Informe de Pruebas

Grupo C2.008

https://github.com/Luis-Giraldo-Santiago3/Acme-SF-D04

Alejandro Vargas Muñiz → alevarmun1@alum.us.es
David Vargas Muñiz → davvarmun@alum.us.es
Luis Giraldo Santiago → luisgirsan@gmail.com
Rafael Molina García → rafmolgar2@alum.us.es

04/07/2024

TABLA DE CONTENIDOS

1. RESUMEN EJECUTIVO

El presente informe tiene como objetivo proporcionar un análisis general de sobre los casos de pruebas aometidas al codigo realizado por el Student#2.

2. TABLA DE REVISIÓN

Fecha	Versión	Descripción
04/07/2024	1.0	TestingReport Student#2

3. INTRODUCCIÓN

El presente informe presenta un análisis sobre los casos de pruebas realizado al Student#2. Este informe se centra en analizar la cobretura de los casos de pruebas y la optimización del código.

4. Contenido

Capítulo 1: Pruebas Funcionales

Introducción

Este informe presenta los resultados de las pruebas funcionales y de rendimiento realizadas en el proyecto "Acme-SF" respecto al Student#2. Las pruebas funcionales verifican la efectividad en la detección de errores, mientras que las pruebas de rendimiento evalúan el tiempo de respuesta en diferentes computadoras y entre tener o no tener indices. Los resultados ofrecen una visión sobre la estabilidad y eficiencia del proyecto.

Casos de Prueba Implementados para "Contract":

Característica 1: Listado de contratos ("ClientContractListService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes listar sus propios contratos almacenados en el sistema. Esta característica debe asegurarse de que solo se muestren los proyectos asociados con el cliente autenticado.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 94,9%.
  Al ejecutar el "replayer" observamos que todas las líneas están en verde, por lo que se ha entrado y se ha ejecutado todo correctamente, de ahí el alto porcentaje.

Característica 2: Detalles de contratos ("ClientContractShowService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes ver los detalles de sus propios contratos almacenados en el sistema. Esta característica debe asegurarse de que solo se muestren los detalles de los contratos asociados con el cliente autenticado.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 95,8%.
  Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo:
  - Líneas amarillas:
    - **Línea 36:** `client = contract == null ? null : contract.getClient();` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un cliente.
    - **Líneas 37:** `status = super.getRequest().getPrincipal().hasRole(client) && contract != null;` Esta línea está en amarillo porque el contrato nunca va a ser nulo.
    - **Línea 55:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

Característica 3: Crear proyecto ("ClientContractCreateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes crear nuevos contratos en el sistema. Esta característica debe asegurarse de que solo los clientes autenticados puedan crear contratos, que cuando se crea el contrato no se ponga un código que tenga otro contrato ya creado, que el nombre del proveedor y el nombre del cliente esté entre 0 y 75 caracteres, que la

meta esté entre 0 y 100 caracteres, que el presupuesto esté entre 0 y 10000 horas y tiene que tener asociado un proyecto. Todos los atributos son obligatorios.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 93,5%.
  Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo:
  - Líneas amarillas:
    - **Línea 70:** `if (!super.getBuffer().getErrors().hasErrors("budget"))` Esta línea está en amarillo, porque cuando llegamos a este if, siempre se ha dado el caso de que había algún budget y no se ha probado que pasaría si enviamos una petición con un proyecto pero sin presupuesto, haciendo la otra parte del if.
    - **Líneas 83 y 84:** `super.state(object.getBudget() <= 10000, "budget", "client.contract.form.error.higher-hour");` `super.state(object.getBudget() >= 0, "budget", "client.contract.form.error.lower-hour");` Esta línea está en amarillo porque como el atributo esta acotado con un minimo y un máximo no es posible salirse de los límites y no se ejecutan, pero se han probado los casos -1 y 10001 y salta error.
    - **Línea 52, 66, 90 y 97:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

