

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

## TESTING REPORT



Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas II

Curso 2024 – 2025

Grupo C2.021 - Student 4		
Autor	Correo	Rol
Paula Sánchez Gómez	<a href="mailto:pausangom1@alum.us.es">pausangom1@alum.us.es</a>	Desarrollador/Tester/Manager

Repositorio: <https://github.com/DP2-C1-021/Acme-ANS-C2>

02 de julio de 2025

**Control de versiones**

<b>Versión</b>	<b>Fecha</b>	<b>Descripción</b>
v1.0	18/02/2025	Primera versión del documento
v2.0	02/07/2025	Segunda versión del documento

# Índice

1. Resumen ejecutivo	4
2. Introducción	4
3. Testing Funcional	5
a. Listar claims completadas de un AssistanceAgent	5
b. Listar claims pendientes de un AssistanceAgent	5
c. Mostrar detalles de una claim	6
d. Crear una claim	7
e. Eliminar una claim	8
f. Editar una claim	9
g. Publicar una claim	10
h. Listar tracking logs de una claim de un Assistance Agent	11
i. Mostrar detalles de un tracking log	12
j. Crear un tracking log	13
k. Eliminar un tracking log	14
l. Editar un tracking log	15
m. Publicar un tracking log	16
n. Crear tracking log excepcional	17
4. Datos obtenidos	18
Porcentaje de cobertura para las funcionalidades de Claim y TrackingLog	18
5. Pruebas de rendimiento	21
6. Pruebas de mutaciones	25
7. Conclusiones	27

# 1. Resumen ejecutivo

Este informe documenta el proceso de verificación y validación de las funcionalidades implementadas para el rol de **Assistance Agent** en el sistema *Acme ANS*, centrado en las operaciones sobre las entidades Claim y TrackingLog, de acuerdo con los requisitos funcionales #8 y #9.

Se han ejecutado pruebas funcionales divididas en casos **positivos, negativos y de hacking**, cubriendo todos los escenarios de uso previstos, desde la creación y edición hasta la publicación y eliminación de reclamaciones y registros de seguimiento.

Asimismo, se ha llevado a cabo un análisis de rendimiento utilizando trazas generadas con *ReplayerLauncher*, y pruebas de mutaciones.

Los resultados obtenidos evidencian que las funcionalidades desarrolladas son estables, seguras y eficientes, demostrando un correcto cumplimiento de los requisitos funcionales y no funcionales establecidos para la entrega D04.

## 2. Introducción

El propósito de este documento es presentar de forma estructurada los resultados del proceso de testing aplicado al desarrollo de funcionalidades para el rol de *Assistance Agent* en el sistema *Acme ANS*. En concreto, se evalúan las operaciones definidas en los **requisitos funcionales 8 y 9**, que incluyen el ciclo completo de gestión de Claims y TrackingLogs.

El testing funcional se ha realizado mediante pruebas, ejecutadas bajo distintos perfiles de usuario para validar restricciones de acceso, reglas de negocio y validaciones en formularios. Estas pruebas se han clasificado en:

- **Casos positivos**, para confirmar el comportamiento esperado.
- **Casos negativos**, donde se introducen valores inválidos.
- **Casos de hacking**, que simulan intentos de acceso no autorizado.

Complementariamente, se ha aplicado testing de rendimiento mediante el uso de *tester.trace* y su posterior análisis con herramientas estadísticas en Excel. También, se ha aplicado el testing de mutaciones.

Este informe detalla los **casos de prueba**, sus resultados, la **cobertura de código**, el **análisis estadístico** y la interpretación de la **prueba Z**, culminando en las **conclusiones** sobre la calidad y eficiencia del sistema.

### 3. Testing Funcional

#### a. Listar claims completadas de un AssistanceAgent

##### **Descripción:**

El objetivo de este test es comprobar que el agente de asistencia puede acceder correctamente al listado de sus claims completadas (aceptadas o rechazadas). También se comprueba que no se muestre correctamente si no hay ninguna claim completada.

##### **Archivo .safe – Casos positivos**

Se accede con un usuario correctamente autenticado como AssistanceAgent (por ejemplo, agent1) y se lista la sección de claims completadas. El sistema responde mostrando correctamente los reclamos que ya han sido finalizados, es decir, aquellos cuyo estado es “Accepted” o “Rejected”. Se validó que todos los datos relevantes aparecen correctamente en pantalla (tipo de claim, fecha, estado final...).

##### **Archivo .safe – Casos negativos**

Dentro del mismo test .safe, se simulan situaciones donde: El agente no tiene ninguna claim completada. El sistema reacciona de forma controlada: mostrando una lista vacía cuando no hay datos. No se lanza ningún error crítico y se respeta el comportamiento esperado.

##### **Archivo .hack – Casos de hacking**

En el fichero list-completed-claim.hack se prueban accesos no autorizados. Por ejemplo:

- Un usuario de otro rol (como administrator) intenta acceder a la URL de listado de claims completadas.

En todos los casos, el sistema responde con un mensaje de error del tipo “Access is not authorized”, bloqueando adecuadamente cualquier intento de acceso indebido. Esto confirma que las restricciones de seguridad están correctamente implementadas.

**Conclusión:** Esta funcionalidad ha sido probada exhaustivamente con casos válidos, inválidos y ataques de tipo hacking. El sistema responde como se espera en todos los casos. No se han detectado errores y se considera que cumple completamente con el requisito funcional correspondiente.

#### b. Listar claims pendientes de un AssistanceAgent

##### **Descripción:**

El objetivo de este test es comprobar que el agente de asistencia puede acceder correctamente al listado de sus claims que aún no han sido completadas, es decir, aquellas que siguen en estado pendiente de resolución. También se valida que si no hay ninguna claim pendiente, el sistema muestre correctamente una lista vacía sin errores.

##### **Archivo .safe – Casos positivos**

Se accede con un usuario autenticado como AssistanceAgent (por ejemplo, agent1) y se navega al apartado de claims pendientes. El sistema muestra únicamente aquellas reclamaciones que todavía no han sido aceptadas ni rechazadas. Se verifica que el contenido se carga correctamente y que los datos visibles (tipo de claim, fecha, etc.) son los esperados.

#### **Archivo .safe – Casos negativos**

Dentro del mismo test .safe, se contemplan escenarios como que el agente no tenga ninguna claim pendiente. En este caso, el sistema responde mostrando correctamente una lista vacía, sin lanzar errores ni mostrar datos incorrectos. También se validó que la interfaz se mantiene estable y no aparecen elementos de otras secciones.

#### **Archivo .hack – Casos de hacking**

En el fichero list-undergoing-claim.hack se simulan accesos no autorizados. Por ejemplo:

- Un usuario de otro rol (como administrator) intenta acceder directamente al listado de claims pendientes de un AssistanceAgent.

En ambos casos, el sistema muestra un mensaje claro de “Access is not authorized”, denegando correctamente el acceso y demostrando que las restricciones de seguridad están correctamente aplicadas.

**Conclusión:** La funcionalidad ha sido verificada con distintos tipos de entradas: casos válidos, situaciones sin datos y pruebas de hacking. El sistema actúa correctamente en todos los casos, mostrando las claims pendientes únicamente al agente autenticado y bloqueando accesos indebidos. No se han detectado errores, por lo que se considera que cumple con los requisitos funcionales establecidos.

### **c. Mostrar detalles de una claim**

#### **Descripción:**

El objetivo de este test es comprobar que un AssistanceAgent puede visualizar correctamente toda la información asociada a una claim específica que ha sido creada por él. Además, se validan comportamientos ante errores como intentar acceder a una claim inexistente o a una que no le pertenece.

