Trabajo Final:
Integración Continua en el Desarrollo Ágil

Amle Martinez Marte

15/02/2025

Curso

Integración Continua en el Desarrollo Ágil

Profesores

José Ramón Hilera González

Daniel Rodríguez García

Tabla de contenido

INTRODUCCIÓN

Este documento describe el procedimiento seguido para desacoplar el servicio previamente integrado en la aplicación y ofrecerlo como un servicio independiente desplegado en Azure. Con esta modificación, cualquier aplicación, ya sea web, móvil o de escritorio, podrá consumir el servicio de manera independiente.

Además, se detallan las nuevas funcionalidades implementadas, como la incorporación de la fecha de fallecimiento de los actores, y los ajustes necesarios en las pruebas unitarias, de integración y funcionales para garantizar la correcta integración de estos cambios. También se explica el uso de SonarQube para analizar la calidad del código y aplicar las correcciones necesarias, asegurando que el número de errores críticos no supere el límite de cinco.

Finalmente, se documentan los desafíos encontrados durante el desarrollo y las soluciones aplicadas para resolverlos, proporcionando una visión clara del proceso y las decisiones tomadas a lo largo del proyecto.

I. Repositorios del Proyecto.

Es Estos son los enlaces de los repositorios del proyecto:

- **Repositorio de la aplicación MovieCards:** [https://github.com/misterlink19/moviecards]
- **Repositorio del servicio MovieCards-Service:** [https://github.com/misterlink19/moviecards-service]

También, las URLs donde están desplegadas la aplicación y el servicio:

- **URL del stage de la aplicación:** [https://moviecards-pre-stage.azurewebsites.net]
- **URL de la aplicación en producción:** [https://moviecards-martinez.azurewebsites.net]
- **URL del servicio desplegado:** [https://moviecards-service-martinez.azurewebsites.net]

II. Desarrollo del Proyecto.

1. Creación y Configuración de la Aplicación MovieCards

Para desplegar la aplicación en Azure, seguí los pasos descritos en la Práctica 5. En el caso del servicio, el proceso fue similar, pero sin la parte de QA, ya que no se utilizó SonarQube en esta parte. Para diferenciar el servicio de la aplicación principal, le agregué "Service" antes del apellido en el nombre del despliegue.

En el entorno de desarrollo utilicé VS Code, donde configuré él .gitignore para excluir archivos innecesarios como .vscode y target. Al inicio, cometí un error en la configuración de este archivo (como se nota en los commits), pero después lo corregí. En el caso del servicio, no hubo este problema.

Modifiqué el archivo main.yml de ambas aplicaciones. En la aplicación desplegada, agregué los tests que corren en una PC hospedada en GitHub. Luego, añadí el trabajo de QA, que corre localmente en un contenedor en mi máquina y utiliza SonarQube para analizar el código. Finalmente, agregué el trabajo de deploy, donde se despliega la aplicación en Azure.

2. Implementación del Servicio MovieCards-Service

La aplicación base la obtuve del repositorio que nos proporcionó el profesor, por lo que no hubo muchos cambios. Básicamente, la subí a un nuevo repositorio propio y configuré su despliegue en Azure.

Después de desplegar el servicio, lo probé con Postman para asegurarme de que estaba funcionando correctamente. Luego, en la aplicación principal, modifiqué varias secciones del código para que apuntaran al servicio desplegado en vez de usar datos internos.

3. Creación y Configuración de la Aplicación MovieCards

Para que la aplicación MovieCards pudiera comunicarse correctamente con el servicio, modifiqué varios archivos, incluyendo ActorServiceImpl, CardServiceImpl, MovieServiceImpl y MovieCardsApplication.

La integración se hizo usando RestTemplate, agregando el siguiente código en MovieCardsApplication:

```
@Bean
public RestTemplate template() {
RestTemplate template = new RestTemplate();
return template;
}
```

Además, en ActorServiceImpl, se cambiaron las llamadas directas a la base de datos por peticiones al servicio:

```
@Autowired
RestTemplate template;
String url = "https://moviecards-service-
martinez.azurewebsites.net/actors";

@Override
public List<Actor> getAllActors() {
Actor[] actores = template.getForObject(url,
Actor[].class);
List<Actor> actoresList = Arrays.asList(actores);
return actoresList;
}

@Override
public Actor save(Actor actor) {
if (actor.getId() != null && actor.getId() > 0) {
template.put(url, actor);
} else {
actor.setId(0);
template.postForObject(url, actor, String.class);
}
return actor;
}


@Override
public Actor getActorById(Integer actorId) {
Actor actor = template.getForObject(url + "/" + actorId,
Actor.class);
return actor;
}
```

