URJC Escuela Técnica Superior de Ingeniería Informática

Práctica 1 - Pruebas y calidad del software

Ampliación de Ingeniería del Software

Curso 2020-2021

Fátima Smounat Mahidar
Milagros Mouriño Ursul

Índice

1. Pruebas automáticas

a. Tests unitarios de la lógica de la aplicación (BookService)

En primer lugar, para para cada test se ejecuta el método *setup* anotado con @BeforeEach que crea los objetos de las clases Book y BookService, además de los mocks de las clases BookRepository y NotificationService.

i. bookSaveUnitaryTest

Utilizamos el método *when* de la librería de Mockito para definir el comportamiento de los dobles tanto para guardar el libro (*save*) por parte del objeto repository de la clase BookRepository como para devolver el título (*getTitle*) por parte de la instancia de NotificationService. Esto permite que el método *save* del objeto de la clase BookService (que queremos testear) se ejecuta adecuadamente, ya que son funciones que se ejecutan internamente dentro de este método. Después, se invoca ese mismo método (when) y, en primer lugar, se verifica que se llamó una vez a ese método con *times*. Por otra parte, se comprueba que se lanzó una notificación por parte del mock correspondiente a la clase NotificationService (al ejecutarse el método *save*) que indica que el libro ha sido creado.

ii. bookDeleteUnitaryTest

Al igual que antes, el método *when* define el comportamiento de los mocks de BookRepository y NotificationService. Se asigna un id al libro para poder borrarlo. Se invoca al método *save* del objeto de BookService y posteriormente se llama al método *delete* de ese mismo objeto (when). Una vez borrado, se verifica (then) si se invocó al método *deleteById* del mock de BookRepository y el método *notify* del doble de NotificationService que indica que el libro con el id anterior ha sido borrado, ya que ambos métodos al estar contenidos dentro del método *delete*, se han debido de ejecutar.

b. Tests E2E de la interfaz web (Selenium)

El método *setupClass* anotado con @BeforeAll que se ejecuta antes de todos los métodos sirve para descargarse automáticamente el driver del navegador. Además, el método *setup* anotado con @BeforeEach abre el navegador antes de que se ejecute cada test y la función *teardown* con @AfterEach sirve para cerrar el mismo tras haberse ejecutado el test.

i. bookSaveSeleniumTest

En primer lugar, se obtiene la página del navegador donde reside la aplicación con el método *get* (given). Luego, se procede a ejercitar el SUT (when) haciendo clic primero en el botón hacia la página con el formulario que permite crear el libro. Tras haberse rellenado automáticamente todos los campos con el método *sendKeys* se da clic en un elemento cuyo id es *save*, el cual es el botón que permite dar de alta el libro (funcionalidad a comprobar). Nos dirigimos hacia una página donde se puede eliminar, editar o ir a la página principal. Se da clic en ese último botón que permite ir al menú principal de la aplicación y se verifica (then) que aparece el link del libro creado recientemente con el método *assertNotNull*.

ii. bookDeleteSeleniumTest

Al igual que antes, se obtiene la página de la aplicación con *get* (given) y nos desplazamos a la página donde se procede a crear un nuevo libro. Tras haberlo creado, utilizando el método *findElement* (when) comprobamos que se ha realizado la operación correctamente invocando al método *assertNotNull* que verifica que existe el link del libro en el menú principal de la aplicación. A continuación, se procede a borrar el libro (when). En primer lugar, se hace clic en el link del libro que se quiere eliminar y se emplea el método *getCurrentUrl* para obtener el URI esa página, **ya que contiene el id del libro**. Este identificador permite localizar el botón que permite borrar el libro con el método *xpath* de la librería By. Una vez se hace clic en ese botón, se verifica mediante el método *assertThrows* que se lanza una NoSuchElementException al intentar acceder a un link que no existe, ya que el libro ha sido eliminado.

c. Tests E2E de la API REST (RESTAssured)

i. bookSaveRestAssuredTest

En primer lugar, se crea un libro realizando una petición post donde la información del mismo se encuentra en el body del mismo en formato JSON. A continuación, se obtiene el id de ese libro que permite realizar una petición *get* (when) para obtener el recurso. Posteriormente, se verifica (then) que se produce adecuadamente la operación (código de estado 200 OK) y, finalmente, se comprueba que la información que se obtiene con el *get* se corresponde con la del libro que se creó anteriormente con el método *post*.

ii. bookDeleteRestAssuredTest

Se procede exactamente igual que en el apartado anterior. Primero, creamos el libro con el método *post* donde los datos en formato JSON se colocan en el body de la petición. Se obtiene el id que permite realizar la petición *delete* (when) y se verifica que se produce

