DP II - 2022-2023

TEST REPORT

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

Alumno: ÁNGEL LORENZO CASAS

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 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 VERSIONADO

Versión	Fecha	Descripción
1.0	06/07/2023	Primera versión del documento
1.1	06/07/2023	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 5, los cuáles se han realizado en el Sprint 4 proporcionando resultados de que las features realizadas cumplen con la expectativas y no tienen bugs que provocan que nuestros clientes no están contento con nuestro trabajo. Además analizaremos cómo ha sido el rendimiento de su ejecución de la aplicación en varios Pcs.

5 CONTENIDO

Pruebas Funcionales

Listar Audits

Test Positivo

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/audit/list-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int auditRecordIndex, final String code, final String conclusion, final String modeMark) {
    // HINT: this test authenticates as an auditor, lists his or her audits only,
    // HINT+ and then checks that the listing has the expected data.

    super.signIn("auditor4", "auditor4");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.checkColumnHasValue(auditRecordIndex, 0, code);
    super.checkColumnHasValue(auditRecordIndex, 1, conclusion);
    super.checkColumnHasValue(auditRecordIndex, 2, modeMark);

    super.signOut();
}
```

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

Test Hacking

```
    @Test
    public void test300Hacking() {

        super.checkLinkExists("Sign in");
        super.request("/auditor/audit/list-all");
        super.checkPanicExists();

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

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

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

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

        super.signIn("assistant1", "assistant1");
        super.request("/auditor/audit/list-all");
        super.checkPanicExists();
        super.signOut();
    }
```

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

Show Audit

Test Positive

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/audit/show-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int auditIndex, final String code, final String conclusion, final String strongPoints,
    final String weakPoints, final String course, final String modeMark) {

    super.signIn("auditor4", "auditor4");

    super.clickOnMenu("Auditor", "Audit List");
    super.sortListing(0, "asc");
    super.clickOnListingRecord(auditIndex);
    super.checkFormExists();

    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("conclusion", conclusion);
    super.checkInputBoxHasValue("strongPoints", strongPoints);
    super.checkInputBoxHasValue("weakPoints", weakPoints);
    super.checkInputBoxHasValue("course", course);
    super.checkInputBoxHasValue("modeMark", modeMark);

    super.signOut();
}
```

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

Test Hacking

```
@Test
public void test300Hacking() {

    Collection<Audit> audits;
    String param;

    audits = this.repository.findManyAuditsByAuditorUsername("auditor3");
    for (final Audit audit : audits)
        if (audit.isDraftMode()) {
            param = String.format("id=%d", audit.getId());

            super.checkLinkExists("Sign in");
            super.request("/auditor/audit/show", param);
            super.checkPanicExists();

            super.signIn("administrator", "administrator");
            super.request("/auditor/audit/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("auditor5", "auditor5");
            super.request("/auditor/audit/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("lecturer1", "lecturer1");
            super.request("/auditor/audit/show", param);
            super.checkPanicExists();
            super.signOut();


            super.signIn("student1", "student1");
            super.request("/auditor/audit/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("company1", "company1");
            super.request("/auditor/audit/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("assistant1", "assistant1");
            super.request("/auditor/audit/show", param);
            super.checkPanicExists();
            super.signOut();
        }
}
```

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

Create Audit

Test Positive

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/audit/create-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int auditIndex, final String code, final String conclusion,
    final String strongPoints, final String weakPoints, final String course, final String modeMark) {

    super.signIn("auditor13", "auditor13");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();

    super.clickOnButton("Create");
    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("conclusion", conclusion);
    super.fillInputBoxIn("strongPoints", strongPoints);
    super.fillInputBoxIn("weakPoints", weakPoints);
    super.fillInputBoxIn("course", course);
    super.clickOnSubmit("Create");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");
    super.checkColumnHasValue(auditIndex, 0, code);
    super.checkColumnHasValue(auditIndex, 1, conclusion);
    super.checkColumnHasValue(auditIndex, 2, modeMark);

    super.clickOnListingRecord(auditIndex);
    super.checkFormExists();

    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("conclusion", conclusion);
    super.checkInputBoxHasValue("strongPoints", strongPoints);
    super.checkInputBoxHasValue("weakPoints", weakPoints);
    super.checkInputBoxHasValue("course", course);
    super.checkInputBoxHasValue("modeMark", modeMark);

    super.clickOnButton("Audit Records");
    super.checkListingExists();
    super.checkListingEmpty();

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un auditor, esto hará que nos redirija al menú de auditor, se nos mostrará la lista de audits 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 audits asociado a ese auditor, comprobamos que la lista muestra correctamente los datos del audit 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 Audit Records y que al pulsar sobre él da lugar a una lista de auditing records 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 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 que se genera un botón que deriva a los auditings records de ese audit.

Test Negative

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/audit/create-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int auditIndex, final String code, final String conclusion,
    final String strongPoints, final String weakPoints, final String course, final String modeMark) {

    super.signIn("auditor13", "auditor13");

    super.clickOnMenu("Auditor", "Audit List");
    super.clickOnButton("Create");
    super.checkFormExists();

    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("conclusion", conclusion);
    super.fillInputBoxIn("strongPoints", strongPoints);
    super.fillInputBoxIn("weakPoints", weakPoints);
    super.fillInputBoxIn("course", course);
    super.clickOnSubmit("Create");
    super.checkErrorsExist();

