DP II - 2022-2023

TESTING - INDIVIDUAL - REPORT

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

Tutor: RAFAEL CORCHUELO GIL

Alumno: CAROLINA CARRASCO DÍAZ

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 "Student #2" 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	03/07/2023	Primera versión del documento
1.1	05/07/2023	Añadidos introducción, resumen ejecutivo y contenido del documento
2.0	06/07/2023	Añadidas las gráficas y análisis comparativo

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

Pruebas Funcionales > List Enrolment> Test Positivo

```
public class StudentEnrolmentListMineTest extends TestHarness {

    @ParameterizedTest
    @CsvFileSource(resources = "/student/enrolment/list-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
    public void test100Positive(final int enrolmentRecordIndex, final String code, final String workTime) {

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

        super.clickOnMenu("Student", "Enrolment List");
        super.checkListingExists();
        super.sortListing(0, "asc");

        super.checkColumnHasValue(enrolmentRecordIndex, 0, code);
        super.checkColumnHasValue(enrolmentRecordIndex, 1, workTime);

        super.signOut();
    }
```

En este test en primer lugar, lo que hacemos es iniciar sesión como ***student1***, esto hará que nos redirija al menú de *student*, se nos mostrará la lista de forma ascendente de los *enrolments* asociados (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 *enrolments* asociados a ese usuario son correctos y que no aparecen otros pertenecientes a otros usuarios.

Pruebas Funcionales > List Enrolment >Test Hacking

```
@Test
public void test300Hacking() {

    super.checkLinkExists("Sign in");
    super.request("/student/enrolment/list-all");
    super.checkPanicExists();

    super.signIn("administrator", "administrator");
    super.request("/student/enrolment/list-all");
    super.checkPanicExists();
    super.signOut();

    super.signIn("assistant1", "assistant1");
    super.request("/student/enrolment/list-all");
    super.checkPanicExists();
    super.signOut();

    super.signIn("lecturer1", "lecturer1");
    super.request("/student/enrolment/list-all");
    super.checkPanicExists();
    super.signOut();

    super.signIn("company1", "company1");
    super.request("/student/enrolment/list-all");
    super.checkPanicExists();
    super.signOut();

    super.signIn("auditor1", "auditor1");
    super.request("/student/enrolment/list-all");
    super.checkPanicExists();
    super.signOut();
}
```

En este test, lo que hacemos es ir probando que ninguno de los demás roles, exceptuando el nuestro de *student*, puedan acceder a los listado de los *enrolments* y en ese caso, de intentar acceder, se devuelve un *error 500* indicando que no estamos autorizados.

PRUEBAS FUNCIONALES > SHOW ENROLMENT> SHOW POSITIVE

```
@ParameterizedTest
@CsvFileSource(resources = "/student/enrolment/show-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int enrolmentIndex, final String code, final String motivation, final String goals, final String course, final

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

    super.clickOnMenu("Student", "Enrolment List");
    super.sortListing(0, "asc");
    super.clickOnListingRecord(enrolmentIndex);
    super.checkFormExists();

    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("motivation", motivation);
    super.checkInputBoxHasValue("goals", goals);
    super.checkInputBoxHasValue("course", course);
    super.checkInputBoxHasValue("expiryDate", expiryDate);
    super.checkInputBoxHasValue("cvc", cvc);
    super.checkInputBoxHasValue("creditCard", creditCard);
    super.checkInputBoxHasValue("holderName", holderName);

    super.signOut();
}
```

En este test, al igual que en el List Positivo, lo que hacemos es iniciar sesión con *student1*, en este caso con *student*, esto hará que nos redirija al menú de *enrolment*, se nos mostrará la lista de forma ascendente de los enrolment asociado y posteriormente 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.