Característica 4: Actualizar contratos ("ClientContractUpdateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes actualizar los contratos en el sistema. Esta característica debe asegurarse de que solo los clientes autenticados puedan actualizar contratos que no esté publicados, que cuando se actualiza el contrato no se ponga un código que tenga otro contrato ya creado, que el nombre del proveedor y el nombre del cliente esté entre 0 y 75 caracteres, que la meta esté entre 0 y 100 caracteres, que el presupuesto esté entre 0 y 10000 horas y tiene que tener asociado un proyecto. Todos los atributos son obligatorios.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 93,0%.
  Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo:
  - Líneas amarillas:
    - **Línea 37:** `client = contract == null ? null : contract.getClient();` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un cliente.
    - **Líneas 38:** `status = contract != null && !contract.isPublished() && super.getRequest().getPrincipal().hasRole(client);` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null".
    - **Línea 75:** `if (!super.getBuffer().getErrors().hasErrors("budget"))` Esta línea está en amarillo, porque cuando llegamos a este if, siempre se ha dado el caso de que había algún budget y no se ha probado que pasaría si enviamos una petición con un proyecto pero sin presupuesto, haciendo la otra parte del if.

- **Líneas 87 y 88:** `super.state(object.getBudget() <= 10000, "budget", "client.contract.form.error.higher-hour"); super.state(object.getBudget() >= 0, "budget", "client.contract.form.error.lower-hour");` Esta línea está en amarillo porque como el atributo esta acotado con un minimo y un máximo no es posible salirse de los límites y no se ejecutan, pero se han probado los casos -1 y 10001 y salta error.
- **Líneas 57, 71, 94 y 101:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
- **Línea 83 y 84:** `final Contract contract2 = object.getCode().equals("") || object.getCode() == null ? null : this.repository.findOneContractById(object.getId());` La condición que siempre se cumple es que "object.getCode()" siempre va a ser distinto de null.

Característica 5: Publicar contratos ("ClientContractPublishService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes publicar los contratos en el sistema. Esta característica debe asegurarse de que solo los clientes autenticados puedan publicar contratos que no esté publicados, que cuando se publique el contrato no se ponga un código que tenga otro contrato ya creado, que el nombre del proveedor y el nombre del cliente esté entre 0 y 75 caracteres, que la meta esté entre 0 y 100 caracteres, que el presupuesto esté entre 0 y 10000 horas y tiene que tener asociado un proyecto. Todos los atributos son obligatorios. Además de lo mencionado anteriormente, el presupuesto del contrato nunca debe ser mayor al coste de un proyecto.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 93,9%.
  Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo:
  - Líneas amarillas:
    - **Línea 38:** `client = contract == null ? null : contract.getClient();` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un cliente.
    - **Líneas 39:** `status = contract != null && !contract.isPublished() && super.getRequest().getPrincipal().hasRole(client);` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null".
    - **Línea 80:** `if (!super.getBuffer().getErrors().hasErrors("budget"))` Esta línea está en amarillo, porque cuando llegamos a este if, siempre se ha dado el caso de que había algún budget y no se ha probado que pasaría si enviamos una petición con un proyecto pero sin presupuesto, haciendo la otra parte del if.
    - **Líneas 95 y 96:** `super.state(object.getBudget() <= 10000, "budget", "client.contract.form.error.higher-hour"); super.state(object.getBudget() >= 0, "budget", "client.contract.form.error.lower-hour");` Esta línea está en amarillo porque como el atributo esta acotado con un minimo y un máximo no es posible

salirse de los límites y no se ejecutan, pero se han probado los casos -1 y 10001 y salta error.

- **Líneas 58, 72, 103 y 111:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
- **Línea 90 y 91:** `final Contract contract2 = object.getCode().equals("") || object.getCode() == null ? null : this.repository.findOneContractById(object.getId());` La condición que siempre se cumple es que "object.getCode()" siempre va a ser distinto de null.

Característica 6: Borrar contrtos ("ClientContractDeleteService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes borrar contratos del sistema. Esta característica debe asegurarse de que solo los clientes autenticados puedan eliminar contratos que no esté publicados.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 57,7%.
  Al ejecutar el "analyser" observamos que hay algunas líneas en amarillo y en rojo.
  - Líneas amarillas:
    - **Línea 37:** `client = contract == null ? null : contract.getClient();` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un cliente.
    - **Líneas 38:** `status = contract != null && !contract.isPublished() && super.getRequest().getPrincipal().hasRole(client);` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null".
    - **Líneas 56, 70, 75:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
  - Líneas rojas: se corresponden al método "unbind", este método no se ejecuta debido a que no se tiene que ejecutar, ya que sirve para mostrar los datos cuando devuelve un error y en el delete no hay errores que devolver por tanto no se cargan los datos.