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un auditor, esto hará que nos redirija al menú de auditor, 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("/auditor/audit/create");
    super.checkPanicExists();

    super.signIn("administrator", "administrator");
    super.request("/auditor/audit/create");
    super.checkPanicExists();
    super.signOut();

    super.signIn("student1", "student1");
    super.request("/auditor/audit/create");
    super.checkPanicExists();
    super.signOut();

    super.signIn("company1", "company1");
    super.request("/auditor/audit/create");
    super.checkPanicExists();
    super.signOut();

    super.signIn("lecturer1", "lecturer1");
    super.request("/auditor/audit/create");
    super.checkPanicExists();
    super.signOut();

    super.signIn("assistant1", "assistant1");
    super.request("/auditor/audit/create");
    super.checkPanicExists();
    super.signOut();
}
```

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

Update Audit

Test Positive

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/audit/update-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int auditIndex, final String code, final String conclusion,
    final String strongPoints, final String weakPoints, final String course, final String modeMark) {

    super.signIn("auditor3", "auditor3");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(auditIndex);
    super.checkFormExists();
    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("conclusion", conclusion);
    super.fillInputBoxIn("strongPoints", strongPoints);
    super.fillInputBoxIn("weakPoints", weakPoints);
    super.fillInputBoxIn("course", course);
    super.clickOnSubmit("Update");

    super.checkListingExists();
    super.sortListing(0, "asc");
    super.checkColumnHasValue(auditIndex, 0, code);
    super.checkColumnHasValue(auditIndex, 1, conclusion);
    super.checkColumnHasValue(auditIndex, 2, modeMark);

    super.clickOnListingRecord(auditIndex);
    super.checkFormExists();
    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("conclusion", conclusion);

    super.checkInputBoxHasValue("strongPoints", strongPoints);
    super.checkInputBoxHasValue("weakPoints", weakPoints);
    super.checkInputBoxHasValue("course", course);
    super.checkInputBoxHasValue("modeMark", modeMark);

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un auditor, esto hará que nos redirija al menú de auditor, se nos mostrará la lista de audits de forma ascendente (comprobando que se muestra), pulsamos en cada audit asociados a ese auditor 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 audits asociado a ese auditor, comprobamos que la lista muestra correctamente los datos del audit 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.

Test Negative

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/audit/update-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int auditIndex, final String code, final String conclusion,
    final String strongPoints, final String weakPoints, final String course, final String modeMark) {

    super.signIn("auditor3", "auditor3");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(auditIndex);
    super.checkFormExists();
    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("conclusion", conclusion);
    super.fillInputBoxIn("strongPoints", strongPoints);
    super.fillInputBoxIn("weakPoints", weakPoints);
    super.fillInputBoxIn("course", course);
    super.clickOnSubmit("Update");

    super.checkErrorsExist();

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un auditor, esto hará que nos redirija al menú de auditor, se nos mostrará la lista de audits de forma ascendente (comprobando que se muestra), pulsamos en cada audit asociados a ese auditor 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<Audit> audits;
    String param;

    audits = this.repository.findManyAuditsByAuditorUsername("auditor1");
    for (final Audit audit : audits) {
        param = String.format("id=%d", audit.getId());

        super.checkLinkExists("Sign in");
        super.request("/auditor/audit/update", param);
        super.checkPanicExists();

        super.signIn("administrator", "administrator");
        super.request("/auditor/audit/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("lecturer2", "lecturer2");
        super.request("/auditor/audit/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("student2", "student2");
        super.request("/auditor/audit/update", param);
        super.checkPanicExists();
        super.signOut();


       super.signIn("auditor2", "auditor2");
       super.request("/employer/job/update", param);
       super.checkPanicExists();
       super.signOut();

       super.signIn("company2", "company2");
       super.request("/auditor/audit/update", param);
       super.checkPanicExists();
       super.signOut();

       super.signIn("assistant2", "assistant2");
       super.request("/auditor/audit/update", param);
       super.checkPanicExists();
       super.signOut();
    }
}
```

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

Publish Audit

Test Positive

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/audit/publish-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int auditIndex, final String code) {
    // HINT: this test authenticates as an employer, lists his or her jobs,
    // HINT: then selects one of them, and publishes it.

    super.signIn("auditor10", "auditor10");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");
    super.checkColumnHasValue(auditIndex, 0, code);

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

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un auditor, esto hará que nos redirija al menú de auditor, se nos mostrará la lista de audits de forma ascendente (comprobando que se muestra), pulsamos en cada audit asociados a ese auditor 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 audit los cuales tienen un auditing record asociado.

Test Negative

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/audit/publish-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int auditIndex, final String code) {
    // HINT: this test attempts to publish a job that cannot be published, yet.

    super.signIn("auditor5", "auditor5");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.checkColumnHasValue(auditIndex, 0, code);
    super.clickOnListingRecord(auditIndex);
    super.checkFormExists();
    super.clickOnSubmit("Publish");
    super.checkAlertExists(false);

    super.signOut();
}
```

En este test, lo que hacemos es iniciar sesión con un auditor, esto hará que nos redirija al menú de auditor, se nos mostrará la lista de audits de forma ascendente (comprobando que se muestra), pulsamos en cada audit asociados a ese auditor 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 audits los cuales no tienen un auditing record asociado.

Test Hacking

```
@Test
public void test300Hacking() {

    Collection<Audit> audits;
    String params;

    audits = this.repository.findManyAuditsByAuditorUsername("auditor10");
    for (final Audit audit : audits)
        if (audit.isDraftMode()) {
            params = String.format("id=%d", audit.getId());

            super.checkLinkExists("Sign in");
            super.request("auditor/audit/publish", params);
            super.checkPanicExists();

            super.signIn("administrator", "administrator");
            super.request("auditor/audit/publish", params);
            super.checkPanicExists();
            super.signOut();

            super.signIn("lecturer1", "lecturer1");
            super.request("auditor/audit/publish", params);
            super.checkPanicExists();
            super.signOut();

            super.signIn("student1", "student1");
            super.request("auditor/audit/publish", params);
            super.checkPanicExists();
            super.signOut();


            super.signIn("company1", "company1");
            super.request("auditor/audit/publish", params);
            super.checkPanicExists();
            super.signOut();

            super.signIn("company1", "company1");
            super.request("auditor/audit/publish", params);
            super.checkPanicExists();
            super.signOut();

            super.signIn("assistant1", "assistant1");
            super.request("auditor/audit/publish", params);
            super.checkPanicExists();
            super.signOut();
        }
}
```

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

Test Hacking

