SCROLLAD

Informe de la práctica para la asignatura

de Herramientas de Desarrollo

Nombre Integrante	DNI	Login
Diego Codesido Iglesias	45907274X	diego.codesido@udc.es
Antón Valladares Poncela	45906005Y	antón.valladares@udc.es
Laura Lestón Otero	58002900M	laura.leston1@udc.es
Beltrán José Aceves Gil	17463695W	beltran.aceves@udc.es
Carlos Fabián Omar Serrano	49331639C	fabian.serrano@udc.es

Índice

1. Descripción del Proyecto

La temática de esta práctica es el uso eficiente y correcto de las herramientas software para el desarrollo de una aplicación web, usando un arquetipo predefinido que consta de Spring MVC como back-end, Thymeleaf como front-end y Maven como herramienta de automatización.

La temática de la aplicación es una página de compraventa de productos a través de la publicación de anuncios para publicitarlos, entre las funcionalidades de esta aplicación uno puede: subir anuncios con sus correspondientes campos (imágenes, descripción, título, …), seguir usuarios, buscar anuncios utilizando una serie de filtros, mantener conversaciones con otros usuarios, entre otros.

2. Funcionalidades

Las funcionalidades de la aplicación son las siguientes:

- Registro de un usuario: usuario, contraseña, nombre, apellidos, y ciudad.
- Autenticación por credenciales: usuario y contraseña.
- Subida de anuncios con sus correspondientes campos.
- Visualización de la lista de anuncios subidos por los distintos usuarios.
- Búsqueda de anuncios, pudiendo aplicar una serie de filtros: ciudad, fecha, …
- Dar “Like” a los distintos anuncios presentes en la lista.
- Visualizar en una vista los anuncios que le gustan al usuario.
- Puntuar un anuncio.
- Seguir un anuncio.
- Visualizar en una vista los anuncios que sigue el usuario.
- Comprar los productos asociados a un anuncio.
- Visualizar en una vista los pedidos realizados.
- Mejorar el perfil de usuario a premium.
- Chat para poder conversar con los distintos usuarios.

3. Herramientas Empleadas

Breve resumen de todas las herramientas empleadas:

- Eclipse:IDE

  Para el desarrollo de código software, mayoritariamente código Java, aunque permite otros lenguajes utilizando plugins. Es ampliamente utilizado debido a la fácil integración de servicios: conexión con bases de datos, gestores de proyectos, gestores Git, etc.

- Maven

  Herramienta de automatización de proyectos. Se utiliza mayoritariamente para proyectos Java, aunque permite otros. Mvn gestiona principalmente cómo se construye un proyecto software y sus dependencias. Mediante un archivo xml define cómo está construido el proyecto, dependencias que usa, desarrolladores y más información.

- GIT

  Sistema de control de versión distribuido para el desarrollo de proyectos software. Permite desarrollo distribuido entre programadores, desarrollo no lineal, buena eficiencia para grandes proyectos y compatibilidad con protocolos como HTTP o FTP.

- Jetty

  Se trata de un servidor Java HTTP capaz de proporcionar contenido estático y dinámico. Se ha concebido para un rendimiento escalable ante un gran número de conexiones.

- Tomcat

  Se trata de un contenedor web con soporte de servlets y JSps. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web de apache, como en el caso de nuestra práctica.

- Jenkins

  Se trata de un servidor automatizado escrito en Java. Facilita la integración continua, vital para proyectos grandes y mejora los aspectos técnicos de la entrega continua. Soporta Git como herramienta de control de versiones y es compatible con proyectos que usen Maven.

- Sonar

  Se trata de una herramienta para la inspección continua de código. Utiliza una serie de métricas que aportan información sobre el código, por ejemplo, código duplicado, vulnerabilidades, bug´s, etc.

- Spring MVC

  Se trata de un framework Java para el desarrollo de aplicaciones web. Sigue el patrón MVC (Model-View-Controller). La capa vista se encarga de la parte de la aplicación, la capa controller de la lógica de negocio y la modelo se encarga de la información. Por información se puede entender una clase Java.

- Tymeleaf

  Se trata de una biblioteca Java que implementa un motor de plantillas HTML. Funciona de manera eficiente como la capa vista de un patrón de diseño MVC. Soporta muy bien la internacionalización.

- Hibernate

  Se trata de una herramienta de mapeado objeto-relacional para Java. Permite mapear entidades y atributos Java a tablas y atributos de bases de datos. También proporciona métodos para recuperación de información. La mayor ventaja es la generación de llamadas Sql automáticas eliminando esta responsabilidad del programador.

- Failsafe

  Se trata de un plugin de Maven que ejecuta las pruebas de integración durante la fase de pruebas del ciclo de vida de una aplicación. Se utiliza en las fases de integration-test y verify de Maven. También genera archivos .txt y .xml.

- Mockito

  Se trata de un framework Java para pruebas. Permite la creación de "mockitos", objetos mockito, que se utilizan en tests unitarios evitando así el acceso a base de datos.

4. Arquitectura Global

- **scrollad**
  - **src/main/java**: Este paquete contiene todas las clases, servicios, controllers y dto's que dan funcionalidad a la aplicación. Este se divide de la siguiente forma:
    - **es.udc.fi.dc.fd.controller.***: Paquete que contiene los controladores tanto de las entidades.
    - **es.udc.fi.dc.fd.model.*:** Paquetes que contiene las clases Java correspondientes al modelo.
    - **es.udc.fi.dc.fd.repository:** Paquete que contiene las entidades Java presentes en la aplicación.
    - **es.udc.fi.dc.fd.service.*:** Paquete que contiene los servicios Java.
  - **src/main/resources**: Este paquete contiene archivos de configuración para hibernate, thymeleaf, bases de datos (h2), archivos de propiedades, etc
  - **src/test/java**: Este paquete contiene los tests con los que se probarán tanto las clases, como los controladores y los servicios.

