Informe de Testing Individual

Integrantes del grupo C2.026:

* Ignacio Blanquero Blanco ([ignblabla@alum.us.es](mailto:ignblabla@alum.us.es))
* María de la Salud Carrera Talaverón ([marcartal1@alum.us.es](mailto:marcartal1@alum.us.es))
* Joaquín González Ganfornina ([joagongan@alum.us.es](mailto:joagongan@alum.us.es))

Fecha: Sevilla, 8 de Julio 2024

Tabla de contenido

[Resumen del Informe 3](#_Toc170381051)

[Historial de Versiones 4](#_Toc170381052)

[Introducción 5](#_Toc170381053)

[Testing Funcional 6](#_Toc170381054)

[Assignation 6](#_Toc170381055)

[Create 6](#_Toc170381056)

[Delete 6](#_Toc170381057)

[List 6](#_Toc170381058)

[Show 6](#_Toc170381059)

[Project 7](#_Toc170381060)

[Create 7](#_Toc170381061)

[Delete 7](#_Toc170381062)

[List 7](#_Toc170381063)

[Publish 7](#_Toc170381064)

[Show 8](#_Toc170381065)

[Update 8](#_Toc170381066)

[User Story 9](#_Toc170381067)

[Create 9](#_Toc170381068)

[Delete 9](#_Toc170381069)

[List All 9](#_Toc170381070)

[List 9](#_Toc170381071)

[Publish 10](#_Toc170381072)

[Show 10](#_Toc170381073)

[Update 10](#_Toc170381074)

[Testing de Rendimiento 11](#_Toc170381075)

[Primer análisis 11](#_Toc170381076)

[Segundo análisis 12](#_Toc170381077)

[Contraste 13](#_Toc170381078)

[Conclusiones 14](#_Toc170381079)

[Bibliografía 15](#_Toc170381080)

# Resumen del Informe

El presente informe se centra en describir las pruebas realizadas sobre las funcionalidades implementadas por el estudiante 1. Se describe brevemente cada una de ellas, detallando cómo han ayudado a detectar problemas en el código, si procede.

Adicionalmente, se analiza el rendimiento de la aplicación utilizando como medida el tiempo empleando por esta en ejecutar los tests grabados. Se comparan los resultados de esta primera implementación, la implementación inicial que se hizo en el tercer entregable junto con los cambios introducidos en la segunda convocatoria, con los resultados de una segunda implementación que incluye el uso de índices en las entidades implicadas. Se valoran positivamente ambos resultados y se concluye que el más eficiente es el del primer enfoque.

# Historial de Versiones

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Contenidos** | **Fecha** | **Contribuyente** |
| V1.0 | Informe de testing. | 27/05/2024 | Mª de la Salud Carrera Talaverón |
| V2.0 | Modificaciones de segunda convocatoria. Pruebas descritas más profundamente. Nuevos análisis de rendimiento. | 27/06/2024 | Mª de la Salud Carrera Talaverón |
|  |  |  |  |

# Introducción

Este informe se divide en dos capítulos: testing funcional y testing de rendimiento o performance. En el capítulo de testing funcional se describen las pruebas realizadas sobre las tres entidades del estudiante 1: Assignation, Project y UserStory. Los tests están agrupados por entidad, funcionalidad y naturaleza del mismo: test seguro (casos positivos y negativos) o test de hacking. Para cada uno de ellos se explica en qué consiste el test y, si procede, los bugs de código que ha ayudado a encontrar.

Por otro lado, en el capítulo de testing de performance se analiza el rendimiento de la aplicación. Para ello se muestra en una gráfica el tiempo medio que tarda en realizarse cada petición durante la ejecución de los tests y se proporcionan estadísticas sobre estos tiempos. A continuación, se modifica el código para añadir índices en las entidades del estudiante 1, siguiendo las consultas de sus respectivos repositorios, con el objetivo de mejorar el rendimiento. Se proporciona un análisis sobre esta segunda ejecución y se contrastan ambos resultados.

Finalmente, el informe incluye una conclusión sobre los resultados obtenidos y las medidas tomadas respecto a ellos.

