Ferramentas de Desenvolvemento

Memoria de la Práctica

Wallashop

Miembros.

Álvaro Pérez Pedreira	-	alvaro.perez.pedreira@udc.es
Miguel Rodríguez Pérez	-	miguel.rodriguezp@udc.es
Martín Torreira Portos	-	m.tportos@udc.es
Ignacio Freiría López	-	ignacio.freiria@udc.es
Borja Fernández Plana	-	borja.fernandez.plana@udc.es
Rogelio Daniel Franco Martínez	-	r.d.franco@udc.es
David Vega Guisasola	-	david.vguisasola@udc.es

Índice

1- Introducción.

La aplicación que hemos desarrollado llamada Wallashop se basa en la publicación de varios tipos de posts, con diferentes categorías y precios. Los usuarios podrán registrarse e iniciar sesión dentro de la aplicación y cada usuario contará con una serie de campos en el perfil a los cuales tendrá acceso en caso de querer modificarlos.

Una vez un usuario haya iniciado sesión en la aplicación podrá publicar o comentar en un post, a su vez también podrá responder a un comentario. El usuario que reciba una respuesta ya sea a un post que ha creado o a un comentario que ha escrito recibirá una notificación.

El usuario a su vez podrá modificar los campos de un post que haya creado y en caso de verlo conveniente, podrá también eliminarlo de la aplicación. A la hora de añadir o eliminar un post, todos los usuarios activos en la aplicación recibirán una notificación para recargar la página y poder consultar todas las novedades.

La aplicación también contará con un sistema de búsquedas donde podrás filtrar las publicaciones por varios campos, y en varios órdenes (ascendente o descendente).

Los usuarios también podrán marcar un post para que si en algún caso dicho post cambia algún parámetro estos sean notificados.

2- Herramientas usadas.

2.1- Eclipse.

Para el desarrollo de la práctica, hemos usado este entorno de desarrollo integrado (IDE) que es muy utilizado y de código abierto, también nos ha facilitado el desarrollo del software ya que es un excelente entorno para el desarrollo de aplicaciones Java. Este IDE cuenta con múltiples perspectivas y vistas, que nos han facilitado a la hora de programar, debuggear y testear la aplicación y también consultar la base de datos para confirmar que nuestras sentencias se ejecutaban correctamente.

Cabe destacar que es una plataforma extensible, lo que nos ayudó a la hora de instalar algún plugin para ayudarnos a la hora del desarrollo.

2.2- Git y Gitlab.

Git es un sistema de control de versiones distribuido diseñado para rastrear cambios en el código durante el desarrollo, siendo este uno de los más utilizados sino el que más. Nosotros lo hemos usado para subir los archivos a gitlab, gracias a esta herramienta hemos podido seguir un desarrollo de la práctica mucho más cómodo, debido a que hemos creado diferentes ramas principales (main, development, hotfix, release) de las que partían otras ramas (en este caso de development) en las que cada uno trabajaba independientemente del otro; una vez finalizadas todas las funcionalidades utilizamos git para hacer un merge a la rama principal solucionando los conflictos.

Gracias a esta herramienta hemos conseguido que el desarrollo de la práctica sea más rápido que lo que sin ella hubiese dificultado el trabajo en grupo, y también cabe destacar que nos ha permitido ir a versiones anteriores del código en caso de serlo necesario.

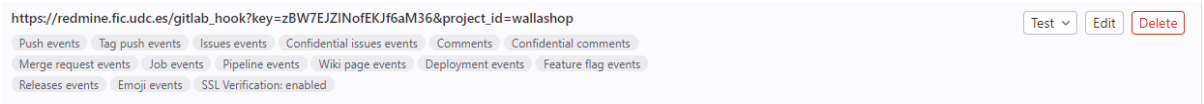
2.3- Redmine.

Redmine es una plataforma de gestión de proyectos y seguimiento de problemas de código abierto basada en la web. Esta herramienta fue vinculada al gitlab para ayudarnos a gestionar nuestras tareas.

Con esta herramienta aprendimos la manera correcta de crear las ramas de gitlab, ya que tienen que tener su estándar (feature/0000-example), también aprendimos a estimarlas a medida que las íbamos haciendo ya que normalmente las subestimamos.

