Red Social de Imágenes

Ferramentas de Desenvolvemento

7 de Diciembre de 2018

Pablo Rivas Camino (pablo.rivas.camino@udc.es)
Bran López Pintor (bran.lopezp@udc.es)
Yanko Nión Ferreiro (yanko.ferreiro@udc.es)
Adrián Gómez García (a.gomezg@udc.es)

Índice:

1. Descripción del Trabajo:

1.1 Funcionalidades:

El trabajo realizado trata sobre la creación de una red social de imágenes, en la que los usuarios puedan subir y compartir contenido entre si. La aplicación se ha creado utilizando herramientas de desarrollo que permiten agilizar la creación de contenido, así como probar y testear que funcione correctamente. Este proyecto consta de los casos de uso a continuación descritos:

- Usuarios:
    - Posibilidad de darse de alta y autenticarse en la aplicación.
    - Un usuario puede seguir a otros con el fin de poder ver las imágenes que éstos suben, en un feed global.
    - Posibilidad de bloquear a usuarios para que no puedan visualizar las publicaciones compartidas o creadas, tanto si son historias como si son posts permanetes.
    - Distinción entre perfiles públicos, que pueden ver todos los usuarios y perfiles privados, en los que se ha de solicitar una petición previa al dueño del mismo por un usuario previamente autenticado.
    - Búsqueda de perfiles de usuario por nombre.
    - Un usuario puede hacer que otro deje de seguirle.
- Posts:
    - Distinción de los *feeds* de usuario con las publicaciones específicas de cada uno, el *feed* global donde cada usuario ve las publicaciones de la gente a la que sigue y el *feed* de historias donde las publicaciones son temporales.
    - Posibilidad de dar “me gusta” a las publicaciones de los usuarios.
    - Cada usuario puede ver las visualizaciones recibidas en cada publicación tanto si son anónimas como de otros usuarios ya registrados (especificando en este último caso de quién se trata).
    - Implementación y búsqueda de publicaciones con “*hashtags”*.
    - Cada publicación puede ser comentada y compartida por otros usuarios.

1.2. Herramientas:

Para la realización de la práctica se han utilizado las siguientes herramientas de las cuales se profundizarán en ellas más adelante a lo largo del documento:

- **Eclipse:** IDE multiplataforma compuesta de vistas y editores para el desarrollo de aplicaciones junto con la adición de diferentes plugins. En este caso se usó para desarrollo en el lenguaje de programación *Java*.

- **Git:** sistema de control de versiones distribuido.

- **Redmine:** aplicación web de gestión de proyectos flexible basada en *issues* o tareas.

- **Maven:** herramienta para la gestión y construcción de proyectos *Java* facilitando las labores de empaquetado, testeo, compilación… junto con la posibilidad de generar un informe "*maven site*" para resumir dependencias e instrucciones de ejecución de la aplicación.

- **Tomcat:** contenedor web de aplicaciones, en el que probamos la aplicación localmente.

- **Jenkins:** herramienta de código abierto de integración continua escrita en Java que proporciona servicios de integración continua para el desarrollo de software.

- **Sonar:** plataforma de administración de calidad del software creada para analizar proyectos software. Detecta problemas en el estilo de codificación, bugs potenciales, defectos de codificación, falta de cobertura de código…

- **JMetter:** herramienta para realizar tests de rendimiento permitiendo simular una gran carga de trabajo en uno o varios servidores.

- **Java Mission Control:** herramienta de profiling que para recopilar continuamente información de tiempo de ejecución mediante un marco de recopilación de perfiles y eventos (*Flight Recorder*).

- **Mockito:** framework que permite escribir pruebas simuladas mediante objetos de pega evitando el acceso a servicios de terceros.

- **Selenium:** herramienta de automatización de aplicaciones web con el fin de realizar pruebas contra las mismas en un navegador.

- **Spring MVC:** tecnología que ofrece una arquitectura "*modelo-vista-controlador*" para separar la lógica de negocio de la interfaz

- **Thymeleaf:** librería Java basada en plantillas HTML para el desarrollo del "*front-end*" de la aplicación

- **Liquibase:** biblioteca independiente de la base de datos utilizada para implementar y administrar cambios en el esquema bases de datos.
- **Cargo:** Manipulador de contenedores de aplicaciones que proporciona una api para configurarlo.
- **SpotBugs:** Herramienta para buscar bugs en en código localmente.

2. Arquitectura Global:

La práctica se ha dividido en un módulo padre que contiene todas las dependencias comunes a toda la aplicación, y que a su vez contiene tres submódulos:

- **Model:** parte encargada de la persistencia de la aplicación, conteniendo todo lo necesario para realizar las operaciones pedidas a lo largo de la práctica. Contiene diferentes servicios junto con sus entidades correspondientes.

- **CubisoftWeb:** parte encargada de todo lo que tiene que ver con el front-end y la parte web de la aplicación.

- **CubisoftREST:** parte encargada de proporcionar un servicio *REST*.

La distribución de los paquetes utilizada para cada módulo en la práctica es la siguiente:

- **/src/main/java:** carpeta en la que se almacenan todas las clases java de la aplicación.

    - **es.udc.fi.dc.fd.config:** paquete que contiene las clases java (anotadas con *@configuration*) encargadas de la configuración de diferentes partes de la aplicación, tales como la seguridad y la subida de imágenes.

    - **es.udc.fi.dc.fd.controller:** paquete que contiene los subpaquetes que albergan las clases java (anotadas con *@Controller*) para procesar las peticiones HTTP y realizar las operaciones de visualizar las páginas, guardar datos, procesar peticiones del usuario…

    - **es.udc.fi.dc.fd.model:** paquete con las clases java (anotadas con *@Entity*) encargadas de modelar las entidades del modelo.

    - **es.udc.fi.dc.fd.repository:** paquete con las interfaces java (anotadas con *@Repository*) encargadas de acceder a los datos de las entidades almacenados en la base de datos correspondiente.

    - **es.udc.fi.dc.fd.service:** paquete con las clases java (anotadas con *@Service*) encargadas de proporcionar las operaciones para ejecutar los casos de uso de la aplicación.