#### **Archivo .safe – Casos positivos**

Se accede con un usuario autenticado como AssistanceAgent y se selecciona una claim propia desde el listado. El sistema muestra correctamente todos los campos relevantes de la claim: tipo, descripción, pasajero, vuelo, fecha, estado, entre otros. Se comprobó que los datos corresponden exactamente con los valores registrados y que no hay errores de carga ni visualización.

#### **Archivo .safe – Casos negativos**

No se realizan pruebas negativas explícitas en este test. El archivo está centrado únicamente en comprobar que el acceso a los detalles de una claim válida, propiedad del agente, funciona correctamente.

### **Archivo .hack – Casos de hacking**

En el fichero show-claim.hack se intenta acceder a los detalles de una claim de otro agente de asistencia o desde un rol no autorizado. Por ejemplo:

- Un AssistanceAgent accede a la URL de una claim que no es suya.
- Un usuario sin permisos intenta forzar la vista de detalles de una claim.

En todos los casos, el sistema bloquea el acceso y muestra el mensaje “Access is not authorized”. Esta respuesta confirma que las reglas de control de acceso están implementadas correctamente y que no es posible visualizar información de otras claims fuera del alcance del usuario autenticado.

**Conclusión:** La funcionalidad ha sido verificada con pruebas completas: visualización correcta de datos, gestión de errores y protección contra accesos indebidos. El sistema actúa como se espera en todos los escenarios y cumple con los requisitos funcionales establecidos, garantizando tanto la usabilidad como la seguridad.

## **d. Crear una claim**

### **Descripción:**

Esta funcionalidad permite a un AssistanceAgent crear una nueva claim asociada a un tramo de vuelo válido. El objetivo del test es comprobar que el sistema permite registrar correctamente una nueva reclamación con datos válidos, y que reacciona adecuadamente ante errores de validación y accesos no autorizados.

### **Archivo .safe – Casos positivos**

El agente de asistencia inicia sesión correctamente y accede al formulario de creación de claims. Se introducen todos los datos requeridos de forma válida (tipo, pasajero, vuelo, descripción...). El sistema procesa el formulario, guarda la nueva claim y redirige al listado, confirmando que la operación se ha realizado con éxito.

### **Archivo .safe – Casos negativos**

Dentro del mismo archivo .safe se prueban múltiples combinaciones de entrada inválida:

- Envío del formulario con campos vacíos.
- Valores no válidos.
- Datos que no cumplen las validaciones definidas.

En todos los casos, el sistema **rechaza la creación de la claim**, muestra los mensajes de error correspondientes en pantalla y **no se guarda ningún dato incorrecto**. Se verifica que

las validaciones tanto del lado del cliente como del servidor están funcionando como se espera.

### **Archivo .hack – Casos de hacking**

En create-claim.hack se prueba el intento de acceso al formulario o la acción de crear una claim sin tener el rol adecuado:

- Un usuario que no es AssistanceAgent intenta acceder directamente a la URL de creación.
- Se intenta crear una claim con leg = 404, AAA1144: el leg no está público, el vuelo sí.
- Se intenta crear una claim con leg = 407, AAA1147, el vuelo no está público, el leg sí.
- Si se intenta con una claim que es nula: 999.
- Se intenta crear una claim con leg = 403, AAA9999, no está público ni el leg ni el flight.
- Se intenta hackear los campos de registrationMoment, type e indicator.

En ambos casos, el sistema responde correctamente con el mensaje “Access is not authorized”, impidiendo la operación. En el caso de registrationMoment e indicator, el sistema simplemente lo ignora. En el caso de type, el sistema lanza un Invalid valued. Esto demuestra que los permisos están correctamente controlados.

**Conclusión:** La funcionalidad de creación de claims ha sido probada con múltiples entradas válidas e inválidas, así como con accesos no autorizados. El sistema reacciona como se espera en todos los casos, asegurando tanto la calidad de los datos introducidos como la seguridad de acceso. No se han detectado errores, por lo que se considera que cumple completamente con los requisitos funcionales establecidos.

## **e. Eliminar una claim**

### **Descripción:**

Esta funcionalidad permite a un AssistanceAgent eliminar una claim que haya creado, siempre y cuando no haya sido publicada. El objetivo de este test es comprobar que se pueden eliminar correctamente las claims propias en estado no publicado, y que el sistema bloquea los intentos no válidos o no autorizados.

### **Archivo .safe – Casos positivos**

Se inicia sesión con un AssistanceAgent (por ejemplo, agent1) y se elimina una de sus claims que aún no ha sido publicada. El sistema realiza la operación correctamente, eliminando la claim de la base de datos y redirigiendo al listado sin mostrar errores. Se confirma que la claim ya no aparece en el sistema tras la acción.

### **Archivo .safe – Casos negativos**



No se realizan pruebas negativas explícitas en este test. El archivo está centrado únicamente en comprobar que se elimina la claim del Assistance Agent, funciona correctamente.

### **Archivo .hack – Casos de hacking**

En delete-claim.hack se prueban intentos de eliminar una claim sin los permisos adecuados:

- Un AssistanceAgent intenta eliminar una claim que no ha creado, manipulando la URL o el ID.
- Si se intenta eliminar una claim que ya ha sido publicada, el sistema no permite la operación.
- Un usuario de otro rol (como administrator o anonymous) intenta forzar la eliminación.
- Si se intenta eliminar una claim que es nula.

En todos los casos, el sistema responde correctamente con el mensaje “Access is not authorized”.

**Conclusión:** La funcionalidad de eliminación de claims ha sido probada de forma exhaustiva. El sistema permite eliminar correctamente las claims válidas, impide la eliminación de claims publicadas o ajenas, y gestiona los errores de manera controlada. Se considera que cumple completamente con los requisitos funcionales establecidos.

## **f. Editar una claim**

### **Descripción:**

Esta funcionalidad permite a un AssistanceAgent modificar los datos de una claim que haya creado previamente, siempre que no esté publicada. El objetivo del test es comprobar que se pueden actualizar correctamente los campos cuando los datos son válidos, que las validaciones funcionan adecuadamente ante errores de entrada, y que se bloquean los accesos indebidos.

### **Archivo .safe – Casos positivos**

El usuario inicia sesión como AssistanceAgent (por ejemplo, agent1), accede al formulario de edición de una claim propia **no publicada** y modifica correctamente los campos (tipo, pasajero, vuelo, descripción...). Tras enviar el formulario, el sistema actualiza la información y redirige a la vista de detalles sin errores. Se confirma que los cambios han sido guardados.

### **Archivo .safe – Casos negativos**

Dentro del mismo test .safe se prueba:

- Enviar el formulario con campos vacíos o mal formateados.
- Introducir datos que no cumplen las restricciones definidas.

El sistema reacciona correctamente: muestra mensajes de error en el formulario, **no guarda cambios** y mantiene los datos originales. Se valida así que las restricciones de integridad y formato están bien implementadas.

### **Archivo .hack – Casos de hacking**

En update-claim.hack se simulan intentos de editar una claim de forma no autorizada:

- Un AssistanceAgent intenta editar una claim que no le pertenece. - assistance-agent/claim/update?id=555
- Un usuario de otro rol accede directamente al formulario de edición. assistance-agent/claim/update , assistance-agent/claim/update?id=555
- Se intenta modificar una claim ya publicada. - assistance-agent/claim/update?id=556
- Se intenta crear una claim con leg = 404, AAA1144: el leg no está público, el vuelo sí.
- Se intenta crear una claim con leg = 407, AAA1147, el vuelo no está público, el leg sí.
- Si se intenta con una claim que es nula: 999.
- Se intenta crear una claim con leg = 403, AAA9999, no está público ni el leg ni el flight.
- Se intenta hackear los campos de registrationMoment, type e indicator.

En todos los casos, el sistema bloquea la operación con un mensaje del tipo “Access is not authorized”. No se permite acceder al formulario ni modificar la información. Se valida que los controles de seguridad funcionan correctamente.

