DP2 2021-2022

Informe de Pruebas

Proyecto Acme Toolkits

https://github.com/Alex-GF/Acme-Toolkits.git

Miembros:

- VICENTE CAMBRON TOCADOS (viccamtoc@alum.us.es)
- FRANCISCO JAVIER CAVERO LOPEZ (fracavlop@alum.us.es)
- IRENE XIANG DOMINGUEZ GAROZ (iredomgar4@alum.us.es)
- ALEJANDRO GARCIA FERNANDEZ (alegarfer4@alum.us.es)
- JOSE LUIS GARCIA MARIN (josgarmar31@alum.us.es)
- MARTA REYES LÓPEZ (marreylop4@alum.us.es)

GRUPO E1-03

Versión 1.0

02/06/22

Índice

Resumen ejecutivo

El objetivo de este documento es presentar y detallar los conocimientos obtenidos de pruebas de software durante el desarrollo del proyecto.

Tabla de revisiones

Fecha	Versión	Descripción	Sprint
02/06/2022	1.0	Versión inicial	1

Introducción

El objetivo de este documento es describir el proceso de pruebas llevado a cabo durante el desarrollo de este proyecto. Este proceso conlleva la verificación y validación de las distintas funcionalidades incluidas, consiguiendo así, la identificación y corrección de errores y bugs existentes.

Gracias a esto podemos conseguir una calidad del software satisfactoria, asegurando al usuario final un correcto funcionamiento de la aplicación.

Además, se describirá los distintos conocimientos que el equipo ha desarrollado durante la ejecución de las pruebas.

El documento está estructurado de tal forma que primero observaremos una sección “Contenido”, donde se mencionará los distintos conceptos que hemos aprendido en las clases de teoría, los tipos de pruebas que se realizan despendiendo de las funcionalidades implementadas y, por último, las distintas pruebas implementadas en nuestra aplicación. Finalmente, habrá un apartado de “Conclusiones”, donde se aportará una última recapitulación de todo lo expuesto en el apartado anterior.

Contenido

Conceptos teóricos previos

A continuación, se van a describir cuales son los conocimientos que hemos obtenido en la asignatura que están relacionados de manera directa o indirecta con las pruebas.

Primero, se establecen los requisitos sobre lo que nuestro cliente desea que implementemos con respecto a los datos. Además, se deberá tener en cuenta la funcionalidad y la no funcionalidad (algunos requisitos estarán implícitos) y algunos requisitos de gestión.

Entendemos una funcionalidad como grupo de requisitos que constituyen una unidad lógica.

Por otra parte, un bug es algo mal en el código desarrollado. Por ejemplo, si se escribió el operador “menos que” en lugar del operador “menor o igual”. El objetivo de las pruebas funcionales es encontrar tantos errores como sea posible.

Un fallo, sin embargo, es un error en tiempo de ejecución. Por ejemplo, "el sistema no transfirió dinero". Las pruebas funcionales detectan fallos comparando los resultados reales de una solicitud con sus resultados esperados.

Una vez definidos los conceptos más simples, vamos a los conceptos de programación.

La depuración será el proceso donde se rastrea una fallo hasta llegar a los errores que lo causaron.

Un error es un mensaje emitido por el sistema informando que no se puede hacer algo debido a problemas en la solicitud. Los errores se deben a problemas de enlace o restricciones de validación que no cumplen.

Distinguimos, además, un pánico que se produce cuando se lanza una excepción, por ejemplo cuando una solicitud no está autorizada o cuando hay un problema de conexión con la base de dato. Los pánicos no se pueden recuperar generalmente.

Tipos de casos de prueba

El SUT es el sistema bajo prueba, en concreto, será la aplicación Acme Jobs y la probaremos funcionalidad por funcionalidad.

Seguiremos el esquema de la forma "<user-role> realiza <command> en <entity> sujeto a <constraints>". Todos los requisitos de una característica implican el mismo rol de usuario y entidad, siendo el comando y las restricciones lo que varía.

El accesorio consiste en una red de artefactos que debe configurar para que se pruebe su SUT, será la base de datos.

Distinguimos e casos de pruebas a la hora de implementar las pruebas: positivos y negativos.

- Un caso de prueba positivo comprueba que el SUT funciona bien en un contexto en el que se espera que funcione bien. Es decir, que el SUT no emite ningún error o excepción y produce los resultados esperados.
- Un caso de prueba negativo comprueba que el SUT funciona bien en un contexto en el que se espera que informe de un error o se espera que entre en pánico.
- Un caso de prueba hacking comprueba que el SUT no permite realizar acciones para las cuales un usuario no tiene permisos y explotar vulnerabilidades del sistema.