- **/src/main/resources:** se encuentran los archivos de configuración relativos a *Spring*, *Thymeleaf*, base de datos…

- **/src/main/test:** se encuentran las clases java (con métodos anotados con *@Test*) encargadas de ejecutar los test unitarios y de integración de la aplicación.

- **/src/main/webapp:** carpeta donde se encuentran las vistas y *fragments* HTML manejadas por la aplicación, scripts y recursos tales como imágenes;junto con el fichero de configuración *web.xml*.

3. Git:

Para el control de versiones la práctica ha sido almacenada en un repositorio *Git* de la facultad (*Gitlab*) donde se han ido incorporando todos los cambios a medida que se fueron realizando por cada uno de los miembros del grupo. Dichos cambios se han ido incorporando en las diferentes ramas siguiendo la siguiente filosofía de trabajo:

- **Develop:** cuando una funcionalidad de la aplicación se terminaba, esta se incorporaba directamente con un *commit* a esta rama.

- **Ramas 'feature':** en caso de que una funcionalidad no pudiese ser resuelta en un solo *commit*, esta se iba incorporando en una rama extra para dicha funcionalidad en varios *commits* para que finalmente se incorporase nuevamente en la rama *develop*.

- **Release:** cada *commit* de esta rama corresponde a un entregable de la aplicación correspondiente al fin de una de las cuatro iteraciones de la práctica, siendo a su vez este una incorporación de todos los cambios realizados sobre la iteración en la rama *develop*.

- **Master:** cada *commit* de esta rama corresponde nuevamente a un entregable de la aplicación correspondiente al fin de una de las cuatro iteraciones de la práctica habiendo sido este revisado previamente en la rama *release*. Además, estos contarán con un *tag* que simbolice la iteración para facilitar su descarga.

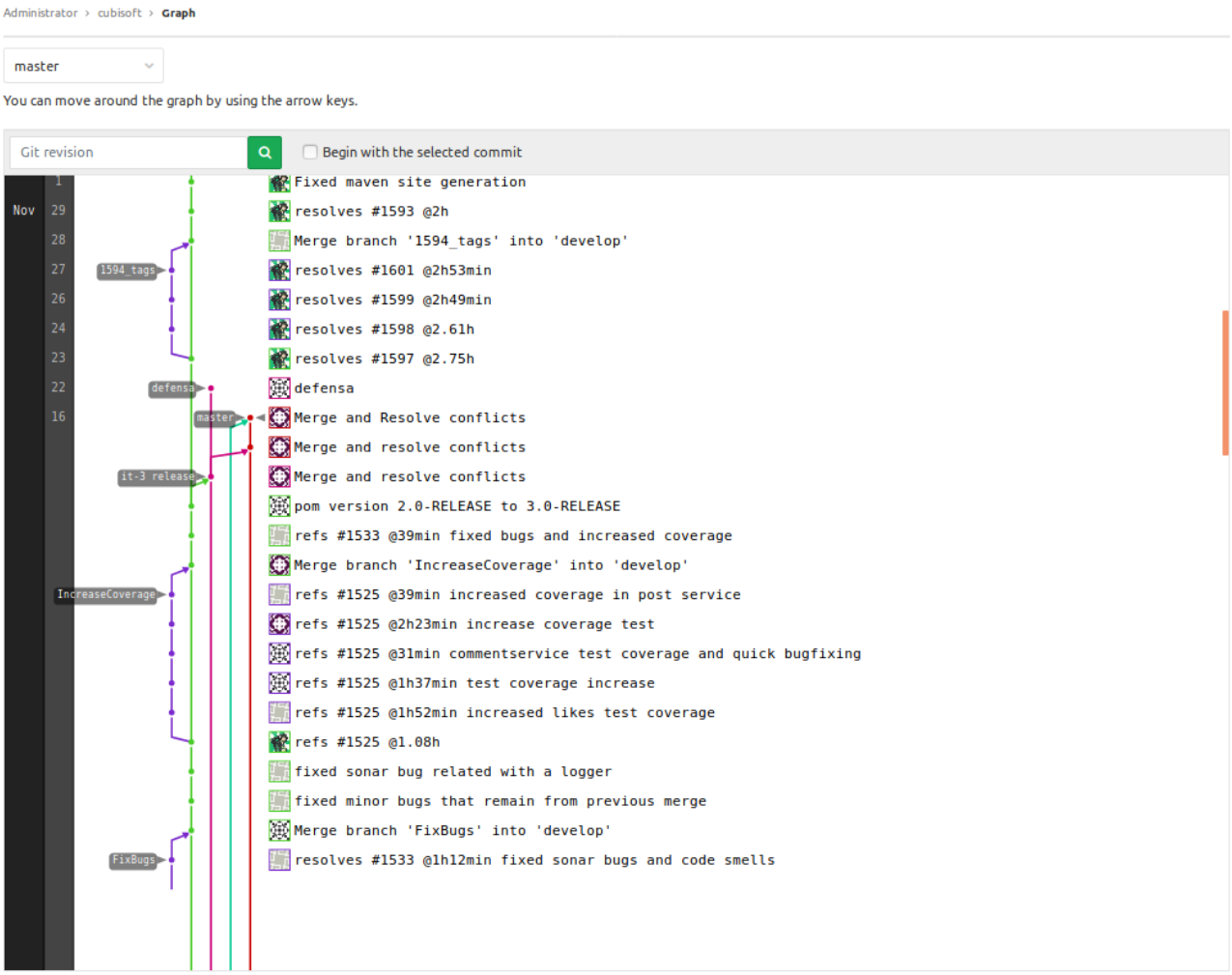

Fig.1: gráfico de las ramas utilizadas en Git

4. Planificación con Redmine:

Para la realización de la práctica se ha dividido el proyecto de *Redmine* en versiones, contando con una versión para cada iteración.

Respecto a las especificaciones y casos de uso proporcionados para la realización de la misma, estos se agregaron a los diferentes *tracks* de la herramienta como tareas, soporte o errores dependiendo de si se trataban de desarrollo, configuración o arreglo de fallos respectivamente. Estas, se fueron subdividiendo en subtareas estimando su duración en tiempo y prioridad según la complejidad de las mismas, asignándose según se fueron creando a algún miembro de la práctica para su posterior realización.

