PRÁCTICA FINAL

ESCUELA POLITÉCNICA SUPERIOR

MÁSTER EN DESARROLLO ÁGIL PARA LA WEB

2025-2026

Asignatura: Integración Continua en el Desarrollo Ágil

Alumno: Juan Higuero López

Fecha: 22 de febrero de 2026

Índice:

1. Introducción 1

2. Desarrollo de la Práctica (Tema 5) 1

2.1. Implementación de Build y Test 1

2.2. Implementación de QA 2

2.3. Despliegue en Microsoft Azure 3

2.4. Aprobación Manual 4

2.5. Planificación Ágil con GitHub Projects 6

2.6. Gestión de Ramas y Resolución de Conflictos 7

3. Creación de moviecards-service 11

4. Refactorización de la Aplicación Principal 12

5. Implementación del Atributo deadDate 13

6. Entorno de Pre-producción y Ciclo de Calidad 13

1. Introducción

El presente proyecto se enmarca en la evolución de la plataforma Moviecards, una aplicación web desarrollada con Spring Boot para la gestión de fichas de actores y películas. El objetivo principal de esta fase del desarrollo ha sido la modernización de su arquitectura y la automatización total de su ciclo de vida mediante técnicas de Integración Continua (CI) y Despliegue Continuo (CD) utilizando GitHub Actions.

Tradicionalmente, la aplicación operaba bajo una estructura monolítica con una base de datos local. En esta actualización, se ha procedido a la extracción de la lógica de negocio hacia un servicio externo denominado moviecards-service, desplegado de forma independiente en Azure Web Apps. Este cambio ha permitido que la aplicación principal actúe como un cliente REST, consumiendo datos a través de una API externa, lo que mejora la escalabilidad y el desacoplamiento del sistema.

Además de la reestructuración arquitectónica, se ha implementado un flujo de trabajo automatizado que garantiza la calidad del software en cada cambio realizado. Este flujo incluye:

- **Gestión Ágil**: Organización del trabajo mediante Milestones e Issues en GitHub para el seguimiento de tareas.
- **Validación Técnica**: Ejecución de baterías de pruebas unitarias, de integración y funcionales (End-to-End) para validar nuevas funcionalidades como la fecha de fallecimiento de los actores.
- **Control de Calidad**: Integración con SonarQube para monitorizar la deuda técnica y asegurar que ningún código con problemas críticos llegue a producción.
- **Estrategia de Despliegue**: Uso de entornos de pre-producción (Stage) y despliegue final en producción supeditado a aprobaciones manuales.

2. Desarrollo de la Práctica (Tema 5)

En este bloque de la asignatura se procedió a la preparación de toda la infraestructura necesaria para el ciclo de vida del software. El trabajo comenzó con la creación de cuentas en GitHub y Azure, y la instalación de los recursos tecnológicos necesarios: Visual Studio, Maven, GitHub Desktop y Docker.

2.1. Implementación de Build y Test

Se desplegó un contenedor de SonarQube mediante Docker para el análisis estático de código. Tras crear el repositorio en GitHub, se inició la construcción del archivo main.yaml, siguiendo un proceso de validación iterativo en cada uno de sus componentes:

- **Apartado Build**: Se configuró la etapa de compilación. Para verificar la robustez del pipeline, se provocó un error forzado en la construcción,

confirmando que el flujo se detenía correctamente, y se procedió a su posterior corrección.

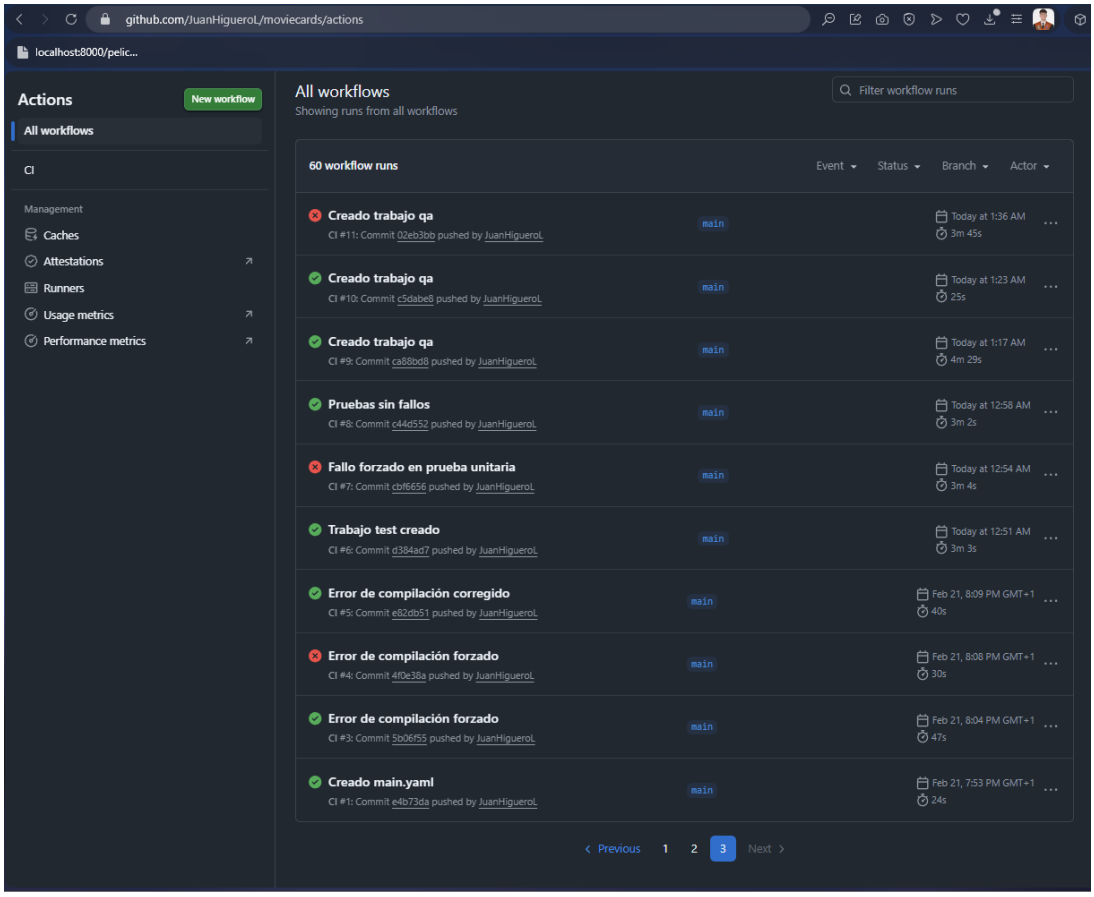
- **Apartado Test**: Se implementó la fase de pruebas automáticas siguiendo las directrices de la práctica. Al igual que en el paso anterior, se forzó un error para validar que el sistema de integración continua detectaba fallos en los tests antes de restablecer el código a su estado funcional.

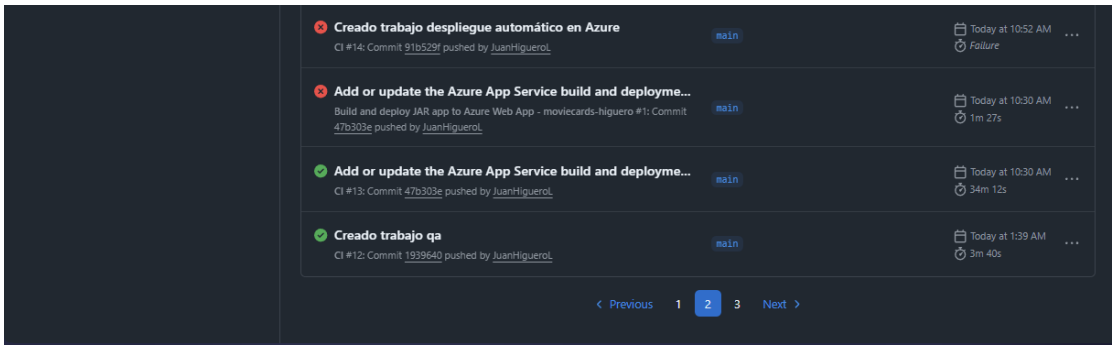
2.2. Implementación de QA

Para la ejecución del análisis de calidad (QA), se optó por una configuración basada en un Self-hosted Runner:

- **Configuración del Runner**: Se hizo uso de actions-runner para crear una instancia propia denominada "mi-runner", la cual se ejecutó y gestionó desde PowerShell.
- **Integración de SonarQube**: Se modificó el main.yaml integrando los parámetros de acceso y la contraseña configurada en SonarQube.
- **Ajuste de Umbrales**: Durante las pruebas de calidad, se aplicó inicialmente una restricción máxima de 5 errores críticos. Dado que el proyecto no superaba este filtro en su estado inicial, se aumentó el límite a 8 errores para permitir la continuidad del flujo y el avance hacia las siguientes fases.

