# PRÁCTICA 2. TESTING EN E2E Y SONAR.

Ampliación de Ingeniería del Software.

TERCER CURSO. INGENIERÍA INFORMÁTICA.

Loreto Uzquiano Esteban y Javier Romero García.

# ÍNDICE

# DETECCIÓN DEL BUG

## DESCRIPCIÓN BUG ENCONTRADO

El bug encontrado en la web llamada “Nitflex” se produce al crear una película nueva (con sus respectivos campos), guardarla, darle a editar y posteriormente a cancelar. Al pulsar este último botón, se produce un error de “mapping”, cuando en realidad, la web debería redirigirte a la pantalla anterior. Esto mismo ocurre cuando se intenta editar una película ya creada.

## TEST DE REGRESIÓN – FALLA

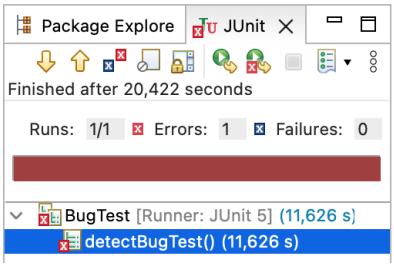
Hemos creado un test de regresión en el que creamos una película (con sus respectivos campos), la guardamos e intentamos editarla. Posteriormente, le damos a cancelar, y comprobamos que el título y la sinopsis de la película coinciden con los datos que se introdujeron al crear esta misma película.

Además, guardamos la URL de la web antes de darle al botón de editar y la URL de la web después de darle al botón de cancelar. Posteriormente, comprobamos que ambas URLs sean idénticas. Esto significaría que, al pulsar el botón de cancelar, la web redirige al usuario automáticamente a la página anterior (donde se muestran los detalles de la película).

A continuación, mostramos una captura de pantalla en la que podemos observar el código de este mismo test.

```java
package es.codeurjc.ais.nitflex.Selenium;

import static org.assertj.core.api.Assertions.assertThat;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.web.server.LocalServerPort;

import es.codeurjc.ais.nitflex.Application;

@SpringBootTest(
        classes = Application.class,
        webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class BugTest {
    @LocalServerPort
    int port;

    WebDriver driver;

    @BeforeEach
    public void setup() {
        driver = new ChromeDriver();
    }

    @AfterEach
    public void teardown() {
        if(driver != null) {
            driver.quit();
        }
    }

    @Test
    public void detectBugTest() throws InterruptedException {
        driver.get("http://localhost:" + this.port + "/");

        driver.findElement(By.id("create-film")).click(); //Click "New Film"

        //Fill in the fields of a new film
        driver.findElement(By.name("title")).sendKeys("101 dalmatas");
        driver.findElement(By.name("releaseYear")).sendKeys("1961");
        driver.findElement(By.name("url")).sendKeys("https://www.tebeosfera.com/T3content/img/T3_series/a/s/101_dalmatas.jpeg");
        driver.findElement(By.name("synopsis")).sendKeys("Los dalmatas Pongo y Perdita son una feliz pareja canina que vive rodeada de sus cachorros y con sus amos, Roger y Anita. "
                + "pero su felicidad esta amenazada por Cruella De Ville, una perfida mujer que adora los abrigos de pieles.");

        driver.findElement(By.id("Save")).click(); //Click "Save"
        String urlActual = driver.getCurrentUrl();

        driver.findElement(By.id("edit-film")).click(); //Click "Edit"

        driver.findElement(By.id("cancel")).click(); //Click "Cancel"
        String urlPosterior = driver.getCurrentUrl();

        String titleNuevo = driver.findElement(By.id("film-title")).getText();
        String synopsisNuevo = driver.findElement(By.id("film-synopsis")).getText();

        assertThat(titleNuevo).isEqualTo("101 dalmatas");
        assertThat(synopsisNuevo).isEqualTo("Los dalmatas Pongo y Perdita son una feliz pareja canina que vive rodeada de sus cachorros y con sus amos, Roger y Anita. "
                + "pero su felicidad esta amenazada por Cruella De Ville, una perfida mujer que adora los abrigos de pieles.");
        assertThat(urlActual).isEqualTo(urlPosterior);
    }

}
```

A continuación, mostramos una captura de pantalla donde se observa que este test falla.