```
@Test
public void test301Hacking() {

    Collection<Audit> audits;
    String params;

    super.signIn("auditor1", "auditor1");
    audits = this.repository.findManyAuditsByAuditorUsername("auditor1");
    for (final Audit audit : audits)
        if (!audit.isDraftMode()) {
            params = String.format("id=%d", audit.getId());
            super.request("/auditor/audit/publish", params);
            super.checkPanicExists();
        }
    super.signOut();
}
```

En este test, lo que hacemos es comprobar que no pueden publicar un audit 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<Audit> audits;
    String params;

    super.signIn("auditor1", "auditor1");
    audits = this.repository.findManyAuditsByAuditorUsername("auditor2");
    for (final Audit audit : audits) {
        params = String.format("id=%d", audit.getId());
        super.request("/auditor/audit/publish", params);
        super.checkPanicExists();

    }
    super.signOut();
}
```

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

Listar Auditing Record

Test Positivo

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/auditing-record/list-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int auditRecordIndex, final String code, final int auditingRecordRecordIndex,
    final String subject) {

    super.signIn("auditor1", "auditor1");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.checkColumnHasValue(auditRecordIndex, 0, code);
    super.clickOnListingRecord(auditRecordIndex);
    super.checkInputBoxHasValue("code", code);
    super.clickOnButton("Audit Records");

    super.checkListingExists();
    super.checkColumnHasValue(auditingRecordRecordIndex, 0, subject);
    super.clickOnListingRecord(auditingRecordRecordIndex);

    super.signOut();
}
```

En este test en primer lugar, lo que hacemos es iniciar sesión con un auditor, esto hará que nos redirija al menú de auditor, se nos mostrará la lista de forma ascendente de audits asociados a ese auditor(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 audit. Tras esto, se accede al formulario de cada audit y se comprueba que el valor de “code” sigue coincidiendo con el que le pasamos como parámetro. Comprobado esto se pulsará el botón Audit Records para mostrar la lista de auditing records asociados al audit(se comprueba que aparece) y por último se comprueba que el atributo “subject” 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 audits asociados a ese auditor son correctos y que no aparecen otros los cuales pertenecen a otros auditor, además que los auditing records asociados a ese audit aparezca correctamente y no los de otro audit y sobre todo que se muestran las listas con los datos correctos.

Test Hacking

```
@Test
public void test300Hacking() {
    Collection<Audit> audits;
    String param;

    audits = this.repository.findManyAuditsByAuditorUsername("auditor1");
    for (final Audit audit : audits)
        if (audit.isDraftMode()) {
            param = String.format("id=%d", audit.getId());

            super.checkLinkExists("Sign in");
            super.request("/auditor/auditing-record/list", param);
            super.checkPanicExists();

            super.signIn("administrator", "administrator");
            super.request("/auditor/auditing-record/list", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("auditor2", "auditor2");
            super.request("/auditor/auditing-record/list", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("lecturer1", "lecturer1");
            super.request("/auditor/auditing-record/list", param);
            super.checkPanicExists();
            super.signOut();


            super.signIn("student1", "student1");
            super.request("/auditor/auditing-record/list", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("company1", "company1");
            super.request("/auditor/auditing-record/list", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("assistant1", "assistant1");
            super.request("/auditor/auditing-record/list", param);
            super.checkPanicExists();
            super.signOut();
        }
}
```

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

Show Auditing Record

Test Positivo

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/auditing-record/show-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int auditIndex, final String code, final int auditingRecordRecordIndex,
    final String subject, final String assessment, final String markType, final String initialPeriod,
    final String finalPeriod, final String link) {

    super.signIn("auditor1", "auditor1");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");
    super.clickOnListingRecord(auditIndex);
    super.clickOnButton("Audit Records");
    super.checkListingExists();
    super.clickOnListingRecord(auditingRecordRecordIndex);
    super.checkFormExists();

    super.checkInputBoxHasValue("subject", subject);
    super.checkInputBoxHasValue("assessment", assessment);
    super.checkInputBoxHasValue("markType", markType);
    super.checkInputBoxHasValue("initialPeriod", initialPeriod);
    super.checkInputBoxHasValue("finalPeriod", finalPeriod);
    super.checkInputBoxHasValue("link", link);

    super.signOut();
}
```

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

Tras esto, se accede al formulario de cada audit y se pulsará el botón Audit Records para mostrar la lista de auditing records asociados al audit(se comprueba que aparece) y por último se pulsará sobre cada auditing record 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 audits asociados a ese auditor son correctos y que no aparecen otros, los cuales, pertenecen a otros auditor, además que los auditing records asociados a ese audit aparezca correctamente y no los de otro audit y sobre todo que se muestran las listas y los formularios asociados con los datos correctos según las reglas de negocio impuestas.

Test Hacking

```
@Test
public void test300Hacking() {

    Collection<AuditingRecord> auditingRecords;
    String param;

    super.signIn("auditor2", "auditor2");
    auditingRecords = this.repository.findManyAuditingRecordByAuditorUsername("auditor2");
    super.signOut();
    for (final AuditingRecord auditingRecord : auditingRecords)
        if (auditingRecord.getAudit().isDraftMode()) {
            param = String.format("id=%d", auditingRecord.getAudit().getId());

            super.checkLinkExists("Sign in");
            super.request("/auditor/auditing-record/show", param);
            super.checkPanicExists();

            super.signIn("administrator", "administrator");
            super.request("/auditor/auditing-record/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("auditor1", "auditor1");
            super.request("/auditor/auditing-record/show", param);
            super.checkPanicExists();
            super.signOut();
```

```
            super.signIn("lecturer1", "lecturer1");
            super.request("/auditor/auditing-record/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("student1", "student1");
            super.request("/auditor/auditing-record/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("company1", "company1");
            super.request("/auditor/auditing-record/show", param);
            super.checkPanicExists();
            super.signOut();

            super.signIn("assistant1", "assistant1");
            super.request("/auditor/auditing-record/show", param);
            super.checkPanicExists();
            super.signOut();
        }
    }
```

En este test, lo que hacemos es ir probando que ninguno de los demás roles y los demás auditor, no pueden acceder a los datos de los auditing records y que devuelve un error 500 indicando que no estamos autorizados.

