Cash FIC

Ferramentas de Desenvolvemento

Adrián Ulla Rubinos, adrian.rubinos

Santiago Paz Pérez, santiago.paz1

Martín Rama Castro, martin.rama.castro

Ángel Paderne Cózar, a.pcozar

Juán Lopez Rodriguez, juan.lopez

Índice:

1. Descripción de la Aplicación:

1.1 Funcionalidades**:**

El objetivo de esta asignatura ha sido crear una aplicación web de compra y venta de bienes anunciados por los usuarios, estos usuarios podrán subir o buscar anuncios según su interés con un filtrado. Además, estos usuarios se podrán seguir, valorar y chatear con otros usuarios. Esta aplicación ha sido implementada siguiendo una serie de métodos y usando una serie de herramientas que describiremos más adelante.

Usuarios :

- Podrán registrarse y autenticarse en la plataforma
- Puede subir anuncios, especificando un título, una descripción, un precio e imágenes que ilustran el bien anunciado.
- Pueden buscar los anuncios subidos por otros usuarios mediante filtros como: palabras clave, fecha, precio, ciudad y valoración del anunciante.
- Se podrá valorar y seguir a otros usuarios.
- Podrán poner en espera sus anuncios o comprar los de otros usuarios.
- También se da la opción a los usuarios de convertirse en Premium para conseguir un mejor emplazamiento de su anuncio.
- Se proporciona un servicio de chat entre usuarios.

Anuncios:

- Se podrá dar me gusta a un anuncio.
- Emplazar anuncios de los usuarios premium en el tope de la lista´
- Poner anuncio en espera
- Comprar el elemento de un anuncio
- Eliminar anuncio

1.2. Herramientas**:**

Como comentamos anteriormente, aquí enunciamos y escribimos las herramientas utilizadas para el desarrollo de esta aplicación:

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 usado.

Redmine: Aplicación web de gestión de proyectos flexible basada en issues.

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.

Jetty: contenedor de aplicaciones web en modo desarrollo.

Tomcat: contenedor web de aplicaciones, en el cual probaremos la aplicación localmente.

Sonar: Plataforma de control de calidad del software creada para analizar proyectos sofware. 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 objetosde pega evitando el acceso a servicios de terceros.

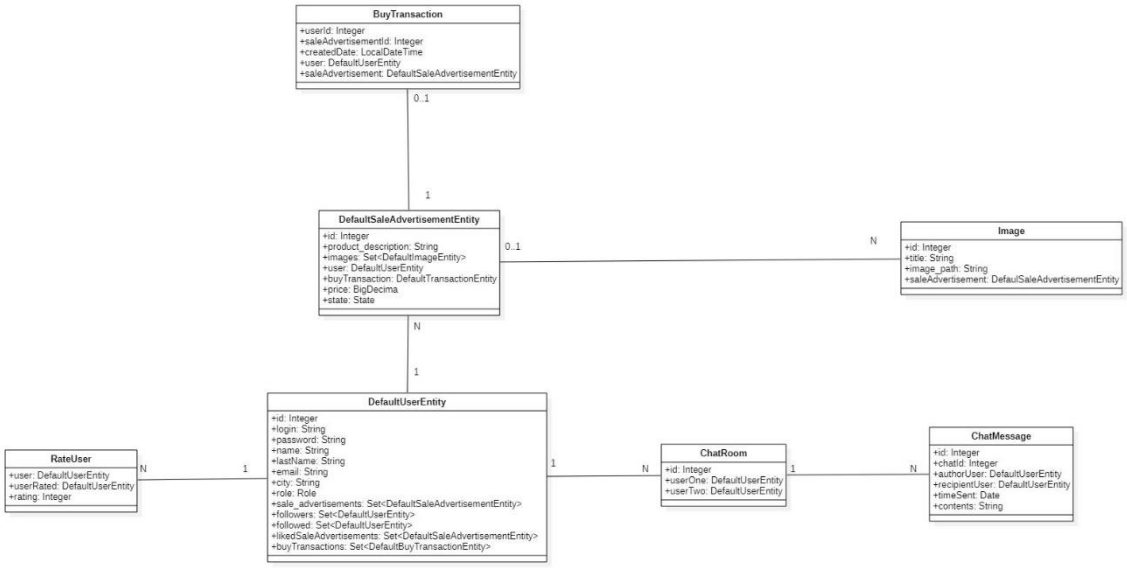
Selenium: Herramienta de automatización de aplicaciones web con el fin de realizar pruebas contra las mismas en un navegador.