Una clase de prueba encapsulará a un caso de prueba positivo y posiblemente negativo con respecto a una característica particular.

Por tanto, las pruebas consistirán en encontrar fallos. Primero, se prueba una característica en el SUT utilizando un script. Luego, comprueba si la salida es la esperada o no. Si no lo es, se detecta un fallo, es decir, el SUT tiene un error

Respecto a la taxonomía aplicada, en nuestro caso, las pruebas se han aplicado los requisitos funcionales y no funcionales de forma dinámica, ejecutando el código y comparando los con lo que se espera que produzca. Además, dado que las pruebas realizadas solo son conscientes de la interfaz, se denominan pruebas de caja negra. Con la metodología seguida, se realizan pruebas E2E, comprobando desde la interfaz de usuario hasta la base de datos y viceversa. Es decir, pruebas de integración E2E.

Pruebas en Acme-Toolkits

A continuación, se detallarán las distintas pruebas realizadas en nuestro proyecto. Se han realizado las pruebas dependiendo del rol y operación, cada una con ellas con su caso positivo y negativo. Como roles distinguimos administrador, inventor, patron. Respecto a las operaciones tenemos listar, mostrar, crear, actualizar, publicar y eliminar. Cada prueba tendrá su fichero CSV correspondiente.

Prueba de listar y mostrar

Primero, se inicia sesión con las credenciales adecuadas según el rol. Se accede a la vista correspondiente a través del menú principal. Se comprueba que haya un listado y se ordena según alguno de sus atributos. A continuación, por cada entrada de la lista, se comprueba columna a columna que los datos corresponden con los datos que contiene esa misma entrada en el fichero CSV. Se hacen las validaciones correspondientes y, finalmente, si es necesario, se cierra sesión.

A continuación, vemos la prueba de listar del inventor-item:

```
@ParameterizedTest
@CsvFileSource(resources = "/inventor/item/list.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void positiveTest(final int recordIndex, final String type, final String name, final String code, final String technology,
    final String description, final String retailPrice, final String link, final String published) {

    super.signIn("inventor1", "inventor1");

    super.clickOnMenu("Inventor", "My Items");
    super.checkListingExists();

    super.sortListing(1, "asc");

    super.checkColumnHasValue(recordIndex, 0, name);
    super.checkColumnHasValue(recordIndex, 1, code);
    super.checkColumnHasValue(recordIndex, 2, technology);
    //super.checkColumnHasValue(recordIndex, 3, retailPrice);
    super.checkColumnHasValue(recordIndex, 4, type);
    super.checkColumnHasValue(recordIndex, 5, published);

    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();
    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("name", name);
    super.checkInputBoxHasValue("technology", technology);
    //super.checkInputBoxHasValue("retailPrice", retailPrice);
    super.checkInputBoxHasValue("description", description);
    super.checkInputBoxHasValue("link", link);
    super.checkInputBoxHasValue("published", published);


    if(published.trim() == "false") {
```

Prueba de crear

Caso positivo: primero, se inicia sesión con las credenciales adecuadas según el rol. Se accede a la vista correspondiente a través del menú principal. Se comprueba que haya un listado y se hace click en el boton de crear. Se rellena el formulario con los valores necesarios y se hace click en el botón de submit. Una vez creado, se accede a la vista de listado, se ordena y se accede a la entrada que acabamos de entrar. Una vez dentro de la vista, se comprueba que contiene los datos introducidos anteriormente

A continuación, vemos la prueba positiva de crear del inventor-item:

```
public void positiveTest(final int recordIndex, final String type, final String name, final String technology, final
    super.signIn("inventor1", "inventor1");

    super.clickOnMenu("Inventor", "My Items");
    super.checkListingExists();

    super.clickOnButton("Create");
    super.fillInputBoxIn("name", name);
    super.fillInputBoxIn("technology", technology);
    super.fillInputBoxIn("retailPrice", retailPrice);
    super.fillInputBoxIn("description", description);
    super.fillInputBoxIn("link", link);
    super.fillInputBoxIn("type", type);
    super.clickOnSubmit("Create");

    super.clickOnMenu("Inventor", "My Items");
    super.checkListingExists();

    super.sortListing(0, "asc");

    super.checkColumnHasValue(recordIndex, 0, name);
    super.checkColumnHasValue(recordIndex, 2, technology);
    super.checkColumnHasValue(recordIndex, 3, retailPrice);
    super.checkColumnHasValue(recordIndex, 4, type);
    super.checkColumnHasValue(recordIndex, 5, published);

    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();
    super.checkInputBoxHasValue("name", name);
    super.checkInputBoxHasValue("technology", technology);
```

Caso negativo: primero, se inicia sesión con las credenciales adecuadas según el rol. Se accede a la vista correspondiente a través del menú principal. Se comprueba que haya un listado y se hace click en el boton de crear. Se rellena el formulario con algún valor incorrecto (por alguna restricción existente) y se hace click en el botón de "submit". Se comprueba que hay algún error.

