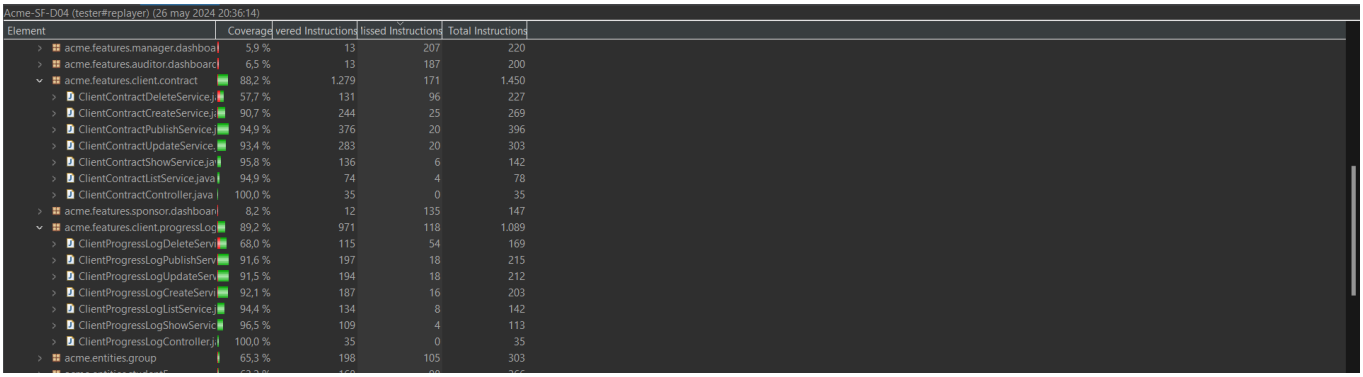
Informe de Pruebas

Capítulo 1: Pruebas Funcionales



Introducción

Este informe presenta los resultados de las pruebas funcionales y de rendimiento realizadas en el proyecto "Acme-SF" respecto a los Contracts y ProgressLog. 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. Los resultados ofrecen una visión sobre la estabilidad y eficiencia del proyecto.

Se observó que no siempre daba la misma covertura con los mismo test, algunos resultados pueden diferir a los míos, asi que que en este documento se detallarán lo que me salieron en el ultimo analisis de covertura.

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 y en rojo:
  - 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;` 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.
  - 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 y que la divisa sea "EUR" y tiene que tener asociado un proyecto. Todos los atributos son obligatorios.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 90,7%.
  Al ejecutar el "replayer" observamos que tenemos todas las líneas verdes menos todos los asserts, los cuales están en amarillo y siempre están en amarillo debido a que el objeto siempre es distinto de null, ni el perform, el cual está en rojo, esto indica que nunca se ha llegado a crear un contrator satisfactoriamente.

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 y que la divisa sea "EUR" y tiene que tener asociado un proyecto. Todos los atributos son obligatorios.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 93,4%.
  Al ejecutar el "replayer" 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 = super.getRequest().getPrincipal().hasRole(client) && contract != null;` 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 57, 71, 88 y 95:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
  - **Línea 76:** `final Contract contract2 = object.getCode().equals("") || object.getCode() == null ? null : this.repository.findOneContractById(object.getId());` a condición que siempre se cumple es que "object.getCode()" siempre va a ser distinto de null.

Característica 5: Publicar proyecto ("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 y que la divisa sea "EUR" 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:** Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo:
  - Líneas amarillas:
    - **Línea 39:** `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 40:** `status = super.getRequest().getPrincipal().hasRole(client) && contract != null;` 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 59, 73, 101 y 113:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

- **Línea 87:** `final Contract contract2 = object.getCode().equals("") || object.getCode() == null ? null : this.repository.findOneContractById(object.getId());` a condición que siempre se cumple es que "object.getCode()" siempre va a ser distinto de null.

Característica 6: Borrar banner ("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 39:** `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 40:** `status = super.getRequest().getPrincipal().hasRole(client) && contract != null;` 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 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 proyectos asociados con el cliente autenticado y pertenezcan al contrato seleccionado.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 94,4%.
  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 = super.getRequest().getPrincipal().hasRole(client) && contract`

`!= null;` 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 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 el contrato selecionado.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 96,5%.
  Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo:
  - Líneas amarillas:
  - Líneas 34: `status = contract != null && 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" por lo que siempre vamos a obtener un cliente.
  - 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 no este 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 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" por lo que siempre vamos a obtener un cliente.

- **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 no este 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 34:** `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" 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 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 no este 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 34:** `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 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() && 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 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

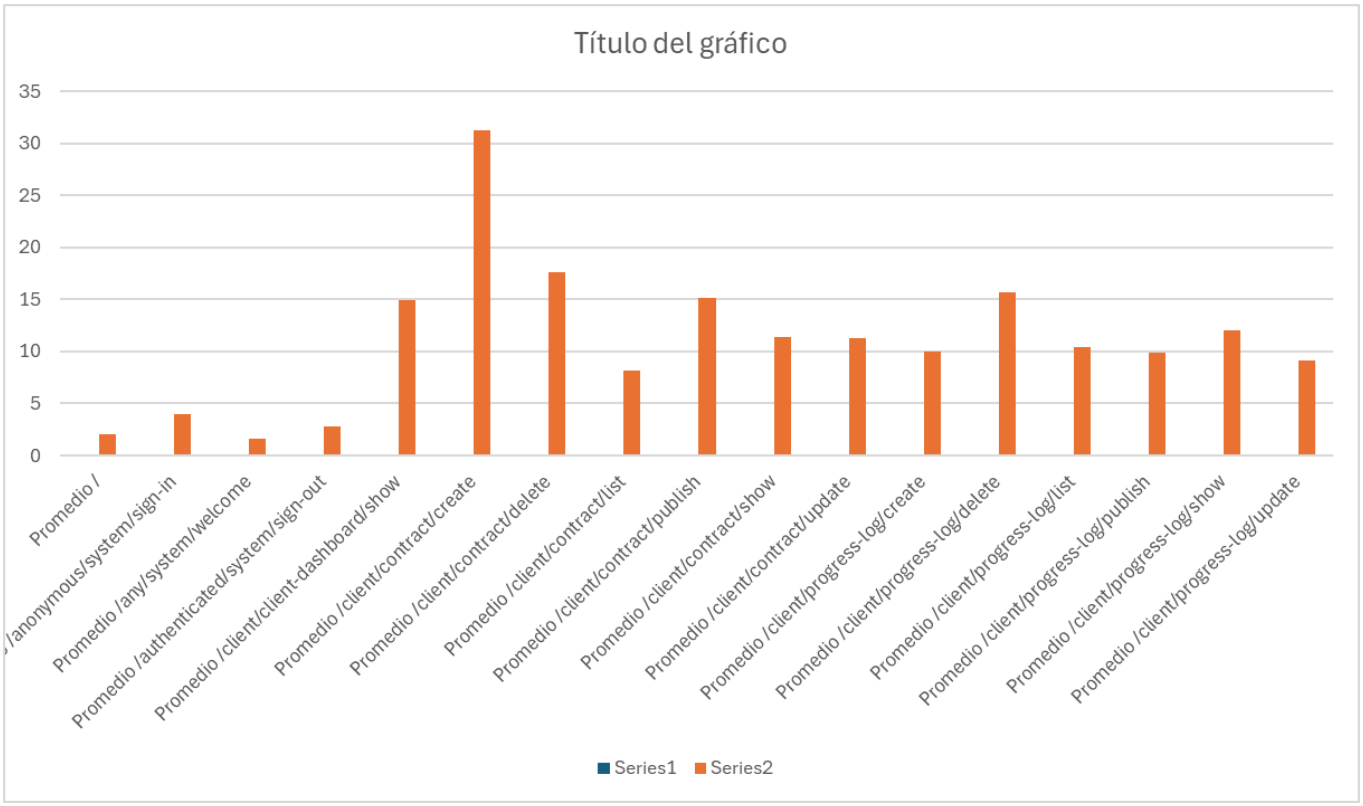
Este capítulo analiza las pruebas de rendimiento realizadas en el proyecto "Acme-SF" respecto a las características de los banners. El objetivo es evaluar el tiempo de respuesta y la capacidad del sistema bajo diferentes cargas. Utilizamos una metodología que incluye la simulación de escenarios reales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