Mientras las tareas se fueron realizando, tras previamente haber sido marcadas como “en curso”, los tiempos de desarrollo han sido añadidos por acuerdo entre los miembros mediante la conexión de la herramienta con git en los comentarios de cada *commit*, especificando en estos a qué tarea referenciaban con el tiempo dedicado proporcionado por el plugin *Timekeeper* de *Eclipse* en caso de que sean tareas de desarrollo. En caso contrario, los tiempos se fueron imputados manualmente.

Una vez las tareas fueron finalizando, estas se marcaron como “resuelta” en la mayor parte de los casos salvo que estas no llegasen a ser completadas y tuviesen que ser pospuestas a una iteración posterior en cuyo caso se terminaron marcando como “cerrada”.

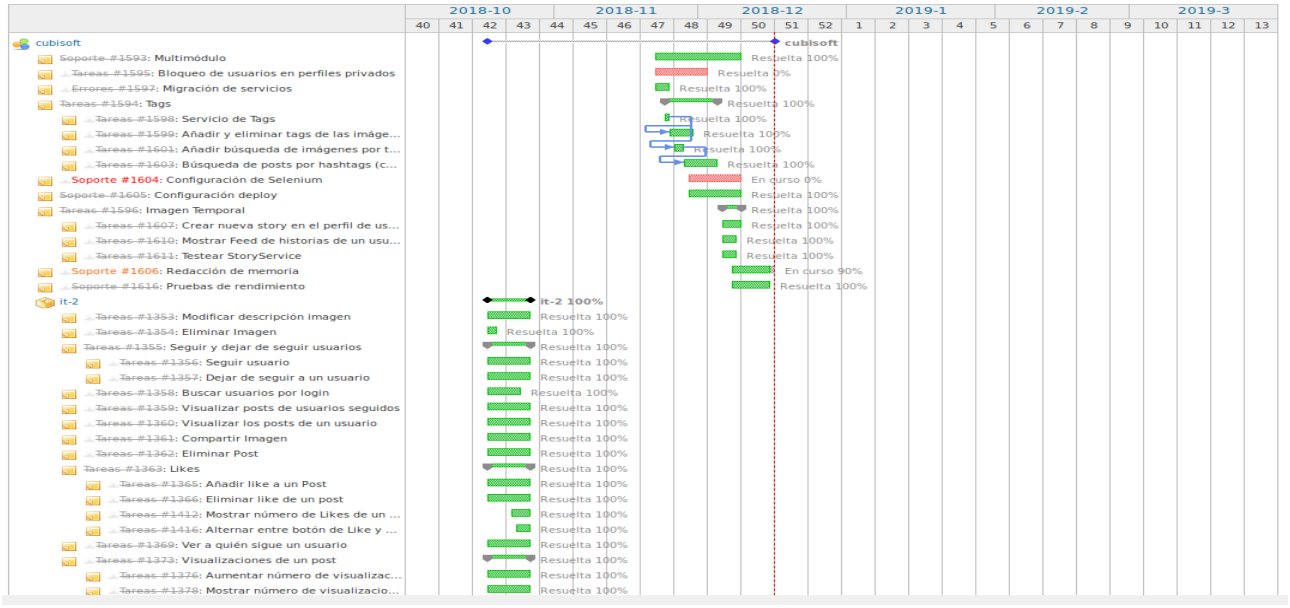

Fig.2: diagrama de las tareas en Redmine

5. Construcciones con Maven:

Para facilitar las construcciones del proyecto, éste se ha implementada como un arquetipo *Maven*. Posteriormente se han añadido las dependencias y plugins que se fueron necesitando a lo largo de la práctica. A mayores, se han añadido diferentes perfiles y metas detallados a continuación para que la aplicación se ejecute de manera diferente según la base de datos o servidor de aplicaciones que estemos usando.

Perfiles	
Nombre	**Descripción**
h2	Construye la aplicación para el uso de una base de datos *h2* en memoria.
mysql	Construye la aplicación para el uso de una base de datos *MySQL*.
tomcat7	Lanza la aplicación en un servidor *Tomcat* 7.
Metas	
Nombre	**Descripción**
site	Genera un *Maven Site* con toda la información a cerca de las dependencias, miembros, pruebas, errores... del proyecto.
tomcat7:run-war	Lanza la aplicación en un servidor *Tomcat 7*.

Fig.3: tabla de perfiles y metas empleadas en la práctica

Nota: los perfiles proporcionados por el arquetipo inicial por defecto "*jetty*" y "*postgres*" han sido deshabilitados al tener problemas con la subida de imágenes y desuso respectivamente.

6. Pruebas:

En este apartado se tocarán los aspectos relativos a los diferentes tipos de pruebas realizadas para probar la aplicación.

6.1. Pruebas de Unidad:

Para probar los servicios de la aplicación y por tanto el modelo de la misma, se han realizado las pruebas de unidad con la herramienta *JUnit* en adición a *Mockito*, herramienta que evita el acceso a la base de datos de la aplicación al convertir todas las clases repositorio y servicios en "objetos de pega" para poner énfasis en los test del servicio a probar.

Una vez ejecutadas las pruebas, estas generarán un informe de cobertura gracias al plugin de *JaCoCo* que se empleará más adelante por la herramienta *Sonar*.

6.2 Pruebas de Integración:

En el caso de esta práctica, las pruebas de integración se han hecho contra la interfaz web de la práctica como si fuesen pruebas de aceptación para poder probar el comportamiento entre todas las clases posibles del *front-end* y el modelo previamente testado con pruebas de unidad. Para estas pruebas se ha hecho uso de la herramienta *Selenium* que como previamente se ha mencionado, automatiza las mismas contra un navegador web; apoyándose en el plugin "*Cargo*" que previamente levanta un servidor *Tomcat* con la aplicación para su ejecución.

Una vez ejecutadas las pruebas, estas generarán otro informe de cobertura nuevamente con el plugin de *JaCoCo* que se empleará más adelante por la herramienta *Sonar*.

