SDI – Sistemas Distribuidos e Internet

ENUNCIADO PRÁCTICA 2 – NODE.Js & Servicios Web

INFORME

Grupo 914-909

Nombre1:	Nerea
Apellidos1:	Valdés Egocheaga
Email1:	UO259000@uniovi.es
Cód. ID GIT	914
% Participación	50
Nombre1:	Miguel
Apellidos1:	Guimarey Lesmes
Email1:	UO247134@uniovi.es
Cód. ID GIT	909
% Participación	50

Índice

Introducción

El trabajo se ha divido en 2 partes, una dedicada a una aplicación web y otro a servicios web.

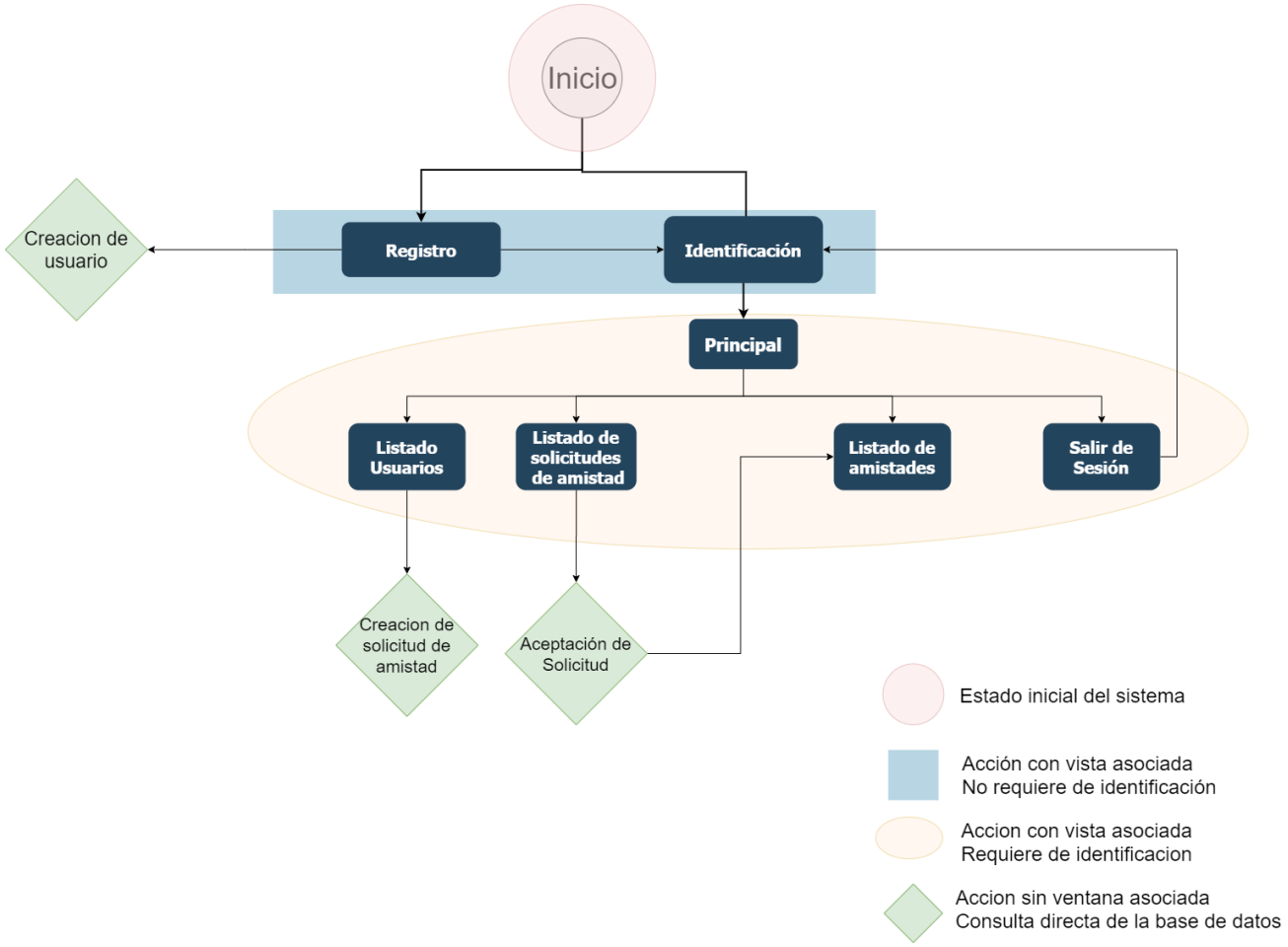
El objetivo de esta aplicación ha sido el desarrollar un sistema capaz de tener un registro de usuarios que puedan ser registrados manualmente e identificarse a continuación.