# Testing Funcional

## Assignation

### Create

* **create.safe:** varios managers crean asignaciones entre sus proyectos e historias de usuario (HU, en adelante). Se han probado todos los casos posibles: enviar el formulario vacío, enviar el formulario con todos los campos rellenos excepto uno cada vez, asignar una HU sin publicar a un proyecto sin publicar (acción permitida), asignar una HU publicada a un proyecto publicado (acción permitida), asignar una HU publicada a un proyecto sin publicar (acción permitida) y asignar una HU sin publicar a un proyecto publicado (acción no permitida).
* **create-wrong-role.hack:** un usuario anónimo introduce la URL de creación de una asignación y recibe un error 500 como respuesta.

### Delete

* **delete.safe:** varios managers borran sus asignaciones. Se han probado todos los casos posibles: borrar una asignación cualquiera de un proyecto sin publicar (acción permitida), borrar la única asignación de un proyecto sin publicar (acción permitida), borrar una asignación cualquiera de un proyecto publicado (acción permitida) y borrar la única asignación de un proyecto publicado (acción no permitida).
* **delete-wrong-role.hack:** un usuario anónimo introduce la URL de borrado de una asignación y recibe un error 500 como respuesta ya que no tiene el rol “manager” que necesita para poder acceder a la URL.

### List

* **list.safe:** todos los managers listan sus asignaciones. Aquellos cuyo listado no esté vacío, utilizan los índices de navegación para ver distintas páginas de la lista y la barra de búsqueda para filtrar las entradas.
* **list-wrong-role.hack:** un usuario anónimo introduce la URL de listado de asignaciones y recibe un error 500 como respuesta.

### Show

* **show.safe:** varios managers muestran sus asignaciones.
* **show-right-role-right-user-wrong-action.hack:** un manager intenta mostrar una asignación inexistente.
* **show-right-role-wrong-user.hack:** el manager2 intenta mostrar una asignación del manager1, recibiendo como respuesta un error que le indica que no está autorizado para realizar esa acción.
* **show-wrong-role.hack:** un usuario anónimo intenta mostrar una asignación, para lo que no está autorizado.

## Project

### Create

* **create.safe:** varios managers crean proyectos nuevos. Para cada atributo, se han probado todos los casos posibles y no posibles, con el fin de comprobar que salta el error adecuado. Se han tenido en cuenta las restricciones personalizadas, como que un código no se pueda repetir o que deba seguir un patrón concreto. El único caso que no se ha probado ha sido el de poner una URL de más de 255 caracteres, pues esto provoca un error 500.
* **create-right-role-right-user-wrong-action.hack:** un manager intenta crear un proyecto introduciendo datos de Script y SQL injection, sin conseguir realizar el hacking. Además, introduce un enlace de más de 255 caracteres en el campo correspondiente y recibe un error 500 como respuesta.
* **create-wrong-role.hack:** un usuario anónimo introduce la URL de creación de un proyecto y recibe un error 500 como respuesta.

### Delete

* **delete.safe:** varios managers borran sus proyectos no publicados. Como esta funcionalidad no tiene ningún requisito más allá de que el usuario sea el manager propietario del proyecto, no tiene mayor complejidad.
* **delete-wrong-role.hack:** un usuario anónimo introduce la URL de borrado de un proyecto y recibe un error 500 como respuesta ya que no tiene el rol “manager” que necesita para poder acceder a la URL.

### List

* **list.safe:** todos los managers listan sus proyectos. Aquellos cuyo listado no esté vacío, utilizan los índices de navegación para ver distintas páginas de la lista y la barra de búsqueda para filtrar las entradas.
* **list-wrong-role.hack:** un usuario anónimo introduce la URL de listado de proyectos y recibe un error 500 como respuesta ya que no tiene el rol “manager” que necesita para poder acceder a la URL.

