UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN DESARROLLO ÁGIL DE SOFTWARE PARA LA WEB

Trabajo Fin de Máster

PROYECTO DE INTEGRACIÓN CONTINUA CON GITHUB ACTIONS

Christian Xavier Dopico Morán

2022

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN

DESARROLLO ÁGIL DE SOFTWARE PARA LA WEB

Trabajo Fin de Máster

"PROYECTO DE INTEGRACIÓN CONTINUA CON GITHUB ACTIONS"

Autor: **Christian Xavier Dopico Morán**

Director: **José Ramón Hilera González**

Tribunal:

Presidente: .....

Vocal 1º: ..... Vocal 2º: .....

Calificación: .....

Fecha: ..... de ..... de .....

ÍNDICE RESUMIDO

ÍNDICE DETALLADO

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

La integración continua es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas.

La integración continua se refiere en su mayoría a la fase de creación o integración del proceso de publicación de software y conlleva un componente de automatización (p. ej., CI o servicio de versiones) y un componente cultural (p. ej., aprender a integrar con frecuencia). Los objetivos clave de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones de software.

Anteriormente, era común que los desarrolladores de un equipo trabajasen aislados durante un largo periodo de tiempo y solo intentasen combinar los cambios en la versión maestra una vez que habían completado el trabajo. Como consecuencia, la combinación de los cambios en el código resultaba difícil y ardua, además de dar lugar a la acumulación de errores durante mucho tiempo que no se corregían. Estos factores hacían que resultase más difícil proporcionar las actualizaciones a los clientes con rapidez.

Con la integración continua, los desarrolladores envían los cambios de forma periódica a un repositorio compartido con un sistema de control de versiones como Git. Antes de cada envío, los desarrolladores pueden elegir ejecutar pruebas de unidad local en el código como medida de verificación adicional antes de la integración. Un servicio de integración continua crea y ejecuta automáticamente pruebas de unidad en los nuevos cambios realizados en el código para identificar inmediatamente cualquier error.

Con la entrega continua, se crean, prueban y preparan automáticamente los cambios en el código y se entregan para la fase de producción. La entrega continua amplía la integración continua al implementar todos los cambios en el código en un entorno de pruebas y/o de producción después de la fase de creación.

Finalmente, el despliegue continuo, el despliegue a producción se hace de manera automática una vez se han ejecutado todas las pruebas.

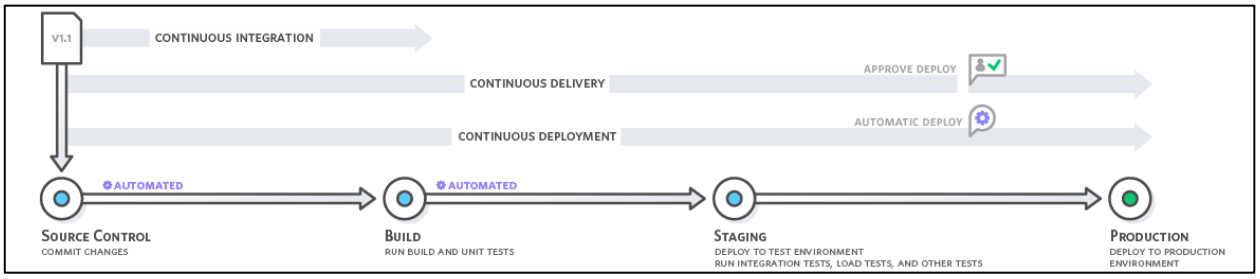


Ilustración 1 – Diagrama de integración continua [1]

2. OBJETIVOS DEL PROYECTO

El principal objetivo es simular la realización de un proyecto de desarrollo de una aplicación web Java aplicando los principios de la Integración Continua y el Desarrollo Ágil.

Como segundo objetivo, se debe realizar el proyecto usando Github Actions como servidor de integración continua.

El proyecto que se lleva a cabo en este trabajo está basado en la práctica realizada en la asignatura "Integración Continua e el Desarrollo Ágil" del máster, y se ha reutilizado el código fuente de la aplicación Java de la práctica. Tano el documento de la práctica como el código fuente están disponibles con licencia CC-BY-SA en https://github.com/josehilera/integracion-continua. [2]

Finalmente, se debe decidir si se recomienda utilizar Github Actions en la materia Integración Continua en el Desarrollo Ágil.

3. TECNOLOGÍA UTILIZADA

3.1. Lenguajes de programación

3.1.1. Java

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet, Java está en todas partes.

Java Runtime Environment (JRE) es lo que se obtiene al descargar el software de Java. JRE está formado por Java Virtual Machine (JVM), clases del núcleo de la plataforma Java y bibliotecas de la plataforma Java de soporte. JRE es la parte de tiempo de ejecución del software de Java, que es todo lo que necesita para ejecutarlo en el explorador web. [3]

3.2. Editor de código

3.2.1. Visual Studio Code

Visual Studio Code es un editor de código fuente ligero pero potente que se ejecuta en su escritorio y está disponible para Windows, macOS y Linux. Viene con soporte incorporado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros lenguajes (como C++, C#, Java, Python, PHP, Go) y tiempos de ejecución (como .NET y Unity). [4]

3.3. Herramientas de construcción

3.3.1. Maven

Apache Maven es una herramienta de comprensión y gestión de proyectos de software. Basado en el concepto de un modelo de objetos de proyecto (POM), Maven puede administrar la construcción, los informes y la documentación de un proyecto desde una pieza central de información. [5]

3.4. Gestión de versiones

3.4.1. Git

Git es un sistema de control de versiones distribuido gratuito y de código abierto diseñado para manejar todo, desde proyectos pequeños hasta proyectos muy grandes, con rapidez y eficiencia.

Git es fácil de aprender y ocupa poco espacio con un rendimiento ultrarrápido. Supera a las herramientas SCM como Subversion, CVS, Perforce y ClearCase con características como sucursales locales económicas, áreas de preparación convenientes y múltiples flujos de trabajo. [6]

3.5. Repositorio

3.5.1. Github

GitHub es una plataforma de alojamiento, propiedad de Microsoft, que ofrece a los desarrolladores la posibilidad de crear repositorios de código y guardarlos en la nube de forma segura, usando un sistema de control de versiones, llamado Git.

Facilita la organización de proyectos y permite la colaboración de varios desarrolladores en tiempo real. Es decir, nos permite centralizar el contenido del repositorio para poder colaborar con los otros miembros de nuestra organización.

GitHub está basada en el sistema de control de versiones distribuida de Git, por lo que se puede contar con sus funciones y herramientas, aunque GitHub ofrece varias opciones adicionales y su interfaz es mucho más fácil de manejar, por lo que no es absolutamente necesario que las personas que lo usan tengan un gran conocimiento técnico. Aquí puedes conocer más sobre su historia. [7]

3.6. Gestión de proyectos ágiles

3.6.1. Github Projects

Un proyecto es una hoja de cálculo personalizable que se integra con tus propuestas y solicitudes de cambios en GitHub. Puedes personalizar el diseño si filtras, clasificas y agrupas tus propuestas y solicitudes de cambios. También puedes personalizar los campos para rastrear metadatos. Los proyectos son flexibles para que tu equipo pueda trabajar de la misma forma que es mejor para ellos.

Tu proyecto se mantiene actualizado automáticamente con la información de GitHub. Cuando cambia una solicitud de cambios o propuesta, tu proyecto refleja dicho cambio. Esta integración también trabaja de ambas formas, así que, cuando cambies la información sobre una solicitud de cambios o propuesta de tu proyecto, esta reflejará la información.[8]

3.7. Servidor de integración continua

3.7.1. Github Actions

GitHub Actions es una plataforma de integración y despliegue continuos (CI/CD que te permite automatizar tu mapa de compilación, pruebas y despliegue. Puedes crear flujos de trabajo y crear y probar cada solicitud de cambios en tu repositorio o desplegar solicitudes de cambios fusionadas a producción.

GitHub Actions va más allá de solo DevOps y te permite ejecutar flujos de trabajo cuando otros eventos suceden en tu repositorio. Por ejemplo, puedes ejecutar un flujo de trabajo para que agregue automáticamente las etiquetas adecuadas cada que alguien cree una propuesta nueva en tu repositorio.

GitHub proporciona máquinas virtuales Linux, Windows y macOS para que ejecutes tus flujos de trabajo o puedes hospedar tus propios ejecutores auto-hospedados en tu propio centro de datos o infraestructura en la nube.[9]

3.8. Herramienta de virtualización

3.8.1. Docker

Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones. Docker le permite separar sus aplicaciones de su infraestructura para que pueda entregar software rápidamente. Con Docker, puede administrar su infraestructura de la misma manera que administra sus aplicaciones. Al aprovechar las metodologías de Docker para enviar, probar e implementar el código rápidamente, puede reducir significativamente la demora entre escribir el código y ejecutarlo en producción.[10]

3.9. Herramienta para pruebas unitarias

3.9.1. JUnit

JUnit es un framework para escribir pruebas repetibles. Es una instancia de la arquitectura xUnit para testz unitarios.[11]

3.10. Herramienta para pruebas funcionales

3.10.1. Selenium

Selenium es un proyecto general para una gama de herramientas y bibliotecas que permiten y admiten la automatización de navegadores web.

Proporciona extensiones para emular la interacción del usuario con los navegadores, un servidor de distribución para escalar la asignación del navegador y la infraestructura para implementaciones de la especificación W3C WebDriver que le permite escribir código intercambiable para todos los principales navegadores web.[12]

3.11. Herramientas de análisis de código fuente

3.11.1. Sonarqube

SonarQube es una herramienta de revisión automática de código para detectar errores, vulnerabilidades y code smells en su código. Puede integrarse con su flujo de trabajo existente para permitir la inspección continua de código en las ramas de su proyecto y pull requests.[13]

3.12. Herramientas de despliegue

3.12.1. Heroku

Heroku es una solución de Plataforma como Servicio (PaaS) basada en la nube para que el cliente solo se preocupe de desarrollar su aplicación mientras Heroku se encarga de la infraestructura que hay detrás.

Para proporcionar este servicio se dispone de unos contenedores virtuales que son los encargados de mantener y ejecutar las aplicaciones. Estos contenedores virtuales son totalmente escalables bajo demanda. Tanto en número como en capacidades.

Una ventaja de elegir Heroku es su capacidad de soportar múltiples lenguajes de programación. Los principales a utilizar son: Node.js, Ruby, Python, Java, PHP, Go, Scala y Clojure. Aunque esta cantidad de lenguajes puede aumentar en el caso de utilizar Heroku Buildpacks, que permiten compilar las aplicaciones en multitud de ellos más.[14]

4. INSTALACIÓN DE INFRAESTRUCTURA NECESARIA

Para simular un entorno real, se tomará en cuenta el siguiente diagrama:

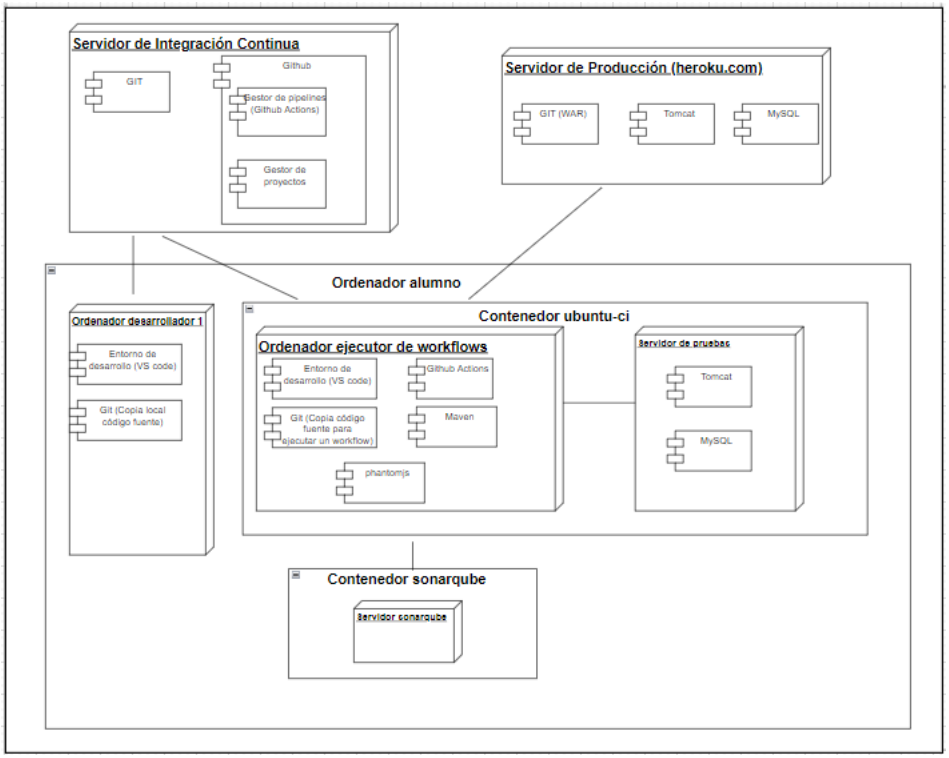


Ilustración 2 – Diagrama de infraestructura. Elaboración Propia

Se puede ver que, el contenedor ubuntu-ci tendrá el ejecutor de pipelines (workflows) y el servidor de prueba.

