DP II - 2022-2023

TESTING REPORT

Repositorio: ***https://github.com/marolmmar1/C2.X02.git***

Miembros:

- CAROLINA CARRASCO DÍAZ
- ÁNGEL LORENZO CASAS
- MARCOS OLMEDO MARÍN

Tutor: RAFAEL CORCHUELO GIL

GRUPO - C2.X02

1 Índice

2 Resumen Ejecutivo

A continuación la finalidad del presente documento es la de recopilar y documentar el análisis de todos los tests realizados para el "Group" los cuáles se han realizado durante el Sprint final (Sprint 4) proporcionando resultados de que las features realizadas cumplen con la expectativas y no presentan ningún bugs que implique a nuestro cliente no estar satisfecho con nuestro trabajo.

Además en el siguiente documento se ha analizado el rendimiento de su ejecución de la aplicación comparándolo con diferentes dispositivos (siendo PC1 y PC2).

3 Tabla de Versionado

Versión	Fecha	Descripción
1.0	04/07/2023	Primera versión del documento
1.1	06/07/2023	Añadidos introducción, resumen ejecutivo y contenido del documento
2.0	07/07/2023	Añadidas las gráficas y análisis comparativo
2.1	08/07/2023	Finalización del documento

4 Introducción

En la siguiente sección se procederá a explicar cómo se han realizado las pruebas funcionales del requisito solicitado así como realizar la comparativa entre el mejor y peor dispositivo con los reportes aportados: ***tester-request-performance.csv*** y ***tester-test-performance.csv***

5 Contenido

5.1 About Testing

A continuación vamos a explicar el funcionamiento y ejecución de las pruebas funcionales que corresponden al requisito grupal #21 (peep).

Pruebas Funcionales > List Peep > Test Positivo

```
@ParameterizedTest
@CsvFileSource(resources = "/any/peep/list-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int peepIndex, final String instantiation, final String title, final String nick) {

    super.clickOnMenu("Anonymous", "Peep List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.checkColumnHasValue(peepIndex, 0, instantiation);
    super.checkColumnHasValue(peepIndex, 1, title);
    super.checkColumnHasValue(peepIndex, 2, nick);

}
```

En este test en primer lugar, lo que hacemos es iniciar sesión con cualquier rol, en este caso con *student*, esto hará que nos redirija al menú de peep, se nos mostrará la lista de forma ascendente de los peeps asociados (hemos comprobado que sí se muestra) y se comprobará que los valores de las columnas coincide con los que le hemos metido como parámetro. Si todo se realiza de forma correcta cerrará sesión y el test estará correcto.

El ejecutar los test de esta manera nos permite detectar que los peeps asociados a ese usuario son correctos y que no aparecen otros pertenecientes a otros usuarios.

Pruebas Funcionales > List Peep >Test Hacking

```
}

@Test
public void test300Hacking() {

}
```

En este test, lo se hace es ir probando que ninguno de los demás roles, exceptuando el actual, puedan acceder a los listado de los peeps y que en cuyo caso de intentar acceder con un rol diferente, devuelva error 500 indicando que no estamos autorizados.

Como puede verse, este tests aparece vacío, al igual que los Test Negativos, esto es debido que para acceder a dicho listado de peeps, puede hacerse con cualquier rol o usuario.

Pruebas Funcionales > Show Peep > Show Positive

```
@ParameterizedTest
@CsvFileSource(resources = "/any/peep/show-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int peepIndex, final String title, final String nick, final String email, final String message, f

    super.signIn("assistant1", "assistant1");

    super.clickOnMenu("Account", "Peep List");
    super.sortListing(0, "asc");
    super.clickOnListingRecord(peepIndex);
    super.checkFormExists();

    super.checkInputBoxHasValue("title", title);
    super.checkInputBoxHasValue("nick", nick);
    super.checkInputBoxHasValue("email", email);
    super.checkInputBoxHasValue("message", message);
    super.checkInputBoxHasValue("link", link);

    super.signOut();
}
```

En este test, al igual que en el List Positivo, lo que hacemos es iniciar sesión con cualquier rol, en este caso con *student*, esto hará que nos redirija al menú de peep, se nos mostrará la lista de forma ascendente de los peeps asociados (hemos comprobado que sí se muestra) y se comprobará que los valores de las columnas coincide con los que le hemos metido como parámetro. Si todo se realiza de forma correcta cerrará sesión y el test estará correcto.