En las siguientes imágenes se muestran los procesos mencionados en Github.



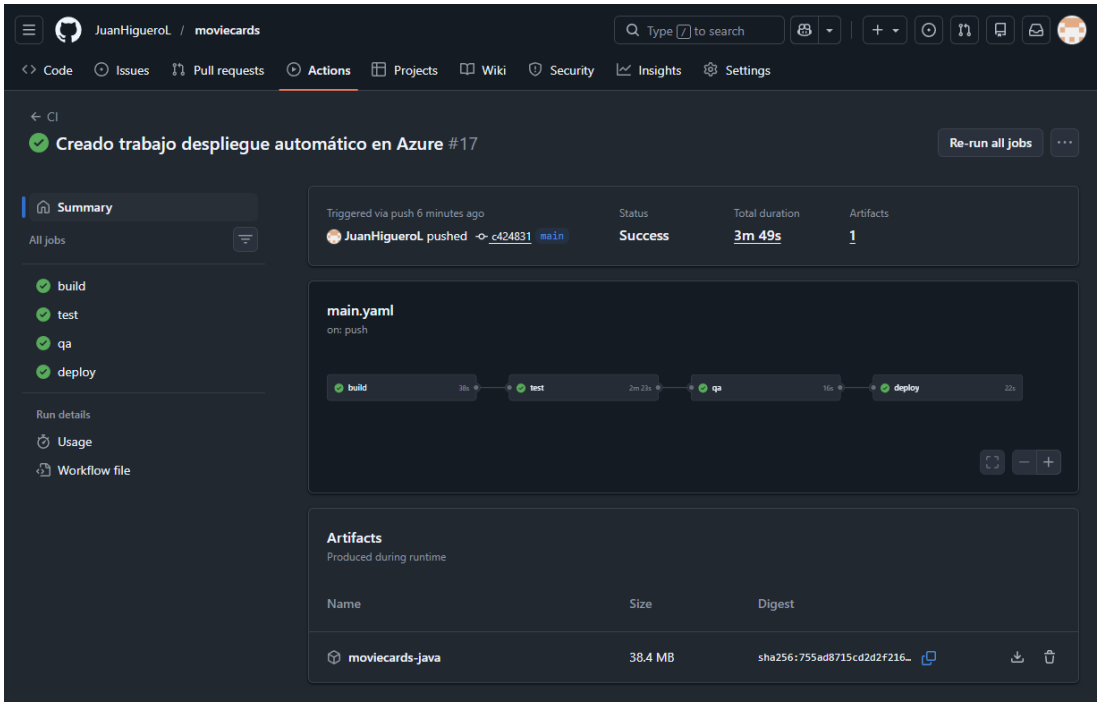


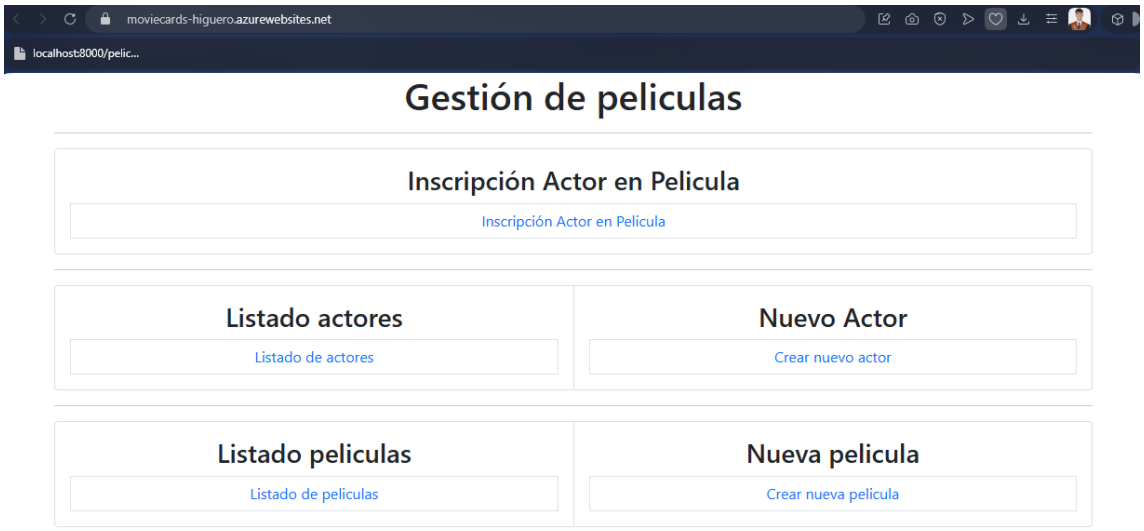
2.3. Despliegue en Microsoft Azure

Una vez validada la estructura del pipeline, se procedió a la puesta en marcha de la aplicación en la nube. Debido a limitaciones técnicas y errores de disponibilidad en otras regiones, se optó por crear la Web App en Azure bajo la región Italy North.

Para integrar el despliegue en el flujo de trabajo, se siguieron los pasos de configuración para conectar Azure con GitHub. Este proceso generó automáticamente un archivo YAML de despliegue específico de Azure; de este archivo se extrajo el secreto de publicación, que fue configurado en los secretos del repositorio de GitHub. Una vez transferida esta configuración al main.yaml principal para unificar el flujo, el archivo generado por Azure fue eliminado para mantener la limpieza del repositorio.

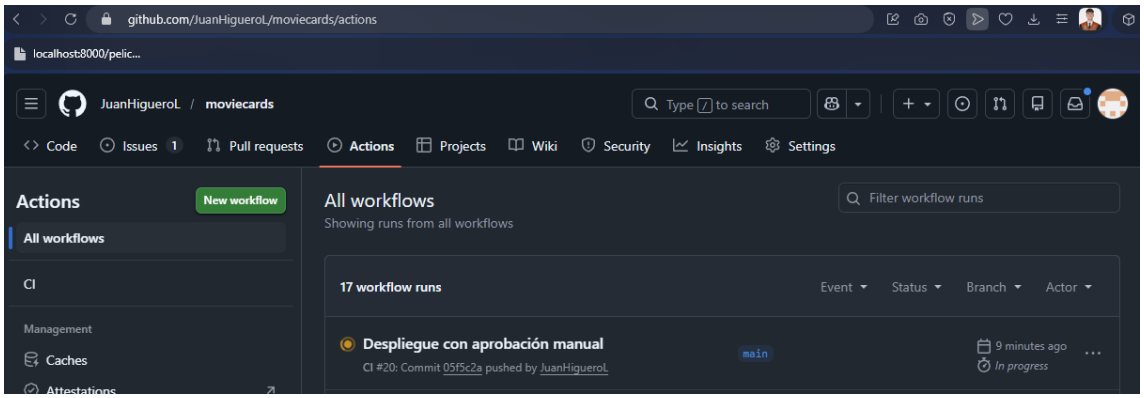
Tras realizar el commit y el push de estos cambios, la aplicación ya puede visualizarse en Azure.