Un contenedor sonarqube será el encargado de ejecutar las pruebas de calidad de código.

Utilizaremos dos servidores externos: gitlhub.com y heroku.com. El resto estará instalado en nuestro ordenador.

En los siguientes subapartados se explica cómo instalar y la razón de usar cada elemento.

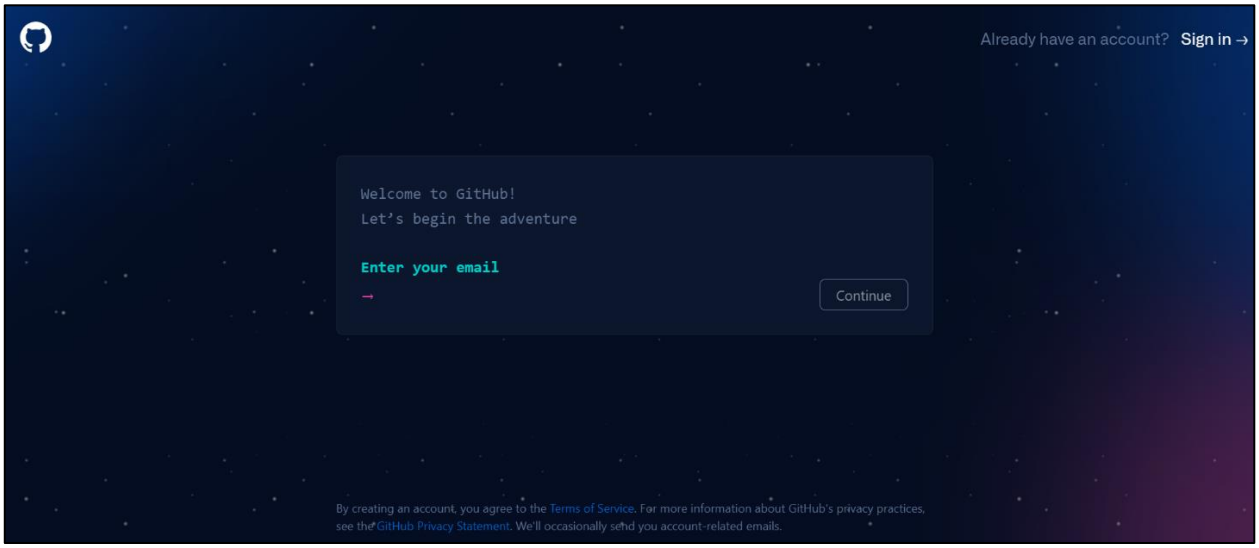
4.1. Crear una cuenta gratuita en github.com

En esta práctica se utilizará github.com, porque es una herramienta que ofrece tres funcionalidades en una única máquina:

- Servidor de integración continua
- Repositorio git para almacenar el código fuente del proyecto
- Tableros Kanban para la gestión ágil de un proyecto, aplicando una metodología como Scrum.

Github ofrece prácticamente toda la funcionalidad en la nube de forma gratuita si un proyecto es público. Y porque de esta forma, el profesor podrá acceder al proyecto y evaluar el trabajo realizado por el alumno.

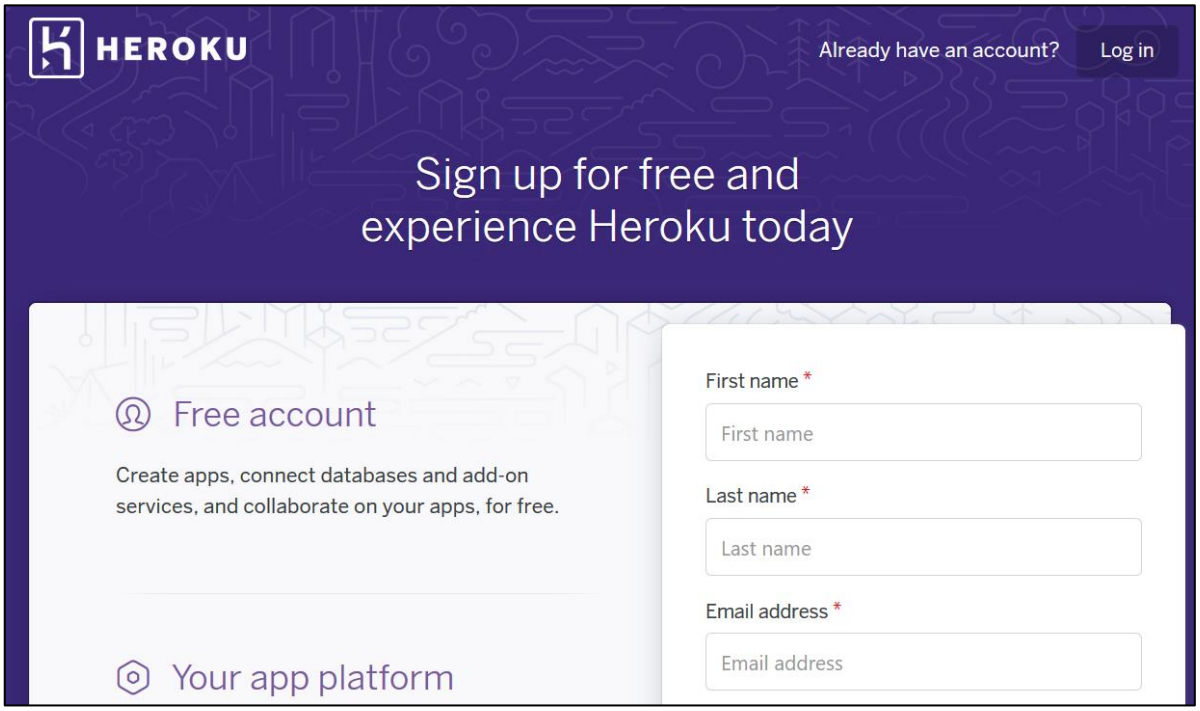
Para crear una cuenta en github.com hay que acceder a https://github.com y elegir Sign up.



4.2. Crear una cuenta gratuita en heroku.com

Para simular un entorno real de producción en el que quede instalada la aplicación web que se desarrolle, se puede utilizar un servicio de hosting en la nube que admita el despliegue mediante comandos desde una consola Linux. En esta práctica se utilizará heroku.com, porque permite desplegar aplicaciones web Java y es gratuito instalar hasta 3 aplicaciones.

Para crear una cuenta, hay que acceder a https://signup.heroku.com/ y registrarse:



4.3. Instalar Docker

Descargar de https://docs.docker.com/get-docker/ para el SO que corresponda. En los ejemplos de esta práctica, se supone que se trabaja en un ordenador Windows, pero funcionaría igual en un ordenador Linux.

Comprobar que está instalado, desde la consola con el comando:

```
C:\> docker --version
Docker version 20.10.6, build 370c289
```

4.4. Instalar Git

Git se necesita para crear y gestionar el repositorio de código del proyecto, así como sus versiones. Es necesario para subir nuevas versiones del código al servidor github.com, que también tiene instalado git, y donde se almacena el repositorio remoto compartido por todos los desarrolladores. Git también se necesita en local para descargar desde github.com las nuevas versiones del código del proyecto que haya modificado otro desarrollador.

Descargar e instalar desde https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Instalaci%C3%B3n-de-Git, eligiendo el sistema que corresponda. Para Windows: https://git-scm.com/download/win

Comprobar que está instalado, desde la consola con el comando:

```
C:\> git --version
git version 2.5.2.windows.2
```

4.5. Instalar visual studio code

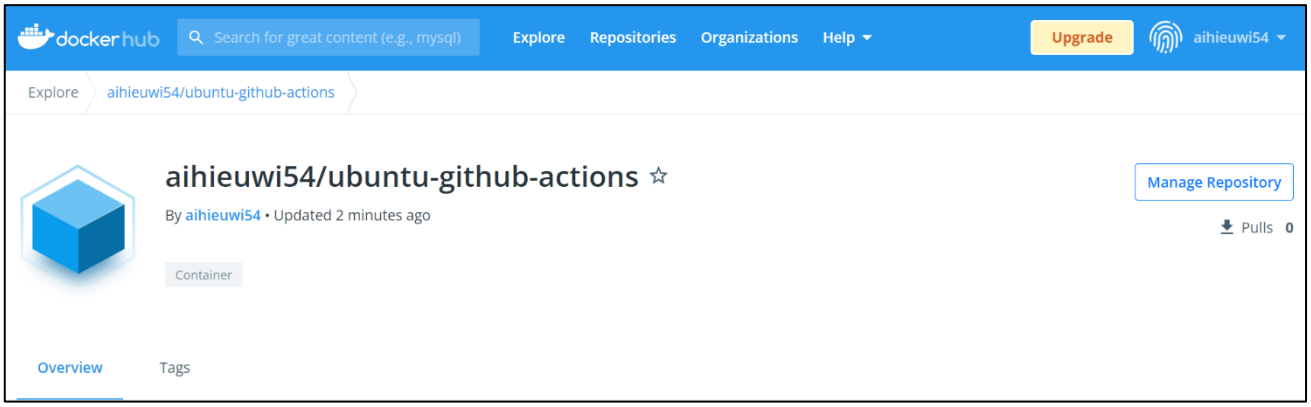
Aunque se puede utilizar cualquier entorno de desarrollo para Java, en esta práctica se asume que el desarrollador va a utilizar Visual Studio Code para hacer cambios en la aplicación web. Esta herramienta tiene la ventaja de ofrecer editores para todos los tipos de archivos que habrá que editar en la práctica, y además se integra con git, de tal forma que el desarrollador sabe en cada momento en que rama está trabajando, y los cambios pendientes de actualizar en el repositorio remoto.

Se descarga desde:

https://code.visualstudio.com/download

4.6. Instalar el contenedor Ubuntu

Descargar la imagen https://hub.docker.com/r/aihieuwi54/ubuntu-github-actions



Mediante el comando:

```
C:\> docker pull aihieuwi54/ubuntu-github-actions
```

Al finalizar, puede comprobarse que se ha descargado, mediante el comando:

```
  C:\>docker image ls
  REPOSITORY                                TAG         IMAGE ID         CREATED
SIZE
  aihieuwi54/ubuntu-github-actions  latest    5f46bb2764dd   13 minutes ago
1.21GB
```

Esta imagen permite crear el contenedor ubuntu-ci que aparece en la figura de la infraestructura del apartado 1, que contiene ya instalados: git, maven[1], github-actions, tomcat, mysql, y phantomjs. Estas herramientas serán necesarias durante la práctica.

