
TESTING REPORT

Tabla de revisiones

Nº de revisión	Fecha	Descripción
1	26/05/24	Primera versión del documento

Información general

Fecha: 26/05/24

Grupo: C1.006

Repositorio: <https://github.com/Danielruizlopezcc/Acme-SF-D04>

Miembro:

- Ramón Gavira Sánchez (ramgavsan@alum.us.es)

Tabla de contenidos

- Resumen ejecutivo
- Introducción
- Contenidos
 - Testing funcional
 - * [Contract](#)
 - * [ProgressLog](#)
 - Testing de rendimiento
 - * Análisis de rendimiento
 - * Intervalo de confianza
 - * Hypothesis contrast
 - Conclusiones
 - Bibliografía

Resumen ejecutivo

El proyecto Acme-SF de la asignatura Diseño y Pruebas II, es un proyecto con fines meramente educativos, con el que se busca mejorar las habilidades y trabajar como desarrolladores web. El objetivo es aprender a producir un sistema de información web típico de tamaño pequeño a mediano basándose en una especificación de requisitos informal y métodos y herramientas de potencia industrial.

Introducción

En el contexto del proyecto Acme-SF, se presenta el siguiente documento de testing, que tiene como objetivo presentar el informe de pruebas realizadas a los servicios de las clases Contract y ProgressLog. Se mostrarán todos los resultados obtenidos en el test funcional, donde se incluirán tanto casos de prueba negativos y positivos, así como los resultados obtenidos en la cobertura de código para cada uno de los servicios de estas clases. También se incluirá una sección para mostrar los resultados obtenidos en el test de rendimiento, acompañados de gráficas y tablas que permitirán visualizar de manera más clara todos los resultados.

Contenidos

Testing funcional

Contract

ClientContractListMineService

- **Safe testing** Para el testing legal de este servicio únicamente fue necesario mostrar los contratos de un cliente, no había forma de realizar un test negativo por como estaba implementado el servicio.
- **Hacking**
 - Listado de contratos de un cliente sin estar logueado (Role: Anonymous)
 - Listado de contratos de un cliente con un rol incorrecto (Role: Developer)
- **Bugs:** No se encontraron bugs en este servicio.

ClientContractShowService

- **Safe testing** Para el testing legal de este servicio, como caso de prueba positivo se muestra un contrato de un cliente, y no hay forma de realizar un test negativo por como estaba implementado el servicio.

- **Hacking**

- Mostrar un contrato de un cliente sin estar logueado (Role: Anonymous)
- Mostrar un contrato de un cliente con un rol incorrecto (Role: Developer)
- Intentar mostrar un contrato que no existe
- Mostrar un contrato de un cliente que no es el que posee el contrato (Role: Client)

- **Bugs:** No se encontraron bugs en este servicio.

ClientContractCreateService

- **Safe testing**

- Casos de prueba positivos: Crear sendos contratos con datos correctos
- Casos de prueba negativos: Creación de contratos probando todas las restricciones de los campos:
 - * Formulario vacío
 - * Campos con valores incorrectos, intentando probar todas las restricciones especificadas tanto por las anotaciones como por la lógica del método validate (por ejemplo, códigos duplicados, divisas no permitidas, presupuestos negativos, fechas incorrectas, etc.)

- **Hacking**

- La única forma de hackear este servicio era intentar entrar al formulario de creación de contratos sin estar logueado (Role: Anonymous) o con un rol incorrecto (Role: Developer)

- **Bugs:** No se encontraron bugs en este servicio.

ClientContractUpdateService

- **Safe testing**

- Casos de prueba positivos: Actualizar sendos contratos con datos correctos
- Casos de prueba negativos: Actualización de contratos probando todas las restricciones de los campos:
 - * Formulario vacío
 - * Campos con valores incorrectos, intentando probar todas las restricciones especificadas tanto por las anotaciones como por la lógica del método validate (se siguieron los mismos casos que en el servicio de creación de contratos)

- **Hacking**

- La única forma de hackear este servicio con un rol incorrecto era copiar la URL de actualización de un contrato e intentar ejecutarla, lo que resultaba en una vista de pánico

-
- En cambio, con rol correcto pero ejecutando acciones ilegales se probó lo siguiente:
 - * Actualizar un contrato publicado (acción ilegal)
 - * Actualizar un contrato de otro cliente (acción ilegal)
 - * Actualizar un contrato que no existe
 - **Bugs:** No se encontraron bugs en este servicio.