6.3 Pruebas de Rendimiento:

Tal y como se pidió en la práctica, se realizaron pruebas de rendimiento en la aplicación con la herramienta *JMetter* simulando a 100 usuarios concurrentes tratando de buscar las publicaciones que contuviesen cierto *hashtag*, funcionalidad que se apoya en un servicio *REST* de la aplicación.

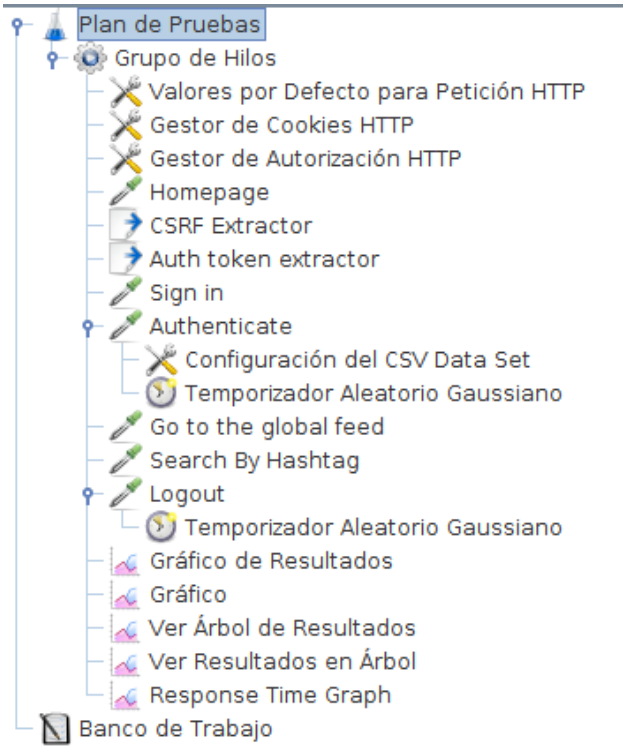

Fig.4: plan de pruebas de la práctica elaborado con JMetter

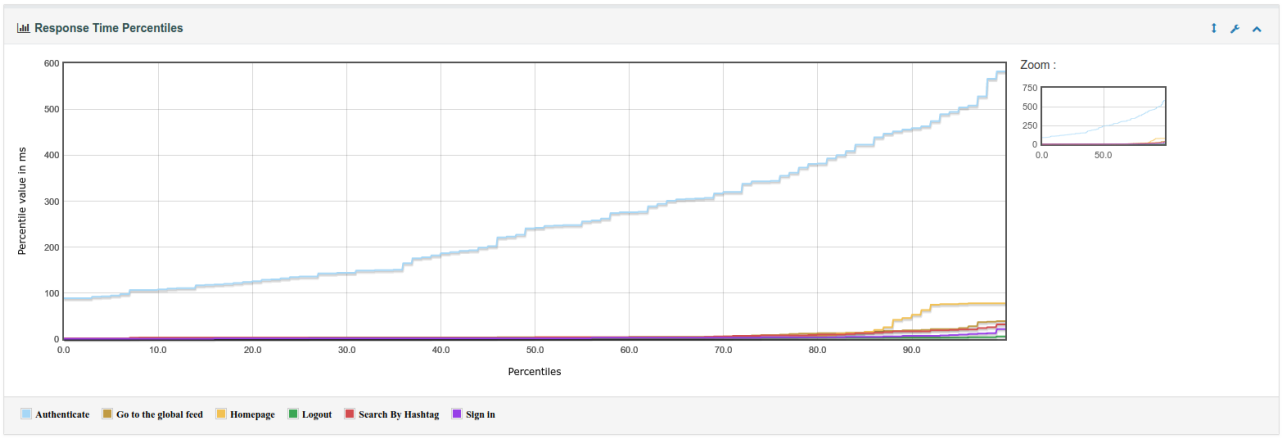

Fig.5: tiempos de respuesta generados al ejecutar el plan de pruebas

6.4 Profiling:

Con el fin de ver como se comportaba la aplicación con la memoria en determinados procesos se ha empleado la herramienta de *profiling Java Mission Control*. Estas pruebas, han sido realizadas mientras la aplicación ejecutaba las pruebas de rendimiento de forma paralela con *JMetter*.

A continuación se muestran algunos de los resultados obtenidos mientras se ejecutaba la prueba:

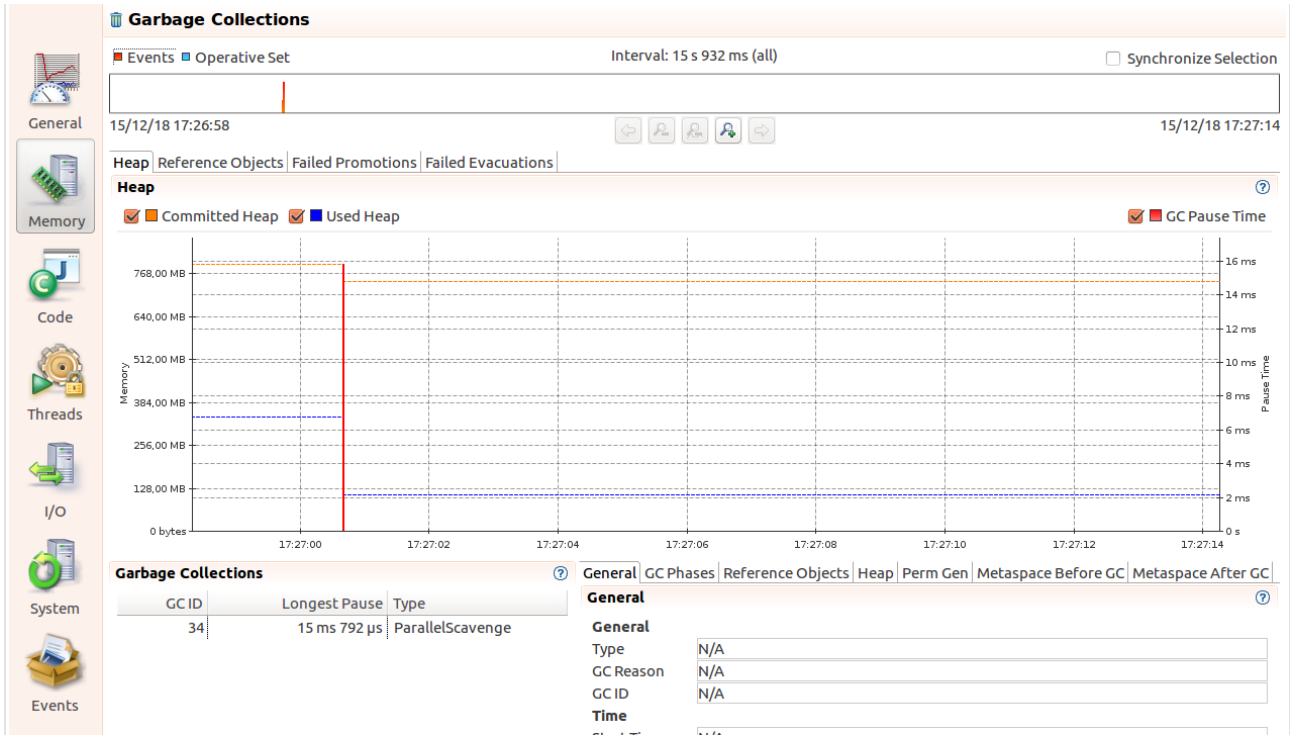

Fig.6: uso de la memoria durante la ejecución de la prueba marcando el paso del recolector de basura