### Publish

* **publish.safe:** varios managers publican sus proyectos en modo borrador. Además de probar para cada atributo sus valores posibles y no posibles (un publish es prácticamente igual que un update y necesita hacer las mismas comprobaciones), se han probado las siguientes opciones: que no se pueda publicar un proyecto con errores fatales, sin HUs o con HUs sin publicar, y que se pueda publicar un proyecto en cualquier otro caso. Esta prueba ha servido para detectar un bug que permitía publicar un proyecto que tuviera errores fatales y se ha podido subsanar el error.
* **publish-right-role-right-user-wrong-action.hack:** un manager intenta publicar un proyecto introduciendo datos de Script y SQL injection, sin conseguir realizar el hacking. Además, introduce un enlace de más de 255 caracteres en el campo correspondiente y recibe un error 500 como respuesta.
* **publish-wrong-role.hack:** un usuario anónimo introduce la URL de publicación de un proyecto y recibe un error 500 como respuesta ya que no tiene el rol “manager” que necesita para poder acceder a la URL.

### Show

* **show.safe:** varios managers muestran sus proyectos. Se ha probado a mostrar tanto proyectos publicados como en modo borrador.
* **show-right-role-right-user-wrong-action.hack:** un manager intenta mostrar un proyecto inexistente.
* **show-right-role-wrong-user.hack:** el manager2 intenta mostrar un proyecto de manager1, recibiendo como respuesta un error que le indica que no está autorizado para realizar esa acción.
* **show-wrong-role.hack:** un usuario anónimo intenta mostrar un proyecto como si fuera un manager, para lo que no está autorizado.

### Update

* **update.safe:** varios managers actualizan sus proyectos en modo borrador. Para cada atributo, se han probado todos los casos posibles y no posibles, con el fin de comprobar que salta el error adecuado. Se han tenido en cuenta las restricciones personalizadas, como que un código no se pueda repetir o que deba seguir un patrón concreto. El único caso que no se ha probado ha sido el de poner una URL de más de 255 caracteres, pues esto provoca un error 500.
* **update-right-role-right-user-wrong-action.hack:** un manager intenta actualizar un proyecto introduciendo datos de Script y SQL injection, sin conseguir realizar el hacking. Además, introduce un enlace de más de 255 caracteres en el campo correspondiente y recibe un error 500 como respuesta.
* **update-wrong-role.hack:** un usuario anónimo introduce la URL de actualización de un proyecto y recibe un error 500 como respuesta ya que no tiene el rol “manager” que necesita para poder acceder a la URL.

## User Story

### Create

* **create.safe:** varios managers crean HUs nuevas. Para cada atributo, se han probado todos los casos posibles y no posibles, con el fin de comprobar que salta el error adecuado. Se han tenido en cuenta las restricciones personalizadas, como que el coste mínimo sea 1. El único caso que no se ha probado ha sido el de poner una URL de más de 255 caracteres, pues esto provoca un error 500.
* **create-right-role-right-user-wrong-action.hack:** un manager intenta crear una HU introduciendo datos de Script y SQL injection, sin conseguir realizar el hacking. Además, introduce un enlace de más de 255 caracteres en el campo correspondiente y recibe un error 500 como respuesta.
* **create-wrong-role.hack:** un usuario anónimo introduce la URL de creación de una HU y recibe un error 500 como respuesta ya que no tiene el rol “manager” que necesita para poder acceder a la URL.

### Delete

* **delete.safe:** varios managers borran sus HUs. Como esta funcionalidad no tiene ningún requisito más allá de que el usuario sea el manager propietario de la HU, no tiene mayor complejidad. Este test ha servido para detectar el siguiente bug: al borrar la HU, se borraban posteriormente las asignaciones que la tenían asociada; esto provocaba una incoherencia, puesto se guardaba una clave foránea inexistente en la base de datos. Se arregló cambiando el orden para que se borrasen primero las asignaciones y luego la HU.
* **delete-wrong-role.hack:** un usuario anónimo introduce la URL de borrado de una HU y recibe un error 500 como respuesta ya que no tiene el rol “manager” que necesita para poder acceder a la URL.

### List All

* **list-all.safe:** todos los managers listan sus HUs. Aquellos cuyo listado no esté vacío, utilizan los índices de navegación para ver distintas páginas de la lista y la barra de búsqueda para filtrar las entradas.
* **list-all-wrong-role.hack:** un usuario anónimo introduce la URL de listado de HUs y recibe un error 500 como respuesta.

