Máster Universitario en Desarrollo Ágil de Software para la Web

Asignatura: Integración Continua en el Desarrollo Ágil

Curso 2024-2025

Trabajo Final

Autor: Jesús Ángel del Hoyo Gabaldón

Correo electrónico: jesus.hoyo@uah.es

Contenido

Índice de ilustraciones

Índice de código

1. Introducción

En este documento se explicarán los pasos seguidos por el alumno para completar la práctica final de la asignatura. Para ello, el alumno se ha apoyado en el manual del quinto tema, donde se explica la secuencia a seguir de forma exhaustiva y detallada, y que era necesario para completar con éxito la primera sección del presente trabajo.

Este proyecto comprende todos los apartados propuestos en el enunciado de la asignatura, puesto que se han completado todos ellos, y se explican en las siguientes secciones. Siguiendo las instrucciones del profesor, se han cumplido los siguientes objetivos:

1. Despliegue de la aplicación *moviecards* en Azure, accesible desde un navegador y a la que se pueden realizar consultas mediante Postman.
2. Despliegue del microservicio *moviecards-service* en Azure, al que se pueden realizar peticiones desde Postman.
3. Integración de *moviecards* con *moviecards-service* a través de su enlace de Azure. Los datos almacenados en *moviecards-service* se muestran desde la interfaz de *moviecards*. Además, se modifican las pruebas unitarias para probar el acceso a un servidor externo.
4. Modificación del código fuente de *moviecards-service* para añadir un nuevo atributo a los actores.
5. Adaptar la aplicación *moviecards* para mostrar el nuevo atributo en la aplicación. Además, se crea un entorno de pre-producción, se modifican las pruebas unitarias, de integración y end to end para considerar el nuevo atributo, y se corrigen los *code smells* críticos para que el código fuente cumpla con las quality gates de SonarQube.

El resto del documento se estructura de la siguiente manera: en la siguiente sección, se muestra el proceso seguido para desplegar el microservicio *moviecards* en Azure. Posteriormente, en la Sección 3, se explica cómo se ha desplegado *moviecards-service*. La Sección 4 indica los pasos realizados para integrar *moviecards-service* con *moviecards*. Por último, las Secciones 5 y 6 muestran cómo se añade un nuevo atributo a la clase de actores en *moviecards-service* y las consecuentes modificaciones en *moviecards* para poder visualizarlo, así como unas conclusiones finales sobre el trabajo en la Sección 7.

2. Despliegue de Moviecards

Para desplegar *moviecards*, se han seguido las instrucciones proporcionadas por los profesores en el tema quinto de la asignatura. El proceso no ha sufrido modificaciones sustanciales con respecto a dicha guía, por lo que este apartado no se describirá en detalle; en su lugar, se adjuntarán capturas de pantalla y otros recursos representativos que demuestran que el alumno ha seguido los pasos necesarios para el despliegue.

Además, la relación con todos los cambios se puede consultar en el siguiente repositorio, creado y gestionado por el alumno:

https://github.com/Jesus-4ngel/moviecards

A continuación, se describe este proceso adjuntando las evidencias y los recursos anteriormente mencionados.

Primero, se han seguido todos los pasos del manual hasta llegar a la creación del archivo YAML que define el workflow. Se destaca el despliegue del contenedor con SonarQube, como se muestra en la **Ilustración 1**. Una vez mostrado esto, se pasa a explicar, directamente, la parte del despliegue en Azure. El proceso de creación del repositorio y la preparación de toda la infraestructura es trivial, y queda evidenciado en el enlace previo, puesto que hay acceso público, por lo que no se detallará en esta explicación.

Para desplegar *moviecards* en Azure, ha sido necesario crear una nueva aplicación web gratuita, que es accesible desde el siguiente enlace:

https://moviecards-delhoyo.azurewebsites.net/

Para crear este recurso en línea, se han seguido los pasos del manual proporcionado como recurso, habiéndose realizado una única modificación necesaria: seleccionar la región East US 2 a la hora de la creación de la aplicación web, puesto que East US no estaba disponible y causaba errores. Una vez completado el proceso de despliegue al completo, se ha obtenido el archivo YAML mostrado en el **Código 1**. Con respecto a lo indicado en el manual, ha sido necesario actualizar la versión de una de las acciones (download-artifact), puesto que en la guía se indica la tercera versión, pero esta ha quedado obsoleta.

Este código mostrado realiza el siguiente proceso: compila la aplicación, ejecuta todas las pruebas sobre la aplicación, comprueba la calidad del código fuente mediante la integración con SonarQube y, finalmente, solicita al propio usuario que acepte o rechace el despliegue en Azure, teniendo así un proceso de entrega continua siempre que la rama *master* reciba cambios. Según la configuración, cuando el resto de ramas reciba modificaciones, se omitirá el trabajo de despliegue, puesto que esos cambios no han sido revisados y aceptados por un desarrollador para que pasen a la rama principal.

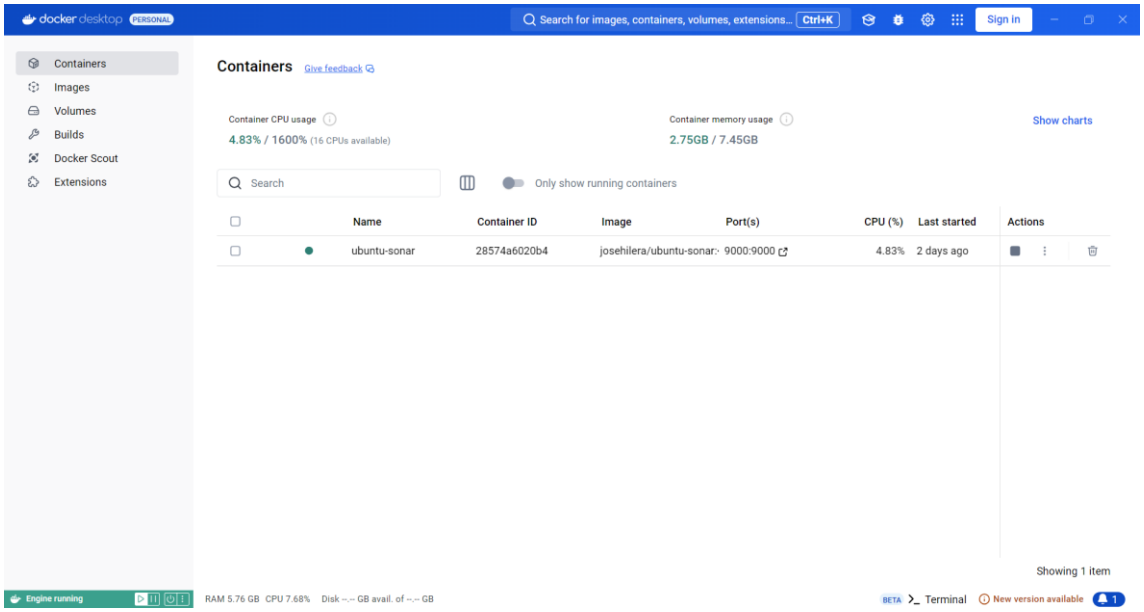

Ilustración 1: Despliegue del contenedor de SonarQube en Docker Desktop

```yaml
name: CI

on: push

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Descargar repositorio
        uses: actions/checkout@v2
      - name: Instalar JDK 11
        uses: actions/setup-java@v2
        with:
          java-version: "11"
          distribution: "adopt"
      - name: Construir con Maven
        run: mvn clean package -DskipTests
      - name: Guardar paquete generado para el trabajo de despliegue
        uses: actions/upload-artifact@v4
        with:
          name: moviecards-java
          path: "${{ github.workspace }}/target/*.jar"

  test:
    needs: build
    runs-on: ubuntu-latest
    steps:
      - name: Descargar repositorio
        uses: actions/checkout@v2
      - name: Instalar JDK 11
        uses: actions/setup-java@v2
        with:
          java-version: "11"
          distribution: "adopt"

      - name: Instalar Chrome y ChromeDriver para pruebas end to end
        run: |
          wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb

          sudo dpkg -i google-chrome-stable_current_amd64.deb

          sudo apt --fix-broken install -y

          CHROMEDRIVER_VERSION=$(curl -sS https://chromedriver.storage.googleapis.com/LATEST_RELEASE)

```

```
46          curl -L -o chromedriver.zip
    https://chromedriver.storage.googleapis.com/$CHROMEDRIVER_VERSION
    /chromedriver_linux64.zip
47
48          unzip chromedriver.zip
49
50          chmod +x chromedriver
51
52          sudo mv chromedriver /usr/local/bin/
53
54      - name: Ejecutar la aplicación para pruebas end to end
55        run: mvn spring-boot:run & sleep 60
56
57      - name: Ejecutar las pruebas unitarias, de integración y
    end to end
58        run: mvn clean verify
59
60  qa:
61    needs: test
62    runs-on: self-hosted
63    continue-on-error: true
64    steps:
65      - name: Descargar repositorio
66        uses: actions/checkout@v2
67
68      - name: Instalar JDK 11
69        uses: actions/setup-java@v2
70        with:
71          java-version: "11"
72          distribution: "adopt"
73
74      - name: Construir con Maven
75        run: mvn clean package -DskipTests
76
77      - name: Revisar la calidad con Sonarqube
78        run: |
79          mvn sonar:sonar -Dsonar.host.url=http://localhost:9000
    -Dsonar.qualitygate.wait=true -Dsonar.login=admin -
    Dsonar.password=admin
80
81  deploy:
82    runs-on: ubuntu-latest
83    needs: qa
84    if: github.ref=='refs/heads/master'
85    environment:
86      name: 'Production'
87      url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}
88
89    steps:
```

```
90        - name: Aprobación manual
91          uses: trstringer/manual-approval@v1
92          with:
93            secret: ${{ secrets.TOKEN }}
94            approvers: Jesus-4ngel
95        - name: Download artifact from build job
96          uses: actions/download-artifact@v4
97          with:
98            name: moviecards-java
99
100       - name: Deploy to Azure Web App
101         id: deploy-to-webapp
102         uses: azure/webapps-deploy@v3
103         with:
104           app-name: 'moviecards-delhoyo'
105           slot-name: 'Production'
106           package: '*.jar'
107           publish-profile: ${{
    secrets.AZUREAPPSERVICE_PUBLISHPROFILE_049D6A1D80B74B53B50362007D
    7B79A9 }}
```

Código 1: Workflow obtenido para el despliegue de *moviecards*

A partir de este momento, todos los cambios que se realicen en la rama *master*, como se ha mencionado anteriormente, desencadenará el workflow al completo. Este comportamiento se muestra en las imágenes expuestas a continuación: en la **Ilustración 2** se muestra la ejecución exitosa del workflow, a excepción del último paso, que requiere de confirmación manual; la **Ilustración 3** muestra el proceso de confirmación del despliegue, que se refleja en la **Ilustración 4**, donde se muestra el workflow completo ejecutado exitosamente; y, por último, la **Ilustración 5** muestra *moviecards* desplegada en Azure.

