

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería

Informática




Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas II

Curso 2023 – 2024

Fecha	Versión
22/10/24	1.0

Grupo de Prácticas: C3.005	
Repositorio: https://github.com/lucantdel/Acme-SF	
Autores por orden alfabético	Correo
Lucas Antoñanzas del Villar	alvbercau@alum.us.es
Mohanad Abulatifa	jaicabher1@alum.us.es
Juan Carlos López Veiga	juacasben@alum.us.es
Álvaro Vázquez Conejo	nicherlob@alum.us.es
Manuel Orta Pérez	ronmonalb@alum.us.es

	Diseño y Pruebas II Testing Report

1. 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.


2. 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 la prueba 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 la prueba de rendimiento, acompañados de gráficas y tablas que permitirán visualizar de manera más clara todos los resultados.

Contenidos

3. Testing funcional

Contract

	<p>Diseño y Pruebas II</p> <p>Analysis Report</p>
---	---

3.2. 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 cómo 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: Manager)

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

3.3. 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 una prueba negativa por cómo 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: Manager&Administrador)
- 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.

3.4. ClientContractCreateService

- **Safe testing**

	<p>Diseño y Pruebas II</p> <p>Testing Report</p>

- 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: Manager&Administrador)

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


3.5. 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)

	<p>Diseño y Pruebas II</p> <p>Analysis Report</p>
---	---

* Actualizar un contrato que no existe

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

3.6. ClientContractDeleteService

- **Safe testing**

- Casos de prueba positivos: Eliminar sendos contratos
- Casos de prueba negativos: No existían acciones negativas 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.

3.7. 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.contract		90,1 %	1.335	147	1.482
>	ClientContractUpdateService.java		93,9 %	292	19	311
>	ClientContractShowService.java		97,6 %	165	4	169
>	ClientContractPublishService.java		92,2 %	357	30	387
>	ClientContractListMineService.java		98,9 %	89	1	90
>	ClientContractDeleteService.java		63,6 %	133	76	209
>	ClientContractCreateService.java		93,9 %	263	17	280
>	ClientContractController.java		100,0 %	36	0	36

Figure 1: Contract Coverage

- **ClientContractListMineService:** 98.9%
- **ClientContractShowService:** 97.6%
- **ClientContractCreateService:** 93.9%
- **ClientContractUpdateService:** 93.9%
- **ClientContractDeleteService:** 63.6%
- **ClientContractPublishService:** 92.2%

	<p>Diseño y Pruebas II</p> <p>Analysis Report</p>
---	---

Siendo la cobertura total del paquete `acme.features.client.contract` un **90.1%**


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.

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

3.8. 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 negativas 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: Manager&Administrador) - 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:

	<p>Diseño y Pruebas II Testing Report</p>

Client)


- Listar los registros de progreso de un contrato que no está publicado.
- **Bugs:** No se encontraron bugs en este servicio.

3.9. 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 negativas 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: Manager&Administrador)
 - 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)
- **Bugs:** No se encontraron bugs en este servicio.

3.10. 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.)

	<p>Diseño y Pruebas II Analysis Report</p>

- **Hacking**

- Crear un registro de progreso de un contrato sin estar logueado (Role: Anonymous)
- Crear un registro de progreso de un contrato con un rol incorrecto (Role: Manager&Administrador)
- Crear un registro de progreso de un contrato que no existe estando logueado (Role: Client).
- Crear un registro de progreso de un contrato que existe estando logueado (Role: Client) pero no publicado.
- Crear un registro de progreso de un contrato que no es del cliente que lo posee (Role: Client)

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


3.11. 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 con un rol incorrecto (Role: Manager&Administrador) y sin estar logueado (Role: Anonymous), lo que resultaba en una vista de pánico
- En cambio, con rol correcto, pero ejecutando acciones ilegales se probó lo siguiente:

	<p>Diseño y Pruebas II</p> <p>Testing Report</p>
---	--

- * 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.

3.12. 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.

3.13. 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		90,6 %	1.247	129
>	ClientProgressLogUpdateService.java		94,1 %	269	17
>	ClientProgressLogShowService.java		96,8 %	121	4
>	ClientProgressLogPublishService.java		93,8 %	271	18
>	ClientProgressLogListService.java		94,3 %	149	9
>	ClientProgressLogDeleteService.java		68,6 %	142	65
>	ClientProgressLogCreateService.java		94,2 %	260	16
>	ClientProgressLogController.java		100,0 %	35	0

Figure 2: ProgressLog Coverage

- **ClientProgressLogListService:** 94.3%
- **ClientProgressLogShowService:** 91.7%
- **ClientProgressLogCreateService:** 94.2%
- **ClientProgressLogUpdateService:** 94.4%
- **ClientProgressLogDeleteService:** 68.6%

Siendo la cobertura total del paquete acme.features.client.progressLogs un 90.6%


	<div>Diseño y Pruebas II</div> <div>Testing Report</div>
---	--

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.

4. 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:

	<p>Diseño y Pruebas II</p> <p>Analysis Report</p>
---	---

	A	B	C	D	
1	request-met	request-path	response-status	Time	
66		Promedio /		74,61	
68		Promedio /administrator/contract/update		10,64	
70		Promedio /administrator/risk/list		101,23	
161		Promedio /anonymous/system/sign-in		58,85	
163		Promedio /any/system/panic		212,63	
271		Promedio /any/system/welcome		20,54	
313		Promedio /authenticated/system/sign-out		55,60	
316		Promedio /client/contract/create		0,00	
328		Promedio /client/contract/delete		254,19	
400		Promedio /client/contract/list-mine		214,48	
411		Promedio /client/contract/publish		210,71	
491		Promedio /client/contract/show		244,53	
531		Promedio /client/contract/update		202,59	
581		Promedio /client/progress-log/create		281,34	
589		Promedio /client/progress-log/delete		41,65	
647		Promedio /client/progress-log/list		182,13	
673		Promedio /client/progress-log/publish		264,87	
714		Promedio /client/progress-log/show		46,91	
745		Promedio /client/progress-log/update		262,02	
746		Promedio general		142,91	
747					
748					

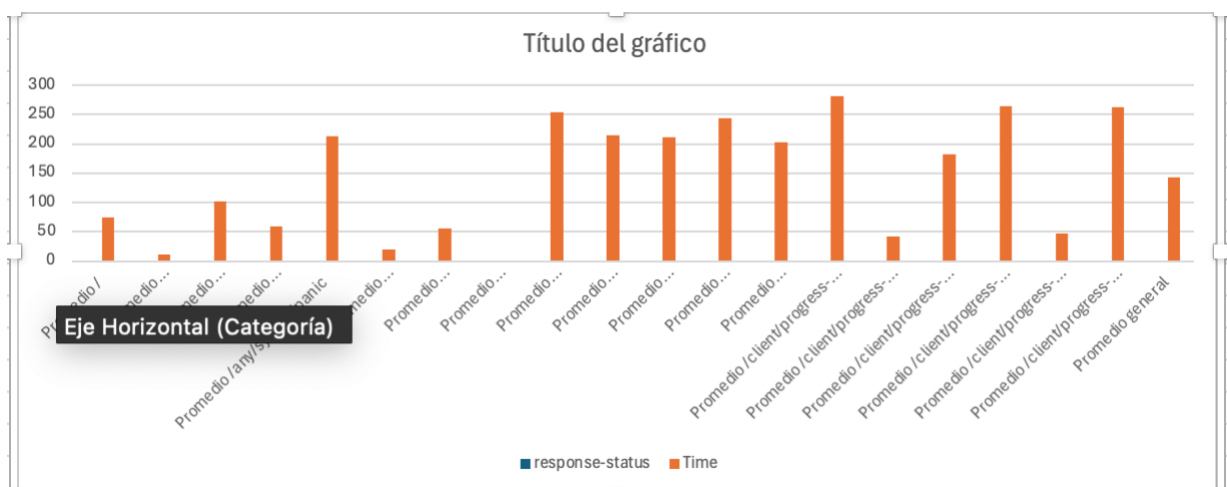



Figure 3: Gráfica de barras

	Diseño y Pruebas II Testing Report

Como vemos en el gráfico, existe grandes diferencias 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

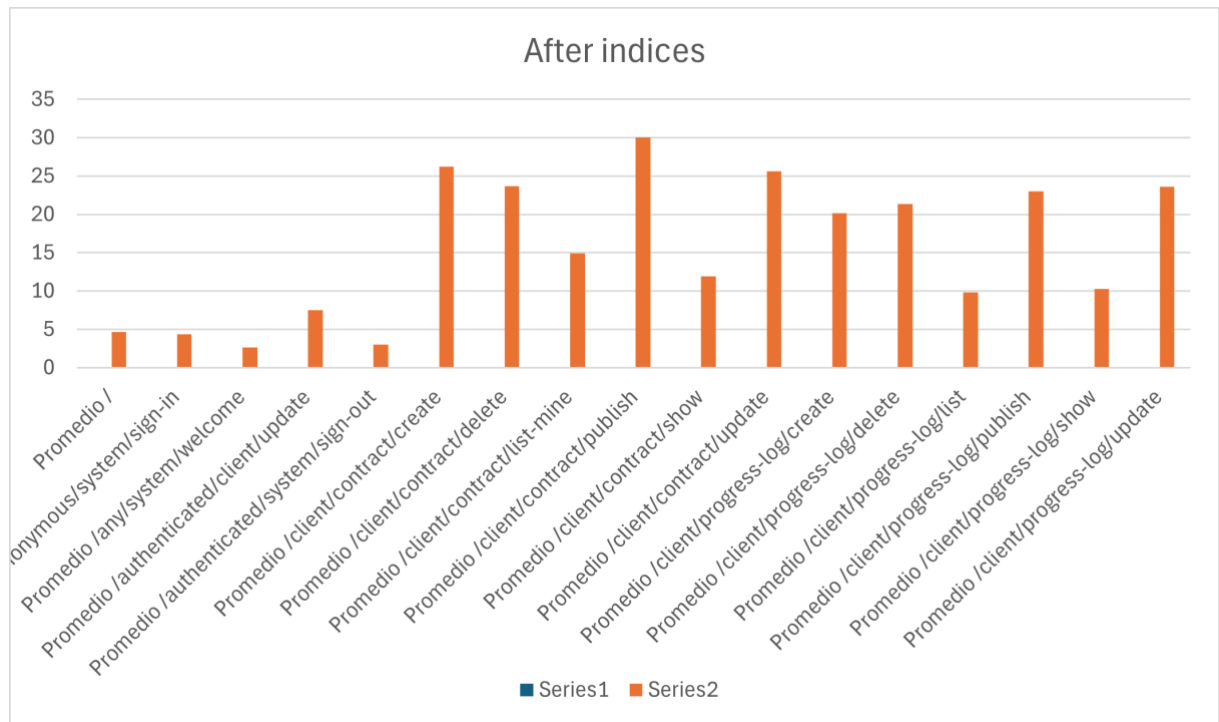



Figure 4: Gráfica de barras

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

7. 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

	Diseño y Pruebas II Testing Report

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.

8. Bibliografía

En Blanco.