### List

* **list.safe:** todos los managers listan sus HUs a través del botón presente en el formulario que muestra sus proyectos. Aquellos cuyo listado no esté vacío, utilizan los índices de navegación para ver distintas páginas de la lista y la barra de búsqueda para filtrar las entradas.
* **list-right-role-right-user-wrong-action.hack:** un manager intenta listar las HUs de un proyecto inexistente.
* **list-right-role-wrong-user.hack:** un manager intenta listar las HUs de un proyecto que no le pertenece.
* **list-wrong-role.hack:** un usuario anónimo introduce la URL de listado de HUs de un proyecto existente y recibe un error 500 como respuesta ya que no tiene el rol “manager” que necesita para poder acceder a la URL.

### Publish

* **publish.safe:** varios managers publican sus HUs en modo borrador. Para cada atributo, se han probado todos los casos posibles y no posibles, con el fin de comprobar que salta el error adecuado. Se han tenido en cuenta las restricciones personalizadas, como que el coste mínimo sea 1. El único caso que no se ha probado ha sido el de poner una URL de más de 255 caracteres, pues esto provoca un error 500.
* **publish-right-role-right-user-wrong-action.hack:** un manager intenta publicar una HU introduciendo datos de Script y SQL injection, sin conseguir realizar el hacking. Además, introduce un enlace de más de 255 caracteres en el campo correspondiente y recibe un error 500 como respuesta.
* **publish-wrong-role.hack:** un usuario anónimo introduce la URL de publicación de una HU y recibe un error 500 como respuesta ya que no tiene el rol “manager” que necesita para poder acceder a la URL.

### Show

* **show.safe:** varios managers muestran sus HUs. Se ha probado a mostrar HUs publicadas y en modo borrador.
* **show-right-role-right-user-wrong-action.hack:** un manager intenta mostrar una HU inexistente.
* **show-right-role-wrong-user.hack:** el manager2 intenta mostrar una HU del manager1, recibiendo como respuesta un error que le indica que no está autorizado para realizar esa acción.
* **show-wrong-role.hack:** un usuario anónimo intenta mostrar una HU, para lo que no está autorizado.

