SDI – Sistemas Distribuidos e Internet

ENUNCIADO PRÁCTICA 2 – NODE.Js & Servicios Web

INFORME

507

Nombre1:	Alejandro
Apellidos1:	León Pereira
Email1:	uo258774@uniovi.es
Cód. ID GIT	507

Índice

Introducción

Éste trabajo se ha hecho de manera individual.

El repositorio de GitHub donde pueden verse los commits de este proyecto y por ende su desarrollo es:

https://github.com/alejandroleon98/sdi1920-entrega2-507

Los test se encuentran en este repositorio:

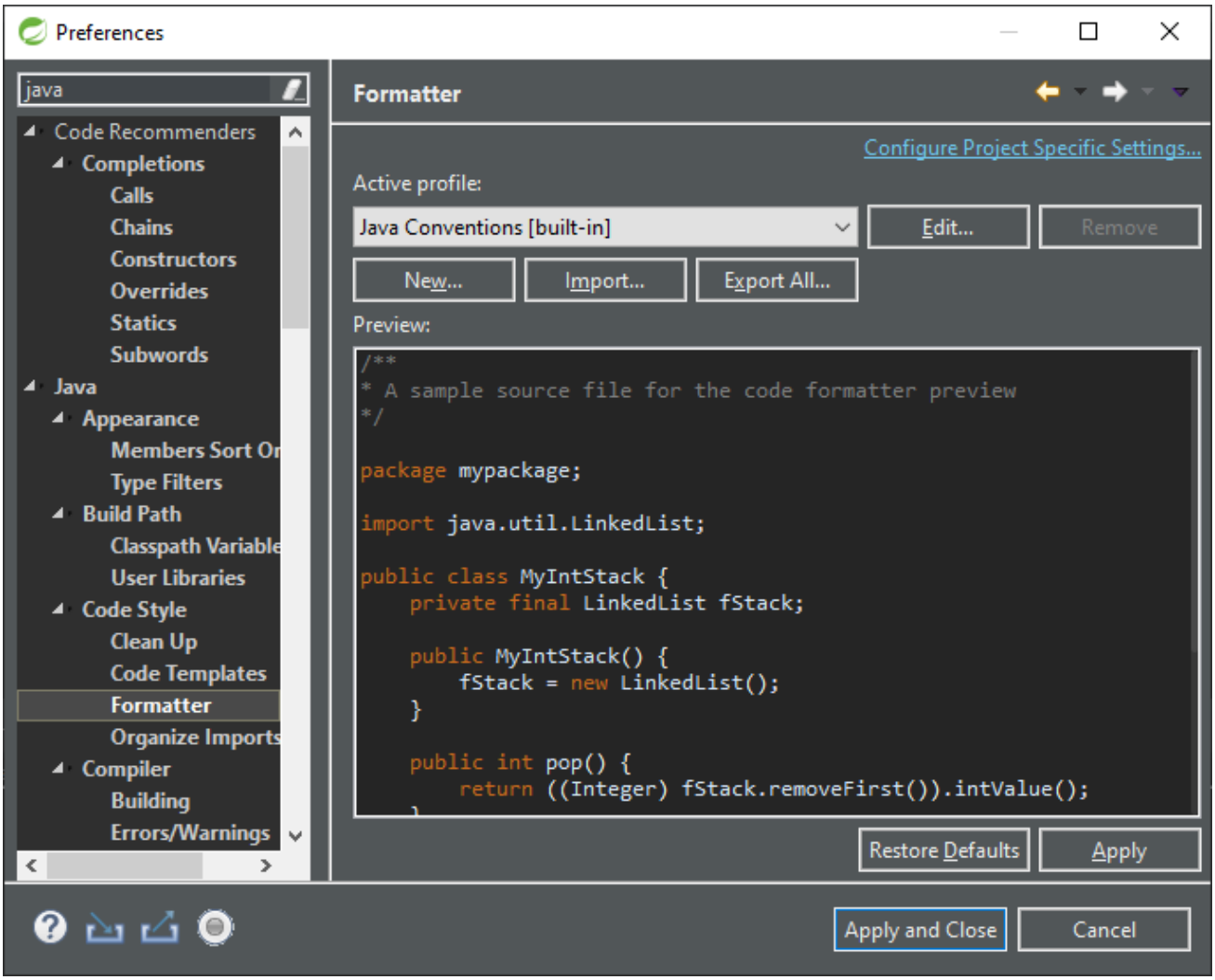
https://github.com/alejandroleon98/sdi1920-entrega2-test-507

Se ha invitado como colaborador a los repositorios a la cuenta https://github.com/sdigithubuniovi

En la realización de los test se ha intentado seguir en la medida de lo posible las "Java Code Conventions", cuyas normas se pueden encontrar aquí:

https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

Para ello se utiliza su formateador:

