

DP II - 2022-2023

TEST REPORT



<https://github.com/pedlopruz/Acme-L3-D04.git>

Miembros:

- Carolina Carrasco Díaz
- Pedro López Ruz
- Ángel Lorenzo Casas
- Manuel Navarro Sicre
- Manuel Ortiz Blanco

Tutor: RAFAEL CORCHUELO GIL

GRUPO - C1.04.12

1 ÍNDICE

Índice	2
Resumen Ejecutivo	2
Tabla de Versionado	2
Introducción	2
Contenido	3
Test Request Performance	30
Test Performance Report	32
Z-Test	33
Conclusión	33
Bibliografía	33

2 RESUMEN EJECUTIVO

A continuación la finalidad del presente documento será la de tener documentado por escrito como se ha realizado la tarea de testing, explicando las conclusiones obtenidas y analizando el rendimiento al aplicar los mismos.

3 TABLA DE VERSIÓNADO

Versión	Fecha	Descripción
1.0	20/03/2023	Primera versión del documento
1.1	24/05/2023	Rellenar el documento

4 INTRODUCCIÓN

La intención de este documento es la de recopilar y documentar el análisis de todos los tests realizados para el estudiante 1, los cuales se han realizado en el Sprint 4 proporcionando resultados de que las features realizadas cumplen con la expectativas y no tienen bugs que provoquen que nuestros clientes no estén contentos con nuestro trabajo. Además analizaremos cómo ha sido el rendimiento de su ejecución de la aplicación en varios PCs.

5 CONTENIDO

Pruebas Funcionales

Listar Course

Test Positivo

```
@ParameterizedTest
@CsvFileSource(resources = "/lecturer/course/list-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int courseRecordIndex, final String code, final String title, final String abstracts) {
    super.signIn("lecturer3", "lecturer3");

    super.clickOnMenu("Lecturer", "List of courses");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.checkColumnHasValue(courseRecordIndex, 0, code);
    super.checkColumnHasValue(courseRecordIndex, 1, title);
    super.checkColumnHasValue(courseRecordIndex, 2, abstracts);

    super.signOut();
}
```

En este test en primer lugar, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de forma ascendente de courses asociados a ese lecturer (comprobando que 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 courses asociados a ese lecturer son correctos y que no aparecen otros los cuales pertenecen a otros lecturer.

Test Hacking

```
@Test
public void test300Hacking() {

    super.checkLinkExists("Sign in");
    super.request("/lecturer/course/list-all");
    super.checkPanicExists();

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

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

    super.signIn("student1", "student1");
    super.request("/lecturer/course/list-all");
    super.checkPanicExists();
    super.signOut();

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

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

En este test, lo que hacemos es ir probando que ninguno de los demás roles, exceptuando lecturer, no pueden acceder a los listado de los courses y que devuelve un error 500 indicando que no estamos autorizados.

Show Course

Test Positive

```
@ParameterizedTest
@CsvFileSource(resources = "/lecturer/course/show-positive.csv", encoding =
public void test100Positive(final int courseIndex, final String code, final

    super.signIn("lecturer3", "lecturer3");

    super.clickOnMenu("Lecturer", "List of courses");
    super.sortListing(0, "asc");
    super.clickOnListingRecord(courseIndex);
    super.checkFormExists();

    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("title", title);
    super.checkInputBoxHasValue("price", price);
    super.checkInputBoxHasValue("abstracts", abstracts);
    super.checkInputBoxHasValue("link", link);
    super.checkInputBoxHasValue("nature", nature);

    super.signOut();
}
```

En este test en primer lugar, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de courses de forma ascendente pulsamos en cada courses asociados a ese lecturer (comprobando que se muestra) y se comprobará que los valores de cada campo 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 courses asociados a ese lecturer son correctos y que no aparecen otros, los cuales pertenecen a otros lecturer y que los valores que se van guardando en cada campo son correctos.

Test Hacking

```
@Test
public void test300Hacking() {
    Collection<Course> courses;
    String param;
    courses = this.repository.findManyCoursesByLecturerUsername("lecturer3");
    for (final Course course : courses)
        if (course.isDraftMode())
            param = String.format("id=%d", course.getId());

    super.checkLinkExists("Sign in");
    super.request("/lecturer/course/show", param);
    super.checkPanicExists();

    super.signIn("administrator", "administrator");
    super.request("/lecturer/course/show", param);
    super.checkPanicExists();
    super.signOut();

    super.signIn("assistant5", "assistant5");
    super.request("/lecturer/course/show", param);
    super.checkPanicExists();
    super.signOut();

    super.signIn("lecturer1", "lecturer1");
    super.request("/lecturer/course/show", param);
    super.checkPanicExists();
    super.signOut();

    super.signIn("student1", "student1");
    super.request("/lecturer/course/show", param);
    super.checkPanicExists();|  
    super.signOut();

    super.signIn("company1", "company1");
    super.request("/lecturer/course/show", param);
    super.checkPanicExists();
    super.signOut();

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

En este test, lo que hacemos es ir probando que ninguno de los demás roles, exceptuando el lecturer asociado a esos courses , no pueden acceder a los formularios de cada course y que devuelve un error 500 indicando que no estamos autorizados que es lo que estamos buscando.