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 de base de datos.

2. Diagrama Entidades:



3. Git:

Hemos usado Git para ir almacenando las distintas versiones de la práctica y poder tener un control sobre ellas. Se han ido incorporando los cambios dependiendo del estado en el que se encontraba la práctica en las ramas Feature, Develop, Release y Master.

Ramas ‘feature’ = Ramas en las cuales trabaja cada desarrollador individualmente e implementaba cierta funcionalidad, una vez se acaba el desarrollo, se hacía una merge request para incorporar la funcionalidad a la rama Develop.

Develop: Rama en la cual se incorporaban las funcionalidades acabadas y aprobadas por una merge request.

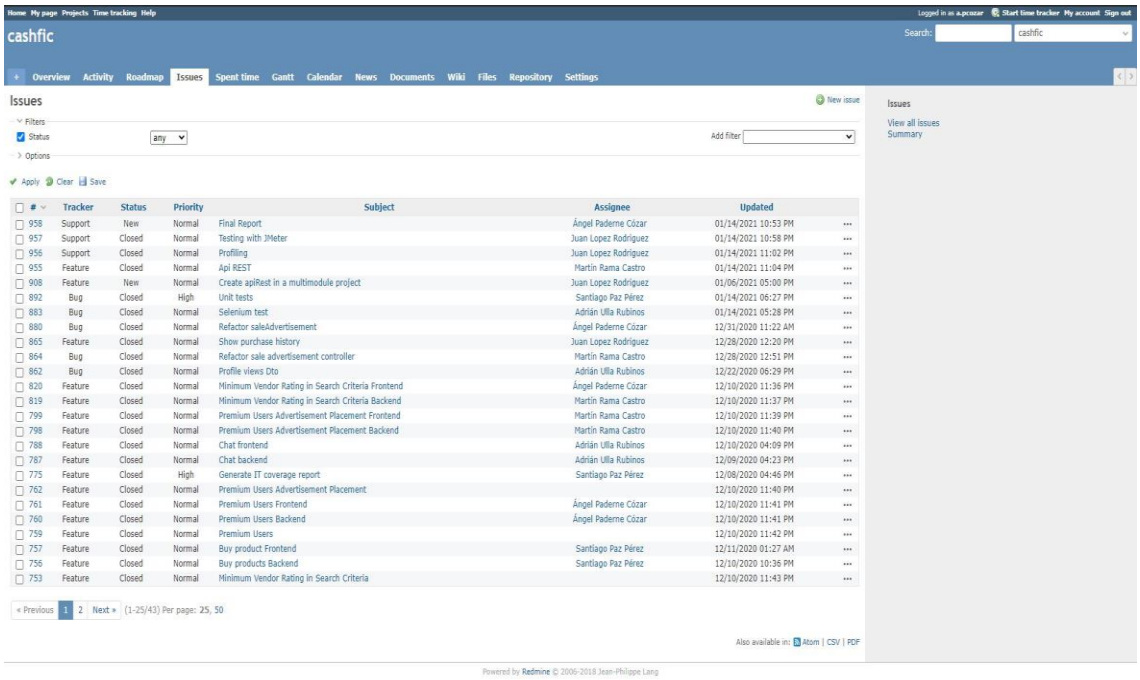
Release: Una vez que se realizaba una versión entregable de la práctica, esta rama era la encargada de alojar cada una de estas versiones. Cada commit corresponde a una versión lista para ser evaluada en cada iteración.

Master: Nuevamente en esta rama se alojan las versiones finales y revisadas de cada una de las versiones entregables de la rama Release.

4. Planificación con Redmine:

En cuanto a la organización y la gestión de este proyecto, hemos usado Redmine para dividir la carga de trabajo en tareas o issues. De esta manera, cada desarrollador cogía una tarea ya creada con anterioridad o creaba una tarea especificando el tiempo previsto en realizar la tarea, prioridad que tiene la misma, si era un trabajo de desarrollo o diseño y asignándose como encargado de la tarea.

Mientras el desarrollador está trabajando en la issue, está se encuentra “en curso”, cuando está tarea está realizada, pero no confirmada y revisada se encuentra “resuelta”, y acaba su ciclo de vida cuando se acepta y se marca como “cerrada”.