Con Redmine también tenemos acceso a varios tipos de vistas, como un Diagrama de Gantt o un calendario para visualizar la planificación del proyecto.

Para configurar esta herramienta necesitamos hacer un webhook en gitlab.



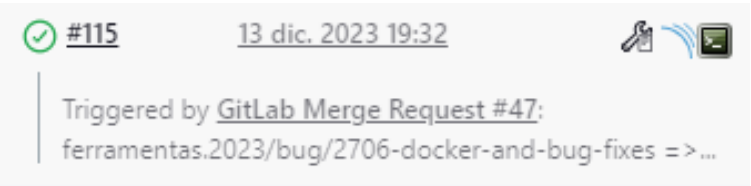
Gracias a esta herramienta conseguimos organizarnos a lo largo del desarrollo del proyecto.

2.4- Jenkins.

Jenkins es una herramienta ampliamente utilizada de código abierto que proporciona automatización de construcción, integración continua y despliegue continuo. Se utiliza para agilizar el proceso de desarrollo de software al automatizar tareas repetitivas y facilitar la integración y entrega de código.

Para usar esta herramienta hemos tenido que configurarla, para que se haga una build automáticamente al hacer un evento de push, un evento de apertura de una merge request y seleccionar la opción de que solo se *buildee* si se han añadido nuevos commits a la merge request, también en la configuración puedes añadir que se realicen antes de la ejecución y también aquí se configura el Sonar.

Una vez configurada la herramienta, las builds se hacen automáticamente y cada build te indica donde se ha *triggereado,* a continuación se muestra un ejemplo.



Gracias a esta herramienta hemos podido comprobar de manera más sencilla que los merges estaban hechos de manera correcta.

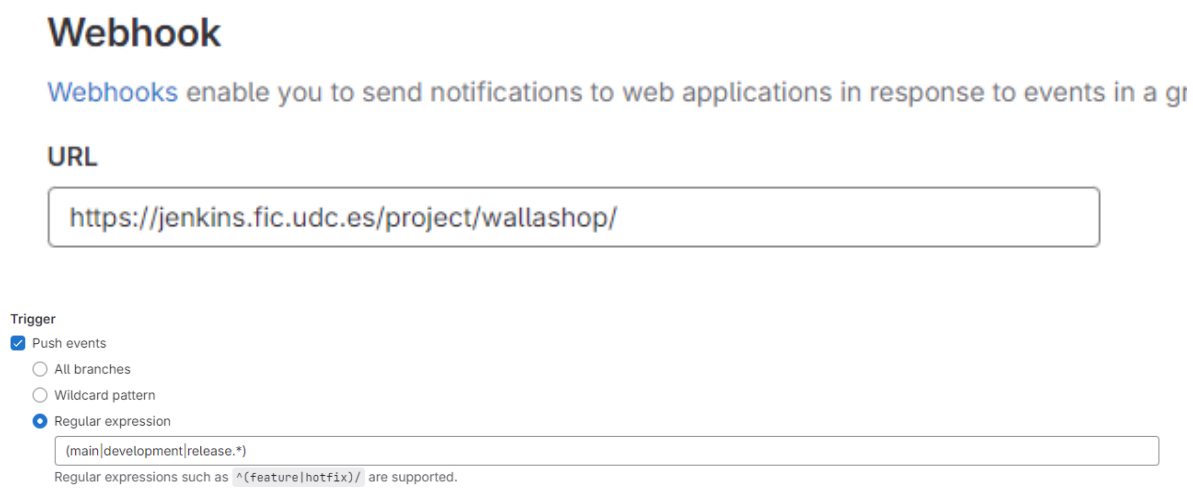
2.5- Sonar.

SonarQube es una plataforma de código abierto diseñada para evaluar y mejorar la calidad del código en proyectos de software, Sonar proporciona análisis estático de código para detectar posibles problemas, vulnerabilidades, bugs y patrones de código maliciosos. Sonar también cuenta con una variedad de métricas de calidad del código, informes detallados y una interfaz web que nos ha permitido monitorear y mejorar continuamente la salud del código de nuestra aplicación.

