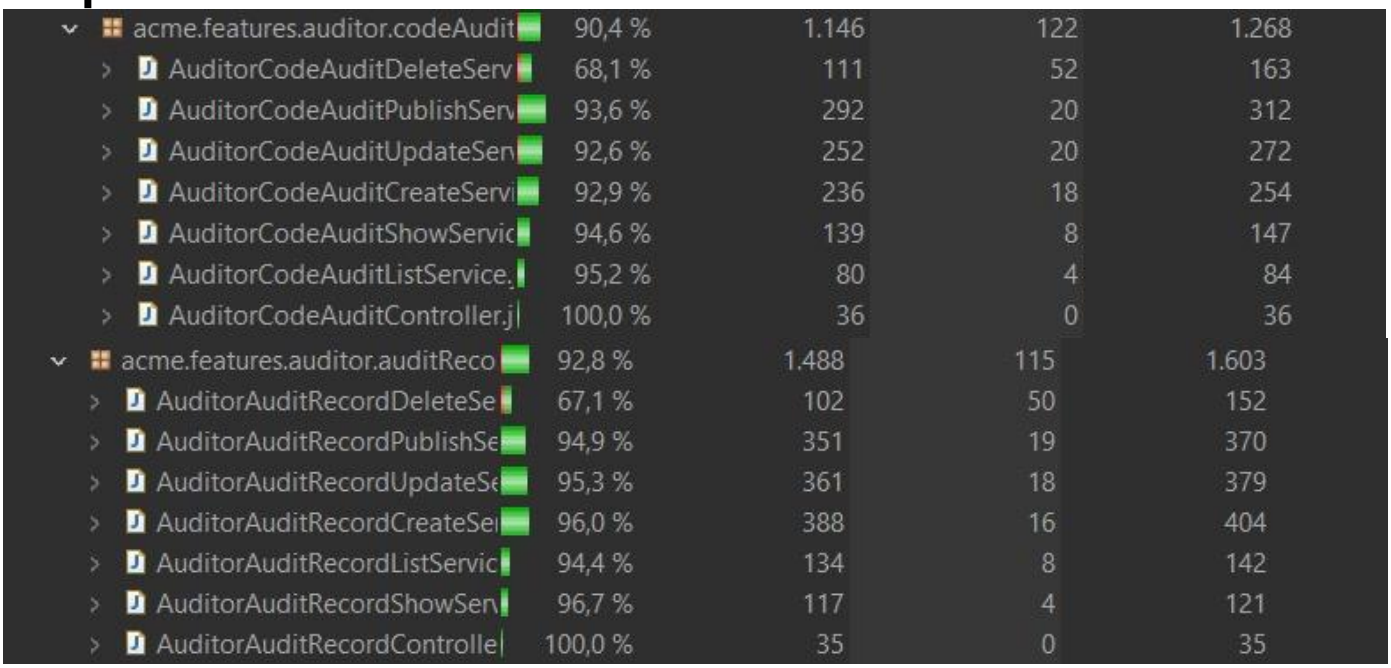
Informe de Pruebas

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 las auditorías de código y los registros de auditoría. 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.

Casos de Prueba Implementados para "CodeAudit":

Característica 1: Listado de CodeAudit ("AuditorCodeAuditListService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores listar las auditorías de código almacenadas en el sistema. Esta característica debe asegurarse de que solo se muestren las auditorías de código asociadas con el auditor autenticado.
- **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 y observamos que hay una líneas en amarillo:
  - Líneas amarillas:
  - **Línea 44:** `assert object != null;` siempre va a ser distinto de null.

Característica 2: Detalles de CodeAudit ("AuditorCodeAuditShowService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores ver los detalles de las auditorías de código almacenadas en el sistema. Esta característica debe asegurarse de que solo se muestren los detalles de las auditorías de código asociadas con el auditor autenticado.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 94,6%.
  Al ejecutar el "analyser" observamos que hay solo una línea en amarillo:
  - Líneas amarillas:
  - **Línea 36:** `auditor = codeAudit == null ? null : codeAudit.getAuditor();` La condición que siempre se cumple es que auditor siempre va a ser distinto de null.
  - **Línea 38 y 39:** `if (auditor != null){ status = super.getRequest().getPrincipal().hasRole(auditor) && auditor.getId() == auditorRequestId;` La condición que siempre se cumple es que auditor.Id igual al auditorRequestId.
  - **Línea 61:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
  - Líneas rojas:
  - **Línea 41:** `status = false;` Este so debe a que en el else que contiene esta línea de código nunca es llamada ya que status siempre va a ser true.

