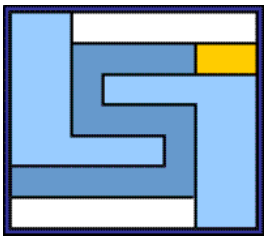


Item 5 - Report

Diseño y Pruebas

Grado de Ingeniería del Software

Curso 3

Armando Garrido Castro
Jorge Puente Zaro
Manuel Enrique Pérez Carmona
César García Pascual
Pablo Tabares García
Rafael Trujillo González

Fecha: 22 de marzo de 2018

Índice

1. Introducción

En este entregable se nos pide "*testear*" un controlador mediante un framework a nuestra elección. Para realizar dicha tarea existen numerosas herramientas, pero nos hemos decantado por el uso del framework *Mockito*, puesto que es el más extendido y el que más documentación y tutoriales posee.

Este framework que permite crear objetos llamados "*mocks*" (instancias de objetos decidiendo el comportamiento que deben tener), que nos permiten simular el funcionamiento de los métodos que deseemos testear para comprobar que su resultado es correcto.

2. Componentes

Para poder utilizar este framework, simplemente necesitamos añadir una dependencia más a nuestro fichero *pom.xml* (es necesario jUnit y el componente de testeo de spring, pero éstos ya se se encuentran por defecto en el workspace suministrado por la asignatura):

```
22        <dependency>
23            <groupId>org.slf4j</groupId>
24            <artifactId>slf4j-log4j12</artifactId>
25            <version>1.7.5</version>
26        </dependency>
27
28        <dependency>
29            <groupId>org.aspectj</groupId>
30            <artifactId>aspectjweaver</artifactId>
31            <version>1.7.4</version>
32        </dependency>
33
34        <!-- Controllers testing -->
35
36
37        <dependency>
38            <groupId>org.mockito</groupId>
39            <artifactId>mockito-core</artifactId>
40            <version>1.9.5</version>
41            <scope>test</scope>
42        </dependency>
43    </dependencies>
```

3. Código necesario

Para comprobar el funcionamiento de esta herramienta, crearemos una sencilla clase para ejecutar un test sobre el controlador de la entidad Zervice (este nombre se debe a un conflicto con la palabra "service"), más concretamente en el método del controlador que lista los servicios para que los usuarios puedan reservarlos para sus reuniones, o para que los managers y el/los administradores puedan verlos.

Dicha clase es la siguiente:

```
public class ControllerTest {

    //Primero creamos el mock del servicio que usa el controlador que testeamos
    @Mock ZerviceService zerviceService;

    @Before
    public void setup()
    {
        //Inicializamos el mock antes de ejecutar el test
        MockitoAnnotations.initMocks( this );
    }

    @Test
    public void testListQuestions()
    {
        // Creamos una coleccion de servicios para simular el funcionamiento
        // del método findAll()
        List<Zervice> zervices = new ArrayList<Zervice>();
        zervices.add( new Zervice());
        when( zerviceService.findAll()).thenReturn( zervices );

        // Creamos una instancia del controlador que vamos a testear
        ZerviceController controller = new ZerviceController();

        // Ajustamos los parámetros del controlador manualmente
        ReflectionTestUtils.setField( controller, "zerviceService", zerviceService );

        // Llamamos al método que queremos testear
        ModelAndView mav = controller.list();

        // Comprobamos que dicho método devuelve el resultado esperado
        assertEquals( zervices, mav.getModel().get( "zervices" ));
        // Comprobamos que el nombre de la vista que se genera es el esperado
        assertEquals( "zervice/list", mav.getViewName());
    }
}
```

Primero, creamos un “mock” del servicio que utilizará el controlador (es una imitación del servicio, a la que después indicaremos que debe realizar en un caso determinado).

Tras esto llamamos al método *initMocks()* para inicializar el mock definido anteriormente.

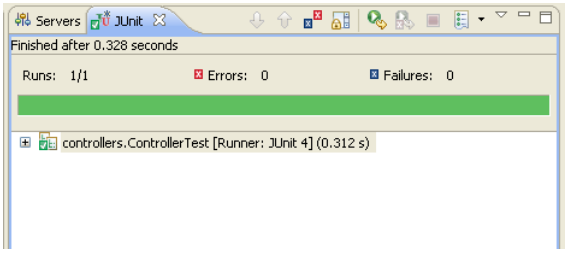
Ya en el método que testea el controlador, primero creamos una colección de servicios a la que añadimos un nuevo servicio (puesto que sólo queremos comprobar que el controlador devuelve una vista que contenga dicha lista, no hemos considerado necesario el crear servicios con atributos). Y acto seguido indicamos al mock que cuando se invoque su método *findAll()* devuelva la lista que acabamos de crear.

Ahora creamos una instancia del controlador que queremos testear y le añadimos los parámetros que éste necesita (el mock que creamos previamente), puesto que no podemos añadírselos mediante la anotación *@Autowired.*

Por último creamos un objeto del tipo *ModelAndView* como resultado de llamar al método del controlador que vamos a testear y comprobamos que dicho objeto contiene la vista de listado de servicios y que su modelo contiene una lista de servicios igual a la que hemos creado al principio y hemos indicado que devuleve nuestro mock.

4. Resultado

Para comprobar si el resultado del test es `positivo, simplemente lo ejecutamos como un test de jUnit:



5. Conclusión

Los tests unitarios son una herramienta crucial a la hora de comprobar el correcto funcionamiento de todas las funcionalidades implementadas para todos los casos de uso planteados a la hora de desarrollar software. En el caso de proyectos como éste, checkear el correcto funcionamiento de los métodos y los servicios de es crucial para comprobar que todas las reglas de negocio se cumplen.

Pero el caso se puede considerar distinto, puesto que al disponer ya de una interfaz gráfica el testeo se puede realizar de una manera más cómoda e intuitiva (incluso delegando esta tarea a personas que no tengan tantos conocimientos de desarrollo software). Pero el uso de herramientas como *Mockito* para realizar tests supone una gran ventaja, puesto que se puede realizar una batería de tests que cubran todas las posibles casuísticas para después ejecutarlos de una forma rápida y eficiente.