Create Course

Test Positive

```

@ParameterizedTest
@CsvFileSource(resources = "/lecturer/course/create-positive.csv", encoding = "utf-8", numLinesToSkip =
public void test100Positive(final int courseIndex, final String code, final String title, final String
super.signIn("lecturer2", "lecturer2");

super.clickOnMenu("Lecturer", "List of courses");
super.checkListingExists();

super.clickOnButton("Create course");
super.fillInputBoxIn("code", code);
super.fillInputBoxIn("title", title);
super.fillInputBoxIn("price", price);
super.fillInputBoxIn("abstracts", abstracts);
super.fillInputBoxIn("link", link);
super.clickOnSubmit("Create course");

super.clickOnMenu("Lecturer", "List of courses");
super.checkListingExists();
super.sortListing(0, "asc");
super.checkColumnHasValue(courseIndex, 0, code);
super.checkColumnHasValue(courseIndex, 1, title);
super.checkColumnHasValue(courseIndex, 2, abstracts);

super.clickOnListingRecord(courseIndex);
super.checkFormExists();
super.checkInputBoxHasValue("code", code);
super.checkInputBoxHasValue("title", title);
super.checkInputBoxHasValue("price", price);
super.checkInputBoxHasValue("abstracts", abstracts);
super.checkInputBoxHasValue("link", link);
super.checkInputBoxHasValue("nature", nature);

super.clickOnButton("Lectures");
super.checkListingExists();
super.checkListingEmpty();

super.signOut();
}
}

```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de courses de forma ascendente (comprobando que se muestra), tras esto al pulsar el botón de create saldrá el formulario de creación, se añadirá valores positivos en cada campo y tras esto se pulsará el botón de create.

Una vez que se ha creado correctamente, comprobamos que existe la lista de course asociado a ese lecturer, comprobamos que la lista muestra correctamente los datos del course 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 añadidos.

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

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

Además se comprueba que los datos metidos en cada campo luego al listar y mostrar en el show son los que hemos metido, y por último se genera un botón que deriva a los lectures de ese course.

Test Negative

```
@ParameterizedTest
@CsvFileSource(resources = "/lecturer/course/create-negative.csv",
public void test200Negative(final int courseIndex, final String coc

    super.signIn("lecturer2", "lecturer2");

    super.clickOnMenu("Lecturer", "List of courses");
    super.clickOnButton("Create course");
    super.checkFormExists();

    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("title", title);
    super.fillInputBoxIn("price", price);
    super.fillInputBoxIn("abstracts", abstracts);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Create course");
    super.checkErrorsExist();

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, 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.

Test Hacking

```
@Test  
public void test300Hacking() {  
  
    super.checkLinkExists("Sign in");  
    super.request("/lecturer/course/create");  
    super.checkPanicExists();  
  
    super.signIn("administrator", "administrator");  
    super.request("/lecturer/course/create");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("student1", "student1");  
    super.request("/lecturer/course/create");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("company1", "company1");  
    super.request("/lecturer/course/create");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("assistant1", "assistant1");  
    super.request("/lecturer/course/create");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("auditor1", "auditor1");  
    super.request("/lecturer/course/create");  
    super.checkPanicExists();  
    super.signOut();  
}
```

En este test, lo que hacemos es ir probando que ninguno de los demás roles, exceptuando lecturer, no pueden crear course y que devuelve un error 500 indicando que no estamos autorizados que es lo que estamos buscando.

Update Course

Test Positive

```

@ParameterizedTest
@CsvFileSource(resources = "/lecturer/course/update-positive.csv", encoding =
public void test100Positive(final int courseIndex, final String code, final !
super.signIn("lecturer1", "lecturer1");

super.clickOnMenu("Lecturer", "List of courses");
super.checkListingExists();
super.sortListing(0, "asc");

super.clickOnListingRecord(courseIndex);
super.checkFormExists();
super.fillInputBoxIn("code", code);
super.fillInputBoxIn("title", title);
super.fillInputBoxIn("price", price);
super.fillInputBoxIn("abstracts", abstracts);
super.fillInputBoxIn("link", link);
super.clickOnSubmit("Update");

super.checkListingExists();
super.sortListing(0, "asc");
super.checkColumnHasValue(courseIndex, 0, code);
super.checkColumnHasValue(courseIndex, 1, title);
super.checkColumnHasValue(courseIndex, 2, abstracts);

super.clickOnListingRecord(courseIndex);
super.checkFormExists();
super.checkInputBoxHasValue("code", code);
super.checkInputBoxHasValue("title", title);
super.checkInputBoxHasValue("price", price);
super.checkInputBoxHasValue("abstracts", abstracts);
super.checkInputBoxHasValue("link", link);
super.checkInputBoxHasValue("nature", nature);

super.signOut();
}

super.checkInputBoxHasValue("course", course);
super.checkInputBoxHasValue("estimatedTime", estimatedTime);

super.signOut();
}

```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de courses de forma ascendente (comprobando que se muestra), pulsamos en cada course asociados a ese lecturer. Tras esto aparecerá el formulario con todos los datos (se comprueba que existe), 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 course asociado a ese lecturer, comprobamos que la lista muestra correctamente los datos del course 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 cumplen con las reglas de negocio impuestas y no saltan errores y permite crear el formulario correctamente.

Test Negative

```

@ParameterizedTest
@CsvFileSource(resources = "/lecturer/course/update-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int courseIndex, final String code, final String title, final String price) {
    super.signIn("lecturer1", "lecturer1");

    super.clickOnMenu("Lecturer", "List of courses");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(courseIndex);
    super.checkFormExists();
    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("title", title);
    super.fillInputBoxIn("price", price);
    super.fillInputBoxIn("abstracts", abstracts);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Update");

    super.checkErrorsExist();
}

super.signOut();
}

```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de courses de forma ascendente (comprobando que se muestra), pulsamos en cada course asociados a ese lecturer. Tras esto aparecerá el formulario con todos los datos (se comprueba que existe), se actualizará 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 cumplen con las reglas de negocio impuestas, saltan excepciones y no permite crear el formulario correctamente.

Test Hacking

