Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Integración de pruebas de interfaz con Cucumber

Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas II

Curso 2019 – 2020

Miembros del equipo

Jorge Andrea Molina

Juan Carlos Cortés Muñoz

María Elena Molino Peña

Alejandro Muñoz Aranda

Mario Ruano Fernández

Fernando Ruíz Robles

https://github.com/dp2-g3-7/petclinic.git

Contenido

Introducción

Behavoir-Driven Development (BDD) o desarrollo guiado por el comportamiento es una práctica propia del desarrollo ágil que trata de motivar a los desarrolladores a que empleen ejemplos concretos y conversaciones para formalizar un entendimiento compartido (con el cliente) de cómo debería funcionar la aplicación.

Cucumber es una herramienta de desarrollo que soporta este tipo de desarrollo. Permite que los comportamientos esperados del software se especifiquen en un lenguaje lógico que los clientes puedan entender. En este caso, se ha integrado para la implementación de pruebas de interfaz de usuario de algunas historias de petclinic. A continuación, se explicará como se ha llevado a cabo este proceso.

Implementación

En primer lugar, es necesario añadir las dependencias de Cucumber al archivo pom.xml del proyecto.

```
<!-- cucumber dependences -->
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>${cucumber.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>${cucumber.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-spring</artifactId>
    <version>${cucumber.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit-platform-engine</artifactId>
    <version>${cucumber.version}</version>
</dependency>
<dependency>
    <groupId>net.masterthought</groupId>
    <artifactId>cucumber-reporting</artifactId>
    <version>4.2.3</version>
</dependency>
<dependency>
    <groupId>de.monochromata.cucumber</groupId>
    <artifactId>reporting-plugin</artifactId>
    <version>4.0.36</version>
</dependency>
```

Además, el profesorado de la asignatura recomendó la descarga del plugin de Cucumber disponible en Eclipse Marketplace. Esto permite una visualización adecuada del código relativo a esta herramienta.

Posteriormente, se creó el paquete bdd dentro de la carpeta con las pruebas de la aplicación. Para que se procesen de forma correcta es necesario crear la clase CucumberUITest; encargada de lanzar todos los test, y AbstractStep, en el paquete bdd.stepdefinitions. Ambas se han obtenido del repositorio de ejemplos de la asignatura.

Por último, basta con crear la clase pruebaUITest.feature para cada una de las historias que se pretenden probar, con el lenguaje entendible por el cliente. Pero esta no puede funcionar si no se implementa el código de la aplicación para cada sentencia, por lo que se crea su correspondiente clase pruebaStepDefinitions extendiendo de AbstractStep. En el próximo epígrafe se presenta un ejemplo de implementación de este tipo de pruebas.

Es importante tener en cuenta que todas las historias se lanzarán a la vez en una misma sesión. Por ello, se ha de escoger una clase principal que se encargue de habilitar la sesión. De lo contrario, surgirá un error el cual impide que se lancen las pruebas.

Ejemplo

Se va a explicar la prueba de interfaz relativa a los escenarios de la historia de usuario “Aceptar o rechazar una solicitud de estancia”. Comenzando por la clase AcceptRejectStay.feature, hemos de definir la funcionalidad a probar:

```
Feature: Accept or reject a stay
  As an admin, I want to accept or reject a pending stay.
```

Luego, es turno de los escenarios. Son tres en este caso: aceptar la solicitud, rechazar la solicitud y rechazar una solicitud que ya ha sido aceptada. Los dos primeros son positivos y el segundo, negativo.

```
Scenario: Accept a stay (positive)
  Given I am not logged in the system
  When Im logged in the system as "admin1"
  And I accept a pending stay request
  Then The stay is accepted

Scenario: Reject a stay (positive)
  Given I am not logged in the system
  When Im logged in the system as "admin1"
  And I reject a pending stay request
  Then The stay is rejected

Scenario: Reject an accepted stay (negative)
    Given I am not logged in the system
    When Im logged in the system as "admin1"
    And I try to reject an accepted stay
    Then I get an error
```

El proceso es claro: el usuario no ha iniciado sesión en el sistema, se loguea e intenta realizar alguna de las acciones descritas. El resultado, recogido en la sentencia “Then”, comprueba una modificación desembocada por las acciones realizadas.

Por último, se han de describir, en un lenguaje más cercano al desarrollador, el flujo que se realiza en la aplicación. Se muestran partes de la clase AcceptRejectStayStepDefinitions:

```
@Given("I am not logged in the system")
public void IamNotLogged() throws Exception{
    getDriver().get("http://localhost:"+port);
    WebElement element=getDriver().findElement(By.xpath("//div[@id='main-navbar']/ul[3]/li/a"));
    if(element==null || !element.getText().equalsIgnoreCase("login")) {
        getDriver().findElement(By.xpath("//div[@id='main-navbar']/ul[3]/li/a")).click();
        getDriver().findElement(By.linkText("Logout")).click();
        getDriver().findElement(By.xpath("//button[@type='submit']")).click();
    }
}
@When("Im logged in the system as {string}")
public void IdoLoginAs(String username) throws Exception {
    loginAs(username,passwordOf(username),port, getDriver());
}
```

Ambos son métodos que se utilizan en todas las clases de definición de pasos para no repetir código excesivamente.

```
@And("I accept a pending stay request")
public void goStaysAndAccept() {
    getDriver().findElement(By.linkText("STAYS")).click();
    getDriver().findElement(By.linkText("Change Status")).click();
    new Select(getDriver().findElement(By.id("status"))).selectByVisibleText("ACCEPTED");
    getDriver().findElement(By.xpath("//option[@value='ACCEPTED']")).click();
    getDriver().findElement(By.xpath("//button[@type='submit']")).click();
}
```

Se registra el flujo que realizaría un usuario, administrador, cuando pretende hacer uso de esta funcionalidad.

```
@Then("The stay is accepted")
public void checkStayIsAccepted() {
    assertEquals("ACCEPTED", getDriver().findElement(By.xpath("//table[@id='stayTables']/tbody/tr[2]/td[3]")).getText());

}
```

Se comprueba que se ha producido una modificación. En este caso, la solicitud de estancia ha sido aprobada.

Este procedimiento se ha llevado a cabo en un total de 15 historias de usuario de Petclinic, cubriendo un 51% de las implementadas en la aplicación.

Conclusión

El buen entendimiento entre el cliente y el equipo de desarrollo es esencial para conseguir el éxito, que es lo que se pretende obtener con la aplicación de BDD y, más específicamente, Cucumber.

Muy pocos usuarios normales, sin conocimiento de la tecnología, sería capaz de comprender la implementación de una prueba cualquiera de la aplicación. En cambio, gracias a Cucumber, la historia se presenta como un guion donde el cliente puede ver el flujo de la aplicación paso a paso. Por si fuera poco, las pruebas de interfaz de usuario son las que mayor grado de fiabilidad otorgan para un cliente ya que a estos no les interesa o no entienden la gran mayoría de las implementaciones de código.

Las pruebas de interfaz de usuario con Cucumber hacen partícipe del desarrollo a los clientes y usuarios de la aplicación. Esto les provocará un mayor interés por participar en el desarrollo de la aplicación y, así, crear una base sólida para un acuerdo exitoso.