Testing Report Student #4
C2.019
Germán Sánchez Carmona

Grupo	C2.019
Repositorio	https://github.com/adolfoborrego/Acme-ANS
Student #2	**ID**: 77873179D **UVUS**: SSK0456 **Nombre**: Martínez Díaz, Ignacio **Roles**: Developer, Analyst, Tester
Student #4	**ID**: 52077055H **UVUS**: TCP2748 **Nombre**: Sánchez Carmona, Germán **Roles**: Project Manager, Developer , Analyst, Tester
Student #5	**ID**: 54794337B **UVUS**: CFV7375 **Nombre**: Regidor García, Miguel **Roles**: Operator, Developer , Analyst, Tester

Índice

1. Introducción

1.1. Propósito del documento

El propósito de este documento es presentar de forma estructurada y rigurosa los resultados obtenidos durante el proceso de pruebas del sistema desarrollado por el equipo. El informe tiene como objetivo principal verificar que las funcionalidades implementadas cumplen correctamente con los requisitos definidos y que el sistema ofrece un comportamiento estable, seguro y eficiente tanto a nivel funcional como de rendimiento.

A través de este documento se detalla la ejecución de pruebas funcionales, diferenciando entre versiones inseguras (.hack) y versiones protegidas (.safe), así como el análisis del rendimiento del sistema en diferentes condiciones, incluyendo comparativas estadísticas e intervalos de confianza. Este análisis permite no solo identificar errores o vulnerabilidades, sino también justificar decisiones técnicas tomadas durante el desarrollo para mejorar la calidad final del producto.

2. Functional Testing

2.1. Introducción

En este apartado se documenta la metodología seguida para realizar las pruebas funcionales (functional testing) de las distintas características implementadas. El objetivo principal de este tipo de pruebas es comprobar que las funcionalidades del sistema devuelven los resultados esperados, tanto en condiciones normales como anómalas, garantizando así la calidad del software desde el punto de vista del usuario final.

Siguiendo la metodología formal estudiada, se han diseñado y ejecutado casos de prueba positivos, negativos (.safe) y de hacking (.hack). Cada uno de estos tipos de pruebas ha sido elaborado respetando los principios de repetibilidad, control de datos y cobertura. Se ha prestado especial atención a la verificación de formularios de edición, la visualización de datos en listados y formularios, la gestión de entradas inválidas y la resistencia del sistema ante intentos de uso indebido.

En las siguientes secciones se detallan los casos de prueba realizados, así como la cobertura obtenida para las funcionalidades implementadas por el *Student #4*.

2.2. Casos de prueba

A continuación se detallan los casos de prueba realizados sobre la entidad *Claim*, desarrollada por los *AssistanceAgent*, cubriendo las funcionalidades principales asociadas a la gestión de reclamaciones. La tabla incluye tanto pruebas positivas como pruebas de hacking, con su correspondiente descripción y una valoración de su eficacia.

Caso de prueba	Descripción	Eficacia
list.safe	Se ha accedido los listados con un usuario logueado con el realm de AssistanceAgent.	No se han detectado errores.
list.hack	Se ha accedido los listados con un usuario logueado con un realm diferente a AssistanceAgent.	No se han detectado errores (Access not authorised).
show.safe	Se ha accedido a una claim desde el listado por parte del AssistanceAgent que la ha creado.	No se han detectado errores.
show.hack	Se ha intentado acceder a una claim perteneciente a otro AssistanceAgent.	Se encontraron fallos en la lógica del authorise() que no se habían tenido en cuenta, y se actualizaron todos los servicios.
create.safe	Se han probado múltiples envíos del formulario de creación de claim, incluyendo datos inválidos y vacíos para validar los mensajes de error y asegurar que no se produce un panic. Finalmente, se ha verificado que una claim con datos válidos se guarda correctamente.	No se han detectado errores.
create.hack	Se ha accedido directamente a la URL de creación desde un realm no autorizado; Se ha intentado modificar datos del formulario mediante herramientas de desarrollo (por ejemplo, manipulando campos SelectChoices con el inspector).	Se comprobó que era posible realizar POST hacking manipulando los valores del SelectChoices, que se arregló posteriormente para lanzar un acceso no autorizado en caso de detectar valores ilegales.
update.safe	Se ha probado la edición de una claim propia por parte del AssistanceAgent, incluyendo pruebas con datos inválidos y válidos para verificar la correcta validación del formulario y la ausencia de errores tipo panic.	No se han detectado errores.