PRUEBAS FUNCIONALES > ENROLMENT> SHOW ENROLMENT> TEST HACKING

```
@Test
public void test300Hacking() {

    Collection<Enrolment> enrolments;
    String param;

    enrolments = this.repository.findManyEnrolmentsByStudentUsername("student3");
    for (final Enrolment enrolment : enrolments)
        if (enrolment.isDraftMode()) {
            param = String.format("id=%d", enrolment.getId());

            super.checkLinkExists("Sign in");
            super.request("/student/enrolment/show", param);
            super.checkPanicExists();

            super.signIn("administrator", "administrator");
            super.request("/student/enrolment/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("assistant5", "assistant5");
            super.request("/student/enrolment/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("student1", "student1");
            super.request("/student/enrolment/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("lecturer1", "lecturer1");
            super.request("/student/enrolment/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("company1", "company1");
            super.request("/student/enrolment/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("auditor1", "auditor1");
            super.request("/student/enrolment/show", param);
            super.checkPanicExists();
            super.signOut();
}
```

Igual que hemos comentado anteriormente; lo que hacemos es ir probando que ninguno de los demás roles, exceptuando el *student1* asociado a esos enrolments ,puedan acceder a los formularios de cada enrolment, puss de ser así devuelve un error 500 indicando que no estamos autorizados que es lo que estamos buscando.

PRUEBAS FUNCIONALES > STUDENT> CREATE ENROLMENT> TEST POSITIVE

```
@ParameterizedTest
@CsvFileSource(resources = "/student/enrolment/create-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int enrolmentIndex, final String code, final String motivation, final String goals, final String course, fina

    super.signIn("student6", "student6");

    super.clickOnMenu("Student", "Enrolment List");
    super.checkListingExists();

    super.clickOnButton("Create");
    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("motivation", motivation);
    super.fillInputBoxIn("goals", goals);
    super.fillInputBoxIn("course", course);
    super.fillInputBoxIn("expiryDate", expiryDate);
    super.fillInputBoxIn("cvc", cvc);
    super.fillInputBoxIn("creditCard", creditCard);
    super.fillInputBoxIn("holderName", holderName);
    super.clickOnSubmit("Create");

    super.checkListingExists();
    super.sortListing(0, "asc");
    super.checkColumnHasValue(enrolmentIndex, 0, code);

    super.clickOnListingRecord(enrolmentIndex);
    super.checkFormExists();
    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("motivation", motivation);
    super.checkInputBoxHasValue("goals", goals);
    super.checkInputBoxHasValue("course", course);
    super.checkInputBoxHasValue("expiryDate", expiryDate);
    super.checkInputBoxHasValue("cvc", cvc);
    super.checkInputBoxHasValue("creditCard", creditCard);
    super.checkInputBoxHasValue("holderName", holderName);

    super.signOut();
}
```

En este test, al igual que en el List Positivo, lo que hacemos es iniciar sesión con *student6*, esto hará que nos redirija al menú de *student*, se nos mostrará la lista de forma ascendente de los *enrolments* asociados 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 *enrolments* asociado a ese rol. Comprobamos que la lista muestra correctamente los datos del *enrolments* 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 *Create* y que al pulsar sobre él da lugar a una lista de *enrolments.*

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 > STUDENT> CREATE ENROLMENT> TEST NEGATIVE

```
@ParameterizedTest
@CsvFileSource(resources = "/student/enrolment/create-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int enrolmentIndex, final String code, final String motivation, final String goals, final String course, fin

    super.signIn("student6", "student6");

    super.clickOnMenu("Student", "Enrolment List");
    super.clickOnButton("Create");
    super.checkFormExists();

    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("motivation", motivation);
    super.fillInputBoxIn("goals", goals);
    super.fillInputBoxIn("course", course);
    super.fillInputBoxIn("expiryDate", expiryDate);
    super.fillInputBoxIn("cvc", cvc);
    super.fillInputBoxIn("creditCard", creditCard);
    super.fillInputBoxIn("holderName", holderName);
    super.clickOnSubmit("Create");
    super.checkErrorsExist();

    super.signOut();
}
```

De igual manera, lo que hacemos es iniciar sesión con *student 6* esto hará que nos redirija al menú de *enrolments*, 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.