El ejecutar los test de esta manera nos permite detectar que los roles asociados a ese peep son correctos y que no aparecen otros, los cuales pertenecen a otros peep y que los valores que se van guardando en cada campo también son correctos.

Pruebas Funcionales > Peep > Show Peep> Test Hacking

```
@Test
public void test300Hacking() {

}
```

Igual que hemos comentado anteriormente; como puede verse, este tests aparece vacío, al igual que los Test Negativos, y es debido a que para acceder a dicho listado de peeps, puede hacerse con cualquier rol o usuario.

Pruebas Funcionales > Peep > Create Peep> Test Positive

```
@ParameterizedTest
@CsvFileSource(resources = "/any/peep/create-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int peepIndex, final String instantiation, final String title, final String nick, f

    super.signIn("student1", "student1");

    super.clickOnMenu("Account", "Peep List");
    super.checkListingExists();

    super.clickOnButton("Publish");
    super.fillInputBoxIn("title", title);
    super.fillInputBoxIn("nick", nick);
    super.fillInputBoxIn("email", email);
    super.fillInputBoxIn("message", message);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Publish");

    super.signOut();
}
```

En este test, al igual que en el List Positivo, lo que hacemos es iniciar sesión con cualquier rol, en este caso con *student*, esto hará que nos redirija al menú de peep, se nos mostrará la lista de forma ascendente de los peeps asociados (hemos comprobado que sí se muestra) y se comprobará que los valores de las columnas coincide con los que le hemos metido como parámetro. Si todo se realiza de forma correcta cerrará sesión y el test estará correcto.

Una vez que se ha creado correctamente comprobamos que existe la lista de peep asociado a ese rol. Comprobamos que la lista muestra correctamente los datos del peep que hemos creado y que al pulsar sobre él y redirigir al formulario de show se comprueba que los datos de cada campo sean los que hemos añadido.

Por último se comprueba que hay un botón denominado *Publish* y que al pulsar sobre él da lugar a una lista de peeps y que esta está vacía.

El ejecutar los test de esta manera nos permite detectar que los valores metidos en el formulario cumple con las reglas de negocio impuestas y no saltan errores, y por lo tanto nos permite crear el formulario correctamente.

Además se comprueba que los datos metidos en cada campo, al listar y mostrar en el show, son los que hemos metido.

Pruebas Funcionales > Peep > Create Peep> Test Negative

```
@ParameterizedTest
@CsvFileSource(resources = "/any/peep/create-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int peepIndex, final String title, final String nick, final String message, final String email, final String

    super.clickOnMenu("Anonymous", "Peep List");
    super.clickOnButton("Publish");
    super.checkFormExists();

    super.fillInputBoxIn("title", title);
    super.fillInputBoxIn("nick", nick);
    super.fillInputBoxIn("email", email);
    super.fillInputBoxIn("message", message);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Publish");
    super.checkErrorsExist();

}
```

De igual manera, lo que hacemos es iniciar sesión con cualquier rol, en este caso con *Anonymous*, esto hará que nos redirija al menú de peep, pulsamos el botón de create y comprobamos que sale el formulario de creación, se añadirá valores negativos en uno de los campo por cada iteración para así comprobar tras pulsar el botón create que en ese campo muestra un mensaje de error.

Al hacer esto comprobamos que al introducir valores que no cumplen con las reglas de negocio, salta un error y así podemos comprobar que no hay error en la implementación de los mismos.

5.2 Performance Requests Reports

A continuación se han generado los reportes de los **performance request** y **performance testing** con el mejor dispositivo y el peor dispositivo del equipo para tener un estudio aproximado de cómo funciona el software en el mejor y en el peor de los casos; ambos obtuvieron muy buenos tiempos. Es por ello que en este documento hemos reflejado los gráficos de uno de ellos para mostrar estos resultados.

➔ Gráficos

Para la realización de este apartado se ha agrupado en una hoja de excel los tiempos recogidos en los reports performance requests y se han agrupado por su simple-path para calcular el promedio de tiempo invertido en estos mismos.