## CORRECCIÓN BUG

Este fallo lo hemos corregido cambiando en la ruta src/main/resources/templates/editFilmPage.html esta línea:

```
<p>
    <button id ="cancel" class="ui button" onclick="location.href='/film/{{film.id}}'; return false;">Cancel</button>
    <input class="ui button orange" type="submit" id="Save" value="Save" />
</p>
```

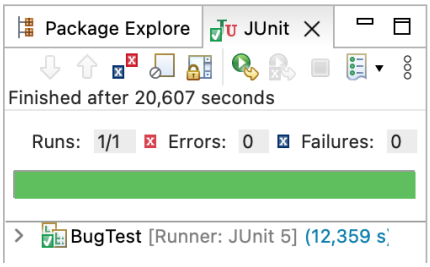
por esta:

```
<p>
    <button id ="cancel" class="ui button" onclick="location.href='/films/{{film.id}}'; return false;">Cancel</button>
    <input class="ui button orange" type="submit" id="Save" value="Save" />
</p>
```

En definitiva, hemos añadido una “s” a la URL del botón con id=”cancel”. Hemos pasado de /film/… a /films/…

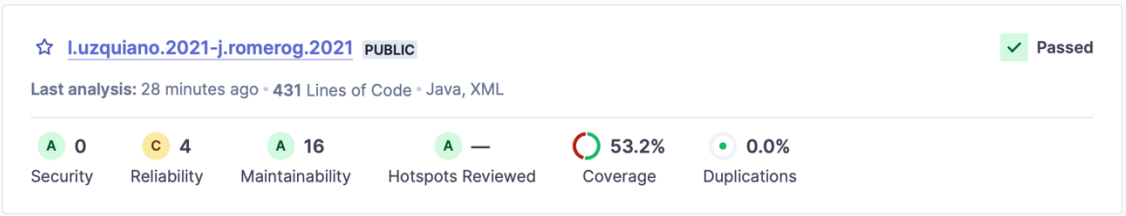
## TEST DE REGRESIÓN – PASA

A continuación, mostramos una captura de pantalla donde observamos que el test proporcionado anteriormente pasa correctamente tras la modificación del fichero “editFilmPage.html” descrita en el apartado anterior.



# SONAR

Comenzamos mostrando el DashBoard después de haber pasado SonarQube por primera vez.




## ISSUES

### 1: SUSTITUIR SYSTEM.OUT.PRINTLN POR LOGGER

Esta issue destaca la importancia de utilizar un sistema de registro adecuado en el desarrollo de software, asegurando que los registros sean accesibles, uniformemente formateados, y seguros al tratar datos sensibles, en lugar de escribir directamente en las salidas estándar.

Además, es real, por tanto, reportamos la issue y la corregimos en el código de la clase "DatabaseInitializer.java" como mostramos a continuación.

Cabe añadir que aparece dos veces la misma issue. Tanto en el try como en el catch, y hemos corregido ambas.

Código original:

```
private boolean isTestingEnviroment(){
    try {
        Class.forName( className: "org.junit.jupiter.api.Test");
        System.out.println("TEST ENV");
        return true;
    } catch (ClassNotFoundException e) {
        System.out.println("PRODUCTION ENV");
        return false;
    }
}
```

Código arreglado:

```
Logger logger = Logger.getLogger(getClass().getName());
private boolean isTestingEnviroment(){
    try {
        Class.forName(className:"org.junit.jupiter.api.Test");
        logger.info(msg:"TEST ENV");
        return true;
    } catch (ClassNotFoundException e) {
        logger.info(msg:"PRODUCTION ENV");
        return false;
    }
}
```

### 2: SUSTITUIR "==" POR EQUALS

Esta issue nos indica que es preferible utilizar los métodos equals() para comparar los valores de objetos String o Integer y asegurarse de obtener resultados correctos, ya que comparar instancias con == o != puede llevar a resultados inesperados y errores en la lógica del programa.

Además, es real, por tanto, reportamos la issue y la corregimos en el código de la clase "FilmService.java" como mostramos a continuación.

Código original:

```
public Film save(Film film) {
    if (film.getTitle() == "") {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "The title is empty");
    }
    urlUtils.checkValidImageURL(film.getUrl());
    Film newFilm = repository.save(film);
    notificationService.notify("Film Event: Film with title="+newFilm.getTitle()+" was created");
    return newFilm;
}
```

Código arreglado:

```
public Film save(Film film) {
    if (film.getTitle().equals(anObject:"")) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, reason:"The title is empty");
    }
    urlUtils.checkValidImageURL(film.getUrl());
    Film newFilm = repository.save(film);
    notificationService.notify("Film Event: Film with title="+newFilm.getTitle()+" was created");
    return newFilm;
}
```

## 3: ELIMINAR VARIABLES NO UTILIZADAS

Esta issue corresponde a código muerto, ya que contribuye a una complejidad innecesaria y genera confusión al leer el código. Por lo tanto, la eliminamos para mantener la claridad y eficiencia.

Además, es real, por tanto, reportamos la issue y la corregimos eliminando la línea de creación de la variable.

