# Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática


Grado en Ingeniería Informática del Software

**Diseño y Pruebas II**

Curso 2021/2022

# TESTING LEARNING REPORT

**Repositorio**: https://github.com/Ginpasfer/Acme-Recipes

| **Grupo de Prácticas** | *S07* |
|---|---|
| **Estudiantes** | **Rol** |
| Pastor Fernández, Ginés | Project Manager<br>Developer<br>Operator<br>Tester |
| Giráldez Álvarez, Pablo | Developer<br>Analyst<br>Tester |
| Rijo Hernández, Badayco | Developer<br>Tester |
| Solís Miranda, Antonio Manuel | Developer<br>Tester |
| Paradas Borrego, Álvaro | Developer<br>Tester |

# Índice

# 1. Resumen ejecutivo

En el presente documento se va a exponer los conocimientos adquiridos respecto a la realización de pruebas de una WIS durante el desarrollo del proyecto **Acme-Recipes**. Se expondrá el aprendizaje adquirido durante la implementación de las pruebas de software, así como los aspectos positivos y negativos que se han observado a la hora de su realización.

# 2. Tabla de revisiones

| Versión | Fecha | Autor | Descripción de cambios |
|---|---|---|---|
| 1.0 | 03/09/2022 | Badayco Rijo Hernández | - Creación del documento |
| 2.0 | 06/09/2022 | Badayco Rijo Hernández | - Añadido más contenido |

# 3. Introducción

Primeramente, se explicarán los distintos aprendizajes adquiridos a lo largo del proyecto, concretando sobre la toma de contacto por parte de los integrantes del *Modo Marioneta* y el método de trabajo con archivos CSV.

En el primer entregable del proyecto, el grupo de trabajo debía indicar cuáles eran los conocimientos sobre la realización de pruebas software, las cuales tienen el objetivo de comprobar el estado del sistema y sus funcionalidades, verificando que todo funciona como se espera. En el documento de dicha entrega se hizo referencia a distintos tipos de pruebas. En primer lugar, se explicó sobre las *pruebas unitarias*, donde se comentó que dichas pruebas comprueban una parte pequeña y aislada del código, es decir, el objetivo era verificar que una funcionalidad operaba correctamente, tienen la granularidad más baja de todas las pruebas y corren de forma muy rápida. Se mencionó su estructura, sus características y buenas prácticas para su correcto desarrollo. Aparte de esto, se comentaron otros tipos de pruebas como los *dobles*, los *stubs* y el framework para pruebas "Mockito".

Durante la realización del proyecto de esta asignatura, se han realizado pruebas de software, cuyo enfoque ha estado principalmente en las pruebas unitarias, implementando durante cada entrega tests positivos y negativos para comprobar las funcionalidades desarrolladas. A continuación, en el apartado de **Contenido** se expondrá el aprendizaje adquirido por parte del equipo durante su implementación, así como los aspectos positivos y problemas encontrados. Por último, se concluirá sobre la realización de los tests durante el proyecto.

# 4. Contenido

Durante el desarrollo de la asignatura el equipo ha aprendido a realizar testing *E2E*. La prueba de extremo a extremo (E2E) es una metodología que evalúa el estado de funcionamiento de un producto complejo en un proceso de principio a fin. Las pruebas de extremo a extremo verifican que todos los componentes de un sistema puedan ejecutarse y funcionar de manera óptima en escenarios del mundo real.

Para el desarrollo de las pruebas se ha hecho uso de diversas herramientas, entre ellas, *Mozilla Firefox* como navegador y *Geckodriver* para automatizar estas.

- **Modo Marioneta (Marionette)**

Todos los integrantes del grupo no tenían conocimiento de la existencia de la funcionalidad *Modo Marioneta* de Firefox, la cual resulta muy útil para poder visualizar cómo se realizan los tests y, en caso de fallo, tener más claro el porqué y el dónde de dicho fallo, agilizando el proceso de solución en las pruebas.

Este modo marioneta también permite probar de forma más precisa la aplicación final que va a recibir el usuario, a coste de un mayor tiempo de testeo.

El equipo ha trabajado con el modo marioneta realizando test que extienden de una clase del framework llamada "TestHarness", que, a su vez, extiende de un "AbstractController" que interactúa con *GeckoDriver* para manipular el comportamiento de Firefox durante las pruebas.