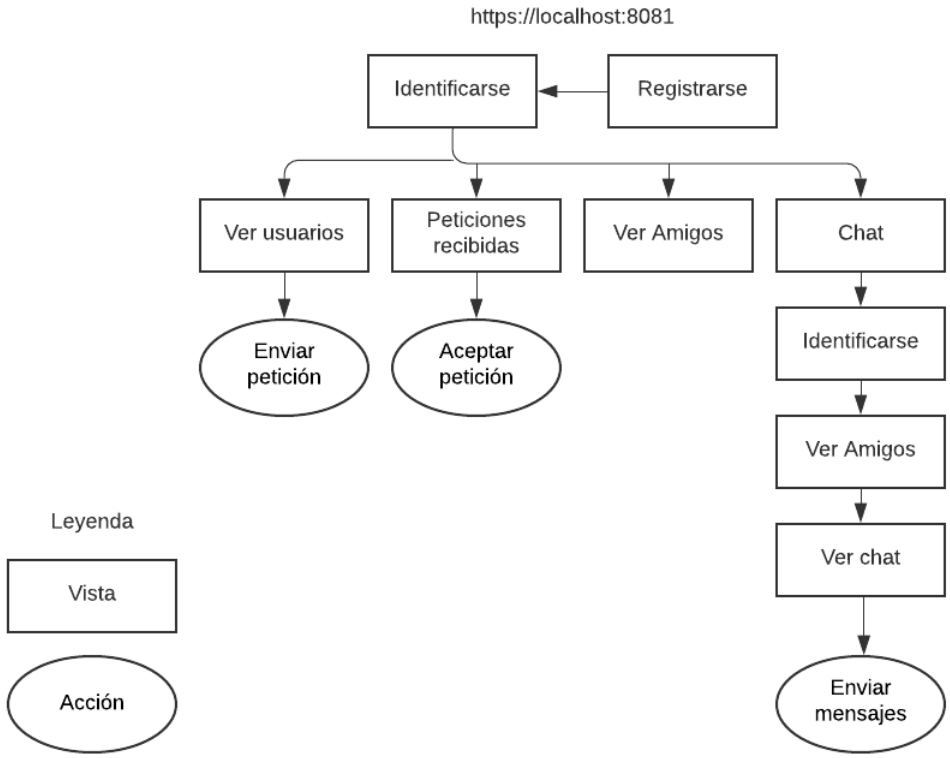


No obstante, hay una recomendación que dice que las líneas de código no deben ser demasiado largas, para poder leer el código sin tener que desplazarse a derecha e izquierda sobre todo en pantallas con

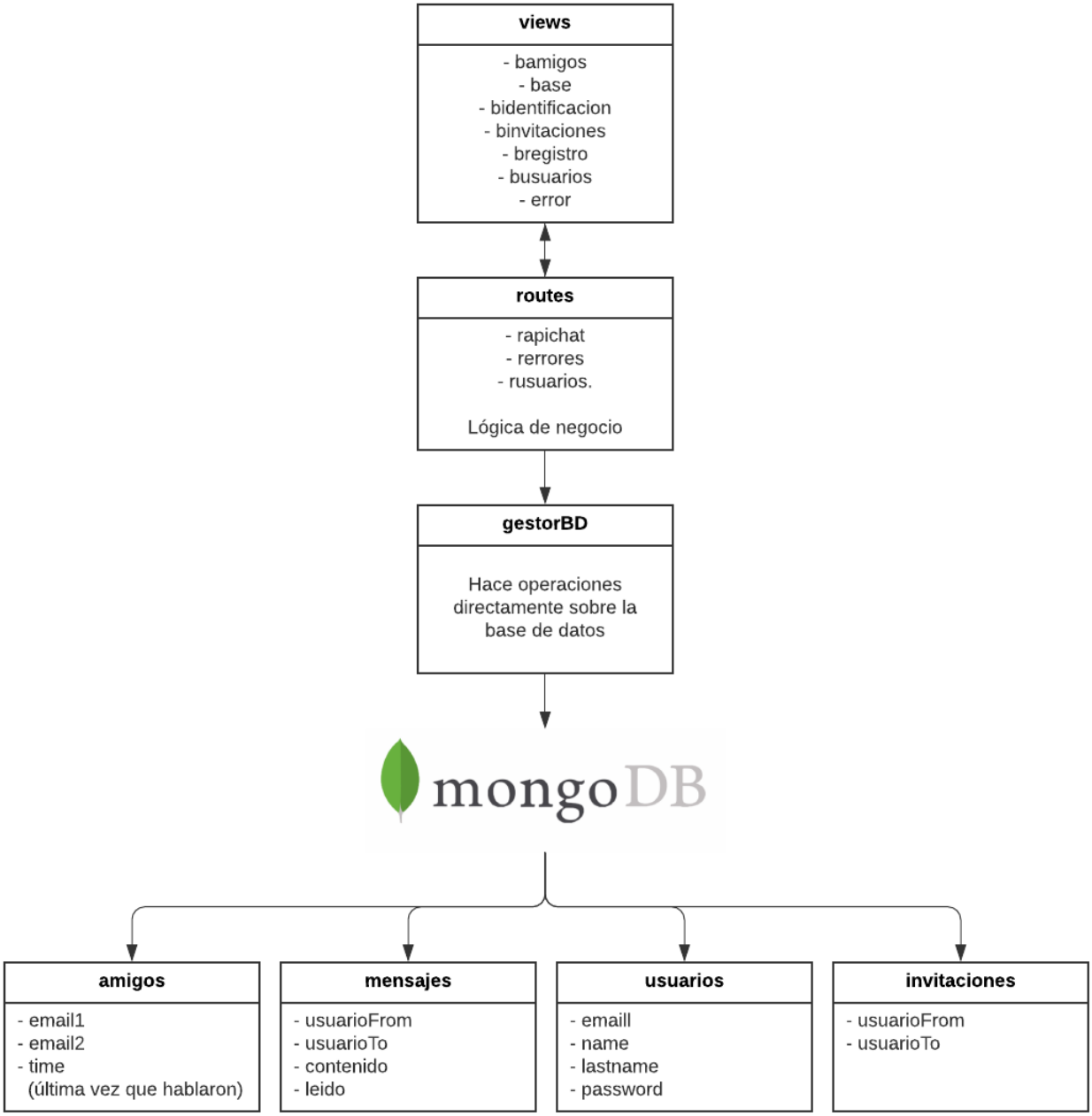
poca resolución. Se ha omitido esta comprobación porque realmente hace más ilegible el código en éste caso, ya que se manejan strings, que de estar partidos, se hace muy difícil su legibilidad debido a las mezclas de diferentes comillas.