Para crear el contenedor hay que ejecutar el comando:

```
  C:\>  docker  run  -it  --name  ubuntu-ci  -p  8080:8080  -p  3306:3306
aihieuwi54/ubuntu-github-actions


  * Starting MySQL database server mysqld
[ OK ]
  Using CATALINA_BASE:   /usr/local/tomcat
  Using CATALINA_HOME:   /usr/local/tomcat
```

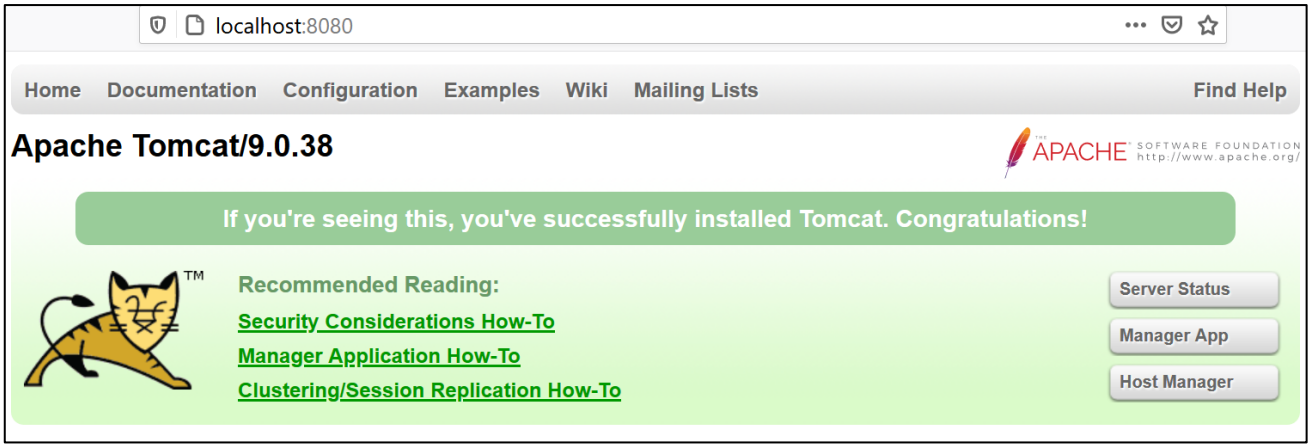
[1] No ha hecho falta instalar Maven en el ordenador local porque en esta práctica se usará desde este contenedor ubuntu-ci, en el que ya está instalado.

```
   Using CATALINA_TMPDIR: /usr/local/tomcat/temp
   Using JRE_HOME:        /usr
   Using CLASSPATH:
/usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar
   Using CATALINA_OPTS:
   Tomcat started.
   root@e72e51ef6029:/#
```

El contenedor está funcionado, es una máquina Linux (Ubuntu) que ha arrancado un servidor Tomcat en el Puerto 8080 y un servidor MySQL en el puerto 3306, después deja abierta la consola para que podamos introducir comandos de Ubuntu.

En la dirección https://hub.docker.com/layers/ubuntu-github-actions/aihieuwi54/ubuntu-github-actions/latest/images/sha256-8090ef06c82674702603bfe01a9a28ce878ef8d2aee409e918d0f5645af723e3?context=explore puede verse el archivo Dockerfile que se usó para definir la imagen a partir de la que se ha creado el contenedor.

Puede comprobarse que el servidor Tomcat, está arrancado, accediendo a la URL http://localhost:8080 desde un navegador:



En el caso de mysql, puede accederse al servidor mediante el comando mysql dentro de la consola del contenedor ubuntu-ci:

```
   # mysql
   Welcome to the MySQL monitor...


   > mysql> show databases;
   +--------------------+
   | Database           |
```

```
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

Para ver el estado del contenedor, sin cerrar la ventana de comandos, se puede abrir otra de Windows y ejecutar:

```
C:\> docker ps -a
CONTAINER ID        IMAGE                    COMMAND                  CREATED
STATUS              PORTS                                             NAMES
e72e51ef6029        aihieuwi54/ubuntu-github-actions "/bin/sh -c 'sudo se…"
17 minutes ago      Up 17 minutes       0.0.0.0:3306->3306/tcp, 0.0.0.0:8080-
>8080/tcp   ubuntu-ci
```

Si en algún momento se detiene el contenedor, se puede arrancar de nuevo mediante:

```
C:\> docker start -ai ubuntu-ci
```

Además, si queremos tener abierta otra consola del contenedor, se puede hacer mediante:

```
C:\> docker exec -it ubuntu-ci /bin/bash
```

4.7. Instalar el contenedor sonarqube

Como se indicaba en la figura de la infraestructura, también necesitaremos en la práctica un contenedor con un servidor sonarqube.

Para crearlo ejecutamos el comando:

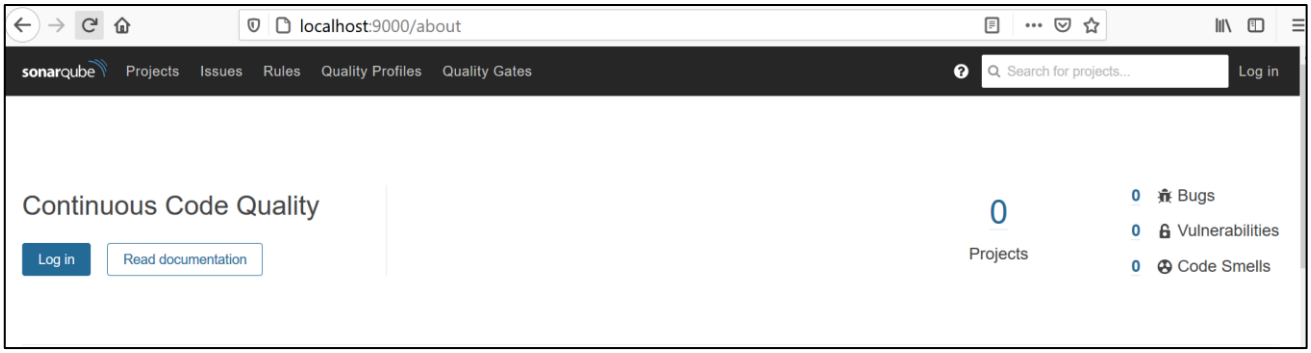
```
C:\> docker run -d --name sonarqube -p 9000:9000 sonarqube
```

Puede comprobarse que está arrancado con el comando docker ps:

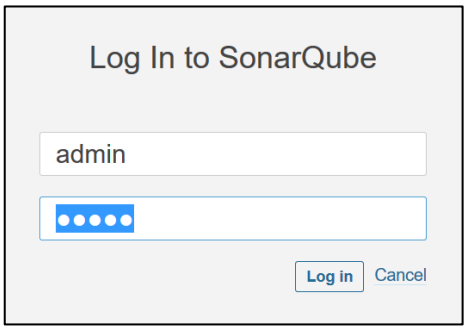
```
C:\> docker ps
CONTAINER ID        IMAGE                    COMMAND                  CREATED
STATUS              PORTS                                             NAMES
```

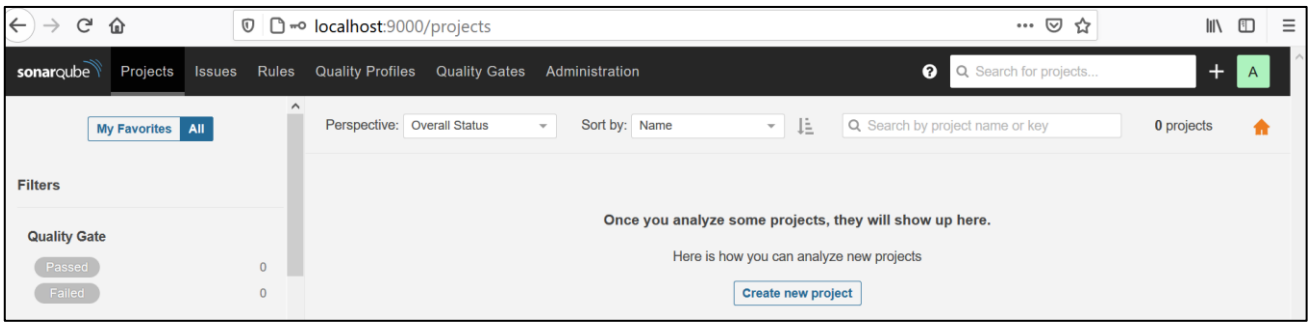
```
   333f1e1bad19        sonarqube                "bin/run.sh bin/sona…"   About
a   minute   ago        Up   About   a   minute        0.0.0.0:9000->9000/tcp
sonarqube

   e72e51ef6029        aihieuwi54/ubuntu-github-actions    "/bin/sh -c 'sudo
se…"    27 minutes ago        Up 27 minutes        0.0.0.0:3306->3306/tcp,
0.0.0.0:8080->8080/tcp   ubuntu-ci
```

Ahora puede accederse al servidor con un navegador, en la URL localhost:9000



Por defecto existe un usuario "admin" con clave "admin".





4.8. Entorno para pruebas locales en el ordenador de cada desarrollador

Para no alargar la práctica, no se usará un entorno para pruebas locales en el ordenador de cada desarrollador, pero hay que tener en cuenta que cada desarrollador trabaja con una copia local del código fuente de la aplicación en su máquina, y que cuando hace cambios en la aplicación, antes de enviarlos al repositorio remoto compartido, debe compilarlos y probarlos previamente en su máquina y asegurase de que funcionan al menos a nivel local, para lo cual necesitará instalarse un entorno de desarrollo, maven, y también su propio servidor web y de base de datos.

Es importante resaltar que estos servidores locales sólo los usa el desarrollador en su máquina para sus pruebas y no están compartidos con otros desarrolladores, que probablemente tendrán algo parecido en sus respectivos ordenadores. Es habitual que cada miembro del equipo utilice una infraestructura local similar o acordada por el equipo, pero lo importante en integración continua es que todos compartan el repositorio remoto de código, donde se harán las integraciones de los cambios de la aplicación.

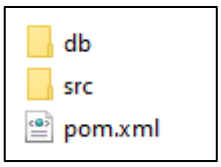
5. CREAR REPOSITORIO LOCAL Y PROYECTO EN GITHUB.COM

Partimos del código fuente de la aplicación Baloncesto disponible en el campus virtual en el archivo Baloncesto.zip.

Hay que descomprimirlo, obteniéndose el código mínimo para compilar y ejecutar la aplicación.

Para los ejemplos de esta práctica se descomprimirá en ordenador Windows, en la raíz de un disco C:, y como programa de comandos se usará cmd de Windows. Se puede utilizar otro lugar del disco, otro sistema operativo y otro intérprete de comandos, como bash, disponible para todos los sistemas.

Al descomprimir se obtiene la carpeta (directorio) "Baloncesto" con la estructura:

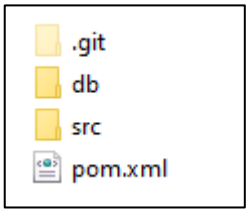


5.1. Crear repositorio git local

Para convertir la carpeta Baloncesto en un repositorio git para el control de versiones, y que permita subir (push) y bajar (pull) cambios en el código al repositorio remoto compartido por todos los desarrolladores del proyecto en el servidor de integración continua github.com, se debe ejecutar el comando git init dentro de la carpeta:

```
C:\Baloncesto> git init
Initialized empty Git repository in D:/Baloncesto/.git/
```

Se ha creado una carpeta .git oculta con los archivos que necesita git para gestionar versiones del repositorio.



Ahora hay que crear en la carpeta del proyecto, con un editor de texto, un archivo con el nombre .gitignore (IMPORTANTE: sin extensión y con punto delante del nombre), en el que se indicarán los archivos y carpetas completas que no deberán subirse al repositorio remoto cuando se hagan envíos con push, ya que son archivos de uso particular del desarrollador, que han de ser ignorados por git, y no subirlos al remoto.

Por ejemplo, hay que incluir la carpeta /target/, ya que contiene los resultados de la compilación de la aplicación que haga el desarrollador para sus pruebas locales, pero que no hay que subirla al repositorio remoto compartido, en que sólo debe almacenarse el código fuente de la aplicación, no los archivos resultantes de la compilación ni ejecutables. El servidor de integración continua github.com se encargará de recompilar la aplicación si se aceptan los cambios que envíen los desarrolladores.

Otros archivos que hay que incluir en .gitignore son aquellos que genere el entorno de programación que utilice el desarrollador. Puede ocurrir que cada desarrollador de un mismo proyecto utilice en su ordenador un entorno de desarrollo diferente (Visual Studio Code, Netbeans, Eclipse, IntelliJ, etc.), que crea archivos en la carpeta del proyecto que no hay que subir al repositorio remoto, en el que el código compartido por todos los desarrolladores debe ser independiente de cualquier entorno de desarrollo. Por ejemplo, si se desarrolla localmente con Netbeans, existirá una carpeta nbproject, en el caso de usar VS Code, existirá la carpeta .vscode, etc, que habrá que ignorar.

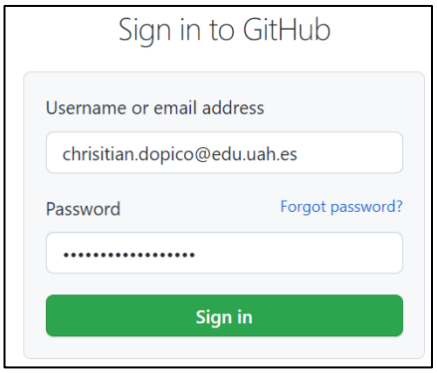
Para evitar que se suban al repositorio remoto, el archivo .gitignore debería ser el siguiente, indicando en cada fila un archivo o carpeta que git debe ignorar al hacer push hacia el repositorio remoto.

Archivo C:\Baloncesto\.gitignore

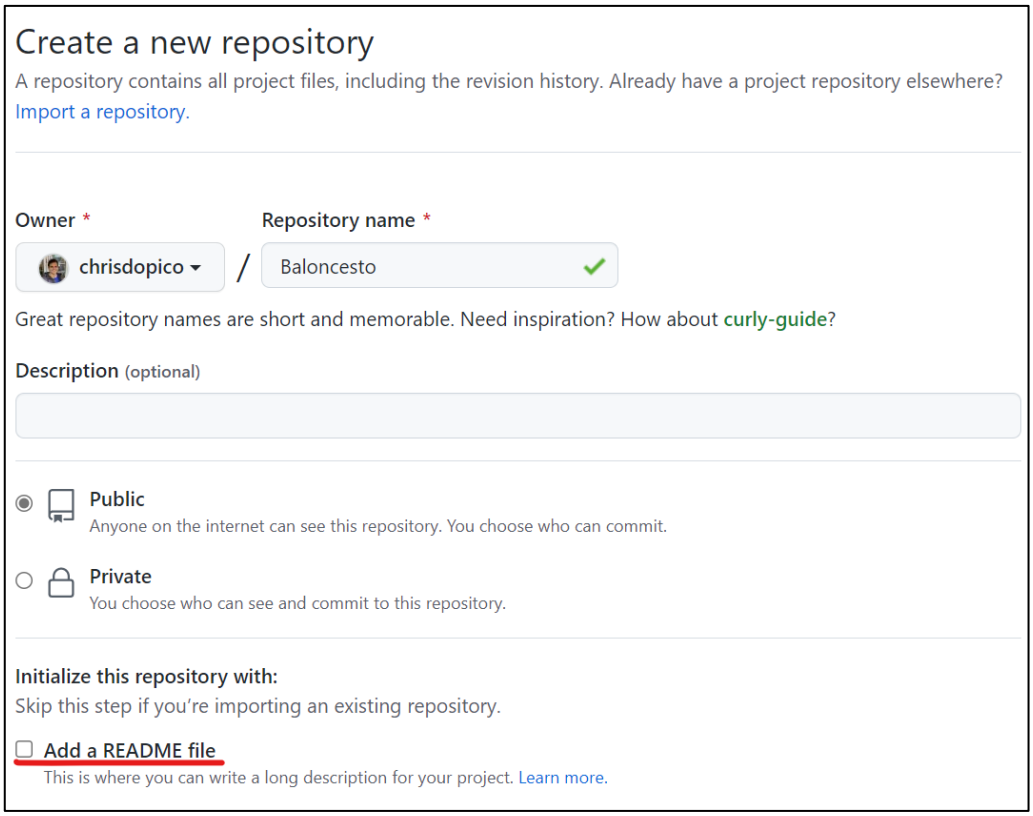
`/target/`
`/nbproject/`
`/.vscode/`

5.2. Crear repositorio remoto y proyecto en Github

Hay que tener una cuenta en github.com, en los ejemplos de esta práctica se utilizará la cuenta chrisitian.dopico. Cada alumno debe utilizar la suya.



Una vez dentro de Github, para crear un repositorio se selecciona el botón New, indicando como nombre “Baloncesto”, visibilidad “Public”, no marcar “Initialize repository with a README”.

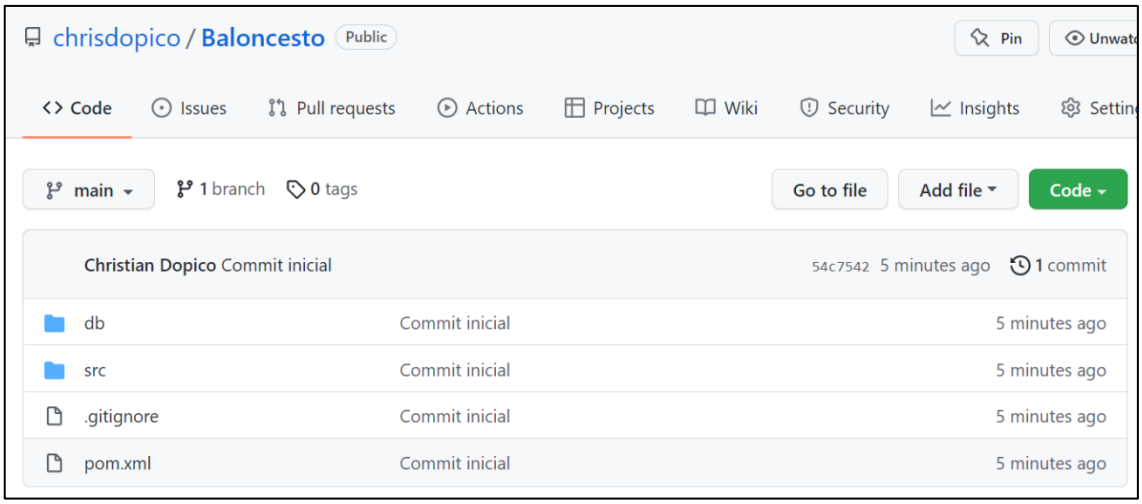


Al pulsar el botón para crear repositorio aparecen sugerencias de comandos para subir un repositorio existente: "Push an existing folder".

Volvemos a la consola de nuestro ordenador y dentro de la carpeta Baloncesto ejecutamos los siguientes comandos:

```
C:\Baloncesto> git remote add origin
https://github.com/chrisdopico/Baloncesto.git
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Commit inicial"
C:\Baloncesto> git branch -M main
C:\Baloncesto> git push -u origin main
```

Si volvemos a github.com, vemos en la sección Repository que, se ha subido una réplica del repositorio local, excepto la carpeta oculta .git, y los archivos y subcarpetas indicados en .gitignore.



5.3. Activar github-self-hosted-runner en el contenedor ubuntu-ci

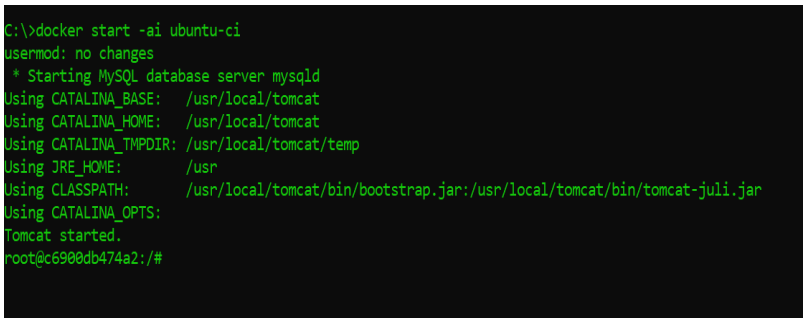
Github.com utiliza el servicio externo github-runner para ejecutar las órdenes de los workflow del proyecto. Ese servicio puede estar instalado en un ordenador Linux, Windows o Mac con conexión a internet, y se recomienda que esté en una máquina diferente a la del servidor github.com. En esta práctica se ha instalado en el contenedor ubuntu-ci que se ha creado en un apartado anterior.

Pero para que quede en funcionamiento y conectado al servidor github.com para que le ordene la ejecución de trabajos, primero hay que registrarlo y luego hay que ejecutarlo.

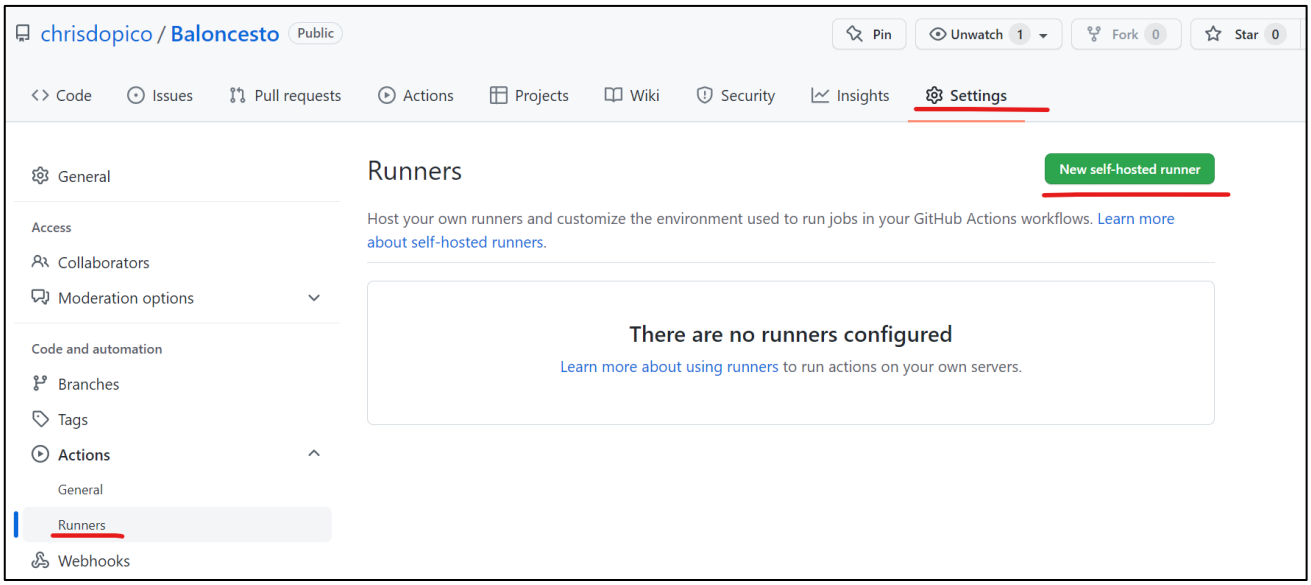
Hay que tener arrancado el contenedor ubuntu-ci, como se indicó en el apartado 4.7. Si estuviera parado, se puede reactivar con el comando:

```
C:\> docker start -ai ubuntu-ci
```

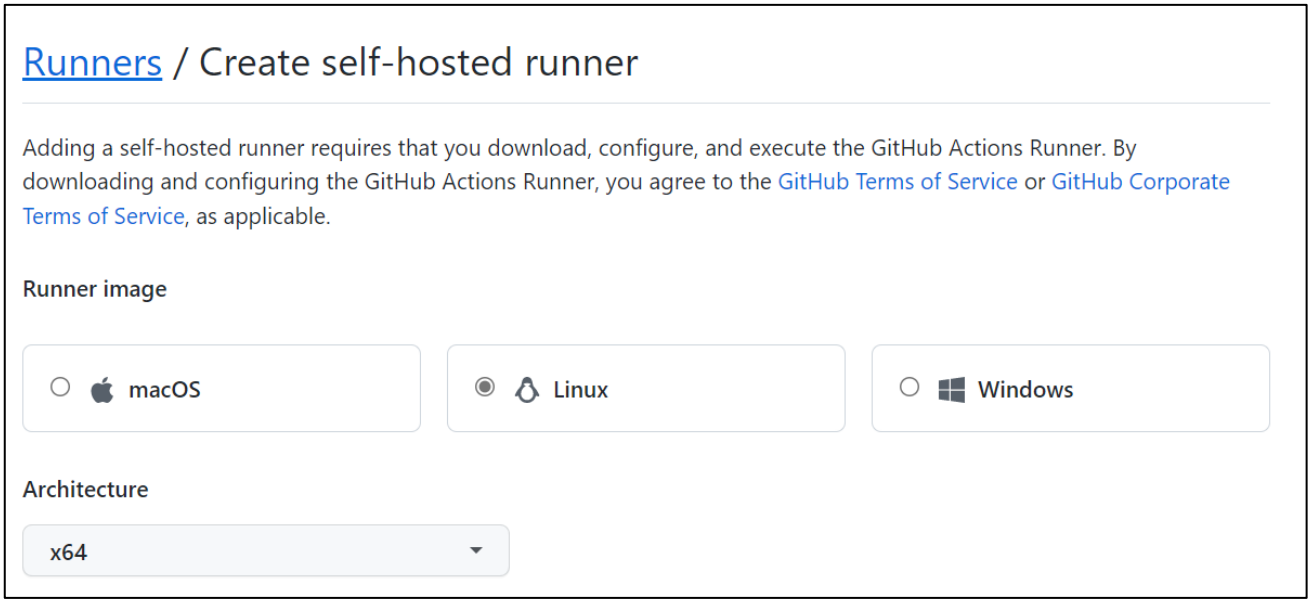
Y aparece la consola de comandos de ubuntu-ci:



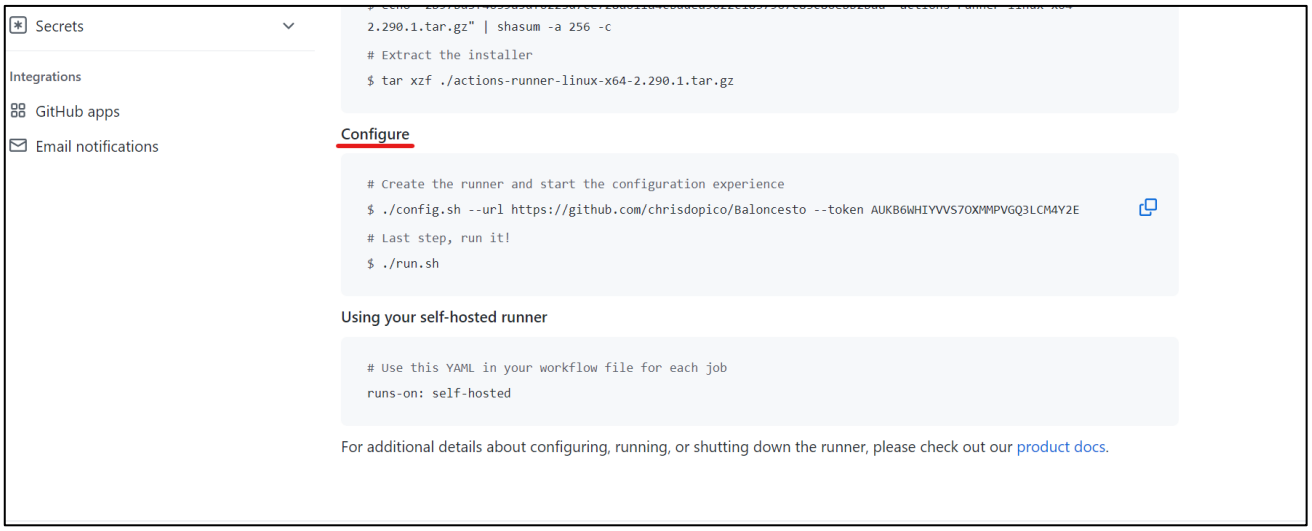
Dentro del repositorio de github nos dirigimos a la sección Settings>Actions>Runners y seleccionamos "New self-hosted runner"



Una vez seleccionado "New self-hosted runner", nos aparecerá la siguiente pantalla donde seleccionaremos la opción de Linux.



En la parte inferior, podremos ver la configuración que necesitamos para crear el runner en nuestra máquina virtual de Ubuntu.



Para poder establecer la configuración, debemos agregar la siguiente variable de entorno al inicio de nuestros comandos:

```
$ RUNNER_ALLOW_RUNASROOT="1"
```

Y después de definir la variable, copiamos la primera línea de la configuración del runner, donde se encuentra el token de nuestro repositorio, la configuración sería la siguiente:

Una vez ingresado el token de nuestro repositorio, procedemos a configurar las características de nuestro runner:

-Para el nombre de grupo, presionamos enter para seleccionar el nombre por defecto:

```
Enter the name of the runner group to add this runner to: [press Enter for
Default]
```

- Para el nombre del runner, escribiremos ubuntu:

```
Enter the name of runner: [press Enter for c6900db474a2] ubuntu
```

- Para el label, presionamos enter para evitar este paso:

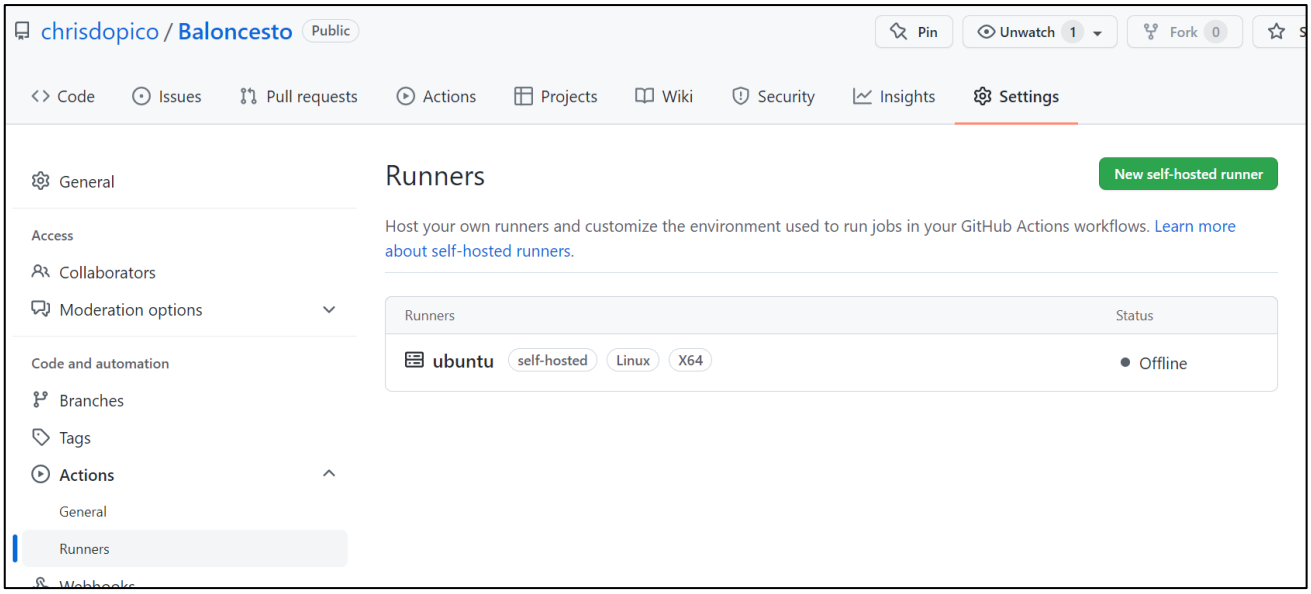
```
This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]
```

- Finalmente, se selecciona el nombre de la carpeta de trabajo, para este caso presionaremos enter para seleccionar el nombre por defecto:

```
Enter name of work folder: [press Enter for _work]
```

```
√ Connected to GitHub

# Runner Registration

Enter the name of the runner group to add this runner to: [press Enter for Default]

Enter the name of runner: [press Enter for c6900db474a2] ubuntu

This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]

√ Runner successfully added
√ Runner connection is good

# Runner settings

Enter name of work folder: [press Enter for _work]

√ Settings Saved.

root@c6900db474a2:/#
```

Si nos redirigimos al repositorio Balocensto>Settings>Actions>Runners podremos ver nuestro runner ya configurado.

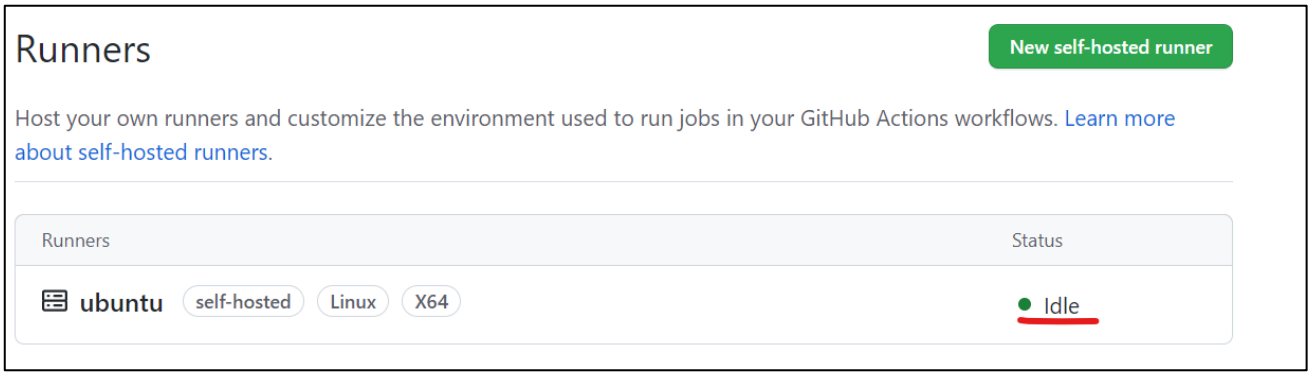


Para iniciar el runner, debeos ejecutar el siguiente comando en nuestro contenedor Ubuntu:

```
RUNNER_ALLOW_RUNASROOT="1" ./run.sh
```

```
root@c6900db474a2:/# RUNNER_ALLOW_RUNASROOT="1" ./run.sh

√ Connected to GitHub

Current runner version: '2.290.1'
2022-04-25 23:44:09Z: Listening for Jobs
```

Ahora el runner nos aparecerá con el estatus Idle, lo que significa que está listo para ejecutar el workflow.

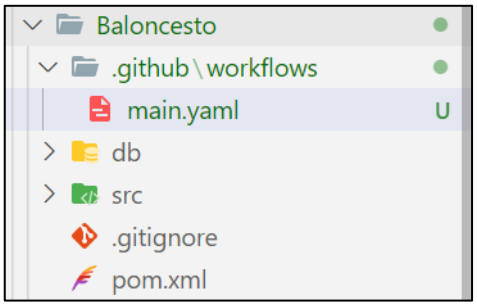


6. DISEÑAR EL FLUJO DE TRABAJOS PARA LA INTEGRACIÓN CONTINUA (PIPELINE)

Github Actions actúa como servidor de integración continua, y en el repositorio creado todos los miembros de proyecto irán subiendo cambios.

Cada vez que se sube un cambio hay que compilar, probar y, en su caso, desplegar la aplicación a producción. Un principio de la integración continua es que el repositorio tenga todo lo necesario para recompilar y ejecutar la aplicación, que ha podido funcionar en el ordenador local, pero se trata de que funcione en uno o varios servidores web externos, bajo el control del servidor de integración.

Para que cada vez que se suba un cambio se recompile la aplicación, hay que preparar un flujo de trabajos (pipeline, sin embargo, en Github Actions se utiliza el término workflow). Para esto, debemos crear el siguiente directorio dentro de Baloncesto: .github\workflows (**IMPORTANTE: debe empezar por punto**). Dentro de este directorio, crearemos el archivo main.yaml.



Este archivo lo creamos en el repositorio local con un editor de texto o un entorno de desarrollo. Se recomienda usar Visual Studio Code, pues ofrece utilidades para sincronizarse con el repositorio remoto, y nos indica en la parte inferior izquierda en que rama del proyecto estamos trabajando en cada momento.

Escribimos en el archivo main.yaml el siguiente contenido:

- Primero definimos el nombre de nuestro workflow. En este caso se llamará "CI".
- Definimos los triggers para nuestro workflow, en este caso, será cada vez que hagamos un push a cualquier rama incluidos los merge request.
- Definimos cinco jobs (pruebas-unitarias, empaquetar, pruebas-funcionales, calidad-código y depliegue en heroku).
- Dentro de cada job debemos especificar que se ejecutará en un servidor propio "self-hosted", que este caso será nuestra máquina virtual de Ubuntu. También podremos definir los steps para cada job, donde incluiremos el nombre de cada uno y su script de ejecución.

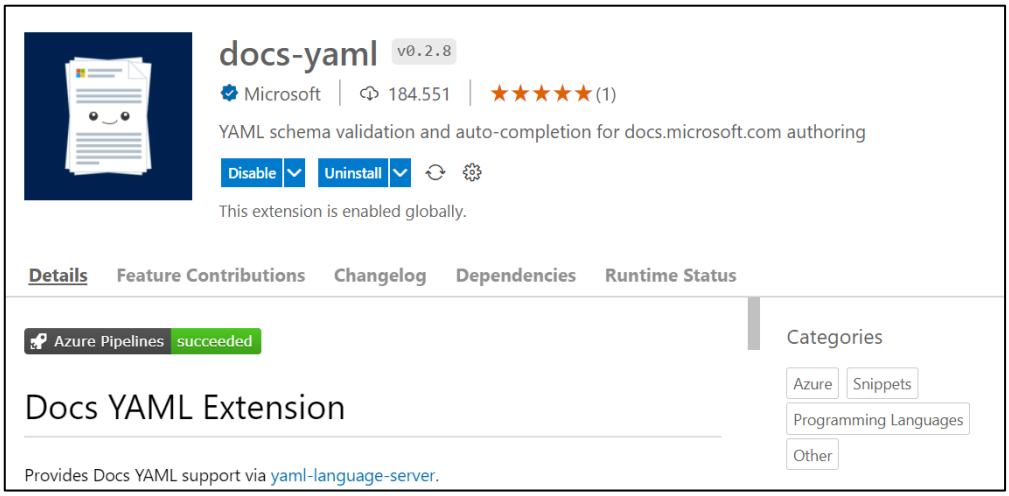
Archivo C:\Baloncesto\.github\workflows\main.yaml

```
name: CI

on:
  push:
    branches:
      - '**'
```

```
jobs:
  pruebas-unitarias:
    runs-on: self-hosted
    steps:
      - name: unit tests
        run: echo "pruebas-unitarias"

  empaquetar:
    runs-on: self-hosted
    steps:
      - name: build
        run: echo "empaquetar"

  pruebas-funcionales:
    runs-on: self-hosted
    steps:
      - name: test
        run: echo "pruebas-funcionales"

  calidad-codigo:
    runs-on: self-hosted
    steps:
      - name: qa
        run: echo "calidad-codigo"

  despliegue-host-heroku:
    runs-on: self-hosted
    steps:
      - name: deploy
        run: echo "despliegue-host-heroku"
```

De momento en el archivo main.yaml sólo hemos definido en los trabajos un comando echo que muestre el nombre del trabajo. En los siguientes apartados los sustituiremos por comandos que realicen de verdad el trabajo previsto en cada caso.

NOTA: Para validar nuestros archivos .yaml en VS Code, podemos hacer uso del siguiente la siguiente extensión:

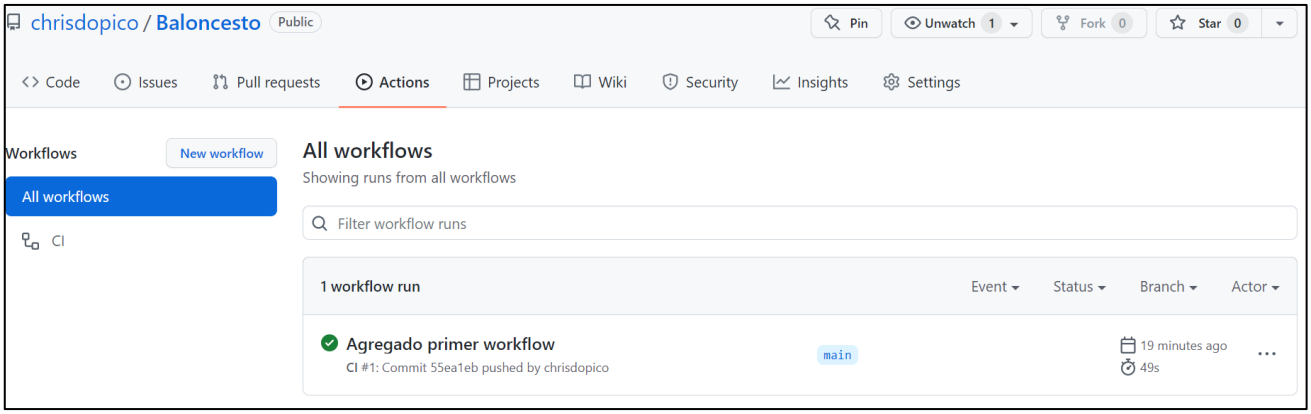


Los trabajos definidos en main.yaml se ejecutarán de forma automática cada vez que un desarrollador envié cambios al repositorio remoto desde su repositorio local mediante un push.

Como acabamos de modificar el repositorio local añadiendo este nuevo archivo, podemos hacer un push contra el repositorio remoto para actualizar los cambios, y comprobaremos que se ejecutan los trabajos definidos en el archivo.

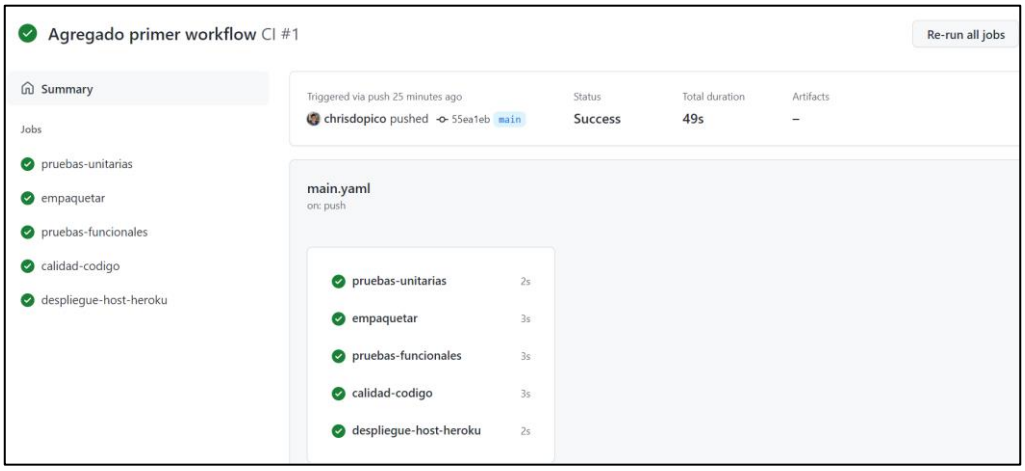
```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Agregado primer workflow"
C:\Baloncesto> git push origin main
```

En la sección Baloncesto>Actions>All workflows de github.com podemos ver las ejecuciones de los trabajos definidos en el archivo main.yaml, cada workflow es la ejecución de todos los trabajos como consecuencia de un push, y el nombre del workflow es el del commit correspondiente a ese push. En este caso "Agregado primer workflow".

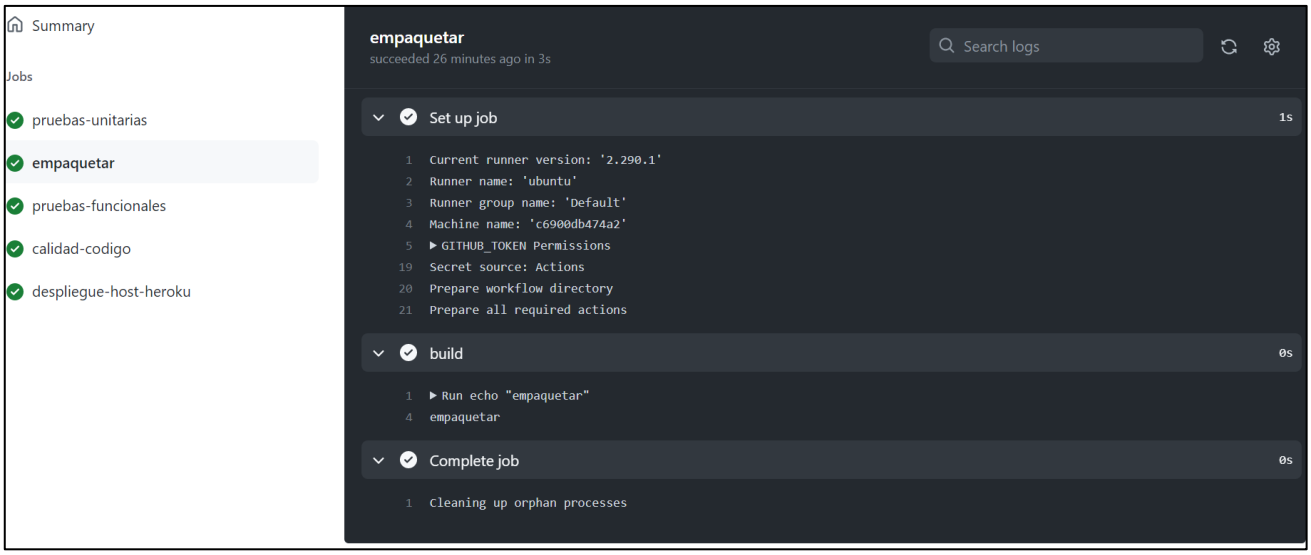


Puede comprobarse que la ejecución del workflow ha sido un éxito porque el status indica en verde "success".

Si seleccionamos el identificador en la columna workflow, vemos un diagrama con las fases y trabajos realizados:



Todos están en verde indicando que han pasado con éxito. Si seleccionamos uno de los trabajos, por ejemplo "empaquetar", podemos ver la consola del contenedor ubuntu-ci en el que está el runner vinculado al proyecto que creamos en el apartado anterior, y lo que ha ido ocurriendo.



Podemos ver que primero se conectó al runner y verifica los permisos del repositorio, después ejecuta el step "build" con el comando echo que había en la sección script de este trabajo. También vemos a la derecha que el trabajo se ha ejecutado en un segundo.

Hay que tener en cuenta que el trabajo no se ha ejecutado en el servidor github.com sino en el ordenador en el que está el runner instalado.

En los siguientes apartados vamos a ir definiendo el detalle de los trabajos a realizar, modificando el archivo main.yaml .

6.1. Trabajo Unit Tests: Trabajo pruebas-unitarias

Antes de empaquetar nuestra aplicación, lo primero que debemos verificar es que todos los test unitarios se ejecuten de manera correcta.

Dentro de nuestro archivo main.yaml, en el job pruebas unitarias, agregamos lo siguiente:

- Un nuevo step "Checkout Repository", con el action "actions/checkout@v3". Esto lo que hará es descargar el repositorio de nuestra aplicación en nuestra máquina de Ubuntu para poder ejecutar la aplicación.
- Modificar el step "unit test" utilizando el siguiente comando "mvn test". Este comando lo que hará es ejecutar todos los test unitarios de nuestro proyecto Java.

Archivo C:\Baloncesto\.github\workflows\main.yaml

