

Ferramentas de Desenvolvimento

Desarrollo de una aplicación web

(2020-2021)

Memoria

MacroHard

Adrián Bueno Leiro

adrian.bueno.leiro@udc.es

Alén Brath Filgueira Caneiro

alen.filgueira@udc.es

Diego Varela Candal

diego.vcandal@udc.es

Jorge Casal Munín

jorge.munin@udc.es

Índice

1.Introducción	3
1.1. Herramientas principales	3
2. Metodología e iteraciones	4
3. Desarrollo de la práctica	5
3.1. Características generales, frameworks y librerías usadas	5
3.2 Eclipse	6
3.3 Control de versiones (GIT)	6
3.4 Gestión de incidencias	8
3.5 Construcción automática	11
3.6 Integración continua	13
3.7 Inspección continua	15
3.9.1 Pruebas de Rendimiento	17
3.9.2. Pruebas Funcionales	19
4. Errores y problemas en la práctica	20

1.Introducción

El objetivo de este proyecto fue construir una aplicación web para la gestión de una tienda online en la que se permitiera a los usuarios añadir anuncios, comprar y hablar entre usuarios, entre otros. Usando las herramientas y tecnologías vistas en teoría, aplicándolas a la práctica, usando sus funcionalidades de forma que nos permitan desarrollar un proyecto minimizando los errores, malas prácticas, y mejorando ampliamente la calidad de desarrollo.

1.1. Herramientas principales

Para la realización de esta práctica se han usado principalmente las siguientes herramientas:

Eclipse

Se utilizó Eclipse IDE, uno de los entornos de desarrollo integrado más usado, que ha sido usado para el desarrollo del software.

Maven

A la hora de la construcción del proyecto usamos Maven. Esta herramienta permite automatizar procesos de compilación, empaquetado, pruebas y despliegue de la app.

GitLab

El control de versiones que se utilizó para coordinar el trabajo en equipo fue GitLab.

RedMine

La gestión del proyecto se realizó a través de Redmine, esta permite la organización del equipo y el seguimiento de las tareas del proyecto.

Jenkins

Para realizar la integración continua usamos Jenkins con la cual nos servimos para hacer la integraciones automáticas del proyecto y detectar fallos lo antes posible.

JaCoCo

Esta librería comprueba la cobertura de los test. JaCoCo analiza el porcentaje de código cubierto por los tests que tengamos en nuestro proyecto.

SonarQube

Con respecto a la gestión de calidad en el código usamos la herramienta SonarQube. Esta herramienta está pensada para evaluar código fuente, analizándolo y detectando bugs potenciales, cobertura de test y código duplicado, entre otros.

JUnit

Los test unitarios se realizaron con JUnit Jupiter 5.5.2. Que nos permite el uso de los frameworks de test para realizar las pruebas unitarias.

Selenium

Selenium nos proporciona un entorno de pruebas software para aplicaciones basadas en la web.

Cargo

Cargo es un wrapper que nos permite levantar contenedores web (como tomcat), usado para los tests de aceptación.

2. Metodología e iteraciones

Para el desarrollo de la práctica se ha seguido una metodología de trabajo basada en un desarrollo incremental mediante una serie de iteraciones. En cada una de ellas se presentaban una serie de requisitos que serían los objetivos a cumplir. Estos requisitos se dividieron entre todos los miembros del equipo al principio de cada iteración.

Estos requisitos se nos iban indicando cada 4 semanas.

En la primera iteración creamos el arquetipo de la práctica e hicimos los siguientes casos de uso:

- Crear un perfil de usuario
- LogIn a la aplicación
- Publicar un anuncio
- Visualizar anuncios

En esta primera parte al no tener un amplio conocimiento de git y de otras herramientas como pueden ser Redmine o Sonar, nos centramos principalmente en el IDE Eclipse en cómo lanzar la aplicación, generar código y otras muchas ventajas que nos proporciona este IDE. Trabajamos sobre la rama master así que la división y organización del trabajo no fue demasiado buena.

En la segunda iteración, añadimos los siguientes casos de uso:

- Buscar productos por unos criterios
- Darle me gusta y no me gusta a los anuncios
- Mostrar los anuncios que te han gustado
- Borrar anuncios
- Poner anuncios en espera
- Seguir usuarios

En esta segunda iteración además de Eclipse aprendimos a usar el issue tracker Redmine y mejoramos nuestro conocimiento sobre git. Aquí ya dividimos mejor el trabajo con una mejor organización, usando Git Flow junto a Redmine.

En la tercera iteración, incluimos los siguientes casos de uso:

- Crear un chat con los vendedores
- Valorar un vendedor
- Mostrar la calificación promedio de un vendedor
- Buscar por calificación mínima
- Ser premium
- Posicionar anuncios premium
- Comprar un producto
- Añadir etiqueta de vendido a un anuncio

En este punto a mayores de lo anterior usamos Jenkins para integrar la aplicación regularmente y SonarQube para comprobar la calidad de nuestro código. Esto nos ayudó a darnos cuenta de que teníamos bastante código duplicado, una serie de bugs a corregir y que debíamos hacer muchos más test por la baja cobertura.

En la cuarta y última iteración añadimos:

- Una API Rest
- Test de rendimiento
- Redactar la memoria

Para ello utilizamos todo lo anterior y JMeter para las pruebas de rendimiento.

3. Desarrollo de la práctica

En este apartado se explican aspectos generales de la práctica, y como ha sido desarrollada en función de todas las herramientas y tecnologías que se han visto en la teoría.

3.1. Características generales, frameworks y librerías usadas

La práctica ha sido desarrollada en la parte servidor con Java EE, y la parte web en HTML5, CSS y JavaScript. Esta práctica sigue un arquetipo que ya incorpora ya una serie de tecnologías, algunas se explicaran a continuación:

Spring MVC

Spring es un framework para el desarrollo de aplicaciones, de código abierto para la plataforma Java. Para esta práctica se usa su versión MVC.