Mapa de navegación

Mapa de navegación por el interfaz de usuario:



Esquema de la arquitectura del proyecto

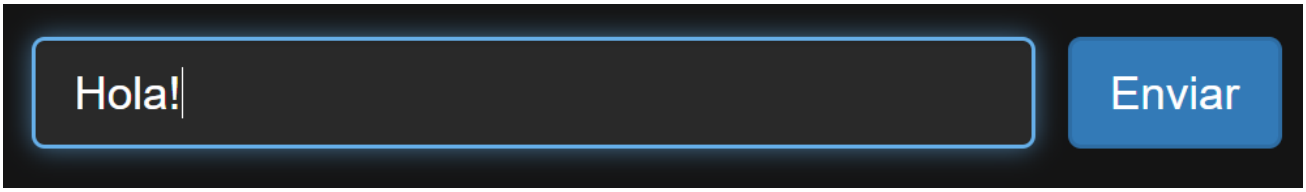


Aspectos técnicos y de diseño relevantes

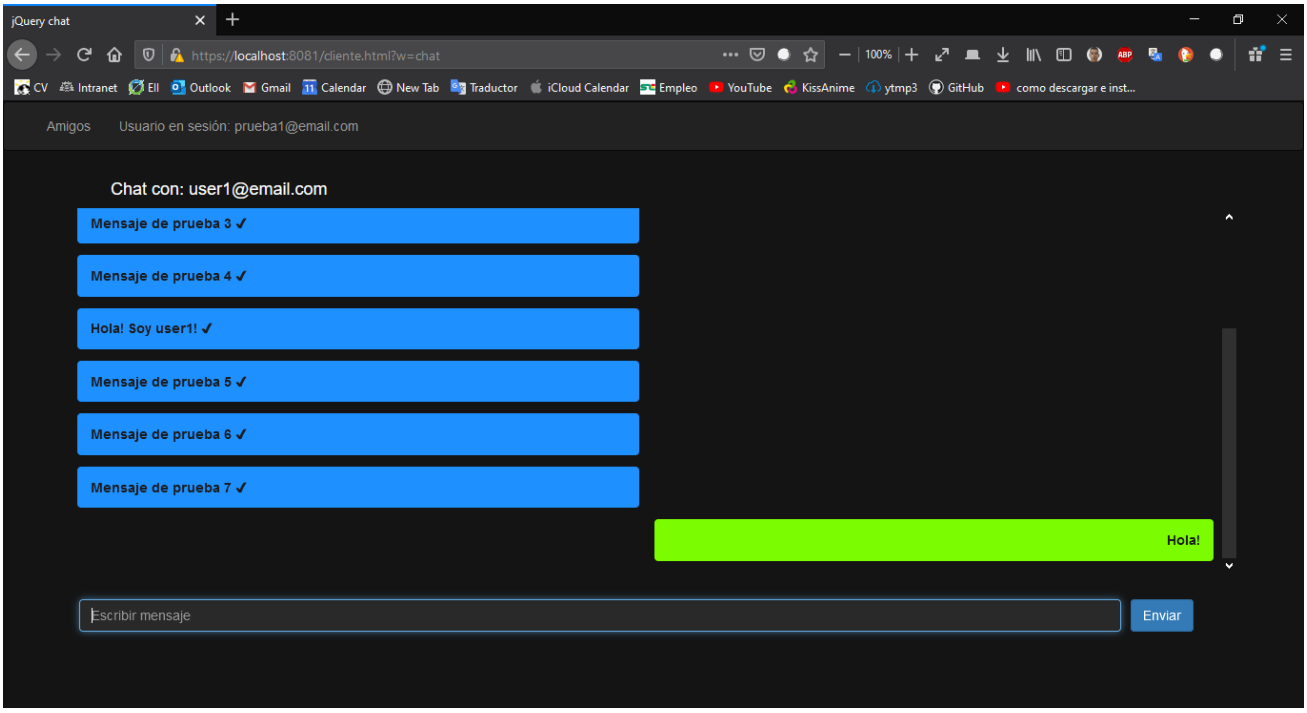
Respecto a la base de datos, no se repiten datos, se utiliza como "primary key" el email de cada usuario, para poder acceder al resto de sus atributos (en este caso es nombre y apellidos, pero si se quisieran añadir más no habría problema ya que no habría que cambiar otras tablas). usuarioFrom y usuarioTo son a su vez emails.