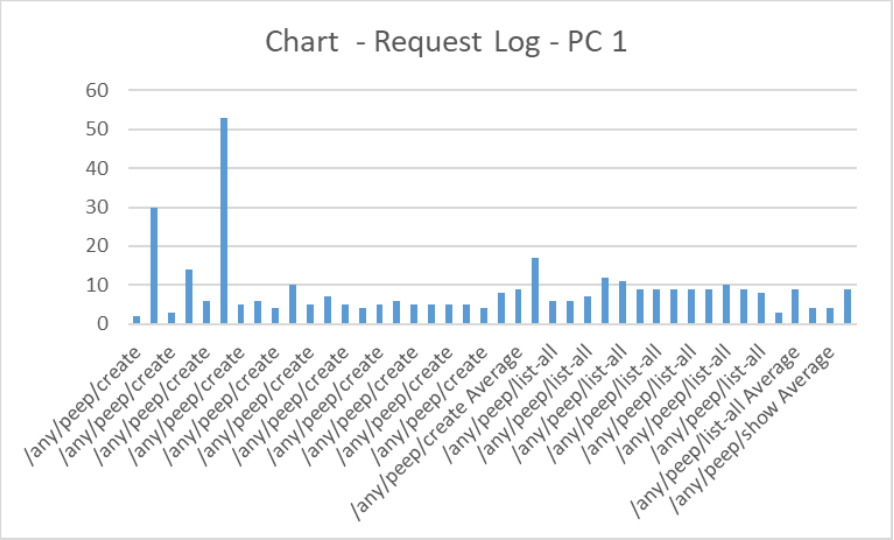

Gráfico del promedio del tiempo de las solicitudes - PC 1

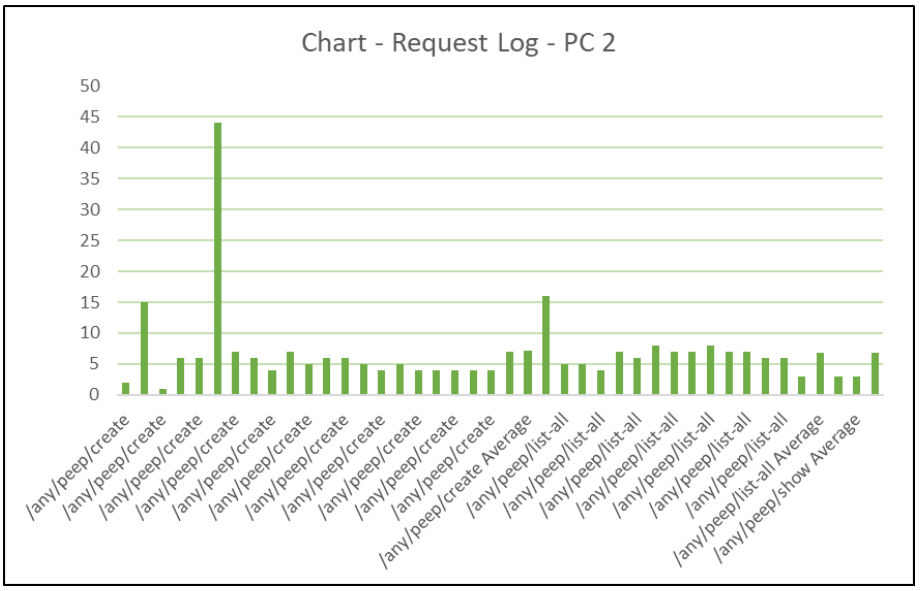

*Gráfico del promedio del tiempo de las solicitudes - **PC 2***

➔ Análisis de datos

Para la realización de este apartado se ha llevado a cabo un análisis estadístico de los tiempos obtenidos en los reportes, haciendo una comparativa de los resultados obtenidos entre dos dispositivos del grupo, para posteriormente analizar los tiempos en cada dispositivo.