Resultados de las Pruebas de Rendimiento

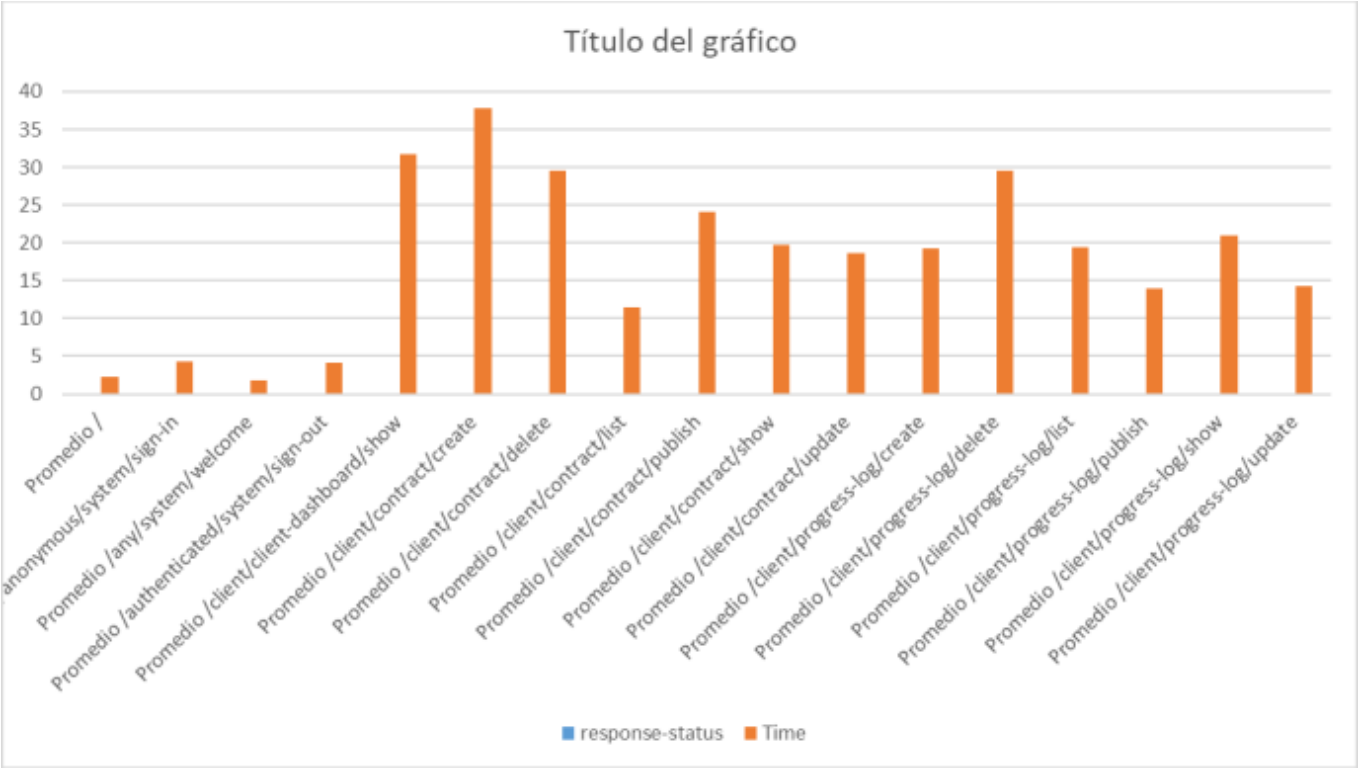
Gráficos

Gráfico del tiempo para la ejecución de las características del cliente sin la optimización 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 en PC2 con optimización.



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:**

Columna1					
			Interval(ms)	10,0350399	12,0073367
Media	11,0211883		Interval(S)	0,01003504	0,01200734
Error típico	0,50202262				
Mediana	8,4677				
Moda	#N/D				
Desviación es	11,6983119				
Varianza de la	136,850502				
Curtosis	83,4628959				
Coeficiente de	6,91010144				
Rango	176,3688				
Mínimo	1,0239				
Máximo	177,3927				
Suma	5984,50525				
Cuenta	543				
Nivel de confia	0,98614837				

- **PC2:**

- **Nivel de Confianza con la optimización:**

Columna1					
			Interval(ms)	14,3907477	16,6827378
Media	15,5367428		Interval(s)	0,01439075	0,01668274
Error típico	0,58344959				
Mediana	13,4214				
Moda	#N/D				
Desviación es	13,880706				
Varianza de la	192,674				
Curtosis	50,106217				
Coeficiente de	4,47934615				
Rango	194,505399				
Mínimo	1,1936				
Máximo	195,698999				
Suma	8793,7964				
Cuenta	566				
Nivel de confi	1,14599509				

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 para medias de dos muestras

	After	*Before*
Media	10,27695127	15,7158765
Varianza (conocida)	136,8505	192,674
Observaciones	592	633
Diferencia hipotética de las medias	0	
z	-7,432135239	
P(Z<=z) una cola	5,34017E-14	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	1,06803E-13	
Valor crítico de z (dos colas)	1,959963985	