Práctica 1. Integración continua

Enunciado

Se desean implementar ciertos controles de calidad de una aplicación que gestiona una librería online. Esta librería ofrece un interfaz web y una API REST para la gestión de libros, con sus correspondientes pruebas unitarias, de API REST y Selenium. Se proporciona el código de dicha aplicación, que podrá obtenerse del aula virtual.

El control de calidad se realizará mediante los siguientes pasos ejecutados en este orden:

1. Implantación del modelo de desarrollo Git Flow para el proyecto. Básicamente crear las ramas necesarias.
2. Definición de workflows con GitHub actions que automaticen la ejecución de pruebas tal como se describe en la sección correspondiente. Estos workflows deben funcionar correctamente antes de pasar al paso 3.
3. Desarrollo de una funcionalidad nueva, que se describe más abajo en su propia sección.

Preparación del repositorio inicial

Cada grupo (de una o dos personas) deberá crear un repositorio de código en GitHub que:

- Debe ser privado
- Debe nombrarse cómo **mca-4.3-usuario-urjc-2021**
    - Ejemplo: mca-4.3-p.perez-2021
- Debe incluir a los profesores Francisco Gortázar (Usuario en Github: **gortazar**) y Michel Maes (Usuario en Github: **Maes95**) invitándoles al repositorio

Consideraciones:

- Los repositorios privados están limitados en cuanto a los recursos que pueden consumir, concretamente:
    - Permite un máximo de 2.000 minutos de ejecución al mes
    - Permite un almacenamiento de artefactos y logs de 500 Mb

  Al realizar múltiples jobs que suban artefactos, nos podemos encontrar con que agotemos este almacenamiento, por ello se proponen algunas soluciones:
    - Utilizar la propiedad **retention-days**: esta propiedad nos permite definir durante cuánto tiempo existirá el artefacto (mínimo 1 día, máximo 90)

```
- name: 'Upload Artifact'
 uses: actions/upload-artifact@v2
 with:
   name: target
   path: target/*.jar
```

```
          retention-days: 1
```

- **Borrar los artefactos**: Una vez completado el workflow, se pueden borrar los artefactos generados (https://docs.github.com/es/actions/managing-workflow-runs/removing-workflow-artifacts)

Implantación del modelo de desarrollo Git Flow

Se crearán todas las ramas necesarias, tal como indica el modelo de desarrollo Git Flow. Algunas consideraciones:

- Cuando haya que mezclar cambios con otra rama se hará mediante pull requests.
- Cada vez que se abra una rama de release para preparar una nueva versión se debe hacer lo siguiente:
  - En la rama de release, eliminar el sufijo "-SNAPSHOT" de la versión de la aplicación que se indica en el pom.xml.
  - En la rama de integración (rama develop en git flow), aumentar el *minor version* de la versión de la aplicación en el pom.xml.

Definición de workflows

Se deben ejecutar diferentes workflows dependiendo de las acciones realizadas en el repositorio git. Para ello en esta fase se incluirán los workflows necesarios como se indica:

- Cada vez que se termine una feature (ramas feature/*), y antes de integrarse en la rama correspondiente (rama develop):
  - Se ejecutarán las pruebas unitarias y de API REST.
- Por cada commit en la rama de integración (rama develop):
  - Se ejecutarán todas las pruebas (unitarias, API REST, Selenium)
- Cada noche, en la rama de integración:
  - Se ejecutarán todas las pruebas (unitarias, de API REST y Selenium) contra la rama de integración (rama develop).
- Cuando se esté preparando una release, en la rama release/**:
  - Se ejecutarán las pruebas unitarias y de API REST por cada commit en la rama.
- Al integrar con la rama de producción:
  - Se ejecutarán todas las pruebas

Consideraciones adicionales:

- De cara a facilitar la identificación de los errores cuando fallan las pruebas, cuando haya que ejecutar varios tipos de prueba, **cada tipo de prueba** (unitaria, API REST, Selenium) **debe ejecutarse en su propio step**. Para ello, se pueden filtrar los test por el nombre del paquete usando el parámetro de Maven "-Dtest":

```
mvn -B '-Dtest=es.urjc.code.daw.library.e2e.selenium.*Test' test
```

- Para extraer la versión de la aplicación por línea de comandos se puede utilizar el siguiente comando que guarda la versión en una variable de entorno:
    - `VERSION=$(mvn -q help:evaluate -Dexpression=project.version -DforceStdout)`

Desarrollo de una funcionalidad nueva

La aplicación actualmente tiene un bug. Cuando se edita un libro y se pulsa el botón cancelar, la aplicación redirige a una url que no existe (error 404). Se desea solucionar este error. El error se encuentra en la plantilla html Mustache que se encuentra en el fichero src/main/resources/templates/editBookPage.html. Esta plantilla contiene un botón cancelar, cuyo evento onclick redirige a '/book/{{book.id}}', cuando en realidad debería ser '/books/{{book.id}}'. Se debe seguir el modelo de desarrollo de git flow para solventar este error, y acabar mezclando una nueva versión con el error corregido en la rama de producción.

Para implementar esta funcionalidad, se seguirán los siguientes pasos:

1. Solución del problema y apertura del pull request. En este punto el PR debe indicar que los tests pasan.
2. Mezcla del PR en la rama develop. Los tests pasan.
3. Nueva rama de release con el error corregido.
4. PR contra producción. Los tests pasan.
5. Aceptación del PR contra producción (obviamos el PR contra develop porque no hay cambios adicionales en la rama release). Todos los tests pasan.

Formato de entrega

Toda la práctica se debe desarrollar en el repositorio GitHub correspondiente que se haya creado siguiendo el procedimiento explicado al principio. Concretamente, este repositorio debe tener las ramas exigidas por el modelo de desarrollo git flow, los cambios realizados deben hacerse siguiendo este modelo, y los profesores evaluarán que el modelo se haya seguido concienzudamente. Por ello, no se deberá borrar ninguna rama.

Adicionalmente la práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del repositorio GitHub. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf-z.gonzalez.zip)
- En el fichero pom.xml se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos):

```
<groupId>es.codeurjc.mca</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Además del código fuente, se deberá elaborar una memoria explicativa del mismo en formato PDF.

- Deberá guardarse en la carpeta raíz del proyecto.
- La portada deberá contener el nombre de la práctica, la asignatura, el curso académico y el nombre del alumno o alumnos.
- Se deberá especificar la URL del repositorio GitHub utilizado.
- Se deberá describir el funcionamiento de los workflows implementados: cuándo se ejecutan (en base a qué eventos en qué ramas) y qué hacen
- La memoria deberá tener una longitud de 3-5 páginas (incluyendo la página de la portada).