update.hack	Se ha intentado actualizar una claim ya publicada (lo cual no está permitido), y también se ha accedido a la URL de edición con un realm incorrecto y se han manipulado datos del formulario mediante inspección de elementos.	No se han detectado errores (Access not authorised).
delete.safe	Se ha realizado correctamente el borrado de una claim no publicada por parte del AssistanceAgent que la creó.	No se han detectado errores.
delete.hack	Se ha intentado eliminar claims de otros AssistanceAgent, así como claims que ya estaban publicadas, manipulando directamente la URL.	No se han detectado errores (Access not authorised).
publish.safe	Se ha realizado la publicación de una claim no publicada, creada por el AssistanceAgent correspondiente, verificando que el sistema responde correctamente.	No se han detectado errores.
publish.hack	Se ha intentado publicar claims ya publicadas o que pertenecen a otros AssistanceAgent, accediendo directamente a la URL.	No se han detectado errores (Access not authorised).

En esta otra tabla se describen los casos de prueba realizados sobre las funcionalidades asociadas a la entidad *TrackingLog* dentro del rol de *AssistanceAgent*. Estas pruebas tienen como objetivo verificar que el sistema permite consultar correctamente los registros de seguimiento de cada reclamación, y que se impide el acceso o manipulación indebida de registros ajenos o no autorizados. Se incluyen tanto pruebas funcionales como intentos de acceso no permitido para garantizar la robustez y seguridad del sistema.

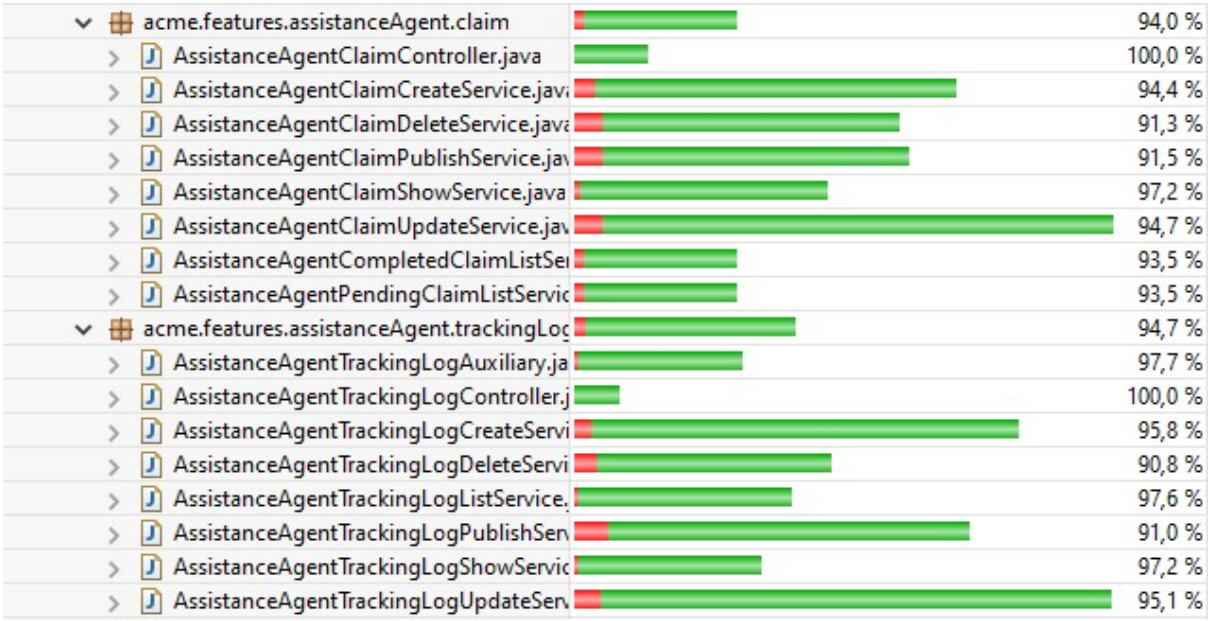
Caso de prueba	Descripción	Eficacia
list.safe	Se ha accedido al listado de tracking logs pertenecientes a una claim publicada del AssistanceAgent.	No se han detectado errores.
list.hack	Se ha intentado acceder al listado de tracking logs de una claim que no pertenece al AssistanceAgent o que no ha sido publicada.	No se han detectado errores (Access not authorised).
show.safe	Se ha accedido a la visualización de un tracking log propio asociado a una claim publicada.	No se han detectado errores.
show.hack	Se ha intentado visualizar tracking logs de otros AssistanceAgent o de claims no publicadas.	No se han detectado errores (Access not authorised).