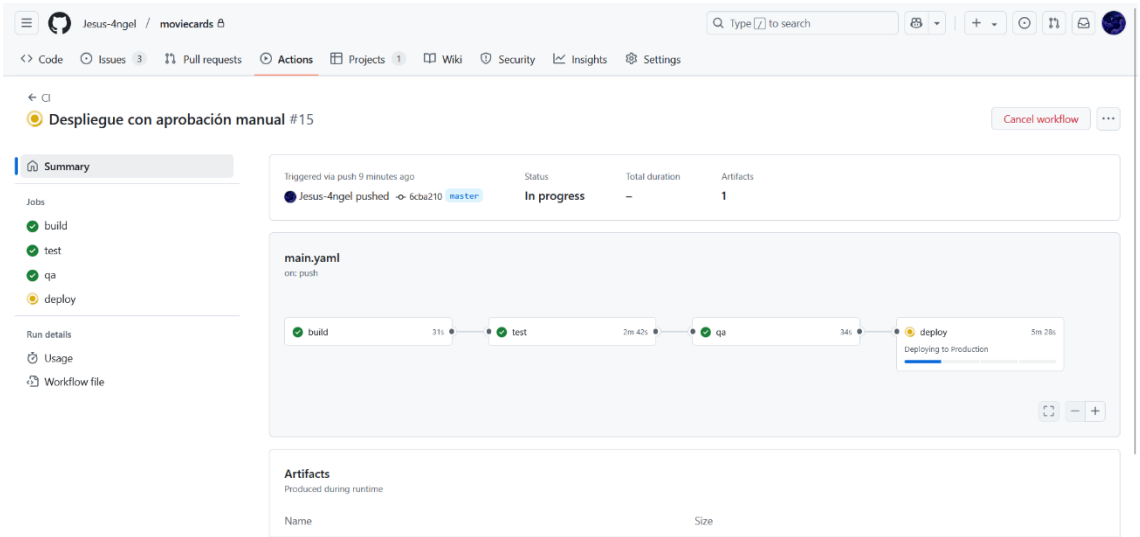

Ilustración 2: Ejecución parcial del workflow a falta de confirmación manual

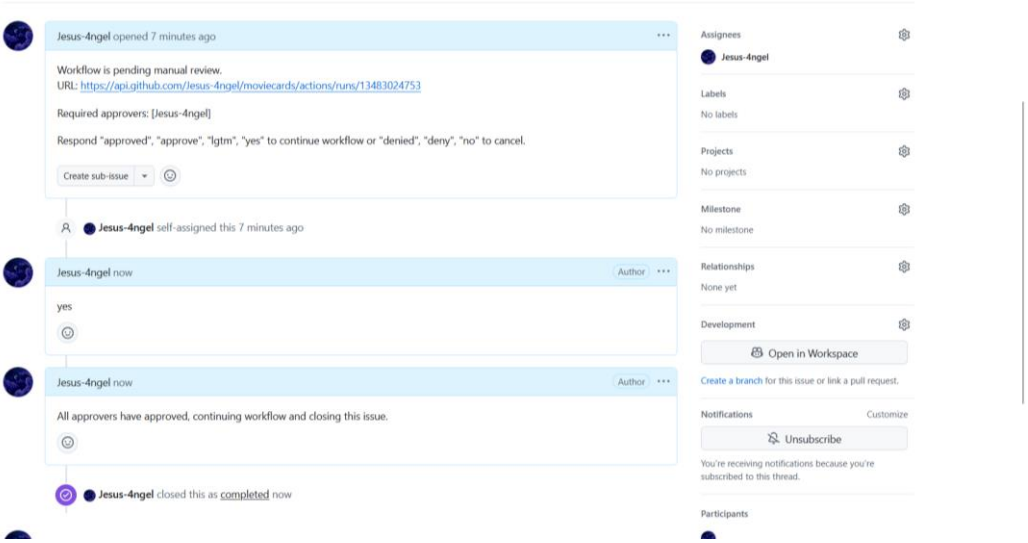

Ilustración 3: Confirmación manual del despliegue

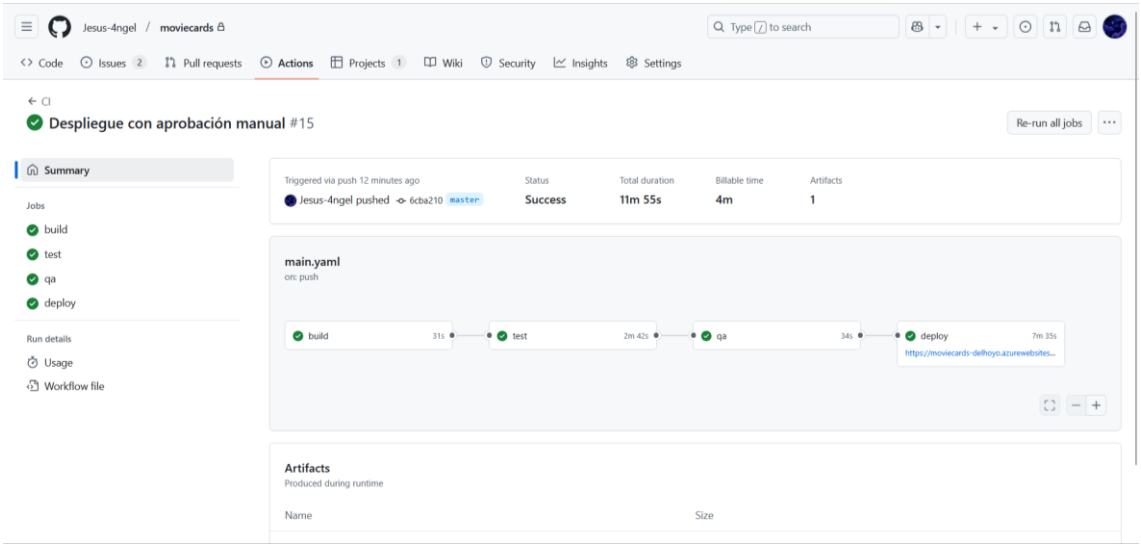

Ilustración 4: Ejecución completa del workflow

Ilustración 5: Demostración del despliegue de la aplicación en Azure

Una vez completado este proceso, lo siguiente es realizar el primer sprint con la modificación de la aplicación. A continuación, se adjuntan evidencias de este procedimiento. Primero, la **Ilustración *6*** e **Ilustración *7*** muestran la creación de las tareas (issues) en el marco del proyecto. Como se puede comprobar, las tareas se están asignando al proyecto y al milestone creado dentro del proyecto. Posteriormente, la **Ilustración *8*** muestra cómo se desplazan las tareas a la sección “En progreso”, puesto que comienza su implementación y, finalmente, se muestra en la **Ilustración *9*** cómo se integran los cambios conflictivos en la rama *master*, la aceptación del *Pull Request* en la **Ilustración *10*** tras solucionar los conflictos, el desplazamiento de las tareas a la sección de completadas en la **Ilustración *11*** y, por último, el sprint finalizado en la **Ilustración *12***. Como se demuestra en la **Ilustración *13***, los cambios de estilo se han aplicado exitosamente sobre *moviecards* desplegado en Azure.

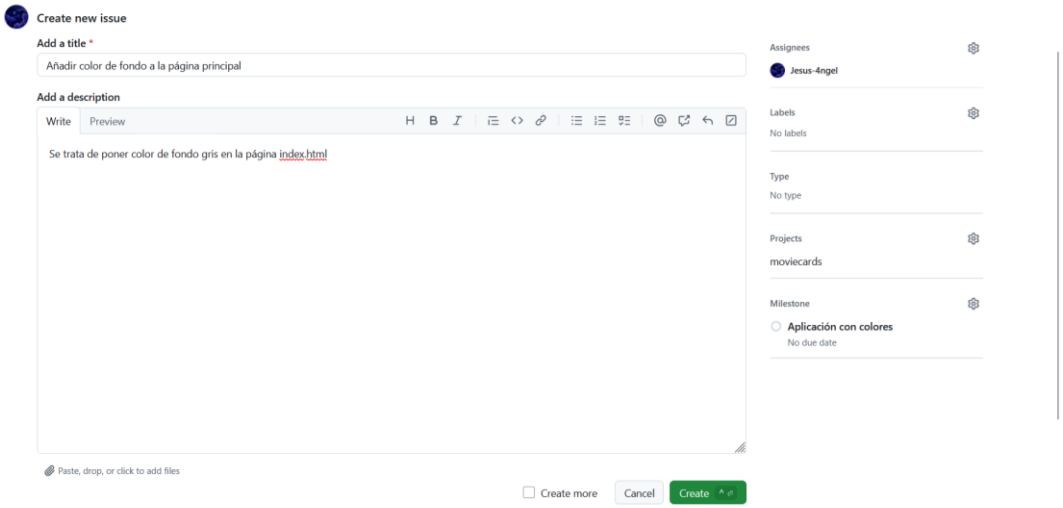

Ilustración 6: Creación de la primera tarea

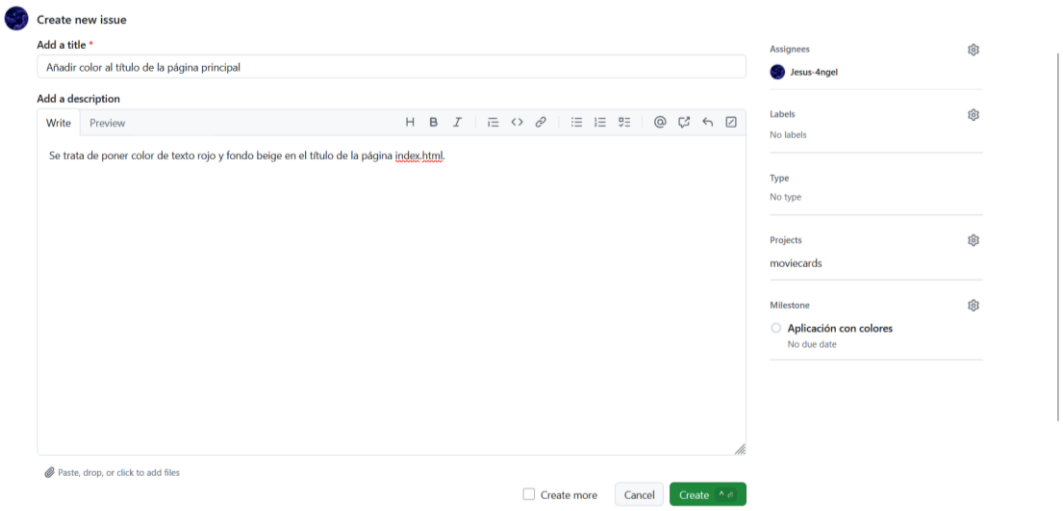

Ilustración 7: Creación de la segunda tarea

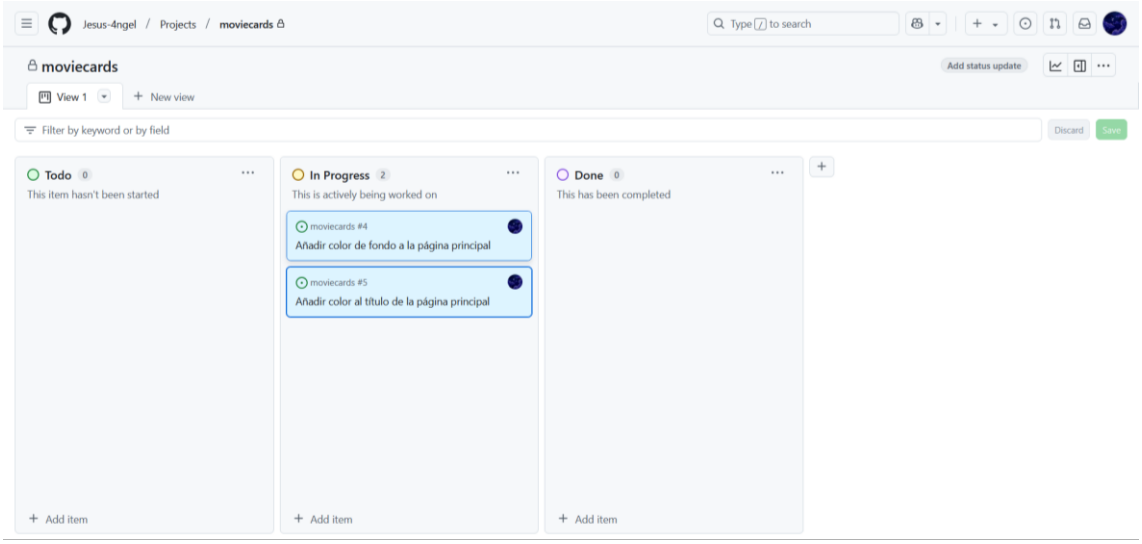

Ilustración 8: Tareas desplazadas a la sección de "En Progreso"

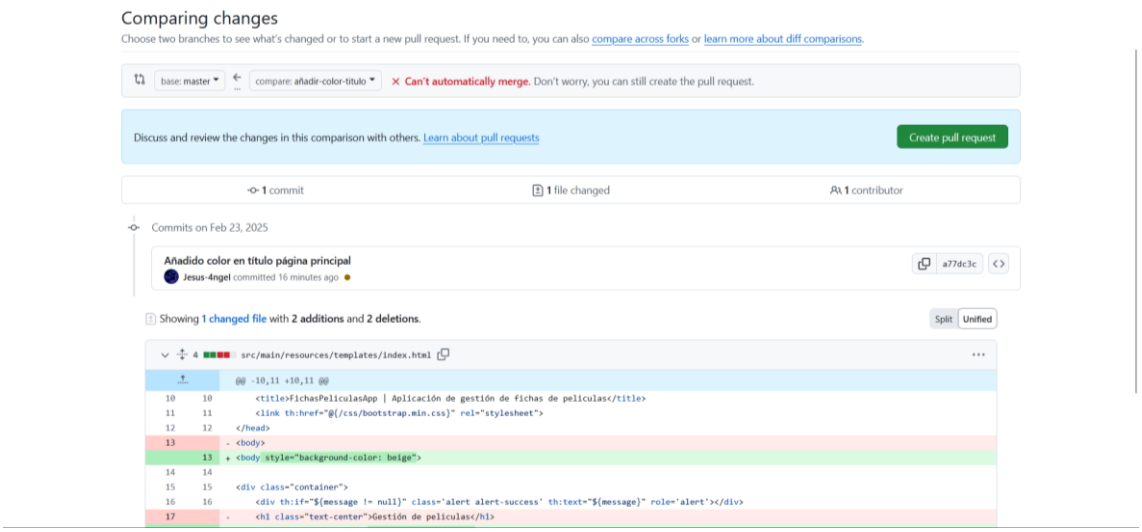

Ilustración 9: Pull Request con cambios conflictivos que demuestra ambos commits

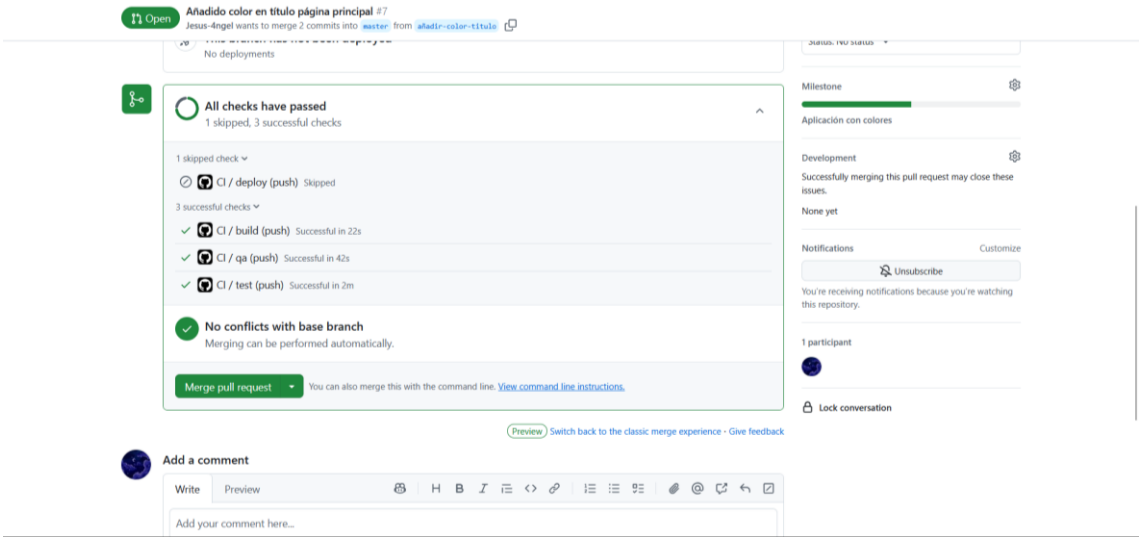

Ilustración 10: Confirmación del Pull Request sobre master tras solucionar conflictos

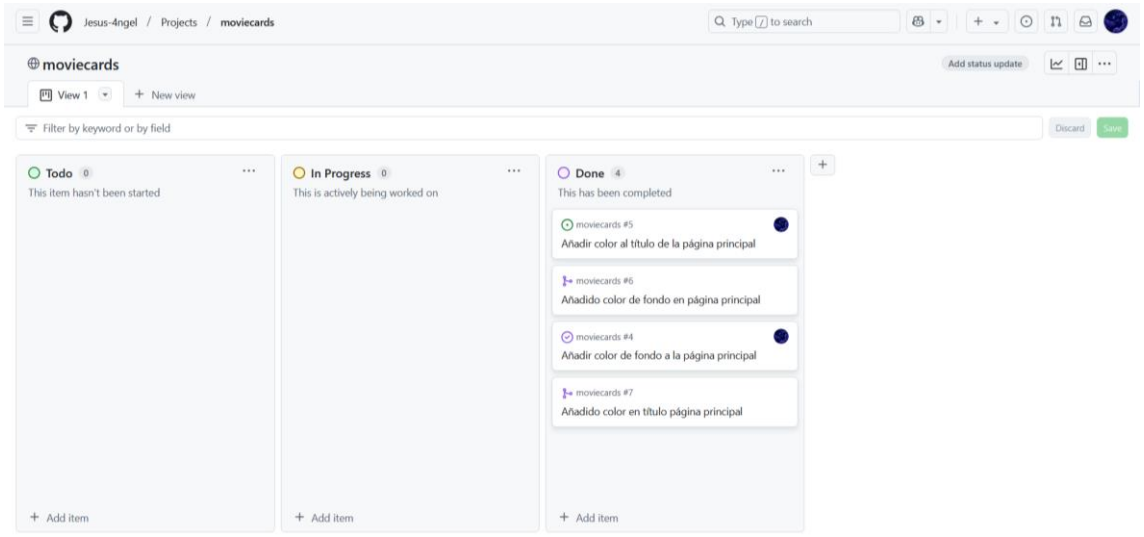

Ilustración 11: Desplazamiento de tareas a la sección "Completado"

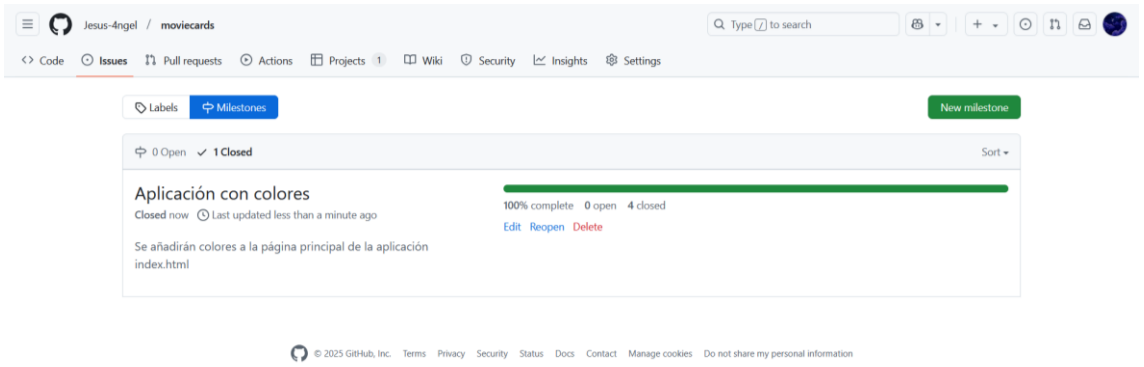

Ilustración 12: Primer sprint (milestone) finalizado con éxito

Ilustración 13: Despliegue de los cambios sobre moviecards en Azure

Por último, si accedemos a Postman, podemos hacer peticiones a este enlace, lo que nos devolverá, como resultado, el código HTML de las diferentes vistas, o los datos que requiramos, dependiendo del *endpoint* al que se haga referencia. En la **Ilustración *14***, se muestra una petición mediante Postman a la vista principal de la aplicación, que devuelve su código HTML.

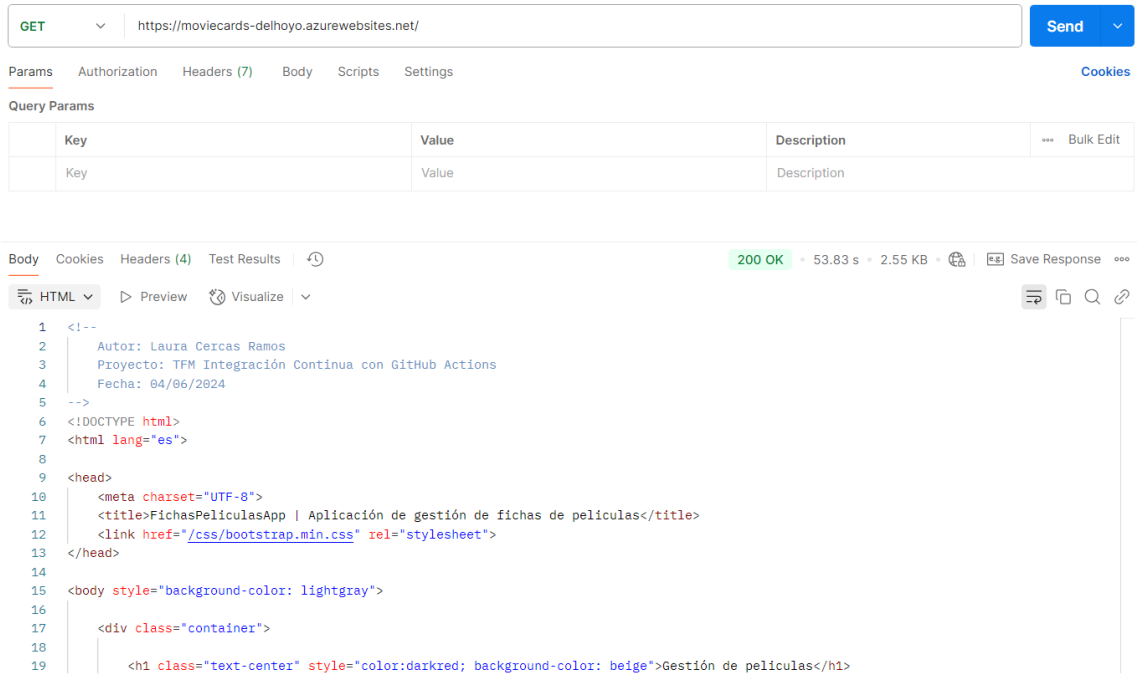

Ilustración 14: Petición GET mediante Postman de la vista inicial de la aplicación

NOTA: las pruebas que puedan realizarse para garantizar el funcionamiento de esta sección van a considerar el atributo *deadDate* para actores, puesto que se han cumplido los objetivos propuestos en los futuros apartados del proyecto.

3. Despliegue de Moviecards-service

Para desplegar este nuevo microservicio en Azure, se ha realizado el mismo procedimiento que en el caso anterior, pero de forma simplificada, puesto que muchos de los recursos necesarios para su despliegue se han podido aprovechar puesto que ya estaban disponibles. Más concretamente, el proceso seguido por el alumno ha sido el siguiente:

1. Descarga del código fuente proporcionado por el profesor en el repositorio https://github.com/josehilera/moviecards-service/tree/master.
2. Creación de repositorio público para alojar la aplicación *moviecards-service*, disponible en https://github.com/Jesus-4ngel/moviecards-service.
3. Creación de una nueva aplicación web en Azure donde se vinculará y desplegará el microservicio, siguiendo las pautas indicadas anteriormente. Una vez creada, se asocia con el repositorio y se modifica el archivo *main.yaml* (**Código 2**) que se incluía en el proyecto proporcionado por el profesor para incluir las modificaciones automáticas realizadas por Azure.
4. Finalmente, tras realizar todas las modificaciones, se envían los cambios al repositorio y se espera a que se realice el despliegue del microservicio, que habrá que confirmar manualmente. La aplicación se despliega en la siguiente

dirección: https://moviecards-service-delhoyo.azurewebsites.net/. Tras esto, se pueden realizar las pruebas pertinentes mediante Postman.

```yaml
name: Continuous Integration (CI)

on: push

jobs:
  build:
    name: Build and Package
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up JDK 11
        uses: actions/setup-java@v2
        with:
          java-version: "11"
          distribution: "adopt"

      - name: Build with Maven
        run: mvn clean package -DskipTests

      - name: Upload artifact for deployment job
        uses: actions/upload-artifact@v4
        with:
          name: moviecards-service-java
          path: "${{ github.workspace }}/target/*.jar"

  test:
    name: Test
    needs: build
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up JDK 11
        uses: actions/setup-java@v2
        with:
          java-version: "11"
          distribution: "adopt"

      - name: Run tests
        run: mvn clean verify

  deploy:
    runs-on: ubuntu-latest
```

```
47      needs: test
48      if: github.ref=='refs/heads/master'
49      environment:
50        name: 'Production'
51        url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}
52
53      steps:
54        - name: Aprobación manual
55          uses: trstringer/manual-approval@v1
56          with:
57            secret: ${{ secrets.TOKEN }}
58            approvers: Jesus-4ngel
59        - name: Download artifact from build job
60          uses: actions/download-artifact@v4
61          with:
62            name: moviecards-service-java
63
64        - name: Deploy to Azure Web App
65          id: deploy-to-webapp
66          uses: azure/webapps-deploy@v3
67          with:
68            app-name: 'moviecards-service-delhoyo'
69            slot-name: 'Production'
70            package: '*.jar'
71            publish-profile: ${{
  secrets.AZUREAPPSERVICE_PUBLISHPROFILE_A27A1AE9F52741FAB8C165CCA8CAEA30 }}
```

Código 2: Workflow obtenido para el despliegue de *moviecards-service*

Para demostrar el funcionamiento, se incluye la creación mediante Postman de un actor y la consulta de todos los actores en la base de datos. La primera consulta se encuentra en la **Ilustración 15** y la segunda consulta se muestra en la **Ilustración 16**.

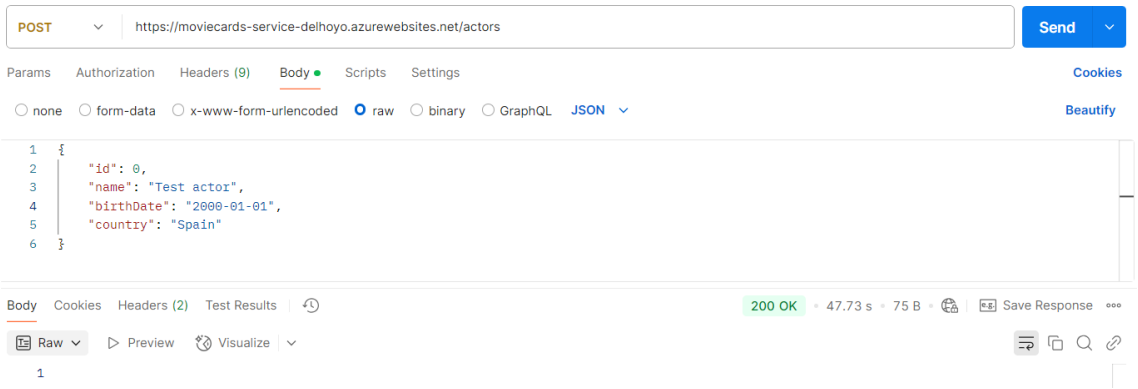

Ilustración 15: Creación de un actor de prueba en *moviecards-service*

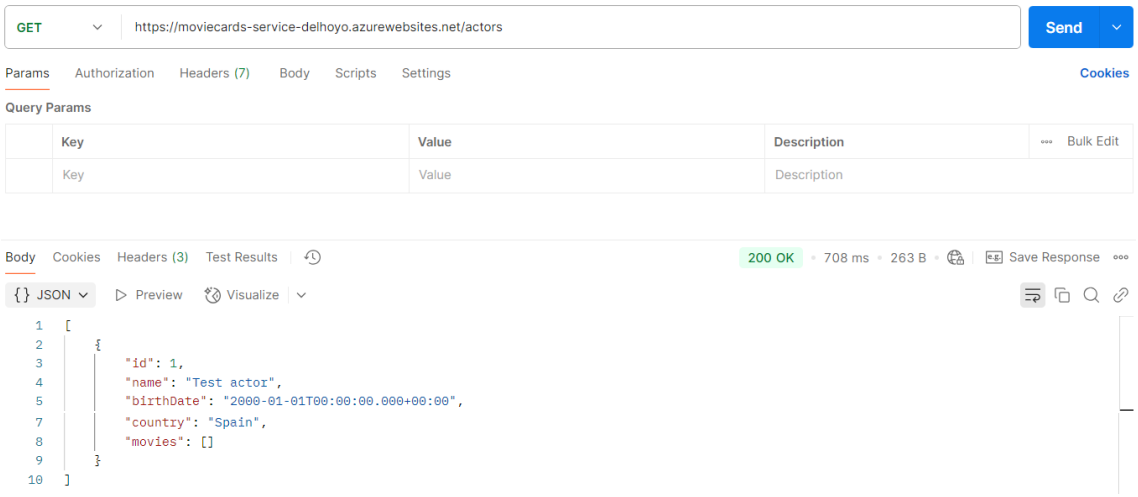

Ilustración 16: Consulta de actores en *moviecards-service*

NOTA: las pruebas que puedan realizarse para garantizar el funcionamiento de esta sección van a considerar el atributo *deadDate* para actores, puesto que se han cumplido los objetivos propuestos en los futuros apartados del proyecto.