Create Auditing Records

Test Positivo

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/auditing-record/create-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int auditRecordIndex, final int auditingRecordRecordIndex,
    final String subject, final String assessment, final String markType, final String initialPeriod,
    final String finalPeriod, final String link) {

    super.signIn("auditor15", "auditor15");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(auditRecordIndex);
    super.clickOnButton("Audit Records");

    super.clickOnButton("Create");
    super.fillInputBoxIn("subject", subject);
    super.fillInputBoxIn("assessment", assessment);
    super.fillInputBoxIn("markType", markType);
    super.fillInputBoxIn("initialPeriod", initialPeriod);
    super.fillInputBoxIn("finalPeriod", finalPeriod);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Create");

    super.checkListingExists();
    super.sortListing(0, "asc");
    super.checkColumnHasValue(auditingRecordRecordIndex, 0, subject);


    super.clickOnListingRecord(auditingRecordRecordIndex);
    super.checkInputBoxHasValue("subject", subject);
    super.checkInputBoxHasValue("assessment", assessment);
    super.checkInputBoxHasValue("markType", markType);
    super.checkInputBoxHasValue("initialPeriod", initialPeriod);
    super.checkInputBoxHasValue("finalPeriod", finalPeriod);
    super.checkInputBoxHasValue("link", link);
    super.signOut();
}
```

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

Tras esto, se accede al formulario de cada audit y se pulsará el botón Audit Records para mostrar la lista de auditing records asociados al audit, después se pulsará el botón de Create y nos saldrá el formulario de creación de un nuevo auditing record, se añadirá los valores correctos, con la reglas de negocio, en cada campo y se pulsará el botón de Create.

Tras esto, se comprueba que el listado de auditing record se muestra correctamente con los valores asociados y finalmente se va accediendo a cada auditing record comprobando que los datos en cada atributo sean los metidos a la hora de haberlos creado.

El ejecutar los test de esta manera nos permite detectar que los audits asociados a ese auditor son correctos y que no aparecen otros, los cuales, pertenecen a otros auditor, además que los auditing records asociados a ese audit aparezcan correctamente y no los de otro audit, y sobre todo que se crean los auditing records con valores que cumplen las reglas de negocio y que al crearse se muestran las listas y los formularios asociados con los datos correctos.

Test Negativo

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/auditing-record/create-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int auditRecordIndex, final int auditingRecordRecordIndex,
    final String subject, final String assessment, final String markType, final String initialPeriod,
    final String finalPeriod, final String link) {

    super.signIn("auditor15", "auditor15");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(auditRecordIndex);
    super.clickOnButton("Audit Records");

    super.clickOnButton("Create");
    super.fillInputBoxIn("subject", subject);
    super.fillInputBoxIn("assessment", assessment);
    super.fillInputBoxIn("markType", markType);
    super.fillInputBoxIn("initialPeriod", initialPeriod);
    super.fillInputBoxIn("finalPeriod", finalPeriod);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Create");
    super.checkErrorsExist();

    super.signOut();
}
```

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

Tras esto, se accede al formulario de cada audit y se pulsará el botón Audit Records para mostrar la lista de auditing records asociados al audit, después se pulsará el botón de Create y nos saldrá el formulario de creación de un nuevo auditing record, se añadirá los valores erróneos de acuerdo con las reglas de negocio en cada campo de forma escalonada para que no salte varios errores a la vez y podamos distinguir que ha fallado, finalmente se pulsará el botón de Create y saltará una error.

El ejecutar los test de esta manera nos permite detectar que los audits asociados a ese auditor son correctos y que no aparecen otros, los cuales, pertenecen a otros auditor, además que los auditing records asociados a ese audit aparezcan correctamente y no los de otro audit, y sobre todo que no se crean los auditing record con valores que no cumplen las reglas de negocio, saltando excepciones en cada uno de ellos.

Test Hacking

```
@Test
public void test300Hacking() {

    final Collection<Audit> audits;
    String param;

    audits = this.repository.findManyAuditsByAuditorUsername("auditor1");
    for (final Audit audit : audits) {
        param = String.format("auditId=%d", audit.getId());

        super.checkLinkExists("Sign in");
        super.request("/auditor/auditing-record/create", param);
        super.checkPanicExists();

        super.signIn("administrator", "administrator");
        super.request("/auditor/auditing-record/create", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("lecturer1", "lecturer1");
        super.request("/auditor/auditing-record/create", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("student1", "student1");
        super.request("/auditor/auditing-record/create", param);
        super.checkPanicExists();
        super.signOut();


          super.signIn("company1", "company1");
          super.request("/auditor/auditing-record/create", param);
          super.checkPanicExists();
          super.signOut();

          super.signIn("assistant1", "assistant1");
          super.request("/auditor/auditing-record/create", param);
          super.checkPanicExists();
          super.signOut();
    }
}
```

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

Test Hacking

```
@Test
public void test301Hacking() {

    final Collection<Audit> audits;
    String param;

    super.checkLinkExists("Sign in");
    super.signIn("auditor14", "auditor14");
    audits = this.repository.findManyAuditsByAuditorUsername("auditor14");
    for (final Audit audit : audits)
        if (!audit.isDraftMode()) {
            param = String.format("auditId=%d", audit.getId());
            super.request("/auditor/auditing-record/create", param);
            super.checkPanicExists();
        }
}
```