create.safe	Se ha probado la creación de tracking logs con datos válidos e inválidos para validar los errores de formulario.	No se han detectado errores.
create.hack	Se ha intentado crear tracking logs en claims que aún no están publicadas, lo cual no está permitido, o para claims de otros AssistanceAgent.	No se han detectado errores (Access not authorised).
update.safe	Se han probado distintas ediciones válidas e inválidas sobre tracking logs propios, verificando la gestión de errores.	No se han detectado errores.
update.hack	Se han intentado actualizar tracking logs de otros usuarios, enviar valores inválidos o acceder directamente a la URL de edición.	No se han detectado errores (Access not authorised).
delete.safe	Se ha eliminado un tracking log propio correctamente, cumpliendo los requisitos establecidos.	No se han detectado errores.
delete.hack	Se ha intentado eliminar tracking logs ajenos o de claims no permitidas mediante manipulación de la URL.	No se han detectado errores (Access not authorised).
publish.safe	Se ha probado la publicación de un tracking log verificando que todas las condiciones del método validate() funcionan correctamente.	Se detectaron algunos problemas en el validate() del servicio correspondiente que fueron arreglados.
publish.hack	Se ha intentado publicar tracking logs ajenos o ya publicados manipulando directamente la URL.	No se han detectado errores (Access not authorised).

2.3. Cobertura de las pruebas

La siguiente tabla muestra la cobertura alcanzada por los paquetes principales asociados a las funcionalidades desarrolladas para *AssistanceAgent*. Esta cobertura se ha obtenido a partir de la ejecución de los casos de prueba descritos.

Paquete	Cobertura
assistance-agent.claim	94.0 %
assistance-agent.tracking-log	94.7 %



A continuación se presenta la cobertura obtenida por servicio dentro del paquete assistance-agent.claim, como resultado directo de los casos de prueba realizados sobre la entidad *Claim*.

Servicio	Cobertura
AssistanceAgentCompletedClaimListService	93.5 %
AssistanceAgentPendingClaimListService	93.5 %
AssistanceAgentClaimShowService	97.2 %
AssistanceAgentClaimCreateService	94.4 %
AssistanceAgentClaimUpdateService	94.7 %
AssistanceAgentClaimDeleteService	91.3 %
AssistanceAgentClaimPublishService	91.5 %

Esta otra tabla detalla la cobertura alcanzada por los servicios del paquete assistance-agent.tracking-log (*), derivada de los casos de prueba diseñados para *TrackingLog*.

Servicio	Cobertura
AssistanceAgentTrackingLogListService	97.6 %
AssistanceAgentTrackingLogShowService	97.2 %
AssistanceAgentTrackingLogCreateService	95.8 %
AssistanceAgentTrackingLogUpdateService	95.1 %
AssistanceAgentTrackingLogDeleteService	90.8 %
AssistanceAgentTrackingLogPublishService	91.0 %

(*) Se ha implementado una clase auxiliar llamada AssistanceAgentTrackingLogAuxiliary.java con una cobertura del 97.7%, la cual no se ha añadido a las tablas por no ser un Service.

2.4. Conclusiones

A lo largo del proceso de functional testing, se han diseñado y ejecutado numerosos casos de prueba que abarcan tanto situaciones normales como excepcionales, incluyendo intentos de acceso no autorizado (hacking). Estos casos se han aplicado sobre las funcionalidades desarrolladas para los roles de AssistanceAgent en las entidades Claim y TrackingLog.

Las pruebas han demostrado ser eficaces no solo para validar el comportamiento esperado del sistema ante entradas válidas, sino también para identificar errores lógicos en los métodos de autorización y validación. En particular, se detectaron fallos en la validación de tracking logs durante el proceso de publicación, los cuales fueron corregidos a raíz de los resultados de las pruebas.

Además, la cobertura alcanzada como resultado de la ejecución de estos casos ha sido notablemente alta. En concreto, se ha superado el 90 % de cobertura en todos los servicios, y se han alcanzado cifras cercanas al 95 % en la mayoría de ellos. Esto indica que los test realizados han ejercitado prácticamente todas las instrucciones relevantes del código, lo que refuerza la fiabilidad y robustez de las funcionalidades implementadas.

En resumen, el proceso de pruebas funcionales ha sido clave para garantizar la calidad del sistema, permitiendo detectar errores, reforzar los controles de seguridad y confirmar que los requisitos funcionales se han cumplido adecuadamente.