```
@Test
public void test300Hacking() {
    Collection<Course> courses;
    String param;
    courses = this.repository.findManyCoursesByLecturerUsername("lecturer1");
    for (final Course course : courses) {
        param = String.format("id=%d", course.getId());

        super.checkLinkExists("Sign in");
        super.request("/lecturer/course/update", param);
        super.checkPanicExists();

        super.signIn("administrator", "administrator");
        super.request("/lecturer/course/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("lecturer2", "lecturer2");
        super.request("/lecturer/course/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("student2", "student2");
        super.request("/lecturer/course/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("assistant2", "assistant2");
        super.request("/lecturer/course/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("company2", "company2");
        super.request("/lecturer/course/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("auditor2", "auditor2");
        super.request("/lecturer/course/update", param);
        super.checkPanicExists();
        super.signOut();
    }
}
```

En este test, lo que hacemos es ir probando que ninguno de los demás roles, exceptuando el lecturer que es el nuestro, puedan actualizar un course y que devuelva un error 500 indicando que no estamos autorizados, que es lo que estamos buscando.

Publish Course

Test Positive

```
@ParameterizedTest
@CsvFileSource(resources = "/lecturer/course/publish-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int courseIndex, final String code) {

    super.signIn("lecturer9", "lecturer9");

    super.clickOnMenu("Lecturer", "List of courses");
    super.checkListingExists();
    super.sortListing(0, "asc");
    super.checkColumnHasValue(courseIndex, 0, code);

    super.clickOnListingRecord(courseIndex);
    super.checkFormExists();
    super.clickOnSubmit("Publish");
    super.checkNotErrorsExist();

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de courses de forma ascendente (comprobando que se muestra), pulsamos en cada course asociados a ese lecturer. Tras esto aparecerá el formulario con todos los datos (se comprueba que existe), tras esto se pulsará el botón de publish y observaremos que no produce errores.

El ejecutar los test de esta manera nos permite detectar que se publica course los cuales tienen un lecture asociado.

Test Negative

```
@ParameterizedTest
@CsvFileSource(resources = "/lecturer/course/publish-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int courseIndex, final String code) {

    super.signIn("lecturer4", "lecturer4");

    super.clickOnMenu("Lecturer", "List of courses");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.checkColumnHasValue(courseIndex, 0, code);
    super.clickOnListingRecord(courseIndex);
    super.checkFormExists();
    super.clickOnSubmit("Publish");
    super.checkErrorsExist();

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de courses de forma ascendente (comprobando que se muestra), pulsamos en cada course asociados a ese lecturer. Tras esto aparecerá el formulario con todos los

datos (se comprueba que existe), tras esto se pulsará el botón de publish y observaremos que produce error.

El ejecutar los test de esta manera nos permite detectar que no se publica courses los cuales tienen un lecture asociado.

Test Hacking

```

@Test
public void test300Hacking() {

    Collection<Course> courses;
    String params;

    courses = this.repository.findManyCoursesByLecturerUsername("lecturer2");
    for (final Course course : courses)
        if (course.isDraftMode())
            params = String.format("id=%d", course.getId());

    super.checkLinkExists("Sign in");
    super.request("/lecturer/course/publish", params);
    super.checkPanicExists();

    super.signIn("administrator", "administrator");
    super.request("/lecturer/course/publish", params);
    super.checkPanicExists();
    super.signOut();

    super.signIn("assistant1", "assistant1");
    super.request("/lecturer/course/publish", params);
    super.checkPanicExists();
    super.signOut();

    super.signIn("student1", "student1");
    super.request("/lecturer/course/publish", params);
    super.checkPanicExists();
    super.signOut();

    super.signIn("company1", "company1");
    super.request("/lecturer/course/publish", params);
    super.checkPanicExists();
    super.signOut();

    super.signIn("auditor1", "auditor1");
    super.request("/lecturer/course/publish", params);
    super.checkPanicExists();
    super.signOut();
}
}

```

En este test, lo que hacemos es ir probando que ninguno de los demás roles, exceptuando el lecturer que es el nuestro, puedan publicar un course y que devuelve un error 500 indicando que no estamos autorizados, que es lo que estamos buscando.

Test Hacking

```
@Test
public void test301Hacking() {

    Collection<Course> courses;
    String params;

    super.signIn("lecturer1", "lecturer1");
    courses = this.repository.findManyCoursesByLecturerUsername("lecturer1");
    for (final Course course : courses)
        if (!course.isDraftMode()) {
            params = String.format("id=%d", course.getId());
            super.request("/lecturer/course/publish", params);
        }
    super.signOut();
}
```

En este test, lo que hacemos es comprobar que no pueden publicar un course que ya ha sido publicado previamente y que devuelve un error 500 indicando que no estamos autorizados, que es lo que estamos buscando.

Test Hacking

```
@Test
public void test302Hacking() {

    Collection<Course> courses;
    String params;

    super.signIn("lecturer1", "lecturer1");
    courses = this.repository.findManyCoursesByLecturerUsername("lecturer2");
    for (final Course course : courses) {
        params = String.format("id=%d", course.getId());
        super.request("/lecturer/course/publish", params);
    }
    super.signOut();
}
```

En este test, lo que hacemos es comprobar que no pueden publicar un course que pertenece a otro lecturer y que devuelve un error 500 indicando que no estamos autorizados, que es lo que estamos buscando.

Crear Course Lecture Session

```
@ParameterizedTest
@CsvFileSource(resources = "/lecturer/course-lecture/create-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int lectureIndex, final int courseIndex, final String course) {
    super.signIn("lecturer17", "lecturer17");

    super.clickOnMenu("Lecturer", "List of courses");

    super.clickOnListingRecord(courseIndex);
    super.clickOnButton("Lectures");
    super.checkListingEmpty();

    super.clickOnMenu("Lecturer", "List of lectures");

    super.checkListingExists();
    super.clickOnListingRecord(lectureIndex);
    super.clickOnButton("Add to course");
    super.fillInputBoxIn("course", course);
    super.clickOnSubmit("Add");

    super.clickOnMenu("Lecturer", "List of courses");

    super.clickOnListingRecord(courseIndex);
    super.clickOnButton("Lectures");
    super.checkNotListingEmpty();

    super.signOut();
}
```

En este test lo que hacemos en primer lugar es iniciar sesión como un lecturer, acceder a un course y comprobar que no tiene lectures asociadas. Después, accedemos a una lecture y la añadimos a un curso, para después comprobar que este curso ya tiene alguna lecture asociada.

