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 C1.021 - Student 4		
Autores	**Correos**	**Rol**
Paula Sánchez Gómez	pausangom1@alum.us.es	Desarrollador/Tester/Manager

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

26 de Mayo de 2025

Control de versiones

Versión	Fecha	Descripción
v1.0	18/02/2025	Primera versión del documento

ÍNDICE

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*, evaluando el sistema en distintos entornos hardware mediante técnicas estadísticas (promedios, intervalos de confianza al 95% y contraste de hipótesis) y herramientas como Excel y VisualVM. Se logró verificar que el tiempo medio de respuesta para todas las operaciones fue **significativamente inferior a 1 segundo**, cumpliendo así los criterios de eficiencia.

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 manuales, 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, así como observación de tiempos de CPU y memoria con VisualVM. Se han utilizado dos dispositivos diferentes para comparar la eficiencia del sistema y se ha comprobado que los tiempos de respuesta cumplen el objetivo de mantenerse por debajo de 1 segundo de media por operación.

Este informe forma parte de la entrega **D04 - Testing Formal**, dentro del marco de la asignatura *Diseño y Pruebas II*, y refleja un proceso completo de validación de calidad software acorde a las metodologías vistas en clase.

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-undergoing-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.
- Un AssistanceAgent distinto intenta ver los reclamos completados de otro agente manipulando la URL.

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-completed-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.
- Un AssistanceAgent intenta visualizar los claims pendientes de otro agente modificando manualmente la URL.

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 entrada: 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.
- Un usuario anónimo o de otro rol intenta enviar el formulario de creación de una claim.

En ambos casos, el sistema responde correctamente con el mensaje “Access is not authorized”, impidiendo la operación. 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.

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.
- Un usuario de otro rol accede directamente al formulario de edición.
- Se intenta modificar una claim ya publicada.

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.
- Un usuario intenta forzar la publicación manipulando la URL.

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 que solo se muestran los logs de la claim correspondiente y que el formato de presentación es correcto.

Archivo .safe – Casos negativos
Este test no incluye pruebas negativas explícitas (por ejemplo, claim sin logs o inexistente). El foco está en verificar que el listado se genera correctamente cuando hay datos válidos. El comportamiento del sistema es estable y no se producen errores.

Archivo .hack – Casos de hacking
En list-trackingLog.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 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-trackingLog.hack se simulan intentos de acceso a tracking logs que no pertenecen al AssistanceAgent autenticado, o realizados desde un rol no autorizado. 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.

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-trackingLog.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).
- Manipular la URL para inyectar IDs de claims ajenas.

El sistema detecta correctamente todos los intentos y muestra el mensaje “Access is not authorized”, impidiendo la acción. 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-trackingLog.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 manipula la URL para inyectar un ID de log ajeno.

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.

l. 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.

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-trackingLog.hack se intenta:

- Editar un tracking log que pertenece a otro agente.
- Acceder al formulario desde un rol no autorizado (como administrator).
- Manipular directamente la URL con IDs de logs ajenos.

El sistema bloquea todos estos intentos con el mensaje “Access is not authorized” y no permite realizar cambios. Esto garantiza que los permisos están correctamente controlados.

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 no contiene pruebas negativas explícitas como intentar publicar dos veces el mismo log o publicar uno ya publicado. 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-trackingLog.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).
- Manipular la URL para intentar publicar logs ajenos.

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
No se incluyen casos negativos en este archivo, por lo que no 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).
- Manipulación de URL o IDs para forzar la acción.

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.

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.assistanceAgent.trackingLog	94,5 %	1.576	91	1.667
AssistanceAgentTrackingLogDeleteService.java	66,2 %	92	47	139
AssistanceAgentTrackingLogPublishService.java	93,9 %	354	23	377
AssistanceAgentTrackingLogCreateService.java	97,4 %	297	8	305
AssistanceAgentTrackingLogShowService.java	96,1 %	124	5	129
AssistanceAgentTrackingLogCreateExceptionalCaseServ	98,8 %	239	3	242
AssistanceAgentTrackingLogListService.java	98,2 %	163	3	166
AssistanceAgentTrackingLogUpdateService.java	99,3 %	266	2	268
AssistanceAgentTrackingLogController.java	100,0 %	41	0	41

acme.features.assistanceAgent.claim	94,2 %	1.193	73	1.266
AssistanceAgentClaimDeleteService.java	61,9 %	117	72	189
AssistanceAgentClaimListCompletedService.java	98,8 %	82	1	83
AssistanceAgentClaimController.java	100,0 %	42	0	42
AssistanceAgentClaimCreateService.java	100,0 %	252	0	252
AssistanceAgentClaimListUndergoingService.java	100,0 %	79	0	79
AssistanceAgentClaimPublishService.java	100,0 %	250	0	250
AssistanceAgentClaimShowService.java	100,0 %	124	0	124
AssistanceAgentClaimUpdateService.java	100,0 %	247	0	247

Se ha realizado un análisis de cobertura de código con los ficheros .safe 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: **94,2%** de cobertura total.
- Módulo trackingLog: **94,5%** 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 de los casos incompletos:

- Los **servicios de eliminación** (DeleteService) tanto en Claim como en TrackingLog presentan una cobertura más baja (**61,9% y 66,2% respectivamente**) debido a que **el método unbind() no llega a ejecutarse** cuando se borra un objeto correctamente. En este tipo de acción, no se necesita cargar datos de vuelta a la vista, por lo que dicha parte del código no se alcanza.

  Esto es **esperado** y **no representa un fallo funcional**, sino una peculiaridad del ciclo de vida del DeleteService.

```
@Override
public void unbind(final TrackingLog trackingLog) {
    Dataset dataset;
    SelectChoices indicatorChoices = SelectChoices.from(Indicator.class, trackingLog.getIndicator());

    dataset = super.unbindObject(trackingLog, "lastUpdateMoment", "step", "resolutionPercentage", "draftMode", "resolution", "indicator");
    dataset.put("indicators", indicatorChoices);

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

```
@Override
public void unbind(final Claim claim) {
    Dataset dataset;
    SelectChoices claimTypeChoices = SelectChoices.from(ClaimType.class, claim.getType());
    SelectChoices indicatorChoices = SelectChoices.from(Indicator.class, claim.getIndicator());
    SelectChoices legChoices = SelectChoices.from(this.repository.findAllLegs(), "flightNumber", claim.getLeg());

    dataset = super.unbindObject(claim, "registrationMoment", "passengerEmail", "description", "draftMode", "leg", "indicator", "type");
    dataset.put("types", claimTypeChoices);
    dataset.put("indicators", indicatorChoices);
    dataset.put("legs", legChoices);

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

Cobertura amarilla (parcial):

- En algunos servicios como AssistanceAgentClaimListCompletedService, AssistanceAgentTrackingLogUpdateService y AssistanceAgentTrackingLogPublishService, hay líneas en color **amarillo** que indican cobertura parcial:
  - En ListCompletedService, el bloque unbind() que construye SelectChoices puede no ejecutarse si no hay datos.
  - En TrackingLogUpdateService, ciertas validaciones sólo se ejecutan bajo condiciones específicas (por ejemplo, un valor concreto de indicator o resolutionPercentage).
  - En PublishService, condiciones del método validate() dependen de estados muy concretos, y por tanto, no todas se activan en una ejecución típica.
  - **Condicionales complejos en validate():** En clases como TrackingLogUpdateService o TrackingLogCreateService, hay múltiples ramas

condicionales que dependen de combinaciones muy específicas de datos, como:

- indicator == PENDING && resolutionPercentage < 100
- resolutionPercentage == 100 && indicator != PENDING
- countTrackingLogsForExceptionalCase(...) == 2

- No todas estas combinaciones se alcanzan con una única ejecución del .safe, por lo que el bloque super.state(...) puede no activarse, dejando parte del código sin cubrir.

- **Lógicas en authorise():** Algunas líneas amarillas aparecen en métodos authorise() que contienen expresiones complejas con múltiples condiciones (claim != null && condition && roleCheck...). Si una condición corta el flujo (por ejemplo, si claim == null), las siguientes ya no se ejecutan, y esto impide alcanzar algunas ramas lógicas.

- **Uso de Optional y validación avanzada:** Se usan construcciones como: Optional.ofNullable(...).map(...).filter(...).isPresent()

  Estas validaciones son dinámicas y solo se activan con datos muy concretos, lo que explica que las líneas asociadas no siempre se marquen como cubiertas. En particular, ocurre en la validación del campo resolution y en la lógica que gestiona el estado draftMode.

Ejemplos:

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

    }

    if (!super.getBuffer().getErrors().hasErrors("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, "resolution", "assistanceAgent.tracking-log.form.error.resolution-not-null");

    }
}


@Override
public void unbind(final TrackingLog object) {
    Dataset dataset;

    Boolean exceptionalCase;
    Claim claim;

    claim = object.getClaim();
    exceptionalCase = this.repository.countTrackingLogsForExceptionalCase(claim.getId()) == 1;

    dataset = super.unbindObject(object, "lastUpdateMoment", "step", "resolutionPercentage", "resolution", "indicator");
    dataset.put("masterId", super.getRequest().getData("masterId", int.class));
    dataset.put("exceptionalCase", exceptionalCase);
    dataset.put("indicators", SelectChoices.from(Indicator.class, object.getIndicator()));

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

```java
@Override
public void authorise() {
    boolean status;
    int masterId;
    AssistanceAgent assistanceAgent;
    Claim claim;
    Boolean exceptionalCase;

    masterId = super.getRequest().getData("masterId", int.class);
    claim = this.repository.findOneClaimById(masterId);
    assistanceAgent = claim == null ? null : claim.getAssistanceAgent();
    exceptionalCase = this.repository.countTrackingLogsForExceptionalCase(masterId) == 1;

    status = claim != null && exceptionalCase && super.getRequest().getPrincipal().hasRealm(assistanceAgent);

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

@Override

@Override
public void authorise() {
    boolean status;
    int masterId;
    Claim claim;

    masterId = super.getRequest().getData("id", int.class);
    claim = this.repository.findClaimById(masterId);
    status = claim != null && claim.isDraftMode() && super.getRequest().getPrincipal().hasRealm(claim.getAssistanceAgent());

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

Conclusión:

La cobertura obtenida se considera adecuada para un entorno de pruebas funcionales. Las líneas no cubiertas o parcialmente cubiertas se deben a decisiones de diseño o a condiciones muy específicas que no afectan al comportamiento general del sistema. Por tanto, el análisis es satisfactorio y refleja una ejecución correcta del conjunto funcional.