Código original:

```
public void checkValidImageURL(String candidateURL){

    String otherUrl;

    // CHECK THAT URL HAS VALID FORMAT
    URL url;
    try {
        url = new URL(candidateURL);
```

Código arreglado:

```
public void checkValidImageURL(String candidateURL){

    // CHECK THAT URL HAS VALID FORMAT
    URL url;
    try {
        url = new URL(candidateURL);
```

## 4: ELIMINAR CÓDIGO COMENTADO

Esta issue corresponde, como la anterior, a código muerto, ya que contribuye a una complejidad innecesaria y genera confusión al leer el código. Por lo tanto, la eliminamos para mantener la claridad y eficiencia.

Además, es real, por tanto, reportamos la issue y la corregimos, eliminando el código comentado.

Código original:

```
} catch ( MalformedURLException ex) {
    throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "The url format is not valid");
    //System.out.println("The url format is not valid");
}
```

Código arreglado:

```
} catch ( MalformedURLException ex) {
    throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "The url format is not valid");
}
```

## 5: ELIMINAR IMPORTS NO USADOS

Esta issue corresponde, como las anteriores, a código muerto, ya que contribuye a una complejidad innecesaria y genera confusión al leer el código. Por lo tanto, la eliminamos para mantener la claridad y eficiencia.

Además, es real, por tanto, reportamos la issue y la corregimos, eliminando la línea donde se importa la librería no utilizada.

Código original:

```
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.server.ResponseStatusException;
import org.springframework.web.servlet.ModelAndView;

import es.codeurjc.ais.nitflex.film.Film;
import es.codeurjc.ais.nitflex.film.FilmService;

import es.codeurjc.ais.nitflex.DatabaseInitializer;
```

Código arreglado:

```
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.server.ResponseStatusException;
import org.springframework.web.servlet.ModelAndView;

import es.codeurjc.ais.nitflex.film.Film;
import es.codeurjc.ais.nitflex.film.FilmService;

@Controller
public class FilmWebController {
```

## 6: SUSTITUIR VARIABLES DUPLICADAS POR UNA CONSTANTE

Esta issue corresponde a la duplicación de cadenas literales en el código fuente. Esto ocurre cuando una misma cadena de texto aparece en múltiples lugares dentro del código, lo que dificulta la refactorización y el mantenimiento, ya que cualquier cambio en la cadena requeriría modificaciones en todos los lugares donde se encuentra, aumentando la posibilidad de introducir errores.

Además, es real, por tanto, corregimos la issue en el código añadiendo dos constantes distintas.

```
public static final String FILMS = "films";
public static final String MESSAGE = "message";
```

A continuación, mostramos el código antes y después de sustituir el String “films” por la constante “FILMS”.

## CASO 1: "SHOWFILMS"

Código original:

```
@GetMapping("/")
public String showFilms(Model model) {

    model.addAttribute( attributeName: "films", filmService.findAll());

    return "films";
}
```

Código arreglado:

```
@GetMapping("/")
public String showFilms(Model model) {

    model.addAttribute(FILMS, filmService.findAll());

    return FILMS;
}
```

## CASO 2: "SHOWFILM"

Código original:

```
@GetMapping("/films/{id}")
public String showFilm(Model model, @PathVariable long id) {

    Optional<Film> op = filmService.findOne(id);
    if(op.isPresent()) {
        Film film = op.get();
        model.addAttribute( attributeName: "film", film);
        return "film";
    }else {
        return "films";
    }
```

Código arreglado:

```
@GetMapping("/films/{id}")
public String showFilm(Model model, @PathVariable long id) {

    Optional<Film> op = filmService.findOne(id);
    if(op.isPresent()) {
        Film film = op.get();
        model.addAttribute( attributeName: "film", film);
        return "film";
    }else {
        return FILMS;
    }
```

## CASO 3: "EDITBOOK"

Código original:

```
@GetMapping("/editfilm/{id}")
public String editBook(Model model, @PathVariable long id) {

    Optional<Film> op = filmService.findOne(id);
    if(op.isPresent()) {
        Film film = op.get();
        model.addAttribute( attributeName: "film", film);
        return "editFilmPage";
    }else {
        return "films";
    }

}
```

Código arreglado:

```
@GetMapping("/editfilm/{id}")
public String editBook(Model model, @PathVariable long id) {

    Optional<Film> op = filmService.findOne(id);
    if(op.isPresent()) {
        Film film = op.get();
        model.addAttribute( attributeName: "film", film);
        return "editFilmPage";
    }else {
        return FILMS;
    }

}
```

A continuación, mostramos el código antes y después de sustituir el el String “message” por la constante “MESSAGE”.