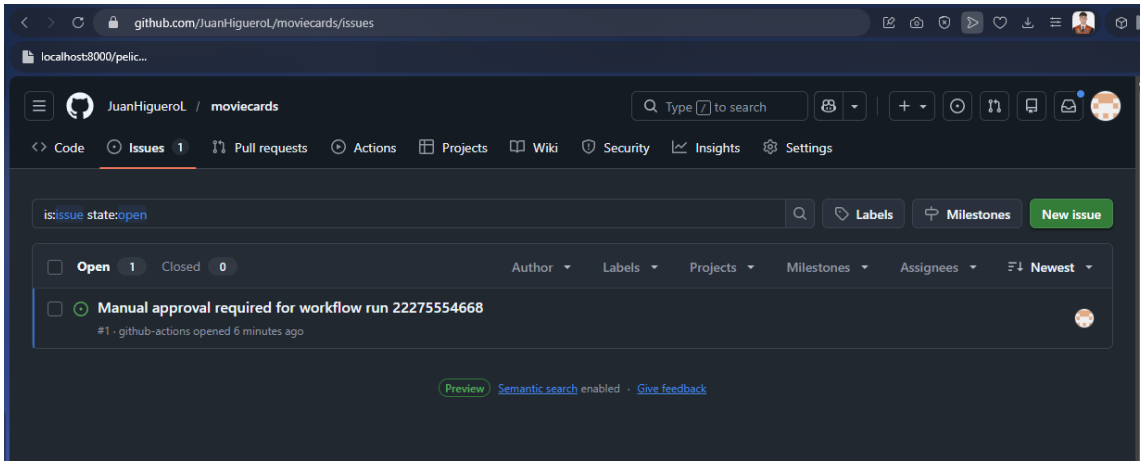


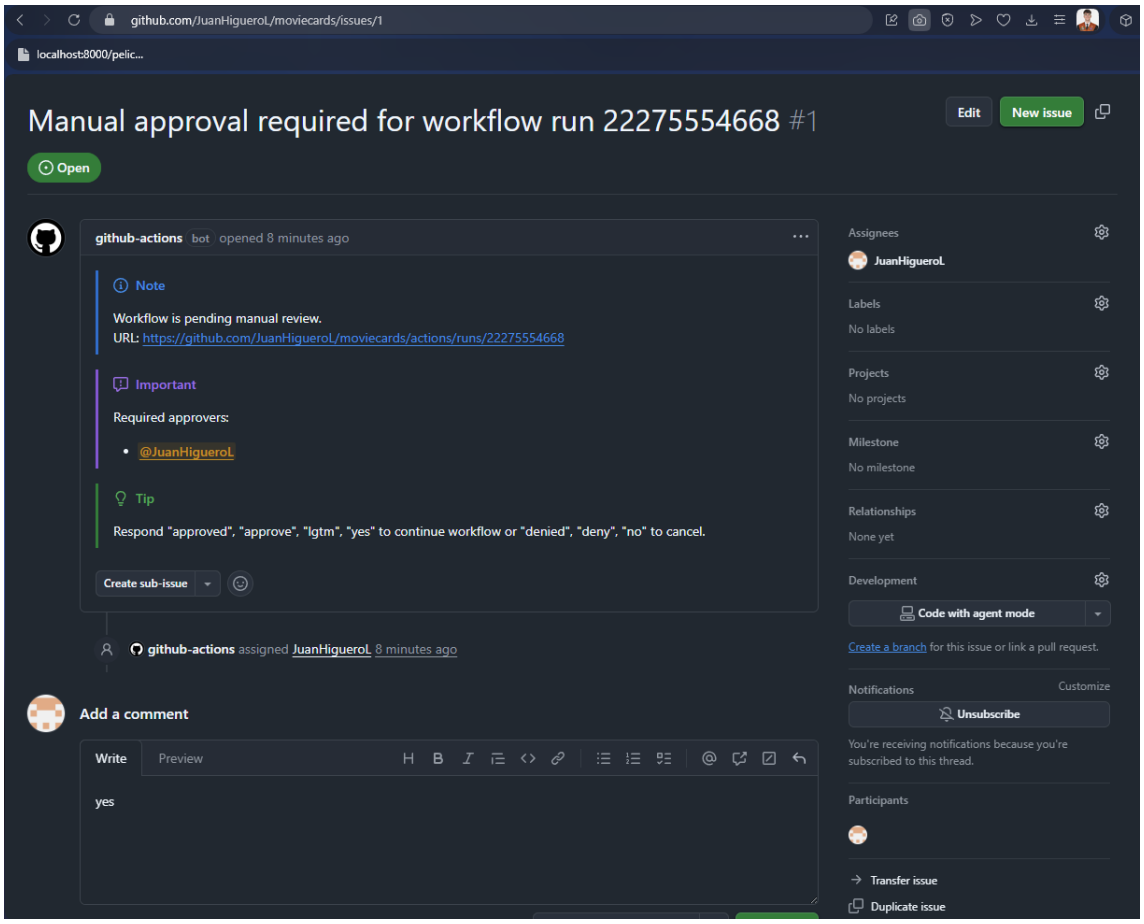


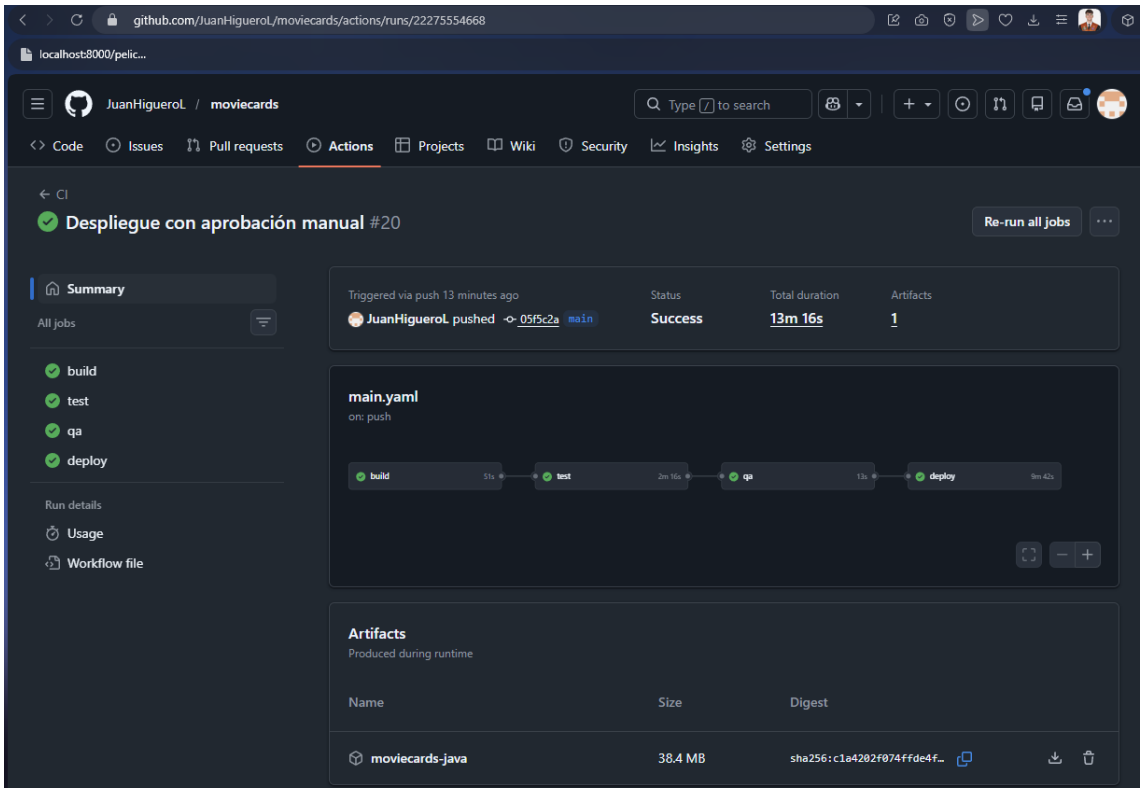
2.4. Aprobación Manual

Siguiendo las directrices de seguridad de la práctica, se implementó un paso de Aprobación Manual antes del despliegue definitivo en producción. Este mecanismo asegura que un responsable deba validar el estado del proyecto antes de que los cambios se apliquen en el entorno real. En el flujo se observa cómo el workflow entra en estado de espera hasta que el usuario autorizado introduce un "yes".