5. GIT

El proyecto se ha almacenado en un repositorio git facilitado por la facultad (https://gitlab.fic.udc.es/ferramentas.2020/scrollad). La manera de trabajar del equipo de desarrollo en Git ha sido la siguiente:

- Rama Develop

  Cada commit de esta rama describe una funcionalidad de la aplicación. Siempre que una funcionalidad estaba terminada y probada, se incluía en develop.

- Rama Master

  Al inicio del proyecto, esta rama era la única utilizada por falta de conocimientos para desarrollo. Posteriormente cada commit en ella ha representado un entregable en la aplicación que el cliente puede utilizar y esta es la principal diferencia con release.

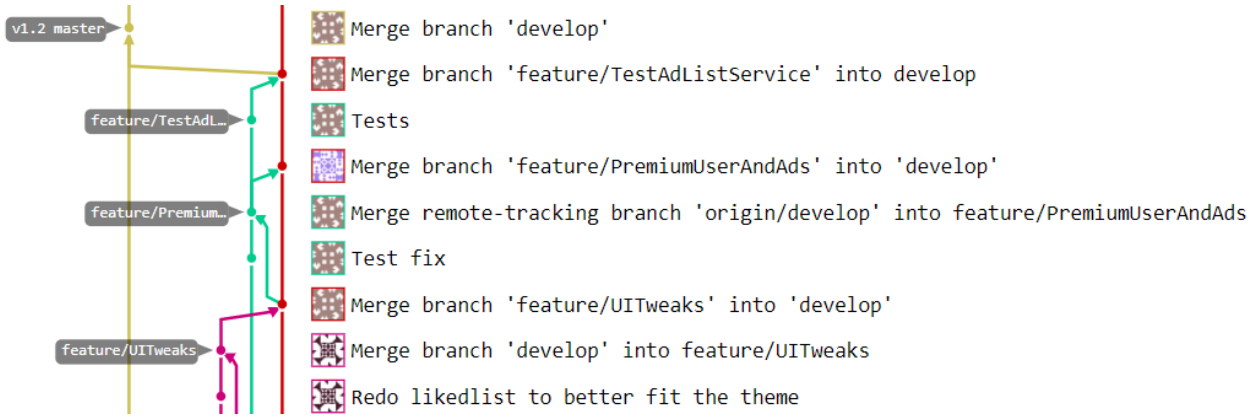
- Rama Release

  Cada entrega que fijaba el cliente se corresponde con un commit de esta rama.

- Ramas Feature

  Cada commit representa una funcionalidad que el cliente ha proporcionado o ha solicitado para la aplicación. Debido a la complejidad de las funcionalidades cada miembro del equipo ha desarrollado una funcionalidad por completo. Una vez terminada y probada, se desplegaba a la rama develop.

  Cada una de las veces que se realizó una iteración de la práctica, se marcó un tag especificando la versión en la que se encontraba la aplicación. En este gráfico se muestra como una vez terminadas las features, estas pasan a la rama develop para posteriormente esta ser incluida en release y master.

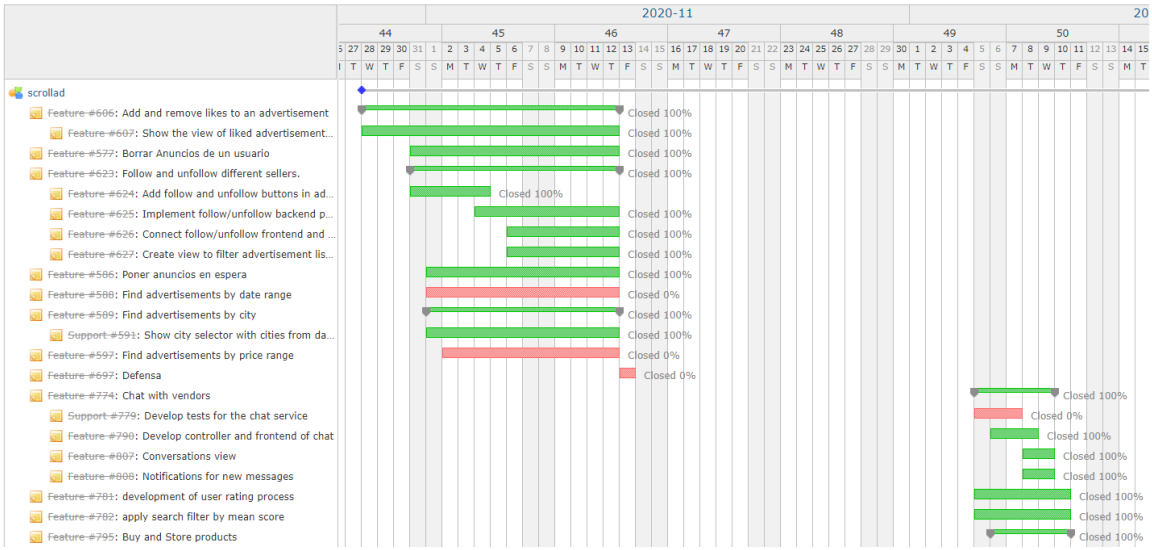


6. Redmine

El proyecto se ha dividido en iteraciones marcadas por el cliente. Cada iteración con su fecha de entrega ha tenido una serie de tareas que han sido llevadas a cabo satisfactoriamente por el equipo de desarrollo. En el inicio de cada iteración el equipo de desarrollo se ha reunido para la creación de las tareas.

Cada una de las tareas o "features" ha tenido una serie de datos asociados: miembro asignado, tiempo estimado, fecha estimada y estado, entre otros valores. Para cada tarea, el miembro del equipo ha imputado las horas dedicadas al trabajo de esta con un comentario identificativo sobre el progreso. Además, al inicio de cada tarea el responsable de esta ha fijado la fecha de inicio. Una vez finalizada una tarea, se ha marcado como resuelta y con un porcentaje asignado al cien por cien.

A continuación, se muestra un ejemplo de la métrica proporcionada por la herramienta en cuestión. Este informe resume la duración empleada para la realización de las distintas de las features que componen nuestro proyecto.