Y Cambios similares se hicieron en MovieServiceImpl y CardServiceImpl. También modifiqué los tests en ActorServiceImplTest y MovieServiceImplTest para que funcionaran con el servicio. Estos cambios se pueden visualizar en la rama master en el commit “1fd08485b73fe1a3d03e651e2b2eda0779d08f98”.

4. Incorporación del Atributo deadDate en la Aplicación.

Agregar el atributo deadDate fue sencillo porque seguí la misma lógica de birthDate.

En la interfaz, agregué una nueva columna y un input adicional para ingresar la fecha de fallecimiento. Además, si el atributo deadDate es null, en la lista de actores aparece el mensaje "Sigue vivo", indicando que el actor no tiene una fecha de fallecimiento registrada.

5. Modificaciones en las Pruebas

Para reflejar estos cambios, actualicé varias pruebas:

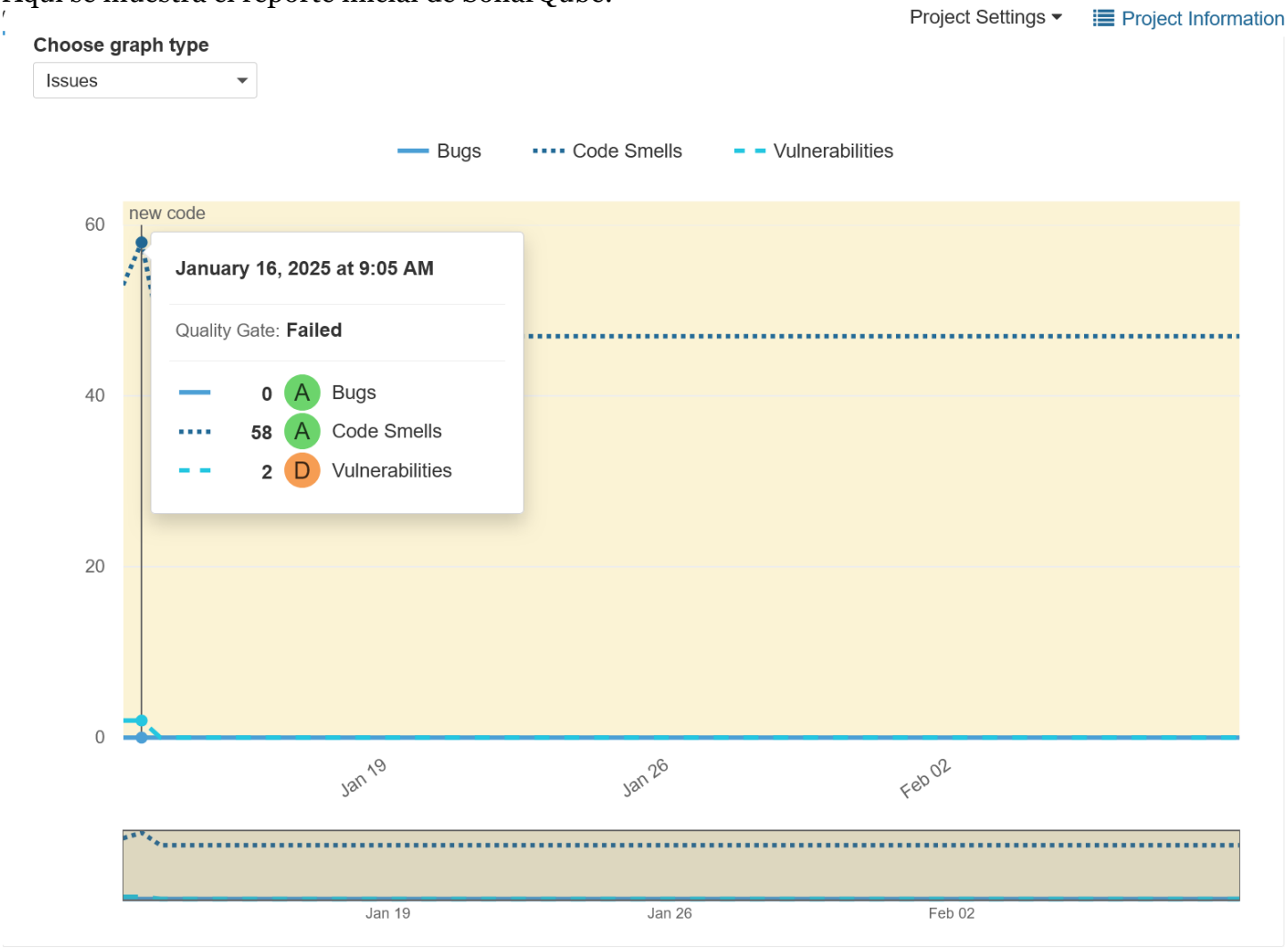
- **Pruebas unitarias:** ActorServiceImplTest, ActorTest y ActorControllerTest, para incluir deadDate.
- **Pruebas de integración:** Solo fue necesario modificar ActorJPAIT.
- **Pruebas funcionales:** Modifiqué ActorE2ETest, aunque al principio tuve errores porque no ajusté bien la numeración de las columnas.