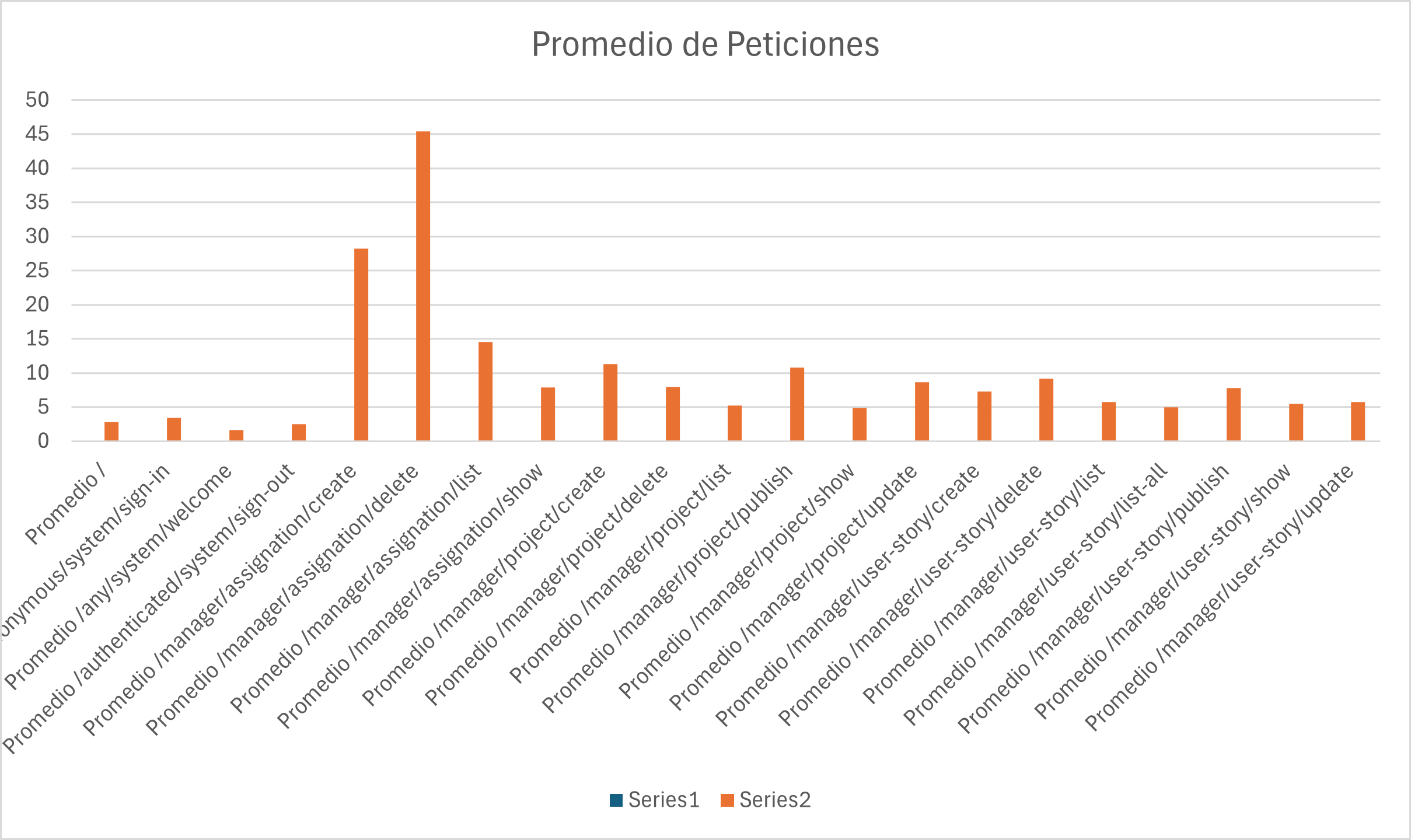
### Update

* **update.safe:** varios managers actualizan sus HUs en modo borrador. Para cada atributo, se han probado todos los casos posibles y no posibles, con el fin de comprobar que salta el error adecuado. Se han tenido en cuenta las restricciones personalizadas, como que el coste mínimo sea 1. El único caso que no se ha probado ha sido el de poner una URL de más de 255 caracteres, pues esto provoca un error 500.
* **update-right-role-right-user-wrong-action.hack:** un manager intenta actualizar una HU introduciendo datos de Script y SQL injection, sin conseguir realizar el hacking. Además, introduce un enlace de más de 255 caracteres en el campo correspondiente y recibe un error 500 como respuesta.
* **update-wrong-role.hack:** un usuario anónimo introduce la URL de actualización de una HU y recibe un error 500 como respuesta ya que no tiene el rol “manager” que necesita para poder acceder a la URL.