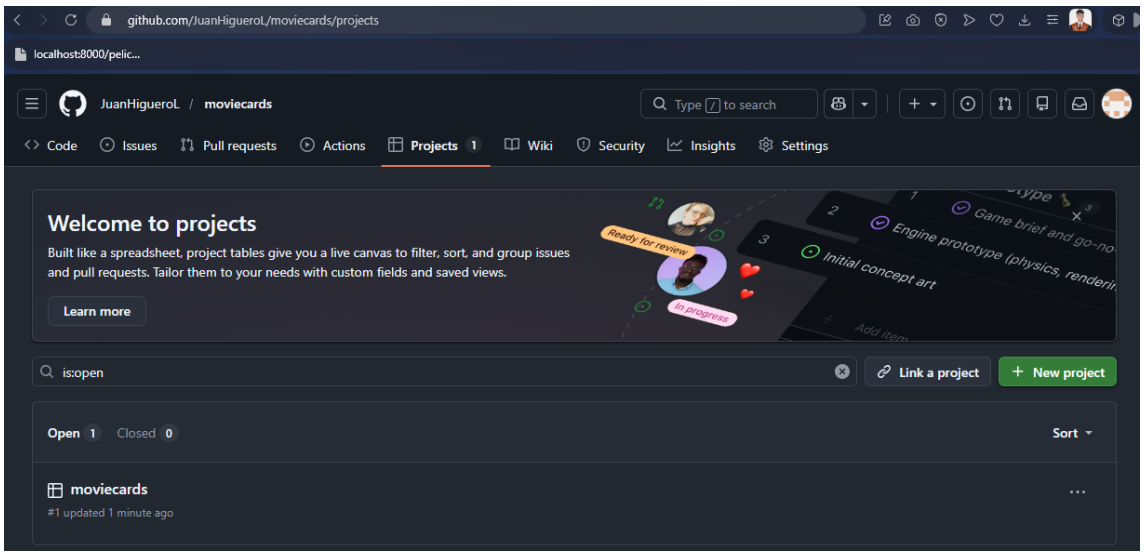

github.com/JuanHigueroL/moviecards/issues/1
Manual approval required for workflow run 22275554668 #1
Edit
New issue
Open
github-actions bot opened 8 minutes ago
Note
Workflow is pending manual review.
URL: https://github.com/JuanHigueroL/moviecards/actions/runs/22275554668
Important
Required approvers:
@JuanHigueroL
Tip
Respond "approved", "approve", "lgtm", "yes" to continue workflow or "denied", "deny", "no" to cancel.
Create sub-issue
github-actions assigned JuanHigueroL 8 minutes ago
Add a comment
Write
Preview
yes
Assignees
JuanHigueroL
Labels
No labels
Projects
No projects
Milestone
No milestone
Relationships
None yet
Development
Code with agent mode
Create a branch for this issue or link a pull request.
Notifications
Customize
Unsubscribe
You're receiving notifications because you're subscribed to this thread.
Participants
Transfer issue
Duplicate issue

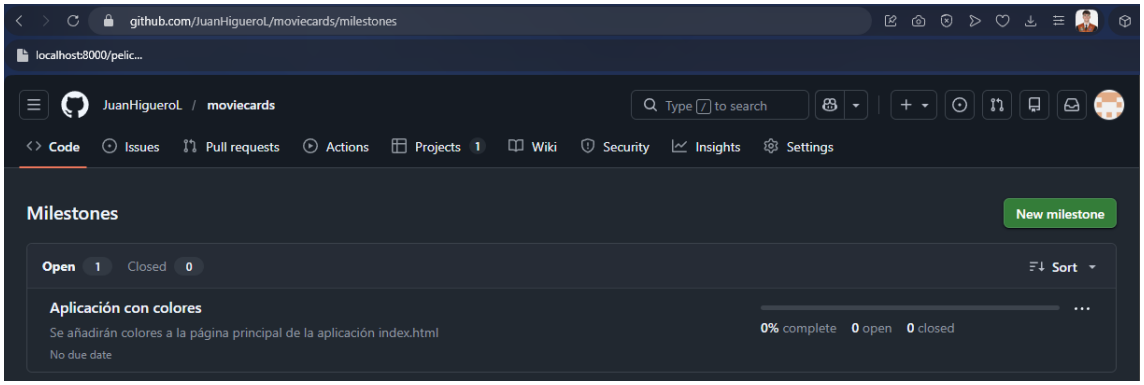

github.com/JuanHigueroL/moviecards/actions/runs/22275554668
JuanHigueroL / moviecards
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
CI
Despliegue con aprobación manual #20
Re-run all jobs
Summary
All jobs
build
test
qa
deploy
Run details
Usage
Workflow file
Triggered via push 13 minutes ago
JuanHigueroL pushed 05f5c2a main
Status
Success
Total duration
13m 16s
Artifacts
1
main.yaml
on: push
Artifacts
Produced during runtime
Name
Size
Digest
moviecards-java
38.4 MB

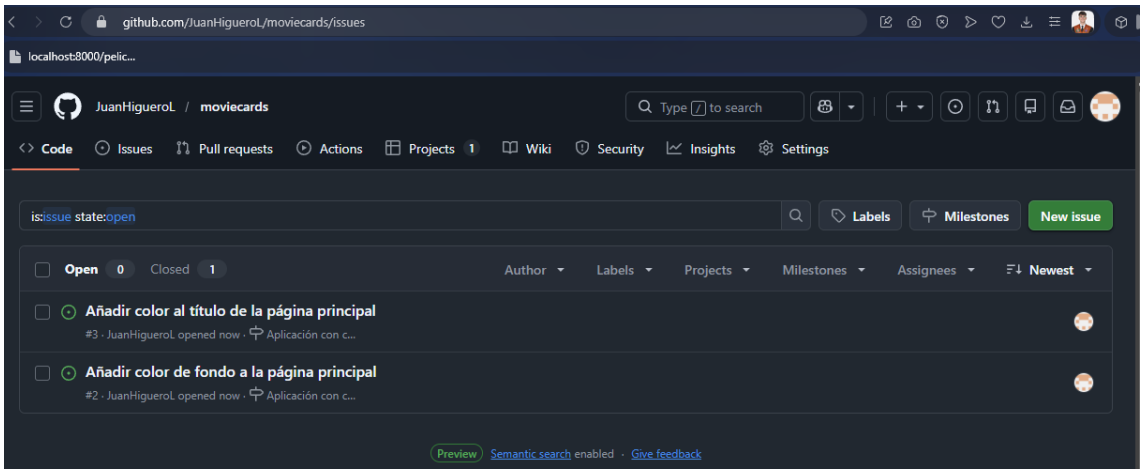
2.5. Planificación Ágil con GitHub Projects