```
@ParameterizedTest
@CsvfileSource(resources = "/lecturer/course-lecture/create-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int lectureIndex, final int courseIndex, final String course) {

    super.signIn("lecturer16", "lecturer16");

    super.clickOnMenu("Lecturer", "List of courses");

    super.clickOnListingRecord(courseIndex);
    super.clickOnButton("Lectures");
    super.checkNotListingEmpty();

    super.clickOnMenu("Lecturer", "List of lectures");

    super.checkListingExists();
    super.clickOnListingRecord(lectureIndex);
    super.clickOnButton("Add to course");
    super.fillInputBoxIn("course", course);
    super.clickOnSubmit("Add");
    super.checkErrorsExist();

    super.clickOnMenu("Lecturer", "List of courses");

    super.clickOnListingRecord(courseIndex);
    super.clickOnButton("Lectures");
    super.checkNotListingEmpty();

    super.signOut();
}
```

En este test lo que hacemos en primer lugar es iniciar sesión como un lecturer, acceder a un course y comprobar que tiene lectures asociadas. Después, accedemos a una lecture y intentamos añadirla al mismo curso, para que nos salte un error de que esta lecture ya está asociada a ese course..

```

@Test
public void test300Hacking() {

    super.checkLinkExists("Sign in");
    super.request("/lecturer/course-lecture/create");
    super.checkPanicExists();

    super.signIn("administrator", "administrator");
    super.request("/lecturer/course-lecture/create");
    super.checkPanicExists();
    super.signOut();

    super.signIn("student1", "student1");
    super.request("/lecturer/course-lecture/create");
    super.checkPanicExists();
    super.signOut();

    super.signIn("company1", "company1");
    super.request("/lecturer/course-lecture/create");
    super.checkPanicExists();
    super.signOut();

    super.signIn("assistant1", "assistant1");
    super.request("/lecturer/course-lecture/create");
    super.checkPanicExists();
    super.signOut();

    super.signIn("auditor1", "auditor1");
    super.request("/lecturer/course-lecture/create");
    super.checkPanicExists();
    super.signOut();
}

```

Listar Lecture

Test Positivo

```

@ParameterizedTest
@CsvFileSource(resources = "/lecturer/lecture/list-positive.csv", encoding = "utf-8"
public void test100Positive(final int lectureRecordIndex, final String title, final

    super.signIn("lecturer1", "lecturer1");

    super.clickOnMenu("Lecturer", "List of lectures");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.checkColumnHasValue(lectureRecordIndex, 0, title);
    super.checkColumnHasValue(lectureRecordIndex, 1, abstracts);
    super.checkColumnHasValue(lectureRecordIndex, 2, estimatedTime);

    super.signOut();
}

// HINT: there aren't any negative tests for this feature since it's a listing that
// HINT+ doesn't involve entering any data into any forms.

```

En este test en primer lugar, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de forma ascendente de courses asociados a ese lecturer (comprobando que se muestra) y se comprobará que el valor de las columna “code” coincide con los que le hemos metido como parámetro para cada course. Tras esto, se accede al formulario de cada course y se comprueba que el valor de “code” sigue coincidiendo con el que le pasamos como parámetro. Comprobado esto se pulsará el botón Lecture para mostrar la lista de lectures asociados al course(se comprueba que aparece) y por último se comprueba que el atributo “título” coincide con el pasado 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 courses asociados a ese lecturer son correctos y que no aparecen otros los cuales pertenecen a otros lecturer, además que los lectures asociados a ese course aparezca correctamente y no los de otro course y sobre todo que se muestren las listas con los datos correctos.

Test Hacking

```
@Test  
public void test300Hacking() {  
  
    super.checkLinkExists("Sign in");  
    super.request("/lecturer/lecture/list-all");  
    super.checkPanicExists();  
  
    super.signIn("administrator", "administrator");  
    super.request("/lecturer/lecture/list-all");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("assistant1", "assistant1");  
    super.request("/lecturer/lecture/list-all");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("student1", "student1");  
    super.request("/lecturer/lecture/list-all");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("company1", "company1");  
    super.request("/lecturer/lecture/list-all");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("auditor1", "auditor1");  
    super.request("/lecturer/lecture/list-all");  
    super.checkPanicExists();  
    super.signOut();  
}
```

En este test, lo que hacemos es ir probando que ninguno de los demás roles ni los demás lecturer, no pueden acceder a los listado de los lectures y que devuelve un error 500 indicando que no estamos autorizados.

Show Lecture

Test Positivo

```
@ParameterizedTest
@CsvFileSource(resources = "/lecturer/lecture/show-positive.csv", encoding = "utf-8",
public void test100Positive(final int lectureIndex, final String title, final String abstracts, final String estimatedTime, final String body, final String nature, final String link) {
    super.signIn("lecturer3", "lecturer3");

    super.clickOnMenu("Lecturer", "List of lectures");
    super.sortListing(0, "asc");
    super.clickOnListingRecord(lectureIndex);
    super.checkFormExists();

    super.checkInputBoxHasValue("title", title);
    super.checkInputBoxHasValue("abstracts", abstracts);
    super.checkInputBoxHasValue("estimatedTime", estimatedTime);
    super.checkInputBoxHasValue("body", body);
    super.checkInputBoxHasValue("nature", nature);
    super.checkInputBoxHasValue("link", link);

    super.signOut();
}
```

En este test en primer lugar, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de forma ascendente de courses asociados a ese lecturer (comprobando que se muestra).

Tras esto, se accede al formulario de cada course y se pulsará el botón Lecture para mostrar la lista de lectures asociados al course(se comprueba que aparece) y por último se pulsará sobre cada

lectures y se comprueba que todos los atributos coinciden con los pasados 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 courses asociados a ese lecturer son correctos y que no aparecen otros, los cuales, pertenecen a otros lecturer, además que los lectures asociados a ese course aparezca correctamente y no los de otro course y sobre todo que se muestren las listas y los formularios asociados con los datos correctos según las reglas de negocio impuestas.

Test Hacking