En este test, lo que hacemos es comprobar que no pueden crear un auditing record asociado a un audit 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() {

    final Collection<Audit> audits;
    String param;

    super.checkLinkExists("Sign in");
    super.signIn("auditor11", "auditor11");
    audits = this.repository.findManyAuditsByAuditorUsername("auditor14");
    for (final Audit audit : audits) {
        param = String.format("auditId=%d", audit.getId());
        super.request("/auditor/auditing-record/create", param);
        super.checkPanicExists();
    }
}
```

En este test, lo que hacemos es comprobar que no pueden crear un auditing record asociado a un audit que no pertenece al auditor asociado y que devuelve un error 500 indicando que no estamos autorizados, que es lo que estamos buscando.

Create Auditing Records Exception

Test Positivo

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/auditing-record/create-exception-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int auditRecordIndex, final int auditingRecordRecordIndex,
    final String subject, final String assessment, final String markType, final String initialPeriod,
    final String finalPeriod, final String link) {

    super.signIn("auditor16", "auditor16");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(auditRecordIndex);
    super.clickOnButton("Audit Records");

    super.clickOnButton("Create Exceptional");
    super.fillInputBoxIn("subject", subject);
    super.fillInputBoxIn("assessment", assessment);
    super.fillInputBoxIn("markType", markType);
    super.fillInputBoxIn("initialPeriod", initialPeriod);
    super.fillInputBoxIn("finalPeriod", finalPeriod);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Create Exception");

    super.checkListingExists();
    super.sortListing(0, "asc");
    super.checkColumnHasValue(auditingRecordRecordIndex, 0, subject);


     super.clickOnListingRecord(auditingRecordRecordIndex);
     super.checkInputBoxHasValue("subject", subject);
     super.checkInputBoxHasValue("assessment", assessment);
     super.checkInputBoxHasValue("markType", markType);
     super.checkInputBoxHasValue("initialPeriod", initialPeriod);
     super.checkInputBoxHasValue("finalPeriod", finalPeriod);
     super.checkInputBoxHasValue("link", link);
     super.signOut();
 }
```

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

Tras esto, se accede al formulario de cada audit y se pulsará el botón Audit Records para mostrar la lista de auditing records asociados al audit, después se pulsará el botón de Create Exceptional y nos saldrá el formulario de creación de un nuevo auditing record exceptional, se añadirá los valores correctos, con la reglas de negocio, en cada campo y se pulsará el botón de Create Exception.

Tras esto, se comprueba que el listado de auditing record se muestra correctamente con los valores asociados y se marca como exception y finalmente se va accediendo a cada auditing record comprobando que los datos en cada atributo sean los metidos a la hora de haberlos creado.

El ejecutar los test de esta manera nos permite detectar que los audits asociados a ese auditor son correctos y que no aparecen otros, los cuales, pertenecen a otros auditor, además que los auditing records asociados a ese audit aparezcan correctamente y no los de otro audit, y sobre todo que se crean los auditing records con valores que cumplen las reglas de negocio y que al crearse se muestran las listas y los formularios asociados con los datos correctos.

Test Negativo

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/auditing-record/create-exception-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int auditRecordIndex, final int auditingRecordRecordIndex,
    final String subject, final String assessment, final String markType, final String initialPeriod,
    final String finalPeriod, final String link) {

    super.signIn("auditor16", "auditor16");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(auditRecordIndex);
    super.clickOnButton("Audit Records");

    super.clickOnButton("Create Exceptional");
    super.fillInputBoxIn("subject", subject);
    super.fillInputBoxIn("assessment", assessment);
    super.fillInputBoxIn("markType", markType);
    super.fillInputBoxIn("initialPeriod", initialPeriod);
    super.fillInputBoxIn("finalPeriod", finalPeriod);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Create Exception");
    super.checkErrorsExist();

    super.signOut();
}
```

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

Tras esto, se accede al formulario de cada audit y se pulsará el botón Audit Records para mostrar la lista de auditing records asociados al audit, después se pulsará el botón de Create Exceptional y nos saldrá el formulario de creación de un nuevo auditing record exceptional, se añadirá los valores erróneos de acuerdo con las reglas de negocio en cada campo de forma escalonada para que no salte varios errores a la vez y podamos distinguir que ha fallado, finalmente se pulsará el botón de Create Exception y saltará una error.

El ejecutar los test de esta manera nos permite detectar que los audits asociados a ese auditor son correctos y que no aparecen otros, los cuales, pertenecen a otros auditor, además que los auditing records asociados a ese audit aparezcan correctamente y no los de otro audit, y sobre todo que no se crean los auditing record con valores que no cumplen las reglas de negocio, saltando excepciones en cada uno de ellos.

Test Hacking

```
@Test
public void test300Hacking() {

    final Collection<Audit> audits;
    String param;

    audits = this.repository.findManyAuditsByAuditorUsername("auditor1");
    for (final Audit audit : audits) {
        param = String.format("auditId=%d", audit.getId());

        super.checkLinkExists("Sign in");
        super.request("/auditor/auditing-record/create-exceptional", param);
        super.checkPanicExists();

        super.signIn("administrator", "administrator");
        super.request("/auditor/auditing-record/create-exceptional", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("lecturer1", "lecturer1");
        super.request("/auditor/auditing-record/create-exceptional", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("student1", "student1");
        super.request("/auditor/auditing-record/create-exceptional", param);
        super.checkPanicExists();
        super.signOut();


        super.signIn("company1", "company1");
        super.request("/auditor/auditing-record/create-exceptional", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("assistant1", "assistant1");
        super.request("/auditor/auditing-record/create-exceptional", param);
        super.checkPanicExists();
        super.signOut();
    }
}
```

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

Test Hacking

```
@Test
public void test301Hacking() {

    final Collection<Audit> audits;
    String param;

    super.checkLinkExists("Sign in");
    super.signIn("auditor15", "auditor15");
    audits = this.repository.findManyAuditsByAuditorUsername("auditor15");
    for (final Audit audit : audits)
        if (audit.isDraftMode()) {
            param = String.format("auditId=%d", audit.getId());
            super.request("/auditor/auditing-record/create-exceptional", param);
            super.checkPanicExists();
        }
}
```

En este test, lo que hacemos es comprobar que no pueden crear un auditing record exceptional asociado a un audit que no haya sido publicado y que devuelve un error 500 indicando que no estamos autorizados, que es lo que estamos buscando.

Test Hacking