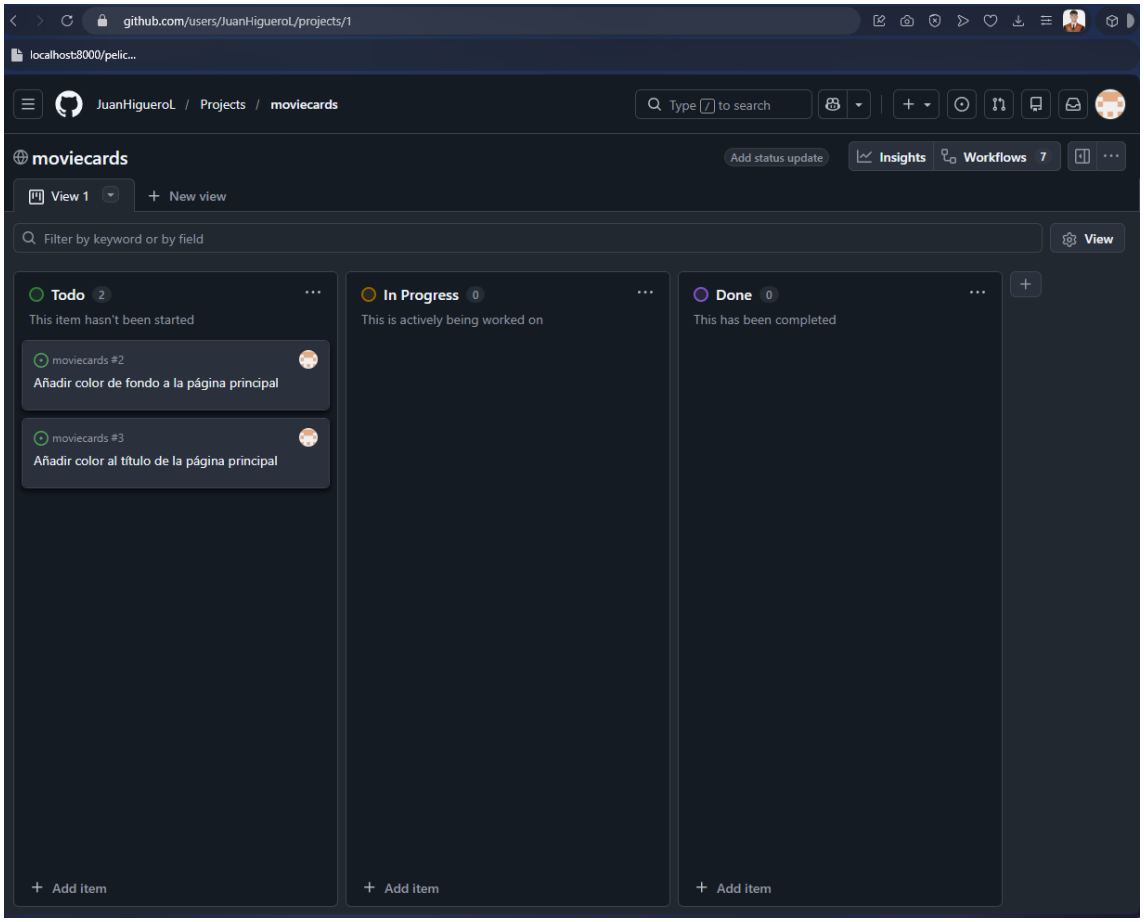
Para la gestión de tareas se hace uso de la metodología ágil con las herramientas nativas de GitHub:

- **Configuración del Proyecto**: Se creó un proyecto denominado "moviecards" para centralizar la gestión de tareas.
- **Milestones y User Stories**: La planificación se organizó mediante Milestones (Sprints), dentro de los cuales se definieron las Issues que representan las historias de usuario.
- **Tablero Kanban**: Se configuró un tablero visual para monitorizar el flujo de trabajo, permitiendo mover las tareas entre los estados Todo, In Progress y Done.







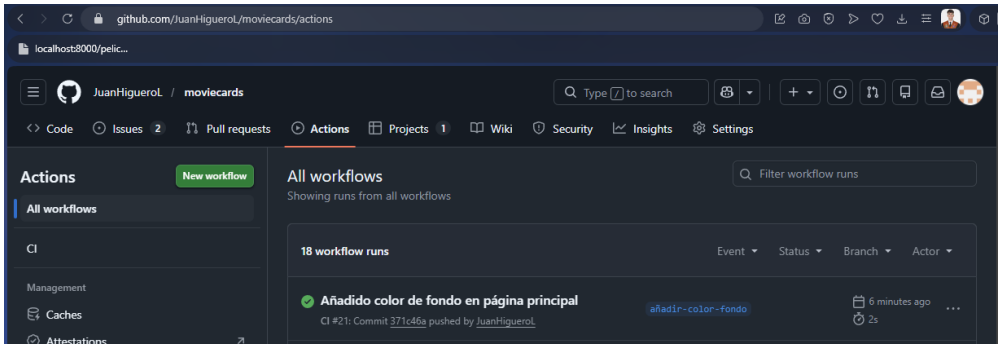


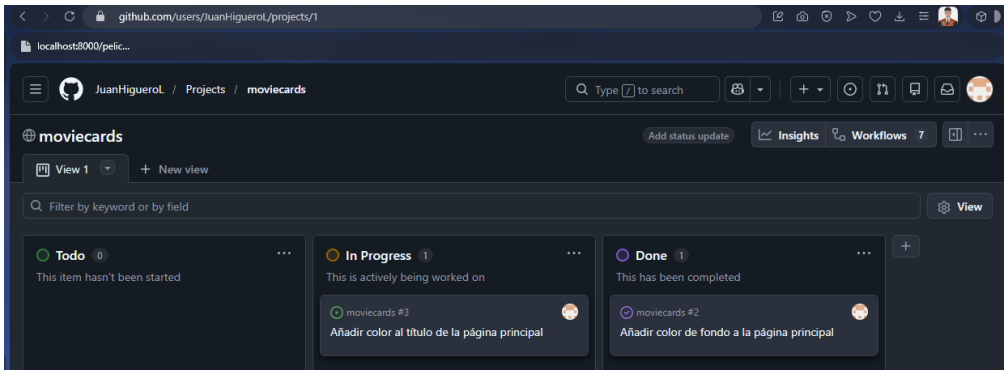
2.6. Gestión de Ramas y Resolución de Conflictos

Se trabajó con un modelo de ramas para implementar cambios estéticos en la interfaz:

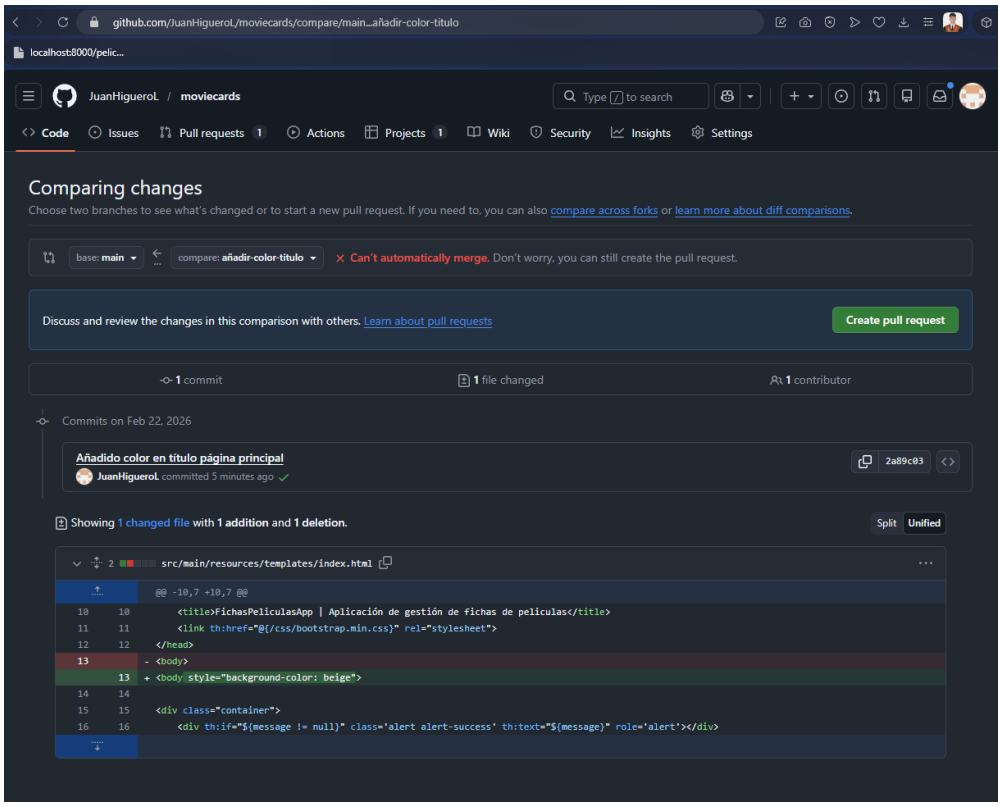
1. **Desarrollo en Ramas**: Se crearon dos ramas específicas, una para modificar el color del fondo y otra para el color del título.
2. **Validación y Merge**: Tras realizar los commits y verificar que las pruebas automatizadas en el pipeline no reportaban errores, se procedió a mover las tareas en el tablero Kanban y realizar el merge hacia la rama `main`.