Característica 3: Crear CodeAudit ("AuditorCodeAuditCreateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores crear nuevas auditorías de código en el sistema. Esta característica debe asegurarse de que solo los auditores puedan crear auditorías de código, que cuando se actualiza la auditoría de código, no se ponga un código que tengo otra auditoría de código ya creada, que la fecha de ejecución sea posterior al 1999/12/31 23:59 y anterior al momento actual, que el tipo sea "Static" o "Dynamic", que las acciones correctivas estén entre 1 y 100 caracteres, y un link opcional, el resto de los atributos mencionados son obligatorios.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 92,9%.
  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 siempre son distintos de null y 1 línea:
  - **Línea 97:** `dataset.put("mark", mark == null ? null : mark.getMark());` Esta línea están en amarillo debido a que siempre se ha comprobado que mark es distinto de null.

Característica 4: Actualizar CodeAudit ("AuditorCodeAuditUpdateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores actualizar auditorías de código en el sistema. Esta característica debe asegurarse de que solo los auditores puedan actualizar auditorías de código, que cuando se actualiza la auditoría de código, no se ponga un código que tenga otra auditoría de código ya creada, que la fecha de ejecución sea posterior al 1999/12/31 23:59 y anterior al momento actual, que el tipo sea "Static" o "Dynamic", que las acciones correctivas estén entre 1 y 100 caracteres, que la nota sea la media de las notas de todos los registros de auditoría publicados redondeado hacia abajo, y un link opcional, el resto de atributos mencionados son obligatorios.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 92,5%.
  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 siempre son distintos de null y 3 líneas:
  - **Línea 45:** `auditor = codeAudit == null ? null : codeAudit.getAuditor();` Esta línea está en amarillo debido a que siempre se ha comprobado que codeAudit es distinto de null.
  - **Línea 46:** `status = codeAudit != null && !codeAudit.isPublished() && super.getRequest().getPrincipal().hasRole(auditor);` Esta línea está en amarillo debido a que siempre se ha comprobado que las 3 condiciones son true siempre.
  - **Líneas 108:** `dataset.put("mark", mark == null ? null : mark.getMark());` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

Característica 5: Publicar proyecto ("ManagerProjectPublishService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores publicar auditorías de código en el sistema. Esta característica debe asegurarse de que solo los auditores autenticados puedan publicar auditorías de código, que la auditoría de código que se quiere publicar no tenga errores fatales, que no este ya publicada, que cuando se actualiza la auditoría de código, no se ponga un código que tenga otra auditoría de código ya creada, que la fecha de ejecución sea posterior al 1999/12/31 23:59 y anterior al momento actual, que el tipo sea "Static" o "Dynamic", que las acciones correctivas estén entre 1 y 100 caracteres, que la nota sea la media de las notas de todos los registros de auditoría publicados redondeado hacia abajo, y un link opcional, el resto de atributos mencionados son obligatorios.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 93,3%.
  Al ejecutar el "analyser" observamos que tenemos todas las líneas verdes menos todos

los asserts, los cuales están en amarillo y siempre están en amarillo debido a que siempre son distintos de null y 5 líneas:

- **Línea 45:** `auditor = codeAudit == null ? null : codeAudit.getAuditor();` Esta línea está en amarillo debido a que siempre se ha comprobado que codeAudit es distinto de null.
- **Línea 46:** `status = codeAudit != null && !codeAudit.isPublished() && super.getRequest().getPrincipal().hasRole(auditor);` Esta línea está en amarillo debido a que siempre se ha comprobado que las 3 condiciones son true siempre.
- **Línea 74 :** `if (!super.getBuffer().getErrors().hasErrors("executionDate")) {` Esta línea está en amarillo debido a que siempre se ha comprobado que la condición es true siempre
- **Línea 81 :** `if (!super.getBuffer().getErrors().hasErrors("mark")) {` Esta línea está en amarillo debido a que siempre se ha comprobado que la condición es true siempre
- **Línea 87 :** `super.state(all.size() == published.size(), "mark", "auditor.codeAudit.form.error.auditRecordsNotPublished");` Esta línea está en amarillo debido a que siempre se ha comprobado que la condición es true siempre

Característica 6: Borrar CodeAudit ("AuditorCodeAuditDeleteService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores borrar auditorías de código. Esta característica debe asegurarse de que solo los auditores autenticados puedan eliminar las auditorías de código asociadas a ellos.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 68,1%.
  Al ejecutar el "analyser" observamos que hay algunas líneas en amarillo y en rojo.
  - Líneas amarillas:
  - **Líneas 54,61,66:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
  - **Línea 35:** `auditor = codeAudit == null ? null : codeAudit.getAuditor();` Esta línea está en amarillo debido a que siempre se ha comprobado que codeAudit es distinto de null.
  - **Línea 36:** `status = codeAudit != null && !codeAudit.isPublished() && super.getRequest().getPrincipal().hasRole(auditor);` Esta línea está en amarillo debido a que siempre se ha comprobado que las 3 condiciones son true siempre.
  - **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 "AuditRecord":

Característica 1: Listado de AuditRecord ("AuditorAuditRecordListService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores listar los registros de auditoría almacenadas en el sistema. Esta característica debe asegurarse de que solo se muestren los registros de auditoría asociados con la auditoría de código asociada al auditor autenticado.
- **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 y observamos que tenemos todas las líneas verdes menos todos los asserts, los cuales están en amarillo y siempre están en amarillo debido a que siempre son distintos de null y 1 línea:
  - Líneas amarillas:
  - **Línea 34:** `status = codeAudit != null && super.getRequest().getPrincipal().hasRole(codeAudit.getAuditor());` Las 2 condiciones siempre se cumplen por lo que siempre será true.

Característica 2: Detalles de AuditRecord ("AuditorAuditRecordShowService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores ver los detalles de los registros de auditorías almacenadas en el sistema. Esta característica debe asegurarse de que solo se muestren los detalles de los registros de auditoría asociados con la auditoría de código asociada al auditor autenticado.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 96,7%.
  Al ejecutar el "analyser" observamos que tenemos todas las líneas verdes menos todos los asserts, los cuales están en amarillo y siempre están en amarillo debido a que siempre son distintos de null y 1 línea:
  - Líneas amarillas:
  - **Línea 34:** `status = codeAudit != null && super.getRequest().getPrincipal().hasRole(codeAudit.getAuditor());` Las 2 condiciones siempre se cumplen por lo que siempre será true.

Característica 3: Crear AuditRecord ("AuditorAuditRecordCreateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores crear nuevos registros de auditoría en el sistema. Esta característica debe asegurarse de que solo los auditores puedan crear registros de auditoría, que cuando se crea el registro de auditoría, no se

ponga un código que tenga otro registro de auditoría ya creada, el periodo de inicio sea posterior a la fecha de ejecución de la auditoría de código y anterior al momento actual, lo mismo para el final del periodo pero además tiene que ser 1 hora posterior al periodo de inicio, que el nota que sea "A+","A","B","C","F" o "F-" y un link opcional, el resto de atributos mencionados son obligatorios.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 95,8%.
  Al ejecutar el "analyser" observamos que tenemos todas las líneas verdes menos todos los asserts, los cuales están en amarillo y siempre están en amarillo debido a que siempre son distintos de null y 4 líneas:
  - **Línea 43:** `status = codeAudit != null && super.getRequest().getPrincipal().hasRole(codeAudit.getAuditor());` Las 2 condiciones siempre se cumplen por lo que siempre será true.
  - **Línea 80:** `super.state(existing == null, "code", "auditor.auditRecord.form.error.duplicated");` La condición no se cumple nunca por lo que siempre será false;
  - **Línea 96:** `if (!super.getBuffer().getErrors().hasErrors("auditPeriodEnd") && !super.getBuffer().getErrors().hasErrors("startDate")) {` Las condiciones siempre se cumplen por lo que será true.
  - **Línea 96:** `super.state(MomentHelper.isAfterOrEqual(startDate, this.lowestMoment) && MomentHelper.isAfterOrEqual(startDate, codeAudit.getExecutionDate()), "auditPeriodStart", "auditor.auditRecord.form.error.codeAuditDate");` Las condiciones siempre se cumplen por lo que será true.

Característica 4: Actualizar AuditRecord ("AuditorAuditRecordUpdateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores actualizar registros de auditoría en el sistema. Esta característica debe asegurarse de que solo los auditores puedan actualizar registros de auditoría, que cuando se actualiza el registro de auditoría, no se ponga un código que tenga otro registro de auditoría ya creada, el periodo de inicio sea posterior a la fecha de ejecución de la auditoría de código y anterior al momento actual, lo mismo para el final del periodo pero además tiene que ser 1 hora posterior al periodo de inicio, que el nota que sea "A+","A","B","C","F" o "F-" y un link opcional, el resto de atributos mencionados son obligatorios.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 95,7%.
  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 siempre son distintos de null y 3 líneas:
  - **Línea 44:** `status = codeAudit != null &&`

`super.getRequest().getPrincipal().hasRole(codeAudit.getAuditor());` Las 2 condiciones siempre se cumplen por lo que siempre será true.

- **Línea 98:** `if (!super.getBuffer().getErrors().hasErrors("auditPeriodEnd") && !super.getBuffer().getErrors().hasErrors("startDate")) {` Las condiciones siempre se cumplen por lo que será true.
- **Línea 109:** `super.state(MomentHelper.isAfterOrEqual(startDate, this.lowestMoment) && MomentHelper.isAfterOrEqual(startDate, codeAudit.getExecutionDate()), "auditPeriodStart", "auditor.auditRecord.form.error.codeAuditDate");` Las condiciones siempre se cumplen por lo que será true.

Característica 5: Publicar AuditRecord ("AuditorAuditRecordPublishService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores publicar registros de auditoría en el sistema. Esta característica debe asegurarse de que solo los auditores puedan publicar registros de auditoría, que cuando se publique el registro de auditoría, no tenga errores fatales, no este ya publicado, no se ponga un código que tenga otro registro de auditoría ya creada, el periodo de inicio sea posterior a la fecha de ejecución de la auditoría de código y anterior al momento actual, lo mismo para el final del periodo pero además tiene que ser 1 hora posterior al periodo de inicio, que el nota que sea "A+","A","B","C","F" o "F-" y un link opcional, el resto de atributos mencionados son obligatorios.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 95,3%.
  Al ejecutar el "analyser" observamos que tenemos todas las líneas verdes menos todos los asserts, los cuales están en amarillo y siempre están en amarillo debido a que siempre son distintos de null y 5 líneas:
  - **Línea 43:** `status = codeAudit != null && super.getRequest().getPrincipal().hasRole(codeAudit.getAuditor());` Las 2 condiciones siempre se cumplen por lo que siempre será true.
  - **Línea 75 :** `if (!super.getBuffer().getErrors().hasErrors("auditPeriodStart")) {` Esta línea está en amarillo debido a que siempre se ha comprobado que la condición es true siempre.
  - **Línea 82 :** `if (!super.getBuffer().getErrors().hasErrors("auditPeriodEnd")) {` Esta línea está en amarillo debido a que siempre se ha comprobado que la condición es true siempre.
  - **Línea 89 :** `if (!super.getBuffer().getErrors().hasErrors("mark")) {` Esta línea está en amarillo debido a que siempre se ha comprobado que la condición es true siempre

- **Línea 98:** `super.state(MomentHelper.isAfterOrEqual(startDate, this.lowestMoment) && MomentHelper.isAfterOrEqual(startDate, codeAudit.getExecutionDate()), "auditPeriodStart", "auditor.auditRecord.form.error.codeAuditDate");` Las condiciones siempre se cumplen por lo que será true.

Característica 6: Borrar AuditRecord ("AuditorAuditRecordDeleteService.java")

- **Descripción:** Prueba la funcionalidad que permite a los auditores borrar registros de auditoría. Esta característica debe asegurarse de que solo los auditores autenticados puedan eliminar los registros de auditoría asociados a las auditorías de código asociadas a ellos.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 67,1%.
  Al ejecutar el "analyser" observamos que tenemos todas las líneas verdes menos todos los asserts, los cuales están en amarillo y siempre están en amarillo debido a que siempre son distintos de null, 1 línea amarillo y unas cuantas en rojo:
  - Líneas amarillas:
  - **Línea 33:** `status = auditRecord != null && !auditRecord.isPublished() && super.getRequest().getPrincipal().hasRole(codeAudit.getAuditor());` Las 2 condiciones siempre se cumplen por lo que siempre será true.
  - **Línea 36:** `status = codeAudit != null && !codeAudit.isPublished() && super.getRequest().getPrincipal().hasRole(auditor);` Esta línea está en amarillo debido a que siempre se ha comprobado que las 3 condiciones son true siempre.
  - **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

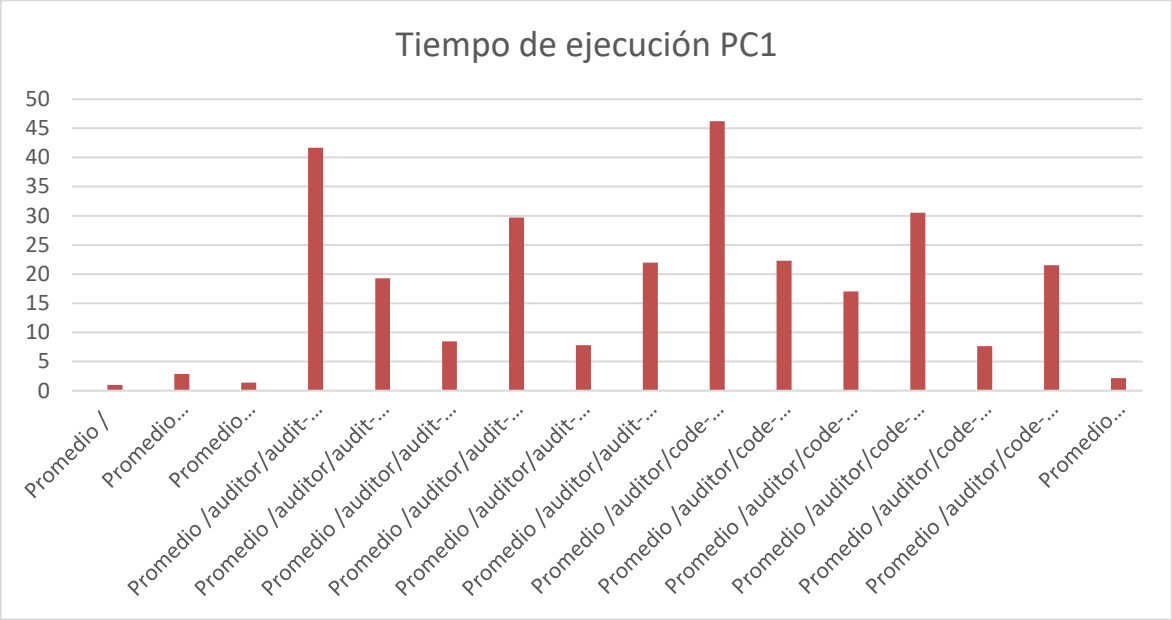
Este capítulo analiza las pruebas de rendimiento realizadas en el proyecto "Acme-SF" respecto a las características de los codeAudits y AuditRecords. El objetivo es evaluar el tiempo de respuesta y la capacidad del sistema bajo diferentes cargas. Utilizamos una metodología que incluye la simulación de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