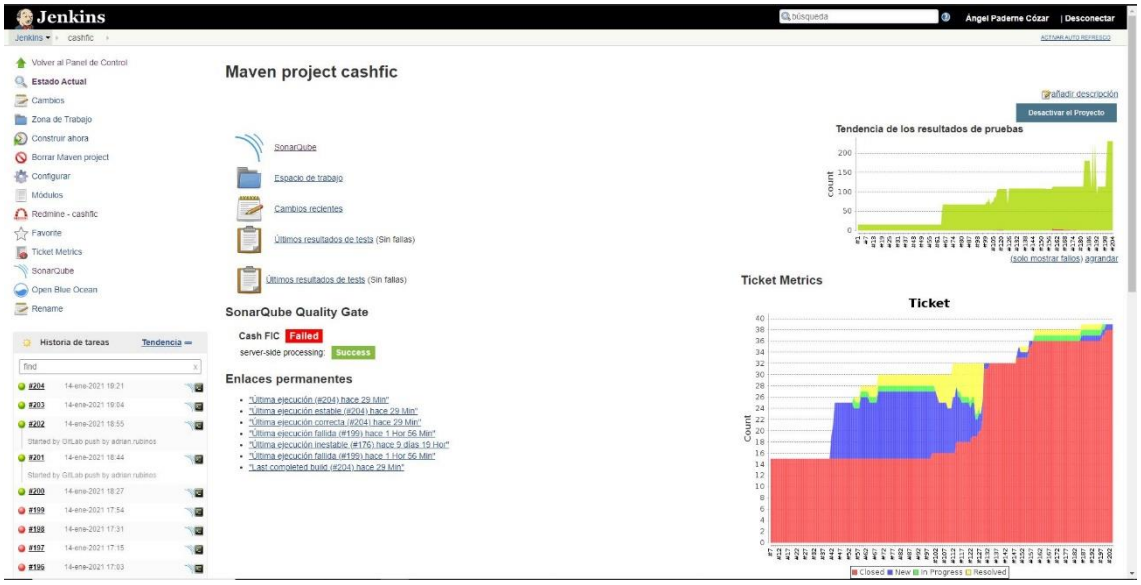
5. Construcciones con Maven:

Hemos utilizado Maven para la compilación usando un arquetipo del que nos han provisto en la asignatura. Luego, se añadieron dependencias y plugins que se fueron necesitando. También se han usado varios perfiles.

6. Integración con Jenkins:

Con Jenkins hemos realizado la integración continua, esto nos permite un control de las construcciones a partir de los cambios que se van añadiendo en la aplicación y poder desplegar de manera automática la aplicación en el servidor de aplicaciones externo y, con la configuración adecuada, conectarlo con Sonar para tener unos datos sobre los resultados en Sonar.

Además, mediante el uso de un webhook, cada vez que se lance un commit sobre Git la herramienta lanza una construcción de manera automática.