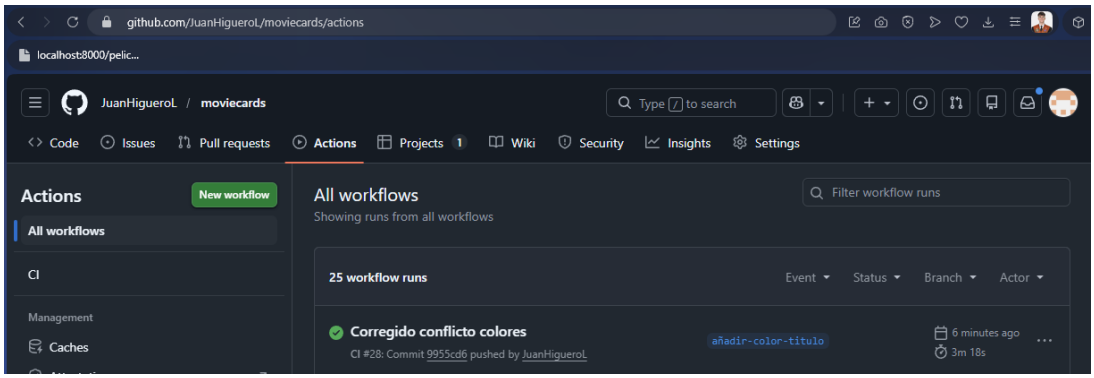
3. **Gestión de Conflictos**: Al trabajar en la segunda rama, surgieron errores de compatibilidad debido a que ambas ramas modificaban las mismas líneas de código. Se procedió a la resolución manual de los conflictos.



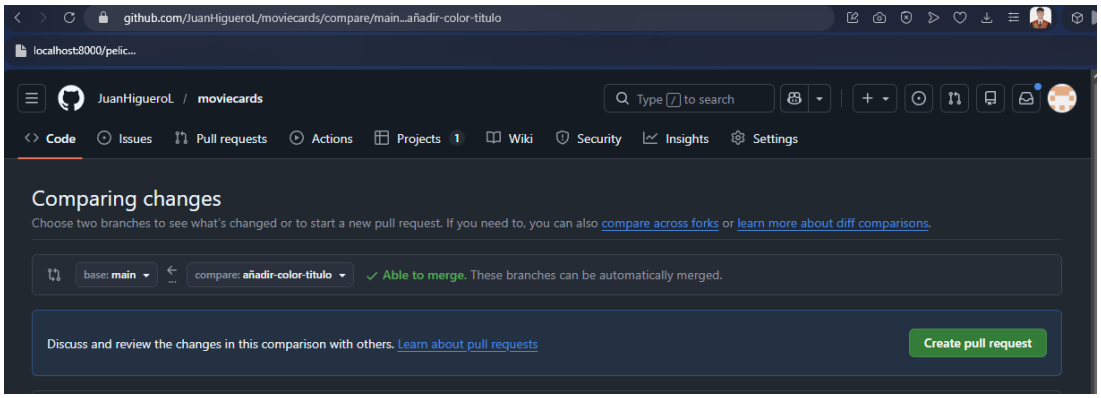


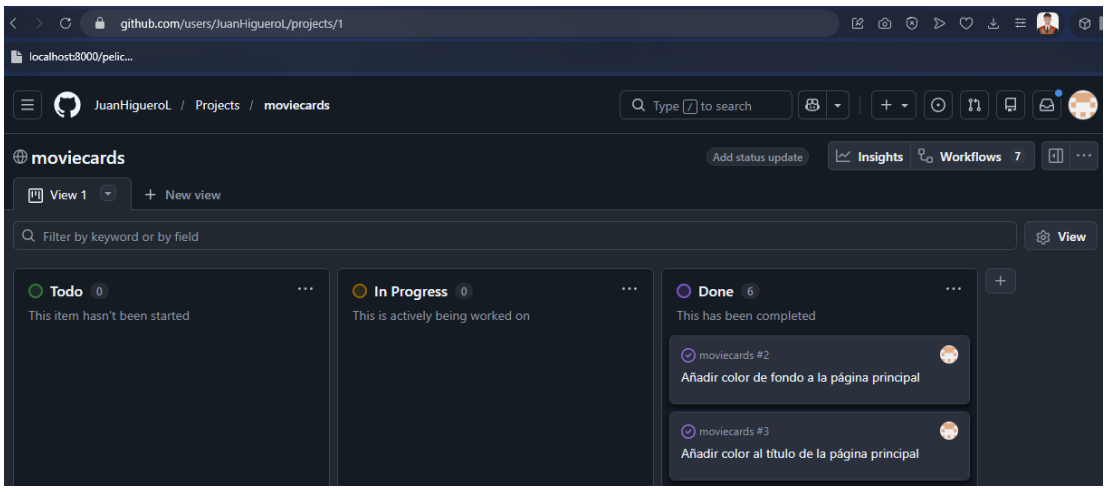
Aviso de conflicto de fusión.



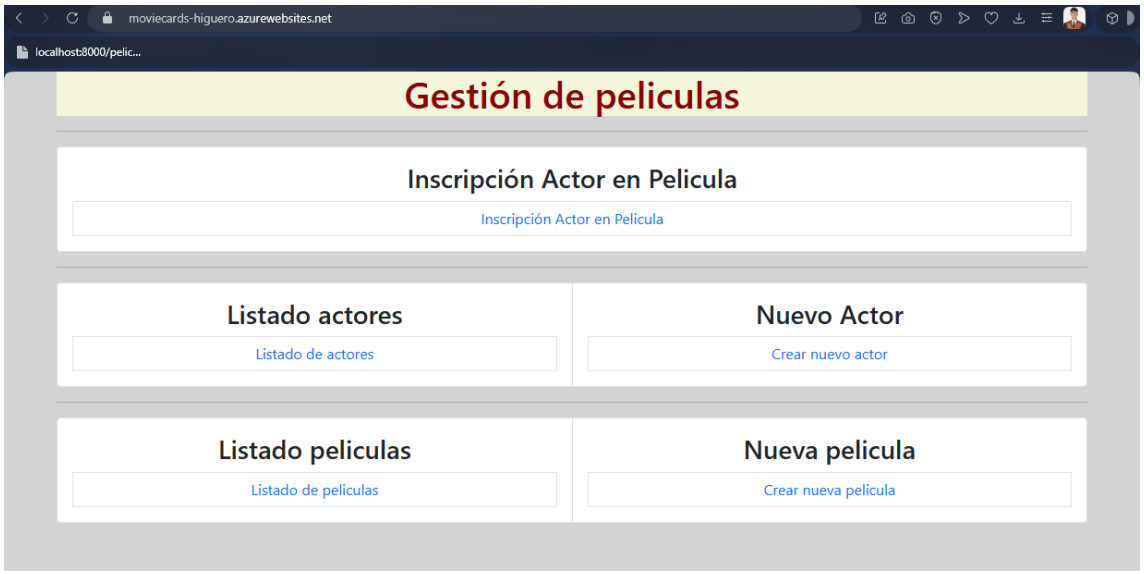


Tras resolver el conflicto de fusión, el merge con la rama main se completa correctamente.