PRUEBAS FUNCIONALES > STUDENT> UPDATE ENROLMENT> TEST POSITIVE

```
@ParameterizedTest
@CsvFileSource(resources = "/student/enrolment/update-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Positive(final int enrolmentIndex, final String code, final String motivation, final String goals, final String course, fir

    super.signIn("student5", "student5");

    super.clickOnMenu("Student", "Enrolment List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(enrolmentIndex);
    super.checkFormExists();
    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("motivation", motivation);
    super.fillInputBoxIn("goals", goals);
    super.fillInputBoxIn("course", course);
    super.fillInputBoxIn("expiryDate", expiryDate);
    super.fillInputBoxIn("cvc", cvc);
    super.fillInputBoxIn("creditCard", creditCard);
    super.fillInputBoxIn("holderName", holderName);
    super.clickOnSubmit("Update");

    super.checkListingExists();
    super.sortListing(0, "asc");

    super.checkColumnHasValue(enrolmentIndex, 0, code);

    super.clickOnListingRecord(enrolmentIndex);
    super.checkFormExists();
    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("motivation", motivation);
    super.checkInputBoxHasValue("goals", goals);
    super.checkInputBoxHasValue("course", course);
    super.checkInputBoxHasValue("expiryDate", expiryDate);
    super.checkInputBoxHasValue("cvc", cvc);
    super.checkInputBoxHasValue("creditCard", creditCard);
    super.checkInputBoxHasValue("holderName", holderName);
```

En este test, lo que hacemos es iniciar sesión con un *student5*, esto hará que nos redirija al menú de *student*, se nos mostrará la lista de *enrolments* de forma ascendente, pulsamos en cada *enrolment* asociados a ese student y nos aparecerá el formulario con todos los datos, que se actualizará con valores positivos en cada campo y tras esto se pulsará el botón de *Update*.

Una vez que se ha creado correctamente comprobamos que existe la lista de *enrolments* asociado a ese *student*, comprobamos que la lista muestra correctamente los datos del *enrolment* que hemos actualizado y que al pulsar sobre él y redirigir al formulario de show se comprueba que los datos de cada campo sean los que hemos añadidos.

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 permite crear el formulario correctamente.

PRUEBAS FUNCIONALES > STUDENT> UPDATE ENROLMENT> TEST NEGATIVE

```
@ParameterizedTest
@CsvFileSource(resources = "/student/enrolment/update-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Negative(final int enrolmentIndex, final String code, final String motivation, final String goals, final String course, f

    super.signIn("student5", "student5");

    super.clickOnMenu("Student", "Enrolment List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(enrolmentIndex);
    super.checkFormExists();

    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("motivation", motivation);
    super.fillInputBoxIn("goals", goals);
    super.fillInputBoxIn("course", course);
    super.fillInputBoxIn("expiryDate", expiryDate);
    super.fillInputBoxIn("cvc", cvc);
    super.fillInputBoxIn("creditCard", creditCard);
    super.fillInputBoxIn("holderName", holderName);

    super.clickOnSubmit("Update");

    super.checkErrorsExist();

    super.signOut();
}
```

De igual manera que se ha mencionado anteriormente, se inicia sesión como *student5*. En este caso, se actualizará el listado de *enrolments* con valores negativos los cuales no cumplen las reglas de negocio en cada campo y tras esto se pulsará el botón de Update.

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

….. (de igual manera con activity…)…..

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.