Respecto al cliente, el código javascript comprueba cada pocos segundos si se han recibido nuevos mensajes cuando el usuario está en la pantalla del chat, y solamente actualiza el chat si es necesario (si se han recibido nuevos). Igualmente con la tabla de amigos, sólo se actualiza si hay mensajes nuevos y por tanto cambia el orden de los usurios, o el número de mensajes sin leer de alguno de ellos.

En las vistas html se comprueba la validez de los campos, aunque también se hace en los routers. Es el éstos últimos en donde se hacen las comprobaciones de la lógica de negocio (como por ejemplo, no poder aceptar a un usuario sin que te haya invitado primero, o no poder invitar a un usuario que ya es amigo). No obstante, las vistas siempre intentan validar lo máximo posible, como en el chat, donde el botón de enviar mensaje está bloqueado hasta que se inserta una cadena de entre 1 y 400 caracteres.



Por cierto, el cliente tiene un tema oscuro:



Respecto a los test, se intenta simular la interacción con el usuario lo máximo posible, por lo que se evita navegar por las url's, se intenta hacerlo siempre que se pueda por los botones de la interfaz de usuario. Las url's solo se utilizan para probar casos de la seguridad, por ejemplo que no se pueda agregar a un amigo sin que te haya mandado una invitación, o mandar invitaciones o añadir usuarios sin estar logueado.

Tener unos casos de prueba ayuda mucho a la hora de implementar nueva funcionalidad, ya que si tienes que andar probando que todo funciona a mano despues de hacer algún commit con muchos cambios, es muy tedioso. Está muy bien poder pasar los test después de hacer grandes cambios para ver no ha afectado a las funcionalidades de antes, o para ver qué a dejado de funcionar exactamente.

Una cosa importante a comentar es que he quitado los console.log(error), ya que es buena costumbre. Una traza de error podría tener información valiosa para un atacante. Sobre todo en el cliente, donde los mensajes se pueden ver en la consola del navegador. Otros console.log() donde se imprime un string como "mensaje enviado" no hacen daño. Toda la información importante como registros e inicios de sesión, y toda la actividad que realiza un usuario tanto en la aplicación como en el cliente, queda registrada en un log (en /logs/FriendsManager.log). Por ejemplo:

[10-05-2020 21:05:23] [INFO] - Cliente: user1@email.com envia mensaje a: prueba1@email.com. Contenido: Hola! Soy user1!

Cosas a mejorar:

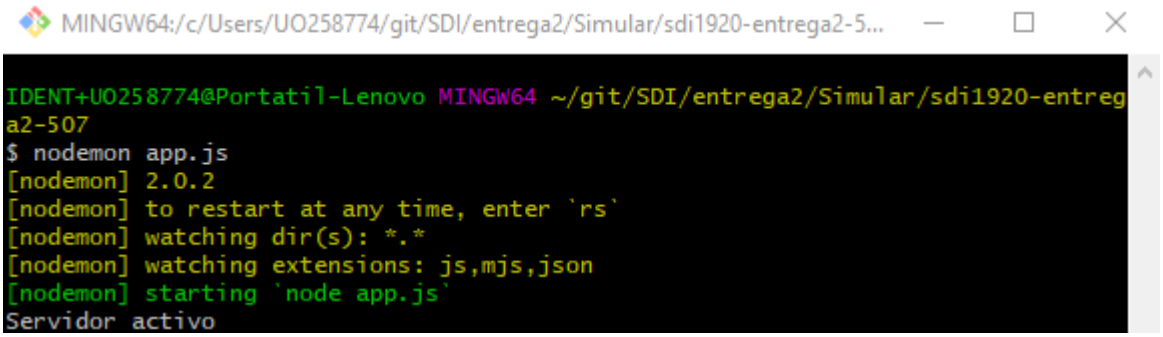
- Si tuviera que empezar un proyecto ahora lo haría con bootstrap 4, ya que tiene muchas facilidades para hacer temas oscuros, además de una interfaz más amigable, y es muy fácil de utilizar. Dedicando poco tiempo se puede lograr una interfaz de usuario muy lograda. Lo malo es que al haber utilizado en bootstrap 3 en el comienzo, cambiarlo sería difícil ya que cambiaría la aplicación entera y habría que tocar muchas cosas.
- Estaría bien adaptar la aplicación para pantallas pequeñas con css.
- Hacer el código en inglés en su totalidad hubiera estado bien. Suelo codear en inglés pero como este proyecto estaba empezado en español seguí para adelante. El problema es que ahora está en inglés y en español, y alguien que tuviera que trabajar en el tendría que entender los dos idiomas.
- En un router, hay una dirección (https://localhost:8081/borrarUsuarios) que al visitarla borra todos los usuarios de la base de datos, junto con sus mensajes, amigos y peticiones. Esto se hace con la intención de poder ejecutar los test una y otra vez bajo el mismo entorno de prueba, y obviamente en caso de desplegar ésta aplicación habría que resetear la base de datos de otro método más seguro, ya que ahora cualquier usuario sin siquiera estar logueado puede borrar la base de datos con solo acceder a esa dirección. No se ha implementado ninguna medida de seguridad para acceder a esa dirección porque ya de por sí no esta bien si quiera tener una url de esa manera, solo se hace para facilitar los test en un entorno de desarrollo, y al desplegar la aplicación habría que quitar ese método y solo utilizarlo cuando se vayan a pasar los test, en otra rama por ejemplo.