**Conclusión:** La funcionalidad de edición de claims ha sido verificada con datos válidos, inválidos y con pruebas de acceso indebido. El sistema actúa como se espera en todos los casos: actualiza cuando procede, valida correctamente, y protege el recurso ante accesos ilegítimos. Se considera que cumple totalmente con los requisitos funcionales definidos.

## **g. Publicar una claim**

### **Descripción:**

Esta funcionalidad permite a un AssistanceAgent publicar una claim que ha creado y que aún no ha sido publicada. El objetivo del test es comprobar que la acción de publicación se realiza correctamente cuando las condiciones son válidas, y que el sistema bloquea los intentos no autorizados o ilegítimos.

### **Archivo .safe – Casos positivos**

Un usuario autenticado como AssistanceAgent (por ejemplo, agent1) accede a una claim propia en estado no publicado y ejecuta la acción de publicar. El sistema realiza el cambio de estado correctamente, actualizando la claim a “published” y redirigiendo a la vista de detalle o al listado. No se detectan errores en el proceso, y la claim ya no puede editarse o eliminarse después.

### **Archivo .safe – Casos negativos**

En este test no se simulan casos negativos explícitos como intentar publicar una claim ya publicada. El archivo se centra únicamente en verificar el caso de uso exitoso con una claim válida y propia. No obstante, el sistema actúa correctamente y no lanza excepciones inesperadas.

### **Archivo .hack – Casos de hacking**

En publish-claim.hack se prueba lo siguiente:

- Un AssistanceAgent intenta publicar una claim que no le pertenece.
- Un usuario de otro rol (como administrator) accede directamente a la acción de publicación.
- Si se intenta con una claim no válida, por ejemplo 9999.

En todos estos escenarios, el sistema responde con el mensaje “Access is not authorized” y bloquea la acción, sin modificar el estado de la claim. Se valida que los controles de acceso están correctamente definidos.

**Conclusión:** La funcionalidad de publicación de claims ha sido probada con éxito en escenarios válidos y ante intentos de acceso indebido. El sistema actúa de forma robusta y segura, permitiendo publicar solo las claims propias no publicadas, y rechazando cualquier intento de manipulación externa. Cumple con los requisitos funcionales establecidos.

## **h. Listar tracking logs de una claim de un Assistance Agent**

### **Descripción:**

Esta funcionalidad permite a un AssistanceAgent visualizar todos los registros de seguimiento (tracking logs) asociados a una claim concreta que él mismo haya creado. El objetivo de este test es comprobar que se muestran correctamente los logs cuando la claim es legítima y que se impide el acceso en caso contrario.

### **Archivo .safe – Casos positivos**

Un usuario autenticado como **AssistanceAgent** accede al listado de tracking logs de una de sus propias claims. El sistema responde correctamente mostrando todos los logs registrados, incluyendo información como el título, el cuerpo del mensaje, la fecha de creación y el estado de publicación. Se valida lo siguiente:

- **Logs "noMore":** Se valida que los logs que corresponden a casos "noMore" (es decir, aquellos en los que no hay más acciones posibles) se gestionen de manera correcta, asegurando que no se presenten en el listado de forma inapropiada.
- **Logs excepcionales:** En este caso, se valida si en la lista tienes la opción de poder crear un tracking log excepcional.

- **Logs normales:** Los logs comunes y normales se muestran según lo esperado, permitiendo que el agente de asistencia revise el progreso de la reclamación sin problemas.

Se valida que solo se muestran los logs de la reclamación correspondiente, y que el formato de presentación es el correcto.

#### **Archivo .safe – Casos negativos**

Dentro del mismo test .safe, se simulan situaciones donde: El agente no tiene ningún tracking log en una claim. El sistema reacciona de forma controlada: mostrando una lista vacía cuando no hay datos. No se lanza ningún error crítico y se respeta el comportamiento esperado.

#### **Archivo .hack – Casos de hacking**

En list-tracking-log.hack se prueba el acceso no autorizado al listado de logs de una claim:

- Un AssistanceAgent intenta listar los logs de una claim que no le pertenece.
- Un AssistanceAgent intenta listar los logs de una claim no válida.
- Un usuario de otro rol o no autenticado intenta acceder directamente a la funcionalidad.

En todos los casos, el sistema responde con el mensaje “Access is not authorized” y no permite ver ningún dato. Esto confirma que los filtros de acceso están correctamente definidos.

#### **Conclusión:**

La funcionalidad ha sido validada correctamente en escenarios legítimos y de acceso indebido. El sistema muestra únicamente los logs de claims propias y bloquea los intentos externos. No se han detectado errores, y se cumple el requisito funcional establecido.

## **i. Mostrar detalles de un tracking log**

#### **Descripción:**

Esta funcionalidad permite a un AssistanceAgent visualizar el contenido completo de un tracking log específico, asociado a una claim que él mismo haya creado. El objetivo del test es comprobar que el sistema muestra correctamente los datos cuando se accede a un log propio, y que protege esta información frente a accesos indebidos.

#### **Archivo .safe – Casos positivos**

Se inicia sesión con un usuario AssistanceAgent (por ejemplo, agent1) y se accede al detalle de un tracking log asociado a una de sus claims. El sistema responde correctamente mostrando el contenido completo del log: título, cuerpo del mensaje, fecha de creación y estado de publicación. No se detectan errores de carga ni problemas de visualización.

#### **Archivo .safe – Casos negativos**

No se han realizado pruebas negativas dentro de este archivo. El test se centra únicamente en el caso exitoso de visualización del log propio. El comportamiento del sistema es estable en todo momento.

### **Archivo .hack – Casos de hacking**

En el archivo show-tracking-log.hack se simulan intentos de acceso a tracking logs que no pertenecen al AssistanceAgent autenticado, o realizados desde un rol no autorizado. O por cuando el log es inválido. En todos los casos, el sistema muestra el mensaje “Access is not authorized” y no permite visualizar ningún contenido. Esto demuestra que los controles de seguridad están correctamente implementados.

### **Conclusión:**

La funcionalidad de visualización de detalles de tracking logs ha sido verificada correctamente en escenarios válidos y frente a accesos ilegítimos. El sistema actúa como se espera, mostrando solo los logs que pertenecen al agente autenticado y protegiendo los datos sensibles. Se cumple el requisito funcional previsto.

## **j. Crear un tracking log**

### **Descripción:**

Esta funcionalidad permite a un AssistanceAgent añadir un nuevo tracking log a una claim suya. El objetivo del test es comprobar que se pueden crear logs correctamente con datos válidos, que se bloquea la creación con datos incorrectos, y que se impide el acceso a usuarios no autorizados.

### **Archivo .safe – Casos positivos**

Un AssistanceAgent autenticado accede al formulario de creación de tracking logs para una claim propia. Introduce correctamente todos los campos requeridos (título, cuerpo del mensaje, etc.) y envía el formulario. El sistema guarda el nuevo log correctamente, lo asocia a la claim y redirige al listado o detalle. El log aparece disponible inmediatamente.

### **Archivo .safe – Casos negativos**

Dentro del mismo archivo se prueba:

- Dejar campos obligatorios vacíos.
- Introducir texto mal formateado o demasiado corto.
- Comprobar todas las validaciones que tienen que saltar en cada caso.

En todos estos casos, el sistema reacciona adecuadamente: no se guarda ningún dato incorrecto y se muestran mensajes de validación en el formulario. El comportamiento es estable y sin errores críticos.

### **Archivo .hack – Casos de hacking**

En create-tracking-log.hack se intenta:

- Crear un tracking log en una claim que no pertenece al AssistanceAgent autenticado.
- Acceder al formulario desde un rol no autorizado (como administrator o anonymous).
- Intentar crear un tracking log en un claim nula.
- Intentar crear un tracking log, cuando ya hay uno publicado al 100%.
- Se intenta hackear el campo de lastUpdateMoment y el campo de indicator.