- **Datos y métodos de prueba**

El equipo de trabajo también ha aprendido a realizar las pruebas con diferentes archivos **CSV** en función de qué tipo de método de prueba se trate (métodos positivos o negativos, métodos de crear,

mostrar, borrar...).
Estos datos los recibirá el test como cadenas de texto que usará para interactuar con la página, ya sea creando un elemento nuevo, editándolo...
Finalmente, realiza una comprobación/comparación de la información que se encuentra en el fichero *CSV* con la información introducida en la página.
En el caso de probar el *show* y *list*, el primer paso antes citado no se realiza, la clase de test solamente se encargaría de comprobar que se ha mostrado o listado correctamente, según corresponda.
La **ilustración 1** se corresponde con un fichero CSV para el caso de la prueba de software positiva para la funcionalidad de *Peep* **(ilustración 2***).* Como se puede observar el fichero CSV se compone de un recorIndex y los atributos para el test correspondiente.

| recordIndex | heading | writer | text | email | confirmation |
|---|---|---|---|---|---|
| 0 | This is a heading | Pedro | AA this is a new | pedro@gmail.co | true |
| 1 | Other heading | Jose | BB Other text | jose@gmail.com | true |
| 2 | This is other | Antonio | CC this is new | antonio@gmail.c | true |
| 3 | Test of Heading | Manuel | DD This is a piece | manuel@gmail.c | true |
| 4 | Heading5 | Juan | EE this is a short t | | true |
| 5 | Other heading | Jose | FF Other texting | jose@gmail.com | true |
| 6 | Another one | Antonio | GG this is a short | antonio@gmail.c | true |
| 7 | IMPORTANT | Manuel | HH This is a piec | manuel@gmail.c | true |
| 8 | Heading | Juan | II Testing | | true |
| 9 | My new heading | Manuel | JJ This is a piece | manuel@gmail.c | true |
| 10 | New title | Juan | KK New descripti | | true |

*Ilustración 1: CSV de prueba positiva de la funcionalidad "peep"*

```
@ParameterizedTest
@CsvFileSource(resources = "/any/peep/create-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void testPositive(final int recordIndex, final String heading, final String writer,
    final String text, final String email, final String confirmation) {
    super.clickOnMenu("Menu","Peeps list");
    super.checkListingExists();

    super.clickOnButton("Create");
    super.fillInputBoxIn("heading", heading);
    super.fillInputBoxIn("writer", writer);
    super.fillInputBoxIn("text", text);
    super.fillInputBoxIn("email", email);
    super.fillInputBoxIn("confirmation", confirmation);
    super.clickOnSubmit("Create");

    super.clickOnMenu("Menu","Peeps list");
    super.checkListingExists();

    super.sortListing(3, "asc");
    super.checkColumnHasValue(recordIndex, 1, heading);
    super.checkColumnHasValue(recordIndex, 2, writer);
    super.checkColumnHasValue(recordIndex, 3, text);
    super.checkColumnHasValue(recordIndex, 4, email);
}
```

*Ilustración 2: Prueba de software positiva de la funcionalidad de "peep"*

En la **ilustración 3** se puede observar el test negativo correspondiente a la funcionalidad de peep. Esta prueba es prácticamente similar a la positiva, salvo que el fichero CSV cambia ciertos valores de los atributos (**ilustración 4**). Para el fichero CSV negativo se han eliminado valores como el título, la descripción, la confirmación se ha definido como “false” en algunos casos, … Todo ello se

ha realizado para comprobar que aparecen errores esperados.

```
@ParameterizedTest
@CsvFileSource(resources = "/any/peep/create-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(20)
public void testNegative(final int recordIndex, final String heading, final String writer, final
    String text, final String email, final String confirmation) {
    super.clickOnMenu("Menu","Peeps list");

    super.checkListingExists();

    super.clickOnButton("Create");
    super.fillInputBoxIn("heading", heading);
    super.fillInputBoxIn("writer", writer);
    super.fillInputBoxIn("text", text);
    super.fillInputBoxIn("email", email);
    super.fillInputBoxIn("confirmation", confirmation);
    super.clickOnSubmit("Create");

    super.checkErrorsExist();
}
```