```
@Test
public void test300Hacking() {
    Collection<Lecture> lectures;
    String param;
    lectures = this.repository.findManyLecturesByLecturerUsername("lecturer3");
    for (final Lecture lecture : lectures)
        if (lecture.isDraftMode()) {
            param = String.format("id=%d", lecture.getId());

            super.checkLinkExists("Sign in");
            super.request("/lecturer/lecture/show", param);
            super.checkPanicExists();

            super.signIn("administrator", "administrator");
            super.request("/lecturer/lecture/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("assistant5", "assistant5");
            super.request("/lecturer/lecture/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("lecturer1", "lecturer1");
            super.request("/lecturer/lecture/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("student1", "student1");
            super.request("/lecturer/lecture/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("company1", "company1");
            super.request("/lecturer/lecture/show", param);
            super.checkPanicExists();
            super.signOut();

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

En este test, lo que hacemos es ir probando que ninguno de los demás roles puedan asociar lectures a courses y que devuelva un error 500 indicando que no estamos autorizados.

Create Lecture

Test Positivo

```

@ParameterizedTest
@CsvFileSource(resources = "/lecturer/lecture/create-positive.csv", encoding = "UTF-8")
public void test100Positive(final int lectureIndex, final String title, final String abstracts, final String estimatedTime, final String body, final String nature, final String link)
{
    super.signIn("lecturer2", "lecturer2");

    super.clickOnMenu("Lecturer", "List of lectures");
    super.checkListingExists();

    super.clickOnButton("Create lecture");
    super.fillInputBoxIn("title", title);
    super.fillInputBoxIn("abstracts", abstracts);
    super.fillInputBoxIn("estimatedTime", estimatedTime);
    super.fillInputBoxIn("body", body);
    super.fillInputBoxIn("nature", nature);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Create lecture");

    super.clickOnMenu("Lecturer", "List of lectures");
    super.checkListingExists();
    super.sortListing(0, "asc");
    super.checkColumnHasValue(lectureIndex, 0, title);
    super.checkColumnHasValue(lectureIndex, 1, abstracts);
    super.checkColumnHasValue(lectureIndex, 2, estimatedTime);

    super.clickOnListingRecord(lectureIndex);
    super.checkFormExists();
    super.checkInputBoxHasValue("title", title);
    super.checkInputBoxHasValue("abstracts", abstracts);
    super.checkInputBoxHasValue("estimatedTime", estimatedTime);
    super.checkInputBoxHasValue("body", body);
    super.checkInputBoxHasValue("nature", nature);
    super.checkInputBoxHasValue("link", link);

    super.clickOnButton("Add to course");
    super.checkFormExists();

    super.signOut();
}

```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de lectures de forma ascendente (comprobando que se muestra), tras esto al pulsar el botón de create saldrá el formulario de creación, se añadirá valores positivos en cada campo y tras esto se pulsará el botón de create.

Una vez que se ha creado correctamente, comprobamos que existe la lista de lectures asociado a ese lecturer, comprobamos que la lista muestra correctamente los datos del lecture 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 añadidos.

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

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

Test Negativo

```
@ParameterizedTest
@CsvFileSource(resources = "/lecturer/lecture/create-negative.csv", encoding = "utf-8",
public void test200Negative(final int lectureIndex, final String title, final String abs
super.signIn("lecturer2", "lecturer2");

super.clickOnMenu("Lecturer", "List of lectures");
super.clickOnButton("Create lecture");
super.checkFormExists();

super.fillInputBoxIn("title", title);
super.fillInputBoxIn("abstracts", abstracts);
super.fillInputBoxIn("estimatedTime", estimatedTime);
super.fillInputBoxIn("body", body);
super.fillInputBoxIn("nature", nature);
super.fillInputBoxIn("link", link);
super.clickOnSubmit("Create lecture");
super.checkErrorsExist();

super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, pulsamos el botón de create lecture 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.

Test Hacking

```
@Test  
public void test300Hacking() {  
  
    super.checkLinkExists("Sign in");  
    super.request("/lecturer/lecture/create");  
    super.checkPanicExists();  
  
    super.signIn("administrator", "administrator");  
    super.request("/lecturer/lecture/create");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("student1", "student1");  
    super.request("/lecturer/lecture/create");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("company1", "company1");  
    super.request("/lecturer/lecture/create");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("assistant1", "assistant1");  
    super.request("/lecturer/lecture/create");  
    super.checkPanicExists();  
    super.signOut();  
  
    super.signIn("auditor1", "auditor1");  
    super.request("/lecturer/lecture/create");  
    super.checkPanicExists();  
    super.signOut();  
}
```

En este test, lo que hacemos es ir probando que ninguno de los demás roles, no pueden crear lectures y que devuelve un error 500 indicando que no estamos autorizados.

Update Lecture

Test Positivo

