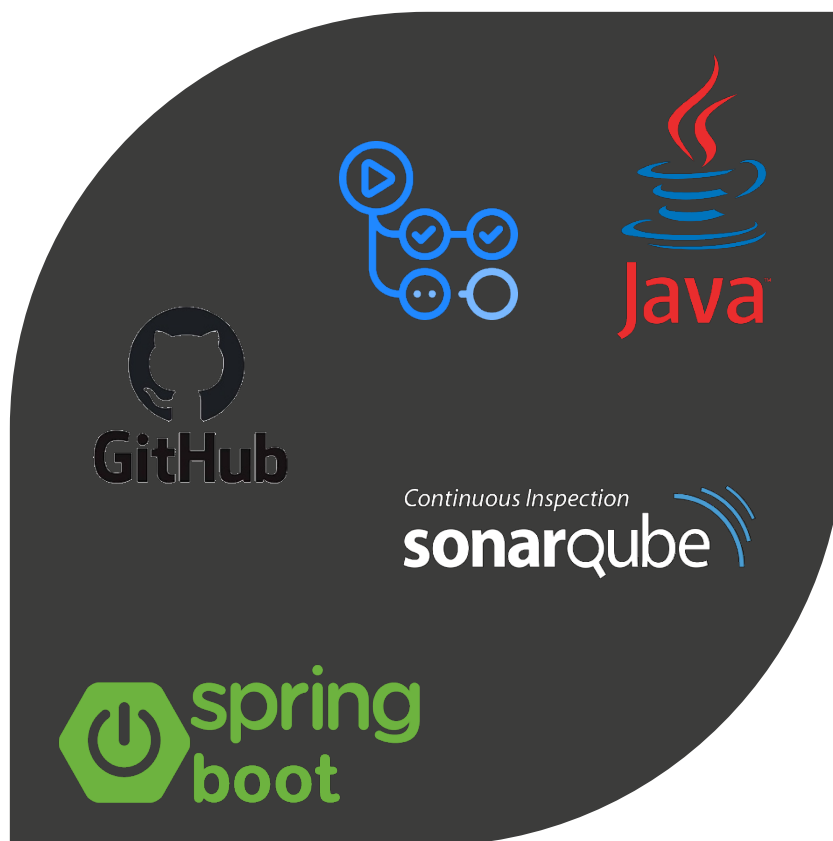


TRABAJO FINAL

INTEGRACIÓN CONTINUA

EN EL DESARROLLO ÁGIL

EDISON IGNACIO DE LA CRUZ RODRÍGUEZ



Contenido

Introducción	3
Desarrollo Práctica Inicial tema 5.....	4
Creación de repositorio y despliegue del Moviecards-Service.....	5
Nueva Versión de la Aplicación MovieCards para conectarse a otro servicio.....	6
Modificación del microservicio Moviecards-Service	7
Despliegue a nuevo entorno de Pre-Producción.....	8
Nueva Versión de la Aplicación para Manejar la Fecha de Fallecimiento.....	9
Garantía de Calidad con SonarQube	12
Enlaces a Repositorios y Video	14
Conclusión	15

Introducción

La integración continua (CI, por sus siglas en inglés) es una práctica fundamental en el desarrollo ágil de software que busca mejorar la eficiencia y calidad de los procesos de desarrollo. Esta metodología permite identificar y solucionar problemas de manera rápida y efectiva, reduciendo el riesgo de errores y conflictos en el código que, de no detectarse a tiempo, podrían convertirse en problemas críticos y de alto costo para su solución.

En este trabajo, se presenta la implementación de los requerimientos del trabajo final sobre la aplicación MovieCards, la cual permite registrar actores y películas, así como generar relaciones entre ellos. Además, se aborda la práctica del tema 5, que sirvió como base para el desarrollo de esta solución.

El proyecto incorpora mejoras significativas, como la integración con un servicio independiente (moviecards-service) y la adición de funcionalidades adicionales, como el manejo de la fecha de fallecimiento de los actores. También se implementaron pruebas unitarias, de integración y funcionales (end-to-end), se desplegó la aplicación y su servicio en Azure, y se aseguró el cumplimiento de los requisitos técnicos establecidos. Finalmente, se aplicó un control de calidad del código mediante herramientas como SonarQube, reduciendo los errores críticos a su mínima expresión y garantizando un producto robusto y confiable.

Desarrollo Práctica Inicial tema 5

Como requisito previo para el trabajo final, se completó la práctica del tema 5, que consistió en desplegar la aplicación MovieCards. Esta aplicación permite registrar actores y películas, y establecer relaciones entre ellos. Durante esta fase, se llevaron a cabo las siguientes acciones:

1. Instalación de la infraestructura necesaria:

- Creación de cuentas en GitHub y Azure.
- Instalación de Docker y configuración de un contenedor con SonarQube para el análisis estático del código.

2. Creación del repositorio:

- Se generó un repositorio local a partir del código proporcionado por el profesor.
- El repositorio se subió a GitHub para su gestión y control de versiones.

3. Diseño del flujo de trabajo (workflow):

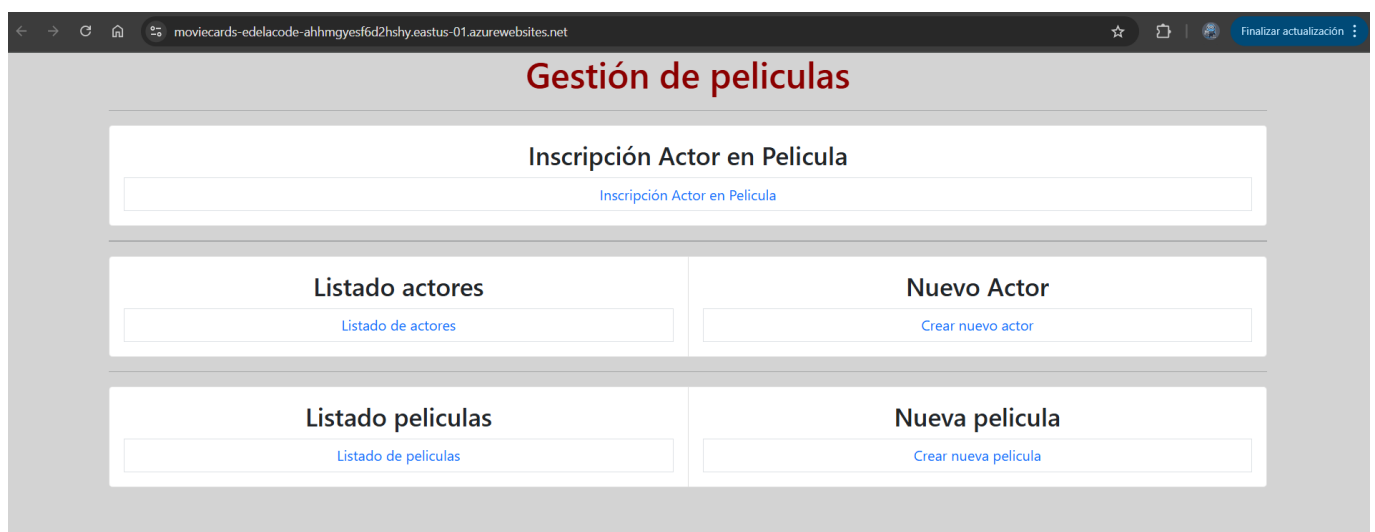
- Se configuró un pipeline de integración continua utilizando GitHub Actions, definiendo las etapas de construcción, pruebas y despliegue.

4. Segunda versión de la aplicación:

- Se realizaron modificaciones en la interfaz de usuario, ajustando colores y estilos para mejorar la experiencia del usuario.

La aplicación fue desplegada en Azure y quedó disponible en la siguiente URL:

<https://moviecards-edelacode-ahhmgyesf6d2hshy.eastus-01.azurewebsites.net/>



Creación de repositorio y despliegue del Moviecards-Service

Se creó un nuevo repositorio en GitHub llamado moviecards-service, el cual contiene el código del servicio que será consumido por la aplicación principal. Para desplegar este servicio en Azure, se realizaron los siguientes pasos:

1. Creación de una nueva aplicación web en Azure:
 5. Se configuró una aplicación web en Azure, conectada directamente al repositorio de GitHub para facilitar el despliegue continuo.
2. Configuración de GitHub Actions:
 6. Se implementó un flujo de trabajo (workflow) en un archivo YAML para automatizar el despliegue del servicio en Azure. Este workflow incluye las siguientes etapas:
 7. **Build:** Compilación del código del servicio.
 8. **Test:** Ejecución de pruebas unitarias y de integración.
 9. **Deploy:** Despliegue automático en Azure.

El servicio quedó funcionando en la siguiente URL:

<https://moviecards-service-edelacode-facuerczdsbwbygs.eastus-01.azurewebsites.net/>

The screenshot displays the GitHub Actions interface for the repository 'uosip312 / moviecards-service'. The workflow, titled 'Creado trabajo de despliegue automático en Azure #3', is shown as successful. The summary section indicates it was triggered via push 3 days ago, with a status of 'Success', a total duration of '1m 56s', and '1' artifact. The workflow steps are listed as 'Build and Package' (25s), 'Test' (28s), and 'deploy' (39s). The 'Artifacts' section shows a single artifact named 'moviecards-service-java' with a size of '33.1 MB'.

Name	Size
moviecards-service-java	33.1 MB

Nueva Versión de la Aplicación MovieCards para conectarse a otro servicio

Se realizó una nueva versión de la aplicación MovieCards, modificando su arquitectura para que, en lugar de ser una aplicación monolítica, se conecte al microservicio moviecards-service desplegado en Azure. Para ello, se siguieron los siguientes pasos:

- **Creación de un milestone en GitHub:**

1. Se creó un milestone llamado "Utilizar servicio como backend" para organizar y registrar los cambios.

- **Creación de issues y modificaciones en el código:**

1. Modificar el código de la aplicación en src/main:
 - 1.1. Se adaptaron las clases ActorServiceImpl y MovieServiceImpl para consumir el servicio backend.
 - 1.2. Se implementaron llamadas HTTP al servicio moviecards-service para gestionar actores y películas.
2. Modificar el código de las pruebas en src/test:
 - 2.1. Se ajustaron las pruebas unitarias y de integración para validar el funcionamiento del servicio backend.

La aplicación sigue funcionando en la misma URL:

<https://moviecards-edelacode-ahhmgyesf6d2hshy.eastus-01.azurewebsites.net/>

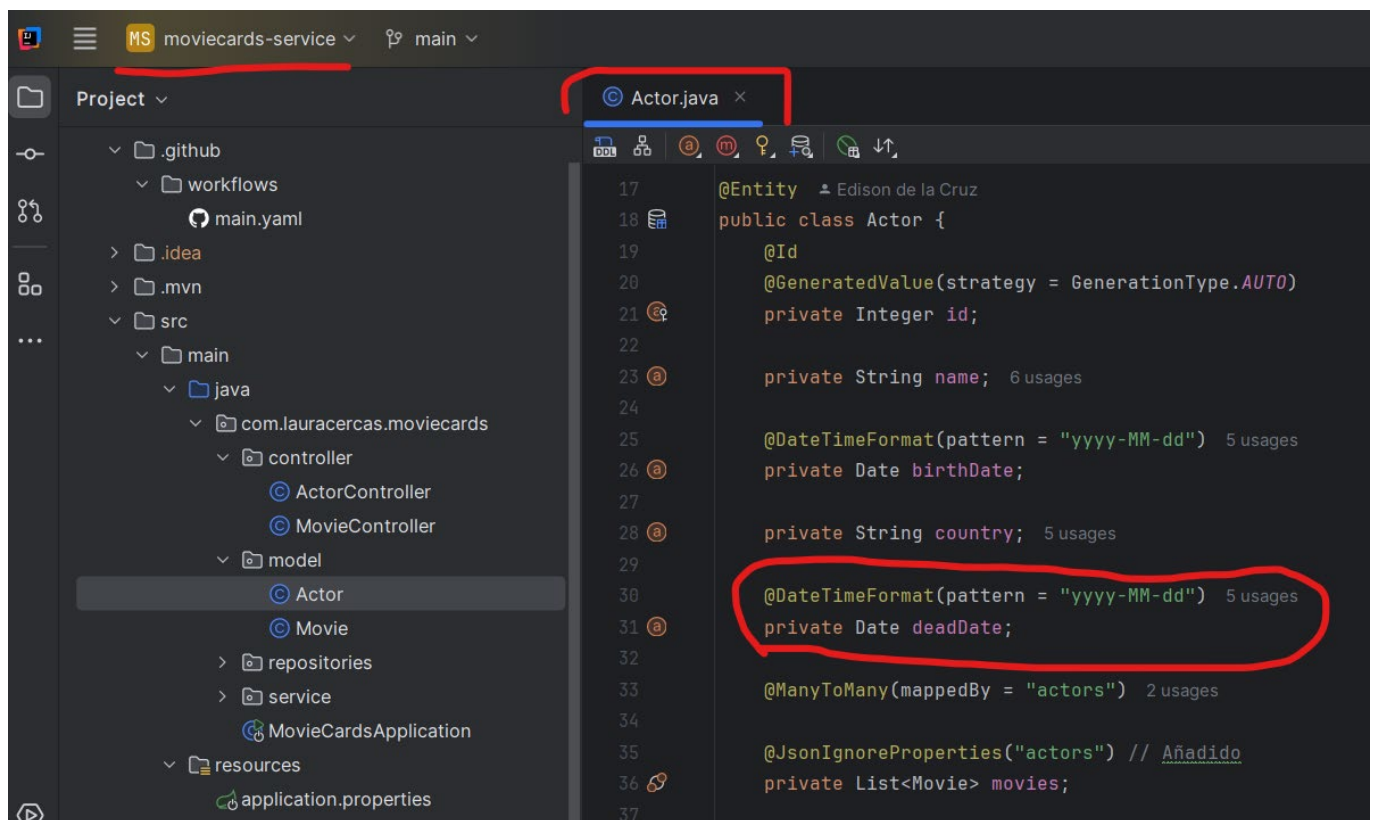
```
13 public class ActorServiceImpl implements ActorService {
14
15     private static final String URL = URL_BASE + "/actors"; 4 usages
16
17     private final RestTemplate restTemplate; 5 usages
18
19     public ActorServiceImpl(RestTemplate restTemplate) { Edison de la Cruz
20         this.restTemplate = restTemplate;
21     }
22
23     @Override 5 usages Edison de la Cruz
24     public List<Actor> getAllActors() {
25         Actor[] actors = restTemplate.getForObject(URL, Actor[].class);
26         assert actors != null;
27         return List.of(actors);
28     }
29 }
```

```
13 public class MovieServiceImpl implements MovieService {
14
15     private static final String URL = URL_BASE + "/movies"; 4 usages
16
17     private final RestTemplate restTemplate; 5 usages
18
19     public MovieServiceImpl(RestTemplate restTemplate) { Edison de la Cruz
20         this.restTemplate = restTemplate;
21     }
22
23     @Override 5 usages Edison de la Cruz
24     public List<Movie> getAllMovies() {
25         Movie[] movies = restTemplate.getForObject(URL, Movie[].class);
26         assert movies != null;
27         return List.of(movies);
28     }
29 }
```

Modificación del microservicio Moviecards-Service

Se modificó el código del repositorio moviecards-service para añadir un nuevo atributo en la clase Actor, llamado deadDate, que almacena la fecha de fallecimiento del actor. Este cambio incluyó:

- **Modificación del modelo de datos:**
 1. Se añadió el campo deadDate en la clase Actor.
 2. Se actualizaron los métodos relacionados con la creación y actualización de actores para manejar este nuevo atributo.
- **Despliegue en Azure:**
 1. Los cambios fueron desplegados nuevamente en Azure utilizando GitHub Actions, manteniendo la misma URL del servicio:
<https://moviecards-service-edelacode-facuerczdsbwbygs.eastus-01.azurewebsites.net/>

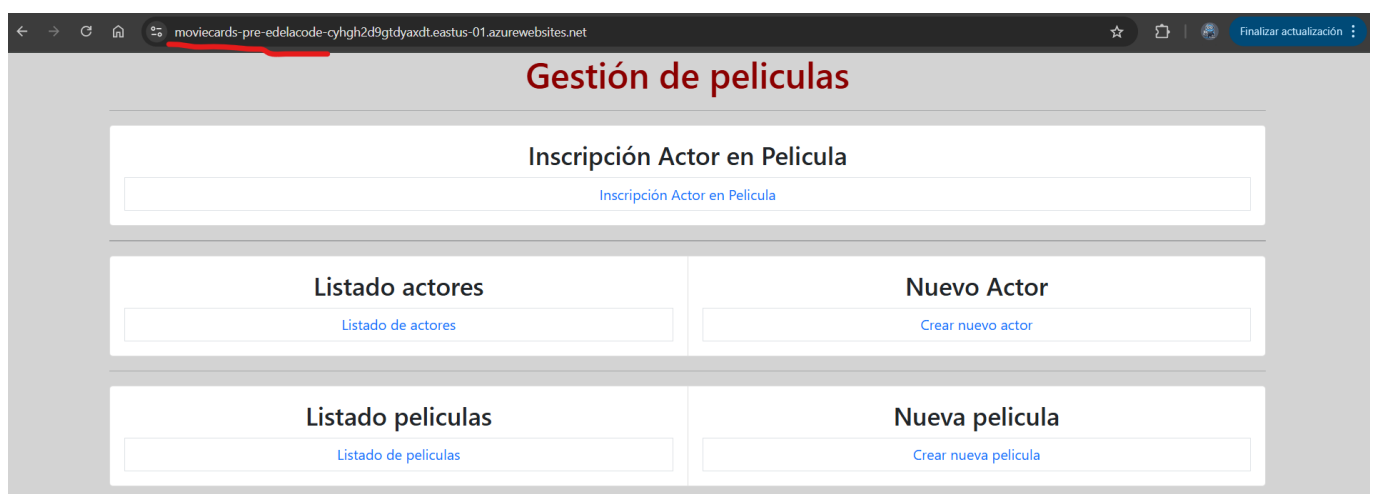


Despliegue a nuevo entorno de Pre-Producción

En esta parte del proyecto final se requiere la creación de una nueva fase en el flujo de trabajo para integración continua, entre los trabajos QA y DEPLOY. Para ello, se siguieron los siguientes pasos:

- Creación de una nueva aplicación web en Azure para desplegar un entorno de pre-producción.
- Modificación del workflow, agregando un nuevo trabajo llamado "stage" entre los trabajos "qa" y "deploy", para desplegar la aplicación en el entorno de pre-producción.
- Despliegue exitoso en el entorno de pre-producción.

```
96 stage:
97   runs-on: ubuntu-latest
98   needs: test
99   if: github.ref=='refs/heads/main'
100  environment:
101    name: 'Pre-Production'
102    url: '${{ steps.deploy-to-webapp.outputs.webapp-url }}'
103
104  steps:
105    - name: Download artifact from build job
106      uses: actions/download-artifact@v4
107      with:
108        name: moviecards-java
109
110    - name: Deploy to Azure Web App
111      id: deploy-to-webapp
112      uses: azure/webapps-deploy@v3
113      with:
114        app-name: 'moviecards-pre-edelacode'
115        slot-name: 'Production'
116        package: '*.jar'
117        publish-profile: '${{ secrets.AZUREAPPSERVICE_PUBLISHPROFILE_F8CEE65FA63045FA8570BB90802B029A }}
```



Nueva Versión de la Aplicación para Manejar la Fecha de Fallecimiento

Se realizó una nueva versión de la aplicación MovieCards para incluir la funcionalidad de manejar la fecha de fallecimiento de los actores.

Fue creado un nuevo milestone en GitHub llamado "Añadir Fecha de Fallecimiento", junto con los siguientes issues:

- Agregar campo `deadDate` en el formulario de Actor: Se modificó el formulario para incluir la fecha de fallecimiento.
- Agregar `deadDate` en la tabla con el listado de Actores: Se añadió una nueva columna en la tabla para mostrar la fecha de fallecimiento.
- Agregar y considerar `deadDate` en las pruebas unitarias: Se actualizaron las pruebas unitarias.
- Agregar y considerar `deadDate` en las pruebas de integración: Se actualizaron las pruebas de integración.
- Agregar y considerar `deadDate` en las pruebas funcionales (end-to-end): Se actualizaron las pruebas funcionales.

The screenshot shows a GitHub Milestone page for "Añadir Fecha de Fallecimiento". The milestone is 100% complete and is past due by 1 day. It contains a list of 7 closed issues, all assigned to user "uosip312".

Issue Title	Issue Number	Status	Assigned To
Calidad del Código	#12	Closed	uosip312
Añadir atributo <code>deaddate</code>	#13	Merged	uosip312
Pruebas End to End (Funcionales)	#11	Closed	uosip312
Pruebas de Integración	#10	Closed	uosip312
Pruebas Unitarias	#9	Closed	uosip312
Nuevo atributo <code>deadDate</code> en Plantillas	#8	Closed	uosip312
Nuevo atributo <code>deadDate</code> para Actor	#7	Closed	uosip312

```
<div class="mb-3">
  <label for="deadDate" class="form-label">Fecha Fallecimiento</label>
  <input
    type="date"
    class="form-control"
    th:field="*{deadDate}"
    id="deadDate"
    name="deadDate"
    placeholder="Escriba la fecha de fallecimiento"
  />
</div>
```

```
11      <div class="card">
12          <h2 th:text="'Listado Actores'" class="card-header"></h2>
13          <div class="card-body">
14              <table class="table table-hover">
15                  <thead class="thead-light">
16                      <tr>
17                          <th scope="col">Identificador</th>
18                          <th scope="col">Nombre</th>
19                          <th scope="col">Fecha Nacimiento</th>
20                          <th scope="col">Pais</th>
21                          <th scope="col">Fecha Fallecimiento</th>
22                          <th scope="col">Editar</th>
23                      </tr>
24                  </thead>
25                  <tbody>
26                      <tr th:each="actor : ${actors}">
27                          <td th:text="{actor.id}"></td>
28                          <td th:text="{actor.name}"></td>
29                          <td th:text="{#dates.format(actor.birthDate, 'dd-MM-yyyy')}"></td>
30                          <td th:text="{actor.country}"></td>
31                          <td th:text="{#dates.format(actor.deadDate, 'dd-MM-yyyy')}"></td>
32                          <td>
33                              <a th:href="@{'/editActor/' + {actor.id}}" class="btn btn-primary">Editar</a>
34                          </td>
35                      </tr>
```

```
@Test Edison de la Cruz
void testSetGetDeadDate() {
    Date deadDateExample = new Date();
    actor.setDeadDate(deadDateExample);
    assertEquals(deadDateExample, actor.getDeadDate());
}
```

```
@Test Edison de la Cruz
public void testSaveActor() {
    Actor actor = new Actor();
    actor.setName("actor");
    actor.setBirthDate(new Date());
    actor.setCountry("spain");
    actor.setDeadDate(new Date());

    Actor savedActor = actorJPA.save(actor);

    assertNotNull(savedActor.getId());

    Optional<Actor> foundActor = actorJPA.findById(savedActor.getId());

    assertTrue(foundActor.isPresent());
    assertEquals(savedActor, foundActor.get());
    assertNotNull(foundActor.get().getDeadDate());
}
```

```
@Test
void shouldGetActorById() {
    Actor actor = new Actor();
    actor.setId(1);
    actor.setName("Sample Actor");
    actor.setDeadDate(new Date());

    when(restTemplate.getForObject(url: URL + "/" + 1, Actor.class)).thenReturn(actor);

    Actor result = sut.getActorById(actorId: 1);

    assertEquals(expected: 1, result.getId());
    assertEquals(expected: "Sample Actor", result.getName());
    assertNotNull(result.getDeadDate());
}
```

```
@Test
public void testPageLoad() {
    driver.get("http://localhost:8089/actors/new");
    assertEquals(expected: "FichasPelículasApp | Aplicación de gestión de fichas de películas", driver.getTitle());

    assertTrue(driver.findElement(By.id("name")).isDisplayed());
    assertTrue(driver.findElement(By.id("birthDate")).isDisplayed());
    assertTrue(driver.findElement(By.id("country")).isDisplayed());
    assertTrue(driver.findElement(By.id("deadDate")).isDisplayed());
}
```

moviecards-edelacode-ahhmgysf6d2hshy.eastus-01.azurewebsites.net/actors/new

Nuevo Actor

Nombre

Escriba el nombre del actor

Fecha nacimiento

dd/mm/aaaa

País

Escriba el país del actor

Fecha Fallecimiento

dd/mm/aaaa

Guardar

Volver a la Página de Inicio

moviecards-edelacode-ahhmgysf6d2hshy.eastus-01.azurewebsites.net/actors

Listado Actores					
Identificador	Nombre	Fecha Nacimiento	País	Fecha Fallecimiento	Editar
1	Mario Simarro	11-08-1970	Colombia	19-07-2020	Editar
2	Test Actor	18-09-0101	USA	19-02-0101	Editar

Garantía de Calidad con SonarQube

Para asegurar la calidad del código, se realizaron las siguientes acciones:

- Modificación del Quality Gates de SonarQube para que no permita más de 5 errores críticos.
- Instalación del paquete JQ en el contenedor de Ubuntu con SonarQube, para recuperar el total de errores críticos en una consulta.
- Corrección del workflow (main.yaml) para que no despliegue a producción si hay más de 5 errores críticos.
- Corrección de errores críticos en la aplicación MovieCards para cumplir con los requisitos de calidad y permitir el despliegue a producción.

The screenshot shows the SonarQube interface for configuring Quality Gates. The 'Control de Calidad' gate is selected, showing a list of conditions:

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A (Technical debt ratio is less than 5.0%)
Reliability Rating	is worse than	A (No bugs)
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A (No vulnerabilities)

Below this, 'Conditions on Overall Code' are listed:


Metric	Operator	Value
Critical Issues	is greater than	5

A message at the bottom states: "You may click unlock to edit this quality gate. Adding extra conditions to a compliant quality gate can result in drawbacks. Are you reconsidering [Clean as You Code](#)? We strongly recommend this methodology to achieve a Clean Code status."

The screenshot shows the SonarQube 'Issues' page. The left sidebar shows filters for 'Type' (Bug, Vulnerability, Code Smell) and 'Severity' (Blocker, Critical, Major, Minor, Info). The main area displays two issues:


- Issue 1:** Replace this persistent entity with a simple POJO or DTO object. Severity: Critical. Location: moviecards / src/.../moviecards/controller/ActorController.java. Effort: 10min.
- Issue 2:** Replace this persistent entity with a simple POJO or DTO object. Severity: Critical. Location: moviecards / src/.../moviecards/controller/MovieController.java. Effort: 10min.

The page indicates 1 / 2 issues and 20min effort.


 **moviecards**

Passed


Last analysis: 1 day ago

 Bugs


0 A

 Vulnerabilities

2 D

 Hotspots Reviewed

– A

 Code Smells

45 A

Coverage

0.0%

Duplications

0.0%

Lines

700 XS Java, XML

← CI

Mejorar yaml para que lea el total critico en Job Deploy #35

Re-run jobs

Summary

Jobs

Run details

Jobs

build

test

qa

stage

deploy

Run details

Usage

Workflow file

Triggered via push yesterday

uosip312 pushed · fdeb861 main

Status

Failure

Total duration

4m 13s

Artifacts

1

main.yaml

on: push

build

23s

test

2m 28s

qa

41s

deploy

5s

stage

26s

Mejorar yaml para que lea el total critico en Job Deploy #35

Re-run jobs

Summary

Jobs

Run details

Jobs

build

test

qa

stage

deploy

Run details

Usage

Workflow file

Annotations

1 error

deploy

failed yesterday in 5s

Search logs

Set up job

4s

Validar fallos criticos antes del despliegue

0s

1 ▶ Run if ! [["\$CRITICAL_ISSUES" == "[0-9]*"]]; then

15 El número de fallos criticos (8) supera el límite permitido. Despliegue cancelado.

16 Error: Process completed with exit code 1.

Download artifact from build job

0s

← CI

Corregir errores critical #36

Re-run all jobs

Summary

Jobs

Run details

Jobs

build

test

qa

stage

deploy

Run details

Usage

Workflow file

Triggered via push yesterday

uosip312 pushed · 75e300c main

Status

Success

Total duration

4m 25s

Artifacts

1

main.yaml

on: push

build

28s

test

2m 24s

qa

34s

deploy

24s

stage

18s

Enlaces a Repositorios y Video

Para facilitar la revisión y replicación del trabajo, se han compartido los siguientes recursos:

- **Repositorio de MovieCards (Producción):** <https://moviecards-edelacode-ahhmgyesf6d2hshy.eastus-01.azurewebsites.net/>
- **Repositorio de MovieCards Service:** <https://moviecards-service-edelacode-facuerczdsbwbygs.eastus-01.azurewebsites.net/>
- **Repositorio de MovieCards (Pre-Producción):** <https://moviecards-pre-edelacode-cyhgh2d9gtdyaxdt.eastus-01.azurewebsites.net/>
- **Video demostrativo:** [Trabajo Final ICDA Edison DelaCruzRodriguez.mp4](#)

Conclusión

En conclusión, este trabajo demostró la importancia de la integración continua y el control de calidad en el desarrollo moderno de software. La automatización de los procesos de construcción, pruebas y despliegue permitió detectar y resolver problemas de manera rápida y eficiente, reduciendo el riesgo de errores críticos en producción.

El trabajo final permitió consolidar los conocimientos adquiridos durante el curso, aplicándolos en el desarrollo y mejora de la aplicación MovieCards. Se logró desplegar tanto la aplicación como su servicio en Azure, cumpliendo con todos los requisitos técnicos y funcionales. La adición del atributo `deadDate` y su integración en la aplicación demostró la importancia de mantener un código modular y escalable, facilitando la incorporación de nuevas funcionalidades.

Además, se reforzaron buenas prácticas de desarrollo, como el uso de pruebas unitarias, de integración y funcionales, así como la garantía de calidad mediante herramientas como SonarQube. En resumen, este trabajo no solo cumplió con los objetivos planteados, sino que también sirvió como una valiosa experiencia para enfrentar desafíos similares en el futuro.