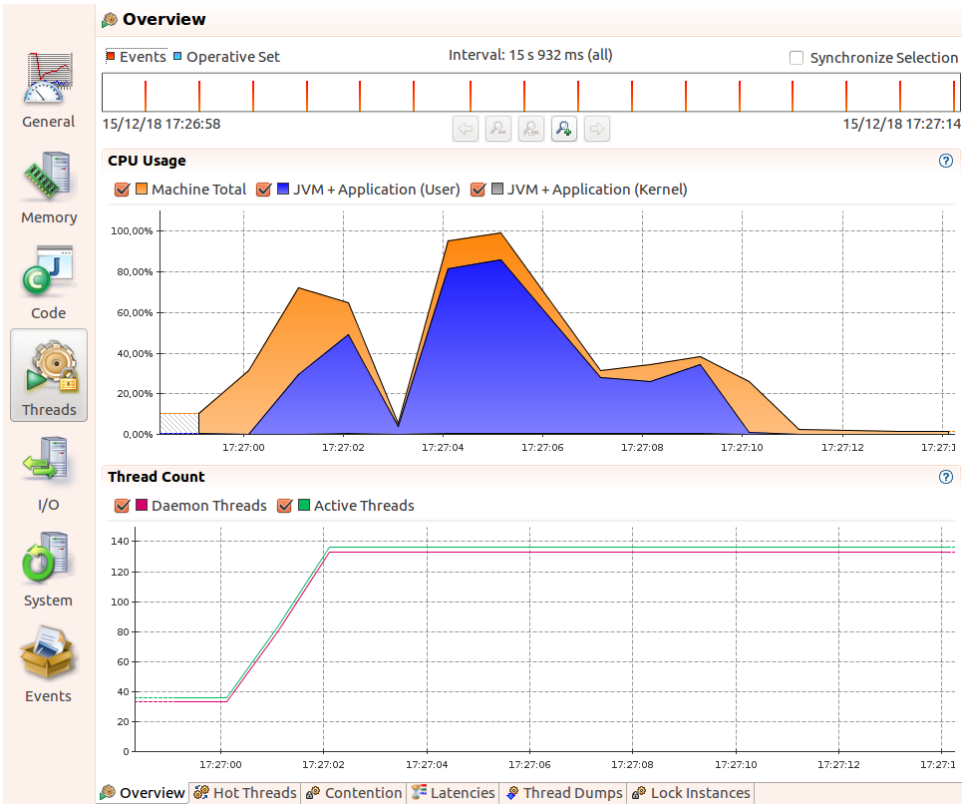

Fig.7: contador de hilos en la aplicación y CPU usada durante la prueba

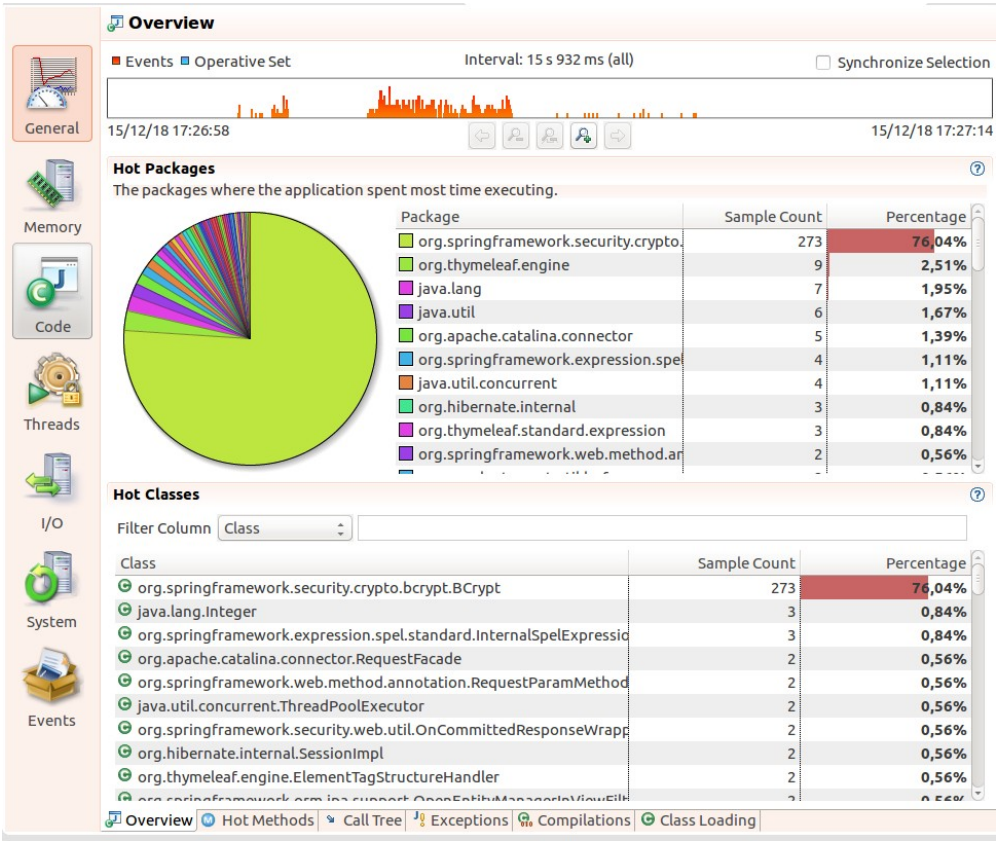

Fig.8: código que ha requerido de más memoria a lo largo de la prueba

7. Jenkins:

El proyecto ha sido integrado con la herramienta de integración continua *Jenkins* para poder controlar las construcciones realizadas sobre el mismo, poder observar la evolución de las pruebas, desplegar de manera automatizada la aplicación en un servidor *Tomcat* externo y actualizar los resultados de las pruebas en *Sonar*.

La herramienta ha sido configurada para que se lancen construcciones de manera automática cada vez que se realice un *commit* sobre el repositorio *Git* en las ramas "*develop*", "*release*" o "*master*" mediante el uso de un *webhook*.

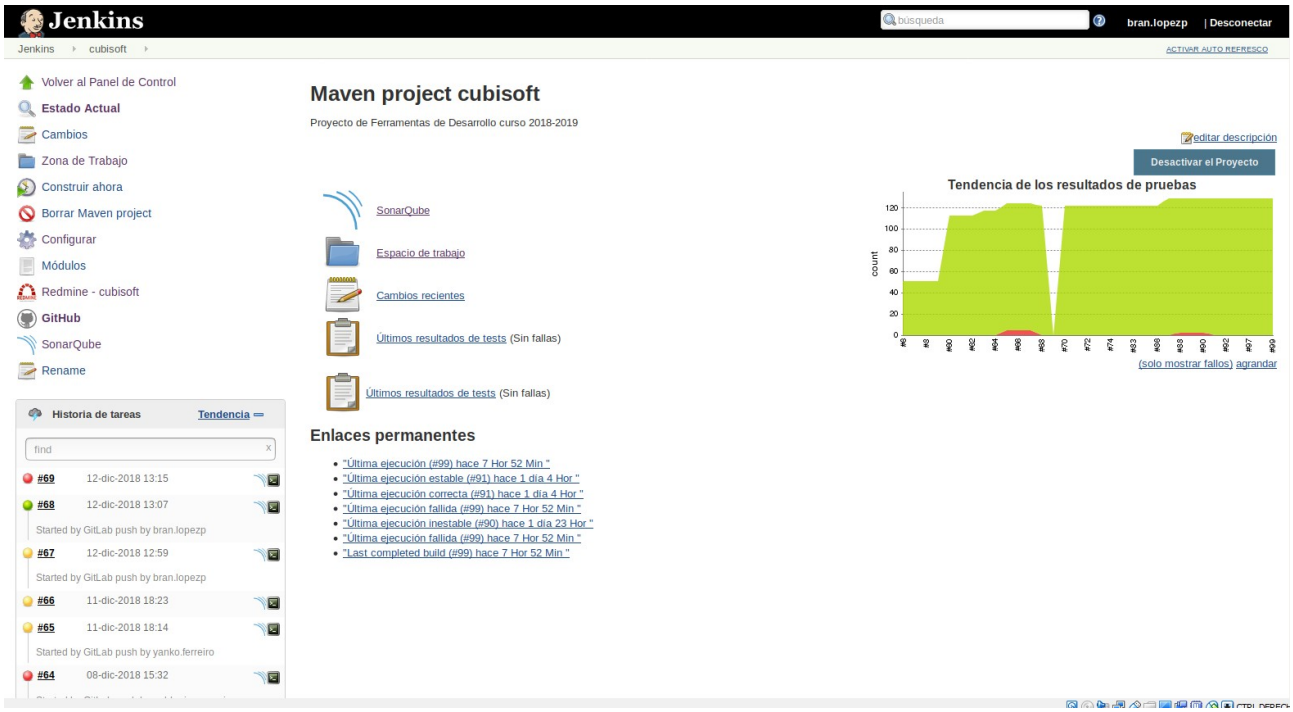

Fig.9: vistazo del proyecto en Jenkins

8. Sonar:

Una vez las construcciones hechas en *Jenkins* pasan como satisfactorias, estas actualizan los datos almacenados en *Sonar*, herramienta que ha sido utilizada para comprobar la calidad del código de la práctica. Esta nos proporciona los errores, *code smells*, vulnerabilidades, duplicados y cobertura que tiene el código en las diferentes clases de la aplicación por cada *commit* nuevo que hagamos y de manera global; en las ramas que configuramos en *Jenkins* (*develop*, *release* y *master*).

Una vez visualizados estos datos mencionados en la herramienta, se fue procediendo al aumento de cobertura en el código mediante la realización de nuevos test de unidad e integración y a la asignación y eliminación de errores y *code smells* encontrados por la herramienta para aumentar la calidad del código. Destacar que *Sonar* no detecta siempre errores reales, por lo que muchos tuvieron que ser marcados como "*falsos positivos*" en los casos que se consideraron que realmente no fuesen errores reales.