## CASO 1: “REMOVEFILM”

Código original:

```
@GetMapping("/removefilm/{id}")
public String removeFilm(Model model, @PathVariable long id) {

    Optional<Film> op = filmService.findOne(id);
    if(op.isPresent()) {
        filmService.delete(id);
        Film removedFilm = op.get();
        model.addAttribute( attributeName: "error",  attributeValue: false);
        model.addAttribute( attributeName: "message",  attributeValue: "Film '"+removedFilm.getTitle()+"' deleted");
        return "message";
    }else {
        return "redirect:/";
    }

}
```

Código arreglado:

```
@GetMapping("/removefilm/{id}")
public String removeFilm(Model model, @PathVariable long id) {

    Optional<Film> op = filmService.findOne(id);
    if(op.isPresent()) {
        filmService.delete(id);
        Film removedFilm = op.get();
        model.addAttribute( attributeName: "error",  attributeValue: false);
        model.addAttribute(MESSAGE,  attributeValue: "Film '"+removedFilm.getTitle()+"' deleted");
        return MESSAGE;
    }else {
        return "redirect:/";
    }

}
```

## CASO 2: “HANDLEEXCEPTION”

Código original:

```
@ExceptionHandler({ResponseStatusException.class, BindException.class})
public ModelAndView handleException(Exception ex){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("message");
    modelAndView.addObject( attributeName: "error",  attributeValue: true);

    if(ex instanceof ResponseStatusException){
        ResponseStatusException resExp = (ResponseStatusException) ex;
        modelAndView.addObject( attributeName: "message", resExp.getReason());
    }else if(ex instanceof BindException){
        modelAndView.addObject( attributeName: "message",  attributeValue: "Field 'year' must be a number");
    }else{
        modelAndView.addObject( attributeName: "message", ex.getMessage());
    }

    return modelAndView;
}
```

Código arreglado:

```java
@ExceptionHandler({ResponseStatusException.class, BindException.class})
public ModelAndView handleException(Exception ex){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName(MESSAGE);
    modelAndView.addObject( attributeName: "error", attributeValue: true);

    if(ex instanceof ResponseStatusException){
        ResponseStatusException resExp = (ResponseStatusException) ex;
        modelAndView.addObject(MESSAGE, resExp.getReason());
    }else if(ex instanceof BindException){
        modelAndView.addObject(MESSAGE, attributeValue: "Field 'year' must be a number");
    }else{
        modelAndView.addObject(MESSAGE, ex.getMessage());
    }

    return modelAndView;
}
```

## 7: ANOTACIÓN @AUTOWIRED EN VARIABLE

Esta issue corresponde con el uso de @Autowired en una variable. Al realizar las inyecciones de dependencias en variables, es posible que se creen objetos en estados inválidos, además de que son incompatibles con variables finales. Esto puede dificultar el testing de la aplicación, por lo que se recomienda usar la anotación con el constructor de la clase en la que se realice la inyección, en donde se dará valor a la variable inyectada.

Puesto que FilmService y FilmRepository no contienen variables finales, consideramos que esta issue no es un fallo. Por ello, la reportamos en SonarQube como “falso positivo”.

En el código se dan tres casos:

### CASO 1: “DATABASEINITIALIZER.JAVA”

```java
@Component
public class DatabaseInitializer {
    2 usages
    Logger logger = Logger.getLogger(getClass().getName());

    @Autowired
    private FilmRepository filmRepository;
```

### CASO 2: “FILMRESTCONTROLLER.JAVA”

```java
@RestController
@RequestMapping("/api/films")
public class FilmRestController {

    @Autowired
    private FilmService filmfilmService;
```

### CASO 3: “FILMWEBCONTROLLER.JAVA”

```java
@Controller
public class FilmWebController {
    4 usages
    public static final String FILMS = "films";
    6 usages
    public static final String MESSAGE = "message";

    @Autowired
    private FilmService filmService;
```

## 8: INSTANCEOF SEGUIDA DE CASTEO

Esta issue corresponde con la asignación de una variable casteada justo después de un instanceof. A partir de java 16, instanceof es capaz de castear y asignar valores a variables usando pattern matching, haciendo el código original verboso. Eliminando estas líneas de código puede hacer el código más mantenible, por lo que lo consideramos una issue real.

Esta issue fué localizada en la clase “FilmWebController.java”, en el manejador de excepciones de tipo “ResponseStatusException”

Código original:

```
if(ex instanceof ResponseStatusException){
    ResponseStatusException resExp = (ResponseStatusException) ex;
    modelAndView.addObject(MESSAGE, resExp.getReason());
```

Código arreglado:

```
if(ex instanceof ResponseStatusException resExp){
    modelAndView.addObject(MESSAGE, resExp.getReason());
```

Para finalizar, mostramos el DashBoard después de haber pasado SonarQube tras haber reportado y corregido todos los issues: