INFORME DE PRUEBAS

Grado en Ingeniería Informática – Ingeniería del Software

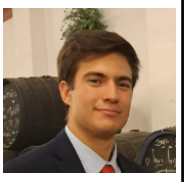
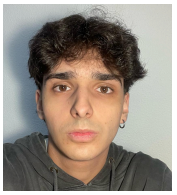
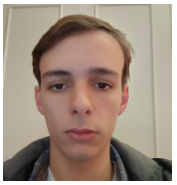
Diseño y Pruebas 2

Curso 2023-2024

SANTOS MARTÍN, JAVIER

Índice

1. Información general del proyecto

NOMBRE DEL PROYECTO			Acme-SF	
PARTICIPANTES				
Nombre	Email	Rol	Nombre de usuario	Foto
Antonio Daniel Porcar Aragón	antporara@alum.us.es	Project Manager, Desarrollador	antporara	
Francisco Miguel Jiménez Morales	frajimmor2@alum.us.es	Tester, Desarrollador	frajimmor2	
Javier Santos Martín	javsanmar5@alum.us.es	Desarrollador, Secretario	javsanmar5	
Javier Ruiz Garrido	javruigar2@alum.us.es	Analista, Desarrollador	Javiruizg	
José García de Tejada Delgado	josgardel8@alum.us.es	Operador, Desarrollador	JoseGTD	
Interesados				
Francisco Miguel Jiménez Morales, Javier Ruiz Garrido, José García de Tejada Delgado, Javier Santos Martín, Antonio Daniel Porcar Aragón and José González Enriquez (the professor).				
Fecha de inicio	Fecha esperada de completado	Entregables	Fecha del documento	
12/02/2024	27/05/2024	4	04/07/2024	

2. Resumen

El presente documento tiene como objetivo realizar un análisis exhaustivo de los distintos tests llevados a cabo sobre el proyecto Acme SF del grupo G004, en concreto, los requisitos individuales del estudiante #4.
Se abordarán: *testing funcional* y *testing de rendimiento*.

3. Tabla de revisión

Versión	Descripción	Fecha
v1.0	Versión inicial	04/07/2024

4. Introducción

Este informe de testing se centra en el análisis de las pruebas realizadas en relación con el estudiante #4. Las pruebas están organizadas en dos categorías principales.

Testing funcional
Estas pruebas buscan confirmar que el sistema se comporta según lo esperado. Para llevar a cabo estas pruebas, se utilizan dos tipos de archivos:

- **Archivos con extensión .safe:** Se utilizan para verificar tanto casos positivos como negativos.

- **Archivos con extensión .hack:** Se emplean para evaluar la resistencia del sistema frente a comportamientos no deseados o malintencionados.

Testing de rendimiento
Estas pruebas tienen como objetivo garantizar que el sistema funcione dentro de los parámetros de rendimiento establecidos. En esta sección, se analizarán los intervalos de confianza y se formularán hipótesis basadas en los resultados obtenidos de dos pruebas diferentes.

5. Contenidos

5.1 Pruebas funcionales

A continuación, se hará un resumen de los casos de prueba realizados, destacando aquellos que identificaron fallos en la implementación de las funcionalidades. Para cada prueba que no superó los criterios esperados o mostró resultados imprevistos, se realizó una corrección del error y se repitió la prueba para verificar que todo funcionara correctamente.

Sponsorship

- **list-mine.safe**
  Buscamos comprobar que un patrocinador pueda listar sus patrocinios.
  Iniciamos sesión como *sponsor1* y listamos sus patrocinios, cerramos sesión e iniciamos como *sponsor2* y repetimos el proceso.
  No se encuentra ningún error.

- **list-mine.hack**
  Buscamos verificar que un usuario, que en principio no debería, pueda realizar un listado de patrocinios.
  Abrimos la aplicación y sin estar autenticados pegamos la url del listado de sponsorship (*/sponsor/sponsorship/list-mine*), recibimos un ERROR 500.
  Ahora si nos autenticamos como *sponsor2* e intentamos acceder al anterior enlace se listarán los patrocinios de este usuario, y no los de otro patrocinador.
  No se ha encontrado ningún bug.

- **show.safe**
  Tratamos de verificar que un patrocinador puede mostrar la información completa de todos sus patrocinios alojados en el listado.
  Iniciamos sesión como *sponsor1*, listamos los patrocinios y accedemos a un patrocinio con la intención de mostrar toda su información, repetimos este proceso para distintos patrocinios.
  No se ha encontrado ningún problema.

- **show.hack**
  La intención es comprobar que un usuario pueda mostrar información sobre patrocinios a los que, en un principio, no debería poder.
  Sin autenticarnos tratamos de acceder a la url para poder mostrar un patrocinio dado su id (/*sponsor/sponsorship/show?id={sponsorshipId}*), recibimos un ERROR 500. Ahora repetimos el mismo proceso iniciando sesión como *sponsor2* e intentamos acceder a un patrocinio perteneciente a otro usuario (*sponsor1),* recibimos un ERROR 500.
  No se encuentran errores.