3. Performance Testing

3.1. Introducción

El propósito de este apartado es analizar el rendimiento del sistema mediante pruebas de tipo performance testing. A diferencia de las pruebas funcionales, cuyo objetivo principal es verificar el comportamiento correcto del sistema, las pruebas de rendimiento se centran en medir el tiempo de respuesta de las distintas funcionalidades bajo condiciones controladas.

Para llevar a cabo este análisis, se han reutilizado los casos de prueba funcionales ya existentes, reproduciéndolos (replay) en dos equipos portátiles diferentes. De este modo, se ha obtenido un conjunto de datos reales de ejecución a partir de los cuales se ha podido evaluar la eficiencia del sistema y comparar el rendimiento relativo de ambos entornos.

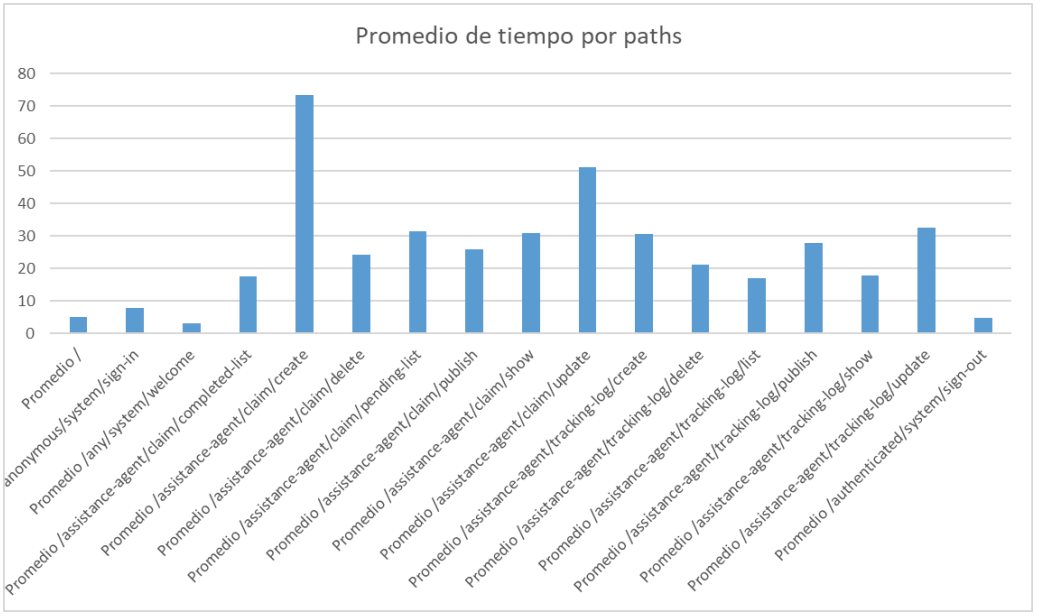
El análisis se ha realizado siguiendo la metodología propuesta, que incluye el cálculo de intervalos de confianza para el tiempo medio de respuesta y la aplicación de una prueba Z para dos muestras independientes, con el objetivo de determinar si las diferencias observadas entre ambos portátiles son estadísticamente significativas.

En las siguientes secciones se presentan los resultados obtenidos, tanto en forma de gráficos de eficiencia media como de análisis estadísticos que permiten extraer conclusiones fundamentadas sobre el rendimiento de la aplicación.