La configuración de Sonar también se lleva a cabo en el jenkins, con un pequeño script que se añade a los pasos previos, el script es el siguiente.

```
#!/bin/bash -xe

if [ -z ${gitlabMergeRequestIid+x} ];  then
  echo sonar.branch.name=$GIT_LOCAL_BRANCH > sonar-project.properties
else
  # si es merge request
  echo sonar.pullrequest.key=$gitlabMergeRequestIid > sonar-project.properties
  echo sonar.pullrequest.branch=$gitlabSourceBranch >> sonar-project.properties
  echo sonar.pullrequest.base=$gitlabTargetBranch  >> sonar-project.properties
fi
```

También se ha tenido que configurar un webhook en el gitlab para que la herramienta Sonar funcione correctamente.



Con esta pequeña expresión conseguimos que el trigger sea cuando haya algún evento de tipo push en las ramas main, development o release y todas las que partan de ella.

3- Librerías empleadas en el proyecto.

3.1- Firebase.

Firebase es una plataforma de desarrollo para aplicaciones que nos ha facilitado una base de datos en tiempo real que se almacena en la nube para almacenar cada una de nuestras imágenes de avatares y posts.

3.2- Next UI.

Es una librería de React que nos facilita algunos componentes, como por ejemplo algunos modales y botones, para la interfaz de usuario. Esta librería está basada en Tailwind CSS y a la hora de usarla hemos tenido que usar un provider en el índice de la app.

3.3- React Icons.

Es una librería de React que nos facilitó la integración de iconos en nuestra aplicación sin tener que preocuparnos por la descarga o por la gestión de archivos de iconos individuales.

3.4- Tailwind.

Tailwind CSS es una biblioteca de utilidades de estilo para la construcción de interfaces de usuario en aplicaciones, a diferencia de otros marcos de diseño tradicionales, este no proporciona componentes predefinidos; sino que se centra en proporcionar un conjunto extenso de utilidades de bajo nivel que permitan la construcción de diseños personalizados altamente responsives.

Por lo general hemos usado esta biblioteca para aplicar de una manera más rápida el CSS a nuestros componentes.

3.5- Material Tailwind.

Es una librería de código abierto de React que nos facilita algunos componentes, como por ejemplo algunos modales y botones, para la interfaz de usuario. Esta librería está basada en Tailwind CSS y a la hora de usarla hemos tenido que usar un provider en el índice de la app.

3.6- Stomp.

STOMP.js es la librería que hemos utilizado para conectarnos a WebSocket. Stomp nos permite “suscribirnos” a un socket que hemos creado con WebSocket en la parte del backend de nuestro proyecto. Una vez suscritos a este socket, cualquier mensaje que se envíe desde el backend será controlado por Stomp.

3.7- WebSocket.

Websocket es un protocolo de comunicación bidireccional en tiempo real que se ejecuta sobre un único socket TCP. En nuestro caso hemos utilizado esta herramienta para construir un socket que se conectará con el frontend de nuestra aplicación para notificar en tiempo real a los usuarios que estén conectados en ese momento sobre cualquier actualización en los posts.

4- Testing.

4.1- Snapshot.

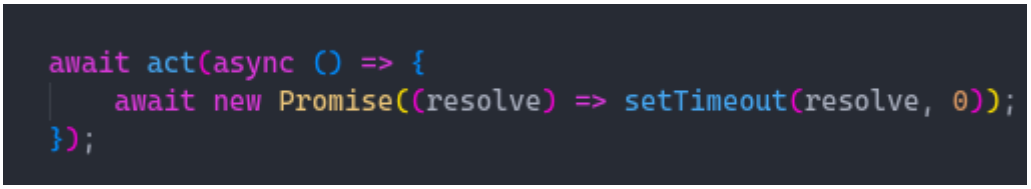
Los tests del tipo “Snapshot” son un tipo de pruebas de software que se utilizan para evaluar el estado o la condición de un sistema en un momento específico. Este tipo de pruebas son llamadas snapshot porque capturan una representación de la aplicación en un instante preciso, en lugar de evaluar su rendimiento a lo largo del tiempo.