```
@Test
public void test302Hacking() {

    final Collection<Audit> audits;
    String param;

    super.checkLinkExists("Sign in");
    super.signIn("auditor11", "auditor11");
    audits = this.repository.findManyAuditsByAuditorUsername("auditor16");
    for (final Audit audit : audits) {
        param = String.format("auditId=%d", audit.getId());
        super.request("/auditor/auditing-record/create-exceptional", param);
        super.checkPanicExists();
    }
}
```

En este test, lo que hacemos es comprobar que no pueden crear un auditing record exceptional asociado a un audit que no pertenece al auditor asociado y que devuelve un error 500 indicando que no estamos autorizados, que es lo que estamos buscando.

Update Auditing Record

Test Positivo

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/auditing-record/update-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test100Positive(final int auditRecordIndex, final int auditingRecordRecordIndex,
    final String subject, final String assessment, final String markType, final String initialPeriod,
    final String finalPeriod, final String link) {

    super.signIn("auditor17", "auditor17");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(auditRecordIndex);
    super.clickOnButton("Audit Records");

    super.clickOnListingRecord(auditingRecordRecordIndex);
    super.checkFormExists();

    super.fillInputBoxIn("subject", subject);
    super.fillInputBoxIn("assessment", assessment);
    super.fillInputBoxIn("markType", markType);
    super.fillInputBoxIn("initialPeriod", initialPeriod);
    super.fillInputBoxIn("finalPeriod", finalPeriod);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Update");

    super.checkListingExists();
    super.sortListing(0, "asc");
    super.checkColumnHasValue(auditingRecordRecordIndex, 0, subject);

     super.clickOnListingRecord(auditingRecordRecordIndex);
     super.checkFormExists();
     super.checkInputBoxHasValue("subject", subject);
     super.checkInputBoxHasValue("assessment", assessment);
     super.checkInputBoxHasValue("markType", markType);
     super.checkInputBoxHasValue("initialPeriod", initialPeriod);
     super.checkInputBoxHasValue("finalPeriod", finalPeriod);
     super.checkInputBoxHasValue("link", link);

     super.signOut();
}
```

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

Tras esto, se accede al formulario de cada audit y se pulsará el botón Audit Records para mostrar la lista de auditing records asociados al audit(se comprueba que aparece), después se pulsará sobre cada auditing record, nos aparece su formulario con los datos asociados a cada campo, se añadirá los valores correctos, con la reglas de negocio, en cada campo y se pulsará el botón de Update.

Tras esto, se comprueba que el listado de auditing record se muestra correctamente con los valores asociados y finalmente se va accediendo a cada auditing record comprobando que los datos en cada atributo sean los metidos a la hora de haberlos actualizarlo.

El ejecutar los test de esta manera nos permite detectar que los audits asociados a ese auditor son correctos y que no aparecen otros, los cuales, pertenecen a otros auditor, además que los auditing records asociados a ese audit aparezcan correctamente y no los de otro audit , y sobre todo que se actualizan los auditing record con valores que cumplen las reglas de negocio y que al actualizarse se muestran las listas y los formularios asociados con los datos correctos.

Test Negative

```
@ParameterizedTest
@CsvFileSource(resources = "/auditor/auditing-record/update-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
public void test200Negative(final int auditRecordIndex, final int auditingRecordRecordIndex, final String subject,
    final String assessment, final String markType, final String initialPeriod, final String finalPeriod, final String link) {

    super.signIn("auditor17", "auditor17");

    super.clickOnMenu("Auditor", "Audit List");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.clickOnListingRecord(auditRecordIndex);
    super.clickOnButton("Audit Records");

    super.clickOnListingRecord(auditingRecordRecordIndex);
    super.checkFormExists();

    super.fillInputBoxIn("subject", subject);
    super.fillInputBoxIn("assessment", assessment);
    super.fillInputBoxIn("markType", markType);
    super.fillInputBoxIn("initialPeriod", initialPeriod);
    super.fillInputBoxIn("finalPeriod", finalPeriod);
    super.fillInputBoxIn("link", link);

    super.clickOnSubmit("Update");

    super.checkErrorsExist();

    super.signOut();
}
```

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

Tras esto, se accede al formulario de cada audit y se pulsará el botón Audit Records para mostrar la lista de auditing records asociados al audit, después se pulsará sobre cada auditing record, nos aparece su formulario con los datos asociados a cada campo, se añadirá valores incorrectos de acuerdo con las reglas de negocio, en cada campo y se pulsará el botón de Update.

El ejecutar los test de esta manera nos permite detectar que los audits asociados a ese auditor son correctos y que no aparecen otros, los cuales, pertenecen a otros auditors, además que los auditing records asociados a ese audit aparezcan correctamente y no los de otro audit , y sobre todo que no se actualizan los auditing records con valores que no cumplen las reglas de negocio, saltando excepciones en cada uno de ellos.

Test Hacking

```
@Test
public void test300Hacking() {
    // HINT: this test tries to update a auditing record with a role other than "Auditor",
    // HINT+ or using an auditor who is not the owner.

    final Collection<Audit> audits;
    String param;

    audits = this.repository.findManyAuditsByAuditorUsername("auditor1");
    for (final Audit audit : audits) {
        param = String.format("id=%d", audit.getId());

        super.checkLinkExists("Sign in");
        super.request("/auditor/auditing-record/update", param);
        super.checkPanicExists();

        super.signIn("administrator", "administrator");
        super.request("/auditor/auditing-record/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("auditor2", "auditor2");
        super.request("/auditor/auditing-record/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("lecturer1", "lecturer1");
        super.request("/auditor/auditing-record/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("student1", "student1");
        super.request("/auditor/auditing-record/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("company1", "company1");
        super.request("/auditor/auditing-record/update", param);
        super.checkPanicExists();
        super.signOut();

        super.signIn("assistant1", "assistant1");
        super.request("/auditor/auditing-record/update", param);
        super.checkPanicExists();
        super.signOut();
    }
}
```

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

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.