```

@ParameterizedTest
@CsvFileSource(resources = "/lecturer/lecture/update-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int lectureIndex, final String title, final String abstracts, final String
super.signIn("lecturer10", "lecturer10");

super.clickOnMenu("Lecturer", "List of lectures");
super.checkListingExists();
super.sortListing(0, "asc");

super.clickOnListingRecord(lectureIndex);
super.checkFormExists();
super.fillInputBoxIn("title", title);
super.fillInputBoxIn("abstracts", abstracts);
super.fillInputBoxIn("estimatedTime", estimatedTime);
super.fillInputBoxIn("body", body);
super.fillInputBoxIn("nature", nature);
super.fillInputBoxIn("link", link);
super.clickOnSubmit("Update");

super.checkListingExists();
super.sortListing(0, "asc");
super.checkColumnHasValue(lectureIndex, 0, title);
super.checkColumnHasValue(lectureIndex, 1, abstracts);
super.checkColumnHasValue(lectureIndex, 2, estimatedTime);

super.clickOnListingRecord(lectureIndex);
super.checkFormExists();
super.checkInputBoxHasValue("title", title);
super.checkInputBoxHasValue("abstracts", abstracts);
super.checkInputBoxHasValue("estimatedTime", estimatedTime);
super.checkInputBoxHasValue("body", body);
super.checkInputBoxHasValue("nature", nature);
super.checkInputBoxHasValue("link", link);

super.signOut();
}

```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de lectures de forma ascendente (comprobando que se muestra), pulsamos en cada lecture asociados a ese lecturer. Tras esto aparecerá el formulario con todos los datos (se comprueba que existe), 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 lecture asociado a ese lecturer, comprobamos que la lista muestra correctamente los datos del lecture 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 cumplen con las reglas de negocio impuestas y no saltan errores y permite crear el formulario correctamente.

Test Negative

```
@ParameterizedTest
@CsvFileSource(resources = "/lecturer/lecture/update-negative.csv", encoding = "utf-8", numLi
public void test200Negative(final int lectureIndex, final String title, final String abstract
    super.signIn("lecturer10", "lecturer10");

    super.clickOnMenu("Lecturer", "List of lectures");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(lectureIndex);
    super.checkFormExists();
    super.fillInputBoxIn("title", title);
    super.fillInputBoxIn("abstracts", abstracts);
    super.fillInputBoxIn("estimatedTime", estimatedTime);
    super.fillInputBoxIn("body", body);
    super.fillInputBoxIn("nature", nature);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Update");

    super.checkErrorsExist();

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de lectures de forma ascendente (comprobando que se muestra), pulsamos en cada lecture asociados a ese lecturer. Tras esto aparecerá el formulario con todos los datos (se comprueba que existe), se actualizará 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.

Test Hacking

```
@Test
public void test300Hacking() {
    Collection<Lecture> lectures;
    String param;
    lectures = this.repository.findManyLecturesByLecturerUsername("lecturer10");
    for (final Lecture lecture : lectures) {
        param = String.format("id=%d", lecture.getId());

        super.checkLinkExists("Sign in");
        super.request("/lecturer/lecture/update", param);
        super.checkPanicExists();

        super.signIn("administrator", "administrator");
        super.request("/lecturer/lecture/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("lecturer2", "lecturer2");
        super.request("/lecturer/lecture/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("student2", "student2");
        super.request("/lecturer/lecture/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("assistant2", "assistant2");
        super.request("/lecturer/lecture/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("company2", "company2");
        super.request("/lecturer/lecture/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("auditor2", "auditor2");
        super.request("/lecturer/lecture/update", param);
        super.checkPanicExists();
        super.signOut();
    }
}
```

En este test, lo que hacemos es ir probando que ninguno de los demás roles puedan actualizar lectures y que devuelve un error 500 indicando que no estamos autorizados.

Publish Course

Test Positive

```
@ParameterizedTest
@CsvFileSource(resources = "/lecturer/lecture/publish-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int lectureIndex, final String title) {

    super.signIn("lecturer5", "lecturer5");

    super.clickOnMenu("Lecturer", "List of lectures");
    super.checkListingExists();
    super.sortListing(0, "asc");
    super.checkColumnHasValue(lectureIndex, 0, title);

    super.clickOnListingRecord(lectureIndex);
    super.checkFormExists();
    super.clickOnSubmit("Publish");
    super.checkNotErrorsExist();

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un lecturer, esto hará que nos redirija al menú de lecturer, se nos mostrará la lista de lectures de forma ascendente (comprobando que se muestra), pulsamos en cada lecture y tras esto aparecerá el formulario con todos los datos (se comprueba que existe), tras esto se pulsará el botón de publish y observaremos que no produce errores.

El ejecutar los test de esta manera nos permite detectar que se publican lectures.

Test Hacking

```
@Test
public void test300Hacking() {

    Collection<Lecture> lectures;
    String params;

    lectures = this.repository.findManyLecturesByLecturerUsername("lecturer2");
    for (final Lecture lecture : lectures)
        if (lecture.isDraftMode()) {
            params = String.format("id=%d", lecture.getId());

            super.checkLinkExists("Sign in");
            super.request("/lecturer/lecture/publish", params);
            super.checkPanicExists();

            super.signIn("administrator", "administrator");
            super.request("/lecturer/lecture/publish", params);
            super.checkPanicExists();
            super.signOut();

            super.signIn("assistant1", "assistant1");
            super.request("/lecturer/lecture/publish", params);
            super.checkPanicExists();
            super.signOut();

            super.signIn("student1", "student1");
            super.request("/lecturer/lecture/publish", params);
            super.checkPanicExists();
            super.signOut();

            super.signIn("company1", "company1");
            super.request("/lecturer/lecture/publish", params);
            super.checkPanicExists();
            super.signOut();

            super.signIn("auditor1", "auditor1");
            super.request("/lecturer/lecture/publish", params);
            super.checkPanicExists();
            super.signOut();
        }
}
```

En este test, lo que hacemos es ir probando que ninguno de los demás roles, exceptuando el lecturer que es el nuestro, no pueden publicar un lecture y que devuelve un error 500 indicando que no estamos autorizados, que es lo que estamos buscando.

Test Hacking

```
@Test  
public void test301Hacking() {  
  
    Collection<Lecture> lectures;  
    String params;  
  
    super.signIn("lecturer1", "lecturer1");  
    lectures = this.repository.findManyLecturesByLecturerUsername("lecturer1");  
    for (final Lecture lecture : lectures)  
        if (!lecture.isDraftMode()) {  
            params = String.format("id=%d", lecture.getId());  
            super.request("/lecturer/lecture/publish", params);  
        }  
    super.signOut();  
}
```

En este test, lo que hacemos es comprobar que no pueden publicar un lecture que ya ha sido publicado previamente y que devuelve un error 500 indicando que no estamos autorizados, que es lo que estamos buscando.

Test Hacking