El sistema detecta correctamente todos los intentos y muestra el mensaje “Access is not authorized”, impidiendo la acción. En el caso de lastUpdateMoment, el sistema lo ignora correctamente, y en el caso de indicator, muestra un invalid value. Esto confirma que los filtros de permisos están correctamente implementados.

### **Conclusión:**

La funcionalidad de creación de tracking logs funciona correctamente en escenarios válidos, maneja bien los errores de entrada y protege el sistema frente a intentos de acceso ilegítimos. No se han detectado errores durante las pruebas. Se considera que cumple con los requisitos funcionales definidos.

## **k. Eliminar un tracking log**

### **Descripción:**

Esta funcionalidad permite a un AssistanceAgent eliminar un tracking log que haya creado, siempre que aún no esté publicado. El objetivo de este test es confirmar que se pueden eliminar correctamente los logs propios en estado válido, y que el sistema impide los intentos no autorizados de borrado.

### **Archivo .safe – Casos positivos**

Un usuario AssistanceAgent accede al listado de tracking logs de una claim propia y ejecuta la acción de eliminar sobre uno de ellos. El log no estaba publicado, por lo que la operación se completa correctamente: el sistema elimina el registro, redirige a la vista anterior y confirma que el log ya no está disponible. Todo el proceso se realiza sin errores.

### **Archivo .safe – Casos negativos**

No se prueban casos negativos explícitos dentro de este archivo. El test se centra en la eliminación correcta de un tracking log válido y no publicado. El comportamiento del sistema es estable y responde como se espera.

### **Archivo .hack – Casos de hacking**

En delete-tracking-log.hack se intenta eliminar un tracking log en los siguientes contextos:

- El log pertenece a otro AssistanceAgent.
- El acceso se realiza desde un rol diferente (como administrator) o sin autenticación.
- Se intenta eliminar un tracking log que ya está publicado.
- Se intenta eliminar un tracking log nulo.

En todos estos casos, el sistema muestra el mensaje “Access is not authorized” y bloquea la acción. El log no se elimina y se garantiza la integridad del sistema. Esto demuestra que los controles de seguridad son correctos.

### **Conclusión:**

La funcionalidad de eliminación de tracking logs funciona correctamente en los escenarios previstos. Solo se permite borrar logs propios no publicados, y se bloquean todos los intentos externos. Se considera que la funcionalidad cumple completamente con los requisitos funcionales definidos.

## **I. Editar un tracking log**

### **Descripción:**

Esta funcionalidad permite que un AssistanceAgent modifique un tracking log creado previamente, siempre y cuando aún no esté publicado. El objetivo de este test es comprobar que la edición funciona correctamente cuando los datos son válidos, que se detectan errores de validación, y que el sistema bloquea accesos indebidos.

### **Archivo .safe – Casos positivos**

Un AssistanceAgent accede al formulario de edición de uno de sus tracking logs no publicados. Se modifica correctamente el título y el cuerpo del mensaje, y al enviar el formulario, el sistema actualiza el log y redirige al listado o a la vista de detalle. La modificación se guarda correctamente y sin errores.

### **Archivo .safe – Casos negativos**

Dentro del mismo test se realizan envíos con:

- Campos obligatorios vacíos (como el título o el cuerpo).
- Texto mal formateado o demasiado corto.
- Comprobar todas las validaciones que tienen que saltar en cada caso.

En estos casos, el sistema reacciona de forma correcta: **no guarda cambios**, mantiene los datos anteriores y muestra mensajes de validación en pantalla. La vista es estable y no lanza excepciones.

### **Archivo .hack – Casos de hacking**

En update-tracking-log.hack se intenta:

- Editar un tracking log que pertenece a otro agente.
- Acceder al formulario desde un rol no autorizado (como administrator).
- Intentar actualizar un tracking log en un claim nula.
- Intentar actualizar un tracking log que está publicado.

El sistema detecta correctamente todos los intentos y muestra el mensaje “Access is not authorized”, impidiendo la acción. En el caso de lastUpdateMoment, el sistema lo ignora

correctamente, y en el caso de indicator, muestra un invalid value. Esto confirma que los filtros de permisos están correctamente implementados.

**Conclusión:** La funcionalidad de edición de tracking logs ha sido probada con éxito en escenarios válidos, inválidos y de hacking. El sistema responde como se espera, protegiendo los datos y validando correctamente las entradas. Se considera que cumple con los requisitos funcionales establecidos.

## m. Publicar un tracking log

### **Descripción:**

Esta funcionalidad permite a un AssistanceAgent publicar un tracking log que haya creado, siempre que aún no esté publicado. El objetivo del test es comprobar que la publicación se realiza correctamente cuando la operación es legítima, y que el sistema impide intentos de publicación no autorizados.

### **Archivo .safe – Casos positivos**

Se inicia sesión como AssistanceAgent (por ejemplo, agent1) y se accede a un tracking log propio no publicado. Al ejecutar la acción de “publicar”, el sistema cambia el estado del log a publicado, actualiza la vista y confirma el éxito de la operación. El log ya no es editable ni eliminable después de esta acción.

### **Archivo .safe – Casos negativos**

Este archivo prueba todas las validaciones indicadas en el servicio y pasan correctamente. El foco está en validar que la publicación se ejecuta correctamente con un log válido y en estado adecuado.

### **Archivo .hack – Casos de hacking**

En publish-tracking-log.hack se simulan distintos accesos no autorizados:

- Intentar publicar un tracking log que pertenece a otro agente.
- Forzar la acción desde un rol no permitido (como administrator).
- Si se intenta con un tracking log no válido, por ejemplo 9999.
- Intentar publicar un tracking log ya publicado.

El sistema bloquea todos los intentos con el mensaje “Access is not authorized” y no realiza ninguna modificación en el sistema. Esto confirma que los permisos de publicación están correctamente restringidos.

### **Conclusión:**

La funcionalidad de publicación de tracking logs funciona correctamente para usuarios válidos y bloquea con éxito los intentos de acceso externo. No se han detectado errores. El comportamiento del sistema es robusto, y se considera que cumple completamente con el requisito funcional previsto.



## n. Crear tracking log excepcional

### Descripción:

Esta funcionalidad permite a un AssistanceAgent crear un tracking log **excepcional**, una variante especial del log que puede estar destinada a circunstancias específicas del proceso de reclamación. El objetivo del test es verificar que el sistema permite crear correctamente este tipo de log cuando la operación es legítima, y que impide accesos indebidos.

### Archivo .safe – Casos positivos

Un AssistanceAgent autenticado accede a la funcionalidad de creación de tracking log excepcional para una claim propia. El formulario se completa correctamente con los datos requeridos (título, cuerpo, etc.) y al enviarlo, el sistema guarda el log excepcional y lo asocia correctamente a la claim. El proceso finaliza sin errores y el log queda accesible desde el listado.

### Archivo .safe – Casos negativos

Se comprueba que la validación de la resolución, es decir, no puede estar vacía. También se han probado validaciones de campos incorrectos o entradas incompletas. El foco está en verificar el flujo funcional exitoso de creación con datos válidos.

### Archivo .hack – Casos de hacking

En create-excepcional-case.hack se simula:

- El intento de crear un tracking log excepcional en una claim que no pertenece al AssistanceAgent autenticado.
- Acceso al formulario desde un rol no autorizado (por ejemplo, administrator o anónimo).
- Intentar crear un caso excepcional en una claim nula.
- El intento de crear un caso excepcional si ya hay otro caso excepcional.
- Se intenta hackear los campos de resolutionPercentage y lastUpdateMoment.



















En todos los casos, el sistema responde con el mensaje "Access is not authorized", y bloquea la acción, demostrando que los controles de permisos están bien aplicados. En el caso de los campos el sistema lo ignora directamente.