Es una técnica para hacer pruebas en Jest enfocada en realizar una captura (snapshot) de aquellos componentes que deben ser renderizados. Para conseguirlo se realiza una simulación del comportamiento de los mismos mediante una serie de funciones ya definidas (en nuestro caso, por lo general, usaremos el framework Mock). A continuación se muestra un ejemplo:

```
jest.mock('../../../backend/postService', () => ({
    getNRates: jest.fn().mockResolvedValue({
      totalRates: 10,
      averageRate: 5,
    }),
}));
```

Como se puede apreciar en el ejemplo, se *mockea* la función “getNRates”, así se simula su comportamiento y devolverá datos simulados. En concreto se devolverá un objeto con un total de 10 rates y un average de 5.

Para este tipo de pruebas, también hemos usado ciertas funciones que nos permiten profundizar más en el comportamiento de los componentes. A continuación se muestra un ejemplo:



```
await act(async () => {
    await new Promise((resolve) => setTimeout(resolve, 0));
});
```

En este código se puede apreciar que se usa la función “act”, esta se usa para sincronizar las acciones que causan las actualizaciones en el árbol de componentes. React espera a que todas las actualizaciones de estado se completen antes de continuar con los tests.

De esta forma se intenta garantizar que todos los componentes se comportan de la forma esperada sin necesidad de tener que lanzar la aplicación.

Cabe destacar el procedimiento de la Snapshot. Primero se define una variable cuyo objetivo sea el de almacenar el JSON esperado de lo que estemos *testeando*. Una vez definida, se puede comparar con la snapshot que crea jest de forma automática una vez se introduce el comando (jest test).

En nuestra práctica, hemos necesitado para la mayoría de casos definir un token y una id de usuario ficticia para poder generar el JSON esperado. Para implementar esto, creamos una constante con dos strings simulando un token y una id de usuario.

```
const fakeLogginContext = {
    token: 'testToken',
    user: 'id'
};
```

Luego, haremos uso de “renderer” de la librería react-test-renderer para poder crear y luego comprobar el JSON esperado con el generado por jest, también se emplea el componente MemoryRouter para simular la navegación sin necesidad de un navegador real que envuelve el componente en cuestión. Este componente en particular necesitará un contexto adecuado, es decir, un contexto de login con token e id de usuario.

```
describe('Navbar component', () => {
  it('matches the snapshot', () => {
    const tree = renderer.create(
        <MemoryRouter>
          <LogginContext.Provider value={fakeLogginContext}>
            <Navbar />
          </LogginContext.Provider>
        </MemoryRouter>
    ).toJSON();
    expect(tree).toMatchSnapshot();
  });
```

4.2- Selenium.

Gracias a Selenium nos permitió automatizar el proceso de prueba de aplicaciones web al interactuar con ellas de la misma manera que lo haría un usuario. Esto incluye acciones como hacer clic en botones, completar formularios y navegar por páginas web.

Se trata de una interfaz que permite comunicar código con los navegadores web, WebDriver proporciona una API que permite usar ciertos navegadores. Para poder usarlo, hemo usado lo siguiente:

Primero, dentro del pom hay que introducir el plugin de failsafe.

```xml
<!-- FAILSAFE PLUGIN -->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-failsafe-plugin</artifactId>
    <version>3.2.2</version>
    <executions>
        <execution>
            <goals>
                <goal>integration-test</goal>
                <goal>verify</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

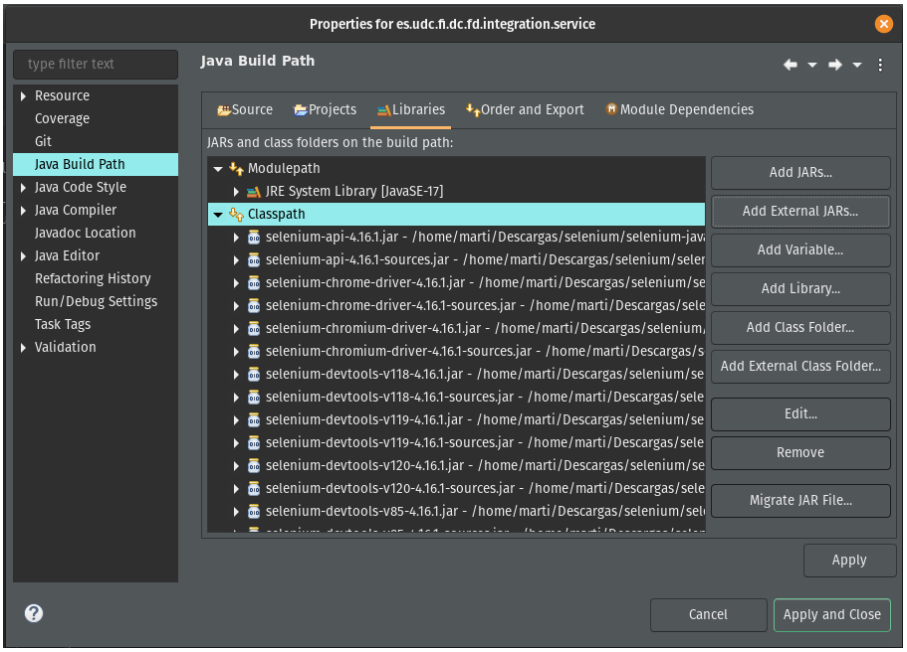
Una vez hecho eso, se hace lo mismo con cargo.

```xml
<!-- CARGO SETUP -->
<plugin>
    <groupId>org.codehaus.cargo</groupId>
    <artifactId>cargo-maven2-plugin</artifactId>
    <version>1.6.7</version>
    <configuration>
        <container>
            <containerId>tomcat8x</containerId>
            <zipUrlInstaller>
                <url>http://archive.apache.org/dist/tomcat/tomcat-8/v8.5.23/bin/apache-tomcat-8.5.23.zip</url>
                <downloadDir>${project.build.directory}/downloads</downloadDir>
                <extractDir>${project.build.directory}/extracts</extractDir>
            </zipUrlInstaller>
        </container>
        <configuration>
            <properties>
                <cargo.jvmargs>${jacoco.agent.itArgLine}</cargo.jvmargs>
                <cargo.servlet.port>7080</cargo.servlet.port>
                <cargo.tomcat.ajp.port>7093</cargo.tomcat.ajp.port>
            </properties>
        </configuration>
        <configuration>
            <properties>
                <cargo.server.settings>jonas1</cargo.server.settings>
            </properties>
        </configuration>
    </configuration>
</plugin>
```

Luego se introducen las dependencias de Selenium y algunas otras necesarias para su funcionamiento.

```xml
<!-- Selenium dependency -->
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
    <scope>test</scope>
</dependency>

<!-- Other dependencies -->
<dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>2.2.3</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
</dependency>
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-api</artifactId>
</dependency>
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-firefox-driver</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.maven.surefire</groupId>
    <artifactId>surefire-junit-platform</artifactId>
    <version>3.0.0</version>
</dependency>
```

Por último, hay que añadir dentro del package correspondiente a las pruebas, aquellos .jar relativos a Selenium cómo “External JARs”.



De esta forma, el proyecto queda configurado y se pueden realizar las pruebas pertinentes.

A continuación se muestra un ejemplo de un test de este tipo, el test en concreto es el que verifica el LoginForm.

Lo primero es definir los atributos necesarios:

```
private static final String BASE_URL = "http://localhost:7080/test-spring-mvc/";
private static WebDriver driver;
```

Luego se realizan las acciones que se llevarán a cabo para cada situación.