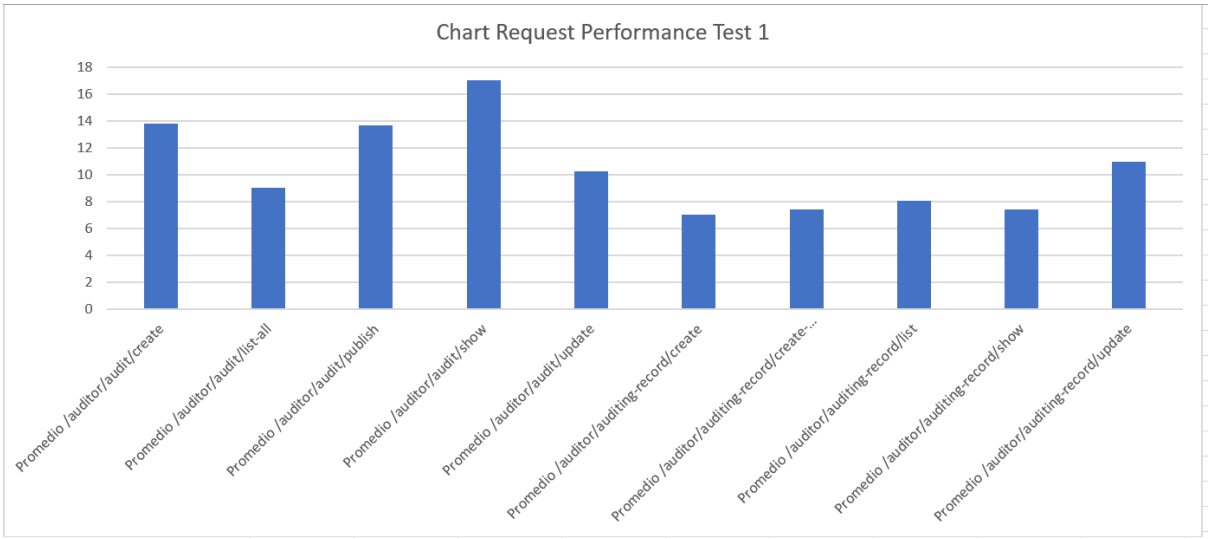

Gráfico del promedio del tiempo de las solicitudes - PC 1

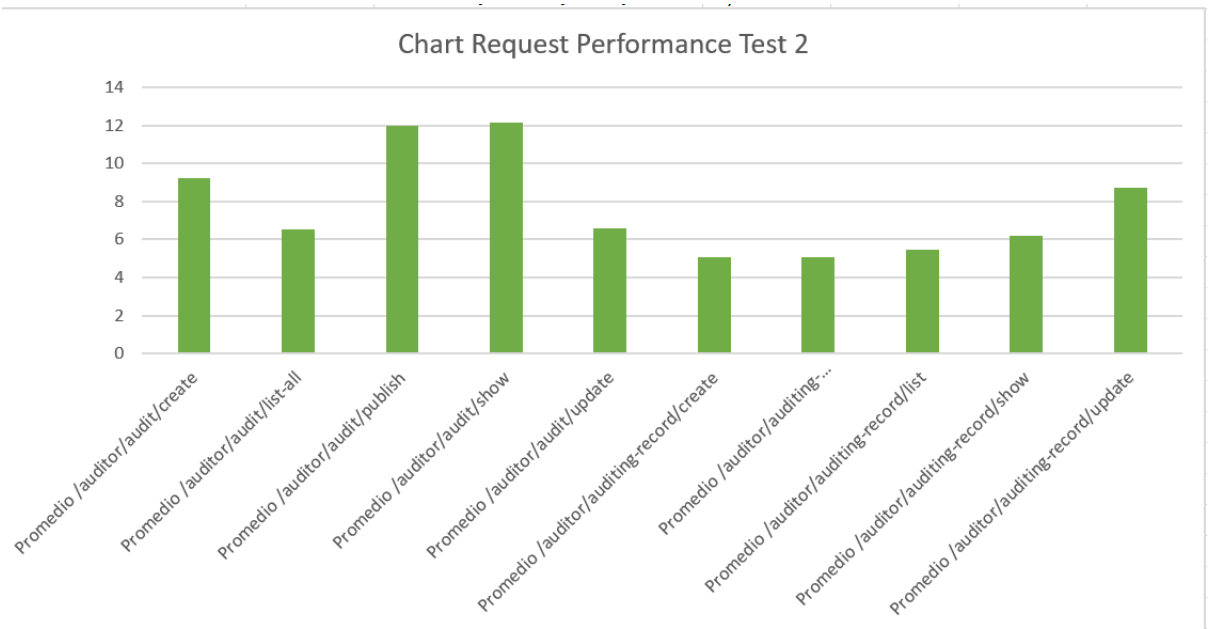

Gráfico del promedio del tiempo de las solicitudes - PC 2

➔ 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.

Result PC 1			*Result PC 2*		
Media	10,44415584		Media	7,43636364	
Error típico	0,332556949		Error típico	0,1861636	
Mediana	9		Mediana	7	
Moda	7		Moda	4	
Desviación estándar	9,228080816		Desviación estándar	5,16583024	
Varianza de la muestra	85,15747555		Varianza de la muestra	26,6858021	
Curtosis	129,9732831		Curtosis	43,818566	
Coeficiente de asimetría	8,967863633		Coeficiente de asimetría	4,5709877	
Rango	167		Rango	68	
Mínimo	1		Mínimo	1	
Máximo	168		Máximo	69	
Suma	8042		Suma	5726	
Cuenta	770		Cuenta	770	
Nivel de confianza(95,0%)	0,652827129		Nivel de confianza(95,0%)	0,36544913	
Interval(ms)	9,791328715	11,096983	Interval(ms)	7,0709145	7,80181277
Interval(s)	0,009791329	0,01109698	Interval(s)	0,00707091	0,00780181

➔ 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.