7. Maven

El arquetipo proporcionado para la realización de la práctica ofrece un pom funcional al que se le han ido incorporando dependencias, plugins y profiles durante el transcurso de las iteraciones. Entre los profiles más destacados tendremos los de H2 el cual constituye una base de datos en memoria. Por otro lado entre las dependencias introducidas en el pom a lo largo de las iteraciones, cabe destacar:

- **Spring Security** para la gestión de usuarios.
- Dependencias de **Thymeleaf** extras para poder tener más flexibilidad a la hora de crear plantillas.
- Versiones más recientes de **bootsrap** y **jquery**.

8. Pruebas

Dentro del apartado de pruebas distinguimos tres tipos:

- Unitarias

  Para las pruebas unitarias (o de unidad) se han utilizado dos herramientas: Junit y el plugin de Jacoco para el informe de cobertura. Jacoco genera un informe que posteriormente utiliza Sonar, la herramienta utilizada para la inspección continua.

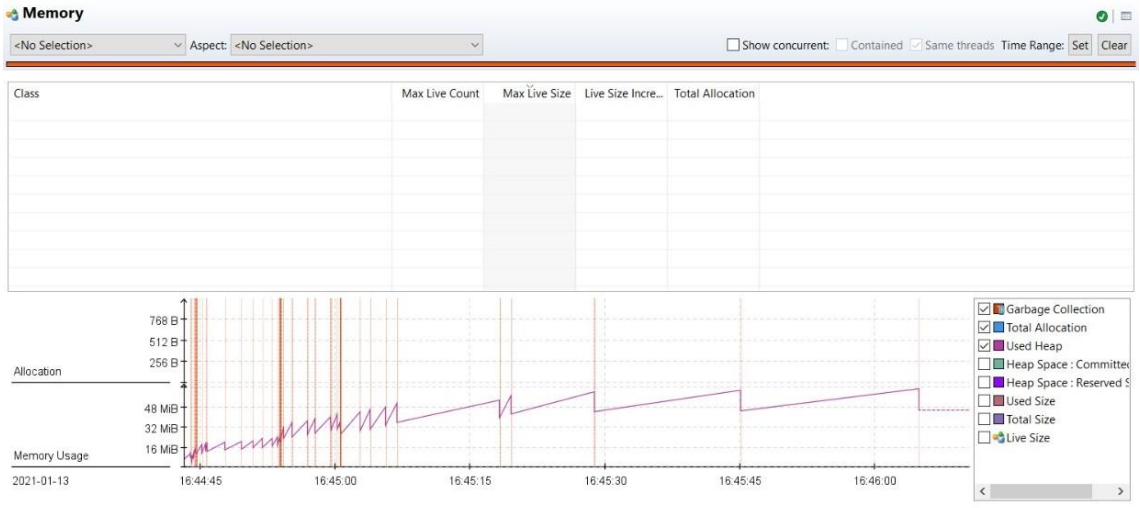
- De Integración

  Para las pruebas de integración se ha utilizado la inyección de dependencias en servicios mediante etiquetas de Spring Framework (@Autowired), además del uso de JUnit 4 para aserciones. Para la ejecución de este tipo de pruebas se ha utilizado el plugin de Maven-failsafe. Una vez probados la mayoría de los servicios, Jacoco analiza la cobertura para que posteriormente Sonar utilice los resultados.

- De Rendimiento

  Para las pruebas de rendimiento se han utilizado las herramientas Java MissionControl, para observar la gestión de la memoria,VisualVM y Eclipse MAT para detectar memory leaks, y JMeter para medir los tiempos de respuesta. Esta última permite lanzar un número predefinido de peticiones contra un recurso de la aplicación.

Gestión de la memoria



Se puede ver cómo el garbage collector va disminuyendo cada poco el uso de la memoria haciendo que su consumo se mantenga más estable.

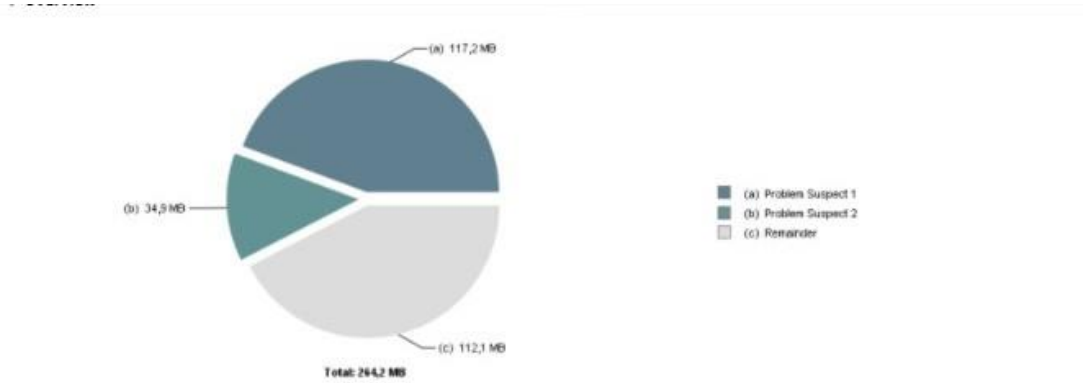
Memory leaks

Haciendo uso de la herramienta VisualVM se genera una archivo heapdump mientras la aplicación está corriendo en el servidor, que será analizado posteriormente con Eclipse MAT.

A continuación se muestran varios informes generados por esta última herramienta sobre las pérdidas de memoria que tienen lugar.

Se pueden observar 2 problemas principales que están consumiendo un gran porcentaje de la memoria.

- Instancias de la clase InstallableUnit del propio Eclipse referenciadas desde HashMap$Node.
- Instancias del LocalMetadataRepository.