Un usuario además tiene la capacidad de ver otros usuarios del sistema y poder enviar una solicitud de amistad, que puede ser aceptada por el usuario receptor para confirmar la amistad entre ambos. Ambas funcionalidades van acompañadas de un sistema de listado de solicitudes y otra de amistades.

En cuanto a seguridad se han implementado un sistema por token de sesión, por lo que para acceder a contenido personal (enviar solicitudes, aceptar amistades…) se requiere estar identificado en ese momento y solo es accesible las funcionalidades para ese usuario en concreto.

Mapa de navegación

1. Aplicación Web



La navegación de esta parte es muy sencilla:

Se cuenta con inicialmente con solo dos opciones, registro e identificación, siendo identificación la ventana por defecto al iniciar la aplicación.

Desde registro podemos introducir nuevos usuarios al sistema siempre que complete el formulario correctamente.

Desde identificación se guarda la sesión de usuario y desde ahí se nos abren nuevas funciones:

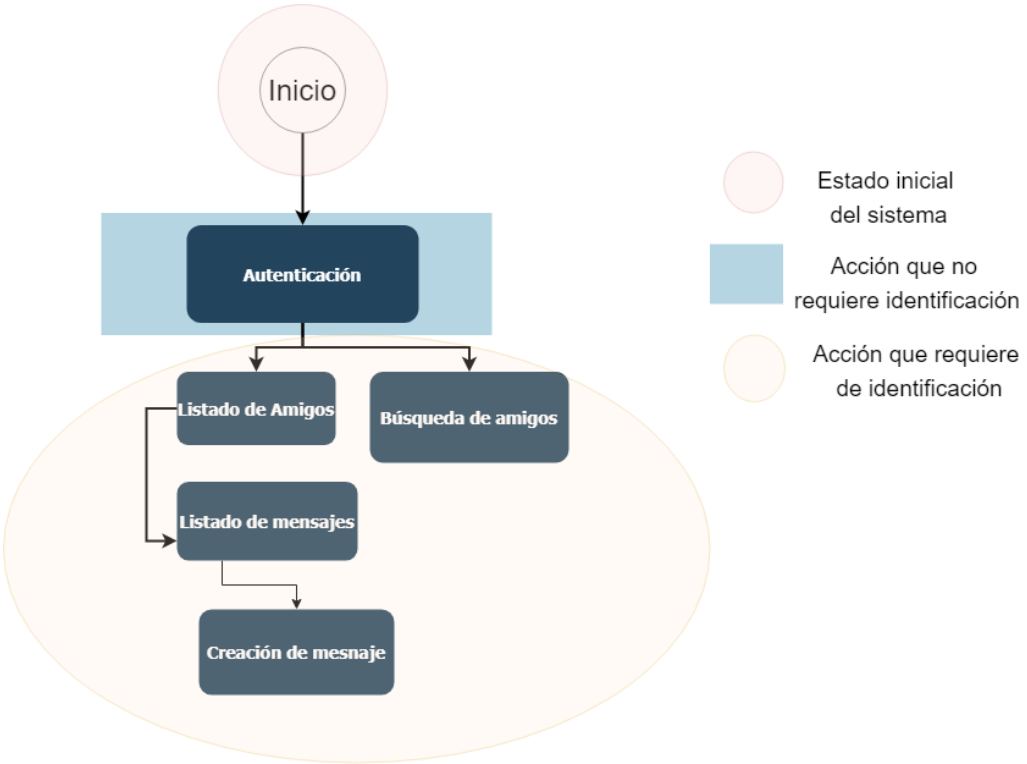
- Listar usuarios del sistema (el usuario identificado también aparecerá), esta es la vista por defecto de estar en sesión, tiene acceso a la creación de solicitudes de amistad de el usuario identificado al usuario objetivo.
- Listar peticiones de amistad, desde esta opción se puede cambiar las peticiones de amistad presentes a estado aceptado y esto nos redirigirá a la siguiente ventana.
- Listado de amistades, donde se visualizan todos usuarios de peticiones aceptadas que involucren al usuario identificado.
- Salir de sesión, esta opción es la que nos devuelve al estado de identificarse inicial.

