuevo mensaje para uno de sus amigos. Para que esta petición funcione se necesita recibir un token válido del usuario.

En primer lugar, se obtienen los usuarios, se filtran dichos usuarios para obtener los amigos del usuario que solicita la información, se selecciona el amigo en concreto y se añade un mensaje entre ambos.

Como resultado el mensaje queda añadido al sistema.

S.4 Usuario identificado: Obtener mis mensajes de una "conversación"

El servicio permite a un usuario obtener la lista de los mensajes que tiene con otro usuario que pertenece a sus amigos. Para que esta petición funcione es necesario recibir un token válido por parte del usuario.

El sistema recibe mediante petición "get" el id del usuario amigo del que quiere obtener la lista de mensajes. Posteriormente se obtiene el usuario solicitante de la decodificación del token de la petición. A continuación, mediante esos dos usuarios se buscarán los mensajes que tienen en común.

Como resultado se obtendrá el listado de los mensajes entre ambos usuarios en formato JSON.

C.1. Autenticación del usuario

En este caso se permitirá al usuario acceder a la aplicación mediante el cliente jQuery. Para ello el usuario introducirá el email y la contraseña correctos en el formulario para acceder a la zona privada.

Para la realización de este caso de uso se ha hecho uso: en primer lugar, de la API Rest mediante la petición POST "/api/signin" al controlador "rapiusers.js", enviando un json con el email y la contraseña obtenidos del formulario en "widget-login.html". En caso de una autenticación correcta se accederá a la zona privada de la web añadiendo un nuevo token que permitirá al usuario navegar a través de ella. En caso contrario, se devolverá al usuario a la pantalla de login notificando que ha habido un error.

C.2. Mostrar la lista de amigos

Una vez logeado en el cliente el usuario accederá directamente a su lista de amigos. En él aparecerá un listado con ellos y además, se permitirá el filtro por nombre de estos.

Se ha implementado utilizando la vista "widget-friends.html" y una petición GET "/api/users" al controlador "rapiusers.js" de la API Rest. Se obtiene el token del usuario actual, se comprueban las amistades que tiene y se obtienen los usuarios amigos que se mostrarán en un listado.

Para la realización del filtro se ha añadido un input en la parte superior de la página y mediante un par de funciones se filtra a los usuarios amigos que cumplan en su nombre ese criterio.

C.3. Mostrar los mensajes

El sistema desde la lista de amigos del caso de uso anterior ofrece un botón mediante el cual se accede a la ventana de chat con el texto “Chatear con <nombre del amigo>”. Al pulsarlo se abrirá una ventana con un chat mediante el cual el usuario podrá leer los mensajes recibidos y enviados entre ambos usuarios.

Estos mensajes se solicitan a la api rest al cargar la página y se muestran en un div contenedor de mensajes con id "chat-container", visualizándose en tiempo real mediante un “setInterval()” de dos segundos.

Los mensajes recibidos del otro usuario aparecerán contenidos individualmente en un div, alineados a la izquierda e identificados con el nombre del usuario que lo envió. Por otro lado, los mensajes propios aparecerán también contenidos individualmente en un div y alineados a la derecha y con otro color de fondo.

C.4. Crear mensaje

En la misma página que el anterior caso de uso e inmediatamente debajo de la lista de mensajes se encontrará un input de texto y un botón “Enviar”. Cuando el botón enviar se pulsa, se realiza una petición a la api rest añadiendo el mensaje que contenía el campo de texto y automáticamente se vacía para que se puedan seguir mandando mensajes.

C.5. Marcar mensajes como leídos de forma automática

Cuando un usuario accede a la lista de mensajes de chat con otro usuario y visualiza los mensajes que estan marcados como no leídos con “read: false”, se ejecuta la función “setMessageRead(idMessage)” que marca como leído el mensaje.

Esta acción se lleva a cabo realizando una petición “put” a la ruta “/api/message/:id“ de la api rest con el id del mensaje y con ésta se actualizar el campo read del mensaje a true.

Cuando el chat realiza la acción de actualizar los mensajes y recarga la lista de mensajes, se les añade un tic verde a los mensajes enviados por el usuario y que han sido leídos por su amigo.

C.6 Mostrar el número de mensajes sin leer

Cuando el usuario accede al cliente web y accede a la vista de usuarios, en esta se indican también los mensajes sin leer que tiene con ese usuario. Esto se realiza con una petición a la api rest del servidor, que trae los mensajes y el número de mensajes no leídos y los añade a la tabla por cada usuario…

PRUEBAS DE SELENIUM

Se ha creado una llamada GET a “/erasedatatest” donde en cada ejecución completa

1.1 [RegVal] Registro de Usuario con datos válidos.