```
@Test  
public void test302Hacking() {  
  
    Collection<Lecture> lectures;  
    String params;  
  
    super.signIn("lecturer1", "lecturer1");  
    lectures = this.repository.findManyLecturesByLecturerUsername("lecturer2");  
    for (final Lecture lecture : lectures) {  
        params = String.format("id=%d", lecture.getId());  
        super.request("/lecturer/lecture/publish", params);  
    }  
    super.signOut();  
}
```

En este test, lo que hacemos es comprobar que no pueden publicar un lecture que pertenece a otro lecturer y que devuelve un error 500 indicando que no estamos autorizados, que es lo que estamos buscando.

6 TEST REQUEST PERFORMANCE

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.

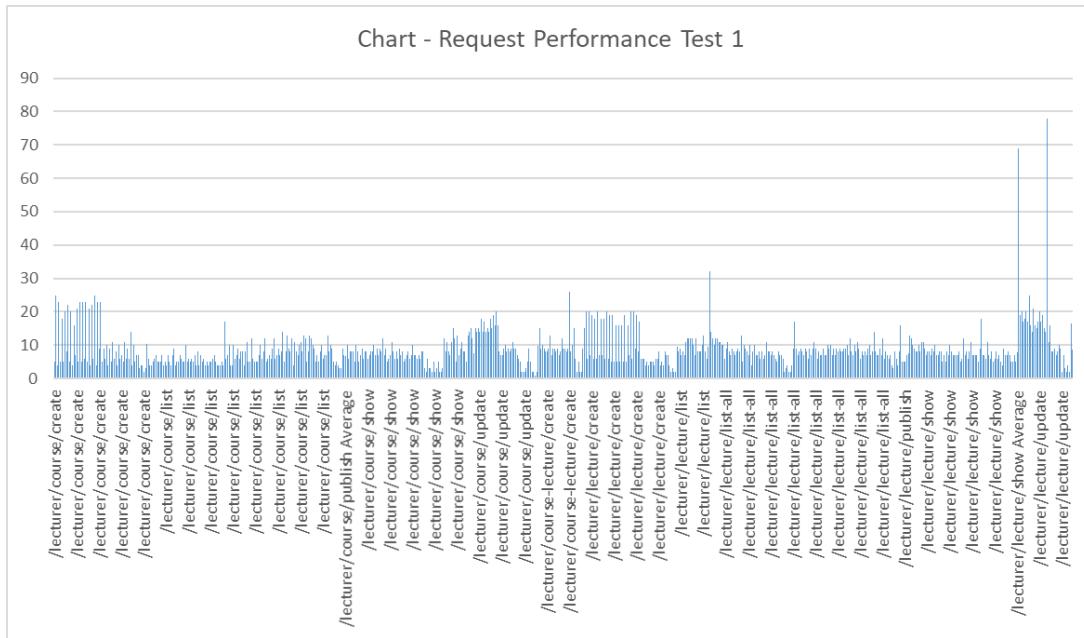


Gráfico del promedio del tiempo de las solicitudes - PC 1

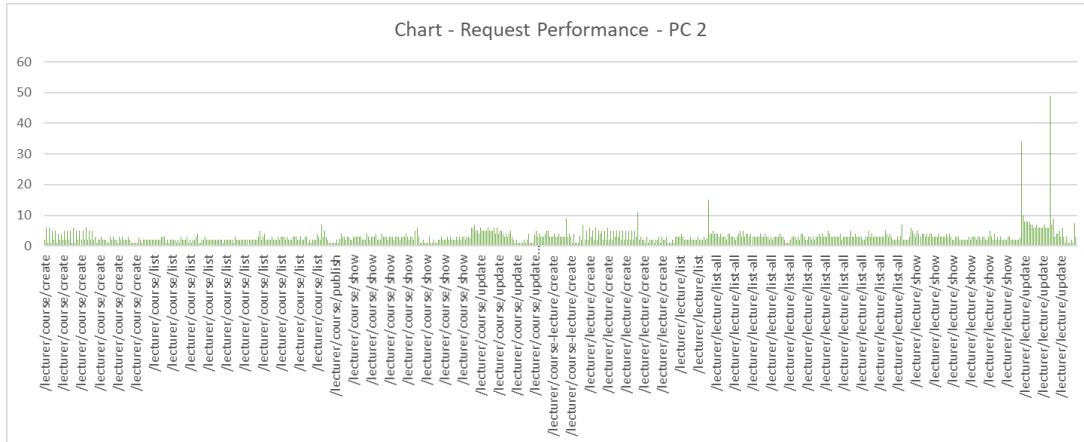


Gráfico del promedio del tiempo de las solicitudes - PC 2

→ 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.

<i>Test - mannavsic - PC 1</i>		<i>Result PC 2</i>	
Mean	8,72238806	Mean	3,077677782
Standard Error	0,222250077	Standard Error	0,1001706
Median	8	Median	3
Mode	7	Mode	2
Standard Deviation	5,752800602	Standard Deviation	2,617885366
Sample Variance	33,09471477	Sample Variance	6,85332379
Kurtosis	48,98540935	Kurtosis	167,0418928
Skewness	5,013254329	Skewness	10,70408039
Range	77	Range	48
Minimum	1	Minimum	1
