PRÁCTICA 2 DE ENTREGA – NodeJS y Servicios Web

MY SOCIALNETWORK

Instrucciones generales

La práctica consta de dos partes más la documentación y el peso de cada parte es el siguiente:

- Parte 1 "aplicación web" 4/11 puntos.
- Parte 2 "servicios web" 6/11 puntos.
- Documentación 1/11 punto.

Para que el trabajo pueda ser evaluado los **Requisitos obligatorios** deben estar implementados, con sus correspondientes pruebas automatizadas ***(no se evaluarán los casos de uso que no contengan pruebas).*** La puntuación máxima a la que se opta por implementar todos los requisito obligatorios de forma perfecta es de 7.0 sobre 11, mientras que los requisitos opcionales implementados de forma perfecta llegarán hasta 3.0 puntos sobre 11 (ver Tabla 1).

Tabla 1. Detalle de la puntuación de la práctica

			Puntos
PARTE1 - APLICACIÓN WEB			**4,00**
	REQUISITOS OBLIGATORIOS		**4,00**
	W1	Perfil Público: registrarse como usuario	0,50
	W2	Perfil Público: iniciar sesión	0,25
	W3	Usuario Registrado: Fin de sesión	0,25
	W4	Usuario registrado: listar todos los usuarios de la aplicación	0,25
	W5	Usuario registrado: buscar entre todos los usuarios de la aplicación	0,50
	W6	Usuario registrado: enviar una invitación de amistad a un usuario	0,25
	W7	Usuario registrado: listar las invitaciones de amistad recibidas	0,25
	W8	Usuario registrado: aceptar una invitación recibida	0,25
	W9	Usuario registrado: listar los usuarios amigos	0,50
	W10	Seguridad	1,00
PARTE2 - SERVICIOS WEB			**6,00**
	PARTE2A - REQUISITOS OBLIGATORIOS		**1,50**
	S1	Identificarse con usuario – token	0,50
	S2	Usuario identificado: listar todos los amigos	0,25
	S3	Usuario identificado: Crear un mensaje	0,25
	S4	Usuario identificado: Obtener mis mensajes de una "conversación"	0,50
	PARTE2A - REQUISITOS OPCIONALES		**0,50**
	*S5	Usuario identificado: Marcar mensaje como leído	0,50
	PARTE2B - REQUISITOS OBLIGATORIOS		**1,50**
	C1	Autenticación del usuario	0,50
	C2	Mostrar la lista de amigos	0,25
	C3	Mostrar los mensajes	0,25
	C4	Crear mensaje	0,50
	PARTE2B - REQUISITOS OPCIONALES		**2,50**
	*C5	Marcar mensajes como leídos de forma automática	1,00
	*C6	Mostrar el número de mensajes sin leer	0,50
	*C7	Ordenar la lista de amigos por último mensaje	1,00
INFORME OBLIGATORIO			**1,00**
TOTAL			**11,00**

RESUMEN	
REQUISITOS OBLIGATORIOS	**7,00**
REQUISITOS OPCIONALES	**3,00**
INFORME OBLIGATORIO	**1,00**
TOTAL	**11,00**

Notas:

- *Para los alumnos que trabajen en grupos se tendrá en cuenta el porcentaje de participación de cada miembro del grupo y su grado de implicación durante el desarrollo del proyecto.*
- *Es requisito indispensable utilizar una base de datos Mongo en la nube (Mongo Cloud).*
- *Esta práctica se podrá realizar en equipos de hasta 2 alumnos (pertenecientes al mismo grupo de prácticas) y su entrega es obligatoria.*

Parte1 “Aplicación Web”

Se trata de desarrollar una aplicación Web de tipo red social en el que existirán perfiles de usuario de tipo: Público (Anónimo) y Usuario Registrado (Administrador y Usuario Estándar). Todos los requisitos son obligatorios. A continuación, se detallan los requisitos:

W1 Público: Registrarse como usuario

Los usuarios deben poder registrarse en la aplicación aportando un email, nombre, apellidos y una contraseña (que deberá repetirse dos veces y coincidir entre sí).

El email del usuario no podrá estar repetido en el sistema, se debe informar al usuario de los errores en el proceso de registro.

Pruebas Funcionales

[Prueba1] Registro de Usuario con datos válidos.

[Prueba2] Registro de Usuario con datos inválidos (email vacío, nombre vacío, apellidos vacíos).

[Prueba3] Registro de Usuario con datos inválidos (repetición de contraseña inválida).

[Prueba4] Registro de Usuario con datos inválidos (email existente).