Se accede al formulario de registro, se rellena el formulario y se comprueba que muestra el mensaje de éxito.

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

Se accede al formulario de registro, se rellena el formulario con la contraseña repetida distinta y se comprueba que muestre el mensaje de error.

2.1 [InVal] Inicio de sesión con datos válidos.

Se accede al formulario de logueo, se rellena el formulario con un mail y contraseña válida, y se comprueba que se accede a la zona privada.

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

Se accede al formulario de logueo, se rellena el formulario con un email no válido y se comprueba que aparece el mensaje de error.

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

Se accede al formulario de logueo, se introducen credenciales válidas y se comprueba que directamente se accede a la zona privada de la aplicación donde se listan los usuarios.

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 acceder a la lista de usuarios sin estar logueado a través de la URL y se comprueba que, efectivamente, no es posible debido a la seguridad implementada.

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

Se loguea con credenciales válidas, se dirige a la vista de listado de usuarios y realiza una búsqueda válida. Se comprueba que aparece la información necesaria.

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 acceder a la lista de usuarios sin estar logueado a través de la URL utilizando la querystring, y se comprueba que, efectivamente, no es posible debido a la seguridad implementada.

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

Se loguea con datos válidos, accede a la lista los usuarios con un email y contraseña correctos, y se envía una petición de amistad a un usuario de la aplicación. Se comprueba que aparece el mensaje de éxito.

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, se podría ocultar el botón de enviar invitación o notificar que ya había sido enviada previamente.

Se loguea con datos válidos, accede a la lista los usuarios con un email y contraseña correctos, se envía una petición de amistad a un usuario de la aplicación al cual ya se le había enviado una anteriormente y se comprueba que aparece el mensaje de error.

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

Se loguea con datos válidos, accede a la lista de peticiones, y comprobamos que existe al menos una petición dentro de la lista que aparece en la página.

7.1 [AcepInvVal] Aceptar una invitación recibida.

Se loguea con datos válidos, accede a la lista de peticiones y se acepta la invitación haciendo click en el botón aceptar.

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

Se loguea con datos válidos, accede a la lista de amigos y comprueba que tiene al menos un amigo llamado Prueba3.

C1.1[[CInVal] Inicio de sesión con datos válidos.

Se loguea con datos válidos en el cliente JQuery, rellenando el formulario con datos válidos. Por último, se comprueba que se ha accedido satisfactoriamente.

C1.2 [CInInVal] Inicio de sesión con datos inválidos (usuario no existente en la aplicación).

Se intenta loguear con datos inválidos y comprueba que se muestra el mensaje de error “Usuario no encontrado”.

C.2.1 [CListAmiVal] Acceder a la lista de amigos de un usuario, que al menos tenga tres amigos.

Para este caso de prueba es necesario primero enviar peticiones de amistad desde el usuario “Yeyas” hacia el resto de los usuarios creados en las pruebas anteriores y aceptarlas por parte de éstos.

Posteriormente se loguea el usuario “Yeyas” y los lista, comprobando que el número de usuarios listados es 3.

C.2.2 [CListAmiFil] Acceder a la lista de amigos de un usuario, y realizar un filtrado para encontrar a un amigo concreto, el nombre a buscar debe coincidir con el de un amigo.

Se loguea el usuario “Yeyas” con datos válidos y accede al listado de amigos, posteriormente se posiciona en el apartado de búsqueda y busca al usuario con nombre Prueba3. Por último, comprueba que el usuario que contiene la cadena buscada está en la página.

C3.1 [CListMenVal] Acceder a la lista de mensajes de un amigo “chat”, la lista debe contener al menos tres mensajes.

Accede al cliente web JQuery y se loguea el usuario “Prueba4” con datos válidos, accede al chat con el usuario “Yeyas” y escribe tres mensajes. Después se loguea el usuario Yeyas con datos válidos y accede al chat con el usuario “Prueba4”. Por último, se comprueba que hay tres mensajes en la lista de mensajes.

C4.1 [CCrearMenVal] Acceder a la lista de mensajes de un amigo “chat” y crear un nuevo mensaje, validar que el mensaje aparece en la lista de mensajes

Se accede al cliente web JQuery y se loguea con el usuario “Prueba3” y se accede al chat con el usuario “Yeyas”, se coloca en el input de mensajes y envía el mensaje “Hola”. Por último, se comprueba que el mensaje aparece en el listado de mensajes.

RESULTADO PRUEBAS