```
@BeforeAll
public static void setupClass() {
    WebDriverManager.firefoxdriver().setup();
}

@BeforeEach
public void setUpTest() {
    driver = new FirefoxDriver();
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
}

@AfterEach
public void tearDownTest() {
    driver.close();
}
```

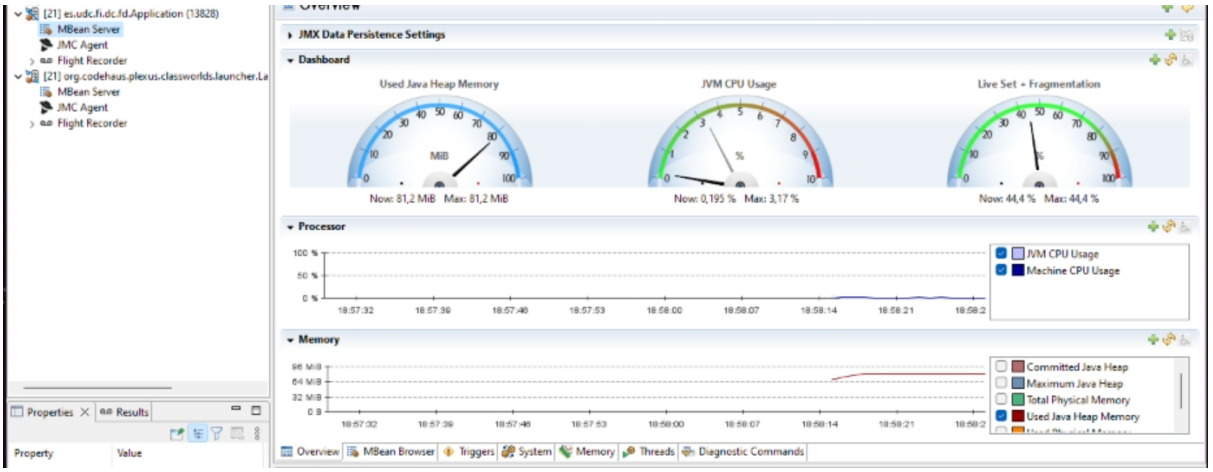
En nuestro caso, decidimos usar el navegador Firefox. Por último haremos el test correspondiente obteniendo los elementos a través de WebElement y *settearemos* los valores más adelante. Por último nos aseguramos de que el botón de submit correspondiente a este form se haya ejecutado correctamente.

```
@Test
public void testLoginForm() {
    driver.get(BASE_URL);
    WebElement usernameInput = driver.findElement(By.name("username"));
    WebElement passwordInput = driver.findElement(By.name("password"));
    WebElement loginButton = driver.findElement(By.xpath("//button[text()='Log in']"));

    assertNotNull(usernameInput);
    assertNotNull(passwordInput);
    assertNotNull(loginButton);

    usernameInput.sendKeys("username");
    passwordInput.sendKeys("password");
    loginButton.click();

    WebElement successMessage = driver.findElement(By.id("success-message"));
    assertNotNull(successMessage);
    assertEquals("Logeado correctamente.", successMessage.getText());