  Es por ello que no se requiere autenticación siquiera para acceder a ese método, ya que en un despliegue estaría comentado o borrado.

Información necesaria para el despliegue y ejecución

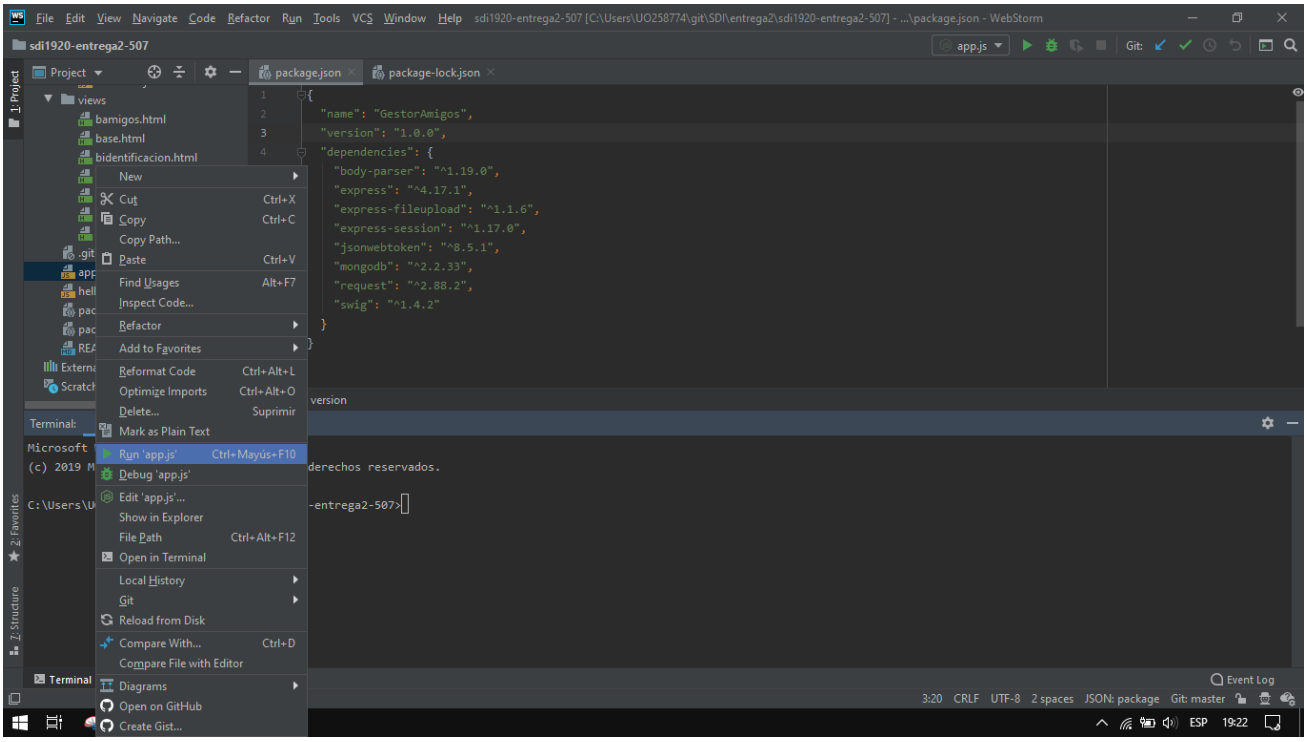
1. Proyecto

Para ejecutar el proyecto basta con descomprimirlo y ejecutar nodemon app.js dentro de la carpeta sdi1920-entrega2-507



Ya tiene instaladas todas las dependencias.