2. Servicios Web

Por este lado, la navegación consta de lo siguiente:

Al entrar a la página web, la única acción posible es la autenticación o inicio de sesión de un usuario ya existente. A continuación, una vez iniciada la sesión, se dirige directamente a la vista de Amigos, donde se puede acceder a las conversaciones de cada uno de los amigos del usuario en sesión.

En la vista de la conversación se listan los mensajes ordenados entre el usuario en sesión y su amigo y se muestra de quién es cada mensaje. Desde esta vista, se puede rellenar el formulario de creación de mensaje para mandar un mensaje a ese amigo de la conversación en la que estamos. Automáticamente, aparecerá el mensaje recién creado en la conversación en la que estamos, junto con los demás.



Aspectos técnicos y de diseño relevantes

Toda la aplicación cuenta con un archivo log (logs/redsocial.log) que registra cada acción realizada por el usuario, todo evento fallido o exitoso tendrá su respuesta programada, su anotación en el log y en ocasiones un mensaje que es visible en la propia aplicación para informar al usuario del último evento.

1. Base de datos

La base de datos será Mongo y estará en la nube mediante Mongo Cloud.

En ella se implementarán 3 colecciones diferentes:

- Usuarios: Que mantendrá registro de todas las cuentas, cada cuenta a su vez tendrá los atributos:
  - **_id:** Que identificará la identidad de ese objeto concreto.
  - **Email:** Email de la cuenta, este será único y será aquello que sirva de unión con otras colecciones.
  - **Surname:** Apellido del usuario, se trata de un String sin limitación alguna.
  - **Password:** Una contraseña que será almacenada de manera cifrada con el protocolo sha256.
- Peticiones: Colección que guardará todas las peticiones de amistad que haga un usuario a otro y también las amistades persé. Cuenta con los atributos:
  - **_id:** Que identificará la identidad de ese objeto concreto.
  - **Emisor:** El email de quien envió la petición en primer lugar, tipo String.
  - **Remitente:** El email de la persona a quien iba dirigida la petición en primer lugar, tipo String.
  - **Aceptada:** Campo booleano, false de manera predefinida, cuando se vuelve true se empieza a comportar como una amistad y no como una solicitud.
- Mensaje:
  - **_id:** Identificador del objeto concreto mensaje.
  - **Emisor:** El email del usuario que envía el mensaje, de tipo String.
  - **Destino:** El email de quien recibe el mensaje, de tipo String.
  - **Texto:** El texto del mensaje, de tipo String.
  - **Fecha:** Se guarda la fecha de creación del mensaje, de tipo Date.
  - **Leído:** Es un valor booleano que por defecto se establece a false cuando se crea un mensaje.

2. Aplicación web

Se desarrolla una aplicación web red social. Para acceder a su funcionalidad tendrá que estar el usuario identificado en el sistema con una cuenta que exista en la base de datos.

Se han trabajado con dos javascript en este apartado:

- rusuarios.js: Encargado de las acciones que recibirá la aplicación a través de url y como trabajará con la información recibida (ya sea de base de datos o de la aplicación web).
- gestorDB.js: Encargado de ser el intermediario entre rusaurios y la base de datos, será el encargado de realizar las consultas pertinentes solicitadas por rusuarios.js.

En cuanto al apartado grafico se cuenta con:

- 2 htmls de propósito general:
    - Base.html: Una plantilla básica para al resto de htmls, les proporciona una barra de navegación superior con las funcionalidades habilitadas para cada momento junto a un script con el objetivo de mostrar mensajes al usuario de posibles eventos.
    - Error.html: Una plantilla para mostrar mensajes error en caso de que se acceda a un contenido que no exista o no se pueda facilitar.
- 5 htmls para los servicios del sistema (véase apartado Mapas de navegación.1).

Decisiones de diseño

En este apartado se especifica qué criterios se han seguido para solucionar apartados técnicos del sistema:

Una sola entidad que referencie tanto amistades como solicitudes de amistad

Para mantener constancia en todo momento que una petición ha sido creada, aunque ya haya sido aceptada, tendrá un apartado booleano que marcará si se encuentra aceptada o no, con lo que no hace necesario borrarla del sistema una vez ya cumplió su cometido.

Tampoco se han creado entidades amigos, ya que para extraer las amistades solo es necesario ver que usuarios aparecen con la solicitud a true.

La solicitud solo puede ser aceptada por aquel a quien va dirigido dicha solicitud con el fin de mantener la integridad y la seguridad.