Result PC 1	
Mean	8,815789474
Standard Error	1,432132403
Median	6
Mode	5
Standard Deviation	8,828257038
Sample Variance	77,93812233
Kurtosis	17,81302021
Skewness	3,929510764
Range	51
Minimum	2
Maximum	53
Sum	335
Count	38
Confidence Level(95,0%)	2,901775881

Interval (ms):	3,815789474	13,81579
Interval (s):	0,003815789	0,013816

Result PC 2	
Mean	6,779984051
Standard Error	1,000299747
Median	6
Mode	6
Standard Deviation	6,48268328
Sample Variance	42,02518251
Kurtosis	27,68327148
Skewness	4,910400816
Range	43
Minimum	1
Maximum	44
Sum	284,7593301
Count	42
Confidence Level(95,0	2,020146321

Interval (ms):	4,75983773	8,80013
Interval (s):	0,004759838	0,0088

5.3 Performance Tests Reports

A continuación se han generado los reportes de los **performance request** y **performance testing** con el mejor dispositivo y el peor dispositivo del equipo para tener un estudio aproximado de cómo funciona el software en el mejor y en el peor de los casos; ambos obtuvieron muy buenos tiempos. Es por ello que en este documento hemos reflejado los gráficos de uno de ellos para mostrar estos resultados.

➔ Gráficos

Para la realización de este apartado se ha agrupado en una hoja de excel los tiempos recogidos en los reports performance tests y se han agrupado por su test-class y por su test-method después, pudiendo así calcular el promedio de tiempo invertido en cada método de cada clase.