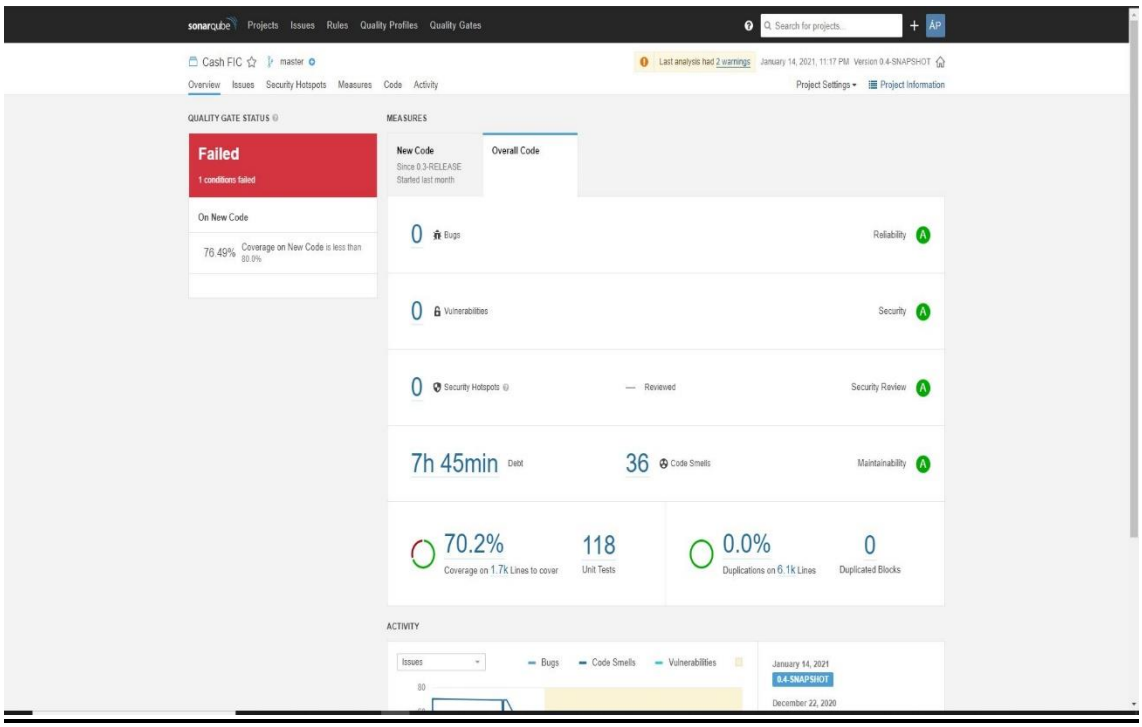


7. Inspección continua con Sonar:

Cada vez que se realiza una construcción satisfactoria en Jenkins, Sonar actualiza sus datos y estadísticas con los resultados de dicha construcción.

Esta herramienta nos permite tener una control sobre la calidad del código, los “Code Smells”, bugs y vulnerabilidades en el código, código replicado y el porcentaje de cobertura obtenido por los tests.

Después de la primera información obtenida por Sonar, fuimos refinando nuestro código y aumentando la cobertura. Un plugin bastante utilizado para realizar este control fue “Sonar on the fly” el cuál nos mostraba las posibles correcciones en vivo.



8. Pruebas:

8.1. Pruebas de Unidad:

Estas pruebas se han realizado con la herramienta JUnit y Mockito, la cual nos permite convertir todas las clases repositorio y servicios en “objetos de pega” para evitar el acceso a la base de datos de la aplicación. Para generar el informe de cobertura que usa Sonar, usamos el plugin JaCoCo.

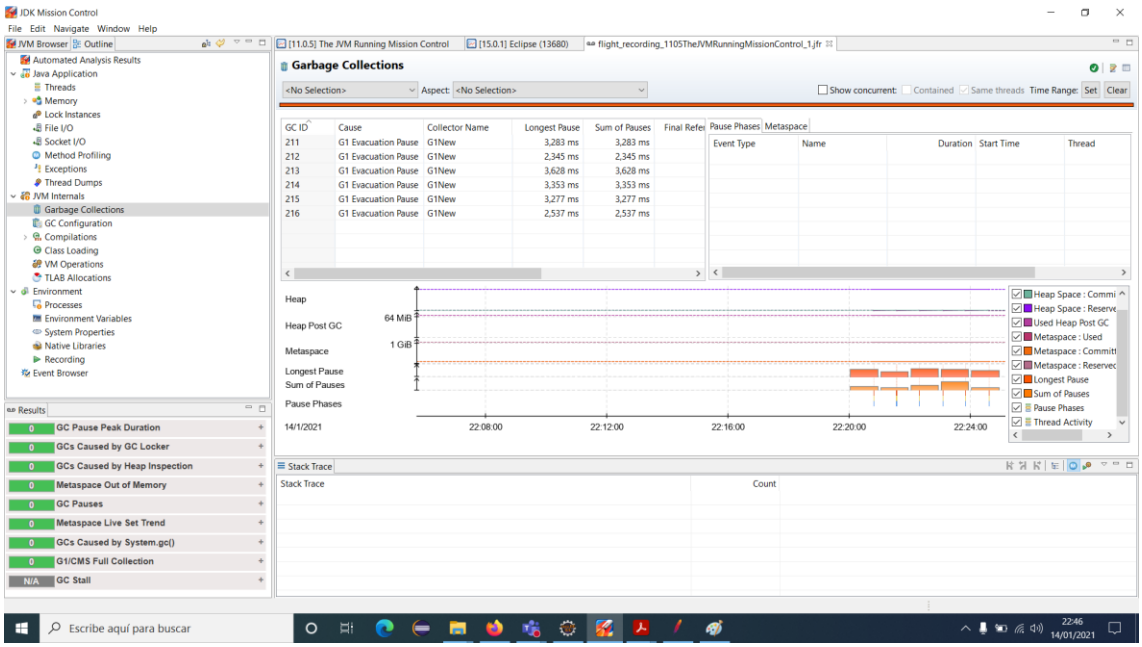
También indicamos al plugin Surefire que tests son pertenecientes a los de Unidad.