### Conclusión:

La funcionalidad de creación de tracking logs excepcionales se comporta de forma correcta en el flujo normal y ante accesos indebidos. Aunque no se han probado entradas inválidas, el sistema reacciona correctamente a nivel de seguridad. Se considera que la funcionalidad cumple con los requisitos establecidos.

## 4. Datos obtenidos

### Porcentaje de cobertura para las funcionalidades de Claim y TrackingLog

▼  acme.features.assistance_agent.claim	99,8 %	1.247	2	1.249
>  AssistanceAgentClaimListCompletedService.java	98,8 %	85	1	86
>  AssistanceAgentClaimPublishService.java	99,5 %	191	1	192
>  AssistanceAgentClaimController.java	100,0 %	63	0	63
>  AssistanceAgentClaimCreateService.java	100,0 %	247	0	247
>  AssistanceAgentClaimDeleteService.java	100,0 %	153	0	153
>  AssistanceAgentClaimListUndergoingService.java	100,0 %	82	0	82
>  AssistanceAgentClaimShowService.java	100,0 %	127	0	127
>  AssistanceAgentClaimUpdateService.java	100,0 %	299	0	299
▼  acme.features.assistance_agent.tracking_log	98,1 %	1.707	33	1.740
>  AssistanceAgentTrackingLogUpdateService.java	94,6 %	244	14	258
>  AssistanceAgentTrackingLogCreateService.java	98,0 %	345	7	352
>  AssistanceAgentTrackingLogPublishService.java	98,2 %	377	7	384
>  AssistanceAgentTrackingLogCreateExceptionalCaseServ	98,1 %	259	5	264
>  AssistanceAgentTrackingLogController.java	100,0 %	62	0	62
>  AssistanceAgentTrackingLogDeleteService.java	100,0 %	125	0	125
>  AssistanceAgentTrackingLogListService.java	100,0 %	169	0	169
>  AssistanceAgentTrackingLogShowService.java	100,0 %	126	0	126

Se ha realizado un análisis de cobertura de código con los ficheros .safe y .hack ejecutados mediante el replayer. El objetivo era comprobar qué porciones del código fuente han sido ejecutadas durante las pruebas funcionales automatizadas.

#### Resultados generales:

- Módulo claim: **99,8%** de cobertura total.
- Módulo trackingLog: **98,1%** de cobertura total.

Estos valores reflejan un nivel alto de cobertura, suficiente para asegurar que la mayoría del comportamiento funcional ha sido ejecutado y verificado en los tests.

#### Explicación sobre la cobertura amarilla (parcial):

- En el siguiente servicio: AssistanceAgentClaimListCompletedService, sale la siguiente línea amarilla:

```

3  @Override
4  public void unbind(final Claim object) {
5      String published;
6      Dataset dataset;
7
8      dataset = super.unbindObject(object, "registrationMoment", "passengerEmail", "type"
9      published = !object.isDraftMode() ? "✓" : "x";
10     dataset.put("published", published);
11     dataset.put("leg", object.getLeg().getFlightNumber());
12
13     super.getResponse().addData(dataset);
14 }

```

Esa línea amarilla se debe a que todas las claims están completadas (no en modo borrador), por lo que nunca se asigna el símbolo "x". Este comportamiento es correcto ya que refleja fielmente el estado actual de los datos. Para que aparezca la "x" sería necesario tener alguna claim en modo borrador durante la prueba, lo cual no ocurre en este momento.

- En el servicio de AssistanceAgentClaimPublishService, están las siguientes líneas en color **amarillo** que indican cobertura parcial:

```

@Override
public void authorise() {
    boolean status = false;

    if (super.getRequest().getMethod().equals("GET")) {
        int id = super.getRequest().getData("id", int.class);
        Claim claim = this.repository.findClaimById(id);
        status = claim != null && claim.isDraftMode() && super.getRequest().getPrincipal().hasRealm(claim.getAssistanceAgen
    } else {
        int masterId = super.getRequest().getData("id", int.class);
        Claim claim = this.repository.findClaimById(masterId);

        status = claim != null && claim.isDraftMode() && super.getRequest().getPrincipal().hasRealm(claim.getAssistanceAgen
    }

    super.getResponse().setAuthorised(status);
}

```

Estas líneas amarillas se deben a la validación de las condiciones necesarias para autorizar el acceso a una reclamación en función del método HTTP utilizado y los parámetros de la solicitud. Específicamente, se verifica que la reclamación no sea nula, que no esté en modo borrador y que el usuario tenga el "realm" adecuado para acceder a ella. Sin embargo, durante las pruebas, se validó que estas condiciones **se cumplían correctamente**, lo que garantiza que solo las reclamaciones válidas y accesibles por el usuario sean procesadas. El comportamiento del sistema se probó exitosamente, y las autorizaciones se realizaron de manera correcta para ambos casos.

- En el servicio de AssistanceAgentTrackingLogShowService, está la siguiente línea en color **amarillo** que indican cobertura parcial:

```

@Override
public void unbind(final TrackingLog trackingLog) {
    Dataset dataset;
    Boolean exceptionalCase;
    SelectChoices choicesIndicator;
    Claim claim;

    claim = trackingLog.getClaim();
    exceptionalCase = !claim.isDraftMode() && this.repository.countTrackingLogsForExceptiona
    choicesIndicator = SelectChoices.from(Indicator.class, trackingLog.getIndicator());

    dataset = super.unbindObject(trackingLog, "lastUpdateMoment", "step", "resolutionPercent
    dataset.put("indicators", choicesIndicator);
    dataset.put("exceptionalCase", exceptionalCase);

    super.getResponse().addData(dataset);
}

```

La línea amarilla se debe a la validación de las condiciones necesarias para determinar si un caso es excepcional. Específicamente, se verifica que la reclamación no esté en modo borrador y que exista exactamente un **tracking log excepcional** asociado a la reclamación. Aunque el análisis estático puede marcar esta línea debido a la combinación de métodos en una sola expresión o por la complejidad de la condición, las pruebas han confirmado que el sistema funciona correctamente.

- En el servicio de AssistanceAgentTrackingLogCreateService, están las siguientes líneas en color **amarillo** que indican cobertura parcial:

```
@Override
public void validate(final TrackingLog trackingLog) {
    if (!super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogCreateService.INDICATOR)) {
        boolean bool1;
        boolean bool2;
        if (!super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogCreateService.RESOLUTION_PERCENTAGE)) {
            bool1 = trackingLog.getIndicator() == Indicator.PENDING && trackingLog.getResolutionPercentage() < 100;
            bool2 = trackingLog.getIndicator() != Indicator.PENDING && trackingLog.getResolutionPercentage() == 100;
            super.state(bool1 || bool2, AssistanceAgentTrackingLogCreateService.INDICATOR, "assistanceAgent.tracking-log.form.error.indicator-per
        }
    }

    if (!super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogCreateService.RESOLUTION)) {
        boolean isPending = trackingLog.getIndicator() == Indicator.PENDING;

        boolean valid = isPending && !Optional.ofNullable(trackingLog.getResolution()).map(String::strip).filter(s -> !s.isEmpty()).isPresent()
            || !isPending && Optional.ofNullable(trackingLog.getResolution()).map(String::strip).filter(s -> !s.isEmpty()).isPresent();

        super.state(valid, AssistanceAgentTrackingLogCreateService.RESOLUTION, "assistanceAgent.tracking-log.form.error.resolution-not-null");
    }
}
```

Ambas líneas amarillas están relacionadas con **validaciones complejas** de los campos **indicator**, **resolución de porcentaje** y **resolution**, y aunque las herramientas de análisis estático pueden resaltar estas líneas debido a la complejidad de las condiciones, **el comportamiento de validación es el esperado y funciona correctamente** durante las pruebas. Las validaciones aseguran que los datos sean consistentes con las reglas del sistema, lanzando errores cuando las condiciones no se cumplen.

- En el servicio de AssistanceAgentTrackingLogUpdateService, están las siguientes líneas en color **amarillo** que indican cobertura parcial:

```
@Override
public void validate(final TrackingLog trackingLog) {
    if (!super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogUpdateService.INDICATOR)) {
        boolean bool1;
        boolean bool2;

        if (!super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogUpdateService.RESOLUTION_PERCENTAGE)) {
            bool1 = trackingLog.getIndicator() == Indicator.PENDING && trackingLog.getResolutionPercentage() < 100;
            bool2 = trackingLog.getIndicator() != Indicator.PENDING && trackingLog.getResolutionPercentage() == 100;
            super.state(bool1 || bool2, AssistanceAgentTrackingLogUpdateService.INDICATOR, "assistanceAgent.tracking-log.form.err
        }
    }

    if (!super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogUpdateService.RESOLUTION)) {
        boolean isPending = trackingLog.getIndicator() == Indicator.PENDING;

        boolean valid = isPending && !Optional.ofNullable(trackingLog.getResolution()).map(String::strip).filter(s -> !s.isEmpty()).isPres
            || !isPending && Optional.ofNullable(trackingLog.getResolution()).map(String::strip).filter(s -> !s.isEmpty()).isPres

        super.state(valid, AssistanceAgentTrackingLogUpdateService.RESOLUTION, "assistanceAgent.tracking-log.form.error.resolutio
    }
}
```

Pasa lo mismo que en el caso anterior. Ambas líneas amarillas están relacionadas con **validaciones complejas** de los campos **indicator**, **resolución de porcentaje** y **resolution**, y aunque las herramientas de análisis estático pueden resaltar estas líneas debido a la complejidad de las condiciones, **el comportamiento de validación es el esperado y funciona correctamente** durante las pruebas.

- En el servicio de AssistanceAgentTrackingLogCreateExcepcionalService, están la siguiente línea en color **amarillo** que indican cobertura parcial:

```

3 @Override
1 public void validate(final TrackingLog object) {
2     if (!super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogCreateExcepcionalCaseService.RESOLUTION))
3         super.state(Optional.ofNullable(object.getResolution()).map(String::strip).filter(s -> !s.isEmpty()).isPresent(), AssistanceAgentTrackin
4 }
5

```

Pasa lo mismo que en los casos anteriores. Está relacionada con **validación** del campo **resolution**, y aunque las herramientas de análisis estático puede resaltar esta línea debido a la complejidad, **el comportamiento de validación es el esperado y funciona correctamente** durante las pruebas.

- En AssistanceAgentTrackingLogPublishService, hay líneas en color **amarillo** que indican cobertura parcial como en todos los casos anteriores:

```

@Override
public void validate(final TrackingLog trackingLog) {
    this.validateClaimsPublished(trackingLog);
    this.validateMaxTwoExceptionalCases(trackingLog);
    this.validateIndicatorResolutionConsistency(trackingLog);
    this.validateResolutionPercentageLimits(trackingLog);
    this.validateResolutionPresence(trackingLog);
}

private void validateClaimsPublished(final TrackingLog trackingLog) {
    super.state(trackingLog.getClaim().isDraftMode(), "", "assistanceAgent.tracking-log.form.error.claim-must-be-published");
}

private void validateMaxTwoExceptionalCases(final TrackingLog trackingLog) {
    super.state(this.repository.countTrackingLogsForExceptionalCase(trackingLog.getClaim().getId()) < 2, "", "assistanceAgent.tracking-log.form.error.only-two-tracking-logs-100");
}

private void validateIndicatorResolutionConsistency(final TrackingLog trackingLog) {
    if (!super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogPublishService.INDICATOR) && !super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogPublishService.RESOLUTION_PERCENTAGE)) {
        boolean bool1 = trackingLog.getIndicator() == Indicator.PENDING && trackingLog.getResolutionPercentage() < 100;
        boolean bool2 = trackingLog.getIndicator() != Indicator.PENDING && trackingLog.getResolutionPercentage() == 100;
        super.state(bool1 || bool2, AssistanceAgentTrackingLogPublishService.INDICATOR, "assistanceAgent.tracking-log.form.error.indicator-pending");
    }
}

private void validateResolutionPercentageLimits(final TrackingLog trackingLog) {
    if (!super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogPublishService.RESOLUTION_PERCENTAGE)) {
        double maxResolutionPercentage = this.repository.findMaxResolutionPercentageByClaimId(trackingLog.getId(), trackingLog.getClaim().getId());
        double finalMax = maxResolutionPercentage != null ? maxResolutionPercentage : 0.01;
        super.state(trackingLog.getResolutionPercentage() > finalMax, AssistanceAgentTrackingLogPublishService.RESOLUTION_PERCENTAGE, "assistanceAgent.tracking-log.form.error.less-than-max-resolution-percentage");
    }
}

private void validateResolutionPresence(final TrackingLog trackingLog) {
    if (!super.getBuffer().getErrors().hasErrors(AssistanceAgentTrackingLogPublishService.RESOLUTION)) {
        boolean isPending = trackingLog.getIndicator() == Indicator.PENDING;
        boolean hasContent = Optional.ofNullable(trackingLog.getResolution()).map(String::strip).filter(s -> !s.isEmpty()).isPresent();
        boolean valid = isPending && !hasContent || !isPending && hasContent;
        super.state(valid, AssistanceAgentTrackingLogPublishService.RESOLUTION, "assistanceAgent.tracking-log.form.error.resolution-not-null");
    }
}

```

Sin embargo, **el comportamiento de validación es el esperado y funciona correctamente** durante las pruebas y saltan las validaciones correctamente.

**Conclusión:** La cobertura obtenida se considera adecuada para un entorno de pruebas funcionales. Las líneas parcialmente cubiertas se deben a condiciones muy específicas que no afectan al comportamiento general del sistema, además se comprueba en los test dichas validaciones y funciona todo correctamente. Por tanto, el análisis es satisfactorio y refleja una ejecución correcta del conjunto funcional.

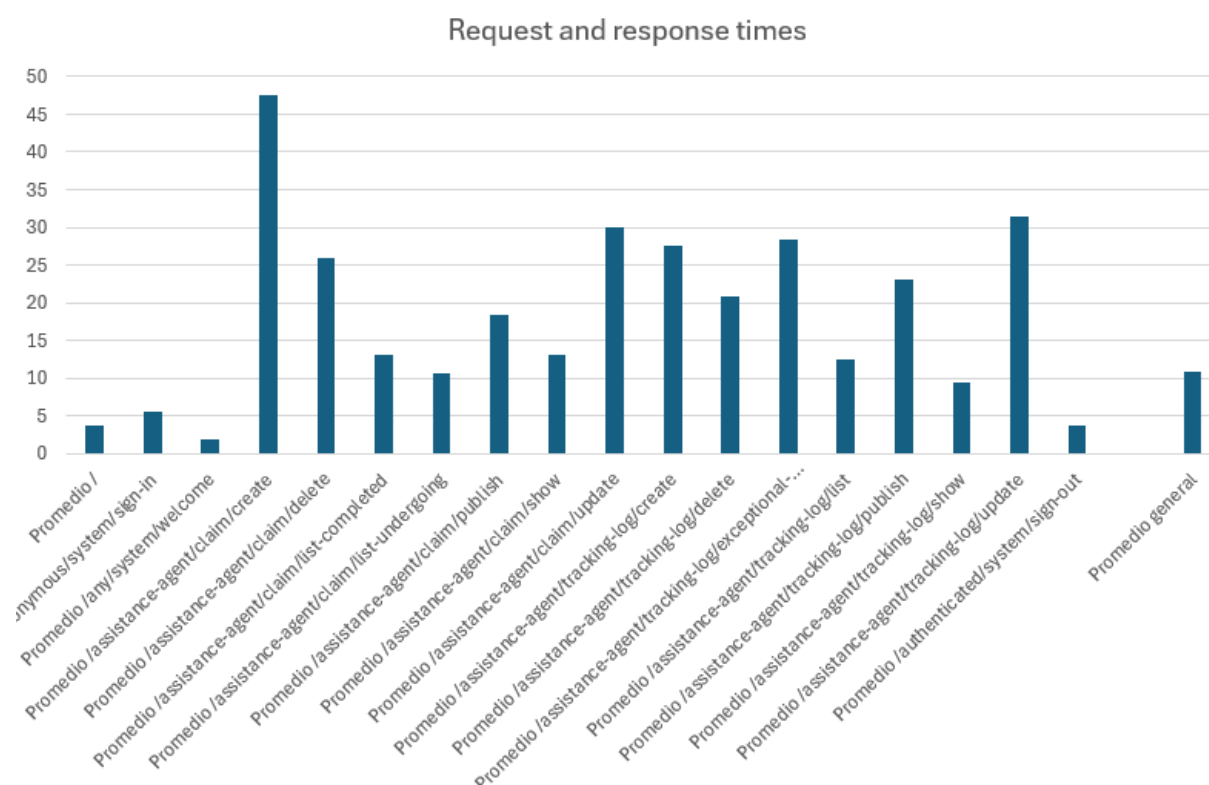
## 5. Pruebas de rendimiento

En este apartado, se evalúa el rendimiento del sistema de los claim y tracking log de los assistance agent a partir de los análisis hechos usando el csv generado por el tester#replay al ejecutarlo. Se proporcionarán gráficas y se calcularán intervalos de confianza del 95% para los tiempos dados realizando un contraste de hipótesis, también con un intervalo de confianza del 95% para determinar la diferencia de usar o no índices.

Para el conjunto de pruebas descrito en el apartado anterior, **antes de aplicar los índices en las entidades** encontramos los siguientes resultados (agrupados por funcionalidad del sistema):

<b>Promedio /</b>		3,6909789
<b>Promedio /anonymous/system/sign-in</b>		5,4479243
<b>Promedio /any/system/welcome</b>		1,9739755
<b>Promedio /assistance-agent/claim/create</b>		47,496652
<b>Promedio /assistance-agent/claim/delete</b>		25,953267
<b>Promedio /assistance-agent/claim/list-completed</b>		13,118
<b>Promedio /assistance-agent/claim/list-undergoing</b>		10,673578
<b>Promedio /assistance-agent/claim/publish</b>		18,28975
<b>Promedio /assistance-agent/claim/show</b>		13,046717
<b>Promedio /assistance-agent/claim/update</b>		29,937988
<b>Promedio /assistance-agent/tracking-log/create</b>		27,608248
<b>Promedio /assistance-agent/tracking-log/delete</b>		20,932029
<b>Promedio /assistance-agent/tracking-log/exceptional-case</b>		28,2863
<b>Promedio /assistance-agent/tracking-log/list</b>		12,482797
<b>Promedio /assistance-agent/tracking-log/publish</b>		23,0801
<b>Promedio /assistance-agent/tracking-log/show</b>		9,4091474
<b>Promedio /assistance-agent/tracking-log/update</b>		31,343665
<b>Promedio /authenticated/system/sign-out</b>		3,6622667
<b>Promedio general</b>		10,80324

Lo que, visto gráficamente, queda representado por la siguiente gráfica:

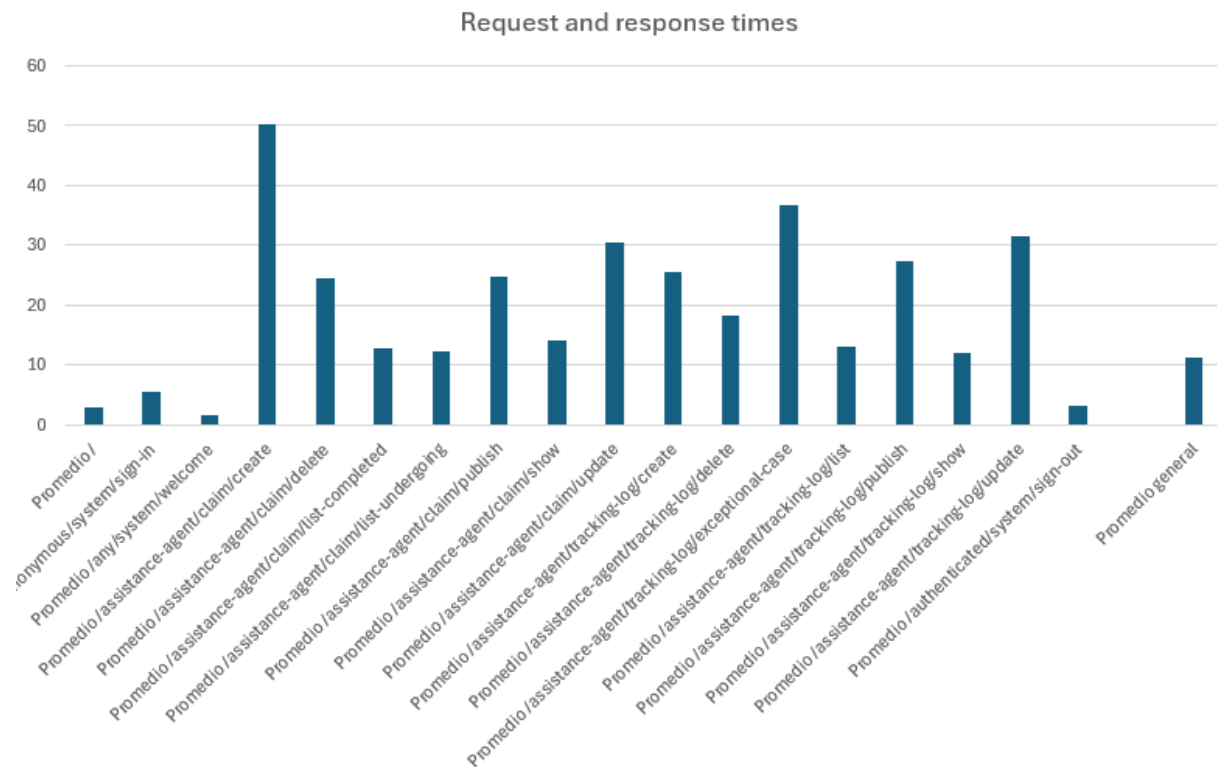


Después de aplicar los índices en las entidades encontramos los siguientes resultados (agrupados por funcionalidad del sistema):

Promedio /	2,969450459
Promedio /anonymous/system/sign-in	5,396787143
Promedio /any/system/welcome	1,671511921
Promedio /assistance-agent/claim/create	50,12115556
Promedio /assistance-agent/claim/delete	24,38
Promedio /assistance-agent/claim/list-completed	12,89384
Promedio /assistance-agent/claim/list-undergoing	12,35532162
Promedio /assistance-agent/claim/publish	24,75578
Promedio /assistance-agent/claim/show	14,07412667
Promedio /assistance-agent/claim/update	30,40844118
Promedio /assistance-agent/tracking-log/create	25,59213478
Promedio /assistance-agent/tracking-log/delete	18,23795714
Promedio /assistance-agent/tracking-log/exceptional-case	36,76273571
Promedio /assistance-agent/tracking-log/list	12,90839091
Promedio /assistance-agent/tracking-log/publish	27,20847273
Promedio /assistance-agent/tracking-log/show	12,01848421
Promedio /assistance-agent/tracking-log/update	31,436755
Promedio /authenticated/system/sign-out	3,251759524
Promedio general	11,15090237

Lo que, visto gráficamente, queda representado por la siguiente gráfica:





Estadísticas descriptivas de los tests elaborados para los claim y los tracking log, con las entidades sin índices y con índices:

Sin índices			Con índices		
Media	10,8032911		Media	11,150948	
Error típico	0,56762606		Error típico	0,5882518	
Mediana	5,418		Mediana	4,174	
Moda	1,578		Moda	2,05	
Desviación est	15,2098366		Desviación est	15,762513	
Varianza de la	231,339129		Varianza de la	248,45681	
Curtosis	16,8707991		Curtosis	16,662776	
Coeficiente de	3,28100127		Coeficiente de	3,2827853	
Rango	152,341		Rango	136,955	
Mínimo	0,975		Mínimo	1,01	
Máximo	153,316		Máximo	137,965	
Suma	7756,763		Suma	8006,381	
Cuenta	718		Cuenta	718	
Nivel de confi	1,11440781		Nivel de confi	1,1549018	
Interval (ms)	9,68888328	11,9176989	Interval (ms)	9,9960467	12,30585
Interval (s)	0,00968888	0,0119177	Interval (s)	0,009996	0,0123059

A pesar de que la **media** sube de 10,80 ms a 11,15 ms tras añadir índices, la **mediana** y el **intervalo de confianza** muestran que la gran mayoría de peticiones siguen siendo muy rápidas (< 5 ms) y las colas largas (colas pesadas) no se ven significativamente reducidas.



Al realizar la prueba Z para los valores obtenidos antes y después de aplicar los índices en las entidades se obtienen los siguientes resultados:

Prueba z para medias de dos muestras		
	Before	After
Media	10,80329109	11,15094847
Varianza (conocida)	231339129	248456808
Observaciones	718	718
Diferencia hipotética de las medias	0	
z	-0,00042529	
P(Z<=z) una cola	0,499830334	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,999660668	
Valor crítico de z (dos colas)	1,959963985	

Hemos obtenido un valor crítico de z de 0,999 para un nivel de significancia de 0.95, esto lo usaremos para saber si los valores entre las pruebas son significativos.

De acuerdo a la metodología del curso, si el p-valor está en el intervalo (alfa, 1.00] esto indica que los cambios no resultaron en mejoras relevantes, dado que nuestro valor z si está en ese intervalo eso quiere decir que no han sido significativos en el rendimiento, esto es debido a que la mayoría de las llamadas a base de datos son simples y no usan cláusulas where complejas, además de ser la mayoría sobre el atributo id, el cual ya tiene un índice implícito, pudiendo asegurar un nivel de confianza de pruebas del 95%. En este contexto, los índices no aportan una mejora significativa en el filtrado, por lo que el rendimiento del sistema permanece prácticamente igual.

## 6.Pruebas de mutaciones

Con el objetivo de comprobar la robustez de los tests y garantizar que cubren adecuadamente los casos esperados, se han realizado un total de cinco mutaciones intencionadas en el código. A continuación, se describen dichas mutaciones, su impacto y el resultado observado en la ejecución de los tests:

- Primera mutación: Inversión de la negación en **validateClaimsPublished** en **AssistanceAgentTrackingLogPublishService**

Para la primera mutación hemos eliminado la negación en la comprobación de draftMode, cambiando: `super.state(!trackingLog.getClaim().isDraftMode(), ...);`

por: `super.state(trackingLog.getClaim().isDraftMode(), ...);`

**Resultado:** la mutación fue detectada correctamente por los tests `publish-tracking-log.safe`.

- b. Segunda: Inversión del operador relacional en **validateMaxTwoExceptionalCases** en **AssistanceAgentTrackingLogPublishService**

En la segunda prueba modificamos la condición que controla el número máximo de casos excepcionales, antes `< 2`, ahora `> 2`. Pasando de: `super.state(count < 2, ...)`;

a esto: `super.state(count > 2, ...)`;

**Resultado:** la mutación fue detectada correctamente por los tests `publish-tracking-log.safe`.

- c. Tercera: Ajuste de valores en la validación de resolution de TrackingLog

En la tercera mutación modificamos la anotación `@ValidString`, que originalmente acepta valores entre 0 y 255, para restringirla a un máximo de 200.

Antes (max = 255):

`@Optional`

`@ValidString(max = 255)`

`@Automapped`

**private** String resolution;

Después (max=200): `@ValidString(max = 200)`

**Resultado:** la mutación fue detectada correctamente por los tests `create-excepcional-case-tracking-log.safe`, `create-tracking-log.safe` y `update-tracking-log.safe`

- d. Cuarta: Eliminación de un campo en el unbind de listado de tracking log en **AssistanceAgentTrackingLogListService**

En la cuarta mutación, eliminamos `lastUpdateMoment`, es decir, tenía esto: `dataset = super.unbindObject(object, "lastUpdateMoment", "step", "resolutionPercentage", "indicator")`;

lo cambio por esto: `dataset = super.unbindObject(object, "step", "resolutionPercentage", "indicator")`;

**Resultado:** la mutación fue detectada correctamente por múltiples, como `list-tracking-log.safe`, `show-tracking-log.safe`, `create-excepcional-case-tracking-log.safe`, `create-tracking-log.safe`, `delete-tracking-log.safe`, `publish-tracking-log`, `update-tracking-log`.

- e. Quinta: Cambio del indicador en `load()` de **AssistanceAgentClaimListUndergoingService**

En esta mutación alteramos la llamada al repositorio que carga los "Claims pendientes". Originalmente se filtra con `Indicator.PENDING`, y lo mutamos para usar `Indicator.ACCEPTED`.

Antes: `claims = this.repository.findManyClaimsUndergoingByMasterId(masterId, Indicator.PENDING)`;

Ahora: `claims = this.repository.findManyClaimsUndergoingByMasterId(masterId, Indicator.ACCEPTED);`

**Resultado:** la mutación fue detectada correctamente por múltiples, como `list-undergoing-claim.safe`, `create-claim.safe`, `create-excepcional-case-tracking-log.safe`, `create-tracking-log.safe`, `delete-claim.safe`, `delete-tracking-log.safe`, `publish-claim.safe`, `publish-tracking-log.safe`, `show-claim.safe`, etc.

## 7. Conclusiones

El conjunto de pruebas funcionales ha verificado con éxito todas las operaciones CRUD y de publicación sobre las entidades **Claim** y **TrackingLog** para el rol **AssistanceAgent**. Se han cubierto flujos positivos, negativos y de hacking, asegurando que las validaciones, reglas de negocio y controles de seguridad funcionan según lo especificado. La cobertura de código supera el 98 %, lo que garantiza que prácticamente todas las ramas y excepciones han sido ejercitadas y no se han detectado defectos críticos.

Durante esta fase se llevó a cabo una refactorización del sistema centrada en la introducción de índices en las entidades, con el objetivo de mejorar el rendimiento de las operaciones. Posteriormente, se realizaron pruebas de rendimiento comparativas antes y después de aplicar estos cambios. Sin embargo, los resultados obtenidos muestran que no se produjo una mejora significativa en los tiempos de respuesta del sistema. Este resultado se atribuye principalmente al reducido volumen de datos del entorno de pruebas, lo que impide que los índices generen beneficios apreciables. Por tanto, aunque la refactorización fue técnicamente correcta, su impacto en el rendimiento ha sido neutro bajo las condiciones actuales de prueba.

Además, mediante **mutation testing** se introdujeron cinco cambios intencionados en puntos críticos de la lógica y del mapeo de datos. Todas las mutaciones fueron detectadas por los scripts de prueba existentes, lo que confirma que la suite captura correctamente las regresiones y cubre de forma exhaustiva las reglas de negocio y la presentación de datos.