- **create.safe**
  Hemos de verificar que un patrocinador puede crear patrocinios siempre que estos contengan datos válidos.
  Iniciamos sesión como *sponsor1* y accedemos al formulario de creación de patrocinios, e intentamos todo tipo de datos inválidos, desde todo el formulario vacío hasta formularios correctos pero haciendo inválidos los datos de cada una de las propiedades invidualmente.
  Comprobamos que no tenía sentido tener que introducir a mano la “Fecha de instanciación” y que esta debería registrarse automáticamente a la hora de crear una instancia de patrocinio. Lo modificamos.

- **create.hack**
  Tratamos de comprobar que un usuario no autenticado como patrocinador no pueda crear patrocinios.
  Accedemos al enlace de creación de patrocinios sin antes autenticarnos, recibimos un ERROR 500.
  No encontramos errores.

- **update.safe**
  Comprobamos que un patrocinador puede modificar patrocinios siempre que estos contengan datos válidos y no estén publicados.
  Iniciamos sesión como *sponsor1* y accedemos a un patrocinio no publicado, modificamos los datos siguiendo el mismo procedimiento que en *create.safe,* tras esto, accedemos a un patrocinio publicado y comprobamos que no tenemos la opción de modificarlo.
  Se encuentra un problema a la hora de modificar la cantidad del patrocinio, a la hora de crear o actualizar facturas no permitimos que tengan divisas distintas a las del patrocinio con el que están asociadas, sin embargo, permitíamos modificar la divisa desde el patrocinio, teniendo ya facturas asociadas.
  Decidimos no permitir esta acción.

- **update.hack**
  Buscamos asegurar que un usuario, que no deba, no pueda modificar los valores de un patrocinio.
  Intentamos acceder al enlace de actualizar patrocinios (*/sponsor/sponsorship/update*) sin estar autenticados, recibimos un ERROR 500. Ahora nos registramos y accedemos al mismo enlace desde la url directamente en lugar de desde su botón, recibimos otro ERROR 500 al no especificar el patrocinio.
  No damos la opción de escoger el objeto a actualizar desde la URL por tanto no podemos realizar más intento de hackeo de esta opción, en nuestro caso el objeto se especifica en el campo *$data* de la request.
  No se encuentran problemas.

  *Explicada la forma de manipular una petición en **A destacar.***

- **delete.safe**
  Tratamos de comprobar que un patrocinador puede eliminar cualquiera de los patrocinios siempre que no estén publicados y todas sus facturas tampoco.
  Iniciamos sesión como *sponsor1* accedemos a algunos patrocinios y los borramos con su botón correspondiente, después, accedemos a un patrocinio publicado y vemos que no aparece el botón de eliminar.
  Se encuentra un problema, podíamos eliminar patrocinios con facturas publicadas, eliminando así sus facturas asociadas, lo que viola el siguiente requisito: "Update or delete an **invoice** as long as it is not published.". Modificamos el servicio de tal forma que no permita eliminar un patrocinio en esta situación.

- **delete.hack**
  Verificamos que un usuario indebido pueda eliminar patrocinios.
  Accedemos a */sponsor/sponsorship/delete* sin estar autenticados, devuelve un ERROR 500. Ahora nos registramos y accedemos al mismo enlace desde la url directamente en lugar desde su botón, recibimos otro ERROR 500 al no especificar el patrocinio, no damos forma de especificarla desde la URL.
  No se encuentran problemas.

  *Explicada la forma de manipular una petición en **A destacar.***

- **publish.safe**
  Tratamos de comprobar que un patrocinador pueda publicar uno de sus patrocinios si todas sus facturas están publicadas.
  Iniciamos sesión como *sponsor1* y creamos un patrocinio, dentro de este creamos una factura con la misma divisa y la misma cantidad total que el patrocinio; volvemos al patrocinio y lo publicamos con éxito. Además, probamos el resto de casos como publicarlo con datos incorrectos (como en el testeo del servicio *create*), con facturas no publicadas y con facturas que no sumen la cantidad del patrocinio, recibiendo en todos los casos un mensaje de que no se puede publicar.
  Por último, accedemos a un patrocinio publicado y comprobamos que no aparece el botón de publicar.
  No se han encontrado errores.

- **publish.hack**
  Aseguramos que no podemos publicar un patrocinio de maneras ilegales.
  Primero hacemos una peticion a la uri */sponsor/sponsorship/publish* y recibimos un ERROR 500, tras esto nos autenticamos como *sponsor1* y realizamos la misma petición consiguiendo el mismo resultado.
  Como en los casos de delete y publish no permitimos escoger el patrocinio desde la uri, por lo que no podemos probar más urls con intento de hackeo.
  No se encuentran problemas.

  *Explicada la forma de manipular una petición en **A destacar.***

Invoice

- **list.safe**
  Buscamos comprobar que un patrocinador pueda listar las facturas asociadas a un patrocinio.
  Iniciamos sesión como *sponsor1* y listamos sus patrocinios, desde aquí accedemos a uno de estos y por último a sus facturas, realizamos este proceso para algunos patrocinios, cerramos sesión e iniciamos como *sponsor2* y repetimos el proceso.
  No se encuentra ningún error.