Prueba z para medias de dos muestras		
	Before	*After*
Media	10,4441558	7,43636364
Varianza (conocida)	85,1574756	26,6858021
Observaciones	770	770
Diferencia hipotética de las medias	0	
z	7,89202131	
P(Z<=z) una cola	1,4433E-15	
Valor crítico de z (una cola)	1,64485363	
P(Z<=z) dos colas	2,8866E-15	
Valor crítico de z (dos colas)	1,95996398	

Ya que observamos que el P(Z<=z) dos colas de z se encuentra entre 0 y alpha(0,05 en este caso) podemos comparar los promedios del tiempo ya que se produjo un cambio, y en este caso positivo ya que la media de tiempo después es menor que la media antes.

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.

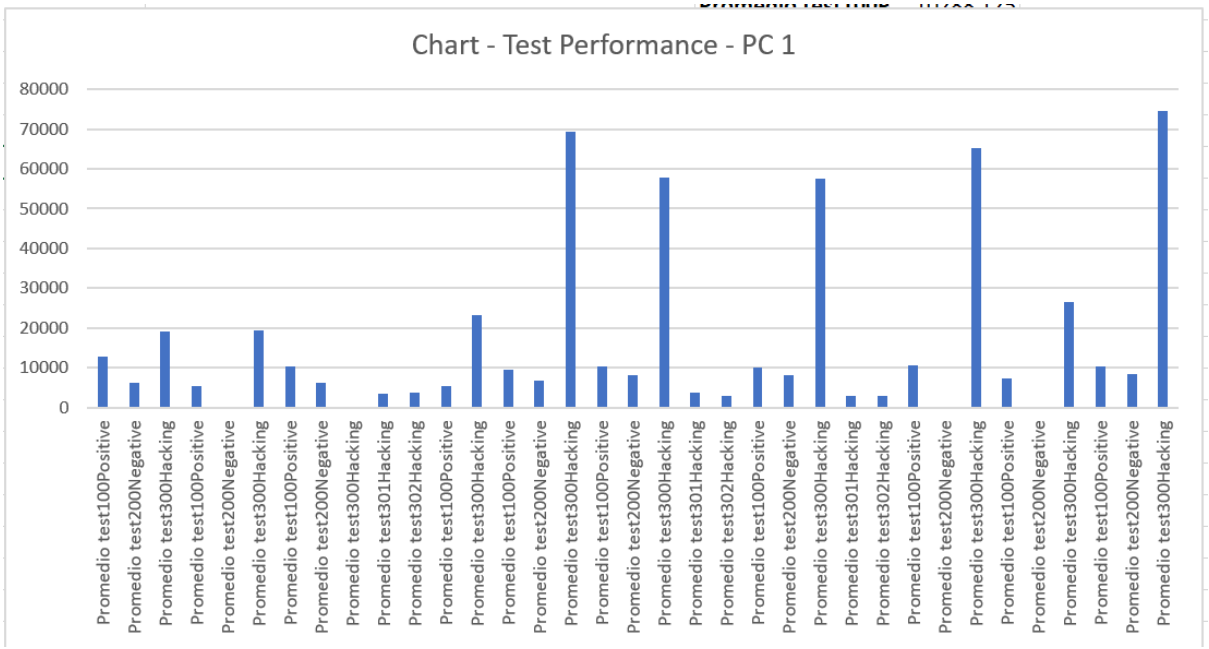

Gráfico del tiempo promedio de ejecución de los tests (de 0 a 75 segundos) - PC1

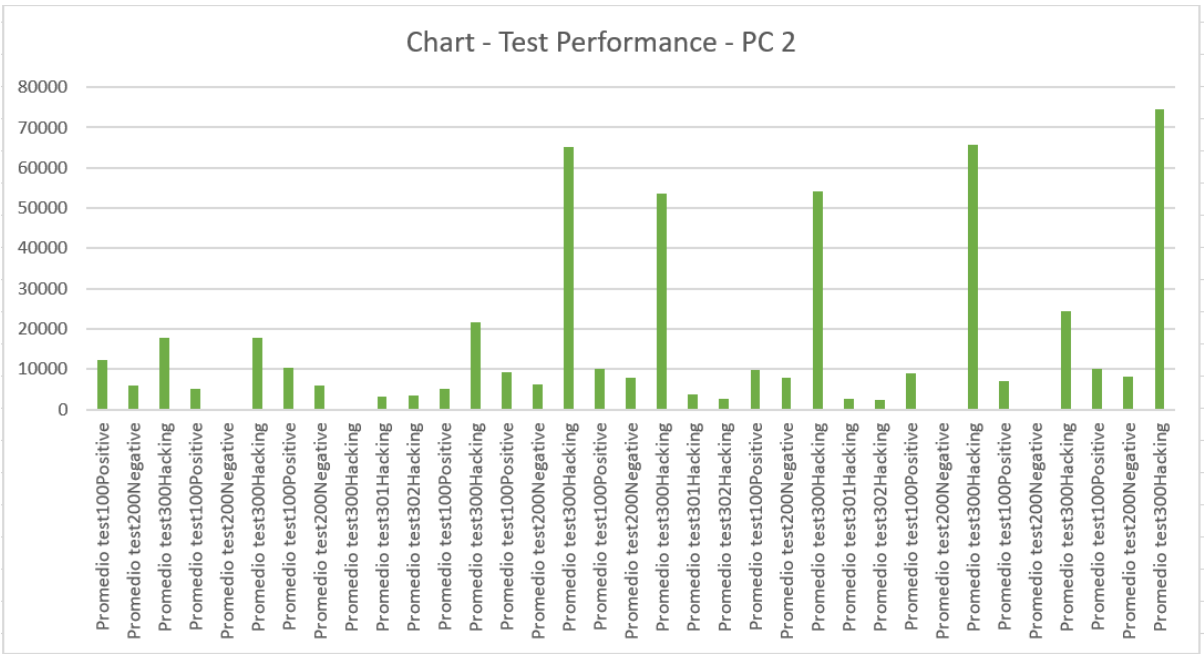

Gráfico del tiempo promedio de ejecución de los tests (de 0 a 73 segundos) - PC2

8 BIBLIOGRAFÍA

Intencionalmente en blanco.