A continuación, vemos la prueba negativa de crear del inventor-item:

```
@ParameterizedTest
@CsvFileSource(resources = "/inventor/item/create-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(20)
public void negativeTest(final int recordIndex, final String type, final String name, final String technology, final

    super.signIn("inventor1", "inventor1");

    super.clickOnMenu("Inventor", "My Items");
    super.checkListingExists();

    super.clickOnButton("Create");
    super.fillInputBoxIn("name", name);
    super.fillInputBoxIn("technology", technology);
    super.fillInputBoxIn("retailPrice", retailPrice);
    super.fillInputBoxIn("description", description);
    super.fillInputBoxIn("link", link);
    super.fillInputBoxIn("type", type);
    super.clickOnSubmit("Create");

    super.checkErrorsExist();

    super.signOut();
}
```

Caso hacking: se comprobará la imposibilidad de entrar a un URL que no existe o una vista la cual no se tiene permisos.

A continuación, vemos la prueba hacking de crear del inventor-item:

```
@Test
@Order(30)
public void hackingTest() {
    super.checkNotLinkExists("Account");
    super.navigate("/inventor/item/create");
    super.checkPanicExists();

    super.signIn("administrator", "administrator");
    super.navigate("/inventor/item/create");
    super.checkPanicExists();
    super.signOut();

    super.signIn("patron2", "patron2");
    super.navigate("/inventor/item/create");
    super.checkPanicExists();
    super.signOut();
}
```

Prueba de publicar

Caso positivo: primero, se inicia sesión con las credenciales adecuadas según el rol. Se accede a la vista correspondiente a través del menú principal. Se comprueba que haya un listado y toma cualquier entrada que tenga el atributo de publicado a false. Se accede al formulario y se hace click en el botón de "publicar". Se comprueba que no hay errores y, si es necesario, se cierra sesión.

A continuación, vemos la prueba positiva de publicar del inventor-item:

```
@ParameterizedTest
@CsvFileSource(resources = "/inventor/item/publish-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void positiveTest(final int recordIndex, final String code) {
    super.signIn("inventor1", "inventor1");

    super.clickOnMenu("Inventor", "My Items");
    super.checkListingExists();
    super.sortListing(1, "asc");
    super.checkColumnHasValue(recordIndex, 1, code);

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

    super.signOut();
}
```

Caso negativo: primero, se inicia sesión con las credenciales adecuadas según el rol. Se accede a la vista correspondiente a través del menú principal. Se comprueba que haya un listado y toma cualquier entrada que tenga el atributo de publicado a true. Se accede al formulario y se comprueba que no existe el botón de “publicar” y, si es necesario, se cierra sesión.

A continuación, vemos la prueba negativa de publicar del inventor-item:

```
@ParameterizedTest
@CsvFileSource(resources = "/inventor/item/publish-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(20)
public void negativeTest(final int recordIndex, final String code) {
    super.signIn("inventor1", "inventor1");

    super.clickOnMenu("Inventor", "My Items");
    super.checkListingExists();
    super.sortListing(1, "asc");
    super.checkColumnHasValue(recordIndex, 1, code);

    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();
    super.checkNotButtonExists("Publish");

    super.signOut();
}
```

Prueba de actualizar

Caso positivo: primero, se inicia sesión con las credenciales adecuadas según el rol. Se accede a la vista correspondiente a través del menú principal. Se comprueba que haya un listado y toma cualquier entrada que tenga el atributo de publicado a false. Se accede al formulario y se rellena el formulario con nuevos parámetros. Se hace click en el botón de “actualizar”. Se vuelve a entrar en la vista del listado y se accede a la vista de la entrada anterior. Se comprueba que los datos corresponden a los introducidos anteriormente. Finalmente, si es necesario, se cierra sesión.