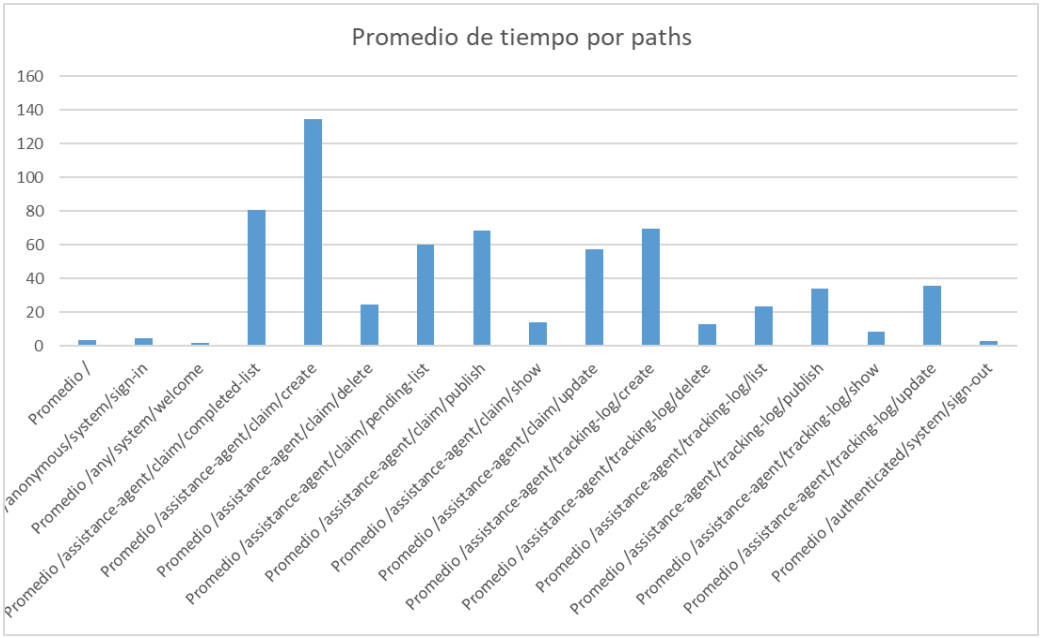
3.2. Gráficos de eficiencia medios

Con el objetivo de analizar visualmente el rendimiento del sistema, se han generado gráficos de eficiencia que muestran el tiempo medio de respuesta por funcionalidad (feature).

Estos gráficos permiten identificar de forma clara las funcionalidades que presentan un mayor coste computacional, denominadas MIR (Most Inefficient Request), las cuales podrían convertirse en cuellos de botella en un entorno real. En ambos casos, las funcionalidades create y update de claim y tracking-log fueron las menos eficientes.



Computer 1



Computer 2

3.3. Estadísticas descriptivas

Como paso previo al contraste de hipótesis, se ha realizado un análisis estadístico descriptivo de los tiempos de respuesta registrados al ejecutar los casos de prueba en ambos portátiles. Este análisis permite obtener una visión general del comportamiento del sistema en cada equipo y calcular el intervalo de confianza para el tiempo medio de respuesta.

Computer 1		
Media	17,8473395	
Error típico	0,79919613	
Mediana	11,9294	
Moda	2,9203	
Desviación estándar	21,6816474	
Varianza de la muestra	470,093833	
Curtosis	22,2520326	
Coeficiente de asimetría	3,63559316	
Rango	236,3699	
Mínimo	2,1094	
Máximo	238,4793	
Suma	13135,6419	
Cuenta	736	
Nivel de confianza(95,0%)	1,56897927	
Interval (ms)	19,4163188	16,2783603
Interval (s)	0,01941632	0,01627836

Computer 2		
Media	24,7932067	
Error típico	3,74249738	
Mediana	6,6647	
Moda	1,3305	
Desviación estándar	101,531409	
Varianza de la muestra	10308,627	
Curtosis	61,0862605	
Coeficiente de asimetría	7,51587101	
Rango	1154,4673	
Mínimo	1,1541	
Máximo	1155,6214	
Suma	18247,8001	
Cuenta	736	
Nivel de confianza(95,0%)	7,34725887	
Interval (ms)	32,1404655	17,4459478
Interval (s)	0,03214047	0,01744595

Ambos portátiles cumplen con el supuesto de que la media de respuesta se mantiene muy por debajo del segundo (1 s), cumpliendo así el requisito de rendimiento utilizado como referencia.

3.4. Hipótesis y conclusiones

Tras obtener los intervalos de confianza y las estadísticas descriptivas de ambos portátiles, se ha realizado un contraste de hipótesis mediante una prueba Z para dos muestras independientes, con el objetivo de determinar si la diferencia entre los tiempos medios de respuesta es estadísticamente significativa. La prueba Z se ha aplicado sobre los conjuntos de datos recogidos en Computer 1 y Computer 2, considerando un nivel de significación del 5 % (α = 0.05).

Prueba z para medias de dos muestras		
	Computer 1	*Computer 2*
Media	17,8473395	24,7932067
Varianza (conocida)	470,093833	10308,627
Observaciones	736	736
Diferencia hipotética de las n	0	
z	-1,81502147	
P(Z<=z) una cola	0,03476031	
Valor crítico de z (una cola)	1,64485363	
P(Z<=z) dos colas	0,06952061	
Valor crítico de z (dos colas)	1,95996398	

Dado que el valor de P(Z<=z) dos colas se encuentra en el intervalo (α,1], podemos afirmar que no existen grandes diferencias en el rendimiento entre ambos portátiles.

4. Historial de versiones

Versión	Fecha	Descripción
0.0	25/05/2025	Borrador inicial.
1.0	26/05/2025	Correcciones generales en todos los apartados.
2.0	01/07/2025	Actualización del documento con los resultados de los tests generados para la segunda convocatoria.

5. Bibliografía

Intentionally Blank.