W2 Público: Iniciar sesión

Suministrando su email y contraseña, un usuario podrá autenticarse ante el sistema. Sólo los usuarios que proporcionen correctamente su email y su contraseña podrán iniciar sesión con éxito.

En caso de que el inicio de sesión fracase, será necesario mostrar un mensaje de error indicando el problema.

En caso de que el inicio de sesión sea correcto se debe dirigir al usuario a la vista que “lista todos los usuarios de la aplicación”

Pruebas Funcionales

[Prueba5] Inicio de sesión con datos válidos (usuario estándar).

[Prueba6] Inicio de sesión con datos inválidos (usuario estándar, campo email y contraseña vacíos).

[Prueba7] Inicio de sesión con datos inválidos (usuario estándar, email existente, pero contraseña incorrecta).

[Prueba8] Inicio de sesión con datos inválidos (usuario estándar, email no existente y contraseña no vacía).

W3 Usuario Registrado: Fin de sesión

Disponer una opción de menú que permita finalizar la sesión enviando al usuario al formulario de Inicio de sesión. Este botón solo se mostrará si un usuario se ha autenticado previamente.

Pruebas Funcionales

[Prueba9] Hacer click en la opción de salir de sesión y comprobar que se redirige a la página de inicio de sesión (Login).

[Prueba10] Comprobar que el botón cerrar sesión no está visible si el usuario no está autenticado.

W4 Usuario registrado: Listar todos los usuarios de la aplicación

Se visualizará en una lista todos los usuarios de la aplicación. Para cada usuario se mostrará su nombre, apellidos y email.

La lista debe incluir un sistema de paginación y mostrar 5 usuarios por página.

Debe incluirse una opción de menú principal, visible solo para usuarios en sesión que permita acceder al listado de todos los usuarios de la aplicación.

Pruebas Funcionales

[Prueba11] Mostrar el listado de usuarios y comprobar que se muestran todos los que existen en el sistema.

W5 Usuario registrado: Buscar entre todos los usuarios de la aplicación

Incluir un sistema que permita realizar una búsqueda por nombre, apellidos y email de usuario. El cuadro de búsqueda contendrá un único campo de texto, la cadena introducida en ese campo utilizará para buscar coincidencias tanto en el campo nombre y apellidos como en el email de los usuarios. Por ejemplo, si escribimos la cadena "mar" deberá retornar usuarios en los que la cadena "mar" sea parte de su nombre, apellidos o su email.

El resultado de la búsqueda debe ser una lista que muestre las coincidencias encontradas, está lista debe incluir un sistema de paginación y mostrar 5 usuarios por página.

Pruebas Funcionales

[Prueba12] Hacer una búsqueda con el campo vacío y comprobar que se muestra la página que corresponde con el listado usuarios existentes en el sistema.

[Prueba13] Hacer una búsqueda escribiendo en el campo un texto que no exista y comprobar que se muestra la página que corresponde, con la lista de usuarios vacía.

[Prueba14] Hacer una búsqueda con un texto específico y comprobar que se muestra la página que corresponde, con la lista de usuarios en los que el texto especificados sea parte de su nombre, apellidos o de su email.

W6 Usuario registrado: Enviar una invitación de amistad a un usuario

Junto a la información de cada usuario presente en la vista de "listar todos los usuarios de la aplicación" debe aparecer un botón con el texto "agregar amigo". Al pulsar el botón el usuario en sesión esta enviando una invitación de amistad a ese usuario. Un usuario no podrá enviarse una invitación de amistad así mismo. Hay que validar la invitación del lado del servidor (No vale sólo con ocultar el botón de envío).

Pruebas Funcionales

[Prueba15] Desde el listado de usuarios de la aplicación, enviar una invitación de amistad a un usuario. Comprobar que la solicitud de amistad aparece en el listado de invitaciones (punto siguiente).

[Prueba16] Desde el listado de usuarios de la aplicación, enviar una invitación de amistad a un usuario al que ya le habíamos enviado 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.

W7 Usuario registrado: Listar las invitaciones de amistad recibidas

Se visualizará en una lista todas las invitaciones de amistad recibidas por el usuario identificado en sesión. Para cada invitación se mostrará el nombre y apellidos del usuario de la aplicación que solicito la amistad.

Debe incluirse una opción de menú principal, visible solo para usuarios en sesión que permita acceder al listado de todas las invitaciones de amistad recibidas.

La lista de invitaciones debe incluir un sistema de paginación y mostrar 5 invitaciones por página.

Pruebas Funcionales

[Prueba17] Mostrar el listado de invitaciones de amistad recibidas. Comprobar con un listado que contenga varias invitaciones recibidas.