Estos cambios se pueden ver en el commit: “80a996b1bd419c2d1548f11baf9f2122e2fe220e”.

6. Medidas de Calidad y SonarQube

Para la parte de SonarQube, me enfoqué en corregir errores críticos, vulnerabilidades y código repetido (Code smells).

Aquí se muestra el reporte inicial de SonarQube:



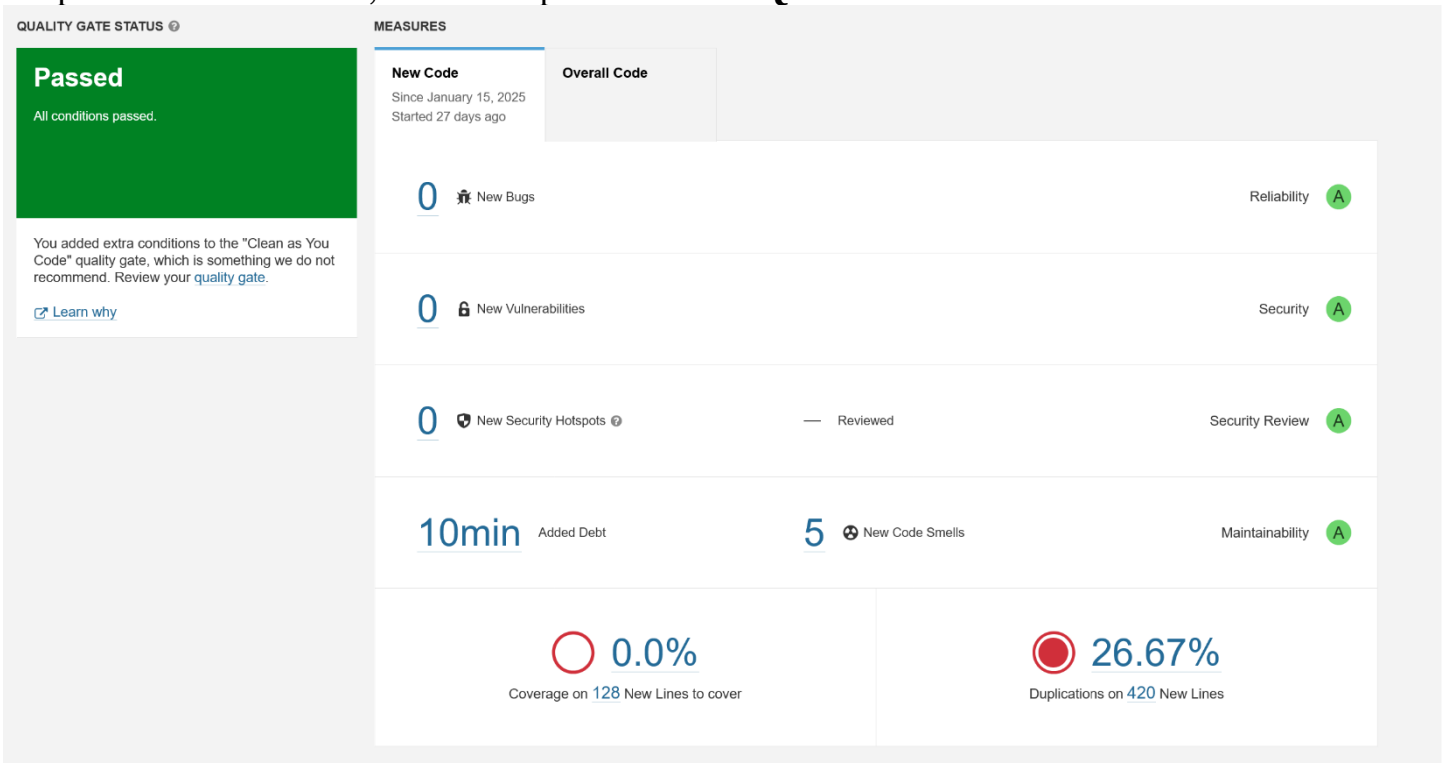
Muchos de estos problemas se debían a que no había DTOs para Actor y Movie, y a código repetido en los controladores. Para solucionar esto:

- Creé **DTOs** para evitar manipular entidades directamente.
- Eliminé código repetido con **constantes estáticas**.

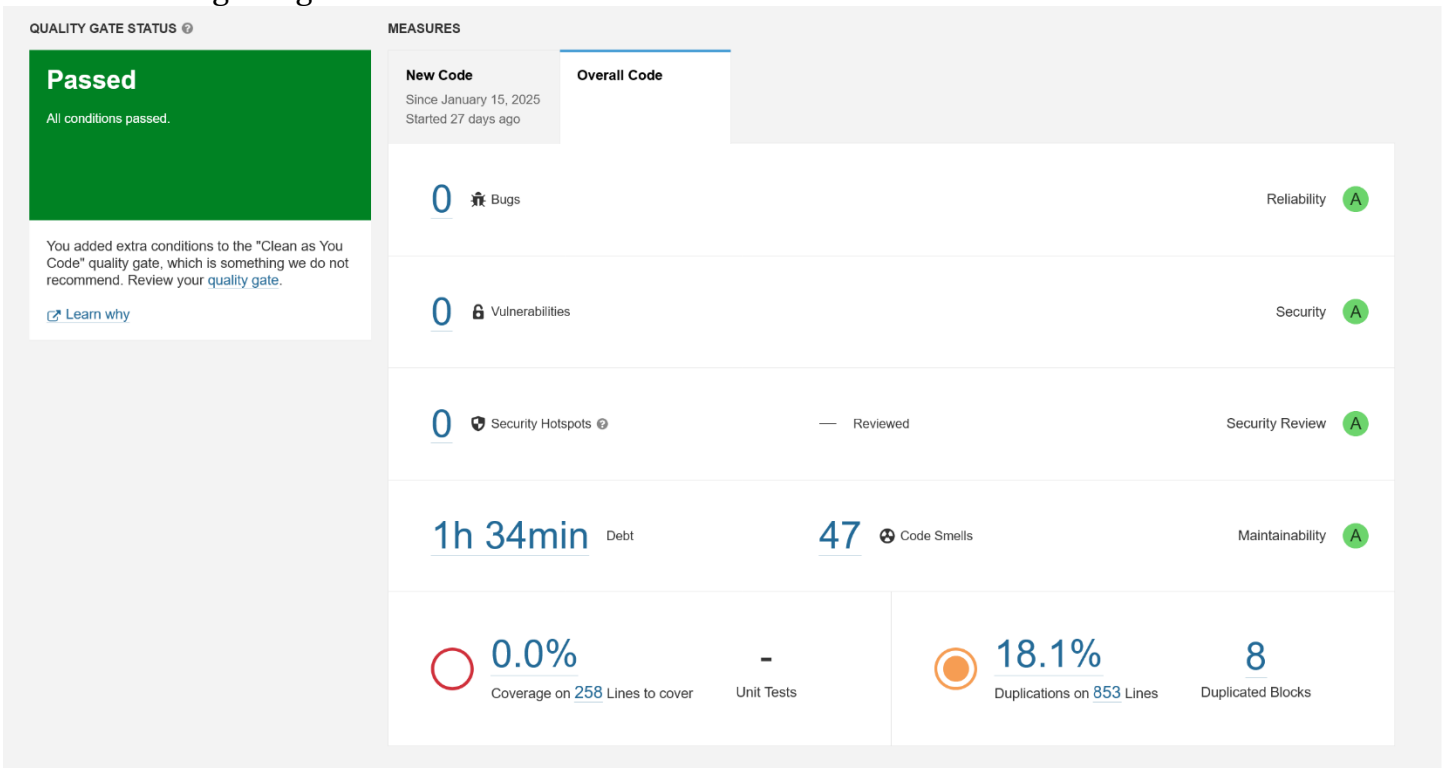
- Agregué un **constructor privado** en la clase Messages para evitar instanciación innecesaria (esto era un Code smell detectado por SonarQube).