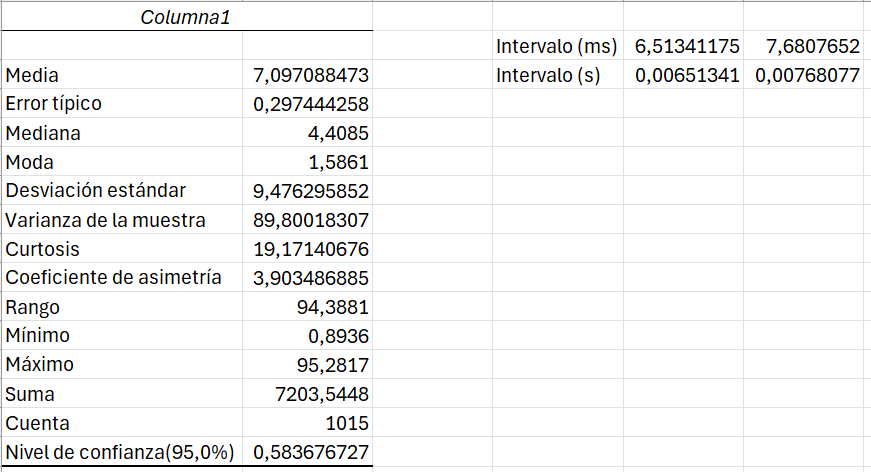
# Testing de Rendimiento

El testing del rendimiento de la aplicación se ha realizado analizando los tiempos de ejecución de las pruebas grabadas descritas en el capítulo anterior. Para ello, se ha realizado una primera ejecución sobre el código implementado en el tercer entregable (con los arreglos pertinentes tras descubrir bugs y errores y tras corregir el proyecto para la segunda convocatoria). Posteriormente, se han modificado las entidades para añadirles índices, lo que debería mejorar el rendimiento, y se ha realizado un segundo análisis. Los archivos con los resultados de los análisis pueden consultarse en la carpeta /reports/Student 1/tester-performance del proyecto.

## Primer análisis



Se puede observar en este gráfico que las peticiones más costosas, con diferencia, corresponden a las de creación y borrado de asignaciones, probablemente debido a que sus dos únicos atributos son relaciones con dos entidades de gran tamaño.

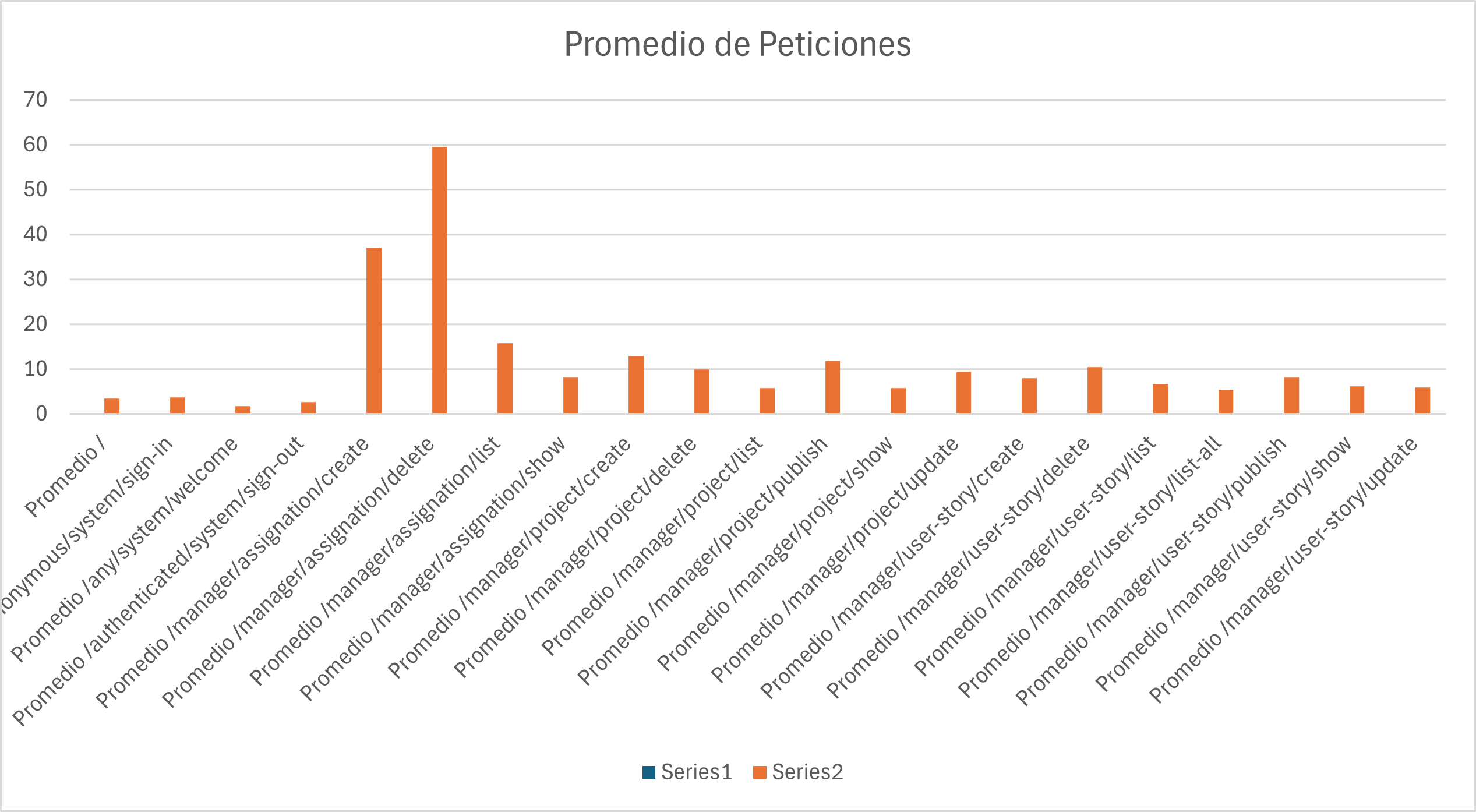
Esta imagen muestra una serie de variables estadísticas sobre los tiempos de todas las peticiones realizadas. Vemos que el intervalo de confianza es, aproximadamente, de [6.51, 7.69) milisegundos. Se trata de un muy buen intervalo de confianza, pues se encuentra muy por debajo del requisito de la asignatura de 1 segundo como máximo (en realidad, no es un requisito real, sino una sugerencia que tomar de referencia).