4. Conexión de Moviecards con Moviecards-service

Esta sección se corresponde con el tercer apartado de la práctica, que solicita la comunicación de *moviecards* con *moviecards-service* a través de su enlace de Azure. Para ello, se ha aprovechado toda la infraestructura preparada en las secciones anteriores, y solo ha sido necesario realizar el seguimiento del proyecto mediante un nuevo sprint y la actualización de código fuente que, posteriormente, se envía al repositorio de *moviecards* para que se ejecute el workflow y los cambios tengan efecto. A continuación, se adjuntan capturas de pantalla representativas que muestran las tareas creadas dentro del nuevo sprint (**Ilustración 17** e **Ilustración 18**) y que se han abordado mediante la creación de ramas independientes, el resultado final del sprint (**Ilustración 19**) y el sprint finalizado (**Ilustración 20**). Además, se adjunta el código que se ha necesitado modificar tanto para la parte funcional como para la parte de los tests.

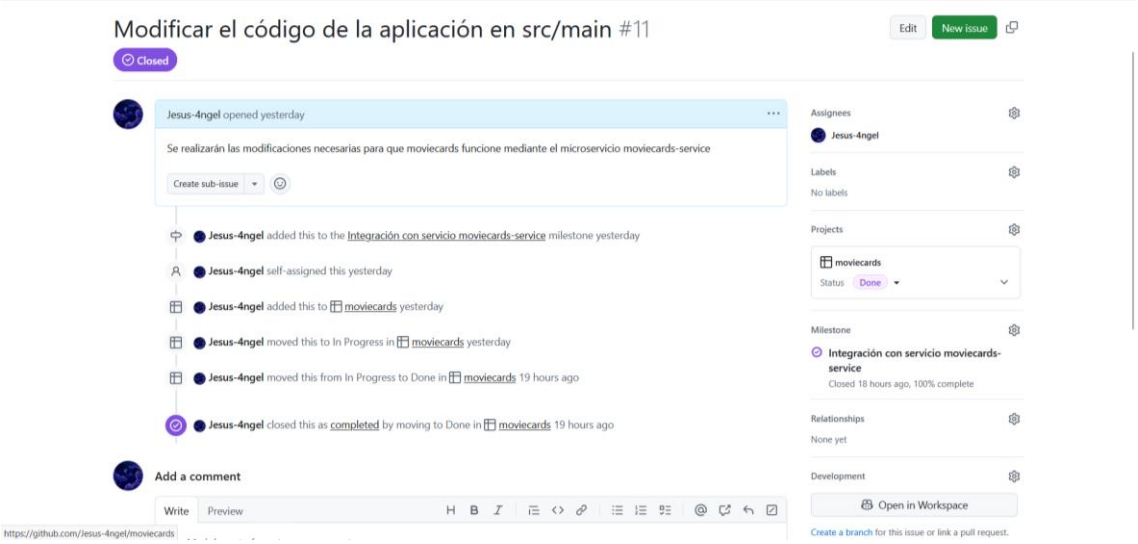

Ilustración 17: Creación de tarea de modificación de código en main

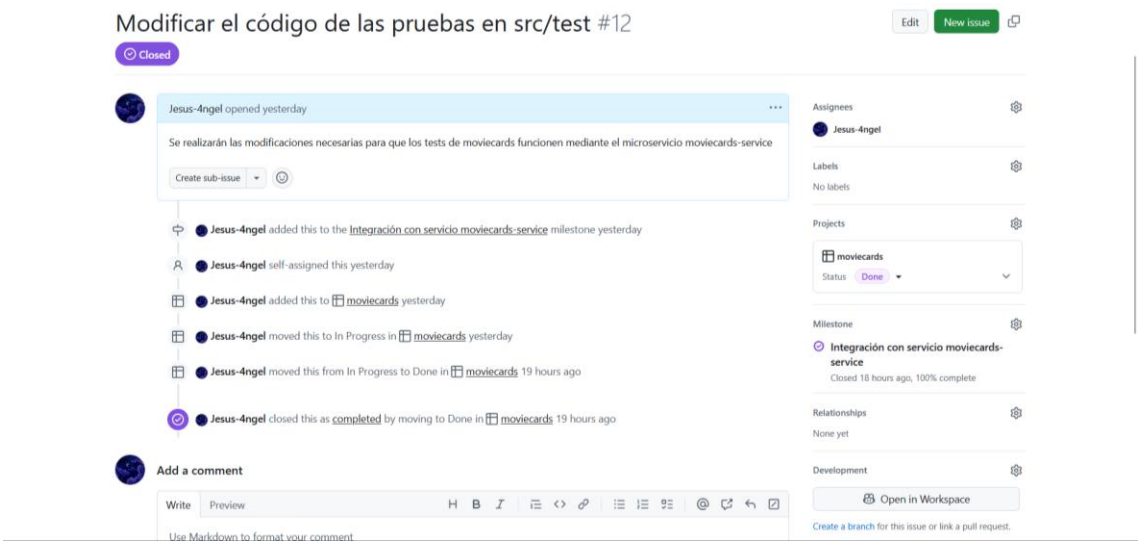

Ilustración 18: Creación de tarea de modificar el código en test

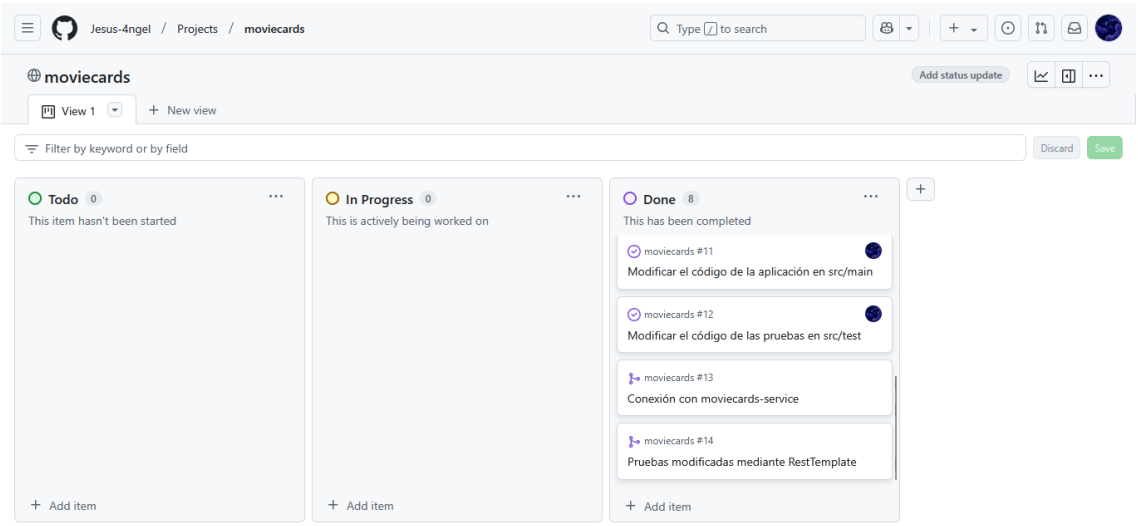

Ilustración 19: Tareas del segundo sprint terminadas

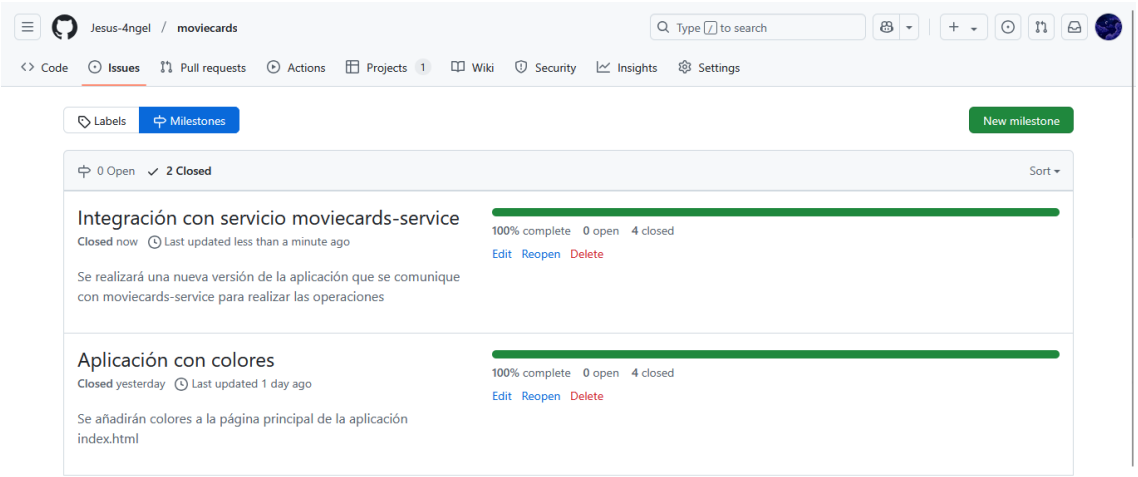

Ilustración 20: Finalización del segundo sprint

En cuanto al código fuente modificado, ha sido necesario cambiar cinco clases, como se explica a continuación.

Modificación de la clase principal

En el **Código 3** se muestra la modificación realizada sobre la clase *src/main/java/com/lauracercas/moviecards/MovieCardsApplication.java*. En ella, se ha añadido el Bean template, que permitirá realizar peticiones a direcciones remotas (Líneas 13-17, marcadas en verde).

```
1 /**
2  * Autor: Laura Cercas Ramos
3  * Proyecto: TFM Integración Continua con GitHub Actions
4  * Fecha: 04/06/2024
5  */
6 @SpringBootApplication
7 public class MovieCardsApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(MovieCardsApplication.class, args);
11     }
12
13     @Bean
14     public RestTemplate template() {
15         RestTemplate template = new RestTemplate();
16         return template;
17     }
18
19 }
```

Código 3: Modificación sobre clase principal para conexión con *moviecards-service*

Modificación de la implementación del servicio de actores

En el **Código 4** se muestran las modificaciones realizadas sobre la clase *src/main/java/com/lauracercas/moviecards/service/actor/ActorServiceImpl.java*. En ella, se ha añadido el RestTemplate, autoinyectado gracias a SpringBoot. Este permitirá realizar las peticiones correspondientes al enlace establecido, obteniendo o enviando, así, los JSONs con los datos pertinentes. Esta clase se ha modificado prácticamente entera, por lo que no se mostrarán las diferencias con respecto a la original.

```
1 /**
2  * Autor: Laura Cercas Ramos
3  * Proyecto: TFM Integración Continua con GitHub Actions
4  * Fecha: 04/06/2024
5  */
6 @Service
7 public class ActorServiceImpl implements ActorService {
8
9     private final RestTemplate template;
10    String url = "https://moviecards-service-
  delhoyo.azurewebsites.net/actors";
11
```

```
12      @Autowired
13      public ActorServiceImpl(RestTemplate template) {
14          this.template = template;
15      }
16
17      @Override
18      public List<Actor> getAllActors() {
19          Actor[] actores = template.getForObject(url,
    Actor[].class);
20          List<Actor> actorList = Arrays.asList(actores);
21
22          return actorList;
23      }
24
25      @Override
26      public Actor save(Actor actor) {
27          Actor savedActor = template.postForObject(url, actor,
    Actor.class);
28          return savedActor;
29      }
30
31      @Override
32      public Actor getActorById(Integer actorId) {
33          Actor actor = template.getForObject(url + "/" + actorId,
    Actor.class);
34          return actor;
35      }
36  }
```

Código 4: Modificación sobre servicio actor para conexión con *moviecards-service*

Modificación de la implementación del servicio de películas

Del mismo modo que en el caso anterior, el **Código 5** muestra las modificaciones realizadas sobre la clase *src/main/java/com/lauracercas/moviecards/service/movie/MovieServiceImpl.java*.

```
1  /**
2   * Autor: Laura Cercas Ramos
3   * Proyecto: TFM Integración Continua con GitHub Actions
4   * Fecha: 04/06/2024
5   */
6  @Service
7  public class MovieServiceImpl implements MovieService {
8
9      private final RestTemplate template;
10
11     String url = "https://moviecards-service-
    delhoyo.azurewebsites.net/movies";
12
13     @Autowired
```

```
14      public MovieServiceImpl(RestTemplate template) {
15          this.template = template;
16      }
17
18      @Override
19      public List<Movie> getAllMovies() {
20          Movie[] movies = template.getForObject(url, Movie[].class);
21          List<Movie> movieList = List.of(movies);
22
23          return movieList;
24      }
25
26      @Override
27      public Movie save(Movie movie) {
28          Movie savedMovie = template.postForObject(url, movie,
    Movie.class);
29          return savedMovie;
30      }
31
32      @Override
33      public Movie getMovieById(Integer movieId) {
34          Movie movie = template.getForObject(url + "/" + movieId,
    Movie.class);
35          return movie;
36      }
37  }
```