JPA

El API de persistencia desarrollada para Java EE, facilitandonos la automatización y gestión de esta parte de forma sencilla.

Bootstrap

Es un framework web o conjunto de herramientas de código abierto para el diseño de aplicaciones web, contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales.

Thymeleaf

Thymeleaf es una librería Java que implementa un motor de plantillas de qué puede ser utilizado tanto en modo web como en otros entornos no web. Se acopla muy bien para trabajar en la capa vista del MVC de aplicaciones web.

Liquibase

Liquibase es una biblioteca de código abierto independiente de la base de datos para rastrear, administrar y aplicar cambios en el esquema de la base de datos. Se usa para gestionar el versionado de la BD.

3.2 Eclipse

Como se explicó al principio, para el desarrollo de la práctica usamos Eclipse, que nos proporciona una serie de plugins y funcionalidades que nos ayudarán ampliamente en el desarrollo, como se verá en esta memoria.

Eclipse nos proporciona una serie de vistas y perspectivas, que se podrán usar y personalizar. Para esta práctica hemos tenido que aprender a usar correctamente, por ejemplo, la perspectiva de debug, fundamental para poder solucionar de forma óptima y rápida los bugs que nos hemos encontrado en el desarrollo.

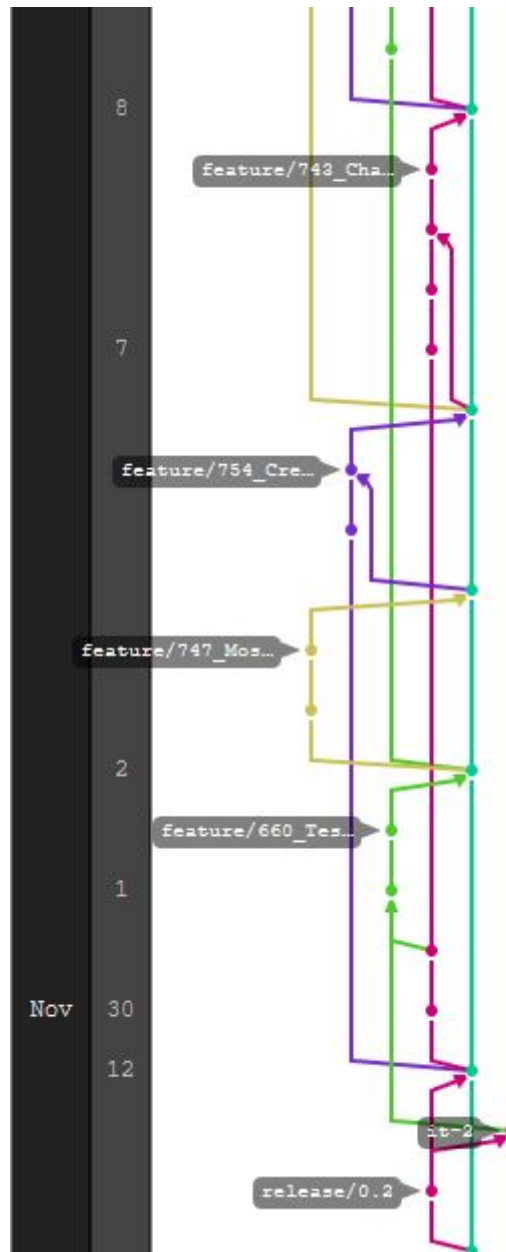
También presenta otras opciones útiles, como el control de versiones local, o el propio control de versiones de Git, integrado en su propia perspectiva. Además de los plugins para la integración con herramientas que veremos a continuación, como Redmine.

3.3 Control de versiones (GIT)

Para el control de versiones, en esta práctica se ha usado Git. En concreto hemos usado el flujo de trabajo de Git Flow, que define un modelo por ramas, que nos facilita la división y organización del repositorio en funciones de las iteraciones y funcionalidades que se van a desarrollar.

Todas las funcionalidades se desarrollan en ramas feature, nombradas como feature/xxx_Caso_de_uso. Estas funcionalidades estarán asignadas a un miembro del equipo de desarrollo, por lo que solo un miembro trabaja en dicha rama. Todas estas ramas siempre se crean de la rama develop. Y cuando la funcionalidad se termina, se realiza el merge en develop.

En la rama develop es donde se ubica el código para la siguiente versión. Cuando tienen las suficientes características, este código se traslada a la rama release correspondiente, por ejemplo release/0.3, donde se le aplicarán los últimos cambios antes de incorporarla a la rama master, que contendrá únicamente las versiones estables del proyecto.



Ramas del proyecto en git

Otra utilidad importante son los Merge Request. Estos han sido usados para solicitar el merge de una rama feature en la rama principal develop. Gracias a esto un desarrollador (el maintainer) asignado podrá revisar los cambios que se han hecho, y poder discutirlos, proponer cambios, etc. De forma que el merge solo se realice una vez que se tenga la aprobación del desarrollador asignado.

3.4 Gestión de incidencias

Para la gestión de incidencias usamos Redmine como Issue Tracker. Vinculamos Redmine con git de manera que cuando creamos una rama en el git le asignamos una tarea ya creada en Redmine. Para esto primero obtenemos el identificador de la tarea ya creada en git, y esta servirá también como identificación de las ramas feature en la convención de nombrado que seguíamos.

Además de esto, cada commit se enlazaba mediante unas palabras claves a la tareas correspondientes, que podrán variar en función de si es un error, se implementa una feature, o simplemente queremos hacer referencia a una tarea, además de añadir la cantidad de tiempo empleada en resolverlas

Redmine nos ayudó también a la hora de repartir el trabajo entre el equipo al dividir el proyecto en tareas.