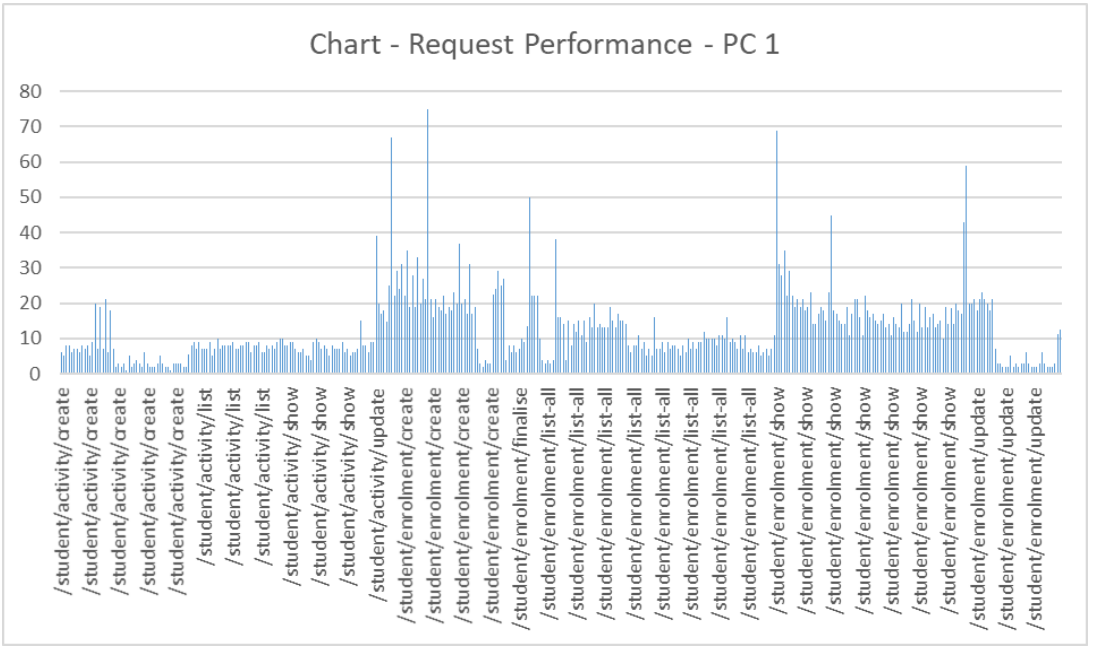

Gráfico del promedio del tiempo de las solicitudes - PC 1

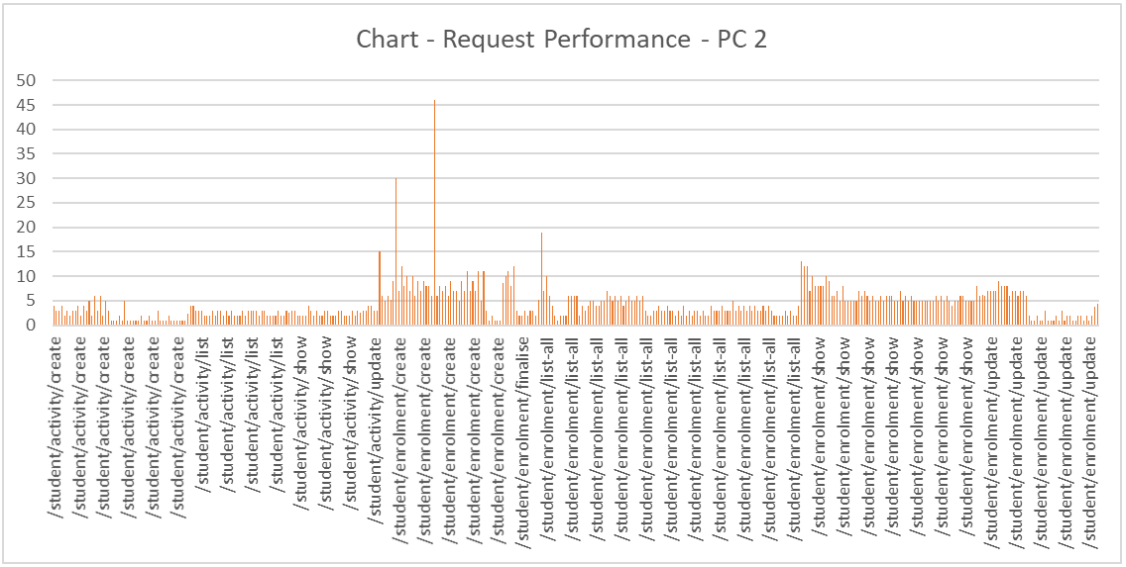

*Gráfico del promedio del tiempo de las solicitudes - **PC 2***

➔ 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	12,41074
Standard Error	0,498375
Median	9
Mode	7
Standard Deviation	9,778814
Sample Variance	95,6252
Kurtosis	10,58609
Skewness	2,520735
Range	74
Minimum	1
Maximum	75
Sum	4778,133
Count	385
Confidence Level(95,0%)	0,979885