ClientContractDeleteService

- **Safe testing**
 - Casos de prueba positivos: Eliminar sendos contratos
 - Casos de prueba negativos: No existían acciones negativa que probar que resultasen en errores de lógica de negocio, esta será una de las razones por las que la cobertura de código baja en este servicio, pero se explicará más adelante.
- **Hacking**
 - La única forma de hackear este servicio con un rol incorrecto era copiar la URL de eliminación de un contrato e intentar ejecutarla, lo que resultaba en una vista de pánico
 - En cambio, con rol correcto pero ejecutando acciones ilegales se probó lo siguiente:
 - * Eliminar un contrato publicado (acción ilegal)
 - * Eliminar un contrato de otro cliente (acción ilegal)
 - * Eliminar un contrato que no existe
- **Bugs:** No se encontraron bugs en este servicio.

ClientContractPublishService

- **Safe testing**
 - Casos de prueba positivos: Publicar sendos contratos
 - Casos de prueba negativos: Publicar contratos probando las restricciones de los campos:
 - * Formulario vacío
 - * Campos con valores incorrectos, intentando probar todas las restricciones especificadas tanto por las anotaciones como por la lógica del método validate (se siguieron los mismos casos que en el servicio de creación de contratos y actualización de contratos, pero se añadió una restricción adicional al presupuesto del contrato: todos los contratos no podrán superar el coste total de su proyecto asociado)
- **Hacking**
 - La única forma de hackear este servicio con un rol incorrecto era copiar la URL de publicación de un contrato e intentar ejecutarla, lo que resultaba en una vista de pánico

- En cambio, con rol correcto pero ejecutando acciones ilegales se probó lo siguiente:
 - * Publicar un contrato ya publicado (acción ilegal)
 - * Publicar un contrato de otro cliente, tanto publicado como sin publicar (acción ilegal)
 - * Publicar un contrato que no existe

Cobertura Contract

A continuación se muestra el porcentaje de cobertura de sentencias de todos los servicios de la clase Contract:

▼	acme.features.client.contracts	90,7 %	1.301	133	1.434
>	ClientContractDeleteService.j	63,6 %	133	76	209
>	ClientContractPublishService.j	94,6 %	351	20	371
>	ClientContractCreateService.j	93,9 %	248	16	264
>	ClientContractUpdateService.j	94,6 %	279	16	295
>	ClientContractShowService.ja	97,6 %	165	4	169
>	ClientContractListMineService	98,9 %	89	1	90
>	ClientContractController.java	100,0 %	36	0	36

Figure 1: Contract Coverage

- **ClientContractListMineService:** 98.9%
- **ClientContractShowService:** 97.6%
- **ClientContractCreateService:** 93.9%
- **ClientContractUpdateService:** 94.6%
- **ClientContractDeleteService:** 63.6%
- **ClientContractPublishService:** 94.6%

Siendo la cobertura total del paquete acme.features.client.contract un 90.7%

En líneas generales la cobertura de código de los servicios de la clase Contract es bastante buena, las únicas líneas de código en las que no se llega se deben a una restricción quizás excesiva en la autorización de los métodos en las que se contemplan casos que sin usar herramientas externas, como por ejemplo Postman, no se pueden probar, por eso hay ciertas branches que no se llegan a cubrir, al igual que con líneas encargadas de la internacionalización al español. También una línea de código que encontramos en todos los métodos.

```
1 assert object != null
```

Esta línea de código no se llega a cubrir del todo en ningún método, pero esta presente en todas las plantillas de los servicios proporcionados en la metodología de la asignatura y se consultó con el

cliente y no se consideró necesario cubrirla del todo.

Caso especial: ClientContractDeleteService Todo el método unbind de este servicio no se llega a cubrir, al no poder realizar pruebas negativas que resulten en errores de lógica de negocio, como se explicó anteriormente. Esto se debe a que el método unbind se encargaría de reconstruir la vista en caso de que se produzca un error, pero al no poder producirse, no se llega a ejecutar.

ProgressLog

ClientProgressLogListService

- **Safe testing:**

- Casos de prueba positivos: Listar los registros de un progreso existente en un contrato existente, publicado y siendo el cliente que lo posee
- Casos de prueba negativos: No existían acciones negativa que probar.