También se puede hacer desde el webstorm



app.js -> Run app.js

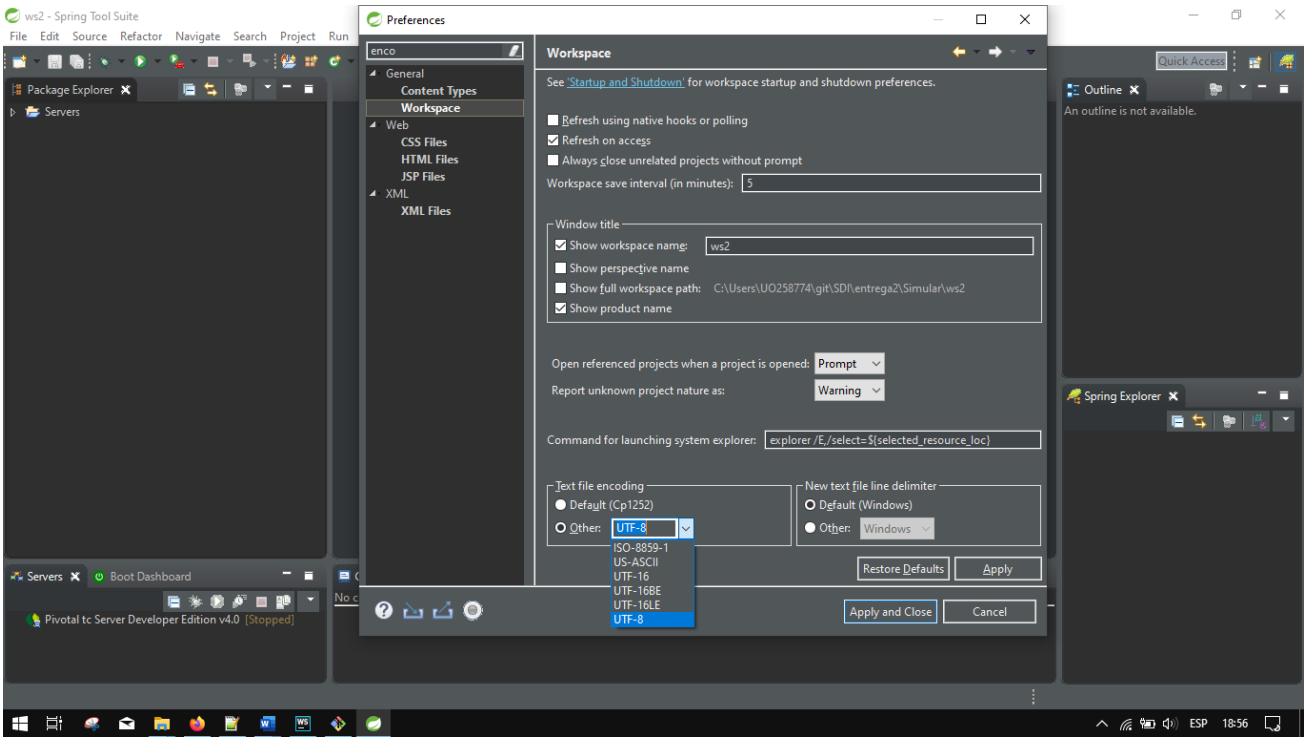
2. Test:

IMPORTANTE

La aplicación está en español, por lo que hay ñ y acentos. Se debe de configurar el workspace de la siguiente manera antes de añadir el proyecto:

Window -> preferences: buscamos “enco” y entramos en Workspace

Selecciomanos UTF-8 en text file encoding:



Si no se hace esto, no pasarán las pruebas.

Guía para añadir el proyecto de pruebas:

- Copiar la carpeta sdi1920-entrega2-test-507 al workspace de eclipse o spring.
- Desde eclipse o spring: file -> import -> general -> existing projects into workspace -> seleccionamos la carpeta del workspace y importamos el proyecto

Ahora hay que configurar los test:

FriendsManagerTests.java

```
1  package com.uniovi.tests;
2  //Paquetes Java
5  import org.junit.*;
18
19
20 //Ordenamos las pruebas por el nombre del mÃ©todo
21 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
22 public class FriendsManagerTests {
23     //En Windows (Debe ser la versiÃƒÂ³n 65.0.1 y desactivar las actualizaciones automÃƒÂ¡ticas):
24     static String PathFirefox65 = "C:\\Program Files\\Mozilla Firefox6501\\firefox.exe";
25     static String Geckdriver024 = "C:\\Users\\UO258774\\git\\SDI\\entrega2\\geckodriver024win64.exe
26     //En MACOSX (Debe ser la versiÃƒÂ³n 65.0.1 y desactivar las actualizaciones automÃƒÂ¡ticas):
27     //static String PathFirefox65 = "/Applications/Firefox 2.app/Contents/MacOS/firefox-bin";
28     //static String PathFirefox64 = "/Applications/Firefox.app/Contents/MacOS/firefox-bin";
29     //static String Geckdriver024 = "/Users/delacal/Documents/SDI1718/firefox/geckodriver024mac";
30     //static String Geckdriver022 = "/Users/delacal/Documents/SDI1718/firefox/geckodriver023mac";
31     //ComÃƒÂºn a Windows y a MACOSX
32     static WebDriver driver = getDriver(PathFirefox65, Geckdriver024);
33     static String URL = "https://localhost:8081";
34
```

Seleccione la ruta a su navegador firefox, y la ruta al archivo geckodriver024win64.exe, que se suministrará junto con éste proyecto.