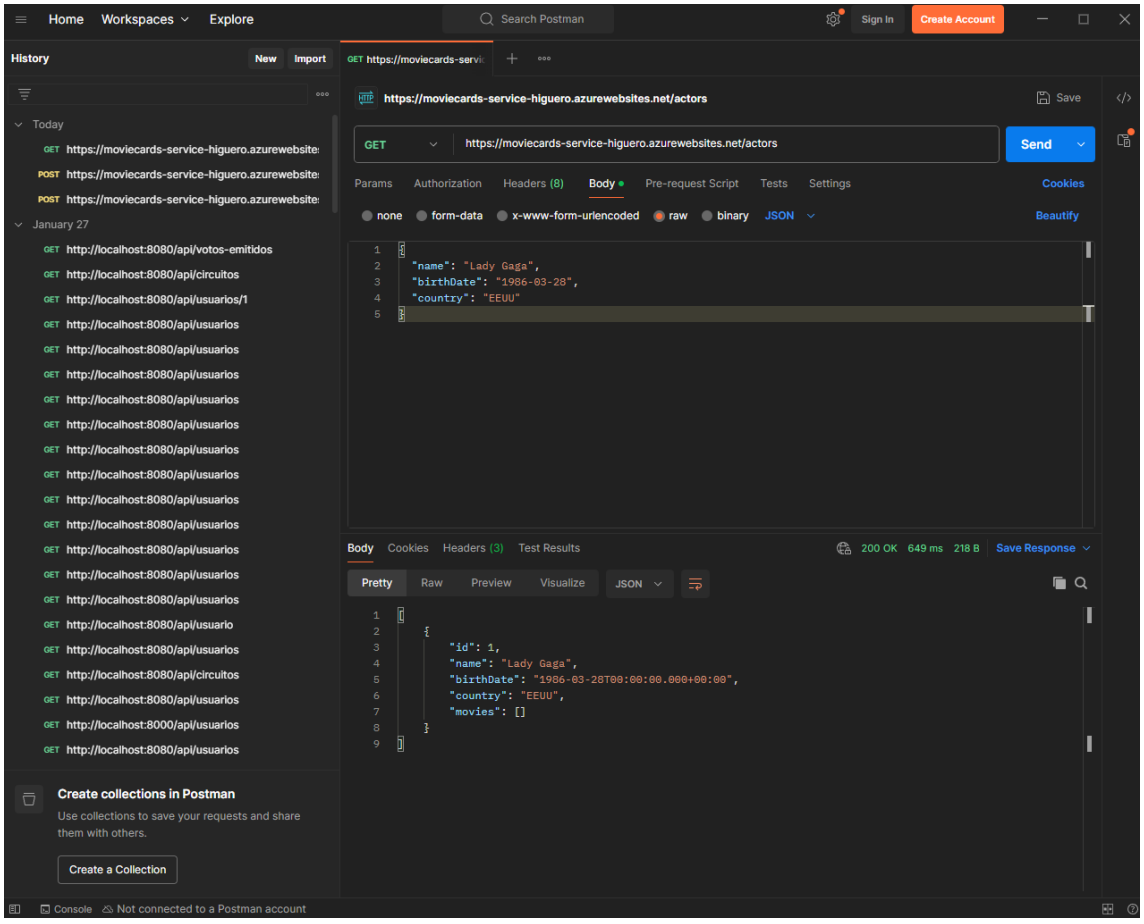
Una vez unido todo en la rama main se pueden visualizar los cambios finales en la página web de Azure.

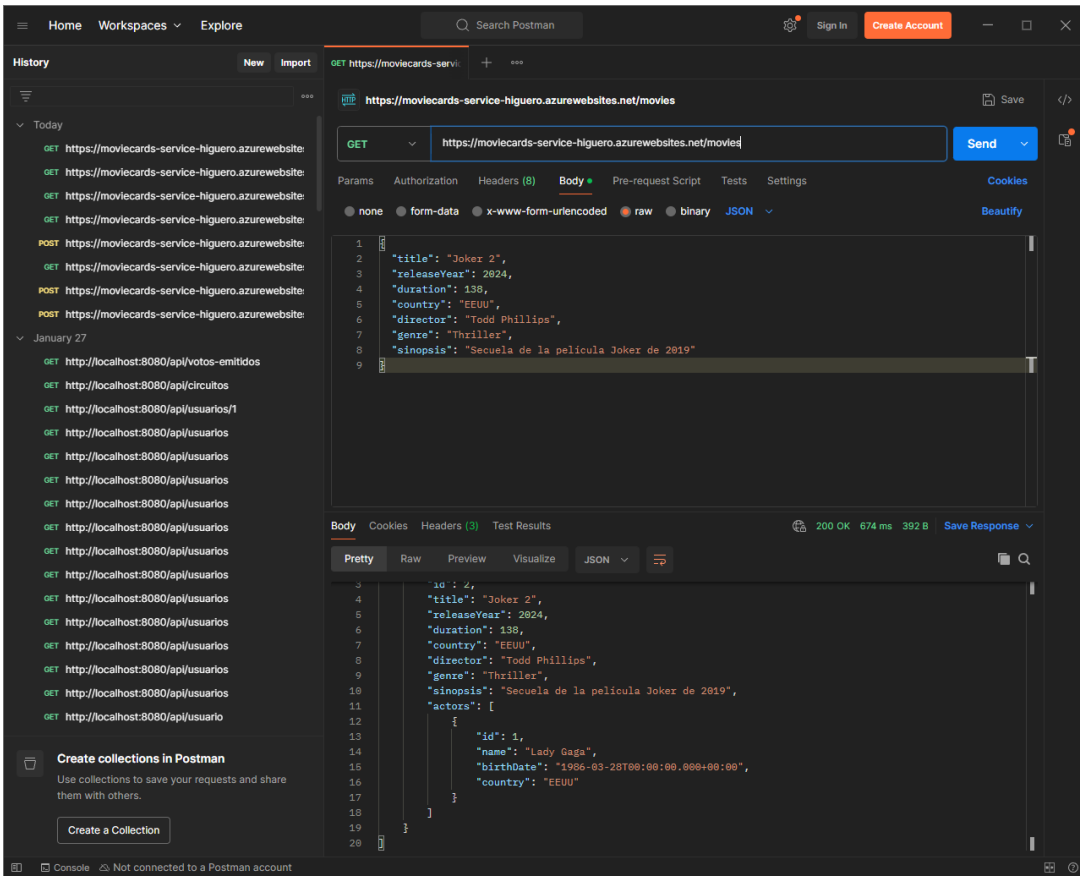

moviecards-higuero.azurewebsites.net
localhost:8000/pelic...
Gestión de peliculas
Inscripción Actor en Pelicula
Inscripción Actor en Pelicula
Listado actores
Listado de actores
Nuevo Actor
Crear nuevo actor
Listado peliculas
Listado de peliculas
Nueva pelicula
Crear nueva pelicula

3. Creación de moviecards-service

Siguiendo los requerimientos del proyecto, se ha procedido a la creación de un nuevo repositorio independiente denominado moviecards-service. Este repositorio contiene la lógica de negocio y el acceso a datos, siendo una API REST para la aplicación web principal.

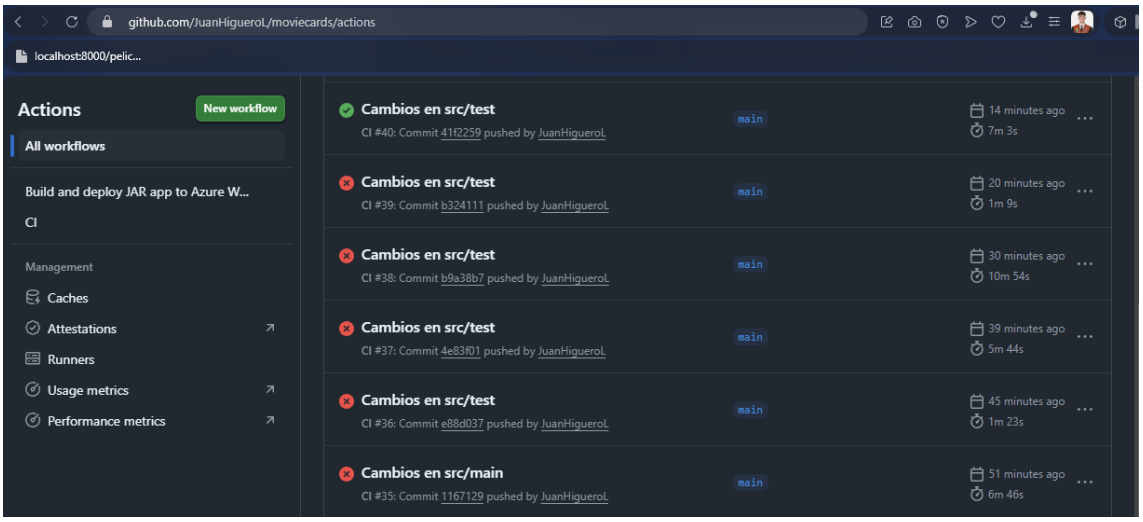
Para su puesta en marcha, se creó una nueva Web App en Azure específica para el servicio. El proceso de despliegue automatizado se replicó siguiendo la metodología establecida en la práctica inicial, asegurando una entrega continua en la URL de producción del servicio. La validación del sistema se realizó mediante Postman, confirmando el correcto funcionamiento de los métodos GET y POST tanto para actores como para películas, así como la funcionalidad de asociación entre ambos.