## Segundo análisis

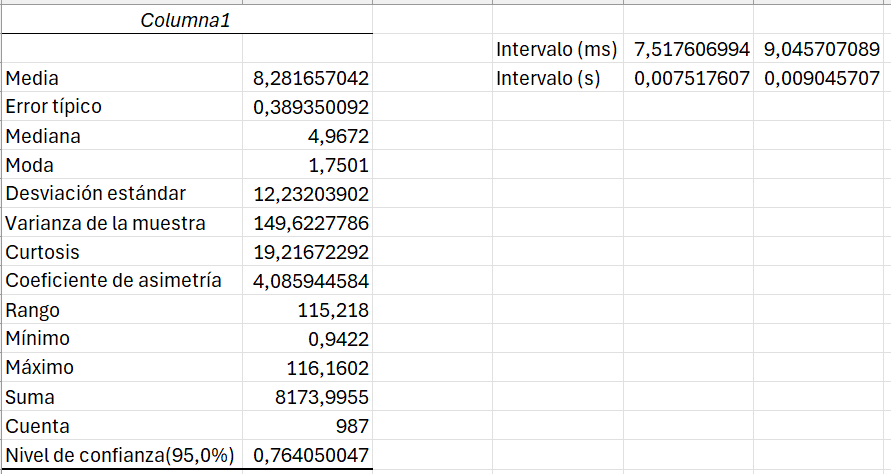
Antes de realizar el segundo análisis, se han añadido los siguientes índices en las clases Java correspondientes, tomando de referencia las consultas hechas en los repositorios de las entidades:

* **Assignation:** @Index(columnList = “project\_id,user\_story\_id”), @Index(columnList = “project\_id”), @Index(columnList = “user\_story\_id”)
* **Project:** @Index(columnList = “manager\_id”), @Index(columnList = “code”)
* **UserStory:** @Index(columnList = “manager\_id”)

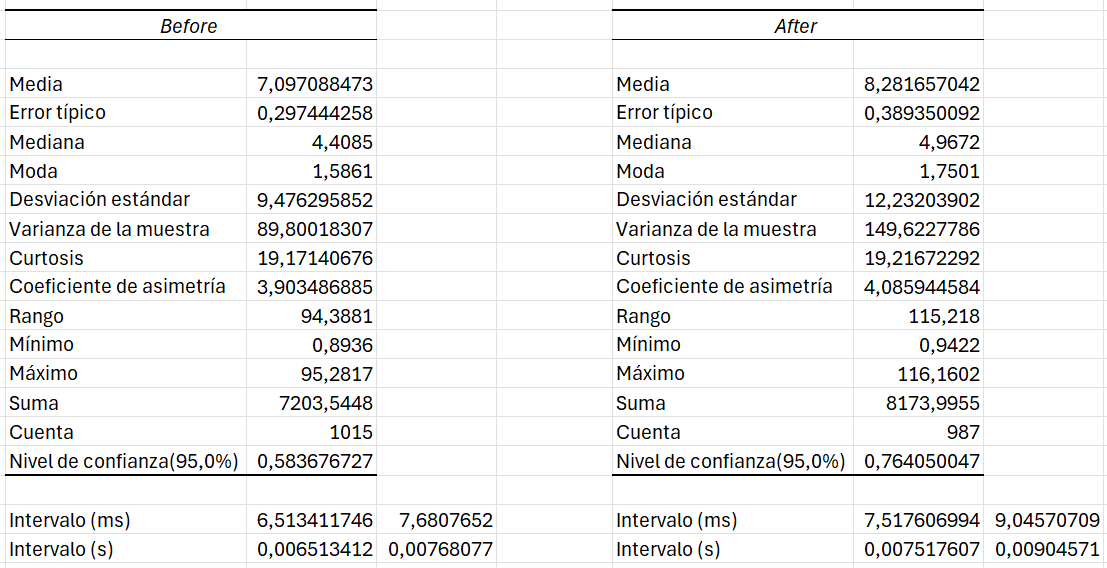
Ahora, los resultados obtenidos han sido los siguientes:



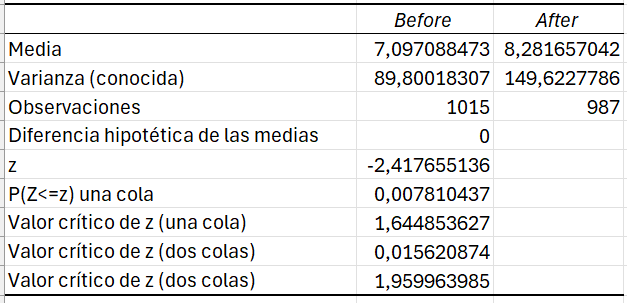
Vemos que, aunque las peticiones más costosas siguen siendo las mismas, estas han aumentado considerablemente su tiempo de ejecución, lo que es mala señal.

En las medidas estadísticas, observamos que ahora el intervalo de confianza está aproximadamente en el rango [7.51, 9.05) milisegundos. A pesar de haber empeorado respecto del intervalo de confianza anterior, sigue siendo muy buena medida, puesto que es muy inferior al límite de 1 segundo como máximo ya mencionado.

## Contraste



En esta imagen, se pueden ver una frente a otra las estadísticas de ambas ejecuciones. Aunque no sea muy evidente, podemos empezar a intuir que la segunda implementación no ha mejorado el rendimiento de la aplicación, sino que lo ha empeorado. Para verlo con más claridad, debemos realizar la “Prueba z para medias de dos muestras”:

La prueba se ha realizado con un valor de alfa igual a 0.05. La variable que nos interesa analizar es el valor crítico de z (dos colas), el primero de los dos que aparecen en la imagen, puesto que es el que va a determinar si nuestro análisis es concluyente o no. Vemos que dicho valor crítico se encuentra en el intervalo [0.00, alfa); por tanto, se pueden comparar las medias. Concluimos que la introducción de índices ha hecho que el rendimiento sea peor, porque el valor de la media del segundo análisis es mayor, por lo que se ha optado por eliminarlos.

Debido a que este es un buen contraste entre ambos análisis, como ya se ha explicado (valor crítico de z), se ha decidido que no es necesario realizar una tercera ejecución en otro equipo. Los resultados obtenidos son concluyentes.

# Conclusiones

Observando todas las pruebas realizadas, se puede decir que estas han cubierto todas las funcionalidades requeridas casi en su totalidad, quedando sin cubrir únicamente aquellos casos de hacking que no se puedan hacer desde la interfaz de la aplicación. Es el caso de enviar las peticiones de borrado, actualización y publicación con datos vacíos o de entidades que no pertenecen al usuario que hace la petición, por ejemplo.

Por otro lado, en cuanto al rendimiento de las ejecuciones, concluimos que el código, aunque aún perfeccionable, es lo suficientemente eficiente. Así lo indica el intervalo de confianza de la media de los tiempos empleados en realizar cada petición. Incluso en la implementación más ineficiente, la que incluía los índices, este intervalo de confianza era bastante positivo.

# Bibliografía

Transparencias de la asignatura DP2 del curso 2023/2024:

* D04: S01 – Formal testing
* D04: S02 – Performance testing