Código 5: Modificación sobre servicio películas para conexión con *moviecards-service*

Modificación de las pruebas unitarias para actores

Originalmente, la clase *src/test/java/com/lauracercas/moviecards/unittest/service/ActorServiceImplTest.java* realizaba pruebas unitarias en las que se probaba el funcionamiento de la clase JPA para la persistencia. No obstante, puesto que el front-end no es el encargado de realizar esta persistencia, sino que lo hará el microservicio *moviecards-service*, se deben sustituir todas las referencias a ese JPA por referencias al RestTemplate que se usa realmente. Además, para que los cambios realizados durante las pruebas no se envíen al entorno real donde se tiene desplegada la aplicación, se utiliza un servidor REST de prueba que actúa como si fuera el real. El resultado final se muestra en el **Código 6**.

```
1  /**
2   * Autor: Laura Cercas Ramos
3   * Proyecto: TFM Integración Continua con GitHub Actions
4   * Fecha: 04/06/2024
5   */
6  class ActorServiceImplTest {
7
8      private MockRestServiceServer mockServer;
9      private RestTemplate restTemplate = new RestTemplate();
10
```

```
11      @Mock
12      private ActorServiceImpl sut;
13      private AutoCloseable closeable;
14
15      private static final String BASE_URL = "https://moviecards-
    service-delhoyo.azurewebsites.net/actors";
16
17      @BeforeEach
18      void setUp() {
19          closeable = openMocks(this);
20          mockServer =
    MockRestServiceServer.createServer(restTemplate);
21          sut = new ActorServiceImpl(restTemplate);
22      }
23
24      @AfterEach
25      void tearDown() throws Exception {
26          closeable.close();
27      }
28
29      @Test
30      public void shouldGetAllActors() {
31
32          String jsonResponse = "[{\"id\":1, \"name\":\"Sample
    Actor\", \"birthDate\":\"2000-01-01T23:00:00.000+00:00\",
    \"country\":\"Spain\", \"movies\":[]}]";

33
34  mockServer.expect(MockRestRequestMatchers.requestTo(BASE_URL))

35  .andRespond(MockRestResponseCreators.withSuccess(jsonResponse,
    MediaType.APPLICATION_JSON));
36
37          List<Actor> result = sut.getAllActors();
38
39          assertEquals(1, result.size());
40
41          mockServer.verify();
42      }
43
44      @Test
45      public void shouldGetActorById() {
46
47          String jsonResponse = "{\"id\":1, \"name\":\"Sample
    Actor\", \"birthDate\":\"2000-01-01T23:00:00.000+00:00\",
    \"country\":\"Spain\", \"movies\":[]}";

48
49
```

```
50
51 mockServer.expect(MockRestRequestMatchers.requestTo(BASE_URL +
   "/1"))

52 .andRespond(MockRestResponseCreators.withSuccess(jsonResponse,
   MediaType.APPLICATION_JSON));
53
54         Actor result = sut.getActorById(1);
55
56         assertEquals(1, result.getId());
57         assertEquals("Sample Actor", result.getName());
58
59         mockServer.verify();
60     }
61
62     @Test
63     public void shouldSaveActor() throws Exception {
64         String jsonRequest = "{\"name\":\"New Actor\"}";
65         String jsonResponse = "{\"id\":1, \"name\":\"New Actor\"}";
66
67
68 mockServer.expect(MockRestRequestMatchers.requestTo(BASE_URL))

69 .andExpect(MockRestRequestMatchers.method(HttpMethod.POST))

   .andExpect(MockRestRequestMatchers.content().json(jsonRequest))

   .andRespond(MockRestResponseCreators.withSuccess(jsonResponse,
   MediaType.APPLICATION_JSON));

70         Actor actor = new Actor();
71         actor.setName("New Actor");
72
73         Actor result = sut.save(actor);
74         assertEquals("New Actor", result.getName());
75
76         mockServer.verify();
77     }
78
79 }
```

Código 6: Modificación sobre pruebas unitarias actores para conexión con *moviecards-service*

Modificación de las pruebas unitarias para películas

Del mismo modo que para el caso anterior, el **Código 7** muestra los cambios realizados para las pruebas *src/test/java/com/lauracercas/moviecards/unittest/service/MovieServiceImplTest.java*

```
1 /**
2  * Autor: Laura Cercas Ramos
```

```
 3  * Proyecto: TFM Integración Continua con GitHub Actions
 4  * Fecha: 04/06/2024
 5  */
 6 class MovieServiceImplTest {
 7
 8     private MockRestServiceServer mockServer;
 9     private RestTemplate restTemplate = new RestTemplate();
10
11     @Mock
12     private MovieServiceImpl sut;
13     private AutoCloseable closeable;
14
15     private static final String BASE_URL = "https://moviecards-
   service-delhoyo.azurewebsites.net/movies";
16
17     @BeforeEach
18     public void setUp() {
19         closeable = openMocks(this);
20         mockServer =
   MockRestServiceServer.createServer(restTemplate);
21         sut = new MovieServiceImpl(restTemplate);
22     }
23
24     @AfterEach
25     void tearDown() throws Exception {
26         closeable.close();
27     }
28
29     @Test
30     public void shouldGetAllMovies() {
31
32         String jsonResponse = "[{\"id\":1, \"title\":\"Sample
   Movie\", \"releaseDate\":\"2000-01-01T23:00:00.000+00:00\",
   \"genre\":\"Action\", \"actors\":[]}, {\"id\":2, \"title\":\"Sample
   Movie 2\", \"releaseDate\":\"2000-01-01T23:00:00.000+00:00\",
   \"genre\":\"Action\", \"actors\":[]}]";

33
34 mockServer.expect(MockRestRequestMatchers.requestTo(BASE_URL))

35 .andRespond(MockRestResponseCreators.withSuccess(jsonResponse,
   MediaType.APPLICATION_JSON));
36
37         List<Movie> result = sut.getAllMovies();
38
39         assertEquals(2, result.size());
40
41         mockServer.verify();
42     }
```

```
43
44      @Test
45      public void shouldGetMovieById() {
46
47          String jsonResponse = "{\"id\":1, \"title\":\"Sample
    Movie\", \"releaseDate\":\"2000-01-01T23:00:00.000+00:00\",
    \"genre\":\"Action\", \"actors\":[]}";


    mockServer.expect(MockRestRequestMatchers.requestTo(BASE_URL +
48  "/1"))

    .andRespond(MockRestResponseCreators.withSuccess(jsonResponse,
49  MediaType.APPLICATION_JSON));

50          Movie result = sut.getMovieById(1);
51
52          assertEquals(1, result.getId());
53          assertEquals("Sample Movie", result.getTitle());
54      }
55
56      @Test
57      public void shouldSaveMovie() {
58
59          String jsonResponse = "{\"id\":1, \"title\":\"New
60  Movie\"}";


61  mockServer.expect(MockRestRequestMatchers.requestTo(BASE_URL))
62
63  .andRespond(MockRestResponseCreators.withSuccess(jsonResponse,
64  MediaType.APPLICATION_JSON));

65          Movie movie = new Movie();
66          movie.setTitle("New Movie");
67
68          Movie result = sut.save(movie);
69          assertEquals("New Movie", result.getTitle());
70
71          mockServer.verify();
72      }
73
74
75  }
76
```

Código 7: Modificación sobre pruebas unitarias películas para conexión con *moviecards-service*

5. Creación de nuevo atributo en Moviecards-service

En esta sección, se ha modificado el microservicio *moviecards-service* para añadir un nuevo atributo *deadDate*. Para ello, se ha accedido a dicho proyecto y se ha modificado el archivo *src/main/java/com/lauracercas/moviecards/model/Actor.java*. En él, las únicas modificaciones necesarias realizadas han sido la adición del atributo y los métodos get y set, puesto que el resto de las operaciones las maneja SpringBoot por defecto. El **Código 8** muestra el resultado final, con las adiciones sobre el archivo original marcadas en verde (Líneas 28-29 y Líneas 70-79).

```
1  /**
2   * Autora: Laura Cercas Ramos
3   * Proyecto: TFM Integración Continua con GitHub Actions
4   * Fecha: 04/06/2024
5   * Cambios: José R. Hilera (2024) para eliminar la parte cliente
    de la aplicación original
6   */
7
8  package com.lauracercas.moviecards.model;
9
10 import org.springframework.format.annotation.DateTimeFormat;
11 import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
12 import javax.persistence.*;
13 import java.util.Date;
14 import java.util.List;
15 import java.util.Objects;
16
17 @Entity
18 public class Actor {
19     @Id
20     @GeneratedValue(strategy = GenerationType.AUTO)
21     private Integer id;
22
23     private String name;
24
25     @DateTimeFormat(pattern = "yyyy-MM-dd")
26     private Date birthDate;
27
28     @DateTimeFormat(pattern = "yyyy-MM-dd") // Añadido por Jesús
    Ángel del Hoyo
29     private Date deadDate;
30
31     private String country;
32
33     @ManyToMany(mappedBy = "actors")
34
35     @JsonIgnoreProperties("actors") // Añadido
36     private List<Movie> movies;
37
38     public Actor() {
```

```
39      }
40
41      public Actor(Integer id, String name) {
42          this.id = id;
43          this.name = name;
44      }
45
46      public Integer getId() {
47          return id;
48      }
49
50      public void setId(Integer id) {
51          this.id = id;
52      }
53
54      public String getName() {
55          return name;
56      }
57
58      public void setName(String name) {
59          this.name = name;
60      }
61
62      public Date getBirthDate() {
63          return birthDate;
64      }
65
66      public void setBirthDate(Date birthDate) {
67          this.birthDate = birthDate;
68      }
69
70      // Añadido por Jesús Ángel del Hoyo
71      public Date getDeadDate() {
72          return deadDate;
73      }
74
75      public void setDeadDate(Date deadDate) {
76          this.deadDate = deadDate;
77      }
78
79      // Fin de la parte añadida por Jesús Ángel del Hoyo
80
81      public String getCountry() {
82          return country;
83      }
84
85      public void setCountry(String country) {
86          this.country = country;
87      }
```

```
88
89      public List<Movie> getMovies() {
90          return movies;
91      }
92
93      public void setMovies(List<Movie> movies) {
94          this.movies = movies;
95      }
96
97      @Override
98      public boolean equals(Object o) {
99          if (this == o)
100             return true;
101         if (o == null || getClass() != o.getClass())
102             return false;
103         Actor actor = (Actor) o;
104         return Objects.equals(id, actor.id) &&
    Objects.equals(name, actor.name)
105                 && Objects.equals(birthDate, actor.birthDate) &&
    Objects.equals(country, actor.country);
106     }
107
108     @Override
109     public int hashCode() {
110         return Objects.hash(id, name, birthDate, country);
111     }
112 }
```

Código 8: Clase Actor de moviecards-service tras añadir deadDate

Una vez realizadas las modificaciones, se pueden comprobar mediante Postman los cambios efectuados. La **Ilustración 21** e **Ilustración 22** muestran el proceso de creación de un nuevo actor y el resultado de obtener la lista de todos los actores, seleccionados por ser las operaciones más representativas.

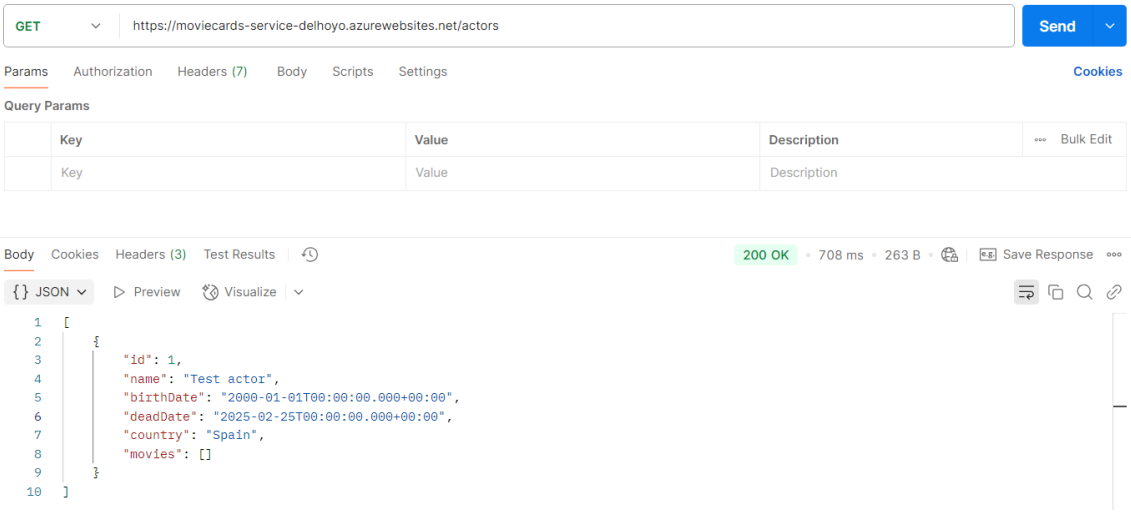

Ilustración 21: Creación de un actor de prueba con deadDate en moviecards-service

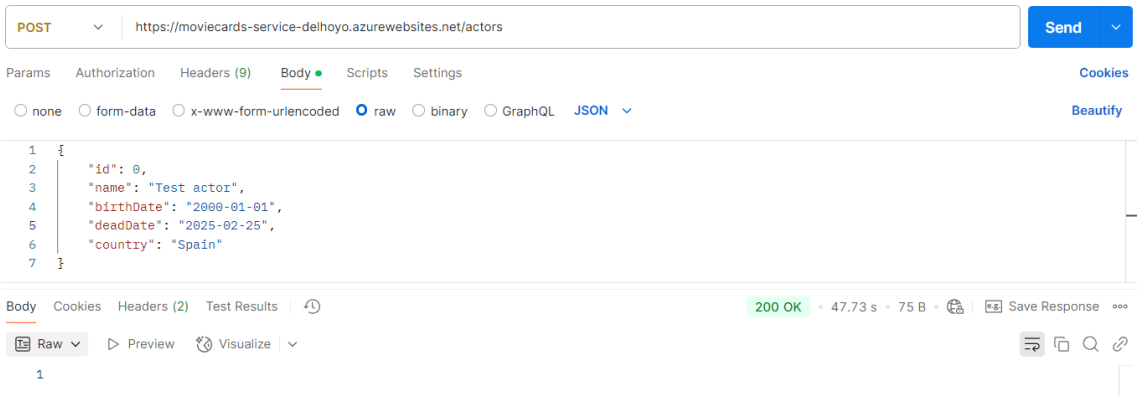

Ilustración 22: Consulta de actores con deadDate en moviecards-service

6. Última versión de Moviecards

Por último, este último apartado de la práctica solicitaba la modificación de *moviecards* para que, en consonancia con la modificación anteriormente hecha, mostrase el atributo *deadDate* en la interfaz. Además, se solicitaba la creación de un entorno de pre-producción, un tercer sprint que englobara todo el trabajo realizado, la modificación de las pruebas unitarias, de integración y funcionales para considerar este nuevo atributo y, por último, solucionar los problemas de calidad del código para cumplir con las garantías de calidad que SonarQube establece.

Primero, se muestran las tareas que se han creado para el sprint *Actualización DeadDate* (**Ilustración 23**), el estado del tablero tras finalizar el sprint (**Ilustración 24**) y, por último, el sprint finalizado (**Ilustración 25**).

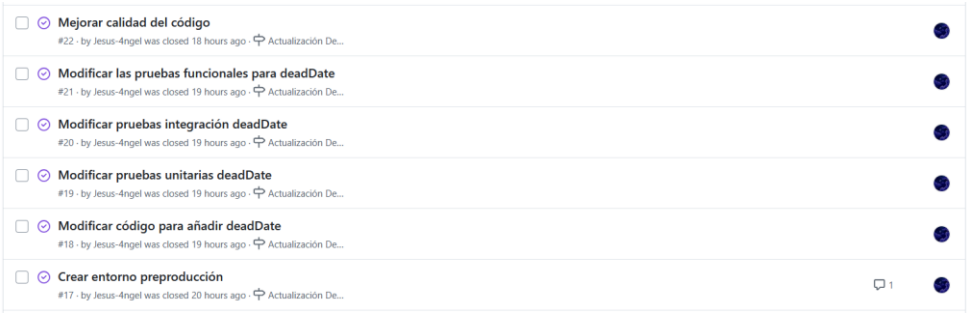

Ilustración 23: Tareas creadas para el tercer sprint

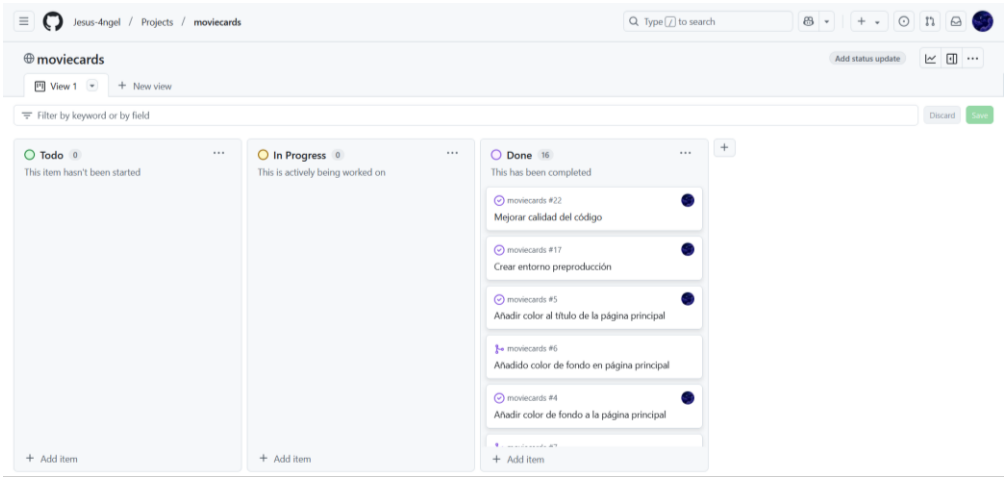

Ilustración 24: Estado del tablero tras terminar el tercer sprint

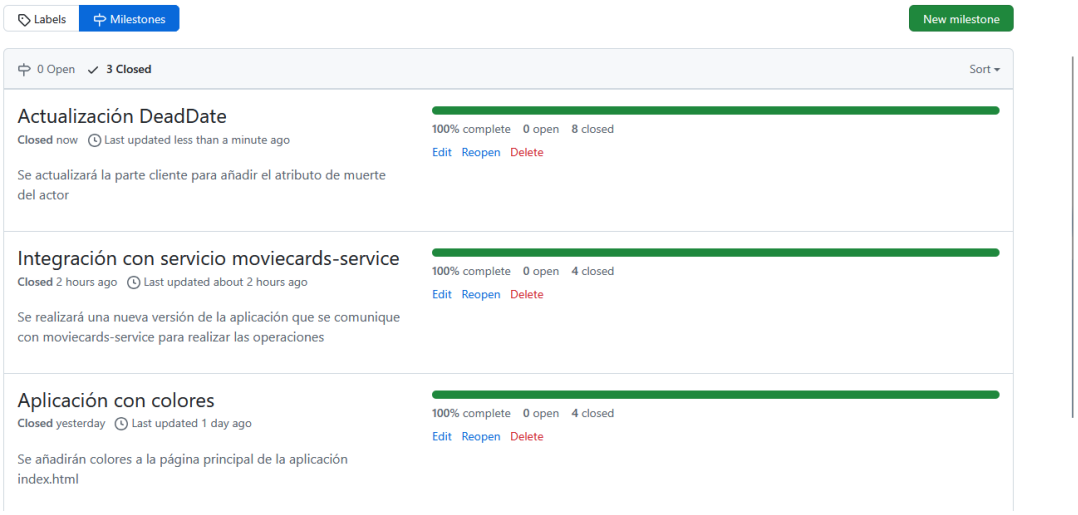

Ilustración 25: Tercer sprint finalizado

Posteriormente, el **Código 9** muestra las instrucciones finales que se utilizan para ejecutar los diferentes pasos en el workflow. Para crear el servidor de pre-producción, se han seguido las mismas pautas que para crear un servidor de producción, con la diferencia de que el despliegue en este entorno se realiza de manera automática (sin necesidad de intervención de un contribuidor). Además, ha sido necesario modificar las dependencias entre los diferentes pasos (stage se ejecuta tras qa, pero antes que deploy), y, por último, para cumplir con las garantías de que el despliegue no se realizará a menos que pase las quality gates de SonarQube, se ha modificado el parámetro *continue-on-error* con un valor de *false*, lo que impedirá la ejecución del resto de pasos del workflow si esta falla.

El entorno de pre-producción es accesible desde el siguiente enlace: https://moviecards-pre-delhoyo.azurewebsites.net

```yaml
name: CI

on: push

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Descargar repositorio
        uses: actions/checkout@v2
      - name: Instalar JDK 11
        uses: actions/setup-java@v2
        with:
          java-version: "11"
          distribution: "adopt"
      - name: Construir con Maven
        run: mvn clean package -DskipTests
      - name: Guardar paquete generado para el trabajo de despliegue
        uses: actions/upload-artifact@v4
        with:
          name: moviecards-java
          path: "${{ github.workspace }}/target/*.jar"
```

```
23
24  test:
25    needs: build
26    runs-on: ubuntu-latest
27    steps:
28      - name: Descargar repositorio
29        uses: actions/checkout@v2
30      - name: Instalar JDK 11
31        uses: actions/setup-java@v2
32        with:
33          java-version: "11"
34          distribution: "adopt"
35
36      - name: Instalar Chrome y ChromeDriver para pruebas end to end
37        run: |
38          wget https://dl.google.com/linux/direct/google-chrome-
  stable_current_amd64.deb
39
40          sudo dpkg -i google-chrome-stable_current_amd64.deb
41
42          sudo apt --fix-broken install -y
43
44          CHROMEDRIVER_VERSION=$(curl -sS
  https://chromedriver.storage.googleapis.com/LATEST_RELEASE)
45
46          curl -L -o chromedriver.zip
  https://chromedriver.storage.googleapis.com/$CHROMEDRIVER_VERSION/chromedr
  iver_linux64.zip
47
48          unzip chromedriver.zip
49
50          chmod +x chromedriver
51
52          sudo mv chromedriver /usr/local/bin/
53
54      - name: Ejecutar la aplicación para pruebas end to end
55        run: mvn spring-boot:run & sleep 60
56
57      - name: Ejecutar las pruebas unitarias, de integración y end to end
58        run: mvn clean verify
59
60  qa:
61    needs: test
62    runs-on: self-hosted
63    continue-on-error: false
64    steps:
65      - name: Descargar repositorio
66        uses: actions/checkout@v2
67
68      - name: Instalar JDK 11
69        uses: actions/setup-java@v2
70        with:
71          java-version: "11"
72          distribution: "adopt"
```

```
73
74        - name: Construir con Maven
75          run: mvn clean package -DskipTests
76
77        - name: Revisar la calidad con Sonarqube
78          run: |
79            mvn sonar:sonar -Dsonar.host.url=http://localhost:9000 -
   Dsonar.qualitygate.wait=true -Dsonar.login=admin -Dsonar.password=admin
80
81    stage:
82      runs-on: ubuntu-latest
83      needs: qa
84      environment:
85        name: 'Stage'
86        url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}
87
88      steps:
89        - name: Download artifact from build job
90          uses: actions/download-artifact@v4
91          with:
92            name: moviecards-java
93
94        - name: Deploy to Azure Web App
95          id: deploy-to-webapp
96          uses: azure/webapps-deploy@v3
97          with:
98            app-name: 'moviecards-pre-delhoyo'
99            slot-name: 'Production'
100           package: '*.jar'
101           publish-profile: ${{
   secrets.AZUREAPPSERVICE_PUBLISHPROFILE_48483DB51B6C4E7E8E452E6B81F0EA61 }}
102
103   deploy:
104     runs-on: ubuntu-latest
105     needs: stage
106     if: github.ref=='refs/heads/master'
107     environment:
108       name: 'Production'
109       url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}
110
111     steps:
112       - name: Aprobación manual
113         uses: trstringer/manual-approval@v1
114         with:
115           secret: ${{ secrets.TOKEN }}
116           approvers: Jesus-4ngel
117       - name: Download artifact from build job
118         uses: actions/download-artifact@v4
119         with:
120           name: moviecards-java
121
122       - name: Deploy to Azure Web App
123         id: deploy-to-webapp
124         uses: azure/webapps-deploy@v3
```

```
125         with:
126           app-name: 'moviecards-delhoyo'
127           slot-name: 'Production'
128           package: '*.jar'
129           publish-profile: ${{
    secrets.AZUREAPPSERVICE_PUBLISHPROFILE_049D6A1D80B74B53B50362007D7B79A9 }}
```

Código 9: Código final con las instrucciones del workflow con pre-producción

Una vez realizados estos cambios, se procede a la modificación del código para aceptar la nueva fecha de muerte de los actores. Realmente, las modificaciones no son muy exigentes, puesto que SpringBoot manejará la mayoría de los aspectos por nosotros. Como consecuencia, se han modificado los siguientes tres archivos: el modelo del actor, el formulario para crear o editar nuevos actores y el listado que muestra dicha información.

Modificaciones realizadas en el modelo

En este caso, de la misma forma que se hizo en *moviecards-service*, se ha añadido el atributo *deadDate* en la clase que modela los actores, que se puede encontrar en la siguiente ruta: *src/main/java/com/lauracercas/moviecards/model/Actor.java.* El **Código 10** muestra el resultado final.

```
 1 /**
 2  * Autor: Laura Cercas Ramos
 3  * Proyecto: TFM Integración Continua con GitHub Actions
 4  * Fecha: 04/06/2024
 5  */
 6 @Entity
 7 public class Actor {
 8     @Id
 9     @GeneratedValue(strategy = GenerationType.AUTO)
10     private Integer id;
11
12     private String name;
13
14     @DateTimeFormat(pattern = "yyyy-MM-dd")
15     private Date birthDate;
16
17     @DateTimeFormat(pattern = "yyyy-MM-dd") // Añadido por Jesús
   Ángel del Hoyo
18     private Date deadDate;
19
20     private String country;
21
22     @ManyToMany(mappedBy = "actors")
23     private List<Movie> movies;
24
25     public Actor() {
26     }
27
28     public Actor(Integer id, String name) {
```

```
29          this.id = id;
30          this.name = name;
31      }
32
33      public Integer getId() {
34          return id;
35      }
36
37      public void setId(Integer id) {
38          this.id = id;
39      }
40
41      public String getName() {
42          return name;
43      }
44
45      public void setName(String name) {
46          this.name = name;
47      }
48
49      public Date getBirthDate() {
50          return birthDate;
51      }
52
53      public void setBirthDate(Date birthDate) {
54          this.birthDate = birthDate;
55      }
56
57      // Añadido por Jesús Ángel del Hoyo
58      public Date getDeadDate() {
59          return deadDate;
60      }
61
62      public void setDeadDate(Date deadDate) {
63          this.deadDate = deadDate;
64      }
65
66      // Fin de la parte añadida por Jesús Ángel del Hoyo
67
68      public String getCountry() {
69          return country;
70      }
71
72      public void setCountry(String country) {
73          this.country = country;
74      }
75
76      public List<Movie> getMovies() {
77          return movies;
```

```
78      }
79
80      public void setMovies(List<Movie> movies) {
81          this.movies = movies;
82      }
83
84      @Override
85      public boolean equals(Object o) {
86          if (this == o) return true;
87          if (o == null || getClass() != o.getClass()) return false;
88          Actor actor = (Actor) o;
89          return Objects.equals(id, actor.id) && Objects.equals(name,
    actor.name) && Objects.equals(birthDate, actor.birthDate) &&
    Objects.equals(country, actor.country);
90      }
91
92      @Override
93      public int hashCode() {
94          return Objects.hash(id, name, birthDate, country);
95      }
96  }
```

Código 10: Modelo de actor modificado en *moviecards-service*

Modificaciones realizadas en el formulario de actores

En el formulario que solicita los datos para añadir o editar un actor, se ha incluido un nuevo campo de la misma forma que su fecha de nacimiento, como muestra el fragmento dispuesto en el **Código 11**. Este puede encontrarse en el archivo *src/main/resources/templates/actors/form.html*.

```
1  <div class="mb-3">
2      <label for="deadDate" class="form-label">Fecha muerte</label>
3      <input
4              type="date"
5              class="form-control"
6              th:field="*{deadDate}"
7              id="deadDate"
8              name="deadDate"
9              placeholder="Escriba la fecha de muerte"
10             required="required"
11     />
12 </div>
```

Código 11: Nuevo campo *deadDate* en el formulario de actores

Modificaciones realizadas en el listado de actores

Por último, la vista encargada de mostrar el listado de actores también deberá mostrar el campo de fecha de muerte. Para ello, se ha modificado el fragmento de código mostrado en el **Código 12**, en el que se han añadido las Líneas 5 y 15 para gestionar la fecha de muerte. Esta vista puede encontrarse en el archivo *src/main/resources/templates/actors/list.html.*

```
<tr>
    <th scope="col">Identificador</th>
    <th scope="col">Nombre</th>
    <th scope="col">Fecha Nacimiento</th>
    <th scope="col">Fecha Muerte</th>
    <th scope="col">Pais</th>
    <th scope="col">Editar</th>
</tr>
</thead>
<tbody>
<tr th:each="actor : ${actors}">
    <td th:text="${actor.id}"></td>
    <td th:text="${actor.name}"></td>
    <td th:text="${#dates.format(actor.birthDate, 'dd-MM-yyyy')}"></td>
    <td th:text="${#dates.format(actor.deadDate, 'dd-MM-yyyy')}"></td>
    <td th:text="${actor.country}"></td>
    <td>
        <a th:href="@{'/editActor/' + ${actor.id}}" class="btn btn-primary">Editar</a>
    </td>
</tr>
```

Código 12: Código final que muestra la fecha de muerte en el listado de actores

Una vez realizados todos estos cambios, se modifican las pruebas unitarias, de integración y funcionales para que tengan en consideración este nuevo atributo. Se han modificado tres clases diferentes.

Modificaciones en las pruebas unitarias

Para que se puedan pasar las pruebas unitarias correctamente, se ha añadido el fragmento mostrado en el ***Código 13*** en la clase *src/test/java/com/lauracercas/moviecards/unittest/model/ActorTest.java*. Esto probará el funcionamiento de este nuevo atributo de la misma forma que se hace con la fecha de nacimiento.

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

Código 13: Prueba unitaria para *deadDate*

Modificaciones en las pruebas de integración

Para que las pruebas de integración consideren todos los atributos correctamente, se ha modificado el fragmento mostrado en el **Código 14** en la clase *src/test/java/com/lauracercas/moviecards/integrationtest/repositories/ActorJPAIT.java*. Más concretamente, se han añadido las Líneas 6 y 24. Esto probará el funcionamiento de este nuevo atributo de la misma forma que se hace con la fecha de nacimiento.

```
1     @Test
2     public void testSaveActor() {
3         Actor actor = new Actor();
4         actor.setName("actor");
5         actor.setBirthDate(new Date());
6         actor.setDeadDate(new Date());
7         actor.setCountry("spain");
8
9         Actor savedActor = actorJPA.save(actor);
10
11        assertNotNull(savedActor.getId());
12
13        Optional<Actor> foundActor =
  actorJPA.findById(savedActor.getId());
14
15        assertTrue(foundActor.isPresent());
16        assertEquals(savedActor, foundActor.get());
17    }
18
19    @Test
20    public void testFindById() {
21        Actor actor = new Actor();
22        actor.setName("actor");
23        actor.setBirthDate(new Date());
24        actor.setDeadDate(new Date());
25        Actor savedActor = actorJPA.save(actor);
26
27        Optional<Actor> foundActor =
  actorJPA.findById(savedActor.getId());
28
29        assertTrue(foundActor.isPresent());
30        assertEquals(savedActor, foundActor.get());
31    }
```

Código 14: Pruebas de integración modificadas para considerar *deadDate*

Modificaciones en las pruebas funcionales

Las últimas modificaciones se han realizado sobre las pruebas funcionales para incluir comparaciones con el HTML generado tras añadir *deadDate*. Estas modificaciones se muestran en el **Código 15**, y se han añadido las Líneas 8 y 35, y modificado las líneas a partir de esta última para considerar la “Fecha de muerte”.

```
1 @Test
2 public void testPageLoad() {
3     driver.get("http://localhost:8089/actors/new");
4     assertEquals("FichasPeliculasApp | Aplicación de gestión de fichas de
  películas", driver.getTitle());
5
6     assertTrue(driver.findElement(By.id("name")).isDisplayed());
7     assertTrue(driver.findElement(By.id("birthDate")).isDisplayed());
8     assertTrue(driver.findElement(By.id("deadDate")).isDisplayed());
```

```
 9      assertTrue(driver.findElement(By.id("country")).isDisplayed());
10
11  }
12
13  @Test
14  public void testNewActorTitle() {
15      driver.get("http://localhost:8089/actors/new");
16      WebElement title = driver.findElement(By.className("title"));
17      assertEquals(NEW_ACTOR_TITLE, title.getText());
18  }
19
20  @Test
21  public void testListActors() {
22      driver.get("http://localhost:8089/actors");
23      WebElement title = driver.findElement(By.className("card-header"));
24      assertEquals("Listado Actores", title.getText());
25
26      WebElement table = driver.findElement(By.className("table-hover"));
27
28      WebElement thead = table.findElement(By.tagName("thead"));
29      assertTrue(thead.isDisplayed());
30
31      WebElement headerRow = thead.findElement(By.tagName("tr"));
32      assertEquals("Identificador",
    headerRow.findElements(By.tagName("th")).get(0).getText());
33      assertEquals("Nombre",
    headerRow.findElements(By.tagName("th")).get(1).getText());
34      assertEquals("Fecha Nacimiento",
    headerRow.findElements(By.tagName("th")).get(2).getText());
35      assertEquals("Fecha Muerte",
    headerRow.findElements(By.tagName("th")).get(3).getText());
36      assertEquals("Pais",
    headerRow.findElements(By.tagName("th")).get(4).getText());
37      assertEquals("Editar",
    headerRow.findElements(By.tagName("th")).get(5).getText());
38
39  }
```

Código 15: Pruebas funcionales modificadas para considerar *deadDate*

Finalmente, se han modificado los quality gates de SonarQube para que no acepte más de cinco problemas críticos y, en consecuencia, se han corregido en el código fuente. La **Ilustración 26** muestra la configuración de la quality gate originalmente creada y, posteriormente, se explican las modificaciones realizadas sobre el programa para solventar los fallos críticos identificados en la **Ilustración 27**.

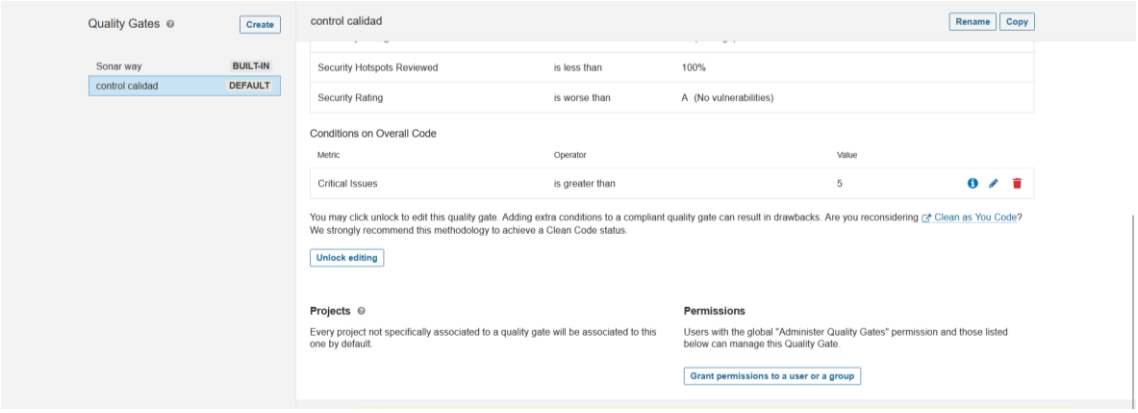

Ilustración 26: Modificación de la quality gate de SonarQube para no aceptar más de 5 errores críticos

Modificaciones realizadas para solucionar los problemas críticos

Para solucionar los problemas identificados en la **Ilustración 27**, se han modificado los siguientes dos archivos:

1. *src/main/java/com/lauracercas/moviecards/controller/ActorController.java*
2. *src/main/java/com/lauracercas/moviecards/controller/MovieController.java*

Los problemas eran tan simples que únicamente consistían en que se repetían varios Strings literales en diferentes secciones de los archivos. Para solucionarlo, se han creado tres constantes al inicio de ambas clases que capturan dichos literales, y se han sustituido los literales en el código fuente por las constantes.

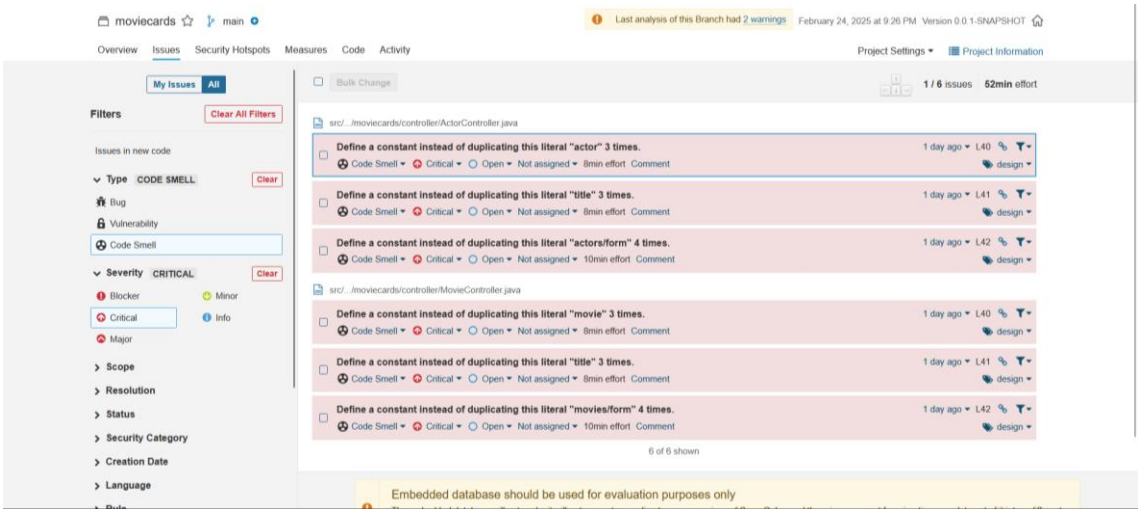

Ilustración 27: Errores críticos detectados por SonarQube

Como consecuencia, SonarQube no detecta errores que incumplan la quality gate en uso, como se muestra en la **Ilustración 28** e **Ilustración 29**, y el workflow puede finalizar correctamente (**Ilustración 30**).

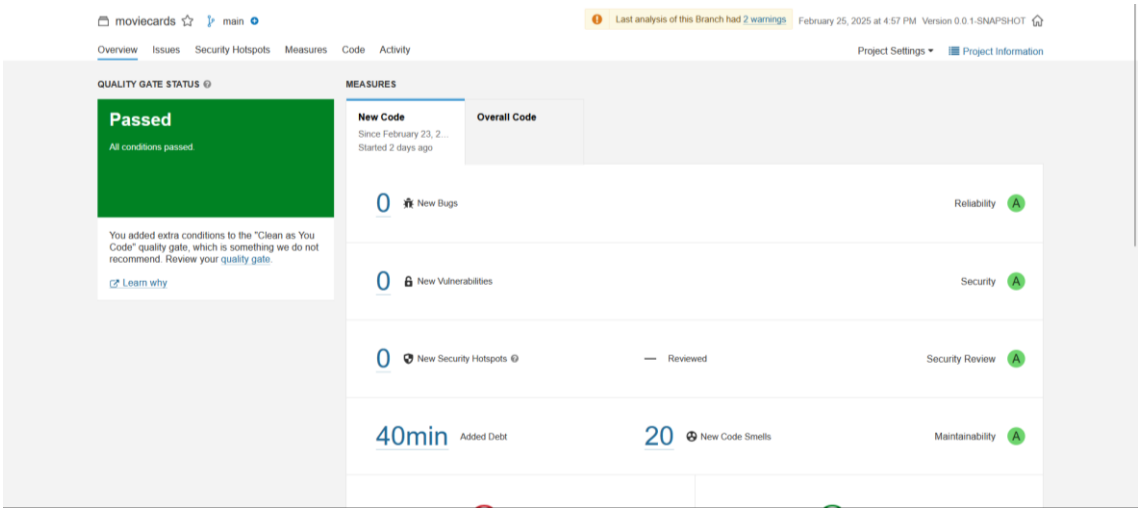

Ilustración 28: SonarQube identifica que todas las condiciones se han cumplido de forma exitosa

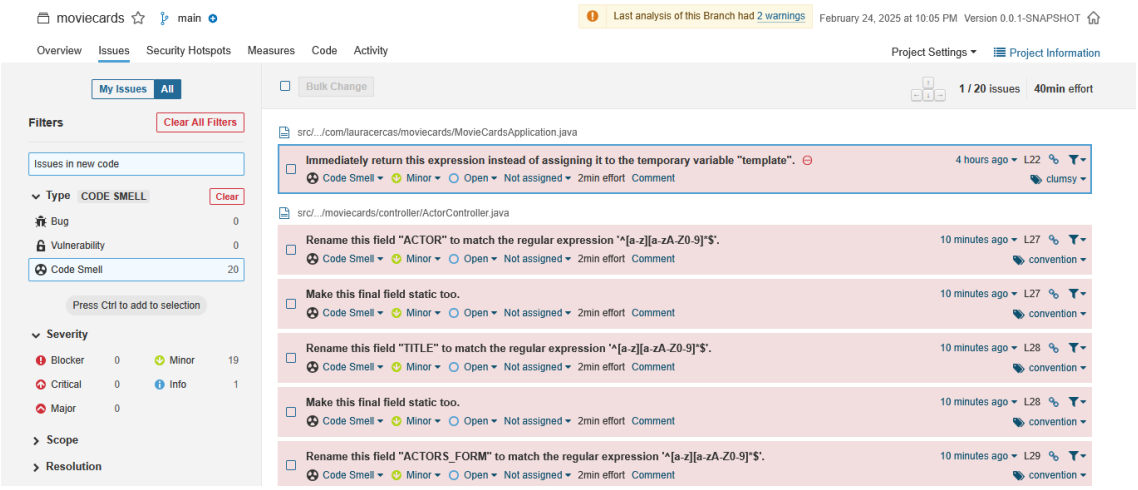

Ilustración 29: SonarQube no ha identificado errores críticos

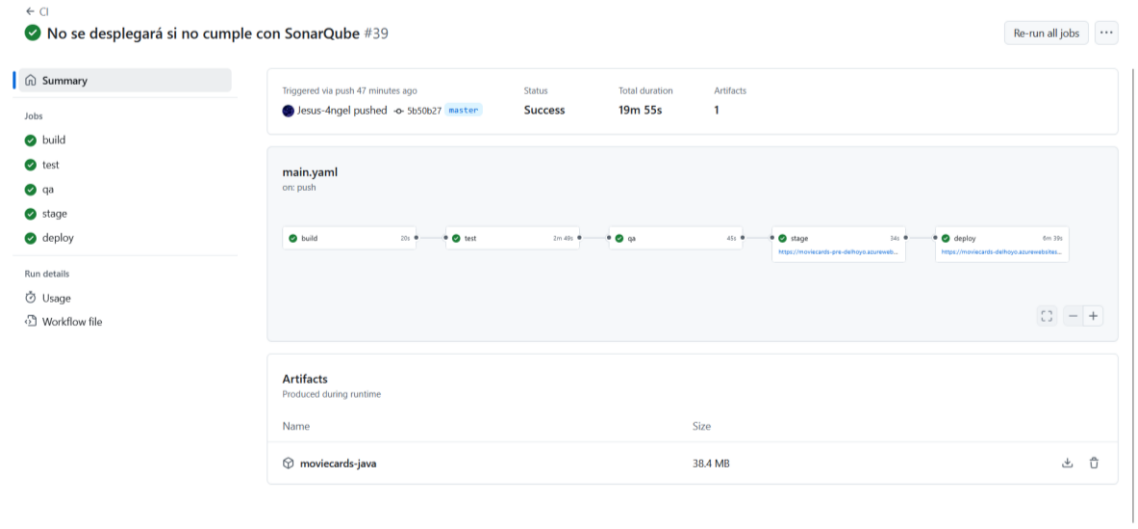

Ilustración 30: Se han ejecutado correctamente todos los pasos del workflow final

Para finalizar con esta sección del documento, la **Ilustración 31** muestra la aplicación en funcionamiento en el entorno de pre-producción (mediante Postman), y la **Ilustración 32** muestra el mismo resultado en el entorno de producción. Además, se muestra el

listado de actores en la etapa de producción con un actor de prueba (**Ilustración 33**), así como el nuevo formulario para la creación de actores (**Ilustración 34**). Además, el funcionamiento del sistema se muestra de forma exhaustiva en el vídeo adjunto a la entrega.

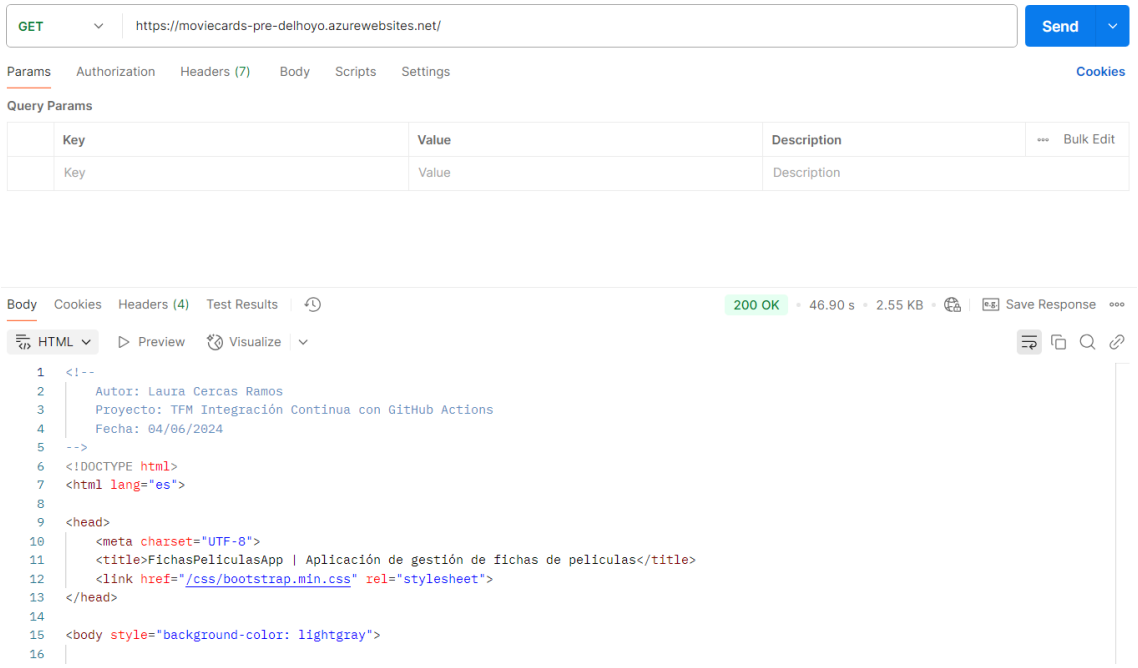

Ilustración 31: Aplicación final en funcionamiento en entorno de pre-producción

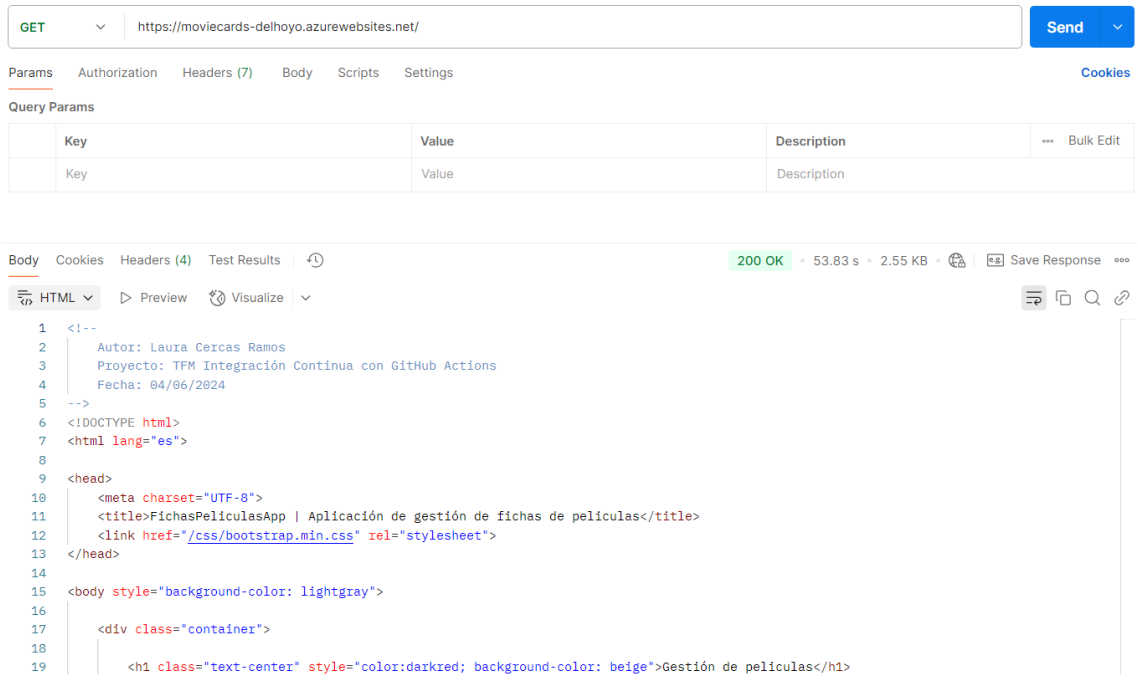

Ilustración 32: Aplicación final en funcionamiento en entorno de producción

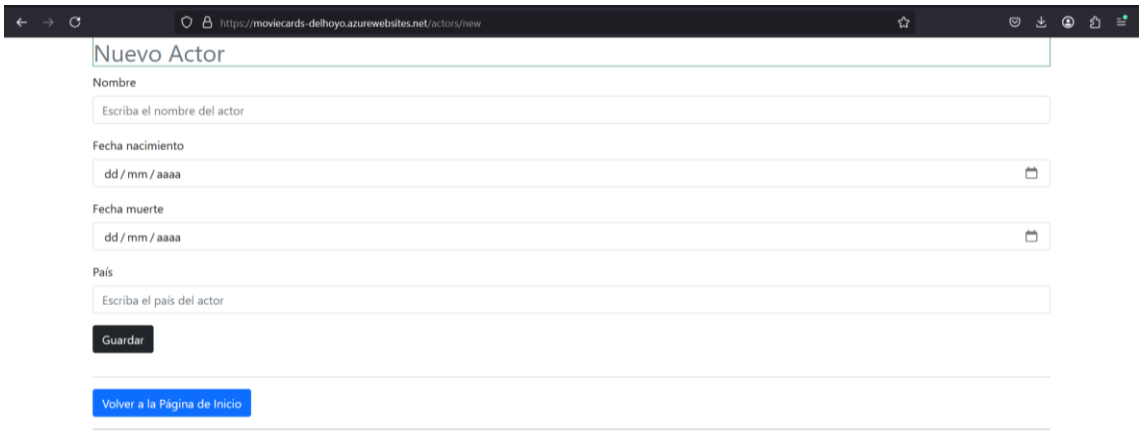

Ilustración 33: Formulario de creación de actor con deadDate en entorno de producción

Ilustración 34: Listado de actores con deadDate en entorno de producción

7. Conclusiones

La realización de esta práctica ha permitido al alumno aplicar de manera efectiva los conocimientos adquiridos en la asignatura, llevando a cabo el despliegue e integración de servicios en Azure, así como la modificación y mejora de una aplicación existente. A lo largo del desarrollo del proyecto, se han cumplido con éxito todos los objetivos propuestos, destacando la correcta implementación de la comunicación entre *moviecards* y *moviecards-service*, la incorporación de un nuevo atributo en la clase de actores y la adaptación de la aplicación para reflejar dichos cambios. Además, se han realizado las pruebas necesarias para garantizar la calidad del código y su correcto funcionamiento en un entorno de pre-producción, asegurando el cumplimiento de las métricas establecidas por SonarQube. En conclusión, este trabajo no solo ha reforzado la comprensión teórica de los conceptos estudiados, sino que también ha proporcionado una valiosa experiencia práctica en el despliegue, integración y mejora de aplicaciones en entornos en la nube.