Casos de Prueba Implementados para "ProgressLog":

Característica 1: Listado de los registros de progreso("ClientProgressLogListService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes listar sus propios registros de progreso almacenados en el sistema. Esta característica debe asegurarse de que solo se muestren los registros de progreso asociados con el cliente autenticado y pertenezcan al contrato ya publicado.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 94,5%.
  Al ejecutar el "replayer" observamos que todas las líneas están en verde, por lo que se ha entrado y se ha ejecutado todo correctamente, de ahí el alto porcentaje.También observamos lineas amarillas.

- Líneas amarillas:
  - **Líneas 34:** `status = contract != null && contract.isPublished() && super.getRequest().getPrincipal().hasRole(contract.getClient());` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null".
  - **Líneas 52, 63:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

Característica 2: Detalles de un registro de progreso ("ClientProgressLogShowService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes ver los detalles de sus propios registros de progreso almacenados en el sistema. Esta característica debe asegurarse de que solo se muestren los detalles de los registro de progreso asociados con el cliente autenticado y pertenezcan al contrato ya publicado.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 96,6%.
  Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo:
  - Líneas amarillas:
    - **Líneas 32:** `status = contract != null && contract.isPublished() && contract.getClient().getUserAccount().getUsername().equals(super.getRequest().getPrincipal().getUsername()) && super.getRequest().getPrincipal().hasRole(contract.getClient());` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null".
    - **Líneas 50:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

Característica 3: Crear un registro de progreso ("ClientProgressLogCreateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes crear nuevos registros de progreso en el sistema. Esta característica debe asegurarse de que solo los clientes autenticados puedan crear registros de progreso y que el contrato al cual va a ir vinculados esté publicado, que cuando se crea el registro de progreso no se ponga un recordId que tenga otro registro de progreso ya creado, que tenga un completitud ente un 0 y un 100 por cien ,que el comentario esté entre 0 y 100 caracteres, que la persona responsable esté entre 0 y 75 caracteres. Todos los atributos son obligatorios.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 92,1%.
  Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo:
  - Líneas amarillas:
    - **Líneas 33:** `status = contract != null && contract.isPublished() && super.getRequest().getPrincipal().hasRole(contract.getClient());` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null".
    - **Líneas 60, 67, 80 y 87:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

Característica 4: Actualizar un registro de progreso ("ClientProgressLogUpdateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes actualizar un registro de progreso en el sistema. Esta característica debe asegurarse de que solo los clientes autenticados puedan crear registros de progreso y que el contrato al cual va a ir vinculados esté publicado, que cuando se actualice el registro de progreso no se ponga un recordId que tenga otro registro de progreso ya creado, que tenga un completitud ente un 0 y un 100 por cien ,que el comentario esté entre 0 y 100 caracteres, que la persona responsable esté entre 0 y 75 caracteres. Todos los atributos son obligatorios.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 91,5%.
  Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo:

    - Líneas amarillas:

        - **Líneas 35:** `status = contract != null && contract.isPublished() && !progressLog.isPublished() && super.getRequest().getPrincipal().hasRole(contract.getClient()) && contract.getClient().getUserAccount().getUsername().equals(super.getRequest().getPrincipal().getUsername());` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un cliente.

        - **Líneas 55, 62, 75 y 82:** `assert object != null;` La condición que siempre se cumple es que "object".

        - **Líneas 67:** `final ProgressLog progressLog2 = object.getRecordId().equals("") || object.getRecordId() == null ? null : this.repository.findOneProgressLogById(object.getId());` La condición que siempre se cumple es que "object.getRecordId() == null" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un registro de progreso.