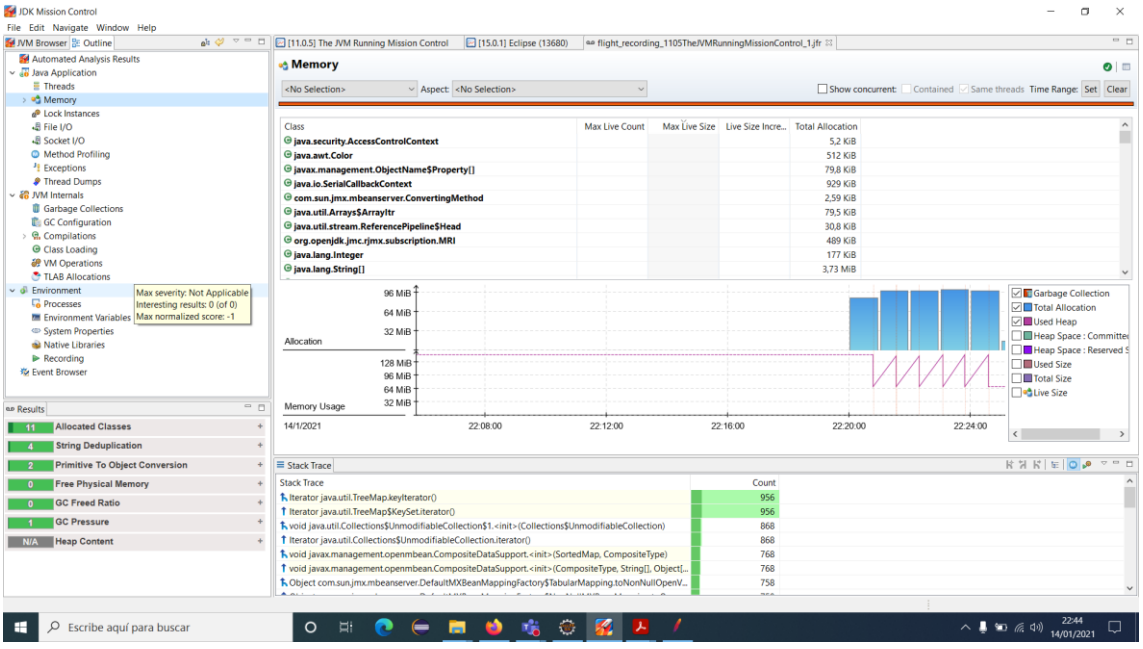
8.2. Pruebas de Integración:

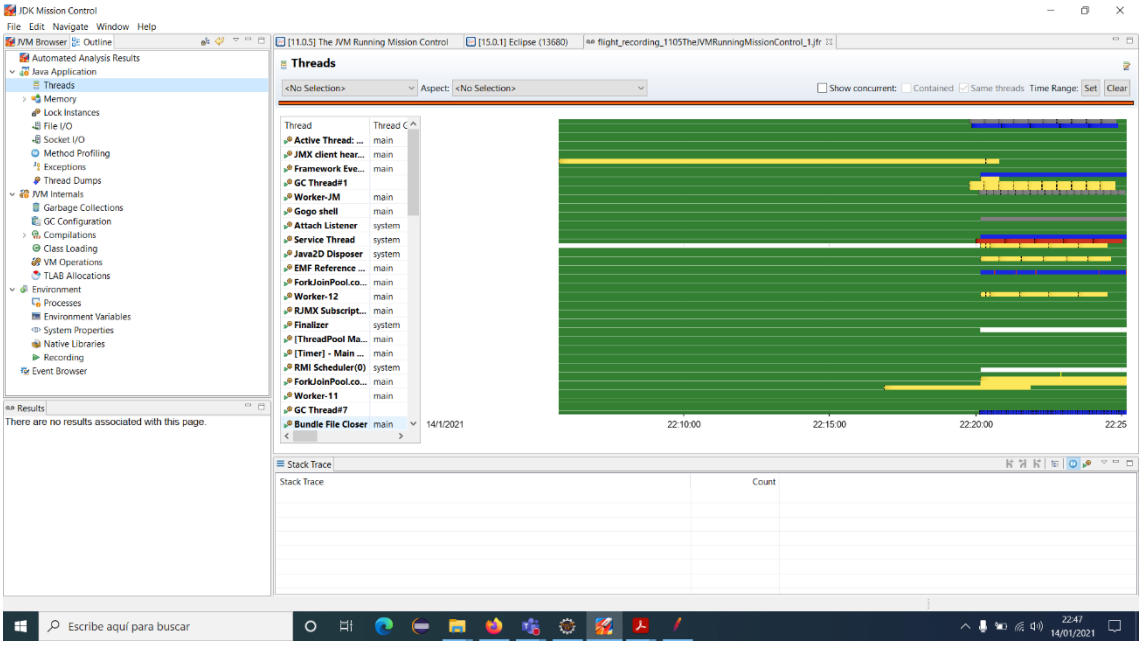
Para las pruebas de Integración usamos el plugin de Failsafe para indicar los tests que serán ejecutados en esta fase. También con Maven, para ejecutar los tests de integración, utilizamos Selenium, ya que este necesita tener levantado un servidor.

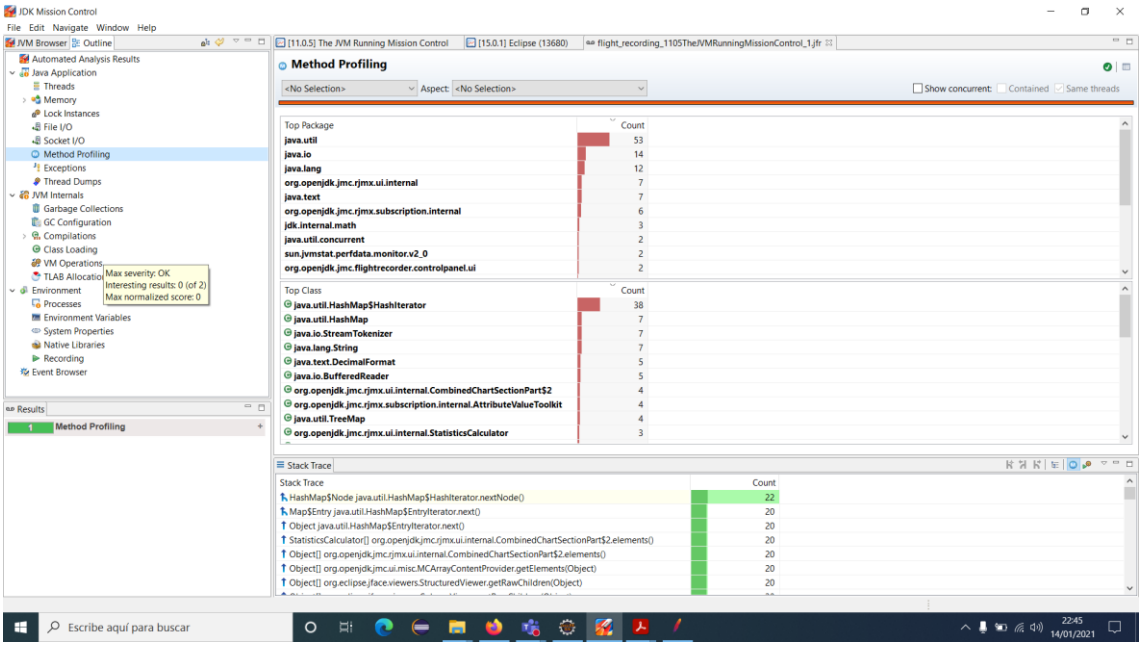
8.3. Profiling:

Para tener un monitorizado del comportamiento de la aplicación, gestión de la memoria, etc… Hemos utilizado la herramienta Java Mission Control. Para esto partimos de un plan de pruebas realizado con JMetter.









Gracias a esta última imagen, podemos observar los paquetes más usados, de esta manera, tenemos una referencia para optimizar y mejorar el rendimiento de la aplicación.