*Ilustración 3: Prueba de software negativa para la funcionalidad "peep"*

| recordIndex | heading | writer | text | email | confirmation |
|---|---|---|---|---|---|
| 0 | sexo | | sexo | | true |
| 1 | | Jose | | jose@gmail.com | true |
| 2 | This is other | hard core | CC this is new | antonio@gmail.c | false |
| 3 | Test of Heading | | DD This is a piece | manuel@gmail.c | true |
| 4 | | Juan | sex | | true |
| 5 | Other heading | | | jose@gmail.com | true |
| 6 | | Antonio | GG this is a short | antonio@gmail.c | false |
| 7 | sex | Manuel | HH This is a piec | manuel@gmail.c | false |
| 8 | | Juan | II Testing | | false |
| 9 | My new heading | Manuel | | manuel@gmail.c | true |
| 10 | hard core | hard core sex | hard core | | true |
| | | | | | |
| | | | | | |

*Ilustración 4: CSV de prueba negativa de la funcionalidad "peep"*

Además, el equipo desconocía algunos métodos necesarios en la asignatura, como es el caso de algunos métodos de “hacking”, que refuerzan la aplicación probando el acceso de distintas autoridades en las páginas existentes. Esto normalmente se realiza navegando hacia *URLs* a las cuales no debería de tener permisos o bien por su autoridad, lo cual, suele encargarse el framework, o bien por otros casos de los cuales se ha encargado el grupo. Se puede observar la estructura de este tipo de prueba en la **ilustración 5.**

```
@Test
@Order(30)
public void hackingTestIngredientsList() {
    super.navigate("/chef/item/list-ingredients");
    super.checkPanicExists();

    super.signIn("administrator", "administrator");
    super.navigate("/chef/item/list-ingredients");
    super.checkPanicExists();
    super.signOut();

    super.signIn("epicure1", "epicure1");
    super.navigate("/chef/item/list-ingredients");
    super.checkPanicExists();
    super.signOut();
}

@Test
@Order(30)
public void hackingTestKitchenUtensilsList() {
    super.navigate("/chef/item/list-kitchen_utensils");
    super.checkPanicExists();

    super.signIn("administrator", "administrator");
    super.navigate("/chef/item/list-kitchen_utensils");
    super.checkPanicExists();
    super.signOut();

    super.signIn("epicure1", "epicure1");
    super.navigate("/chef/item/list-kitchen_utensils");
    super.checkPanicExists();
    super.signOut();
}
```

*Ilustración 5: Prueba de hacking*

Durante el transcurso del desarrollo, el equipo ha comprobado la importancia sobre la necesidad de desarrollar y ejecutar pruebas de software de forma paralela a la programación de código y no una vez finalizado la realización de las funcionalidades, porque de lo contrario, es posible que aparezcan errores y se convierta en una tarea más compleja descubrir la causa del fallo y se tarde mucho más tiempo en solucionarlos de lo que se tardaría si se realizasen a la misma vez. Así mismo, el equipo ha aprendido a parametrizar las pruebas, probando los máximos y mínimos posibles en cada atributo, evitando así posibles errores de validaciones en los distintos formularios. Otro aspecto del que el equipo se ha beneficiado y aprendido ha sido la creación de pruebas software negativas, ya que ha sido vital para comprobar que la aplicación respondía perfectamente cuando no se rellenaban correctamente los formularios o se intentaba acceder a funcionalidades sin la autoridad necesaria, devolviendo así los avisos de error esperados.

## 5. Conclusión

Para concluir, podemos observar que el equipo ha aprendido nuevas formas de probar el código que desarrollan, como es el caso de los tests automáticos que usan “marionette” para probar la interfaz de la WIS. Además, el grupo ha adquirido el conocimiento de realizar tests para las funcionalidades que se realizan, de esta forma el equipo se asegura completamente de su correcto funcionamiento.

El equipo considera que es necesario probar todas las funcionalidades del sistema, para, de esta manera, asegurarse de que todo es comprobado y verificando así que todo funciona y responde como se pide, y que no ocurra ningún error inesperado durante la ejecución de la aplicación.

## 6. Bibliografía

Intencionalmente en blanco.