Característica 5: Publicar un registro de progreso ("ClientProgressLogPublishService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes publicar un registro de progreso en el sistema. Esta característica debe asegurarse de que solo los clientes autenticados puedan crear registros de progreso y que el contrato al cual va a ir vinculados esté publicado, que cuando se publique el registro de progreso no se ponga un recordId que tenga otro registro de progreso ya creado, que tenga un completitud ente un 0 y un 100 por cien ,que el comentario esté entre 0 y 100 caracteres, que la persona responsable esté entre 0 y 75 caracteres. Todos los atributos son obligatorios.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 91,6%.
  Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo:

    - Líneas amarillas:

        - **Líneas 35:** `status = contract != null && !contract.isPublished() && !pstatus = contract != null && contract.isPublished() &&`

`!progressLog.isPublished() && contract.getClient().getUserAccount().getUsername().equals(super.getRequest().getPrincipal().getUsername()) && super.getRequest().getPrincipal().hasRole(contract.getClient());` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null".

- **Líneas 55, 62, 75 y 83:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
- **Líneas 67:** `final ProgressLog progressLog2 = object.getRecordId().equals("") || object.getRecordId() == null ? null : this.repository.findOneProgressLogById(object.getId());` La condición que siempre se cumple es que "object.getRecordId() == null" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un registro de progreso.