Interval (ms):	11,43085	13,39062
Interval (s):	0,011431	0,013391

Result - PC 2	
Mean	4,411887539
Standard Error	0,187300484
Median	3
Mode	3
Standard Deviation	3,66075429
Sample Variance	13,40112197
Kurtosis	49,32647438
Skewness	5,205847656
Range	45
Minimum	1
Maximum	46
Sum	1685,34104
Count	382
Confidence Level(95,(	0,368272067

Interval (ms):	4,043615473	4,780159606
Interval (s):	0,004043615	0,00478016

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.

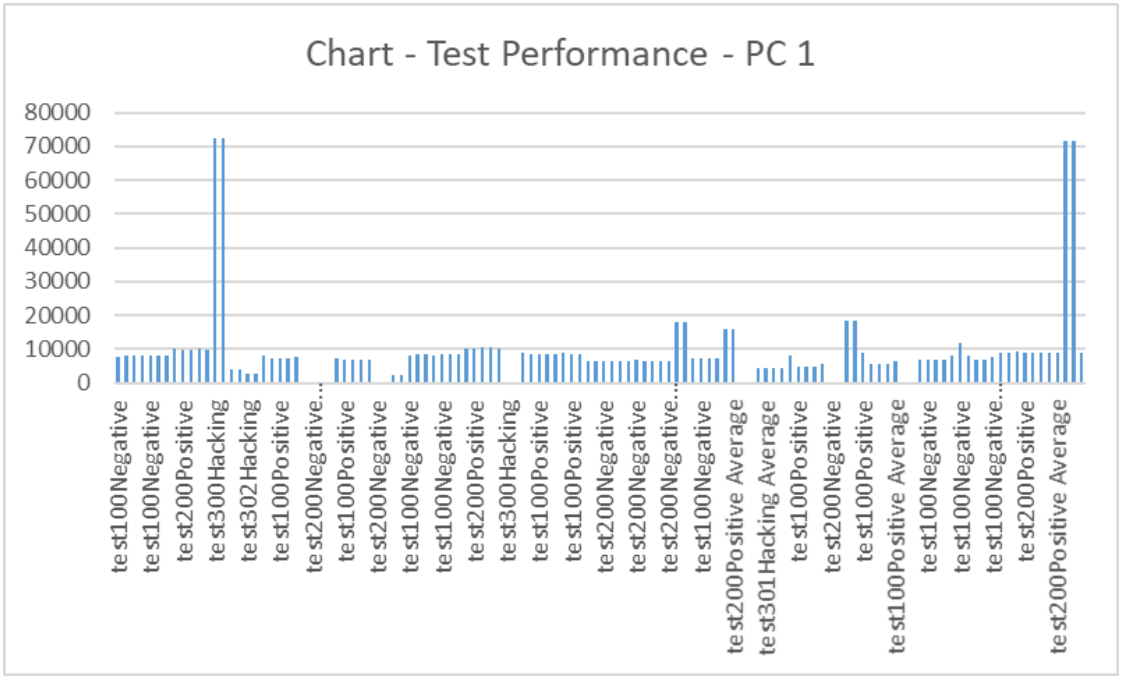

Gráfico del tiempo promedio de ejecución de los tests (de 0 a algo más de 70 segundos) - PC1