Esto se puede visualizar en el commit “e2e4e7ceecf4122cb742db43eb9f0b05b88bef27”

Después de estos cambios, el nuevo reporte de **SonarQube** se veía así:



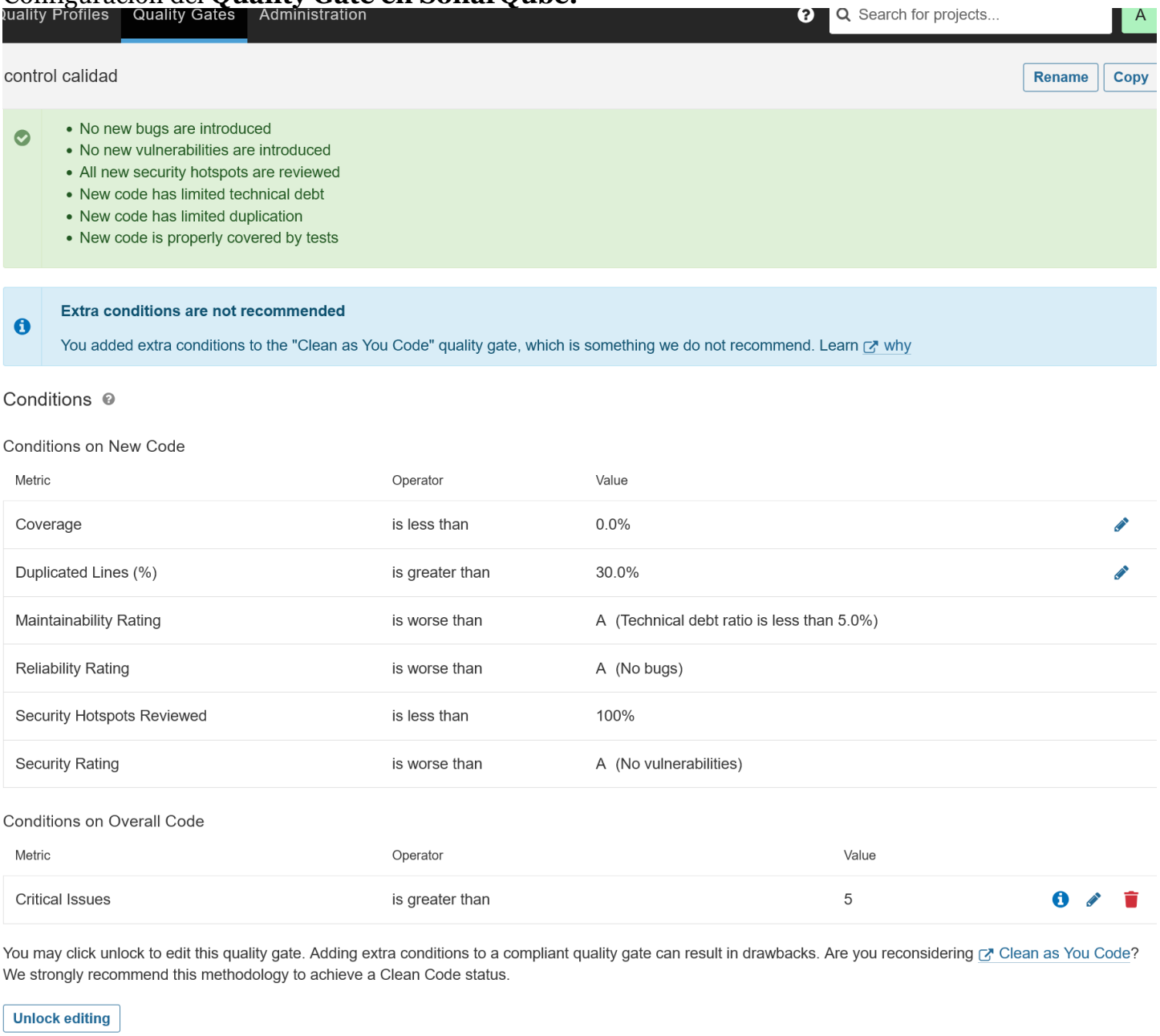
Y esto del Código en general:



Y esto se puede visualizar en el commit “1d3d0067dd8026cad1c3b269f6719a977e865c4d”, es donde se puede visualizar que todos las vulnerabilidades y errores críticos fueron corregidos.