Característica 6: Borrar un registro de progreso ("ClientProgressLogDeleteService.java")

- **Descripción:** Prueba la funcionalidad que permite a los clientes borrar un resgitro de progreso del sistema. Esta característica debe asegurarse de que solo los clientes autenticados puedan eliminar registros de progresos, de lo contratos que no estén publicados.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 68,0%.
  Al ejecutar el "analyser" observamos que hay algunas líneas en amarillo y en rojo.
  - Líneas amarillas:
    - **Líneas 34:** `status = contract != null && contract.isPublished() && !progressLog.isPublished() && contract.getClient().getUserAccount().getUsername().equals(super.getRequest().getPrincipal().getUsername()) && super.getRequest().getPrincipal().hasRole(contract.getClient());` La condición que siempre se cumple es que "contract" siempre va a ser distinto a "null".
    - **Líneas 53, 60, 65:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
  - Líneas rojas: se corresponden al método "unbind", este método no se ejecuta debido a que no se tiene que ejecutar, ya que sirve para mostrar los datos cuando devuelve un error y en el delete no hay errores que devolver por tanto no se cargan los datos.

Capítulo 2: Pruebas de Rendimiento

Introducción

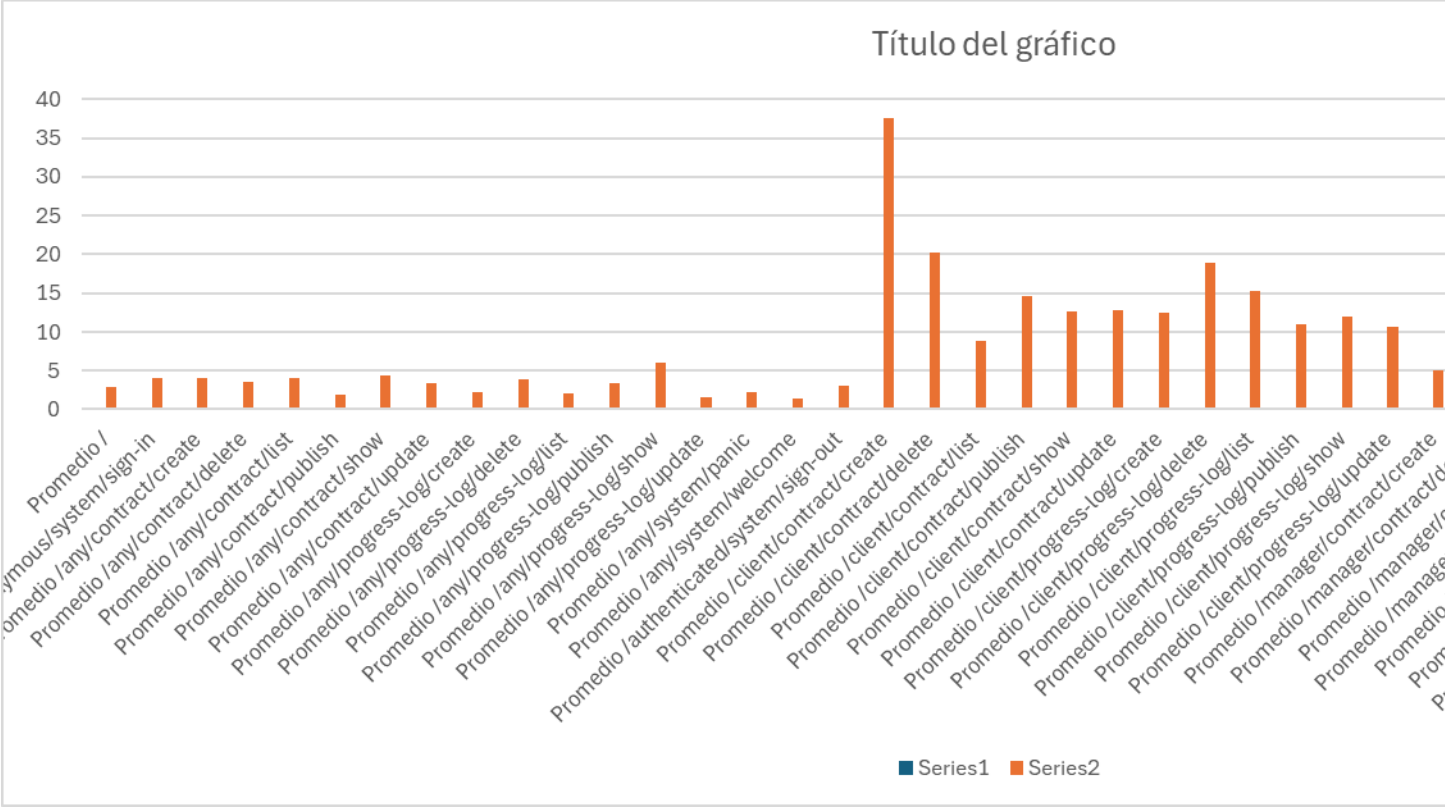
Este capítulo analiza las pruebas de rendimiento realizadas en el proyecto "Acme-SF" respecto a las características del cliente. El objetivo es evaluar el tiempo de respuesta y la capacidad del sistema bajo diferentes cargas. Utilizamos una metodología que incluye la simulación de escenarios reales y el análisis estadístico de los resultados para garantizar la eficiencia y estabilidad del sistema.