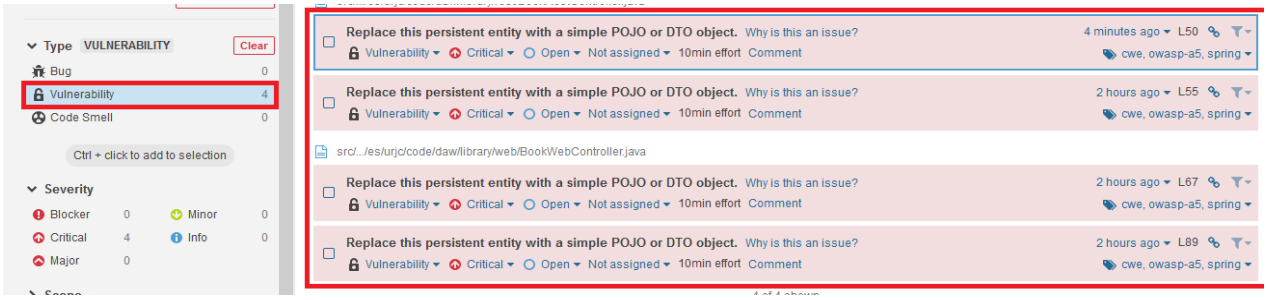
adecuadamente la operación (código de estado 200 OK). Además, se produce una petición *get* y se verifica que el código de estado en este caso es 404 Not Found, ya que el recurso ha sido eliminado.

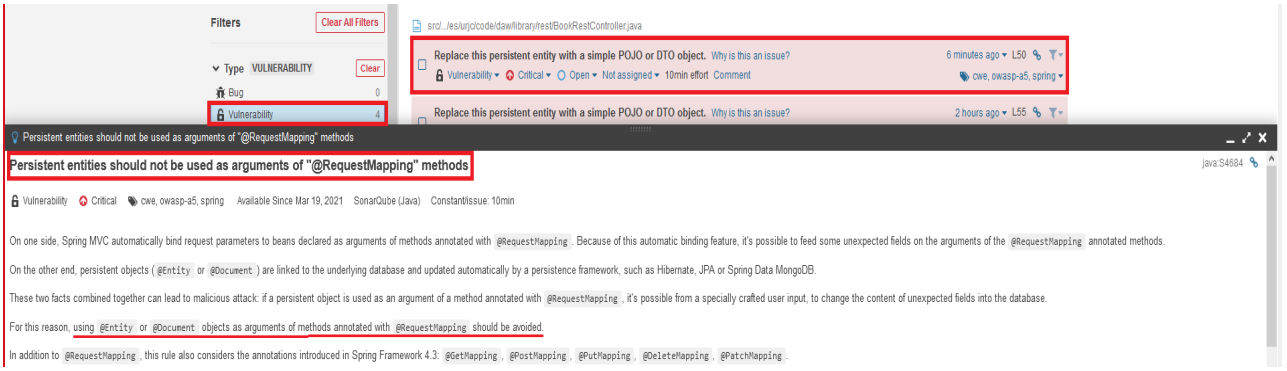
2. Análisis estático de código

a. Issues de tipo Vulnerability

i. Reemplazar una entidad por un objeto DTO.

Se reportaba que era **inseguro** pasar una entidad Book como argumento a algunos métodos dentro de las clases BookRestController y BookWebController. Se proponía reemplazar esa entidad por un objeto POJO o DTO. Finalmente, se optó por reemplazar esas entidades Book por objetos pertenecientes a una nueva clase que denominamos BookDTO con el mismo comportamiento.





b. Issues de tipo Code Smell

i. Evitar la duplicación de código.

Se producía la repetición de la cadena “books” dentro de la clase BookWebController que empeoraba la **mantenibilidad** del código por lo que se resolvió el problema creando una constante estática denominada “BOOKS” que sustituía las cuatro apariciones de la constante literal en la clase controladora.

ii. Eludir la llamada a varios métodos dentro de código en un test.

La invocación de dos métodos *partialLinkText* y *findElement* dentro de la función *assertThrows* de la clase SeleniumTest empeoraba la **legibilidad** del código, por lo que se guardó el resultado de la invocación del método *partialLinkText* en una variable By y ésta ya se pasó como argumento al método *findElement*.

iii. Suprimir el modificador de visibilidad de clases y tests.

Se eliminaron los modificadores de visibilidad tanto de las clases como de los métodos anotados con @Test dado que, al igual que en el issue anterior, el código era menos **legible**.