- **Hacking**

- Listar los registros de progreso de un contrato sin estar logueado (Role: Anonymous)
- Listar los registros de progreso de un contrato con un rol incorrecto (Role: Developer)
- Listar los registros de progreso de un contrato que no existe.
- Listar los registros de progreso de un contrato que no es del cliente que lo posee (Role: Client)
- Listar los registros de progreso de un contrato que no está publicado.

- **Bugs:** No se encontraron bugs en este servicio.

ClientProgressLogShowService

- **Safe testing:**

- Casos de prueba positivos: Mostrar un registro de progreso existente en un contrato existente, publicado y siendo el cliente que lo posee
- Casos de prueba negativos: No existían acciones negativa que probar.

- **Hacking**

- Mostrar un registro de progreso de un contrato sin estar logueado (Role: Anonymous)
- Mostrar un registro de progreso de un contrato con un rol incorrecto (Role: Developer)

-
- Mostrar un registro de progreso de un contrato que no existe.
 - Mostrar un registro de progreso de un contrato que no es del cliente que lo posee (Role: Client)
 - Mostrar un registro de progreso de un contrato que no está publicado.

- **Bugs:** No se encontraron bugs en este servicio.

ClientProgressLogCreateService

- **Safe testing:**

- Casos de prueba positivos: Crear un registro de progreso con datos correctos
- Casos de prueba negativos: Creación de registros de progreso probando todas las restricciones de los campos:
 - * Formulario vacío
 - * Campos con valores incorrectos, intentando probar todas las restricciones especificadas tanto por las anotaciones como por la lógica del método validate (por ejemplo, completitud del registro incorrecta, fechas anteriores a la fecha del contrato, identificadores duplicados, etc.)

- **Hacking**

- La única forma de hackear este servicio era intentar entrar al formulario de creación de registros de progreso sin estar logueado (Role: Anonymous) o con un rol incorrecto (Role: Developer)

ClientProgressLogUpdateService

- **Safe testing:**

- Casos de prueba positivos: Actualizar un registro de progreso con datos correctos
- Casos de prueba negativos: Actualización de registros de progreso probando todas las restricciones de los campos:
 - * Formulario vacío
 - * Campos con valores incorrectos, intentando probar todas las restricciones especificadas tanto por las anotaciones como por la lógica del método validate (se siguieron los mismos casos que en el servicio de creación de registros de progreso)

- **Hacking**

- La única forma de hackear este servicio con un rol incorrecto era copiar la URL de actualización de un registro de progreso e intentar ejecutarla, lo que resultaba en una vista de pánico

-
- En cambio, con rol correcto pero ejecutando acciones ilegales se probó lo siguiente:
 - * Actualizar un registro de progreso de otro cliente (acción ilegal)
 - * Actualizar un registro de progreso que no existe
 - * Actualizar un registro de progreso publicado
 - **Bugs:** No se encontraron bugs en este servicio.

ClientProgressLogDeleteService

- **Safe testing:** Se eliminaron registros de progreso con datos correctos.
- **Hacking**
 - La única forma de hackear este servicio con un rol incorrecto era copiar la URL de eliminación de un registro de progreso e intentar ejecutarla, lo que resultaba en una vista de pánico
 - En cambio, con rol correcto pero ejecutando acciones ilegales se probó lo siguiente:
 - * Eliminar un registro de progreso de otro cliente (acción ilegal)
 - * Eliminar un registro de progreso que no existe
 - * Eliminar un registro de progreso publicado
- **Bugs:** No se encontraron bugs en este servicio.

ClientProgressLogPublishService

- **Safe testing:**
 - Casos de prueba positivos: Publicar un registro de progreso con datos correctos
 - Casos de prueba negativos: Publicación de registros de progreso probando las restricciones de los campos:
 - * Formulario vacío
 - * Campos con valores incorrectos, intentando probar todas las restricciones especificadas tanto por las anotaciones como por la lógica del método validate (se siguieron los mismos casos que en el servicio de creación de registros de progreso y actualización de registros de progreso)
- **Hacking**
 - La única forma de hackear este servicio con un rol incorrecto era copiar la URL de publicación de un registro de progreso e intentar ejecutarla, lo que resultaba en una vista de pánico
 - En cambio, con rol correcto pero ejecutando acciones ilegales se probó lo siguiente:
 - * Publicar un registro de progreso de otro cliente (acción ilegal)

- * Publicar un registro de progreso que no existe
- * Publicar un registro de progreso publicado

- **Bugs:** No se encontraron bugs en este servicio.