Maximum	78	Maximum	49
Sum	5844	Sum	2102,053925
Count	670	Count	683
Confidence Level(95,0%)	0,436391648	Confidence Level(95,0%)	0,19667981
<i>Interval (ms):</i>	8,285996412 9,15878	<i>Interval (ms):</i>	2,880997972 3,274358
<i>Interval (s):</i>	0,008285996 0,009159	<i>Interval (s):</i>	0,002880998 0,003274

7 TEST PERFORMANCE REPORT

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.

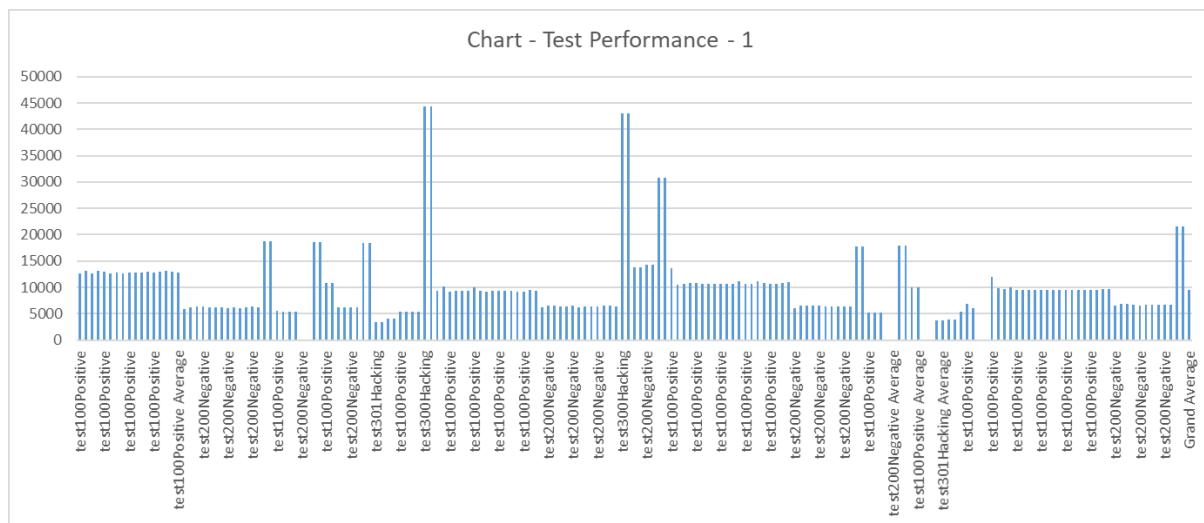


Gráfico del tiempo promedio de ejecución de los tests (de 0 a 450 segundos) - PC1

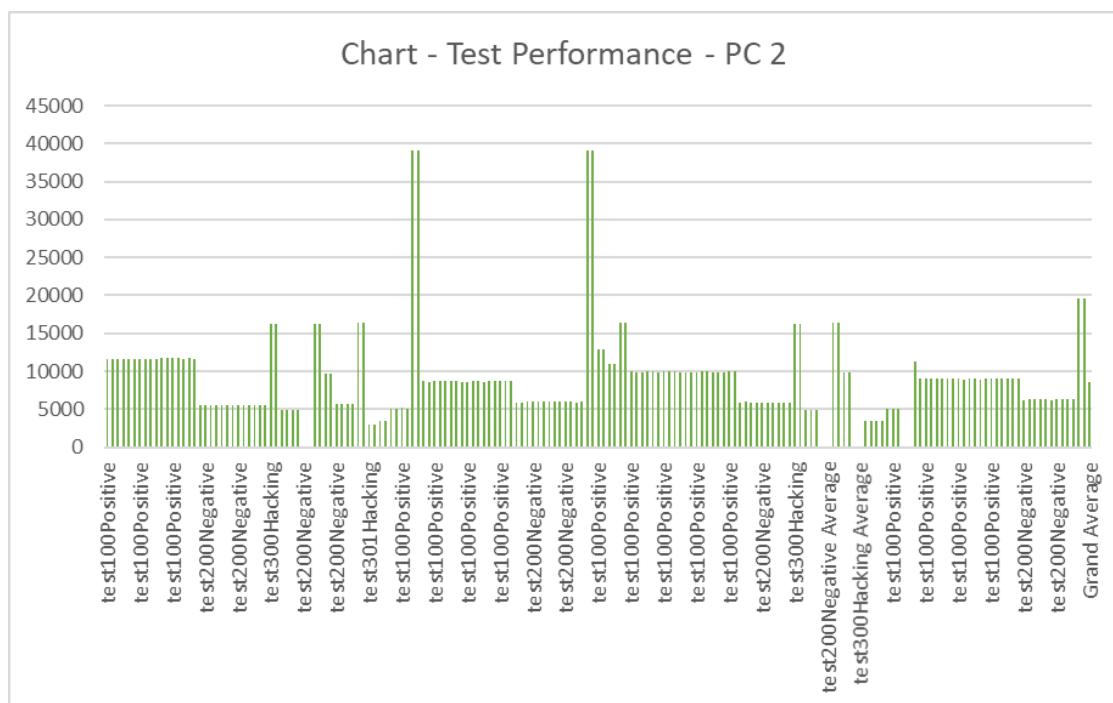


Gráfico del tiempo promedio de ejecución de los tests (de 0 a 400 segundos) - PC2

8 Z-TEST

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

z-Test: Two Sample for Means		
	BEFORE	AFTER
Mean	8,72238806	8,728795315
Known Variance	33,09471477	32,57231778
Observations	670	683
Hypothesized Mean Diff	0	
z	-0,020563431	
P(Z<=z) one-tail	0,491796956	
z Critical one-tail	1,644853627	
P(Z<=z) two-tail	0,983593912	
z Critical two-tail	1,959963985	

$$PC\ 1: \alpha = 1 - Confidence\ level = 1 - 0,43639 = 0,56361$$

$$PC\ 2 : \alpha = 1 - Confidence\ level = 1 - 0,196679 = 0,803321$$

Como observamos el P-Value ($P(Z<=z)$ two tail) que es 0,983593.. 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, 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.

9 CONCLUSIÓN

Intencionalmente en blanco.

10 BIBLIOGRAFÍA

Intencionalmente en blanco.