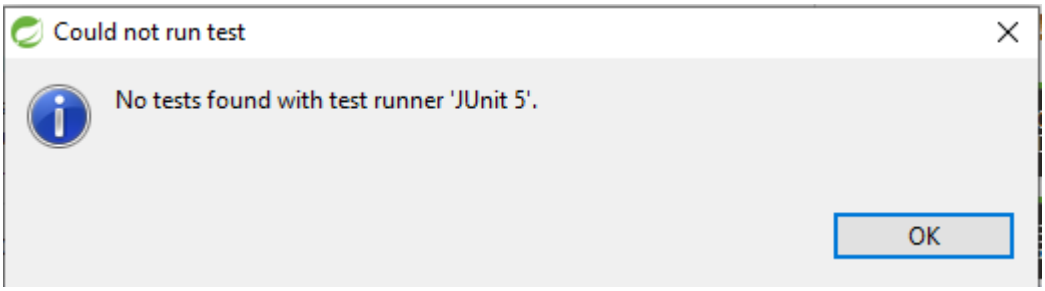
El proyecto consta de tests que simulan la interación del usuario con el entorno gráfico de un navegador.

Si usted está utilizando el sistema operativo MacOS, por favor, descomente las líneas correspondientes en la clase NotaneitorTest.java. Están al principio de la clase.

Ya se pueden ejecutar los test:

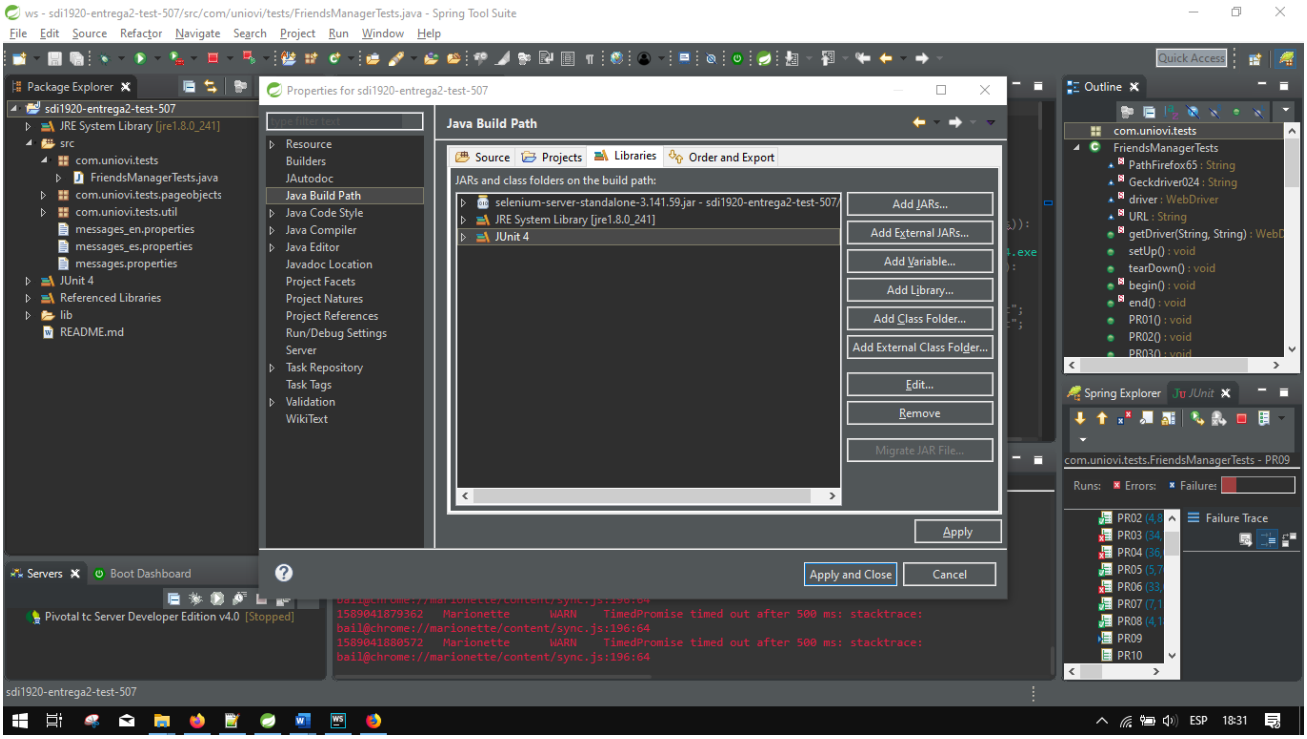
Click derecho sobre la clase FriendsManagerTest.java -> Run As -> JUnit test

Si sale el siguiente error:

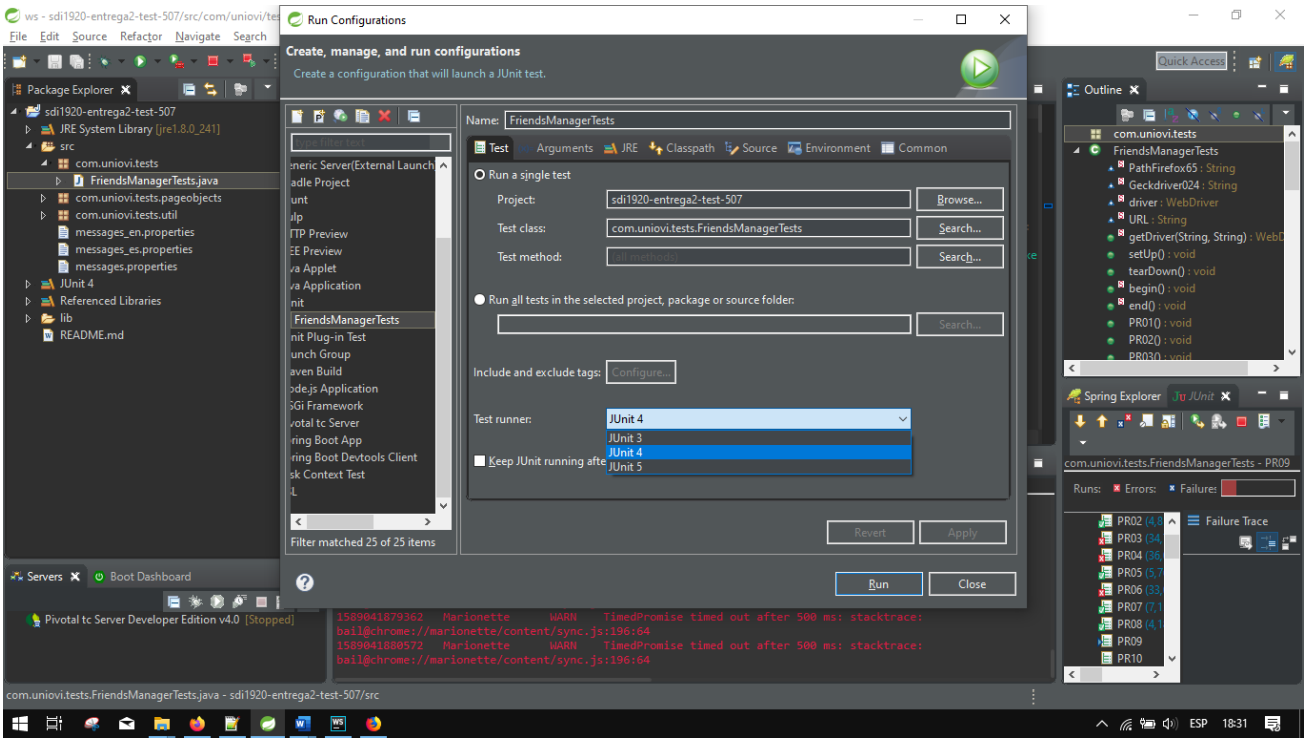


Seguir los siguientes pasos:

- En las propiedades del proyecto, asegurarse de que se ha añadido la librería JUnit 4

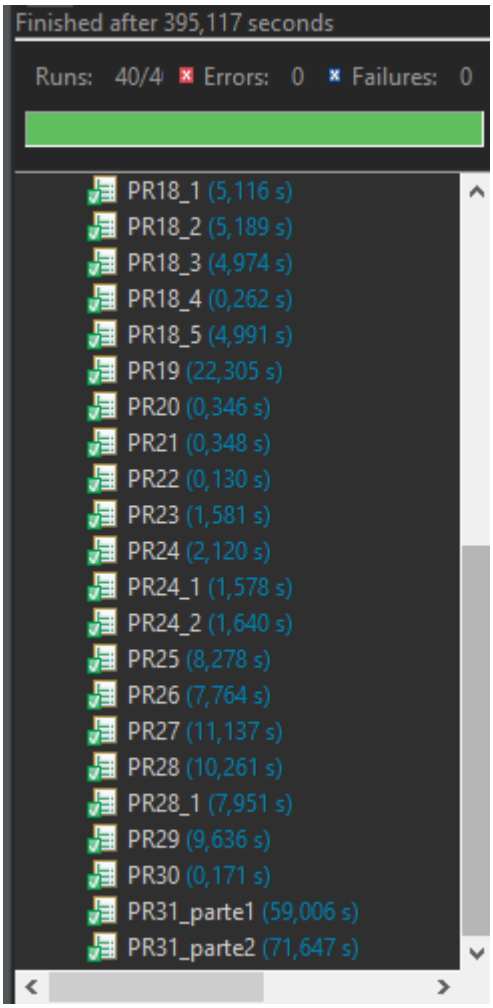


- Ejecutar los test con la configuración siguiente, utilizando JUnit 4



(Para que funcionen los test, primero hay que ejecutar la aplicación, y luego la clase NotaneitorTests.java como caso de pruebas JUnit4)

Una vez ejecutados los test, deberían pasar todos sin problema:



Conclusión

Estoy bastante contento con el resultado de la práctica. Me ha gustado más la parte de la api. La he probado con dos navegadores chateando con dos usuarios a la vez, y todas las funcionalidades funcionan genial. Lo más importante es tener en cuenta qué funciones se pueden ejecutar de manera asíncrona y cuales no.