- **list.hack**
  Buscamos verificar que un usuario, que en principio no debería, pueda realizar un listado de facturas.
  Abrimos la aplicación y sin estar autenticados pegamos la url del listado de facturas asociadas a un patrocinio (*/sponsor/invoice/list?sponsorshipId={sponsorshipId}*), recibimos un ERROR 500.
  Ahora, nos autenticamos como *sponsor2* e intentamos acceder al anterior enlace sabiendo que el patrocinio pasado como parametro en la URL pertenece a otro usuario (*sponsor1*) y recibimos otro ERROR 500.
  No se ha encontrado ningún bug.

- **show.safe**
  Tratamos de verificar que un patrocinador puede mostrar la información completa de todas sus facturas.
  Iniciamos sesión como *sponsor1*, listamos los patrocinios y accedemos a un patrocinio, ahora listamos sus facturas y accedemos a estas. Repetimos este proceso para distintos patrocinios y facturas.
  No se ha encontrado ningún problema.

- **show.hack**
  La intención es comprobar que un usuario pueda mostrar información sobre facturas a las que, en un principio, no debería poder.
  Sin autenticarnos realizamos una petición a la uri del show de una factura, recibimos un ERROR 500. Tras esto, iniciamos sesión como *sponsor2* y realizamos una petición igual sobre una factura asociada a otro sponsor, recibimos otro error 500.
  No se encuentran errores.

- **create.safe**
  Hemos de verificar que un patrocinador puede crear facturas con datos válidos y sobre un patrocinio que no esté ya publicado.
  Iniciamos sesión como *sponsor1* y accedemos al formulario de creación de facturas, e intentamos todo tipo de datos inválidos, desde todo el formulario vacío hasta formularios correctos pero haciendo inválidos los datos de cada una de las propiedades invidualmente. Una vez comprobados todos los casos ilegales, realizamos intentos válidos.

No se encontran problemas.

- **create.hack**
  Debemos comprobar que un un usuario no pueda realizar acciones ilegales sobre la creación de facturas.
  Primero, realizamos la petición a la URL sin estar autenticados, recibimos un ERROR 500. Realizamos el mismo proceso con un usuario logueado y recibimos lo mismo.
  En este caso, otra acción ilegal sería crear una factura sobre un patrocinio ya publicado, habíamos dejado el botón de crearla y generamos un “error” a la hora de hacer el POST, pero al tratarse de una acción ilegal consideramos mejor opción eliminar el botón y devolver un ERROR 500 al acceder manualmente a dicha uri.

- **update.safe**
  Comprobamos que un patrocinador puede modificar facturas siempre que estas contengan datos válidos y no estén publicadas.
  Iniciamos sesión como *sponsor1* y accedemos a una factura no publicada, y probamos a variar sus datos de tantas formas distintas como en el *create*, una vez comprobados, realizamos actualizaciones válidas, y comprobamos que no tenemos la opción de actualizar una factura ya publicada.
  No encontramos problemas.

- **update.hack**
  Buscamos asegurar que un usuario no pueda realizar actualizaciones ilegales sobre facturas.
  Intentamos acceder al enlace de actualizar patrocinios sin estar autenticados, recibimos un ERROR 500. Ahora nos registramos como *sponsor2* e intantamos lo mismo, recibiendo otro ERROR 500.
  No se encuentran problemas.

  *Explicada la forma de manipular una petición en **A destacar.***

- **delete.safe**
  Tratamos de comprobar que un patrocinador puede eliminar cualquiera de sus facturas siempre que no estén publicadas.
  Iniciamos sesión como *sponsor1* accedemos a algunas facturas y las borramos con su botón correspondiente, después, accedemos a una factura publicada y vemos que no aparece el botón de eliminar.
  En este caso, a priori, no teníamos la opción de realizar una petición incorrecta, pues, aparentemente, y así hacer uso del método *unbind* para poder mostrar los datos de nuevo. Al final descubrí que si introducimos datos inválidos, no por parte de nuestro backend (como valores nulos), si no por parte de java directamente, tratar de añadir la cadena “*Lorem*” a una propiedad de tipo *Date*, podíamos obtener este error y así utilizar el método *unbind*.

- **delete.hack**
  Verificamos que un usuario indebido pueda eliminar facturas.
  Accedemos a */sponsor/invoice/delete* sin estar autenticados, devuelve un ERROR 500. Ahora nos registramos y accedemos al mismo enlace desde la url directamente en lugar desde su botón, recibimos otro ERROR 500 al no especificar el patrocinio, no damos forma de especificarla desde la URL.
  No se encuentran problemas.

