Practica 2 AIS

Autores: Raúl Sánchez Benítez, Andrés Muñoz Muñoz.

Asignatura: AMPLIACION DE INGENIERIA DEL SOFTWARE

Curso: 2024-2025

Índice

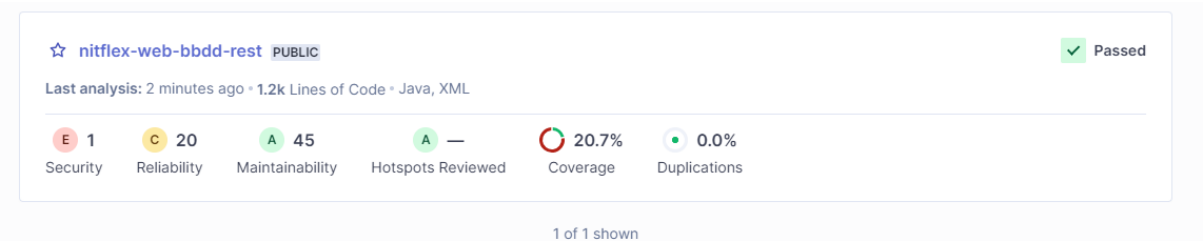
Introducción

En esta parte de la práctica, se ha realizado un análisis del código de la parte anterior, junto con el de la primera práctica, utilizando SonarQube para identificar y corregir errores de calidad y seguridad.

A lo largo del documento, se presentan los errores detectados, sus soluciones y el impacto de las correcciones en el proyecto.

Captura de pantalla del Dashboard

Aquí mostramos la captura de pantalla del dashboard:

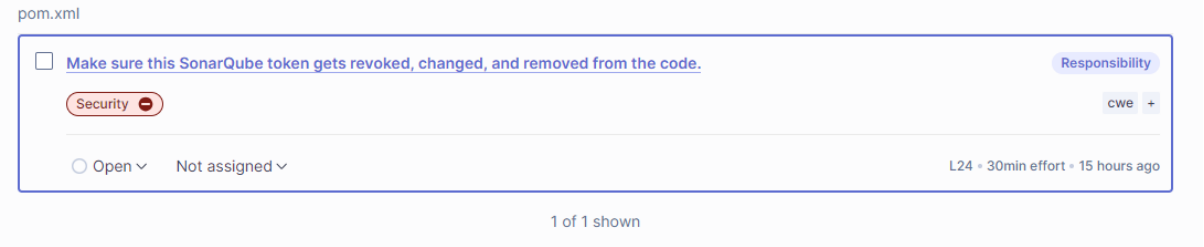


Corrección de errores

Vamos a empezar a corregir los issues:

Errores de Tipo Blocker

Empecemos con los errores de tipo blocker:

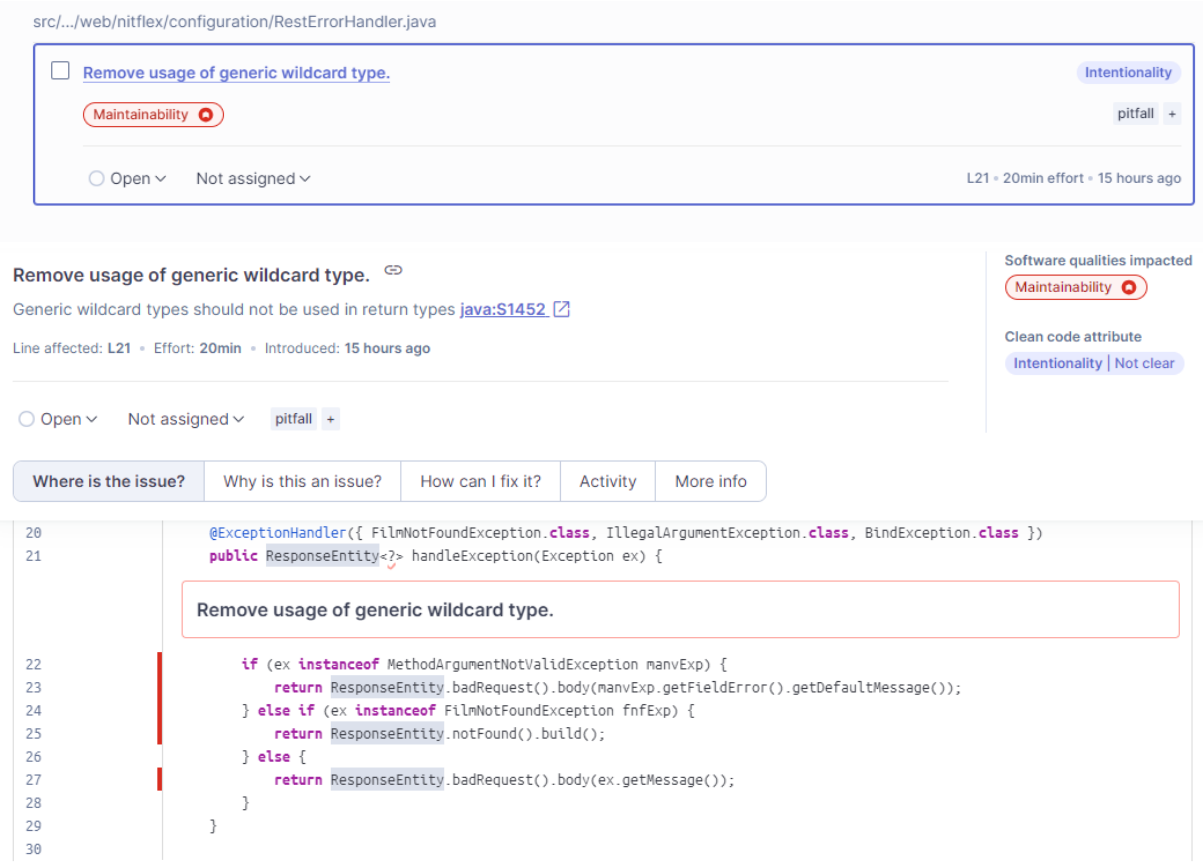


Solamente tenemos este error de seguridad, pero no podemos corregirlo debido a que nos pide borrar el token de sonar, pero lo necesitamos para poder ejecutar el proyecto y que sonar lo pueda analizar. Lo voy a marcar como un falso positivo porque es algo necesario para el análisis.

Errores de Tipo High

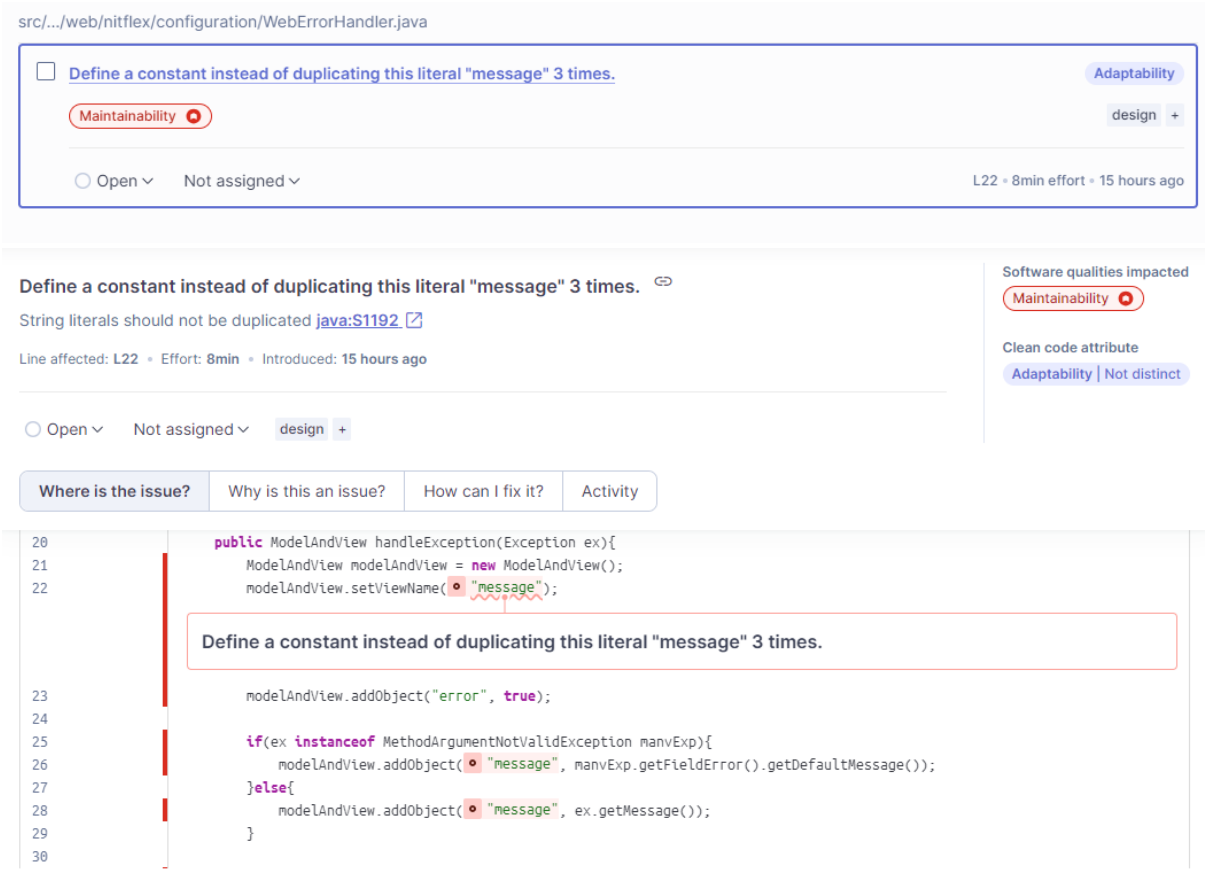
Pasemos a los errores de tipo High:

En SonarCube nos marca que tenemos 8, vamos a verlos:



Nos da error porque devuelve un tipo de datos genérico, y esto se soluciona cambiándolo por String:

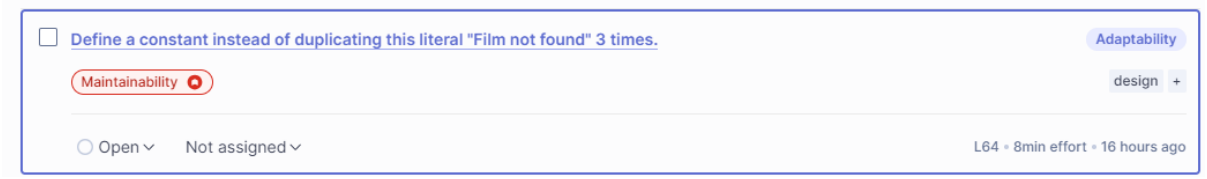
```java
@ExceptionHandler({ FilmNotFoundException.class, IllegalArgumentException.class, BindException.class })
public ResponseEntity<String> handleException(Exception ex) {
    if (ex instanceof MethodArgumentNotValidException manvExp) {
        return ResponseEntity.badRequest().body(manvExp.getFieldError().getDefaultMessage());
    } else if (ex instanceof FilmNotFoundException fnfExp) {
        return ResponseEntity.notFound().build();
    } else {
        return ResponseEntity.badRequest().body(ex.getMessage());
    }
}
```



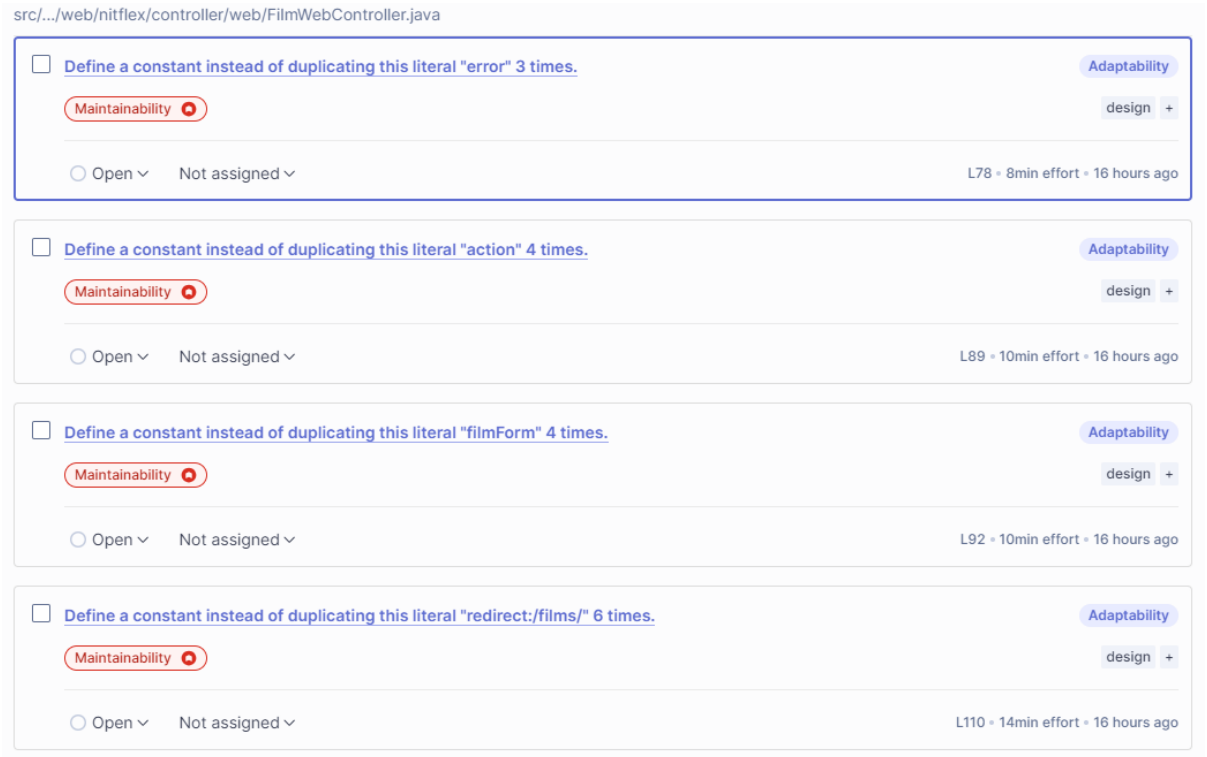
Este error nos dice que, en vez de duplicarlo, creemos una variable.

Para ello creamos una variable que evite los duplicados.

```java
@ExceptionHandler({FilmNotFoundException.class, IllegalArgumentException.class, BindException.class})
public ModelAndView handleException(Exception ex){
    ModelAndView modelAndView = new ModelAndView();
    String a = "message";
    modelAndView.setViewName(a);
    modelAndView.addObject("error", true);

    if(ex instanceof MethodArgumentNotValidException manvExp){
        modelAndView.addObject(a, manvExp.getFieldError().getDefaultMessage());
    }else{
        modelAndView.addObject(a, ex.getMessage());
    }

    return modelAndView;
}
```



Aqui se nos marca el mismo error que el anterior. Para corregirlo, hemos creado una variable que guarde el mensaje y lo hemos sustituido.

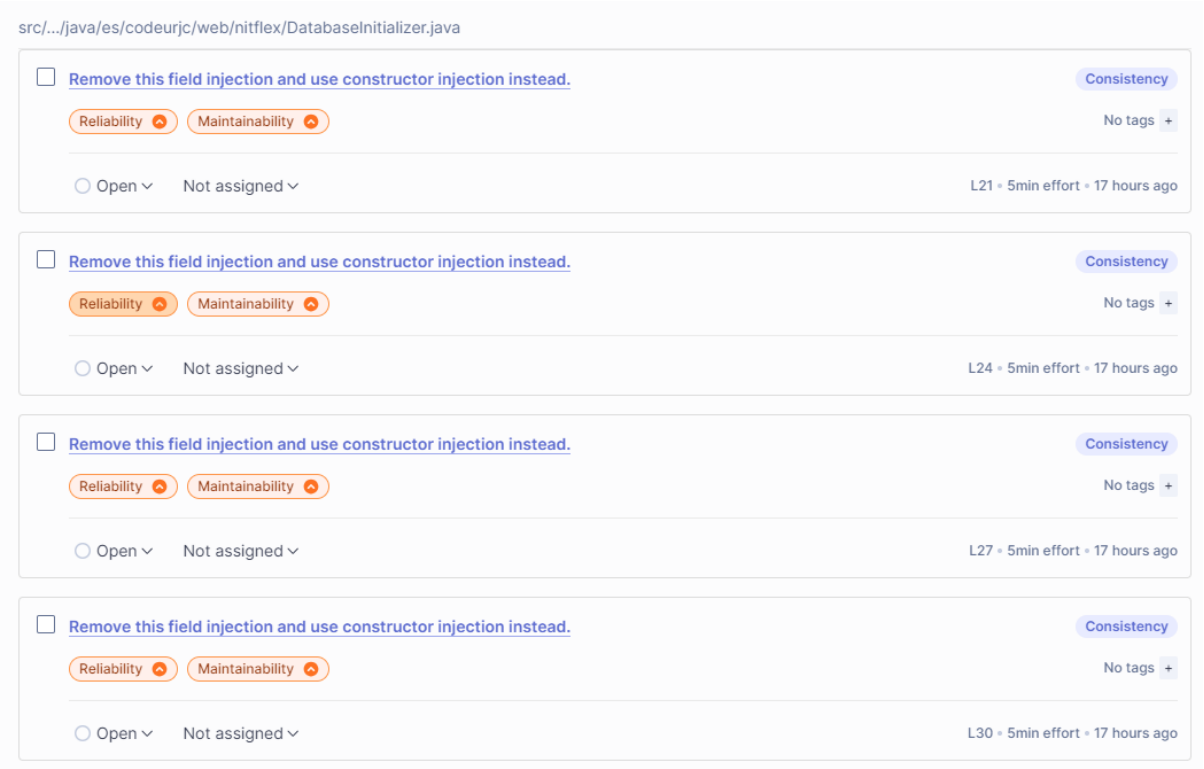


Aqui se nos muestran otros 4 errores más de este mismo tipo.

```
private String filmNotFound = "Film not found";
private String error = "error";
private String action = "action";
private String filmForm =  "filmForm";
private String redirectFilms = "redirect:/films/";
```

Con esto finalizaríamos la corrección de errores de tipo High y vamos a pasar a los errores de tipo Medium.

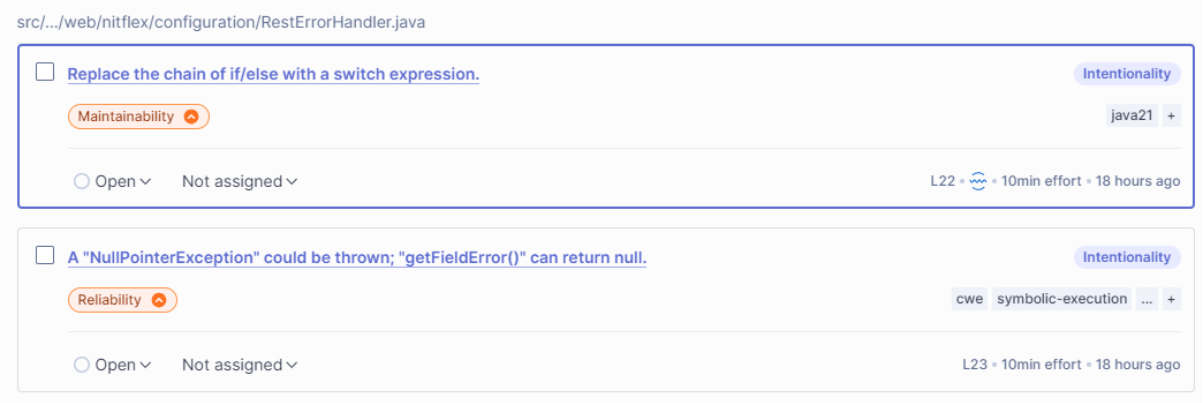
Errores de Tipo Medium



Vamos a empezar por estos que nos piden que borremos la anotación @Autowired y usemos una inyección por constructor:

```java
private final FilmService filmService;
private final ReviewService reviewService;
private final UserRepository userRepository;
private final ImageUtils imageUtils;

public DatabaseInitializer(FilmService filmService,
ReviewService reviewService,
UserRepository userRepository,
ImageUtils imageUtils) {
    this.filmService = filmService;
    this.reviewService = reviewService;
    this.userRepository = userRepository;
    this.imageUtils = imageUtils;
}
```

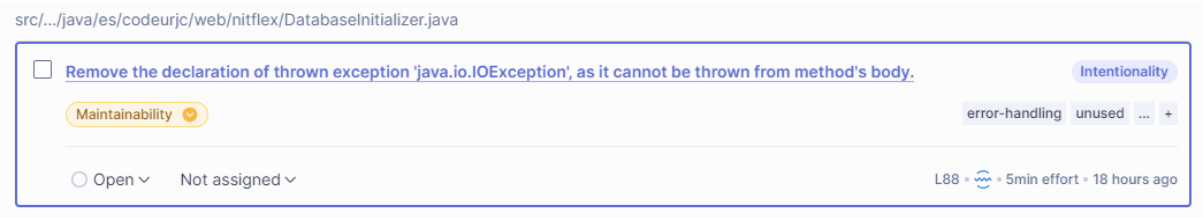
Aquí hemos borrado la anotación @Autowired y lo hemos sustituido por este fragmento de código, porque SonarQube pide cambiarlo en variables por inyección en el constructor para que el código sea más seguro, fácil de probar y claro.



Estos dos errores me piden que use un switch y que arregle una NullPointerException.

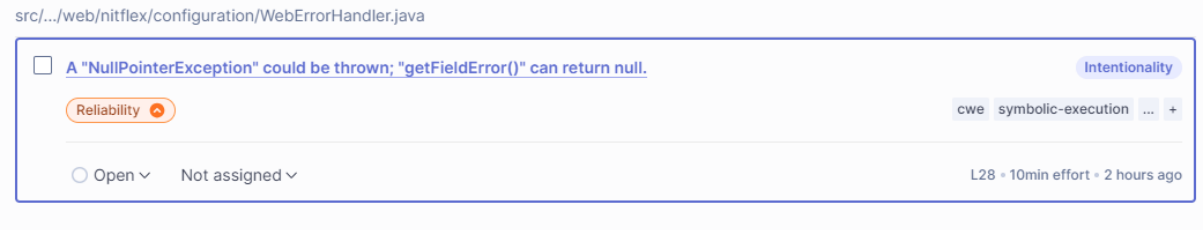
Con este fragmento de código hemos arreglado ambos problemas, con el switch todo queda más claro y estructurado, y con el @SuppressWarnings evitamos la excepción de puntero nulo.

```java
@SuppressWarnings("null")
@ExceptionHandler({ FilmNotFoundException.class, IllegalArgumentException.class, BindException.class })
public ResponseEntity<String> handleException(Exception ex) {
    switch (ex){
        case MethodArgumentNotValidException manvExp:
            return ResponseEntity.badRequest().body(manvExp.getFieldError().getDefaultMessage());
        case FilmNotFoundException fnfExp:
            return ResponseEntity.notFound().build();
        default:
            return ResponseEntity.badRequest().body(ex.getMessage());

    }
}
```

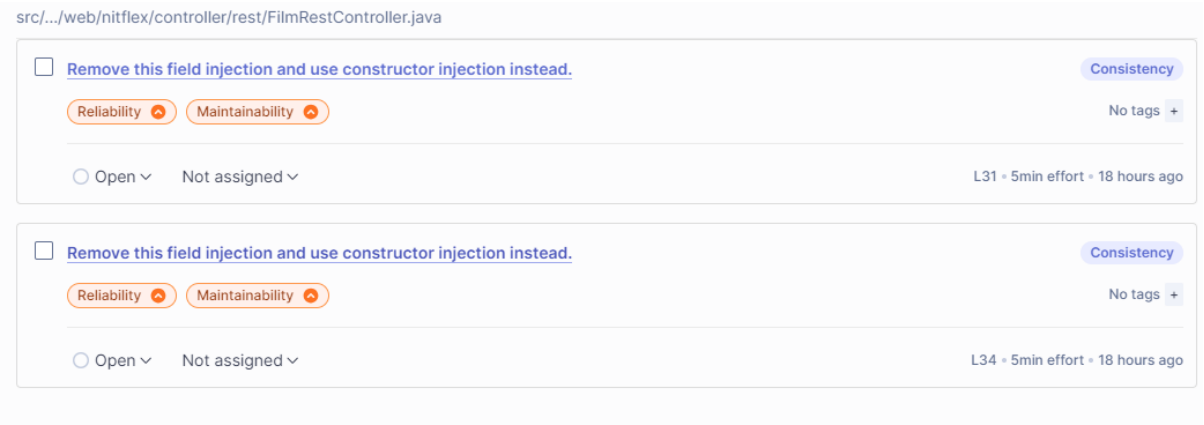


Nos vuelve a pedir que quitemos @Autowired, así que vamos a usar la misma metodología para solucionarlo.

```java
private final UserComponent userComponent;

public UserModelAttributes(UserComponent userComponent) {
    this.userComponent = userComponent;
}
```



Aqui se nos vuelve a presentar el problema con NullPointerException, así que vamos a volver a aplicar la misma solución de antes, es decir, añadir @SuppressWarnings, sabemos que no es la más óptima porque realmente no soluciona el problema, pero así SonarCube deja de marcar el error.

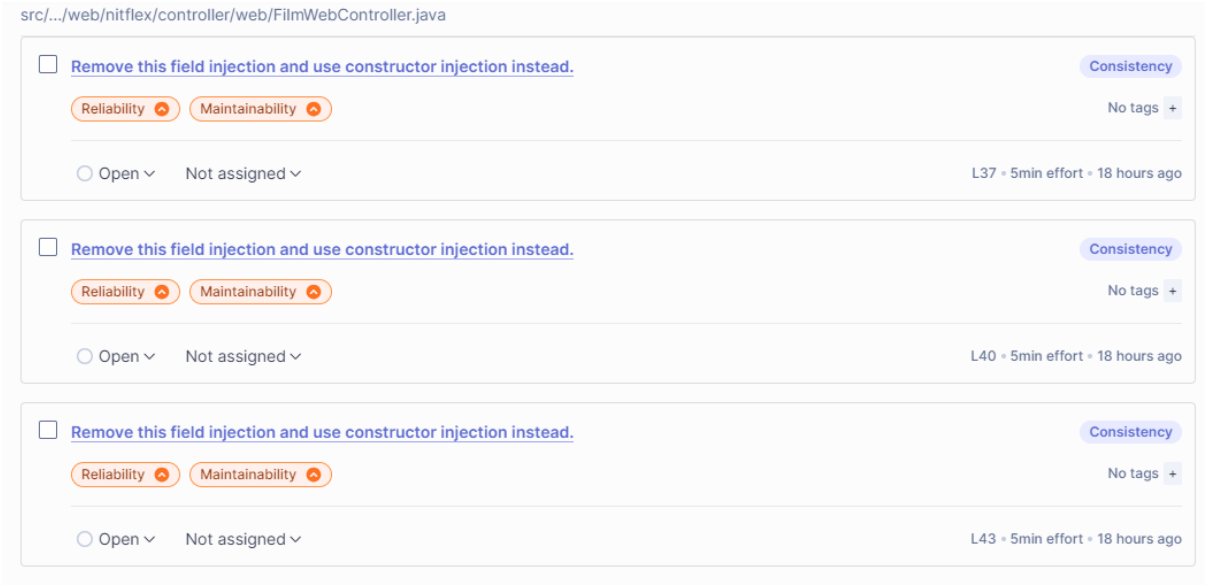


Aqui se nos vuelve a pedir quitar la anotación @Autowired, así que vamos a volver a aplicar la misma metodología.

```
private final FilmService filmService;
private final ReviewService reviewService;

public FilmRestController(FilmService filmService,
ReviewService reviewService) {
    this.filmService = filmService;
    this.reviewService = reviewService;
}
```

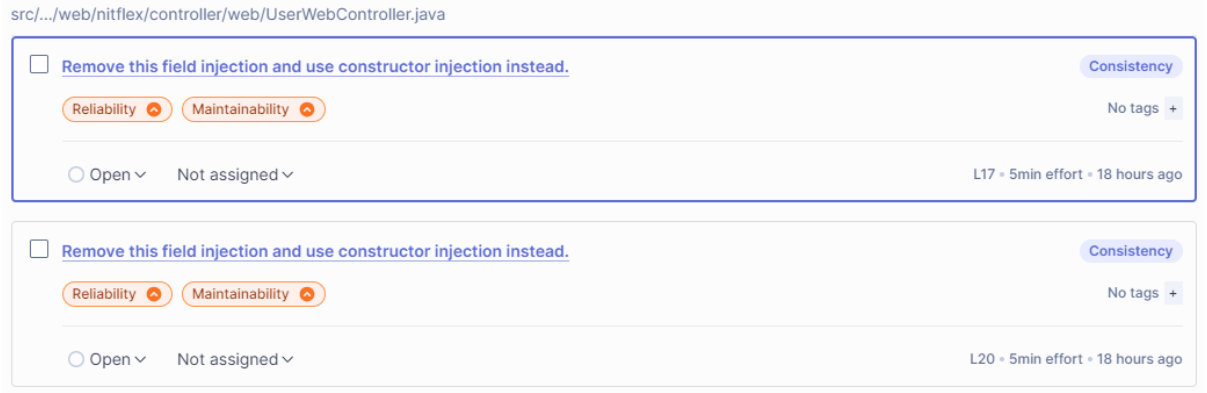
Otra vez lo mismo de quitar @Autowired.

```
private final UserService userService;
private final UserMapper userMapper;

public UserRestController(UserService userService,
UserMapper userMapper) {
    this.userService = userService;
    this.userMapper = userMapper;
}
```



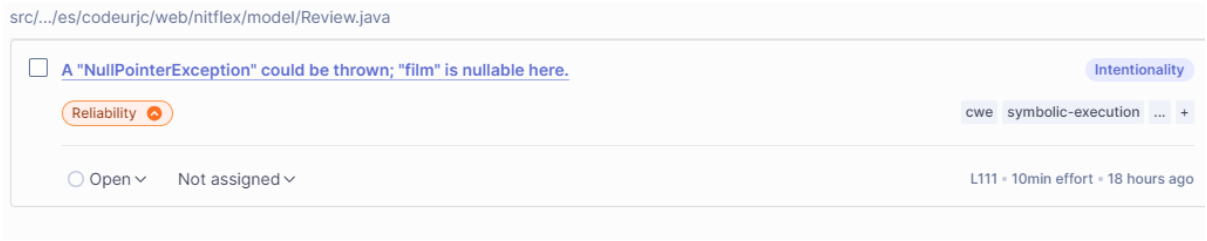
Se vuelve a pedir quitar @Autowired otra vez.

```
private final FilmService filmService;
private final FavoriteFilmService favoriteFilmService;
private final ReviewService reviewService;

public FilmWebController(FilmService filmService,
FavoriteFilmService favoriteFilmService,
ReviewService reviewService) {
    this.filmService = filmService;
    this.favoriteFilmService = favoriteFilmService;
    this.reviewService = reviewService;
}
```



Se vuelve a pedir quitar @Autowired otra vez.

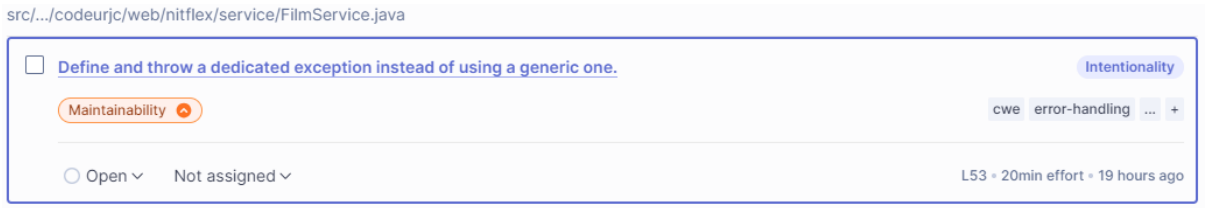
```java
private final UserComponent userComponent;
private final UserService userService;

public UserWebController(UserComponent userComponent,
UserService userService) {
    this.userComponent = userComponent;
    this.userService = userService;
}
```



Aquí se nos muestra que el método puede devolver un NullPointerException porque film puede ser null, para solucionar esto vamos a usar este fragmento de código:

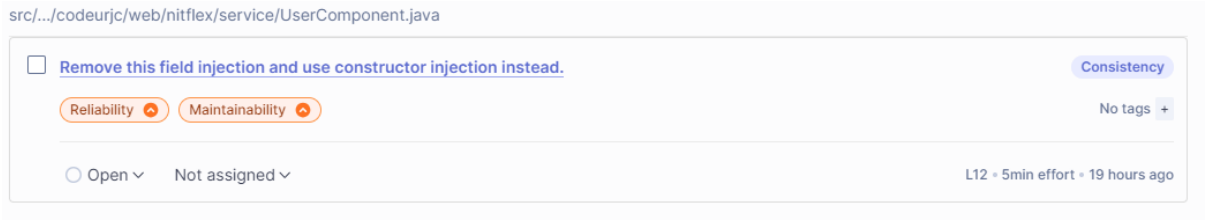
```java
return ((film == null) && (review.film == null)) || ((film != null) && (film.equals(review.film)));
```

Hemos añadido (film != null) para evitar el NullPointerException.



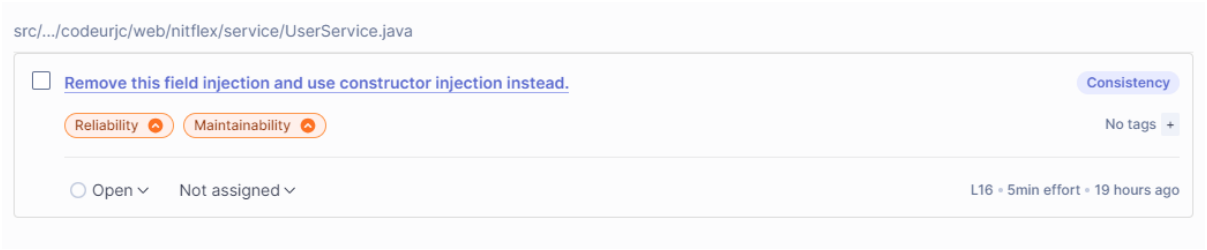
Aquí se nos pide usar una excepción dedicada en vez de una tan genérica. Lo hemos arreglado usando una SQLException en lugar de la RuntimeException que se estaba usando.

```java
public InputStream getPosterFile(long id) throws SQLException {
    Film film = filmRepository.findById(id)
            .orElseThrow(() -> new FilmNotFoundException(id));
    Blob blob = film.getPosterFile();
    try {
        return blob.getBinaryStream();
    } catch (SQLException e) {
        throw new SQLException(reason:"Error getting image from database", e);
    }
}
```



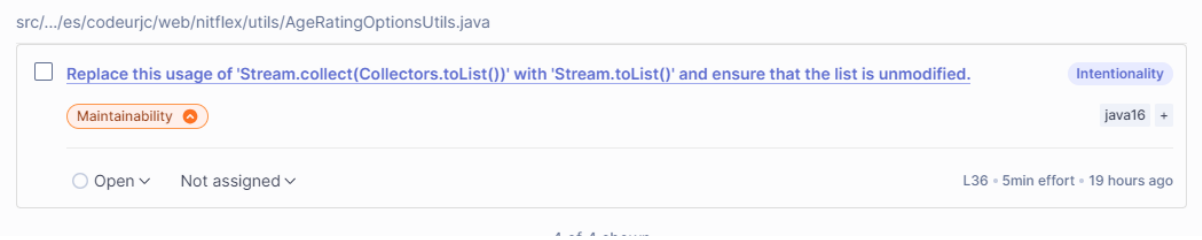
Otra vez nos pide que quitemos @Autowired.

```java
private final UserRepository userRepository;

public UserComponent(UserRepository userRepository) {
    this.userRepository = userRepository;
}
```



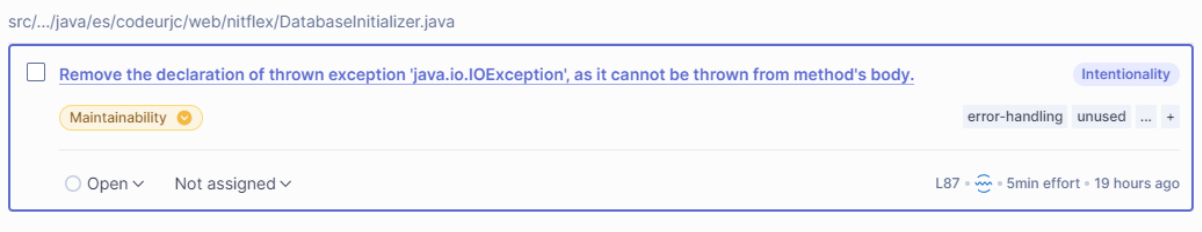
Se vuelve a pedir quitar @Autowired otra vez.

```java
private final UserRepository userRepository;

public UserService(UserRepository userRepository) {
    this.userRepository = userRepository;
}
```



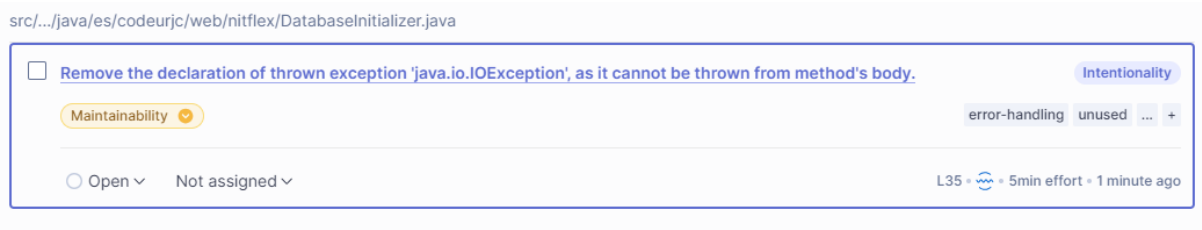
Este error nos dice que cambiemos nuestro código para evitar el uso de .collect(Collectors.toList()). Esto lo hemos solucionado cambiando el código para añadir .toList()

```
public static Collection<AgeRatingOption> getAgeRatingOptions(String selectedAgeRating) {
    return Stream.of(AgeRating.values())
        .map(
            ageRating -> new AgeRatingOption(
                ageRating.getDescription(),
                ageRating.getDescription().equals(selectedAgeRating)
        ))
        .toList();
}
```



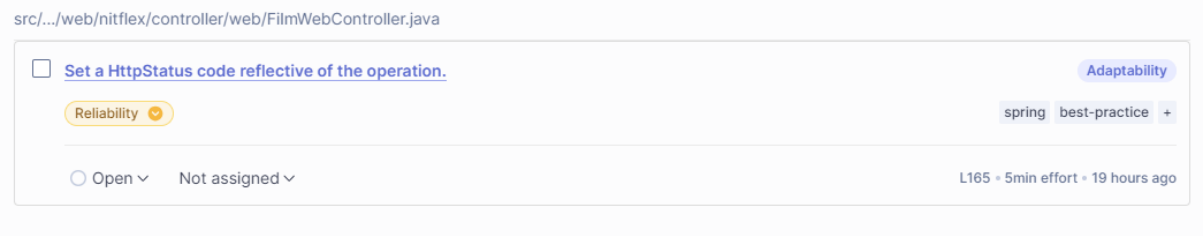
Este issue nos decía que quitásemos el throws porque el método no era capaz de lanzarlo, así que hemos borrado el throws porque no servía de nada porque en el cuerpo de la función nunca se lanzaba una excepción.

```
private FilmDTO saveFilmWithURLImage(CreateFilmRequest film, String localpath){
    return filmService.save(film, imageUtils.localImageToBlob(localpath));
}
```



Hemos quitado el throws IOException por el mismo motivo que en el issue anterior.

```
@PostConstruct
public void init(){
```




```java
@GetMapping("/films/{id}/poster")
public ResponseEntity<Object> getMethodName(@PathVariable long id) throws IOException {
    Resource poster;
    try {
        poster = new InputStreamResource(filmService.getPosterFile(id));
    } catch (Exception e) {
        ClassPathResource resource = new ClassPathResource(path:"static/images/no-image.png");
        byte[] imageBytes = resource.getInputStream().readAllBytes();
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
            .header(HttpHeaders.CONTENT_TYPE, ...headerValues:"image/jpeg").body(imageBytes);
    }
    return ResponseEntity.ok().header(HttpHeaders.CONTENT_TYPE, ...headerValues:"image/jpeg").body(poster);
}
```

Para este error se nos pedía que el mensaje de httpStatus fuese el adecuado, y hemos cambiado de .ok() (200) a NOT_FOUND (404) para que refleje correctamente que no se encontró la imagen q buscaba.

A partir de la versión 16 de java, podemos cambiar el ‘instanceof Film’ por instanceof Film f lo que simplifica el código y lo hace más comprensible.

```java
@Override
public boolean equals(Object o){
    if(o == null) return false;
    if(o == this) return true;
    if(o instanceof Film){
        Film f = (Film) o;
        return f.getId().equals(this.id);
    }
    return false;
}
```

Esta sería la solución:

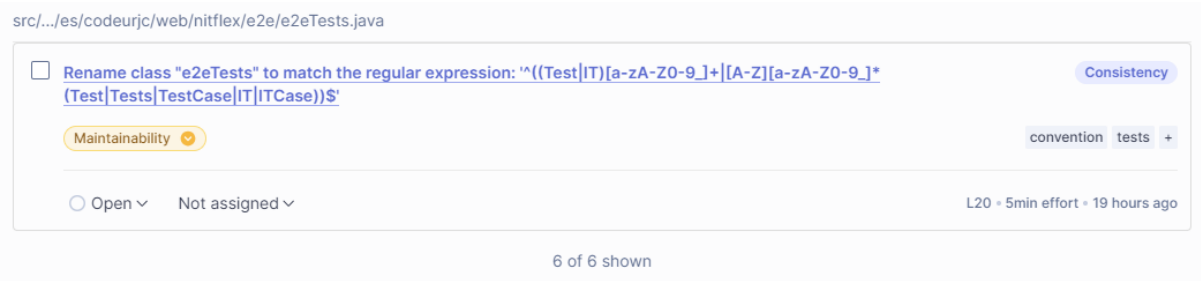
```java
@Override
public boolean equals(Object o){
    if(o == null) return false;
    if(o == this) return true;
    if(o instanceof Film f){
        return f.getId().equals(this.id);
    }
    return false;
}
```

Aquí se nos pide que cambiemos el nombre de una variable que usa '_'. Entonces pasa de ser created_at a createdAt.

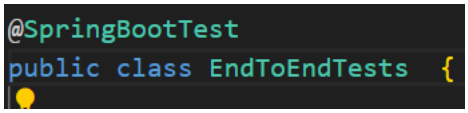
```java
@CreationTimestamp
private Date createdAt;
```

Este es el mismo error anterior, el nombre del método usaba '_' y lo hemos cambiado.

```java
public Date getcreatedAt() {
    return createdAt;
}
```

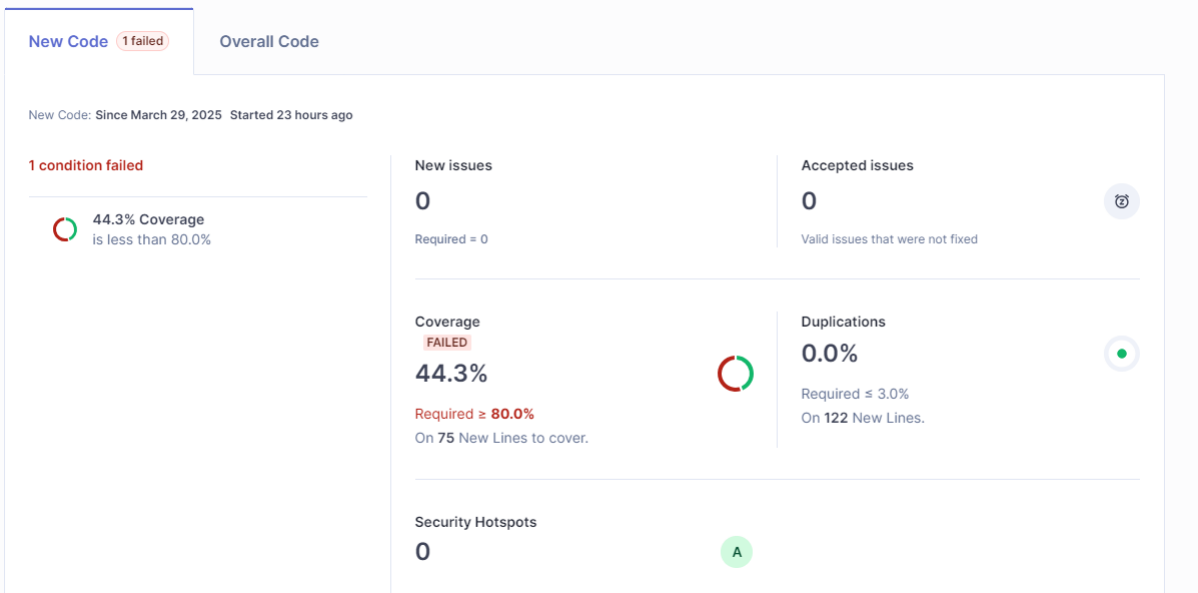


Este error pide que renombremos la clase e2eTests, pondremos EndToEndTests.


@SpringBootTest
public class EndToEndTests {

Vista Previa Final

Tras haber arreglado todos los errores, este es el dashboard en la pestaña New Code



Y en la vista previa de los proyectos se ve así