En este punto, **todas las vulnerabilidades y errores críticos fueron solucionados**. También ajusté las reglas de SonarQube para evitar alertas innecesarias sobre cobertura de código nuevo y repeticiones menores.

Configuración del **Quality Gate en SonarQube:**



7. Implementación del Entorno de Preproducción (Stage)

Para el entorno de **preproducción (Stage)**, básicamente seguí los mismos pasos del despliegue a producción, pero configuré un nuevo trabajo en GitHub Actions llamado stage.

La diferencia es que cuando los cambios se suben a una **rama distinta a la principal**, se despliegan en este entorno en lugar de producción.

La URL del stage es: **https://moviecards-pre-stage.azurewebsites.net**

Esto se puede ver en el commit: "**79aa6837249feeb06df53bb739a551dea1d67c69**"

III. Desafíos y Soluciones

Durante el desarrollo, me encontré con algunos problemas interesantes:

1. **Implementación de Projects en GitHub**
   Al principio, mis **historias de usuario no se enlazaban bien** con las ramas e issues del proyecto moviecards. Por eso, en el **board** aparecen más historias completas de las que realmente hay. Sin embargo, después de algunas pruebas, logré que el sistema funcionara bien.

2. **Implementación del Trabajo Stage:**
   El trabajo de **Stage** no se ejecutaba y me tomó un par de días descubrir por qué. Resulta que el problema era simplemente el app-name del application Service de Azure se llamaba "moviecards-pre-producción" que fue el nombre que le puse yo, pues resulta que porque ese nombre llevaba una tilde en la parte de producción por eso no se me ejecutaba el trabajo, algo tan sencillo me dio tantos problemas. Al cambiarlo por "moviecards-pre-stage", todo funcionó correctamente. Debido a eso se puede ver que hay muchos commits que dan error por esta particularidad y otros eran por lo pruebas funcionales. Todo esto corregido se puede ver en el commit "79aa6837249feeb06df53bb739a551dea1d67c69

3. **Integración de QA y Deploy**
   Quise hacer que el **deploy a producción solo se ejecutara si QA pasaba correctamente**, pero esto no fue posible porque **QA se ejecuta en mi máquina local en un contenedor**, y GitHub Actions no tiene acceso a su resultado, además de que ningún cambio que se realice a la rama principal no se despliega a menos que se aprobado el pull request. Esto explica varios commits de prueba en el historial

IV. Conclusión.

El desarrollo de este proyecto ha sido una experiencia enriquecedora que ha permitido aplicar conceptos clave de la integración continua y el desarrollo ágil. A través del desacoplamiento del servicio y su despliegue independiente en Azure, se ha logrado una arquitectura más flexible y escalable, permitiendo que diferentes tipos de aplicaciones consuman el servicio de manera eficiente.

La incorporación de nuevas funcionalidades, como la fecha de fallecimiento de los actores, y la adaptación de las pruebas unitarias, de integración y funcionales, han sido esenciales para asegurar la correcta integración y funcionamiento de los cambios. El uso de SonarQube ha sido fundamental para mantener la calidad del código, identificando y corrigiendo errores críticos y vulnerabilidades.

A lo largo del proyecto, se han enfrentado diversos desafíos, desde problemas con la configuración de GitHub Projects hasta la implementación del entorno de preproducción. Cada uno de estos desafíos ha sido una oportunidad para aprender y mejorar, aplicando soluciones efectivas que han contribuido al éxito del proyecto.

En resumen, este proyecto no solo ha cumplido con los objetivos planteados, sino que también ha proporcionado valiosas lecciones y experiencias que serán útiles en futuros desarrollos. La documentación detallada y el video explicativo servirán como guía para entender el proceso y las decisiones tomadas, asegurando una presentación clara y profesional del trabajo realizado.