#	Tracker	Status	Priority	Subject	Assignee	Updated	
778	Feature	Closed	Normal	Caso de uso: Comprar productos	Adrián Bueno Leiro	12/10/2020 06:26 PM	...
777	Feature	Closed	Normal	Caso de uso: Implementar API Rest para las búsquedas de anuncios	Diego Varela Candal	01/11/2021 07:20 PM	...
776	Feature	Closed	Normal	Caso de uso: Mostrar compras	Adrián Bueno Leiro	12/10/2020 07:07 PM	...
755	Feature	Closed	Normal	Caso de uso: Posicionar anuncios premium	Jorge Casal Munín	12/10/2020 06:24 PM	...
754	Feature	Closed	Normal	Caso de uso: Crear usuarios premium	Jorge Casal Munín	12/08/2020 11:00 AM	...
748	Feature	Closed	Normal	Caso de uso: Búsqueda por calificación mínima	Alén Brath Filgueira Caneiro	12/10/2020 07:14 PM	...
747	Feature	Closed	Normal	Caso de uso: Mostrar calificación promedio vendedor	Alén Brath Filgueira Caneiro	12/09/2020 07:41 PM	...
745	Bug	Closed	High	Bug: Arreglar encoding en la búsqueda	Diego Varela Candal	12/08/2020 05:27 PM	...
744	Feature	Closed	Normal	Caso de uso: Valorar vendedores	Diego Varela Candal	12/10/2020 03:00 AM	...
743	Feature	Closed	Normal	Caso de uso: Chat con vendedores	Diego Varela Candal	12/08/2020 05:27 PM	...
680	Bug	Closed	High	Arreglar imagen en la visualización por defecto	Jorge Casal Munín	11/12/2020 07:24 PM	...
660	Feature	Closed	Normal	Test:Test de las tareas #522 y #523. Parte Favoritos	Alén Brath Filgueira Caneiro	12/05/2020 02:45 PM	...
641	Feature	Closed	Normal	Caso de uso: Seguir usuario	Adrián Bueno Leiro	11/12/2020 03:24 AM	...
636	Feature	Closed	Normal	Caso de uso: Eliminar anuncio	Adrián Bueno Leiro	11/09/2020 03:58 PM	...
628	Feature	Closed	Normal	Caso de uso: Ver anuncios de un usuario	Jorge Casal Munín	11/19/2020 04:01 PM	...
532	Feature	Closed	Normal	Caso de uso: Poner anuncio en espera	Jorge Casal Munín	11/11/2020 07:11 PM	...
531	Feature	Closed	Normal	Caso de uso: Búsqueda de anuncios	Diego Varela Candal	11/10/2020 09:10 PM	...
530	Feature	Closed	Normal	Caso de uso: Ver detalles anuncio	Diego Varela Candal	11/07/2020 06:24 PM	...
529	Bug	Closed	Normal	Arreglar tamaño de visualización de imágenes	Diego Varela Candal	11/07/2020 06:24 PM	...
523	Feature	Closed	Normal	Caso de uso: Listar anuncios favoritos de un usuario	Alén Brath Filgueira Caneiro	11/08/2020 11:32 PM	...
522	Feature	Closed	Normal	Caso de uso: Darle me gusta a los anuncios por parte de un usuario	Alén Brath Filgueira Caneiro	11/08/2020 11:32 PM	...

Caso de uso: Búsqueda de anuncios

Added by Diego Varela Candal 3 months ago. Updated 2 months ago.

Status:	Closed	Start date:	10/29/2020
Priority:	Normal	Due date:	11/09/2020
Assignee:	Diego Varela Candal	% Done:	<div><div></div></div> 100%
Category:	Funcionalidades	Estimated time:	8.00 h
Target version:	v0.2	Spent time:	4.62 h

Description

Se podrá buscar los anuncios por una serie de formas:

- Búsqueda por keywords
- Rango de fechas
- Rango de precio
- Ciudad

Además se podrán ordenar los anuncios encontrados por:

- Fecha ascendente
- Fecha descendente
- Más relevantes

Files

 WireframeSearchAds.png (9.01 KB)  Diego Varela Candal, 10/29/2020 12:43 AM 

Práctica FD - MacroHard

Redmine también ofrece varias funcionalidades para ver distintas estadísticas o gráficas del proyecto. Como algunas de las que se adjuntan a continuación:

Version	Issue	2020-10	2020-11	2020-12	2021-1	Total time
v0.3			0.38	46.44		46.82
	Feature-#776: Caso de uso: Mostrar compras			2.53		2.53
	Bug-#745: Bug: Arreglar encoding en la búsqueda		0.38			0.38
	Feature-#747: Caso de uso: Mostrar calificación promedio vendedor			1.33		1.33
	Feature-#754: Caso de uso: Crear usuarios premium			5.37		5.37
	Feature-#755: Caso de uso: Posicionar anuncios premium			7.52		7.52
	Feature-#778: Caso de uso: Comprar productos			8.50		8.50
	Feature-#744: Caso de uso: Valorar vendedores			6.20		6.20
	Feature-#743: Caso de uso: Chat con vendedores			9.99		9.99
	Feature-#748: Caso de uso: Búsqueda por calificación mínima			5.00		5.00
v0.2		0.88	40.05			40.93
	Feature-#530: Caso de uso: Ver detalles anuncio		4.38			4.38
	Feature-#532: Caso de uso: Poner anuncio en espera		6.83			6.83
	Feature-#531: Caso de uso: Búsqueda de anuncios		4.62			4.62
	Feature-#636: Caso de uso: Eliminar anuncio		5.50			5.50
	Bug-#529: Arreglar tamaño de visualización de imágenes	0.88				0.88
	Feature-#628: Caso de uso: Ver anuncios de un usuario		3.50			3.50
	Feature-#523: Caso de uso: Listar anuncios favoritos de un usuario		1.00			1.00
	Feature-#522: Caso de uso: Darle me gusta a los anuncios por parte de un usuario		8.22			8.22
	Feature-#641: Caso de uso: Seguir usuario		5.67			5.67
	Bug-#680: Arreglar imagen en la visualización por defecto		0.33			0.33
v0.4					6.70	6.70
	Feature-#777: Caso de uso: Implementar API Rest para las búsquedas de anuncios				6.70	6.70
[none]				1.33		1.33
	Feature-#660: Test:Test de las tareas #522 y #523. Parte Favoritos			1.33		1.33
Total time		0.88	40.43	47.77	6.70	95.79

Historial de las distintas tareas, clasificados por mes y versión

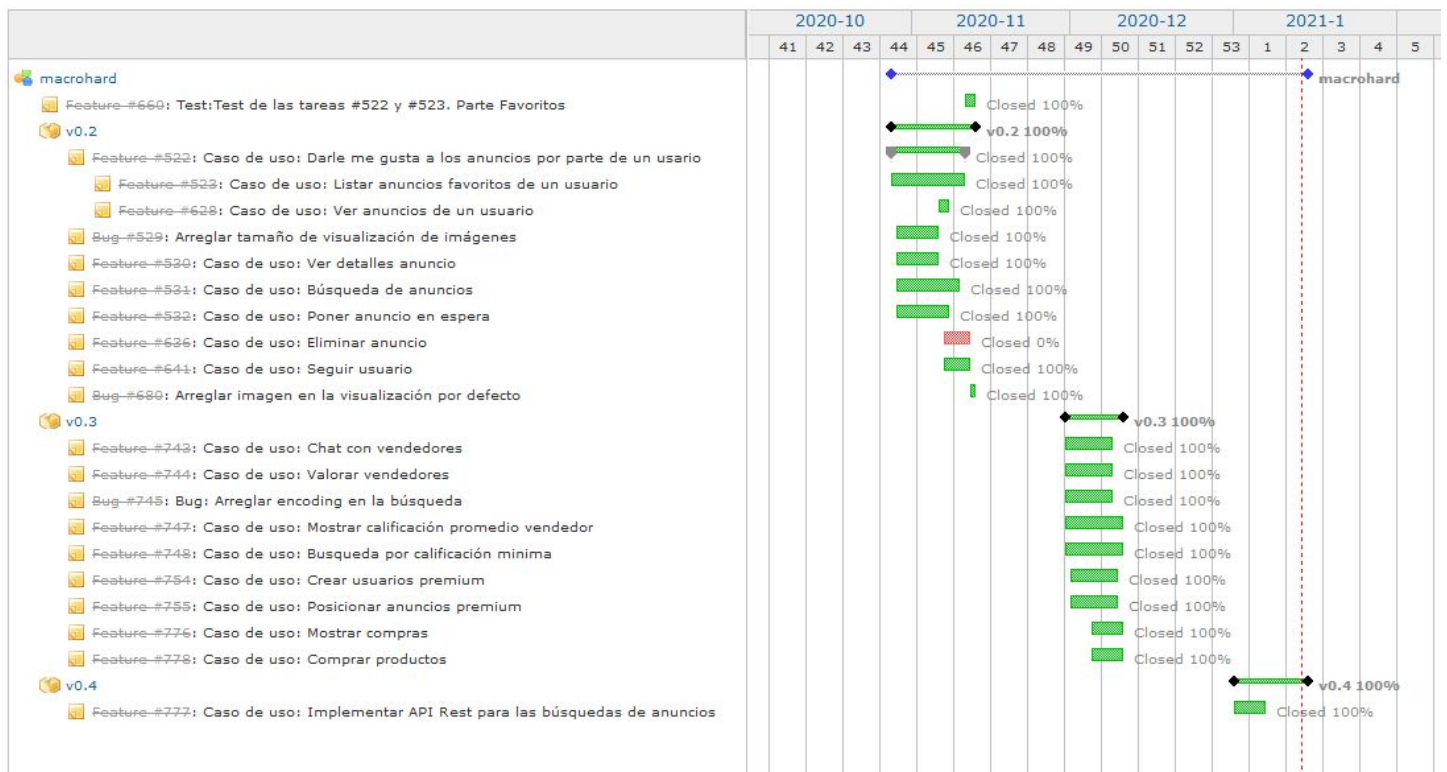


Diagrama de Gantt


Hemos hecho uso de la wiki incluida, para documentar mejor cada versión que se iba planificando.

Roadmap

 **v0.2**

11/12/2020

Segunda iteración del proyecto

 100%
10 issues (10 closed — 0 open)

Versión 0.2

Segunda iteración del proyecto. Funcionalidades añadidas:

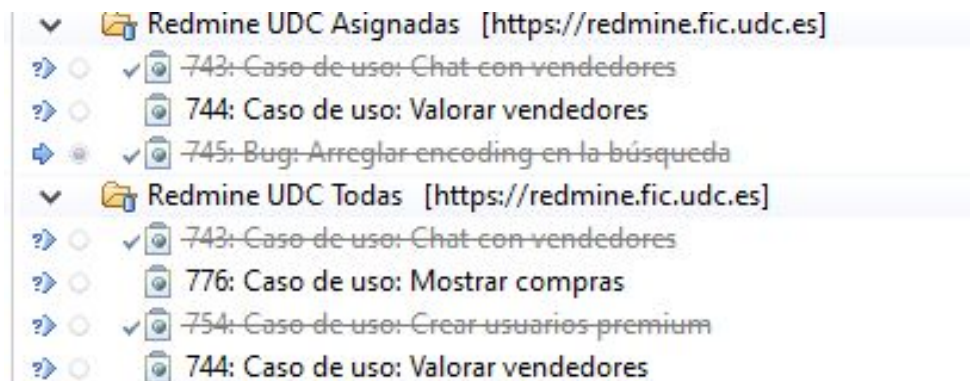
- Ver detalles de un anuncio
- Búsqueda de anuncios
- Botón 'Me gusta' para los anuncios
- Listar 'Me gusta' de un usuario
- Eliminar anuncios
- Seguir anunciantes
- Poner anuncios en espera

Related issues

Bug-#529:	Arreglar tamaño de visualización de imágenes
Bug-#680:	Arreglar imagen en la visualización por defecto
Feature-#522:	Caso de uso: Darle me gusta a los anuncios por parte de un usuario
Feature-#523:	Caso de uso: Listar anuncios favoritos de un usuario
Feature-#530:	Caso de uso: Ver detalles anuncio
Feature-#531:	Caso de uso: Búsqueda de anuncios
Feature-#532:	Caso de uso: Poner anuncio en espera
Feature-#628:	Caso de uso: Ver anuncios de un usuario
Feature-#636:	Caso de uso: Eliminar anuncio
Feature-#641:	Caso de uso: Seguir usuario

Segunda iteración del proyecto

Otra de las utilidades es la integración mediante plugins con Eclipse, con lo que podremos acceder a nuestras tareas asignadas, las que hemos realizado. Esto, junto a el plugin TimeKeeper, nos permite controlar de forma sencilla el tiempo que hemos empleado en cada tarea.



3.5 Construcción automática

En nuestra práctica usamos Maven como herramienta de automatización. Maven utiliza un pom.xml en el cual le indicamos entre otras cosas:

- Nombre y Versión
- Tipo de artefacto
- Localización del código fuente
- Dependencias
- Plugins de Maven
- Perfiles

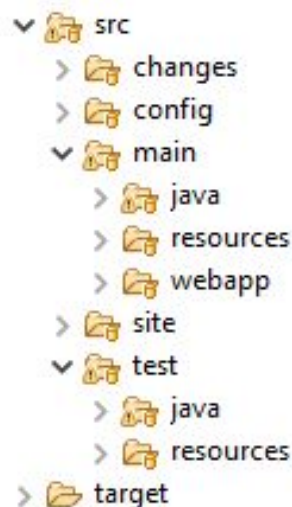
```
<groupId>es.udc.fi.dc.fd</groupId>
<artifactId>MacroHard</artifactId>
<version>0.3-SNAPSHOT</version>
<packaging>war</packaging>

<name>FD Spring MVC MacroHard App</name>
<description>Test project for FD 2020</description>
<url>https://gitlab.fic.udc.es/ferramentas.2020/MacroHard</url>
<inceptionYear>2020</inceptionYear>

<licenses>
  <license>
    <name>MIT License</name>
    <url>http://www.opensource.org/licenses/mit-license.php</url>
    <distribution>repo</distribution>
  </license>
</licenses>
```

Descripción del proyecto en el pom.xml

Nuestro proyecto tiene la estructura estándar de los proyectos Maven:



Estructura del proyecto

Y también sigue el ciclo de vida normal de un proyecto maven:

1. Validate
2. Compile
3. Test
4. Package
5. Integration-Test
6. Verify
7. Install
8. Deploy

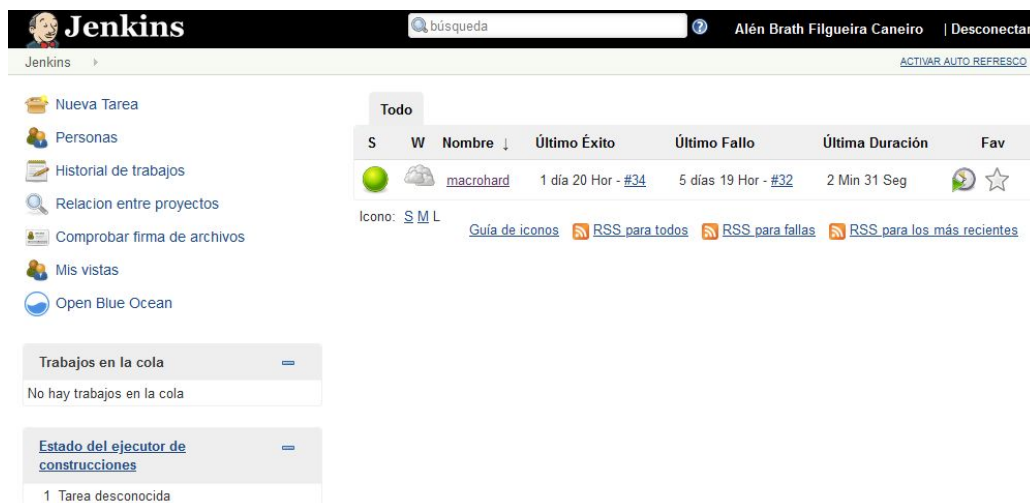
Para lanzar la aplicación usamos el goal **mvn jetty:run-war -P h2,db-properties,jetty** para poder utilizar jetty y el resto de perfiles con maven basta con modificar el pom.xml

3.6 Integración continua

Consiste en integrar de manera regular los cambios realizados en el proyecto. Para la integración continua utilizamos Jenkins. Esta herramienta de software libre está escrita en Java. Provee servicios de integración continua para el desarrollo de software de sistemas basados en servidores que se ejecuta en un servlet.

Como en Redmine, Jenkins está integrado con git, por lo que los cambios que se produzcan en el repositorio se verán reflejados aquí.

En este punto toca decidir en qué rama se hace la inspección continua. Localizamos tres ramas interesantes: Develop, Master y Release. Tomando la definición de integración continua decidimos configurar Jenkins de forma que en el momento que se añadiera una modificación a la rama Develop se ejecutará una nueva build de forma automática. En resumen controlamos la rama con más cambios de todo nuestro git flow.



Jenkins página principal

En la página de jenkins.fic.udc.es podemos visualizar nuestro proyecto en información de las builds generadas. El primer icono en la columna de la “S” es el estado del proyecto de la última ejecución y el segundo icono es el estado de las últimas 5 ejecuciones.

Dentro del proyecto podremos ver el historial de builds así como un menú de navegación a todas las aplicaciones integradas con jenkins. También veremos un gráfico con la tendencia de los resultados de las pruebas.

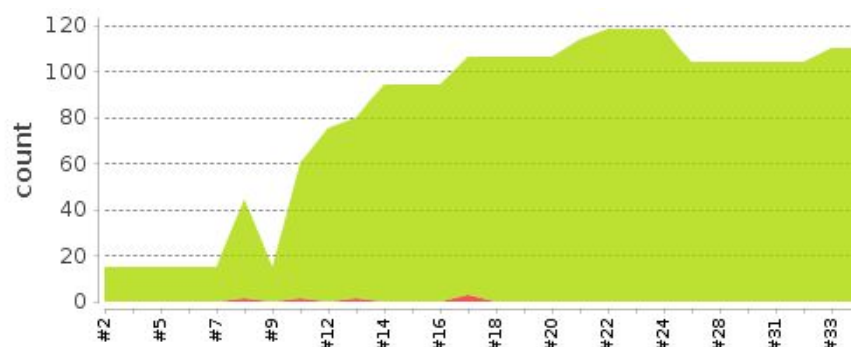


Gráfico pruebas

Historia de tareas		Tendencia
<input type="text" value="find"/>		x
 #34	11-ene-2021 20:11	
Started by GitLab push by Diego Varela Candal		
 #33	11-ene-2021 18:55	
Started by GitLab push by Jorge Casal Munín		
 #32	07-ene-2021 20:59	
 #31	07-ene-2021 20:25	
 #30	07-ene-2021 20:05	
 #29	16-dic-2020 17:31	

Historial de builds.

Cuando se accede a una build ejecutada podremos ver los cambios que se realizaron en ese commit que activó la ejecución, también se puede ver los resultados de los test, además de la salida de la consola.

Compilar FD Spring MVC MacroHard App (11-ene-2021 20:12:05)



Changes

1. Arreglado bug con el path de las imagenes en deploy ([details](#))

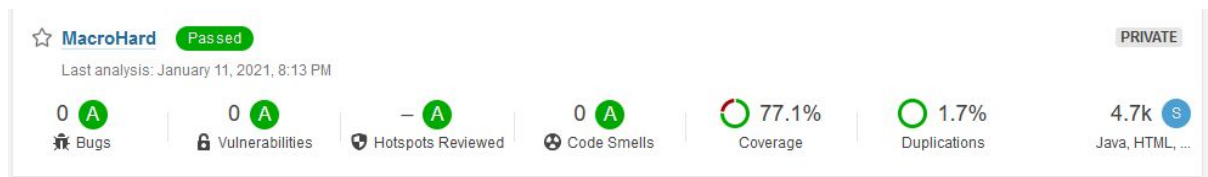


[Resultado de los tests](#) (Sin fallas)

3.7 Inspección continua

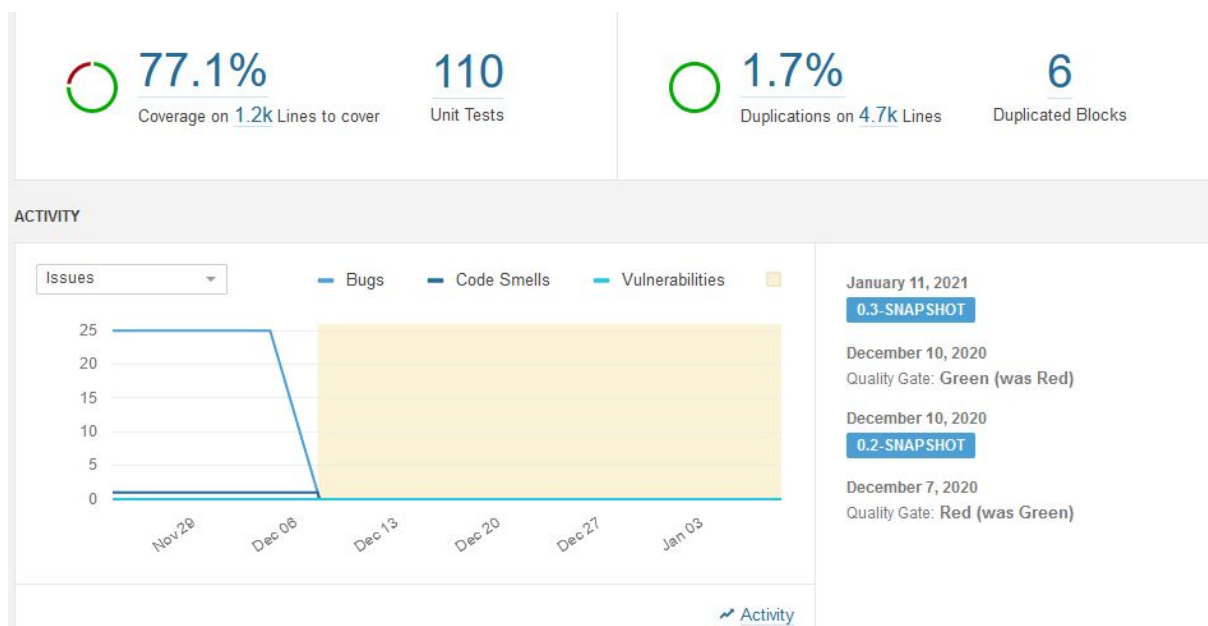
SonarQube es la herramienta utilizada en el desarrollo de la aplicación para emplear este paradigma. Es una herramienta que analiza el código desde diferentes perspectivas.

En este caso Sonar está vinculado con Jenkins, de forma que para cada build exitosa, se mostrarán los resultados actualizados en el proyecto.



Cuando abrimos <https://sonar.fic.udc.es/projects> tenemos la perspectiva resumida del proyecto donde se muestra el número de bugs, vulnerabilidades, la cobertura o las duplicaciones.

Dentro del proyecto tenemos varias pestañas por donde navegar. El resumen del estado del proyecto y issues son las pestañas más importantes donde nos centramos.



Resumen de la cobertura y duplicación con el grafo de actividad.

Sonar proporciona varias vistas de gran utilidad, donde hemos visto los problemas que se han encontrado en el código, como bugs, elementos duplicados, o poca cobertura.