W8 Usuario registrado: Aceptar una invitación recibida

Junto a cada invitación mostrada en la lista de invitaciones de amistad recibidas se debe incluir un botón con el texto "aceptar", al pulsar este botón la invitación de amistad debe desaparecer de la lista de invitaciones y el usuario que la envió pasará a ser un amigo del usuario en sesión y viceversa (A es amigo de B, B es amigo de A).

[Prueba18] Sobre el listado de invitaciones recibidas. Hacer click en el botón/enlace de una de ellas y comprobar que dicha solicitud desaparece del listado de invitaciones.

W9 Usuario registrado: Listar los usuarios amigos

Se visualizará en una lista todos los usuarios amigos del usuario en sesión. Para cada usuario se mostrará su nombre, apellidos y email.

La lista de amigos debe incluir un sistema de paginación y mostrar 5 usuarios por página.

Debe incluirse una opción de menú principal, visible solo para usuarios en sesión que permita acceder al listado de todos los amigos del usuario en sesión.

[Prueba19] Mostrar el listado de amigos de un usuario. Comprobar que el listado contiene los amigos que deben ser.

W10 Seguridad

Deberá diseñarse adecuadamente la política de seguridad con los mecanismos de seguridad de Node.Js para que no existan situaciones de vulnerabilidad en el acceso a recursos/acciones de usuarios registrados.

Pruebas Funcionales

[Prueba20] Intentar acceder sin estar autenticado a la opción de listado de usuarios. Se deberá volver al formulario de login.

[Prueba21] Intentar acceder sin estar autenticado a la opción de listado de invitaciones de amistad recibida de un usuario estándar. Se deberá volver al formulario de login.

[Prueba22] Intentar acceder estando autenticado como usuario standard a la lista de amigos de otro usuario. Se deberá mostrar un mensaje de acción indebida.

Parte 2 "servicios web"

Se deberá implementar los servicios web REST correspondientes a una aplicación de mensajería/chat similar al WhatsApp que permitirá enviar y recibir mensajes de otros usuarios amigos, estos servicios web deberán de estar contenidos en la aplicación web anterior (no hay que desarrollar una nueva aplicación). Además de los servicios implementará también una aplicación jQuery-AJAX que los consuma, esta aplicación debe permitir el intercambio de mensajes en tiempo real. El cliente también se incluirá dentro del mismo proyecto, en la carpeta "public".

Los casos de usos S1 – S4 y C1 – C4 son de carácter obligatorio, para que el trabajo pueda ser evaluado. Además, los casos de uso C1-C4 deben incluir sus correspondientes pruebas automatizadas (no se evaluarán los casos de uso que no contengan pruebas) y la puntuación máxima a la que se opta por implementar estos 8 casos de forma perfecta es de 5 sobre 10 dentro de la parte 2 "servicios web". Mientras que los casos de uso S1 – S4 no requerirán pruebas automatizadas.

Los casos de uso S5 y C5 – C7 son de carácter opcional, la puntuación máxima a la que se opta por implementar todos los casos de forma perfecta es de 5 sobre 10 dentro de la parte 2 "servicios web"

Parte 2A - Implementación de la API de Servicios Web REST

S1 Identificarse con usuario – token

El servicio recibe un nombre de usuario y una contraseña, en caso de que exista coincidencia con algún usuario almacenado en la base de datos retornará un token de autenticación. Sí no hay coincidencia retornará un mensaje de error de inicio de sesión no correcto.

S2 Usuario identificado: Listar todos los amigos

El servicio retorna una lista con los identificadores de todos los amigos del usuario identificado.

Para permitir listar los amigos el usuario debe de estar identificado en la aplicación, por lo tanto, la petición debe contener un token de seguridad válido.

S3 Usuario identificado: Crear un mensaje

El servicio debe permitir crear nuevos mensajes.

Las propiedades del mensaje son: emisor, destino, texto, leído (true/false), por defecto el mensaje se crea como no leído.

Para permitir crear un nuevo mensaje, el usuario destino debe ser amigo del usuario identificado, por lo tanto, la petición debe contener un token de seguridad valido.

S4 Usuario identificado: Obtener mis mensajes de una "conversación"

El servicio recibe el identificador de dos usuarios y retorna una lista con todos los mensajes para cuales esos usuarios son emisor y receptor o viceversa (mensajes enviados en ambos sentidos para esos dos usuarios).

Para permitir obtener los mensajes de una conversación el usuario identificado debe ser el emisor o receptor de los mensajes, por lo tanto, la petición debe contener un token de seguridad valido.

*** S5 Usuario identificado: Marcar mensaje como leído**

Este servicio permite marcar un mensaje como leído, la propiedad leído debe tomar el valor true. El servicio recibe el identificador del mensaje.

Para permitir cambiar el estado de un mensaje, el usuario identificado debe ser el receptor del mensaje, por lo tanto, la petición debe contener un token de seguridad valido.

Parte 2B - Cliente - Aplicación jQuery

C1. Autenticación del usuario

Debe permitir la autenticación a través de un formulario donde se solicite el correo y la contraseña de usuario, se debe informar si la autenticación no se realiza con éxito.

Pruebas Funcionales

[Prueba23] Inicio de sesión con datos válidos.

[Prueba24] Inicio de sesión con datos inválidos (usuario no existente en la aplicación).

C2. Mostrar la lista de amigos

Una vez autenticado se debe redirigir al usuario a su lista de amigos (no aplicar ningún sistema de paginación). Dentro de esta lista se debe ofrecer la posibilidad de filtrar los amigos por su nombre.

Pruebas Funcionales

[Prueba25] Acceder a la lista de amigos de un usuario, que al menos tenga tres amigos.
[Prueba26] 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.

C3. Mostrar los mensajes

Al pulsar sobre el nombre de un usuario de la lista de amigos se debe mostrar el chat, el cual incluye:

- La lista con todos los mensajes relacionados con ese usuario y el usuario identificado en la aplicación.
- Formulario para enviarle un nuevo mensaje a ese usuario

La lista de mensajes debe actualizarse en tiempo real, sin necesidad de recargar la página ni de pulsar ningún botón. Es decir, debe haber un proceso automático que compruebe cada segundo si hay nuevos mensajes en esa conversación.

Pruebas Funcionales
[Prueba27] Acceder a la lista de mensajes de un amigo "chat", la lista debe contener al menos tres mensajes.

C4. Crear mensaje

Desde la vista de chat el usuario debe poder crear un nuevo mensaje para ese amigo. Siendo el usuario el emisor y el amigo el destino.

Pruebas Funcionales
[Prueba28] 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.

*C5. Marcar mensajes como leídos de forma automática

Al entrar en un chat de un amigo todos los mensajes no leídos deben ser marcados como leídos. Junto a cada mensaje mostrado en el chat debe incluirse un <leído> al final (si tiene el estado leído).

Al igual que la lista de mensajes el estado debe actualizarse en tiempo real, sin necesidad de recargar la página ni de pulsar ningún botón. Es decir, debe haber un proceso automático que compruebe cada segundo si hay nuevos mensajes en esa conversación / o cambios en los estados leído.

Pruebas Funcionales

[Prueba29] Identificarse en la aplicación y enviar un mensaje a un amigo, validar que el mensaje enviado aparece en el chat. Identificarse después con el usuario que recibido el mensaje y validar que tiene un mensaje sin leer, entrar en el chat y comprobar que el mensaje pasa a tener el estado leído.

*C6 Mostrar el número de mensajes sin leer

Se debe mostrar junto al nombre de cada amigo (en la lista de amigos) cuantos mensajes sin leer hay en esa conversación.

El número de mensajes sin leer debe actualizarse en tiempo real, sin necesidad de recargar la página ni de pulsar ningún botón. Es decir, debe haber un proceso automático que compruebe cada segundo si hay nuevos mensajes.

Pruebas Funcionales

[Prueba30] Identificarse en la aplicación y enviar tres mensajes a un amigo, validar que los mensajes enviados aparecen en el chat. Identificarse después con el usuario que recibido el mensaje y validar que el número de mensajes sin leer aparece en la propia lista de amigos.

*C7 Ordenar la lista de amigos por último mensaje

Se debe incluir un mecanismo de ordenación automático de la lista de amigos, de forma que los amigos se ordenen por la antigüedad en la creación del último mensaje (teniendo en cuenta mensajes tanto enviados como recibidos). Las conversaciones más recientes deben tener prioridad respecto a las menos.

Pruebas Funcionales

[Prueba31] Identificarse con un usuario A que al menos tenga 3 amigos, ir al chat del ultimo amigo de la lista y enviarle un mensaje, volver a la lista de amigos y comprobar que el usuario al que se le ha enviado el mensaje esta en primera posición. Identificarse con el usuario B y enviarle un mensaje al usuario A. Volver a identificarse con el usuario A y ver que el usuario que acaba de mandarle el mensaje es el primero en su lista de amigos.

Pruebas automatizadas

Se deberá suministrar un proyecto Java JUnit con un mínimo de pruebas unitarias empleando el framework Selenium (Se suministrará en el campus virtual un proyecto plantilla de ejemplo).

Para cada una de estas pruebas se debe añadir al menos un caso de prueba y en caso de querer añadir más casos se añadirá numeración extra al nombre de la prueba. P.e. para la prueba PR06, el primer caso de prueba será el método: PR06, y los extra que se deseen añadir se enumerarán como PR06_1, PR06_2,… .

En caso de desear incluir más se deberán incluir al final de la clase de pruebas JUnit.

Todas las pruebas deben incluir el click en las opciones de menú correspondientes, etc, etc, igual que lo haría un usuario real.

El protocolo de prueba

Para probar cada proyecto el profesor realizará los siguientes pasos:

a) Importar y ejecutar la aplicación Node.Js

b) Importará y ejecutar el proyecto JUnit en STS.

Aspecto Generales

Seguridad

Deberán tenerse en cuenta los siguientes aspectos de seguridad:

- Emplear la técnica de autentificación/autorización más adecuada a este contexto.
- En caso necesario comprobar el permiso de cada usuario para acceder a las URL o recursos solicitados.
- Registrar el acceso de los usuarios y la actividad en un Logger (por ejemplo log4js).

Arquitectura

La aplicación deberá estar obligatoriamente diseñada siguiente el patrón arquitectónico visto en clase. La utilización incorrecta de elementos de esta arquitectura será fuertemente penalizada (por ejemplo, implementar parte de la lógica de negocio de la aplicación en un controlador, acceder a un repositorio desde un controlador, configurar una vista desde un servicio, etc.).

Los servicios web REST deben seguir la arquitectura RESTful, se deben utilizar URLs y métodos Http adecuados en cada caso.

Otros aspectos que serán evaluados

- Claridad y calidad de la implementación del código JavaScript
- Calidad de la implementación de las vistas y usando todas las funcionalidades vistas en clase.

Entrega

El número n de cada trabajo será el ID GIT del integrante del grupo (en caso de un integrante) o la concatenación de los ID GIT(en caso de dos ej: 801-804). Según ese número se deberá crear un proyecto Node.Js (en WebStorm) y un proyecto JUnit (en STS) con los respectivos nombres **sdi1920-entrega2-n y sdi1920-entrega2-test-n**.

Según lo anterior la entrega consistirá en los siguientes puntos:

1. Subir los proyectos Node.Js y JUnit a un repositorio GIT con nombre **sdi1920-entrega2-n**. E invitar al usuario **sdigithubuniovi** como colaborador. Sobre dicho repositorio deberán realizarse actualizaciones frecuentes para que se pueda hacer un seguimiento del ritmo de trabajo.
2. Subir a la tarea del Campus Virtual correspondiente un archivo ZIP (usando el formato ZIP) con el nombre **sdi1920-entrega2-n.zip** (en minúsculas) y que deberá contener en su raíz:
    - El INFORME OBLIGATORIO en formato PDF con nombre **sdi1920-entrega2-n.pdf,** que contenga:

  - Una descripción clara y detallada de la implementación de cada uno de los casos de uso implementados.
  - Cualquier otra información necesaria para una descripción razonablemente detallada de lo entregado y su correcto despliegue y ejecución.

- El CATÁLOGO de CASOS DE PRUEBA con nombre sdi-entrega2-test-n.xlsx y rellenado con los casos de pruebas realizados, fallados y no realizados, así como los comentarios oportunos.
- El proyecto **Node.Js** en formato carpeta (no comprimido) con el nombre **.\sdi1920-entrega2-n**
- El proyecto **Junit** exportado desde STS (no comprimido) con el nombre **.\sdi1920-entrega2-test-n.**
- En resumen, el zip deberá contener en su raíz (**MUY IMPORTANTE**: el proyecto Junit deberá renombrarse con la n correspondiente dentro de STS antes de exportarlos para meterlos en el zip):
  - sdi1920-**entrega2**-n.pdf (Archivo PDF suministrado y renombrado cambiado la n).
  - sdi1920-**entrega2**-n.xlsx (Archivo Excel suministrado y renombrado cambiado la n).
  - sdi1920-**entrega2**-n (Carpeta del proyecto Node.Js renombrada cambiado la n)
  - sdi1920-**entrega2**-test-n (Carpeta del proyecto Junit renombrada cambiado la n)

Fecha máxima de entrega

La fecha límite de entrega es el Lunes día 11 de Mayo a las 23:55. No se recibirá ningún trabajo fuera de plazo.

Evaluación

Se penalizará:

- Que la compilación del código genere "*warnings*".
- Que se presenten problemas durante el despliegue.