Cobertura PL

A continuación se muestra el porcentaje de cobertura de sentencias de todos los servicios de la clase ProgressLog:

















▼  acme.features.client.progressLogs		93,9 %	1.296	84	1.380
>  ClientProgressLogCreateService.java		94,2 %	259	16	275
>  ClientProgressLogDeleteService.java		92,3 %	191	16	207
>  ClientProgressLogPublishService.java		94,4 %	272	16	288
>  ClientProgressLogUpdateService.java		94,4 %	269	16	285
>  ClientProgressLogShowService.java		91,7 %	121	11	132
>  ClientProgressLogListService.java		94,3 %	149	9	158
>  ClientProgressLogController.java		100,0 %	35	0	35

Figure 2: ProgressLog Coverage

- **ClientProgressLogListService:** 94.3%
- **ClientProgressLogShowService:** 91.7%
- **ClientProgressLogCreateService:** 94.2%
- **ClientProgressLogUpdateService:** 94.4%
- **ClientProgressLogDeleteService:** 92.3%

Siendo la cobertura total del paquete acme.features.client.progressLogs un 93.9%

De nuevo la cobertura sigue siendo buena, nos volvemos a encontrar con las mismas líneas de código que no se llegan a cubrir en todos los métodos, y la misma línea de código que no se llega a cubrir del todo en ningún método. Además estos métodos al tener pocas líneas de código gracias a la metodología de la asignatura y del framework, hace que un pequeño cambio en la cobertura la baje significativamente.

Testing de rendimiento

Análisis de rendimiento

Tras la ejecución de los tests, y con los datos recopilados sobre las peticiones realizadas en los mismos, se llevó a cabo un análisis de los resultados obtenidos en cuanto al rendimiento de la aplicación,

tomando como referencia el tiempo de respuesta de los servicios, centrándonos en los servicios de las clases Contract y ProgressLog. A continuación se muestran los datos del promedio de cada petición, aunque para mayor claridad también se proporciona una gráfica de barras:

request-path	response-status	time
Promedio /		5,908516
Promedio /anonymous/system/sign-in		4,898396296
Promedio /any/system/welcome		3,017604286
Promedio /authenticated/client/update		9,7925
Promedio /authenticated/system/sign-out		3,580211111
Promedio /client/contract/create		45,270726
Promedio /client/contract/delete		37,65185
Promedio /client/contract/list-mine		18,52013478
Promedio /client/contract/publish		48,02863056
Promedio /client/contract/show		15,50993882
Promedio /client/contract/update		45,87644146
Promedio /client/progress-log/create		31,937855
Promedio /client/progress-log/delete		33,3858619
Promedio /client/progress-log/list		12,63792791
Promedio /client/progress-log/publish		38,13783103
Promedio /client/progress-log/show		12,40185789
Promedio /client/progress-log/update		39,79748929
Promedio general		21,71568017