```
name: CI

on:
  push:
    branches:
      - '**'
```

```
jobs:
  pruebas-unitarias:
    runs-on: self-hosted
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2.3.3
      - name: unit tests
        run:  mvn test
```

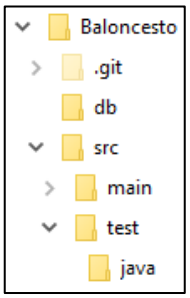
Para ejecutar las pruebas unitarias hay que decidir qué framework se utilizará, en este caso usaremos JUnit 5, por lo que hay que modificar el archivo pom.xml del repositorio local para añadir las dependencias necesarias:

Archivo C:\Baloncesto\pom.xml

```
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.3.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <version>5.3.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.3.1</version>
            <scope>test</scope>
        </dependency>
```

6.1.1. Realizar una prueba fallida

Como es un proyecto con estructura maven, las pruebas deben programarse en la carpeta `src\test\java\`, que hay que crear.



Vamos a crear un archivo `ModeloDatosTest.java` en dicha carpeta, con el siguiente código.

Archivo C:\Baloncesto\src\test\java\ModeloDatosTest.java

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class ModeloDatosTest {

    @Test
    public void testExisteJugador() {
        System.out.println("Prueba de existeJugador");
        String nombre = "";
        ModeloDatos instance = new ModeloDatos();
        boolean expResult = false;
        boolean result = instance.existeJugador(nombre);
        //assertEquals(expResult, result);
        fail("Fallo forzado.");
    }
}
```

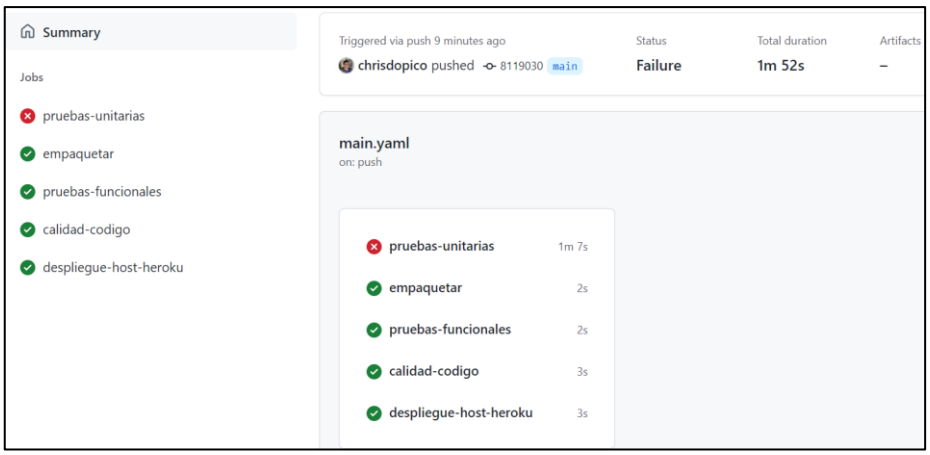
Se trata de una prueba del método `existeJugador()`. Es una prueba ficticia sólo para probar el funcionamiento de la integración continua. Se recomienda que se prepare una prueba real.

En este caso se ha incluido una sentencia "fail" para forzar un error y comprobar que github lo detecta cuando se ejecute el workflow.

Una vez guardados los tres archivos modificados: main.yaml, pom.xml y ModeloDatosTest.java, hacemos push al repositorio remoto, para ver que, al desencadenarse de nuevo el workflow, se interrumpe el trabajo prueba-unitaria.

```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Fallo forzado en prueba unitaria"
C:\Baloncesto> git push
```

Puede comprobarse en el nuevo workflow que ahora falla el trabajo pruebas-unitarias:



Si accedemos al detalle del trabajo pruebas-unitarias, puede verse que ha fallado la prueba `testExisteJugador`, debido a que habíamos forzado la ejecución de la sentencia `fail`.

```
457  Failed tests:   ModeloDatosTest.testExisteJugador(): Fallo forzado.
458
459  Tests run: 1, Failures: 1, Errors: 0, Skipped: 0
460
461  [INFO] ------------------------------------------------------------------------
462  [INFO] BUILD FAILURE
463  [INFO] ------------------------------------------------------------------------
464  [INFO] Total time:  9.356 s
465  [INFO] Finished at: 2022-04-26T21:43:35Z
466  [INFO] ------------------------------------------------------------------------
467  [ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.12.4:test (default-test) on project
     Baloncesto: There are test failures.
468  [ERROR]
469  [ERROR] Please refer to /_work/Baloncesto/Baloncesto/target/surefire-reports for the individual test results.
470  [ERROR] -> [Help 1]
471  [ERROR]
472  [ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
473  [ERROR] Re-run Maven using the -X switch to enable full debug logging.
474  [ERROR]
475  [ERROR] For more information about the errors and possible solutions, please read the following articles:
476  [ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
477  Error: Process completed with exit code 1.
```

6.1.2. Realizar una prueba sin errores

Para que no falle la prueba, hay que borrar o comentar la sentencia `fail` en el archivo `\src\test\java\ModeloDatosTest.java`, y quitar el comentario de la sentencia `assertEquals`:

```
assertEquals(expResult, result);
//fail("Fallo forzado.");
```

Guardar el archivo y volver a subir al repositorio remoto:

```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Prueba unitaria corregida"
C:\Baloncesto> git push
```

Puede comprobarse que este segundo workflow se realiza sin problemas:

6.2. Fase (stage) build: Trabajo (job) empaquetar

Vamos a definir el trabajo "empaquetar" para el proyecto Baloncesto. Se trata de que se compile y se empaquete la aplicación en un archivo baloncesto.war. Lo más sencillo es utilizar maven, pues esto se puede hacer con un único comando. Por lo que modificamos el archivo main.yaml cambiando el comando echo por un comando mvn.

Archivo C:\Baloncesto\.github\workflows\main.yaml

```
name: CI

on:
  push:
    branches:
      - '**'

jobs:
  pruebas-unitarias:
    runs-on: self-hosted
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2.3.3
      - name: unit tests
        run:  mvn test

empaquetar:
    runs-on: self-hosted
    steps:
      - name: build
        run: mvn package -DskipTests=true
```

- Dentro del step "build" con el comando mvn indicamos la opción DskipTest a true, porque las pruebas unitarias se realizarán en el apartado anterior.

Hay que tener en cuenta que cuando el workflow inicia, hace una descarga y/o actualiza nuestro repositorio en la carpeta de trabajo "_work", de hecho, si echamos un vistazo a los directorios, podremos ver una copia de nuestro repositorio en nuestra máquina virtual de Ubuntu.

```
$ cd _work/Baloncesto/Baloncesto
```

```
root@d526d3f34d18:/# cd _work/Baloncesto/Baloncesto
root@d526d3f34d18:/_work/Baloncesto/Baloncesto# ls
db  phantomjsdriver.log  pom.xml  src  target
```

Todavía no hacemos push contra el repositorio remoto, lo haremos en el siguiente apartado.

6.2.1. Compilar con un error forzado

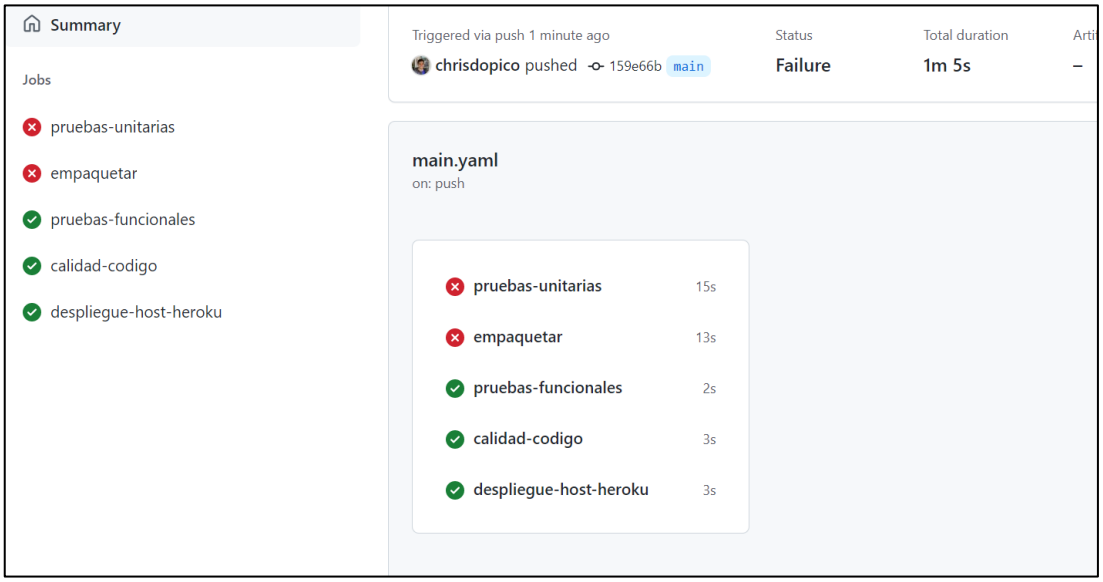
Vamos a ver cómo falla el trabajo de empaquetado forzando un error de compilación. Para ello modificamos con el editor el archivo en src\main\java\Acb.java en nuestro repositorio local, comentando la primera línea de código para forzar un error de compilación.

```
//import java.io.*;
```

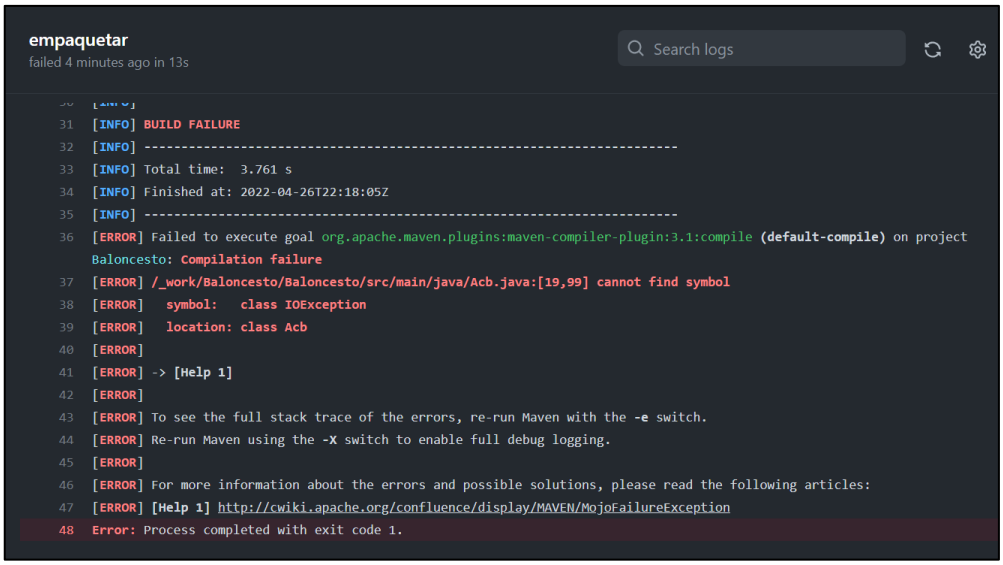
Salvamos todo y lo subimos al repositorio remoto:

```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Error de compilación forzado"
C:\Baloncesto> git push
```

Vemos que se ha creado un nuevo workflow, y que se ha producido un error en el trabajo "empaquetar". Por lo tanto, también en el trabajo pruebas unitarias ya que hemos forzado un error de compilación, por lo que tampoco se pueden ejecutar los tests.