4. Refactorización de la Aplicación Principal

Con el servicio externo operativo, se inició la versión 2.0 de la aplicación moviecards. Para gestionar este cambio, se creó un nuevo Sprint en GitHub, dividiendo la tarea en dos historias de usuario principales: la modificación del código fuente en src/main para sustituir el acceso a base de datos por llamadas mediante RestTemplate, y la actualización de los tests en src/test para reflejar esta nueva arquitectura. A continuación, se muestra el historial de commits que demuestran su realización.



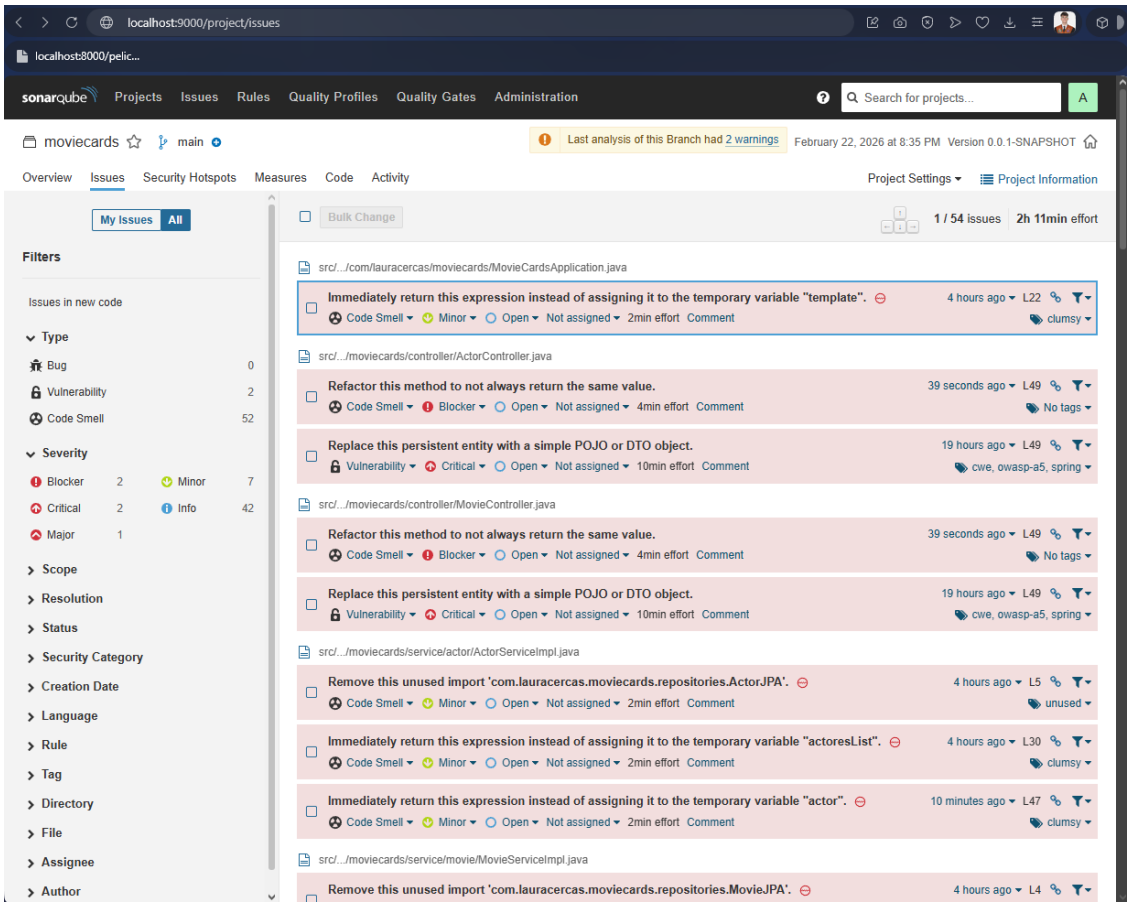
5. Implementación del Atributo deadDate

Para dar soporte a la fecha de fallecimiento, se modificó el código del repositorio moviecards-service. Se integró el atributo deadDate en la clase Actor, replicando la lógica existente para birthDate tanto en la declaración de variables como en sus métodos de acceso (getters y setters).

6. Entorno de Pre-producción y Ciclo de Calidad

La fase final del proyecto se centró en la integración visual de la nueva funcionalidad y la robustez del pipeline:

- **Entorno de Stage**: Se ha escalado la infraestructura en Azure creando una nueva aplicación denominada moviecards-pre-higuero. En el archivo main.yaml, se ha insertado un nuevo trabajo llamado stage entre las fases de "qa" y "deploy". Este trabajo utiliza un secreto específico de la nueva instancia de Azure para permitir la validación en un entorno de pre-producción antes de la salida a producción definitiva.
- **Integración de deadDate**: Se ha actualizado la interfaz de usuario para solicitar la fecha de fallecimiento en el formulario de creación y mostrarla en el listado general de actores. Este desarrollo se realizó en una rama independiente antes de su integración final en la rama main.
- **Batería de Pruebas**: Se han modificado los tests unitarios, de integración y funcionales (End-to-End) para validar que el sistema maneja correctamente el atributo deadDate en todas sus capas.
- **Garantía de Calidad y SonarQube**: Inicialmente, el proyecto presentaba 8 errores críticos; tras una refactorización de los controladores ActorController y MovieController se ha logrado reducir la cifra a solo 2 problemas críticos, superando con éxito el umbral de calidad exigido.


localhost:9000/project/issues
localhost:8000/pelic...
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
Search for projects...
A
moviecards
main
Last analysis of this Branch had 2 warnings
February 22, 2026 at 8:35 PM
Version 0.0.1-SNAPSHOT
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
My Issues
All
Filters
Issues in new code
Type
Bug 0
Vulnerability 2
Code Smell 52
Severity
Blocker 2
Minor 7
Critical 2
Info 42
Major 1
Scope
Resolution
Status
Security Category
Creation Date
Language
Rule
Tag
Directory
File
Assignee
Author
Bulk Change
1 / 54 issues
2h 11min effort
src/.../com/lauracercas/moviecards/MovieCardsApplication.java
Immediately return this expression instead of assigning it to the temporary variable "template".
4 hours ago L22
Code Smell Minor Open Not assigned 2min effort Comment
clumsy
src/.../moviecards/controller/ActorController.java
Refactor this method to not always return the same value.
39 seconds ago L49
Code Smell Blocker Open Not assigned 4min effort Comment
No tags
Replace this persistent entity with a simple POJO or DTO object.
19 hours ago L49
Vulnerability Critical Open Not assigned 10min effort Comment
cwe, owasp-a5, spring
src/.../moviecards/controller/MovieController.java
Refactor this method to not always return the same value.
39 seconds ago L49
Code Smell Blocker Open Not assigned 4min effort Comment
No tags
Replace this persistent entity with a simple POJO or DTO object.
19 hours ago L49
Vulnerability Critical Open Not assigned 10min effort Comment
cwe, owasp-a5, spring
src/.../moviecards/service/actor/ActorServiceImpl.java
Remove this unused import 'com.lauracercas.moviecards.repositories.ActorJPA'.
4 hours ago L5
Code Smell Minor Open Not assigned 2min effort Comment
unused
Immediately return this expression instead of assigning it to the temporary variable "actoresList".
4 hours ago L30
Code Smell Minor Open Not assigned 2min effort Comment
clumsy
Immediately return this expression instead of assigning it to the temporary variable "actor".
10 minutes ago L47
Code Smell Minor Open Not assigned 2min effort Comment
clumsy
src/.../moviecards/service/movie/MovieServiceImpl.java
Remove this unused import 'com.lauracercas.moviecards.repositories.MovieJPA'.
4 hours ago L4