Resultados de las Pruebas de Rendimiento

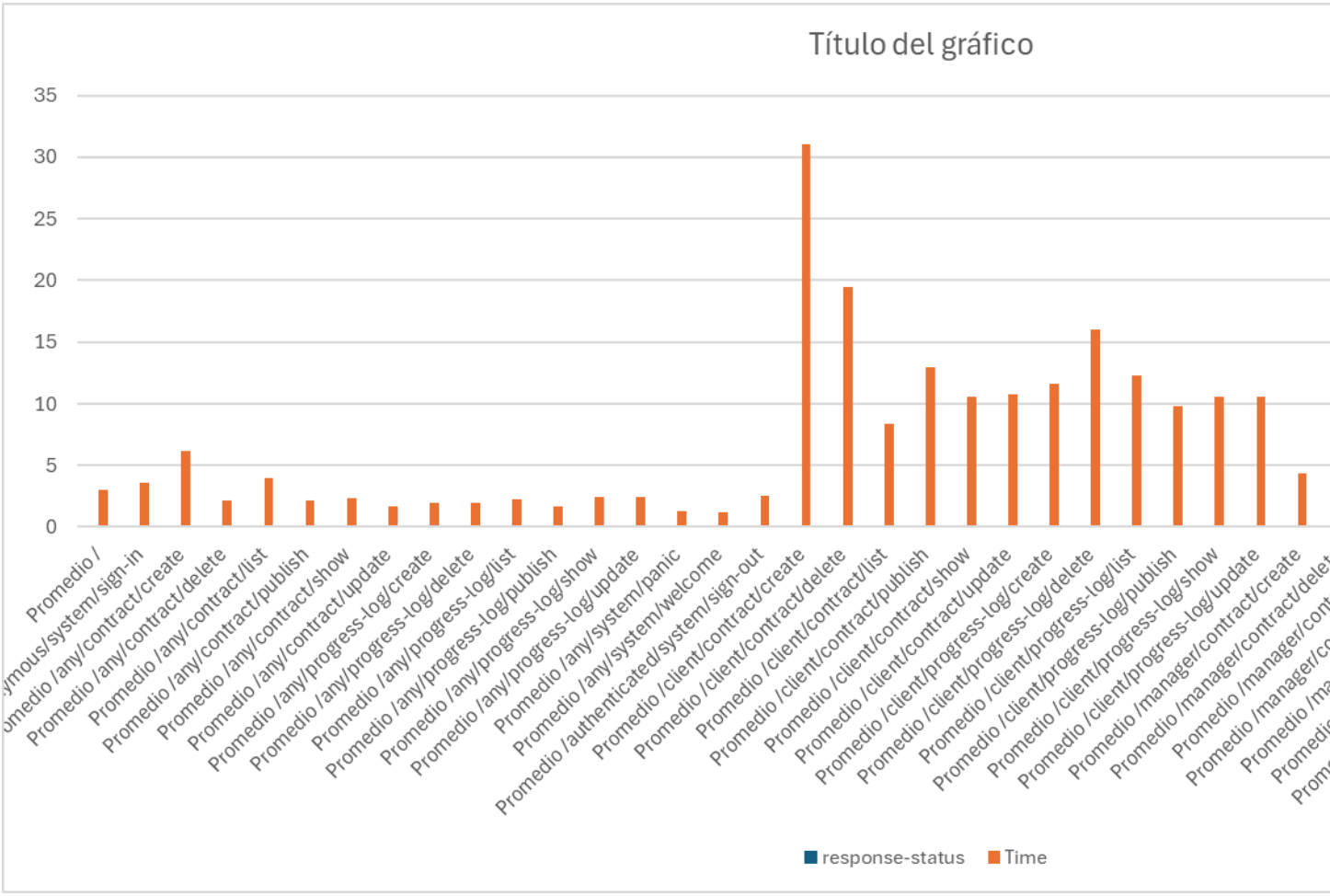
Gráficos

Gráficos del tiempo para la ejecución de las características del cliente en PC1.

- **Gráfico 1:** Gráfico del tiempo para la ejecución de las características del cliente en PC1.