Problem Suspect 1

37.953 instances of **"org.eclipse.equinox.internal.p2.metadata.InstallableUnit"**, loaded by **"org.eclipse.equinox.p2.metadata"** occupy **122.900.448 (44,37 %)** bytes. These instances are referenced from one instance of **"java.util.HashMap$Node[]"**, loaded by **"<system class loader>"**

Keywords
org.eclipse.equinox.p2.metadata
java.util.HashMap$Node[]
org.eclipse.equinox.internal.p2.metadata.InstallableUnit

Details »

Problem Suspect 2

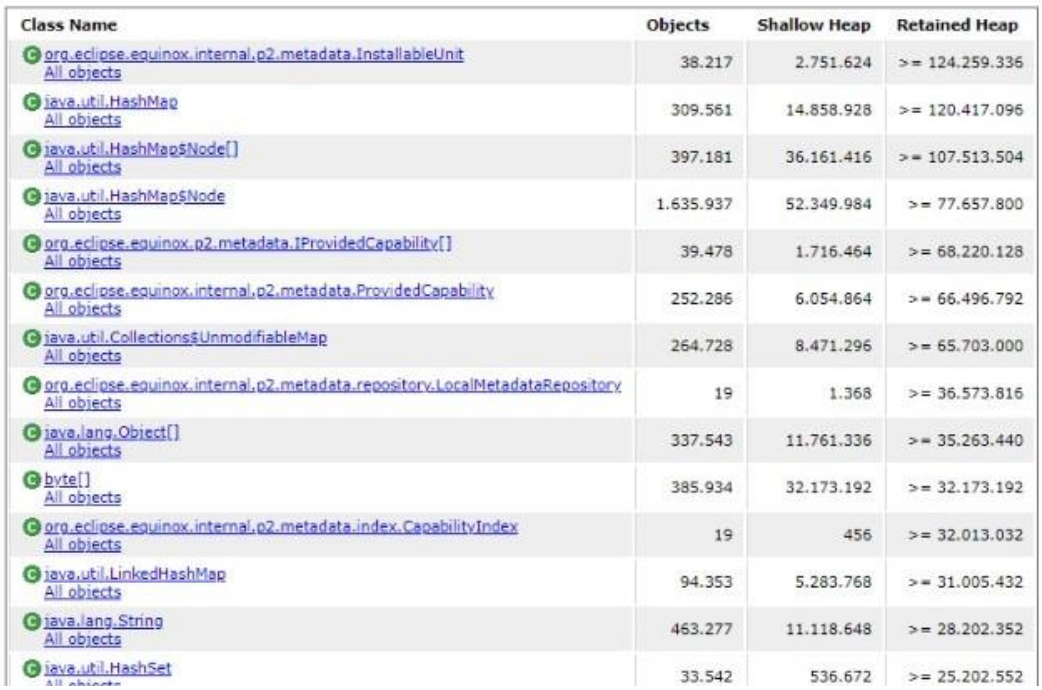
19 instances of **"org.eclipse.equinox.internal.p2.metadata.repository.LocalMetadataRepository"**, loaded by **"org.eclipse.equinox.p2.metadata.repository"** occupy **36.573.712 (13,20 %)** bytes.

Biggest instances:

- org.eclipse.equinox.internal.p2.metadata.repository.LocalMetadataRepository @ 0xcad608b0 - 3.845.528 (1,39 %) bytes.
- org.eclipse.equinox.internal.p2.metadata.repository.LocalMetadataRepository @

A continuación se muestra un histograma con las clases que más memoria consumen, en el cual se pueden ver cómo las primeras corresponden a las anteriormente mencionadas.

Class Histogram



Class Name	Objects	Shallow Heap	Retained Heap
org.eclipse.equinox.internal.p2.metadata.InstallableUnit All objects	38.217	2.751.624	>= 124.259.336
java.util.HashMap All objects	309.561	14.858.928	>= 120.417.096
java.util.HashMap$Node[] All objects	397.181	36.161.416	>= 107.513.504
java.util.HashMap$Node All objects	1.635.937	52.349.984	>= 77.657.800
org.eclipse.equinox.p2.metadata.IProvidedCapability[] All objects	39.478	1.716.464	>= 68.220.128
org.eclipse.equinox.internal.p2.metadata.ProvidedCapability All objects	252.286	6.054.864	>= 66.496.792
java.util.Collections$UnmodifiableMap All objects	264.728	8.471.296	>= 65.703.000
org.eclipse.equinox.internal.p2.metadata.repository.LocalMetadataRepository All objects	19	1.368	>= 36.573.816
java.lang.Object[] All objects	337.543	11.761.336	>= 35.263.440
byte[] All objects	385.934	32.173.192	>= 32.173.192
org.eclipse.equinox.internal.p2.metadata.index.CapabilityIndex All objects	19	456	>= 32.013.032
java.util.LinkedHashMap All objects	94.353	5.283.768	>= 31.005.432
java.lang.String All objects	463.277	11.118.648	>= 28.202.352
java.util.HashSet	33.542	536.672	>= 25.202.552

Tiempos de respuesta

Para este proyecto se han escogido tres situaciones diferentes de carga: 10, 100 y 200 usuarios concurrentes (hilos). Este último nivel de carga ha resultado en un desbordamiento de memoria, por lo que se ha descartado para un análisis más detallado.

Además, se han probado para cada nivel de carga una serie de peticiones HTTP para analizar los tiempos de respuesta de la aplicación. En concreto se han probado:

- Petición GET a / (mostrar página principal)
- Petición POST a /login (autenticación de usuario)
- Petición GET a /advertisement/list (mostrar lista de anuncios)
- Petición POST a /advertisement (publicación de anuncio)

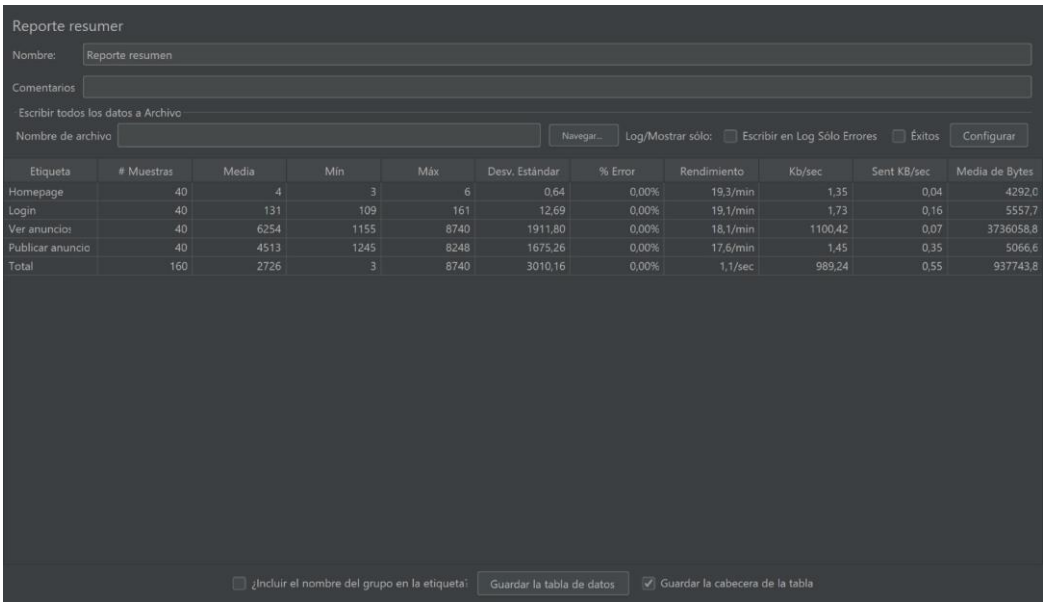
Se han ejecutado varias veces para obtener gráficos con mayor cantidad de datos.

A continuación, se muestran algunos de los gráficos resultantes después de realizar las pruebas con Jmeter.

Reporte resumen

- 10 usuarios

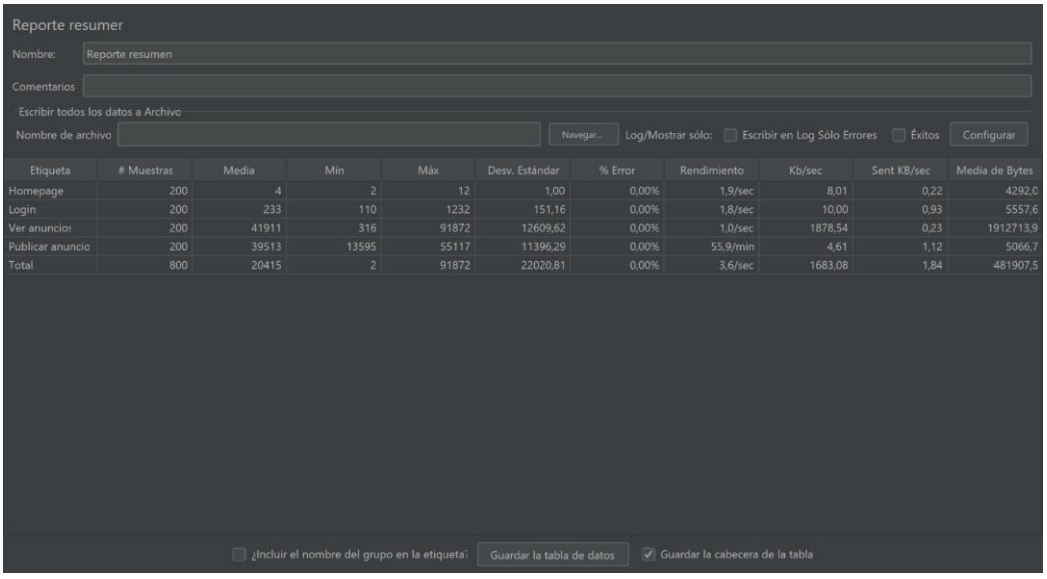
Se puede observar que todas las peticiones tienen un rendimiento más o menos parecido. La aplicación tiene de media un rendimiento de 1 petición por segundo.



Etiqueta	# Muestras	Media	Min	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
Homepage	40	4	3	6	0,64	0,00%	19,3/min	1,35	0,04	4292,0
Login	40	131	109	161	12,69	0,00%	19,1/min	1,73	0,16	5557,7
Ver anuncios	40	6254	1155	8740	1911,80	0,00%	18,1/min	1100,42	0,07	3736058,8
Publicar anuncio	40	4513	1245	8248	1675,26	0,00%	17,6/min	1,45	0,35	5066,6
Total	160	2726	3	8740	3010,16	0,00%	1,1/sec	989,24	0,55	937743,8

- 100 usuarios

  Con un mayor número de muestras, la aplicación ofrece un rendimiento de 3.6 peticiones por segundo.

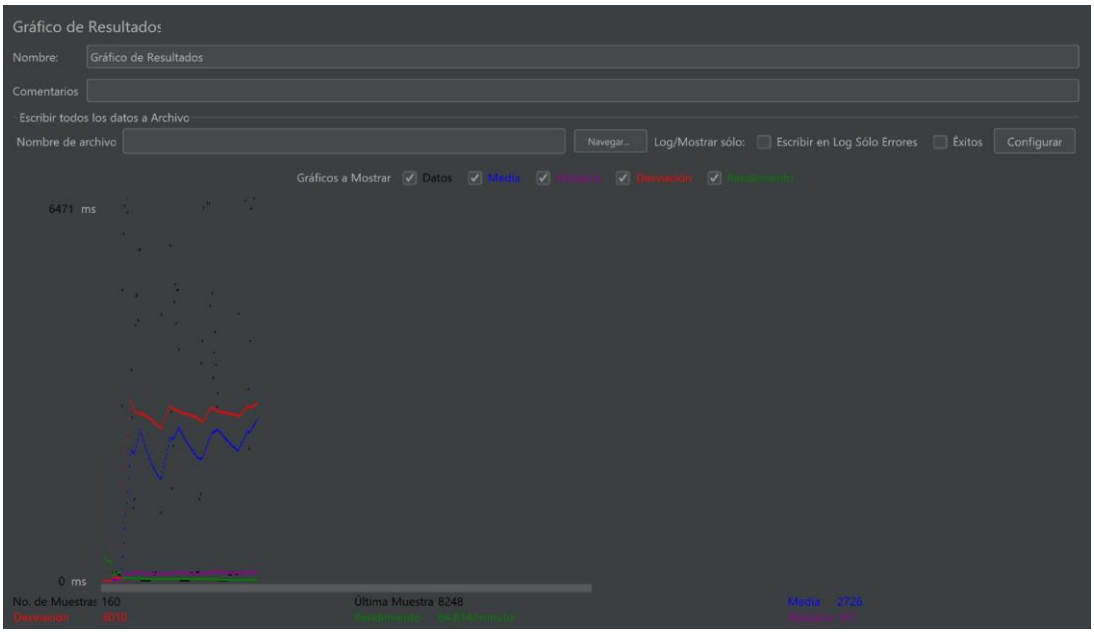


Etiqueta	# Muestras	Media	Min	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
Homepage	200	4	2	12	1,00	0,00%	1,9/sec	8,01	0,22	4292,0
Login	200	233	110	1232	151,16	0,00%	1,8/sec	10,00	0,93	5557,6
Ver anuncios	200	41911	316	91872	12609,62	0,00%	1,0/sec	1878,54	0,23	1912713,9
Publicar anuncio	200	39513	13595	55117	11396,29	0,00%	55,9/min	4,61	1,12	5066,7
Total	800	20415	2	91872	22020,81	0,00%	3,6/sec	1683,08	1,84	481907,5

Gráfico general de resultados

- 10 usuarios

  Se puede observar que para este nivel de carga, la aplicación atiende un total de 64814 peticiones por minuto.



- 100 usuarios

  Para este nivel de carga, la aplicación atiende un total de 214582 peticiones por minuto. Se observa que el número de peticiones aumenta considerablemente con respecto a la situación anterior.

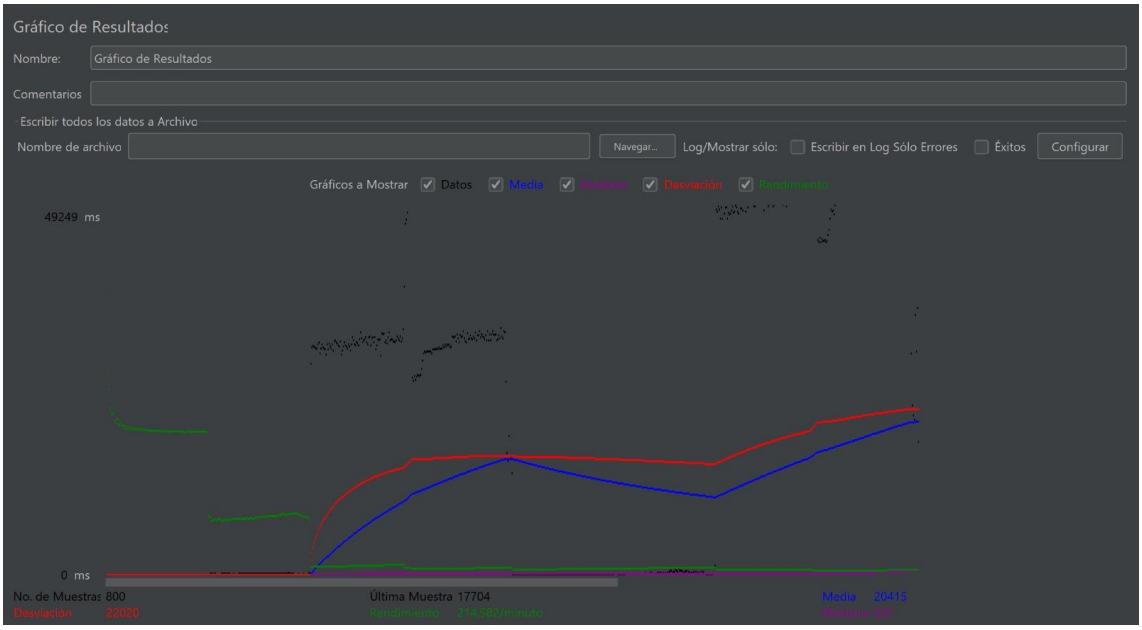


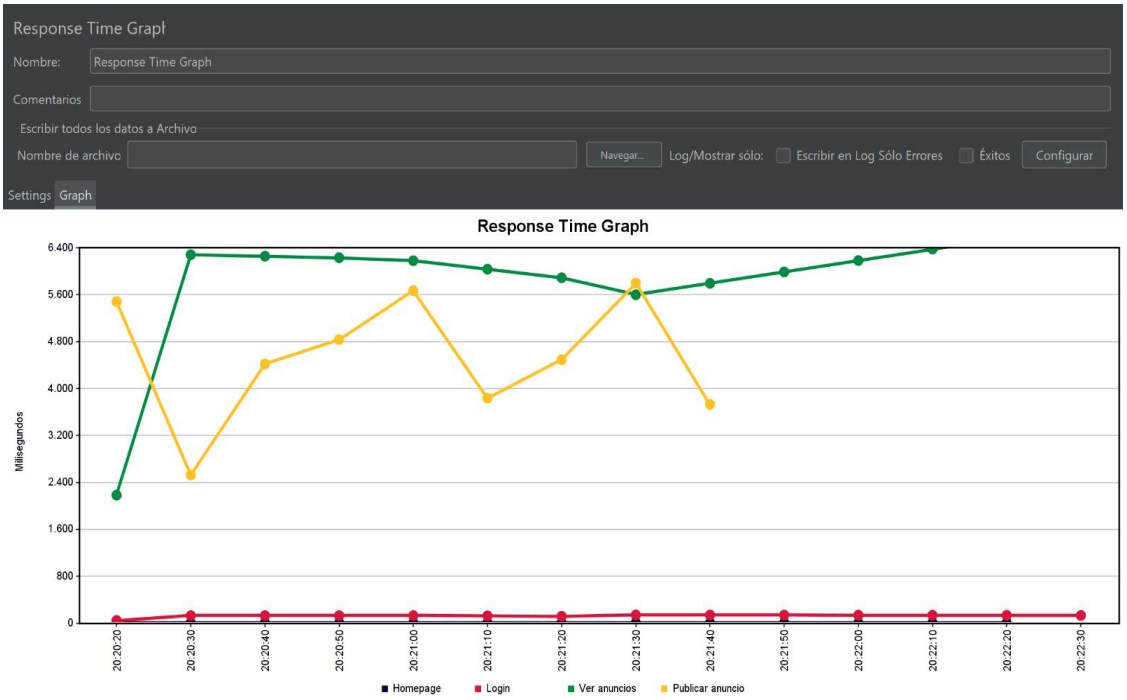
Gráfico de tiempos de respuesta

- 10 usuarios

  En este gráfico se puede ver cómo las peticiones de ver y publicar anuncios son las que suponen unos mayores tiempos de respuesta para la aplicación.

En el caso de ver anuncios observamos que los tiempos se mantienen bastante constantes a lo largo de toda la ejecución. Es la petición con mayor tiempo de respuesta, manteniéndose en un rango de entre 5,6 y 6,4 segundos.

Con respecto a publicar anuncio, podemos ver una mayor variación en los tiempos de respuesta, con picos mínimo y máximo de 2,4 y 5,6 segundos respectivamente.

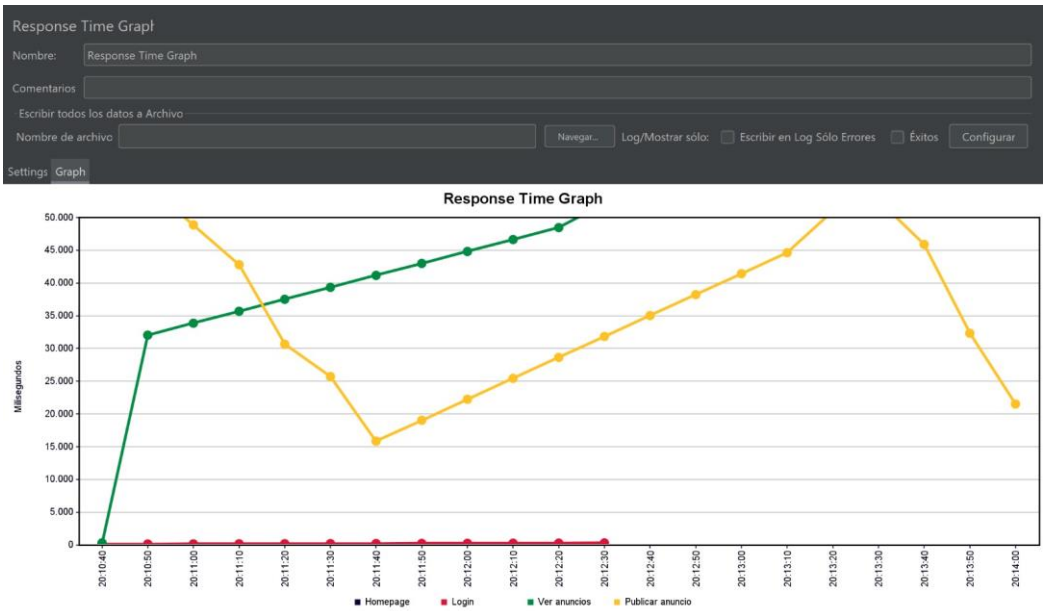


- 100 usuarios

  Para este caso también se siguen manteniendo ver y publicar anuncio como peticiones con mayores tiempos de respuesta. Sin embargo se puede ver que los tiempos en los que se mueven las peticiones son mucho mayores (de hasta 50 segundos).

  En el caso de ver anuncio vemos un crecimiento lineal de la gráfica que comienza con un valor de 32 segundos y aumenta hasta superar valores de 50 segundos.

  Para la petición publicar anuncio también se ve un crecimiento lineal de los tiempos de respuesta, siendo el menor de 15 segundos y el mayor de aproximadamente 55 segundos.



9. Jenkins

Jenkins se ha utilizado como herramienta de integración continua. Facilita así el despliegue de manera automática en un servidor Tomcat y gracias a su compatibilidad con Sonar, ésta actualiza los resultados de cada análisis.

Una vez que el proyecto es ejecutado correctamente por Jenkins, se despliega a un servidor Tomcat localizado en http://deploy.fic.udc.es.

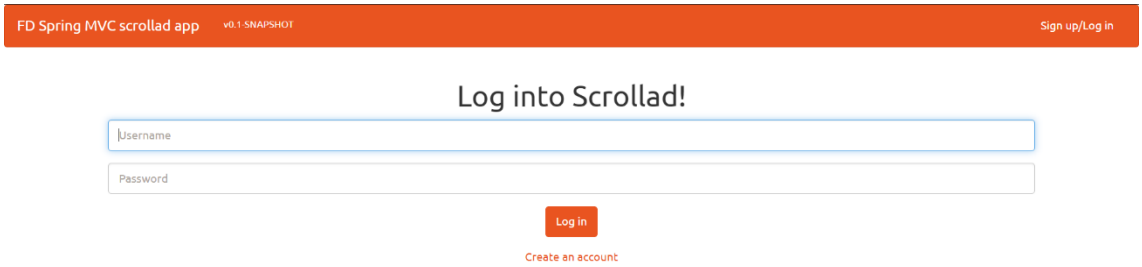
10. Sonar

Sonar ha sido la herramienta para la inspección continua. Debido a la configuración de Jenkins, Sonar actúa por cada ejecución realizando un análisis completo de cobertura de código, vulnerabilidades de seguridad, bugs, duplicaciones, etc. Además del análisis aporta otra información como líneas de código y el lenguaje de programación utilizado.