}
```

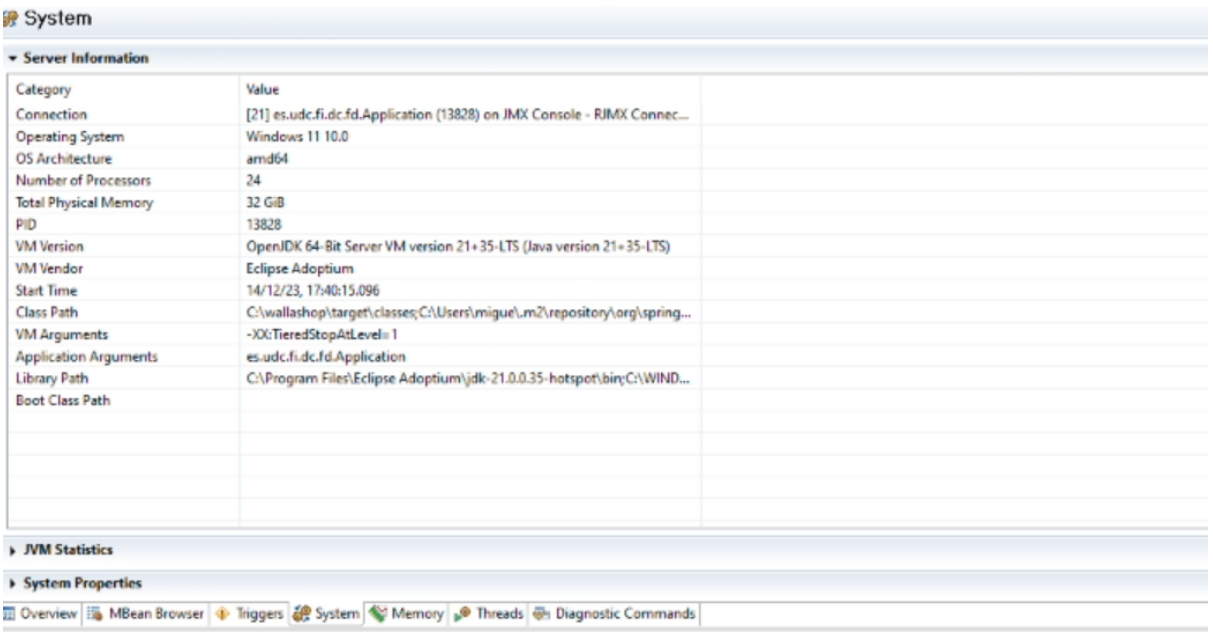
4.3- Profiling con JVM.

El profiling con la JVM se realiza mediante herramientas especializadas que se conectan a la máquina virtual y recopilan datos relevantes, como por ejemplo, el uso de memoria, el tiempo de ejecución de los métodos y la frecuencia de invocación de los métodos.

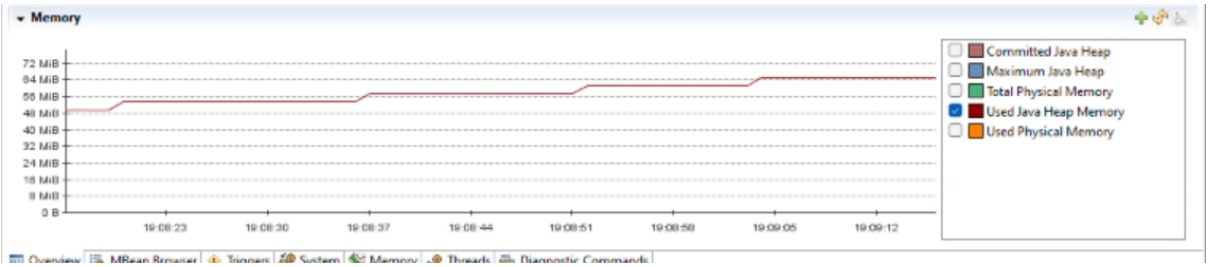
Este es el dashboard al que se accede después de pulsar en MBean:



Desde la pestaña de “system” podemos acceder a la información del server.



Desde esta view podemos ver el uso de memoria.

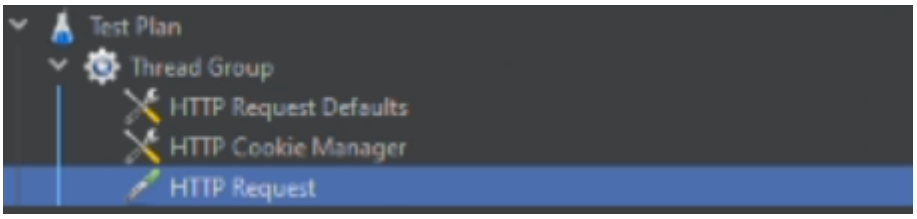


También podemos hacer un “flight recording” para grabar eventos detallados durante la ejecución de la aplicación. Esto nos proporciona información sobre el rendimiento, uso de recursos y otros aspectos del comportamiento de la aplicación. También una vez realizado, podemos acceder a la vista de memoria para identificar algún problema de memoria.

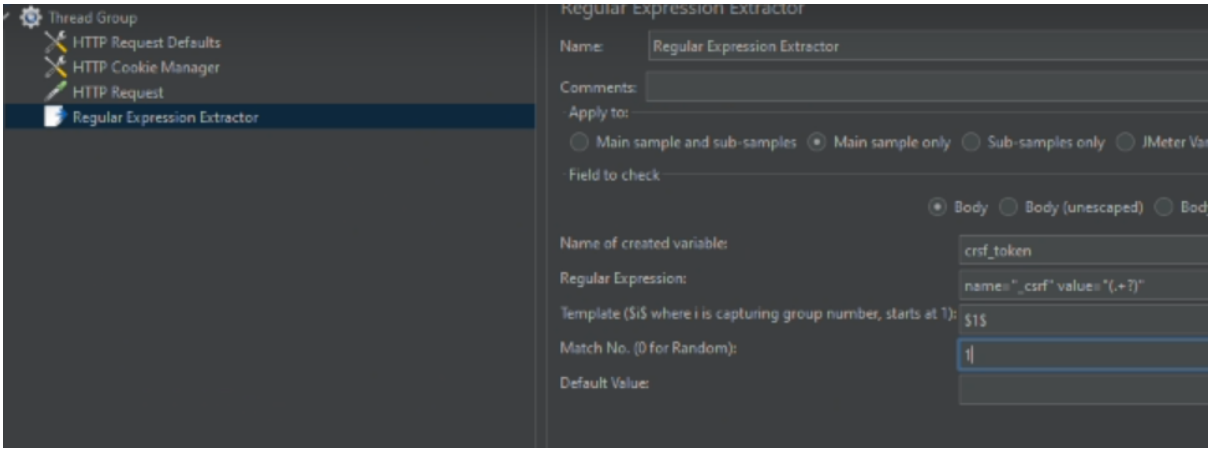
4.4- JMeter.

JMeter sirve para realizar pruebas de rendimiento, este tipo de pruebas son esenciales para evaluar la capacidad y la eficiencia de las aplicaciones web y servidores.

Para configurar JMeter primero creamos el Thread Group, y dentro de este creamos el “HTTP Request Default”, “HTTP Cookie Manager” y “HTTP Request”. En el “HTTP Request Default” indicamos el puerto, la ip y el path por defecto.



También para extraer los datos de la página web podemos usar un Post Processor como por ejemplo, el CSRF.



5- Lecciones aprendidas.

A lo largo de esta asignatura hemos visto y descubierto nuevas herramientas que nos han facilitado el desarrollo de la misma.

Lo primero que hay que destacar es el trabajo en grupo, ya que tuvimos que dividirnos las tareas y trabajar sin interferir en el trabajo de otros, todo esto mientras se seguía trabajando de manera eficiente.

