WHAT HAVE WE LEARNT ABOUT TESTING

S06

Tabla de contenidos

Resumen ejecutivo

En el siguiente reporte se tratará de forma detallada los conocimientos adquiridos por los alumnos a lo largo de la asignatura sobre testing.

Tabla de revisión

Versión: 1.0	06/09/2022	Creación del documento

Contenido

El contenido se organizará en una sección donde se encontrará una breve introducción sobre los conocimientos previos a la asignatura y que dará pie a la sección principal donde se expondrá aquellos nuevos conocimientos adquiridos por los alumnos y la utilidad de estos mismos. Las últimas secciones se corresponden con la sección de conclusiones con un breve resumen de todo lo anterior y bibliografía.

Testing

Tal y como se comentó en el documento sobre testing del D01, las pruebas se deben programar a la vez que se desarrolla el software, pues facilitan conocer si al avanzar con las implementaciones, se ha modificado alguna funcionalidad antigua y ésta ha dejado de funcionar. Existen numerosos tipos de pruebas, siendo las más comunes las pruebas unitarias. Es imposible probar el 100% del código de una aplicación, pero se pueden programar pruebas que se encargan de asegurar el correcto funcionamiento de bloques de código importantes para determinadas funcionalidades.

Comenzando con lo aprendido en la asignatura, podemos decir que el testing consiste en encontrar comportamientos que no son los que el cliente espera en nuestra aplicación. Existen muchos tipos de test atendiendo a diferentes clasificaciones, como tests estáticos o dinámicos, de caja blanca o de caja negra, funcionales o no funcionales… Los test en los que nos hemos centrado en la asignatura han sido test E2E (test end-to-end) en los que los alumnos podemos ver cómo se abre el navegador Firefox y se empiezan a abrir las ventas y hacer las cosas que hemos programado. Este acercamiento a los test nos ha parecido una forma nueva y muy adecuada de ejecutar tests ya que es más cercano al usuario y se puede observar en qué momento hay algún fallo si lo hay y es más fácil de programar.

A la hora de hacer test, hay que tener en cuenta que se debe intentar obtener la mayor cobertura posible, esto se obtiene explotando los dos tipos de cobertura que hemos dado, cobertura de rutas y cobertura de datos. Para explotar al máximo la cobertura de rutas, se deben probar el máximo número de secuencias, condicionales y bucles de la aplicación. Para explotar la cobertura de datos, los test programados deberán tener la mayor variabilidad de datos posible.

Por otro lado y atendiendo a los tipos de casos de pruebas, hemos aprendido tres: tests positivos, tests negativos y tests de hacking. Los tests positivos son aquellos en los que probamos el correcto funcionamiento de la aplicación en casos en los que los datos introducidos son correctos y no violan ninguna de las restricciones programadas. Por otro lado, los tests negativos son aquellos en los que se intenta probar que el sistema está preparado para soportar que se introduzcan datos incorrectos y que las restricciones y mensajes del sistema están bien programados. Por último, los tests de hacking se pueden dividir en tres tipos: tests que prueban que un usuario con un determinado rol no pueda acceder a funcionalidades pertenecientes a otro rol, tests que prueban que un usuario con el mismo rol que otro no pueda acceder a secciones pertenecientes al otro usuario y a las que no debería tener acceso; y tests que prueban que aunque un usuario no pueda hacer ciertas acciones que no debería.

En nuestro proyecto se han programado pruebas para listados, show, creaciones, actualizaciones y publicación de entidades. Para listados, las pruebas que se han programado son positivas:

```
@ParameterizedTest
@CsvFileSource(resources = "/any/peep/list.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void positiveTest(final int recordIndex, final String instantiationMoment, final String heading, final String writer, final String text, final String optionalEmail) {

    super.clickOnMenu("Anonymous", "Recent peeps");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.checkColumnHasValue(recordIndex, 0, instantiationMoment);
    super.checkColumnHasValue(recordIndex, 1, heading);
    super.checkColumnHasValue(recordIndex, 2, writer);
    super.checkColumnHasValue(recordIndex, 3, text);
    super.checkColumnHasValue(recordIndex, 4, optionalEmail);

}
```

Para los show, las pruebas que se han programada son pruebas positivas también:

```
@ParameterizedTest
@CsvFileSource(resources = "/authenticated/systemConfiguration/show.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void positiveTest(final int recordIndex, final String acceptedCurrencies, final String defaultSystemCurrency,
    final String spamTerms, final Double spamThreshold) {
    super.signIn("epicure1", "epicure1");
    super.clickOnMenu("Authenticated", "Currencies Information");

    super.checkInputBoxHasValue("defaultSystemCurrency", defaultSystemCurrency);
    super.checkInputBoxHasValue("acceptedCurrencies", acceptedCurrencies);
}
```

Para creación, actualización y publicación, se han programado pruebas positivas, negativas y de hacking:

```
@ParameterizedTest
@CsvFileSource(resources = "/chef/memorandum/create-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void positiveTest(final int recordIndex, final String report, final String optionalLink, final String fineDishCode) {

    super.signIn("chef1", "chef1");
    super.clickOnMenu("Chef", "My memorandums");

    super.checkListingExists();
    super.checkButtonExists("Create");
    super.clickOnButton("Create");

    super.checkFormExists();
    super.fillInputBoxIn("report", report);
    super.fillInputBoxIn("optionalLink", optionalLink);
    super.fillInputBoxIn("fineDish", fineDishCode);
    super.fillInputBoxIn("confirmation", "true");
    super.clickOnSubmit("Create");

    super.checkListingExists();
    super.sortListing(1, "desc");
    super.checkColumnHasValue(recordIndex, 2, report);
    super.checkColumnHasValue(recordIndex, 3, optionalLink);

    super.signOut();

}
```

```
@ParameterizedTest
@CsvFileSource(resources = "/chef/memorandum/create-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void negativeTest(final int recordIndex, final String report, final String optionalLink, final String fineDishCode) {

    super.signIn("chef1", "chef1");
    super.clickOnMenu("Chef", "My memorandums");

    super.checkListingExists();
    super.checkButtonExists("Create");
    super.clickOnButton("Create");

    super.checkFormExists();
    super.fillInputBoxIn("report", report);
    super.fillInputBoxIn("optionalLink", optionalLink);
    super.fillInputBoxIn("fineDish", fineDishCode);
    super.fillInputBoxIn("confirmation", "true");
    super.clickOnSubmit("Create");

    super.checkErrorsExist();

    super.signOut();

}
```

```java
@Test
@Order(10)
public void hackingTest() {

    super.checkNotLinkExists("Account");
    super.navigate("/chef/memorandum/create");
    super.checkPanicExists();

    super.signIn("administrator", "administrator");
    super.navigate("/chef/memorandum/create");
    super.checkPanicExists();
    super.signOut();

    super.signIn("epicure1", "epicure1");
    super.navigate("/chef/memorandum/create");
    super.checkPanicExists();
    super.signOut();
}
```

Conclusión

En este documento se ha explicado qué conocimientos han adquirido los alumnos sobre testing a lo largo de DP2. Aprender sobre testing es muy importante ya que permite detectar fallos en el sistema a tiempo para arreglarlos. Esto garantiza calidad, fiabilidad y seguridad. Además, a lo largo del proyecto, ambos integrantes del grupo hemos sido testers por lo que el grupo entero ha adquirido nuevos conocimientos. Consideramos que lo que hemos aprendido es muy útil y lo podremos usar en futuros trabajos.

Bibliografía

Material de la asignatura.