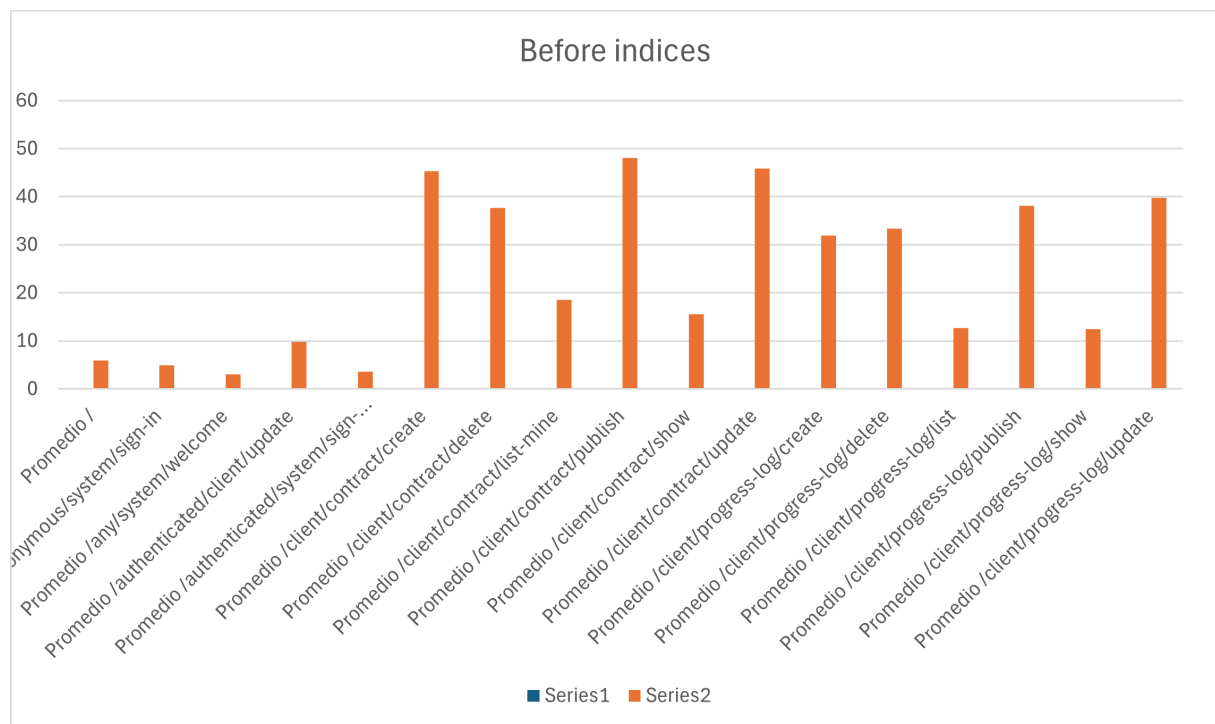


Figure 3: Gráfica de barras

Como vemos en la gráfico, existe grandes diferencia entre los tiempos de respuesta de las peticiones de

inicio de sesión o bienvenido, comparado con las *features* de ambas entidades que estamos analizando. En concreto nos centraremos en los servicios de la clase Contract, que parece tener una de las peticiones más ineficientes y lentas de la aplicación, por ejemplo publicar o actualizar un contrato superan los 40 ms de promedio, 10 veces más que por ejemplo el inicio de sesión. El promedio de todas las respuestas es de 21 ms.

Esta identificación de los servicios más lentos fueron el punto de partida para optimizar la aplicación, y se llevaron a cabo una serie de cambios en el código, en concreto se añadieron índices a las tablas de la base de datos de la entidad Contract, con el objetivo de mejorar el rendimiento de las peticiones. A continuación se muestra una gráfica de barras con los resultados obtenidos tras la optimización y los nuevos promedios de tiempo de respuesta:

request-path	response-status	time
Promedio /		4,657679592
Promedio /anonymous/system/sign-in		4,328559615
Promedio /any/system/welcome		2,638077941
Promedio /authenticated/client/update		7,5367
Promedio /authenticated/system/sign-out		3,0289
Promedio /client/contract/create		26,201002
Promedio /client/contract/delete		23,68526
Promedio /client/contract/list-mine		14,88534891
Promedio /client/contract/publish		30,00122778
Promedio /client/contract/show		11,89312941
Promedio /client/contract/update		25,58937561
Promedio /client/progress-log/create		20,140055
Promedio /client/progress-log/delete		21,35645714
Promedio /client/progress-log/list		9,798788372
Promedio /client/progress-log/publish		23,00974828
Promedio /client/progress-log/show		10,2967125
Promedio /client/progress-log/update		23,59516429
Promedio general		14,52606037



Figure 4: Gráfica de barras

Observamos la notable diferencia en los tiempos de respuesta en prácticamente todas las peticiones, ya que se añadieron ciertos índices también a ProgressLog, pero se bajó el promedio 6 ms, lo que supone una mejora del 28.6% en el rendimiento de la aplicación, y en concreto en los servicios más ineficientes conseguimos bajarlos casi 20 ms, lo que supondría una mejora de casi el 50%.

Intervalo de confianza

Se analizó el intervalo de confianza de los tiempos de respuesta de las peticiones de la aplicación, con el objetivo de coprobar si la aplicación cumple con el requisito del rendimiento establecido: *que el tiempo de respuesta de las peticiones no supere el segundo de media* . Para ello se realizó un análisis de los datos obtenidos en los tests, y se calculó el intervalo de confianza de los tiempos de respuesta de las peticiones, con un nivel de confianza del 95%. La herramienta utilizada para el análisis ha sido excel, que proporciona un complemento llamado *Herramientas para el análisis*. A continuación se muestra el resultado obtenido:

Before			After indices			Before	After
Media	3,188909515		Media	2,059284414		126,6202	84,2874
Error típico	0,08050475		Error típico	0,050961663		15,8898	12,2285
Mediana	1,62135		Mediana	0,8817		36,4329	27,9194
Moda	0,9342		Moda	0,7675		4,5041	4,9792
Desviación estándar	7,72774263		Desviación estándar	4,881425082		4,6233	6,4509
Varianza de la muestra	60,41601974		Varianza de la muestra	23,82831084		6,2996	4,961
Curtosis	66,6332717		Curtosis	85,10059574		12,9461	10,4999
Coefficiente de asimetría	7,175389294		Coefficiente de asimetría	7,181478835		4,4757	3,1133
Rango	150,925		Rango	124,5625		3,3569	2,9598
Mínimo	0,6248		Mínimo	0,5978		42,9543	28,1195
Máximo	151,5498		Máximo	125,1603		30,0159	19,5686
Suma	29727,0145		Suma	18893,9345		36,1275	17,4263
Cuenta	9322		Cuenta	9175		34,0004	21,6617
Nivel de confianza(95,0%)	0,157806902		Nivel de confianza(95,0%)	0,099896204		36,3204	28,7294
						22,9306	17,9142
						45,0303	20,2329
Interval(ms)	3,031102613	3,346716417	Interval(ms)	1,95938821	2,159180618	24,5665	17,5336
Interval(s)	0,003031103	0,003346716	Interval(s)	0,001959388	0,002159181	22,6882	18,8791

Figure 5: Intervalo de confianza

Hypothesis contrast

Como se comentó anteriormente se ha llevado a cabo una refactorización mediante el uso de índices para aumentar el rendimiento de las *queries* de los repositorios asociados a las entidades TrainingModule y TrainingSession. Para comprobar si esta refactorización ha tenido un impacto en el rendimiento de la aplicación, se ha llevado a cabo un contraste de hipótesis, mas concretamente, se ha realizado nuevamente en excel con el complemento *Herramientas para el análisis*, un Z-test (con $\alpha = 0.05$) para evaluar si los nuevos tiempos de respuesta son mejores o peores y en que medida. Para poder realizar el Z-test, se ha debido de recopilar nuevamente los tiempos de respuesta tras la refactorización y obtener los datos estadísticos. A continuación se muestran los datos del análisis post-refactorización, así como los resultados del Z-test.

Prueba z para medias de dos muestras		
	Before	After
Media	21,71568017	14,52606037
Varianza (conocida)	60,41601974	23,82831084
Observaciones	706	694
Diferencia hipotética de las medias	0	
z	20,76244579	
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	

Figure 6: Z-test

Como podemos observar, el valor crítico de z (dos colas) es 0.00, para saber si los cambios han sido significativos, hemos de comparar el valor z con α . En este caso, nuestro z se encuentra entre el intervalo $[0.00, \alpha)$, por lo que podemos afirmar que los cambios realizados han sido significativos y

han mejorado el rendimiento de la aplicación. Igualmente, esto se veía bastante claro en las imágenes de las gráficas de barras o en los promedios, la refactorización ha sido un éxito.

Conclusiones

A lo largo del documento se ha llevado a cabo un análisis de los servicios de las clases Contract y ProgressLog, en el que se han realizado tests funcionales y de rendimiento, con el objetivo de comprobar el correcto funcionamiento de los servicios y el rendimiento de la aplicación. En cuanto a los tests funcionales, se han realizado pruebas tanto positivas como negativas, y se ha comprobado que los servicios funcionan correctamente y no presentan bugs. En cuanto a los tests de rendimiento, se ha llevado a cabo un análisis de los tiempos de respuesta de las peticiones, y se ha comprobado que la aplicación cumple con el requisito de rendimiento establecido. Además, se ha llevado a cabo una refactorización de los servicios de las clases Contract y ProgressLog, añadiendo índices a las tablas de la base de datos, con el objetivo de mejorar el rendimiento de la aplicación. Tras la refactorización, se ha llevado a cabo un contraste de hipótesis, y se ha comprobado que los cambios realizados han sido significativos y han mejorado el rendimiento de la aplicación.

En resumen, considero que he adquirido valiosas lecciones sobre el testing de aplicaciones web usando el Acme-Framework el cual ha facilitado todo este proceso, también aprendí a utilizar la herramienta excel para un análisis más profundo de los datos obtenidos en los tests, cosa que no había hecho antes.

Bibliografía

- 08 Annexes.