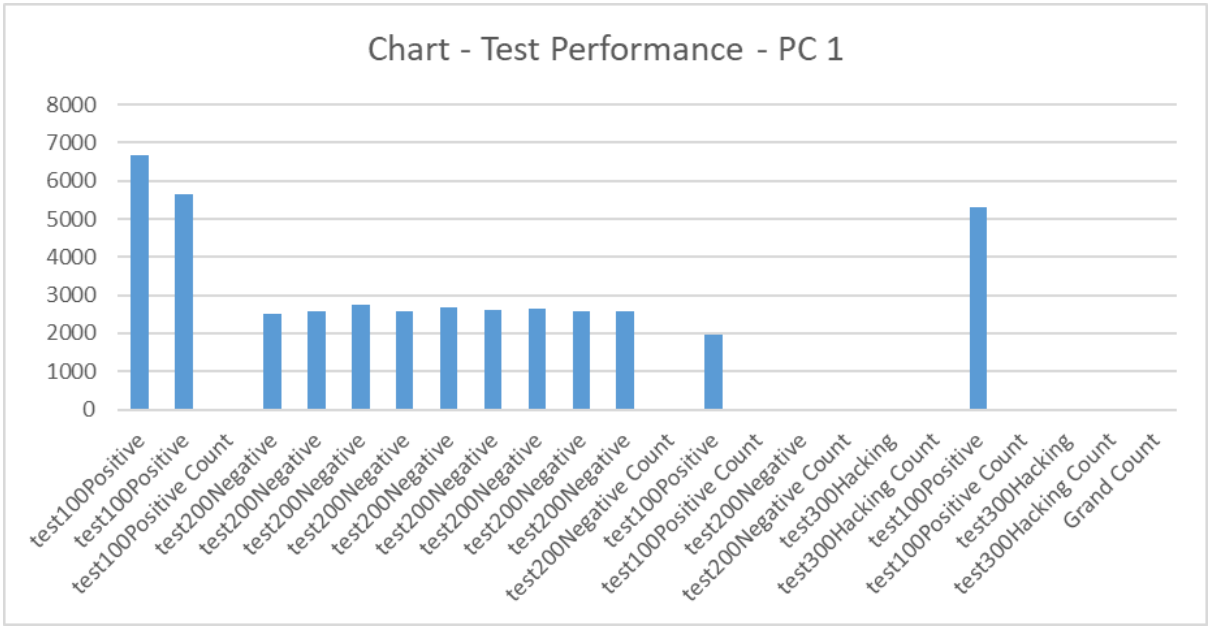

Gráfico del tiempo promedio de ejecución de los tests (de 0 a 70 segundos) - PC1

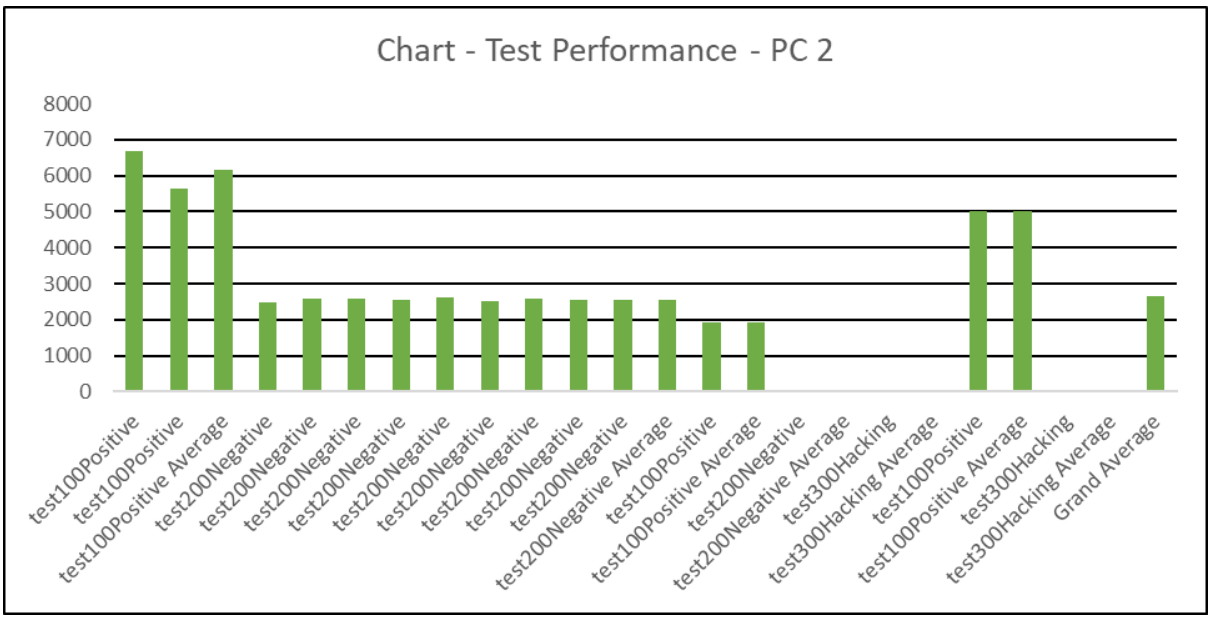

Gráfico del tiempo promedio de ejecución de los tests (de 0 a 70 segundos) - PC1

6 Z-Test

Para la realización de este apartado se ha llevado a cabo un ***<u>análisis Z test</u>*** con los tiempos obtenidos en los reportes de antes y después.

z-Test: Two Sample for Means

	BEFORE	*AFTER*
Mean	6,868421053	6,779984051
Known Variance	46,17140825	42,02518251
Observations	38	42
Hypothesized Mean Differe	0	
z	0,059413446	
P(Z<=z) one-tail	0,476311402	
z Critical one-tail	1,644853627	
P(Z<=z) two-tail	***0,952622804***	
z Critical two-tail	1,959963985	

PC 1: α = 1 - Confidence level = 1- 2,9017= -1,9017
PC 2 : α = 1-Confidence level = 1- 2,02014 = -1,02014

Como observamos el P-Value (P(Z<=z) two tail) que es 0,952622.. se encuentra entre α y 1.00 , es decir entre (α ,1.00] esto quiere decir que los cambios no han supuesto ninguna mejora significativa, es decir, los tiempos de muestreo son diferentes, ***<u>pero globalmente son los mismos.</u>***

<u>Por lo que podemos concluir con que no podemos asegurar cual de los dos PC (PC 1 o PC 2) es más rápido.</u>

7 Lessons Learnt About Testing

A continuación se enumeran las lecciones aprendidas durante este proyecto en grupo acerca del "testeo" llevado a cabo durante este proyecto. Podemos destacar las siguientes lecciones aprendidas como ventajas del testing:

- Permite una organización de una forma más óptima los tests.
- Mejora la gestión del alcance de los test, aportando una mejor optimización en cuanto al tiempo.
- Podemos ver que sin duda el testeo influye muy notable y positivamente respecto a la calidad de código, puesto que podemos detectar errores en una etapa más temprana de desarrollo y de forma más rápida.
- Observando también cómo favorece al trabajo generalizado, pues permite trabajar de manera más ágil.

- Sobre todo cara a los costes de mantenimiento del proyecto, esta es una buena manera de sobrellevarlos.

8 Conclusiones

Intencionalmente en blanco.

9 Bibliografía

Intencionalmente en blanco.