Al desarrollar la práctica en Eclipse, descubrimos muchas vistas nuevas y otras formas de *debbugear* de manera mucho más cómoda, lo que facilita la detección de errores y la corrección de los mismos.

Gracias a Redmine, aprendimos sobre la gestión del proyecto, ya que pudimos planificar, estimar y organizar las “issues” que definimos para desarrollar la aplicación.

Con git, hemos aprendido a gestionar el repositorio ya que tuvimos que crear varias ramas y subramas para desarrollar todas las subtareas, también hemos aprendido a realizar correctamente los merge y resolver los conflictos.

Con la integración de herramientas como Jenkins hemos aprendido a automatizar las builds y gracias a esto pudimos detectar errores de manera más eficiente. Con Sonar descubrimos la importancia de realizar análisis estáticos de código para garantizar la calidad del mismo, esta herramienta también nos ayudó a encontrar vulnerabilidades o bugs en el proyecto.

También a lo largo del desarrollo de la práctica hemos realizado muchos tipos de pruebas y hemos aprendido a desarrollarlas para tratar cada parte de la aplicación, ya sea el rendimiento, los componentes, las funcionalidades…

6- Recomendaciones para el deployment.

En caso de que los posts no se rendericen la primera vez que ejecutes la aplicación tienes que refrescar la página y desloguearte en caso de que tengas un bearer en el local storage (no debería pasar si es la primera vez que ejecutas la aplicación)

A la hora de probar si funciona la notificación de crear o borrar un post, a veces el socket no responde bien y es necesario refrescar la página donde no se muestra la notificación. En nuestro caso hemos probado a registrarse e iniciar sesión en dos navegadores distintos y en casi todos los casos funciona correctamente. El flujo no funciona cuando haces esto en un mismo navegador usando su forma normal y otro usuario desde incógnito.

Los tests de Selenium daban un error por lo que no pudimos integrarlo a la hora de hacer el deployment.