  *Explicada la forma de manipular una petición en **A destacar.***

- **publish.safe**
  Tratamos de comprobar que un patrocinador pueda publicar una de sus facturas con datos correctos.
  Iniciamos sesión como *sponsor1* y nos dirigimos a un patrocinio sin publicar, publicamos algunas de sus facturas, ahora modificamos los datos de una de manera incorrecta e intentamos publicar, de igual manera que con el *create,* comprobando todas las combinaciones.
  No se encuentran problemas.

- **publish.hack**
  Aseguramos que no podemos publicar un patrocinio de maneras ilegales.
  Primero hacemos una peticion a la uri correspondiente sin autenticarnos, recibiendo un ERROR 500. Tras esto nos registramos como *sponsor2* e intentamos lo mismo y recibimos el mismo resultado.
  De forma análoga a los intentos de hackeo anteriores no podemos tratar de manipular el objeto a publicar desde la URL.
  No se encuentran problemas

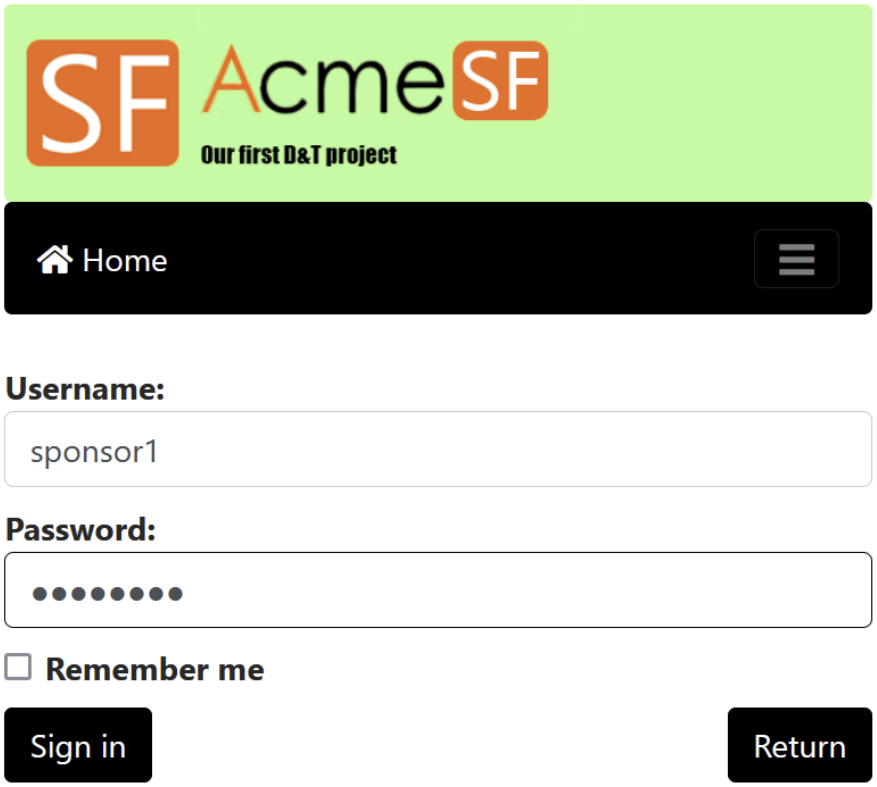
  *Explicada la forma de manipular una petición en **A destacar.***

A destacar

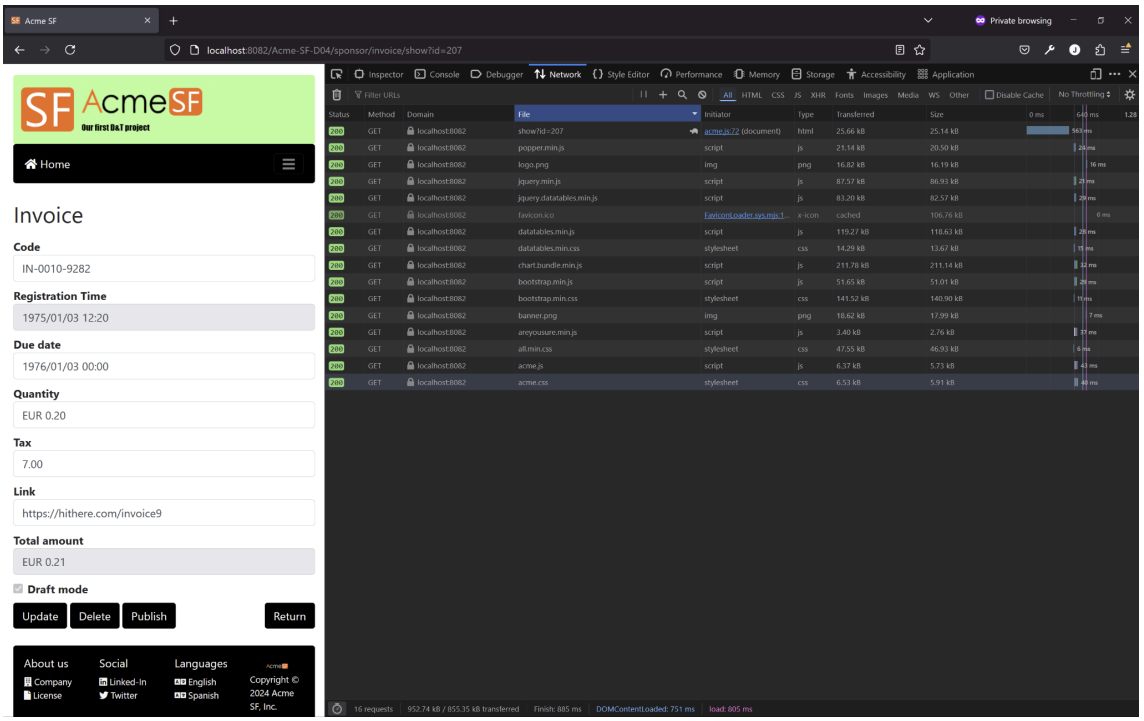
Como hemos visto en las pruebas anteriores existen muchos casos en los que no permitimos hackear desde la uri, puesto que no especificamos el objeto al que realizar la acción en esta, si no que, como explicamos también anteriormente el objeto se especifica en el campo *$data* de la request, ya sea diciendo la Id o adjuntando el objeto en sí.
Para poder ponernos en un caso real de intento de hackeo se han utilizado otras herramientas para poder modificar manualmente una petición, podríamos haber utilizado softwares como **Postman**, pero hemos optado por crear request personalizadas desde las **Herramientas de desarrollador** ofrecidas por *Firefox Developer Edition**.***
Estas pruebas se han realizado para distintos servicios pero en muchos casos, a la hora de realizar el *tester#replayer* se obtenían fallos (FAILED) en los banners cuando se utilizaban estas técnicas. Por tanto, se han tenido que repetir dichos tests sin incluir estas mismas.

Ejemplificaré estas técnicas para un caso concreto, pero este proceso es escalable a cualquier servicio, en nuestro caso: **actualizar una factura ya publicada**

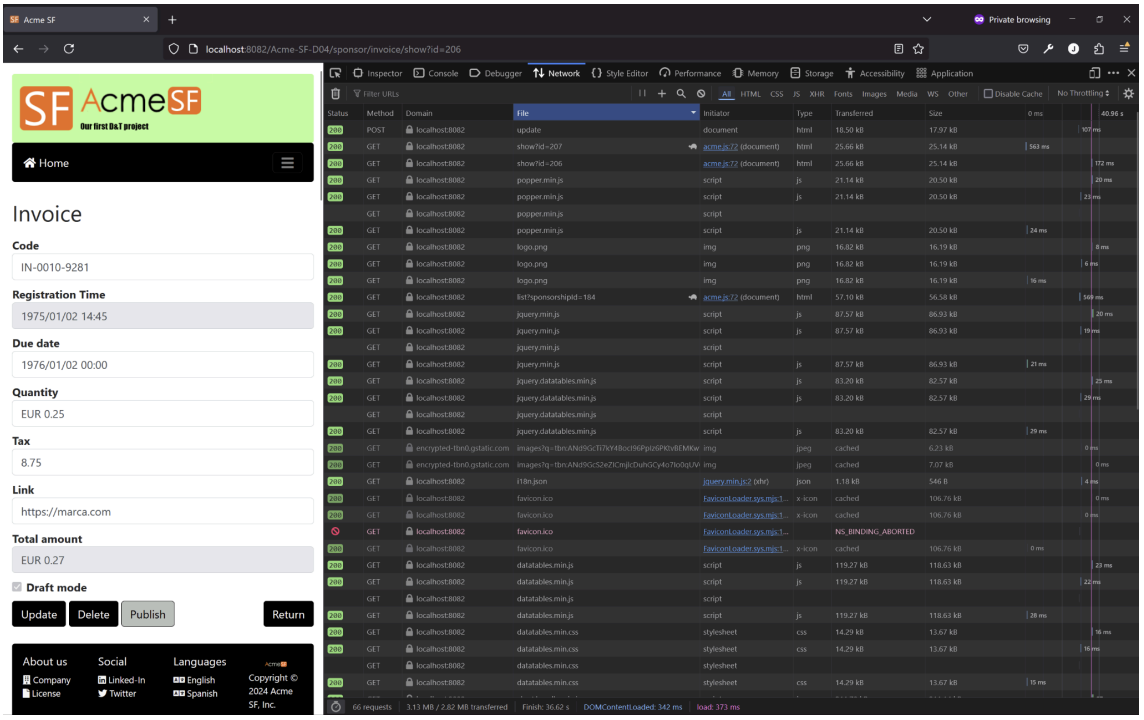
Iniciamos sesión como *sponsor1*



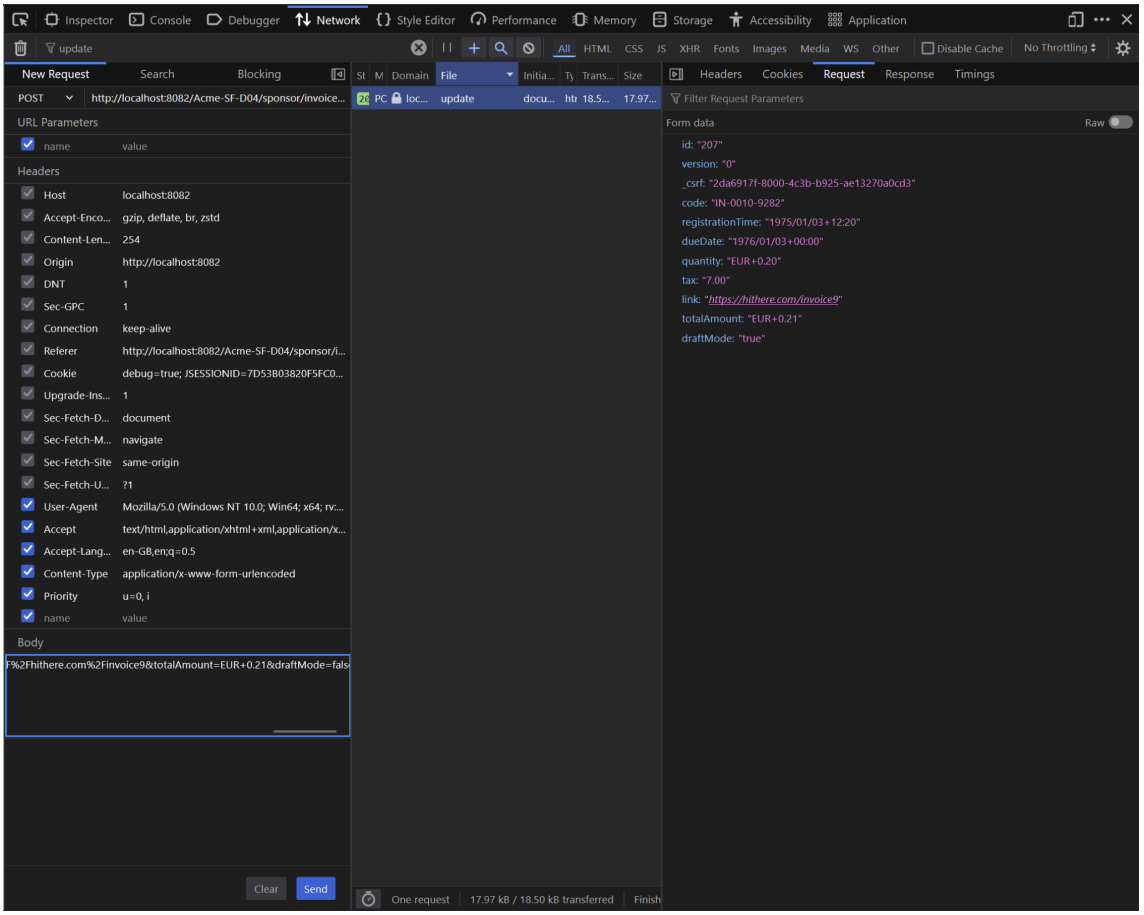
Mostramos los datos de una factura y realizamos una actualización para tomarla como referencia (esto último es opcional, podríamos realizar la petición a mano)



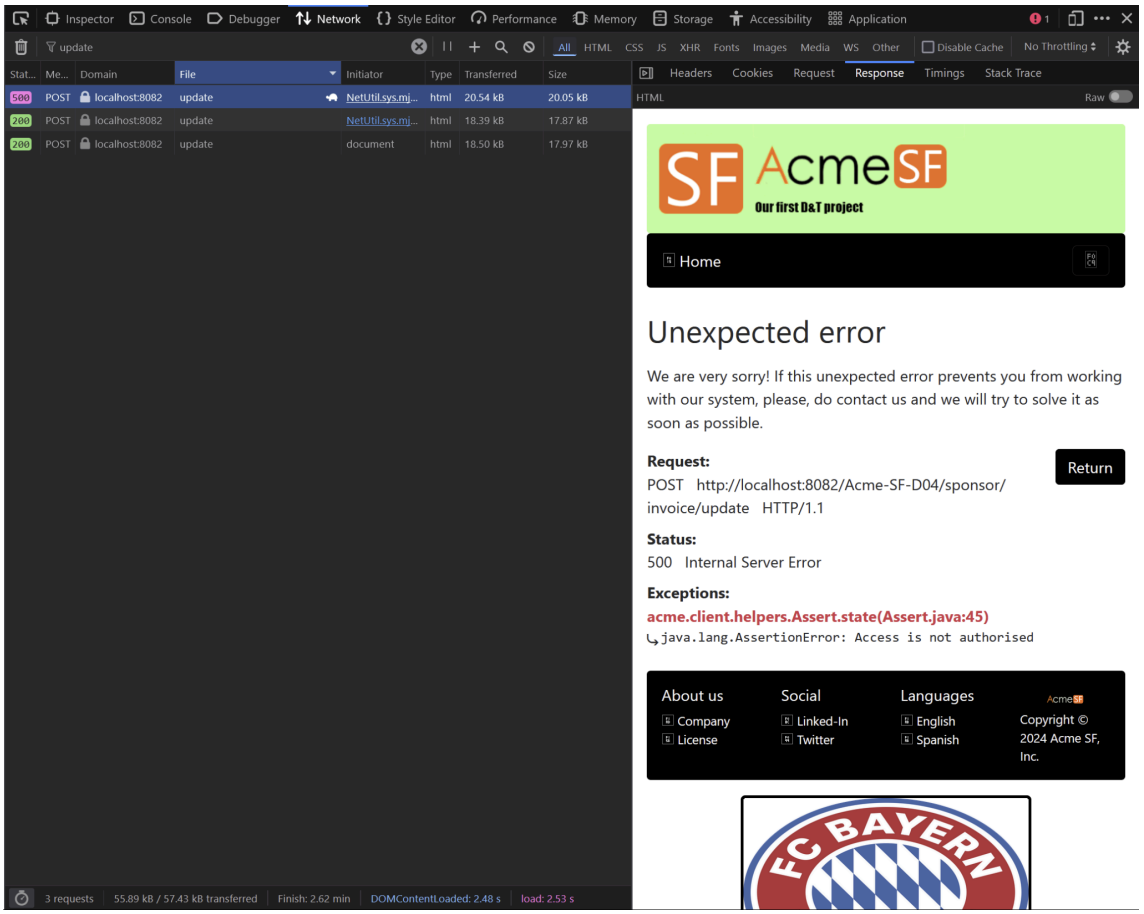
Publicamos la factura de forma que no se pueda modificar



Creamos una nueva request añadiendo en el cuerpo el objeto que acabamos de publicar, pero en esta modificamos el parámetro *draftMode* a *false* puesto que se acaba de cambiar al publicar



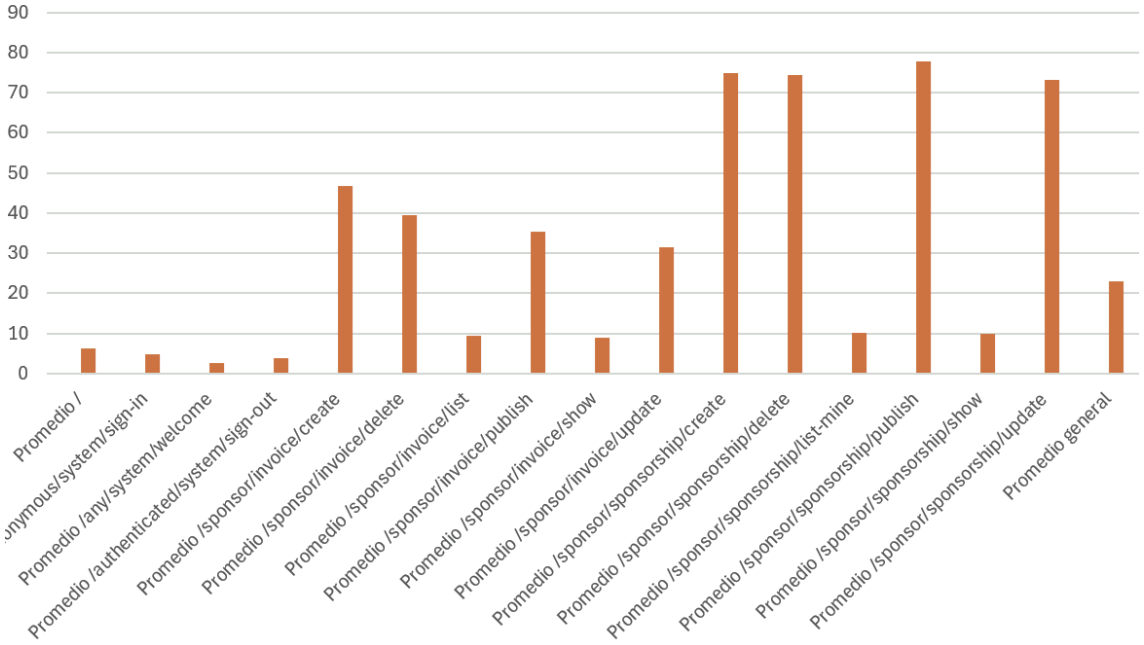
Al enviar la petición obtenemos un ERROR 500 ya que hemos protegido este petición.



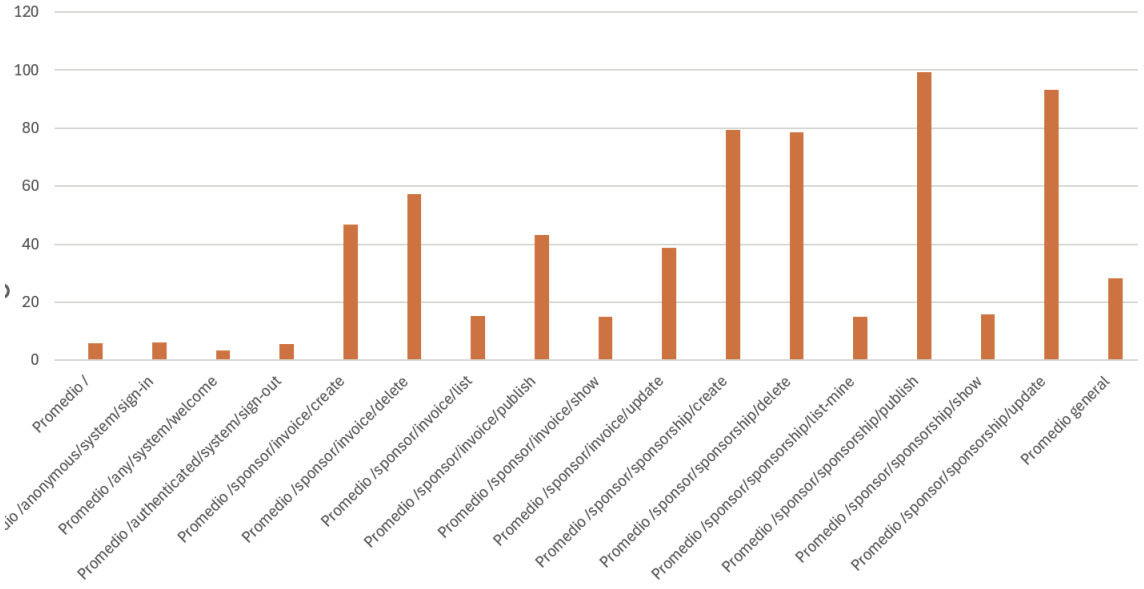
Como se ha mencionado anteriormente este proceso se podría emplear para intentar hackear más servicios y de distintas formas. Sin embargo, hay otros que ni siquiera de esta forma podríamos tratar de hackear y no tenemos que protegernos, por ejemplo tratar de acceder a patrocinios que de *sponsor1* desde *sponsor2* haciendo uso del servicio *list-mine*, ya que en este el id empleado en el filtro del repositorio se manda filtrado en la petición y no el cuerpo como lo explicado anteriormente.

5.2 Pruebas de rendimiento

Pruebas previas a la refactorización con índices



Pruebas posteriores a la refactorización con índices



Análisis: Intervalos de confianza

Antes				*Despues*		
Media	23.0209148			Media	28.2605166	
Error típico	1.32591452			Error típico	1.48779894	
Mediana	7.5529			Mediana	11.143	
Moda	2.5327			Moda	2.3393	
Desviación estándar	31.4048669			Desviación estándar	35.2391703	
Varianza de la muestra	986.265667			Varianza de la muestra	1241.79912	
Curtosis	3.19384381			Curtosis	2.42030012	
Coeficiente de asimetría	1.88304258			Coeficiente de asimetría	1.70705756	
Rango	181.6197			Rango	200.0433	
Mínimo	1.3019			Mínimo	1.4699	
Máximo	182.9216			Máximo	201.5132	
Suma	12914.7332			Suma	15854.1498	
Cuenta	561			Cuenta	561	
Nivel de confianza(95.0%)	2.6043735			Nivel de confianza(95.0%)	2.92234836	
Interval(ms)	20.4165413	25.6252883		Interval(ms)	25.3381682	31.1828649
Interval(s)	0.02041654	0.02562529		Interval(s)	0.02533817	0.03118286

Podemos observar un intervalo de confianza [20.4165413, 25.6252883] para la primera muestra de datos, los datos antes de la refactorización en este caso, y un intervalo [25.3381682, 31.1828649] para el segundo conjunto de datos, en este caso: los datos tras la refactorización de índices, ambos medidos en milisegundos. Ambas medidas son suficientemente aceptables, y el hecho de que no haya disminuido tras el proceso de refactorización de con índices puede no ser significativo y deberse a distintos factores, como por ejemplo que se hayan realizado pocos intentos, y con una muestra de mayor tamaño se hubiese visto esta mejora esperada.

Análisis: Hipótesis de contraste

Prueba z para medias de dos muestras		
	before	*after*
Media	23.0209148	28.2605166
Varianza (conocida)	986.265667	1241.79912
Observaciones	561	561
Diferencia hipotética de las medias	0	
z	-2.6291527	
P(Z<=z) una cola	0.0042799	
Valor crítico de z (una cola)	1.64485363	
Valor crítico de z (dos colas)	0.00855979	
Valor crítico de z (dos colas)	1.95996398	

En este caso, podemos observar que para la prueba z para medias de dos muestras obtenemos un valor de P de *0.004* para un α de *0.95.* Por lo que podemos concluir que la refactorización mejoró los resultados obtenidos.

6. Conclusiones

En el informe se recogen todas las pruebas realizadas por el estudiante 4, las cuales han servido para detectar algún error en el código y calcular el adecuado funcionamiento; y poder calcular el rendimiento, pudiendo sacar así conclusiones sobre la eficiencia de nuestro sistema sirviendo las peticiones.

7. Bibliografía

Intencionalmente en blanco.