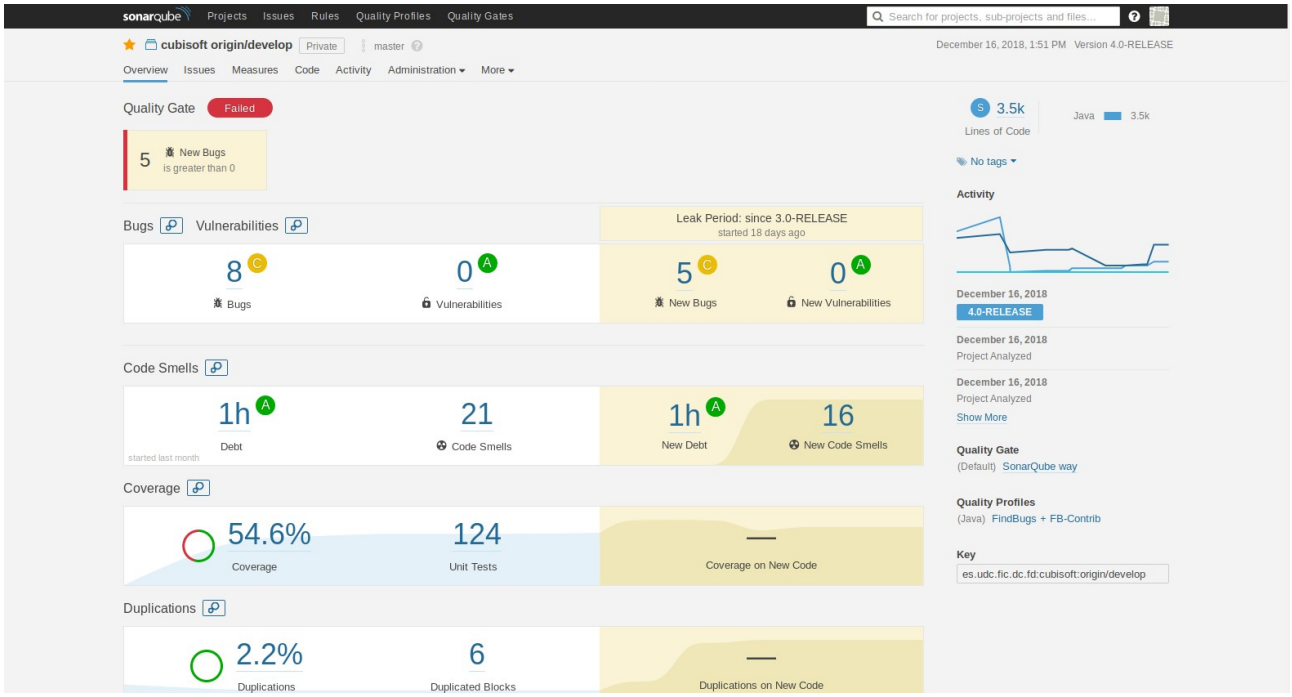

Fig.9: vistazo del proyecto en Sonar

9. Aplicación en funcionamiento:

Una vez una construcción pasa de manera satisfactoria en *Jenkins*, esta se despliega automáticamente en un servidor *Tomcat* de la facultad.

A continuación se muestran algunas capturas de la aplicación en funcionamiento:

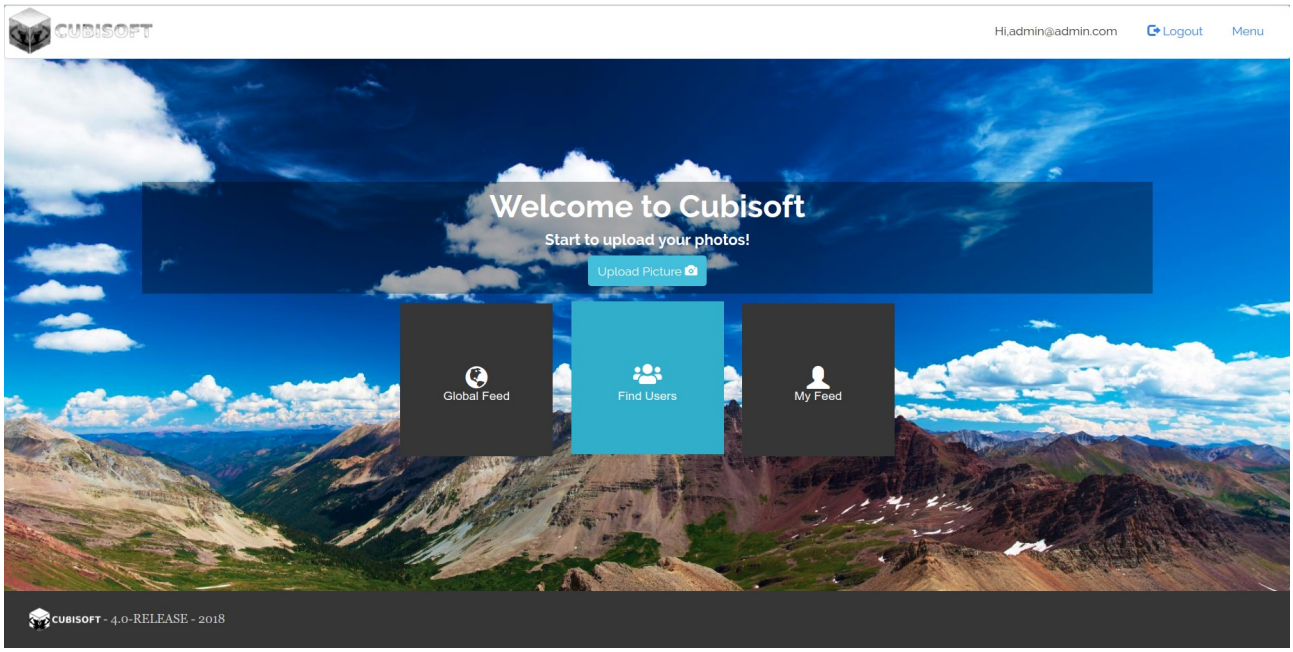

Fig.10: pantalla principal de la aplicación

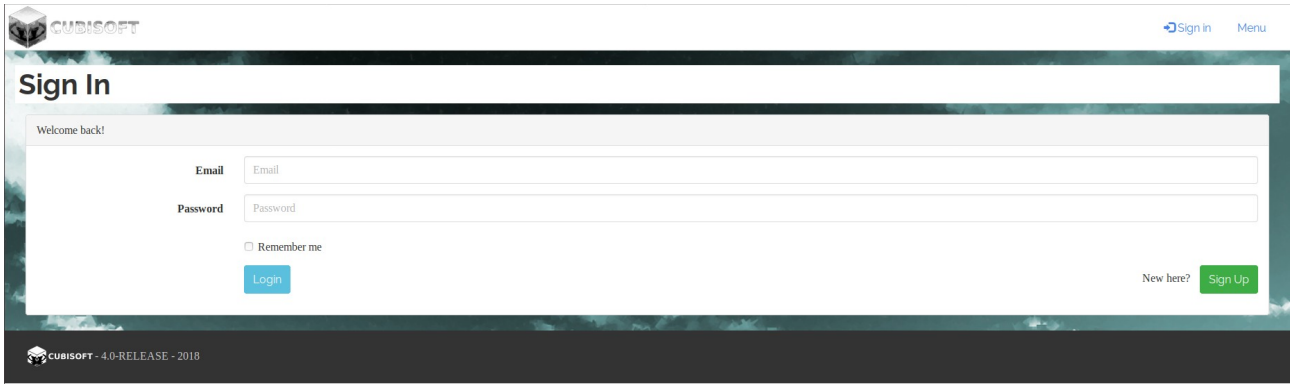

Fig.11: pantalla de login

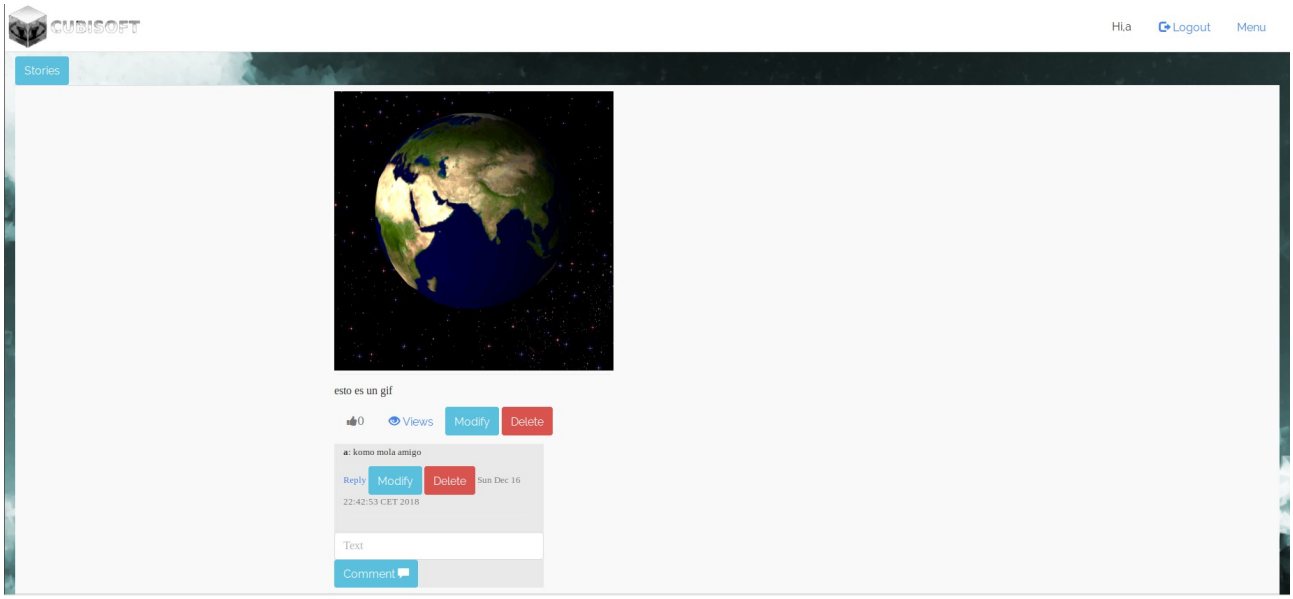

Fig.12: Feed de un usuario con comentario.

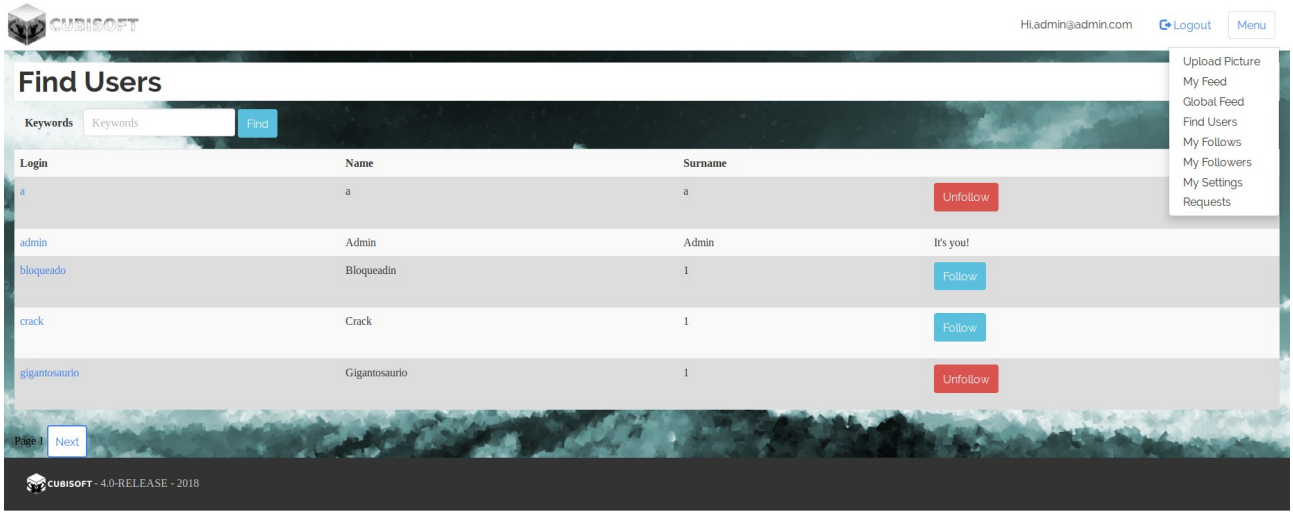

Fig.13: Búsqueda de usuarios seguidos y sin seguir, con menú de la aplicación desplegado.