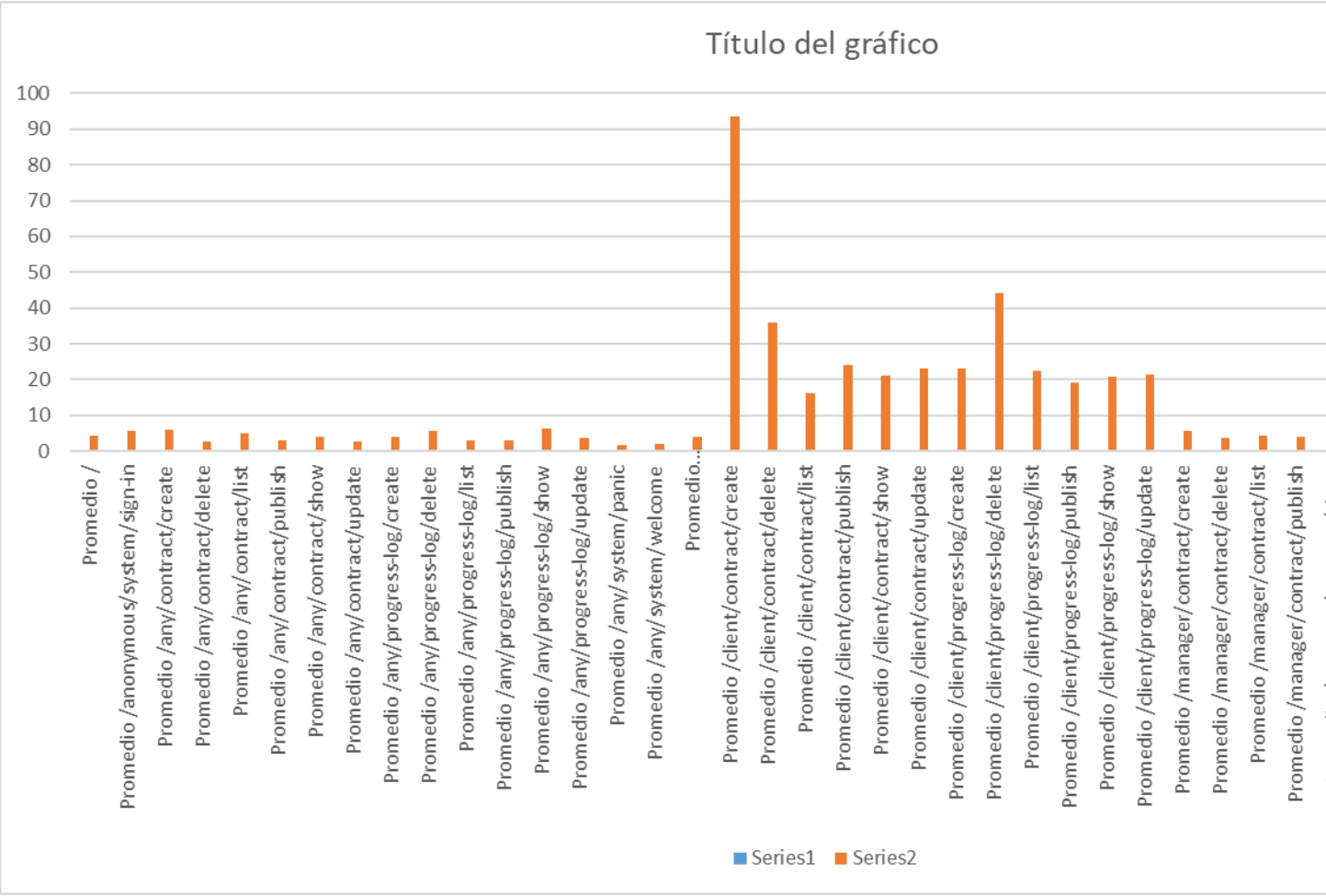


- **Gráfico 2:** Gráfico del tiempo para la ejecución de las características del cliente con optimización en PC1.



Gráficos del tiempo para la ejecución de las características del cliente en PC1.

- **Gráfico 3:** Gráfico del tiempo para la ejecución de las características del cliente con optimización en PC2.



Intervalo de Confianza

Presentar el intervalo de confianza del 95% para el tiempo de respuesta en las dos computadoras.

- PC1:
    - Nivel de confianza sin la optimización: 0,879700058
    - Nivel de confianza con la optimización: 0,79323659
- PC2:
    - Nivel de confianza con la optimización: 2,039863002

Contraste de Hipótesis

Presentar el contraste de hipótesis con un 95% de confianza sobre cuál computadora es más potente.

- **Resultado del Contraste de Hipótesis:**

- **Prueba z entre los datos del mismo PC**

Prueba z para medias de dos muestras		
	Tiempos de PC1 sin índices	*Tiempos de PC1 con índices*
Media	10,44342478	9,116483776
Varianza (conocida)	144,3566	110,6585
Observaciones	678	678
Diferencia hipotética de las medias	0	
z	2,163632581	
P(Z<=z) una cola	0,015246279	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,030492559	
Valor crítico de z (dos colas)	1,959963985	

- **Prueba z entre los datos de distintos PC**

Prueba z para medias de dos muestras		
	Tiempos de PC1 con índices	*Tiempos de PC2 con índices*
Media	9,116483776	20,19655324
Varianza (conocida)	110,6585	776,1922
Observaciones	678	678
Diferencia hipotética de las medias	0	
z	-9,687954122	
P(Z<=z) una cola	0	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0	
Valor crítico de z (dos colas)	1,959963985	

Como podemos observar el la primera imagen, al fijanos en el valor crítico de z(dos cola), vemos que nos da un 0,030492559, este número esta entre [0.00,0.05), todo esto nos indica que ha habído una mejora de tiempo al añadir índices al código.

Cuando hacemos la misma prueba con los tiempo de los dos PCs optimizados, nos da un valor de 0, esto quiere decir que al cambiar de PC a uno mejor, también optenemos una mejor del tiempo que ya habíamos mejorado.

5. CONCLUSIONES

Tras la realización de este documento se vio que la prueba al Student#2 cubren más del 80% del código hecho, notando que el más bajo en ambos caso es el de la clase del borrado.

Con respecto al tiempo, para todos los casos, las decisiones para optimizar los tiempo han sido las indicadas, ya que se mejoraron todos los tiempo en cada caso.

6. BIBLIOGRAFÍA

Intencionalmente en blanco.