Práctica FD - MacroHard

En esta vista vemos los bugs que han surgido, aunque algunos de ellos son reales y los hemos solucionado, otros los hemos catalogado como falsos positivos, que surgen de uso de herramientas como Thymeleaf.

src/main/webapp/WEB-INF/templates/403.html

Add a <title> tag to this page. Why is this an issue?

3 months ago ▾ L6 🔗 🔍

☐ Bug ☒ Major ☒ Resolved (False Positive) ▾ Not assigned Comment

 Diego Varela Can... A fragment with a full header is included, containing a <title> tag using thymeleaf.


No tags last month

src/main/webapp/WEB-INF/templates/404.html

Add a <title> tag to this page. Why is this an issue?

4 months ago ▾ L6 🔗 🔍

☐ Bug ☒ Major ☒ Resolved (False Positive) ▾ Not assigned Comment

 Diego Varela Can... A fragment with a full header is included, containing a <title> tag using thymeleaf.

No tags last month

src/main/webapp/WEB-INF/templates/ad_details.html

Add a <title> tag to this page. Why is this an issue?

2 months ago ▾ L6 🔗 🔍


☐ Bug ☒ Major ☒ Resolved (False Positive) ▾ Not assigned Comment

 Diego Varela Can... A fragment with a full header is included, containing a <title> tag using thymeleaf.







No tags last month

Otras vistas como en la que se nos muestran las duplicaciones de código:

MacroHard View as ☰ List ▾ ↑ ↓ to select files ← → to navigate 148 files

Duplicated Lines 122 

New Code: Since 0.2-SNAPSHOT

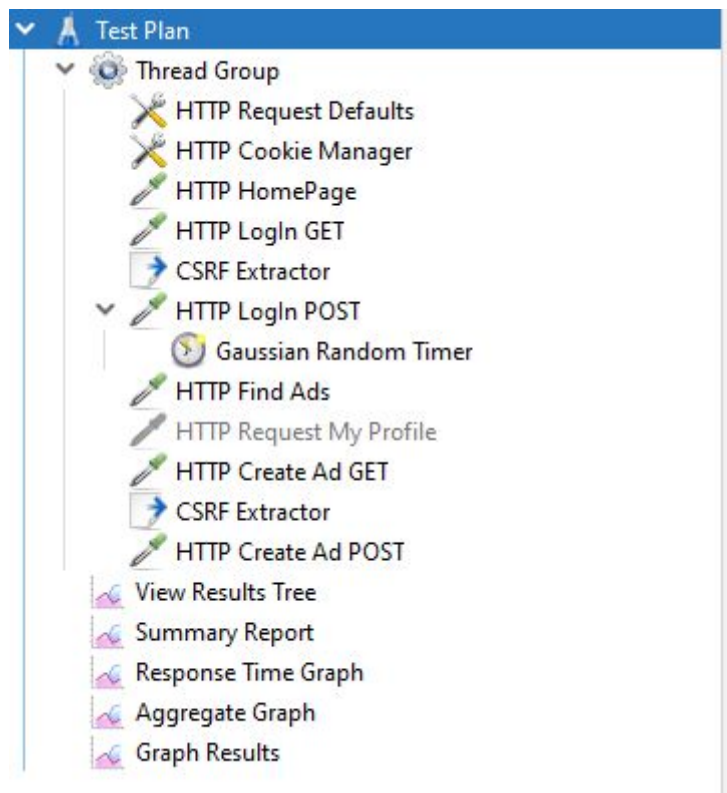
	Duplicated Lines	Duplicated Lines (%)
 src/main/webapp/WEB-INF/templates/fragments/ad_details.html	25	43.9%
 src/main/java/es/udc/fi/dc/fd/model/persistence/Ad.java	22	12.2%
 src/main/java/es/udc/fi/dc/fd/controller/dto/AdPreviewDto.java	22	12.1%
 src/main/webapp/WEB-INF/templates/ad_details.html	21	17.6%
 src/main/webapp/WEB-INF/templates/profile.html	16	12.5%
 src/main/webapp/WEB-INF/templates/rate_form.html	16	31.4%

Es importante tener en cuenta que aunque Sonar ofrece unas funcionalidades muy útiles para mejorar la calidad de nuestro software, es importante analizarlas, por ejemplo, por posibles falsos positivos como antes, o por coberturas de código altas, que no tienen por qué implicar que se están realizando buenos tests.

3.9.1 Pruebas de Rendimiento

Para hacer las pruebas de rendimiento utilizamos **Jmeter**. Es una herramienta utilizada para hacer test de carga y medir los tiempos de una petición en una app web. Genera una serie de informes y gráficos para poder comprobar las estadísticas típicas de respuesta, como la media de tiempo que tarda en cargar una petición o la desviación de tiempo de la misma. Para mostrar un ejemplo, hicimos ejecuciones con con thread groups de entre 500 y 1000 threads.

Un posible plan de ejecución es el siguiente:



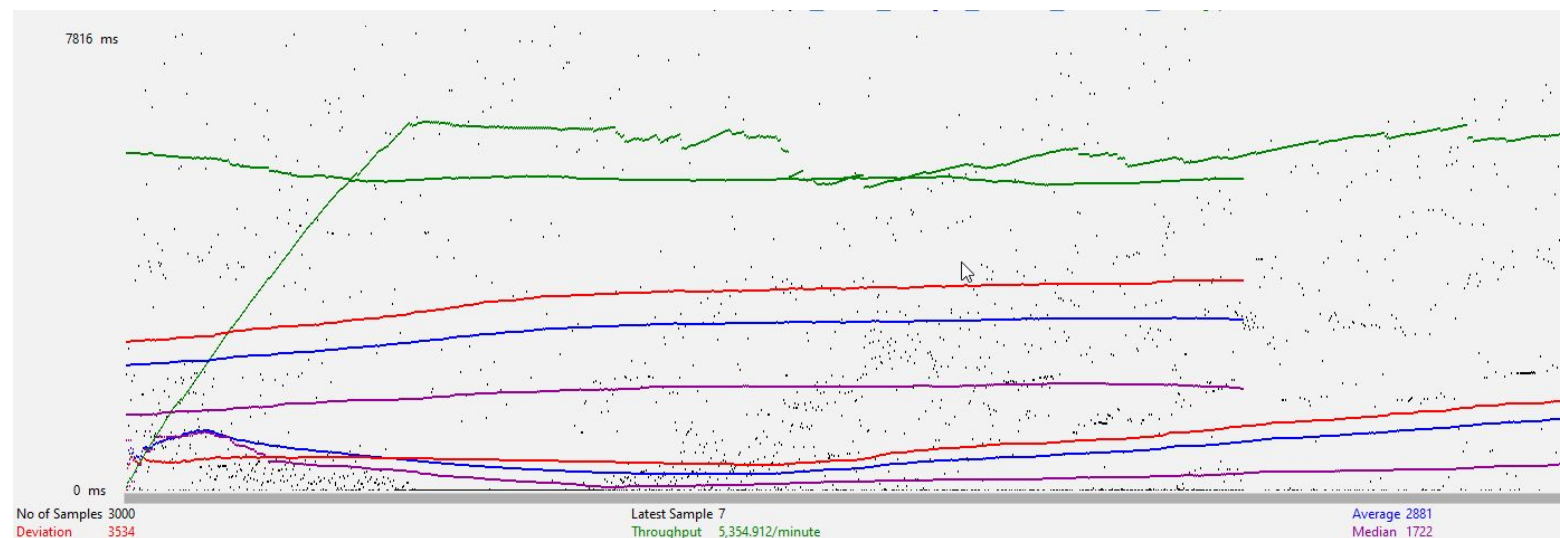
A parte de las configuraciones generales se incluyen las diferentes peticiones que se van a probar, como el Login o la creación de un anuncio. También se añaden listeners, donde al final de las pruebas podremos mirar los resultados. Por ejemplo, una posible ejecución del plan de pruebas anterior:

Label	# Samples	Average ↓	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Login POST	500	6520	143	21356	4137.98	0.00%	17.0/sec	5147.38	8.54	310826.1
HTTP Create Ad POST	500	4349	6	18940	4250.41	0.00%	18.2/sec	157.92	473.39	8892.1
HTTP Find Ads	500	3167	3	13794	2697.45	0.00%	17.0/sec	7783.94	3.51	469573.0
HTTP HomePage	500	1710	4	12861	1867.10	0.00%	22.5/sec	493.71	2.55	22495.1
HTTP Create Ad GET	500	858	1	6052	1443.80	0.00%	17.6/sec	126.24	3.16	7354.0
HTTP Login GET	500	685	1	4003	1083.70	0.00%	20.6/sec	107.94	3.63	5367.0
TOTAL	3000	2881	1	21356	3534.90	0.00%	89.2/sec	11976.90	404.75	137417.9

Tiempos en ms.

Nos muestra una serie de estadísticas de la ejecución, con los tiempos de media, el throughput (número de peticiones por unidad de tiempo), desviación media de los valores frente a la media, mínimos y máximos para cada petición así como el porcentaje de threads que fallaron.

Otro gráfico interesante es el siguiente (debido al límite de puntos que puede tener una línea en el gráfico algunas de ellas se resetean en cierto punto, y aparecen de nuevo al principio hasta su fin).



Aquí se reflejan los datos de antes en un gráfico, donde podemos apreciar mejor la evolución del test a lo largo del tiempo para esta situación de carga.

Tenemos que tener en cuenta también que los datos no son 100% fiables debido a que estamos lanzando la aplicación desde el mismo ordenador donde estamos lanzando el jmeter. Para tener la máxima confianza deberíamos lanzar la aplicación en un servidor y obtener los datos desde una máquina externa.

3.9.2. Pruebas Funcionales

Cada vez que se implementa una nueva funcionalidad o se introducía una nueva unidad de código (Controlador, Entidad, Repositorio, ...) se crea un caso de prueba para comprobar el correcto comportamiento del código.

Las pruebas unitarias en este proyecto se realizaron con JUnit5. JUnit es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

Además de JUnit se utilizó mockito. Mockito es una herramienta de testing con la que podemos simular comportamientos de clases que o aún no han sido implementadas, o que nos nos interesa usar en un momento, como cuando no nos interesa tener el comportamiento real de unos servicios para realizar un test unitario de un controlador, a esto se le llama clases mock.

A partir de estos test se genera la cobertura de código (que después se refleja en sonar), pero que ya podemos comprobar a medida que se realizan los test con el plugin de JaCoCo, que muestra la cobertura de la siguiente forma:

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
MacroHard	90,9 %	9.703	973	10.676
src/test/java	95,1 %	5.900	307	6.207
src/main/java	85,1 %	3.803	666	4.469
es.udc.fi.dc.fd.controller.rest	100,0 %	152	0	152
es.udc.fi.dc.fd.controller.util	95,6 %	263	12	275
es.udc.fi.dc.fd.model.persistence	93,0 %	583	44	627
es.udc.fi.dc.fd.controller.dto	90,4 %	412	44	456
es.udc.fi.dc.fd.controller.entity	90,2 %	1.476	160	1.636
es.udc.fi.dc.fd.service	84,2 %	632	119	751
es.udc.fi.dc.fd.model.exceptions	72,7 %	32	12	44
es.udc.fi.dc.fd.repository	62,4 %	186	112	298
es.udc.fi.dc.fd.controller	38,0 %	54	88	142
es.udc.fi.dc.fd.controller.exception	37,5 %	3	5	8
es.udc.fi.dc.fd.controller.error	25,6 %	10	29	39
es.udc.fi.dc.fd.model.form	0,0 %	0	41	41

4. Errores y problemas en la práctica

En la última iteración, para el deploy, la funcionalidad de chats no funciona. Se ha intentado solucionarlo pero no hemos dado con el origen del error. Aún así esta funcionalidad sí que sigue funcionando de forma correcta en la ejecución local.