A continuación, vemos la prueba positiva de actualizar del inventor-item:

```
@ParameterizedTest
@CsvFileSource(resources = "/inventor/item/update-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void positiveTest(final int recordIndex, final String type, final String name, final String code, final Strin
    super.signIn("inventor1", "inventor1");

    super.clickOnMenu("Inventor", "My Items");
    super.checkListingExists();

    super.sortListing(1, "asc");

    super.checkColumnHasValue(recordIndex, 1, code);
    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();

    super.fillInputBoxIn("name", name);
    super.fillInputBoxIn("technology", technology);
    super.fillInputBoxIn("retailPrice", retailPrice);
    super.fillInputBoxIn("description", description);
    super.fillInputBoxIn("link", link);
    super.fillInputBoxIn("type", type);
    super.clickOnSubmit("Update");

    super.checkListingExists();

    super.sortListing(1, "asc");

    super.checkColumnHasValue(recordIndex, 0, name);
    super.checkColumnHasValue(recordIndex, 2, technology);
    super.checkColumnHasValue(recordIndex, 3, retailPrice);
    super.checkColumnHasValue(recordIndex, 4, type);
    super.checkColumnHasValue(recordIndex, 5, published);
```

Caso negativo: primero, se inicia sesión con las credenciales adecuadas según el rol. Se accede a la vista correspondiente a través del menú principal. Se comprueba que haya un listado y toma cualquier entrada que tenga el atributo de publicado a false. Se accede al formulario y se rellena el formulario con nuevos parámetros con algún valor incorrecto (por alguna restricción existente) y se hace click en el botón de "update". Se comprueba que hay algún error.

A continuación, vemos la prueba negativa de actualizar del inventor-item:

```
@ParameterizedTest
@CsvFileSource(resources = "/inventor/item/update-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(20)
public void negativeTest(final int recordIndex, final String type, final String name, final String code, final String te
    super.signIn("inventor1", "inventor1");

    super.clickOnMenu("Inventor", "My Items");
    super.checkListingExists();

    super.sortListing(1, "asc");

    super.checkColumnHasValue(recordIndex, 1, code);
    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();

    super.fillInputBoxIn("name", name);
    super.fillInputBoxIn("technology", technology);
    super.fillInputBoxIn("retailPrice", retailPrice);
    super.fillInputBoxIn("description", description);
    super.fillInputBoxIn("link", link);
    super.fillInputBoxIn("type", type);
    super.clickOnSubmit("Update");

    super.checkErrorsExist();

    super.signOut();
}
```

Prueba de eliminar

Caso positivo: primero, se inicia sesión con las credenciales adecuadas según el rol. Se accede a la vista correspondiente a través del menú principal. Se comprueba que haya un listado y toma cualquier entrada que tenga el atributo de publicado a false. Se accede al formulario y se hace click en el botón de “eliminar”. Se comprueba que no hay errores y, si es necesario, se cierra sesión.

A continuación, vemos la prueba positiva de eliminar del inventor-item:

```java
@ParameterizedTest
@CsvFileSource(resources = "/inventor/item/delete-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void positiveTest(final int recordIndex, final String code) {
    super.signIn("inventor1", "inventor1");

    super.clickOnMenu("Inventor", "My Items");
    super.checkListingExists();
    super.sortListing(1, "asc");
    super.checkColumnHasValue(recordIndex, 1, code);

    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();
    super.clickOnSubmit("Delete");
    super.checkNotErrorsExist();

    super.signOut();
}
```

Caso negativo: primero, se inicia sesión con las credenciales adecuadas según el rol. Se accede a la vista correspondiente a través del menú principal. Se comprueba que haya un listado y toma cualquier entrada que tenga el atributo de publicado a true. Se accede al formulario y se comprueba que no existe el botón de “eliminar” y, si es necesario, se cierra sesión.

A continuación, vemos la prueba negativa de eliminar del inventor-item:

```java
@ParameterizedTest
@CsvFileSource(resources = "/inventor/item/delete-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(20)
public void negativeTest(final int recordIndex, final String code) {
    super.signIn("inventor1", "inventor1");

    super.clickOnMenu("Inventor", "My Items");
    super.checkListingExists();
    super.sortListing(1, "asc");
    super.checkColumnHasValue(recordIndex, 1, code);

    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();
    super.checkNotButtonExists("Delete");

    super.signOut();
}
```

Conclusiones

Durante el desarrollo de este proyecto hemos adquirido conocimientos tanto teóricos como prácticos sobre el proceso de prueba de software. Hemos comprobado que son muchos los conceptos y pasos previo que son necesarios para ello, pero necesarios para asegurar que el proceso es llevado a cabo correctamente.

Este proceso nos ha permitido identificar errores y bugs de forma progresiva, a medida que íbamos implementando las distintas funcionalidades. Así, hemos podido detectar problemas de forma rápida y temprana, ahorrándonos en futas implementaciones muchos problemas de refactorización y asegurando un alto rendimiento y calidad de la aplicación.

Por tanto, podemos asegurar que el conocimiento adquirido será realmente útil y aplicable a los proyectos futuros, pues es un proceso presente en el desarrollo de cualquier aplicación.

Bibliografía

El temario de la asignatura.