Resultados de las Pruebas de Rendimiento

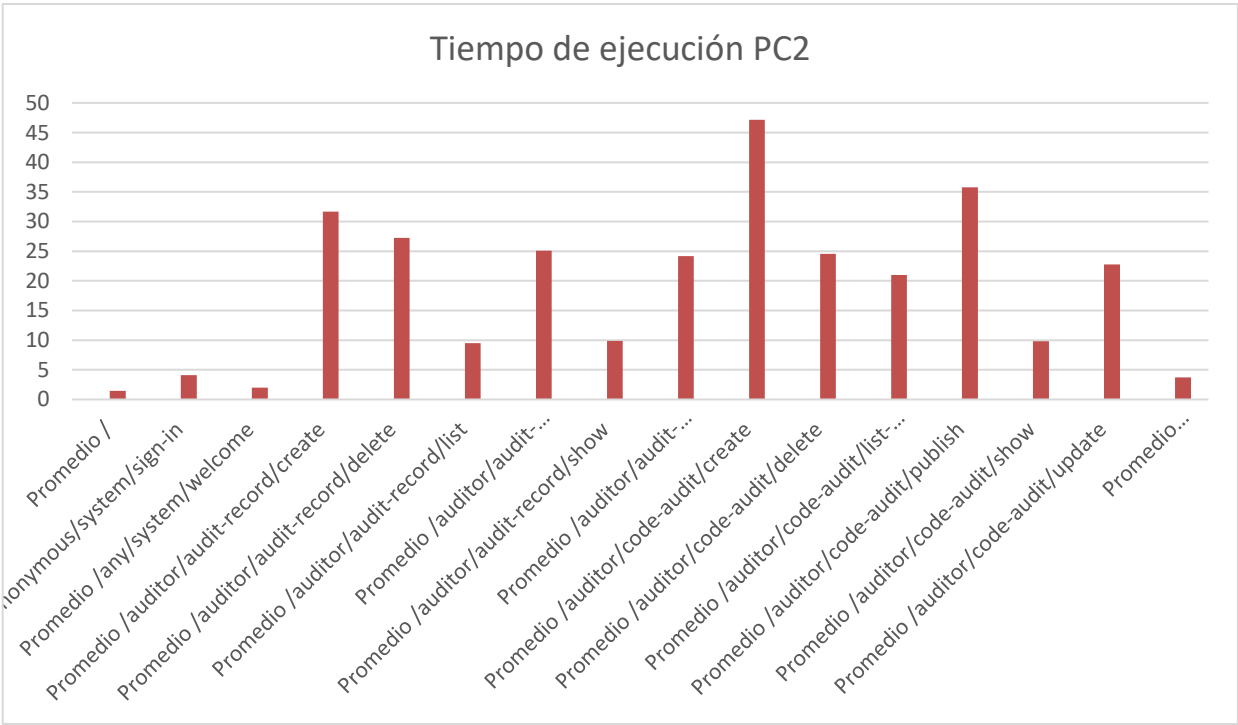
Gráficos

Gráfico del tiempo para la ejecución de las características del gerente sin la optimización en PC1.

- **Gráfico 1:** Gráfico del tiempo para la ejecución de las características del auditor(codeAudits y auditRecords) en PC1



- **Gráfico 2:** Gráfico del tiempo para la ejecución de las características del auditor(codeAudits y auditRecords) en PC2.



Intervalo de Confianza

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

- **PC1:**

Pc1		
Media	17,1065039	
Error típico	0,64330261	
Mediana	9,9539	
Moda	1,2509	
Desviación es	18,4886559	
Varianza de la	341,830397	
Curtosis	5,35748014	
Coeficiente de	1,58264832	
Rango	168,8861	
Mínimo	0,5457	
Máximo	169,4318	
Suma	14129,9722	
Cuenta	826	
Nivel de confi	1,26270243	
Interval(ms)	15,8438015	18,3692063
Interval(s)	0,0158438	0,01836921

PC2:

Pc2		
Media	17,3172042	
Error típico	0,57948171	
Mediana	10,84405	
Moda	1,3373	
Desviación es	16,6544293	
Varianza de la	277,370015	
Curtosis	2,04598927	
Coeficiente de	1,16562729	
Rango	127,6252	
Mínimo	0,8373	
Máximo	128,4625	
Suma	14304,0107	
Cuenta	826	
Nivel de confi	1,13743197	
Interval(ms)	16,1797723	18,4546362
Interval(s)	0,01617977	0,01845464

Contraste de Hipótesis

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

Prueba z para medias de dos muestras		
	Pc1	*Pc2*
Media	17,1065039	17,3172042
Varianza (conocida)	341,830397	277,370015
Observaciones	826	826
Diferencia hipotética de las medias	0	
z	-0,24335464	
P(Z<=z) una cola	0,40386534	
Valor crítico de z (una cola)	1,64485363	
Valor crítico de z (dos colas)	0,80773068	
Valor crítico de z (dos colas)	1,95996398	

- **Resultado del Contraste de Hipótesis:**
  Si observamos el "p-value" para "Before" y "After", su valor es de 0, 80773068, como está contenido en el intervalo (0.05,1.00] entonces los cambios no dieron como resultado ninguna mejora significativa; los tiempos de muestras son diferentes pero son globalmente iguales . De hecho, el tiempo de ejecución medio solo un poco mayor después de ejecutarlo en el 2º pc,1 por lo que vemos que el 1ºº pc es un poco más potente que el 2º pc