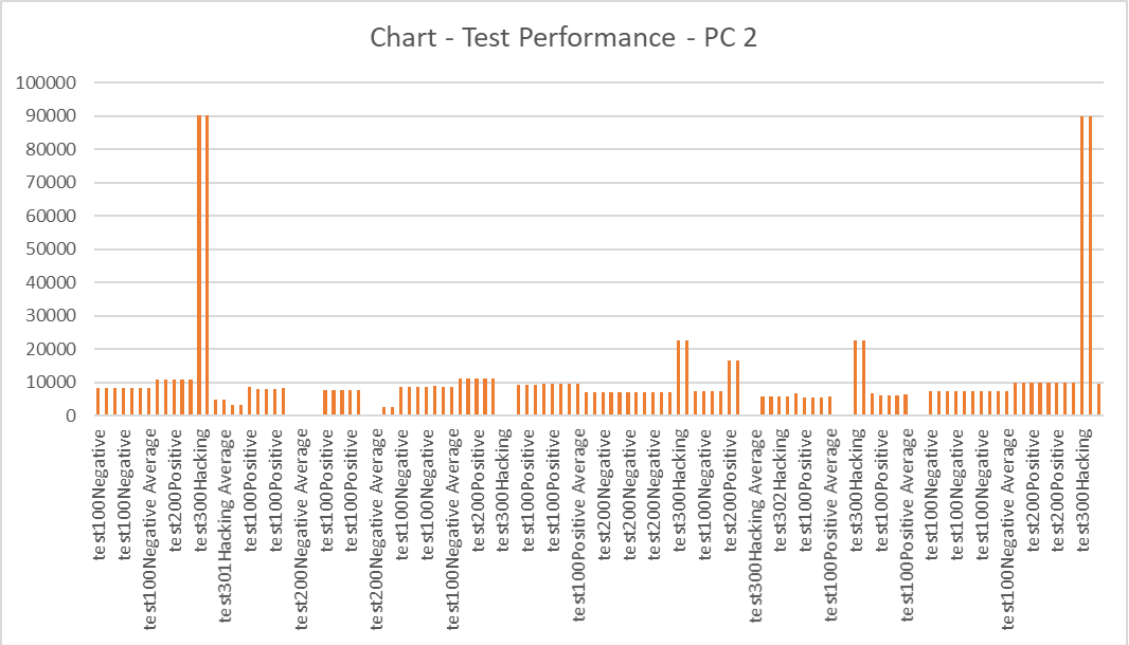

Gráfico del tiempo promedio de ejecución de los tests (de 0 a 90 segundos) PC1

6 Z-Test

➔ **Z-Test (Before-After) con PC 1:**

z-Test: Two Sample for Means		
	BEFORE	*AFTER*
Mean	12,41066667	12,41073535
Known Variance	97,52608913	95,62520367
Observations	375	385
Hypothesized Mea	0	
z	-9,63236E-05	
P(Z<=z) one-tail	0,499961572	
z Critical one-tail	1,644853627	
P(Z<=z) two-tail	0,999923145	
z Critical two-tail	1,959963985	

➔ **Z-Test (Before-After) con PC 2:**

z-Test: Two Sample for Means		
	BEFORE	*AFTER*
Mean	4,411290323	4,411887539
Known Variance	13,67135466	13,40112197
Observations	372	382
Hypothesized Mean Diff	0	
z	-0,00222829	
P(Z<=z) one-tail	0,499111042	
z Critical one-tail	1,644853627	
P(Z<=z) two-tail	0,998222084	
z Critical two-tail	1,959963985	

[α = 1- confidence level]

PC 1: α = 1 - 0,979885 = 0,020115

PC 2 : α = 1- 0,368272067 = 0,631727933

0,00 *α = 0,020115* *α = 0,631727933* *P(Z<=z) two tail = 0,99* *1,00*

Como observamos el P-Value (P(Z<=z) two tail) se encuentra entre α y 1.00 , es decir ***(α ,1.00]*** *esto quiere decir que los cambios no han supuesto ninguna mejora significativa, es decir, los tiempos de muestreo son diferentes, pero globalmente son los mismos. Por lo que podemos concluir con que no podemos asegurar cual de los dos PC (PC 1 o PC 2) es más rápido.*

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.