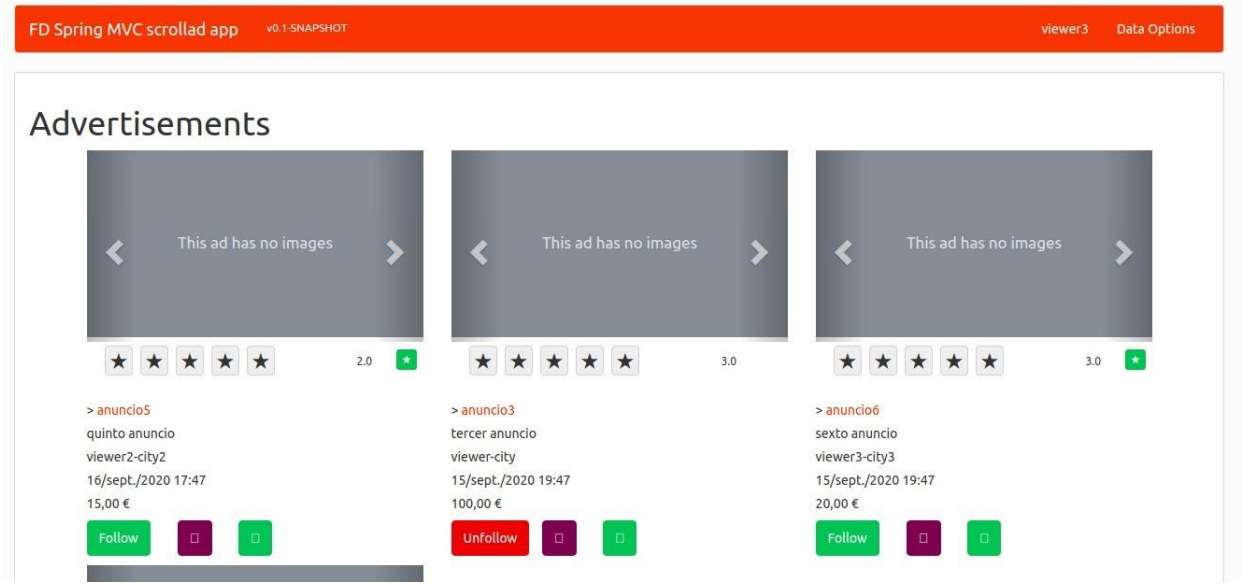
11. Aplicación

A continuación, se muestran algunas imágenes del aspecto de la aplicación:

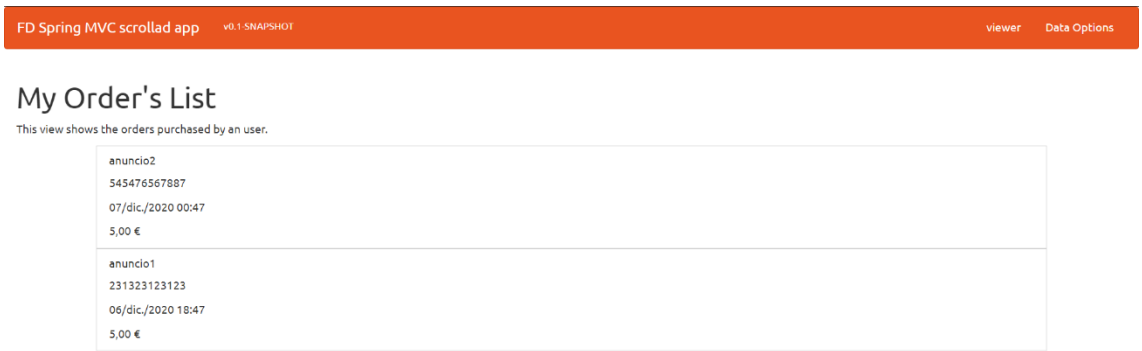
- Pantalla de autenticación:



- Pantalla para visualizar los anuncios subidos:

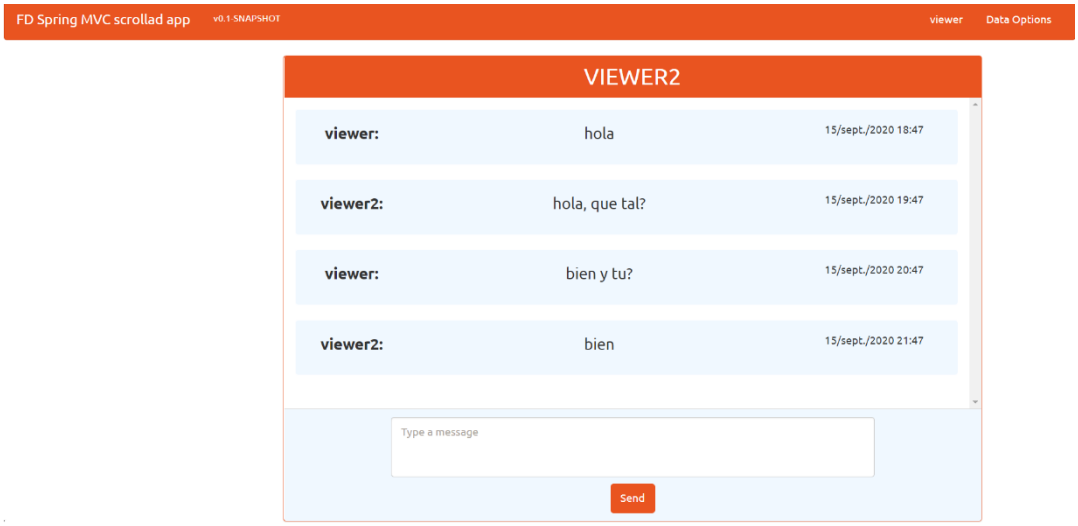


- Pantalla de Mis Pedidos:



- Pantalla del formulario para añadir anuncios:

- Pantalla de Chat:



- Visualización de los anuncios que le gustan al cliente:


FD Spring MVC scrollad app
v0.1-SNAPSHOT
viewer
Data Options
Advertisements
This view allows you to visualize all the advertisements you had liked.
This ad has no images
> anuncio1
primer anuncio
viewer-city
15/sept./2020 18:47
5,00 €
SOLD