Emails como identificadores de usuarios

Como son propiedades únicas de cada usuario hemos decidido que sea la información que se guarde en otras entidades (como solicitudes de amistad o mensajes).

Esto provoca que en aquellas páginas que se requiera un listado completo del usuario, pero trabajen con solicitudes de amistad (listado de peticiones de amistad y amigos), se requiera de una consulta de la base de datos de usuario para que devuelva todos los usuarios que tengan el email de uno de los integrantes de la solicitud, pero no el del usuario de sesión.

Se comprueba con cada petición de rusuarios el usuario en sesión

Por motivos de seguridad, en todo método que requiera de que solo un usuario concreto la pueda hacer se hará una comprobación de que el usuario en sesión sea el buscado, de no ser así se desviara a la ventana de identificación y se mostrará el mensaje de error oportuno junto a la constancia en el log.

3. Servicios web

Se implementará la API de servicios web REST en nuestro mismo proyecto de NodeJS con WebStorm siguiendo la arquitectura RESTful. En nuestro caso, en la carpeta "routes" será el "rapiredsocial.js".

Además, se implementa una aplicación jQuery-AJAX que consume estos servicios, esta aplicación permite el intercambio de mensajes en tiempo real. Esta aplicación se localiza en la carpeta "public" donde se definen los siguientes HTML:

- Cliente.html

- Widget-login.html
- Widget-amigos.html
- Widget-conversacion.html

Información necesaria para el despliegue y ejecución

Base de datos CloudDB

Creación de una base de datos no relacional en MongoDB Cloud https://account.mongodb.com/account/login. Hay que logearse a una cuenta.

Clusters, Create a Cluster, Proveedor AWS en region Ireland. Le ponemos un nombre como "redsocial" y le damos a crear.

Accedemos a Security>Database Access, Add a new user, para crear un usuario con privilegios como usuario "administrador" y contraseña "redsocial914909". Seleccionamos en User Privileges>Alias admin y le damos a Add User.

Vamos a Security>Network Access>IP Whitelist y damos a Add IP Address. Seleccionamos Allow Access From Anywhere, con lo demás por defecto y clicamos a Confirm.

Para obtener la URL de conexión vamos a Clusters>Connect a "redsocial", seleccionamos Choose a connection method, Connecct your Application, seleccionamos como driver "Node.js" con versión "2.2.12 or later" y Connection String Only donde veremos la URL.

La copiamos, le damos a close y vamos a nuestra aplicación web en redsocial.js y establecemos la variable con el nombre de la bd, el usuario y contraseña que hemos creado.

Ejecución Proyecto

Desde WebStorm, abrimos el proyecto que contiene tanto la aplicación web como los servicios web con la aplicación Jquery. Para ponerlo en marcha sólo será necesario especificar la URL de conexión a nuestra BD en las variables de nuestra app.js.

Para acceder a nuestra aplicación web a través del navegador utilizamos la URL https://localhost:8081.

Para acceder a nuestra aplicación Jquery a través del navegador utilizamos la URL https://localhost:8081/cliente.html.

Ejecución Selenium Tests

Para la ejecución de las pruebas en nuestro proyecto STS se debe disponer de Mozilla Firefox instalado, y el ejecutable "geckodriver024win64.exe". Al abrir el proyecto en Spring Tool Suite, se deben establecer los valores estáticos de PathFirefox65 y Geckdriver024 de las rutas concretas a dichos ejecutables en nuestra máquina.

La clase PO_InitAplication.java nos permite definir métodos auxiliares para el borrado y la inicialización de la BD al comenzar a ejecutar las pruebas.

Para ejecutar las pruebas, clic derecho sobre UOChatTests.java > Run as > Junit Test.

Conclusión

Entre los desafíos que nos ha planteado esta entrega, nos hemos visto con:

Recaída en el mismo fallo que en la entrega pasada, donde al realizar una búsqueda en una página con mas de una página, si queríamos pasar a otra página perdíamos la búsqueda previa. Para solucionarlo hemos optado por en caso de que exista alguna búsqueda, devuelva el resultado en una sola página, y en caso de introducirse una búsqueda vacía o que no exista, devolver el resultado paginado.

No obstante, esta vez hemos desarrollado rápidamente una forma eficaz para el modelo de datos para nuestra BD en MongoDB. Esto es, con el menor número posible de colecciones para minimizar el número de consultas a la misma.

Como conclusión final, estamos contentos con el trabajo desarrollado.