Esto supone que habrá que revisar los errores y corregirlos. En la consola del trabajo empaquetar se puede ver cuál ha sido el error:



6.2.2. Empaquetar sin errores

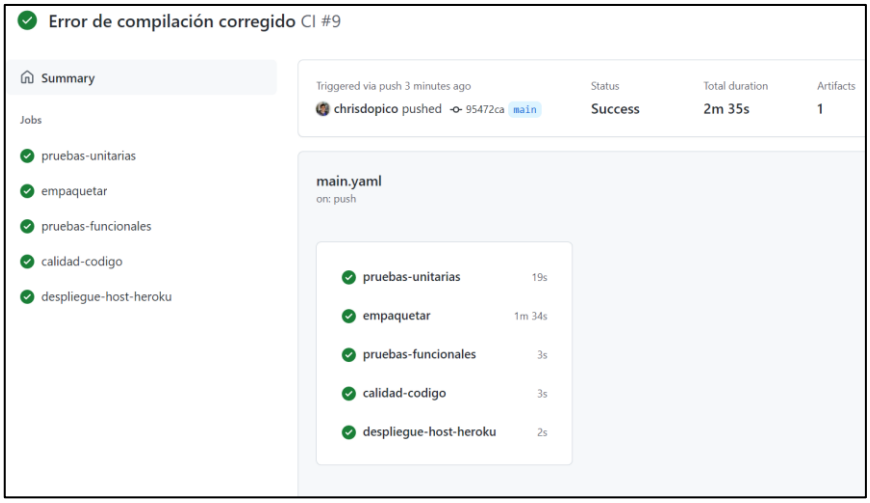
Para corregir el error, editamos de nuevo en local quitando el comentario de Acb.java:

```
import java.io.*;
```

Guardamos y lo volvemos a subir:

```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Error de compilación corregido"
C:\Baloncesto> git push
```

Comprobamos que se arranca un nuevo workflow y que ahora no hay ningún error.



6.3. Fase test: Trabajo pruebas-funcionales

La siguiente fase definida en el workflow es la de test, que se realiza una vez empaquetada la aplicación y superadas las pruebas unitarias. En esta fase hay un solo trabajo denominado pruebas-funcionales.

En este trabajo, hay que preparar los comandos necesarios para de forma automática:

- Instalar la aplicación Baloncesto, que se empaquetó en la fase anterior, en un servidor Tomcat
- Crear una base de datos "baloncesto" en un servidor MySQL
- Realizar pruebas del funcionamiento de la aplicación simulando su uso por parte de un usuario que utiliza un navegador web simulado, para lo cual se utilizarán las librerías Selenium y Phantomjs.

Para este punto vamos a agregar la dependencia entre trabajaos. Primero, haremos que el trabajo de empaquetar se ejecute una vez finalizado el trabajo de pruebas-unitarias. Después haremos que el trabajo pruebas-funcionales se ejecute una vez terminado el trabajo de empaquetar. Esto se realiza por medio de la directiva:

needs: pruebas-unitarias

Hay que modificar el archivo main.yaml cambiando el trabajo los trabajos (pruebas-unitarias, empaquetar y pruebas-funcionales) por el código siguiente:

Archivo C:\Baloncesto\.github\workflows\main.yaml

```
pruebas-unitarias:
  runs-on: self-hosted
  steps:
    - name: Checkout repository
      uses: actions/checkout@v2.3.3
    - name: unit tests
      run:  mvn test

empaquetar:
  runs-on: self-hosted
  needs: pruebas-unitarias
  steps:
    - name: build
      run: mvn package -DskipTests=true

pruebas-funcionales:
  runs-on: self-hosted
  needs: empaquetar
  steps:
    - name: test
      run: |
        cp -r target/Baloncesto /usr/local/tomcat/webapps
        mysql -u root < db/baloncesto.sql
        export DATABASE_HOST="jdbc:mysql://localhost"
        export DATABASE_PORT="3306"
        export DATABASE_NAME="baloncesto"
        export DATABASE_USER="usuario"
        export DATABASE_PASS="clave"
        sh /usr/local/tomcat/bin/catalina.sh stop
        sh /usr/local/tomcat/bin/catalina.sh start
        mvn failsafe:integration-test failsafe:verify
```

Hay que tener en cuenta que estos comandos se ejecutarán en la máquina (contenedor) ubuntu-ci, en la que esté corriendo el runner creado para el proyecto (ver apartado 2.3 de esta práctica). Y en ese contenedor ya están instalados y arrancados un servidor Tomcat y un servidor MySQL (ver apartado 1.6).

Por ello la instalación de la aplicación es tan sencillo como copiar la carpeta target/Baloncesto en la carpeta webapps de Tomcat.

A continuación se explica el contenido de main.yaml:

- **Sección "dependencies":** Para tener la carpeta target cuando el workflow llegue a la fase de "pruebas funcionales", debemos generar una dependencia con los otros trabajos. Esto se realiza por medio de la clave "needs". Para este caso, vamos a realizar un trabajo en serie, es decir: primero se ejecutarán las pruebas unitarias, luego el empaquetado y finalmente las pruebas funcionales.
- **Comando para instalar la aplicación web:** Como sabemos que, al arrancar el trabajo, en el contenedor ubuntu-ci estará la carpeta target con la aplicación generada, podemos ejecutar el comando `cp -r target/Baloncesto /usr/local/tomcat/webapps`, para copiar la subcarpeta Baloncesto en tomcat, y ya estaría instalada la aplicación.
- **Comando para crear la base de datos:** Como al arrancar el trabajo se copia el repositorio del proyecto en el contenedor ubuntu-ci, en la subcarpeta db tenemos el archivo baloncesto.sql con el script de creación de la base de datos, por lo que sólo hay que ordenar su ejecución desde la consola de ubuntu-ci con el comando `mysql -u root < db/baloncesto.sql`. Hay que tener en cuenta que el usuario root por defecto no tiene clave, por eso no es necesaria en este comando.
- **Comandos export para las variables de entorno**: En el código de ModeloDatos.java, los datos de conexión se obtienen de variables de entorno. En los cinco comandos export se da valores a dichas variables.
- **Comandos de parada y re-arranque de Tomcat**: Para que el servidor Tomcat cargue las variables de entorno hay que pararlo y volverlo a arrancar.
- **Comando para ejecutar las pruebas funcionales**: Al utilizar maven, es tan secillo como ejecutar el comando `mvn failsafe:integration-test`. Este comando localiza los archivos cuyo nombre termina en "IT" (Integration Test) dentro de la carpeta src/test/java y los ejecuta. Se supone que en esos archivos están programadas las pruebas de integración de la aplicación, entre las que se encuentran las pruebas funcionales, también llamadas end-to-end o de interfaz de usuario.

La prueba vamos a programar en el siguiente apartado utiliza las librerías Selenium y phantomjs, por lo que hay que añadir dos nuevas dependencias al archivo pom.xml del proyecto:

Archivo C:\Baloncesto\pom.xml

```
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>2.41.0</version>
        </dependency>

        <dependency>
            <groupId>com.github.detro.ghostdriver</groupId>
            <artifactId>phantomjsdriver</artifactId>
            <version>1.1.0</version>
```

```
        </dependency>
```

6.3.1. Realizar una prueba fallida

Vamos a programar una prueba funcional en un archivo que llamaremos PruebasPhantomjsIT.java, ubicado en la carpeta `src\test\java\`.

Cuando se programan pruebas de este tipo hay que preparar pruebas que simulen diferentes navegadores web, con la limitación de que el contenedor ubuntu-ci no tiene interfaz gráfica de usuario, por lo que hay que usar algún navegador sin interfaz de usuario ("headless browser"). Existen versiones headless para Chrome, Firefox, etc., y se anima al alumno a crear la misma prueba para ellos. Por sencillez, en esta práctica usaremos Phantomjs.

El código del archivo para las pruebas es el siguiente:

Archivo C:\Baloncesto\src\test\java\PruebasPhantomjsIT.java

```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.phantomjs.PhantomJSDriver;
import org.openqa.selenium.phantomjs.PhantomJSDriverService;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class PruebasPhantomjsIT
{
      private static WebDriver driver=null;

      @Test
      public void tituloIndexTest()
      {
            DesiredCapabilities caps = new DesiredCapabilities();
            caps.setJavascriptEnabled(true);

      caps.setCapability(PhantomJSDriverService.PHANTOMJS_EXECUTABLE_PATH_PR
OPERTY,"/usr/bin/phantomjs");
            caps.setCapability(PhantomJSDriverService.PHANTOMJS_CLI_ARGS,
new String[] {"--web-security=no", "--ignore-ssl-errors=yes"});
            driver = new PhantomJSDriver(caps);
            driver.navigate().to("http://localhost:8080/Baloncesto/");
            assertEquals("Votacion mejor jugador liga ACB",
driver.getTitle(), "El titulo no es correcto");
        System.out.println(driver.getTitle());
            driver.close();
            driver.quit();
      }
}
```

Esta prueba lo único que hace es abrir la aplicación Baloncesto previamente instalada en Tomcat, y comprueba el contenido de la etiqueta `<title>` del archivo index.html de la aplicación, que es el que se

abre por defecto en la URL `http://localhost:8080/Baloncesto/`. Comprueba si el título es "Votacion mejor jugador liga ACB".

Como en la versión actual del archivo `index.html` está bien, vamos a forzar un error cambiando "ACB" por "NBA" en la etiqueta <title>:

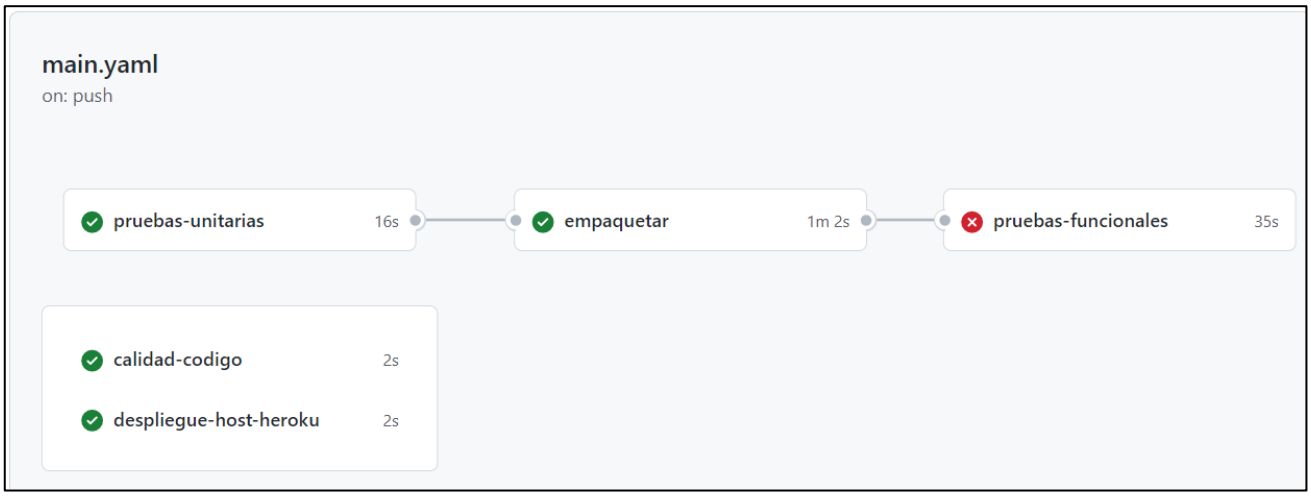
```
<title>Votacion mejor jugador liga NBA</title>
```

Debido a esto deberá generarse un error, porque el contenido de la etiqueta `<title>` esperado sería "Votacion mejor jugador liga ACB".

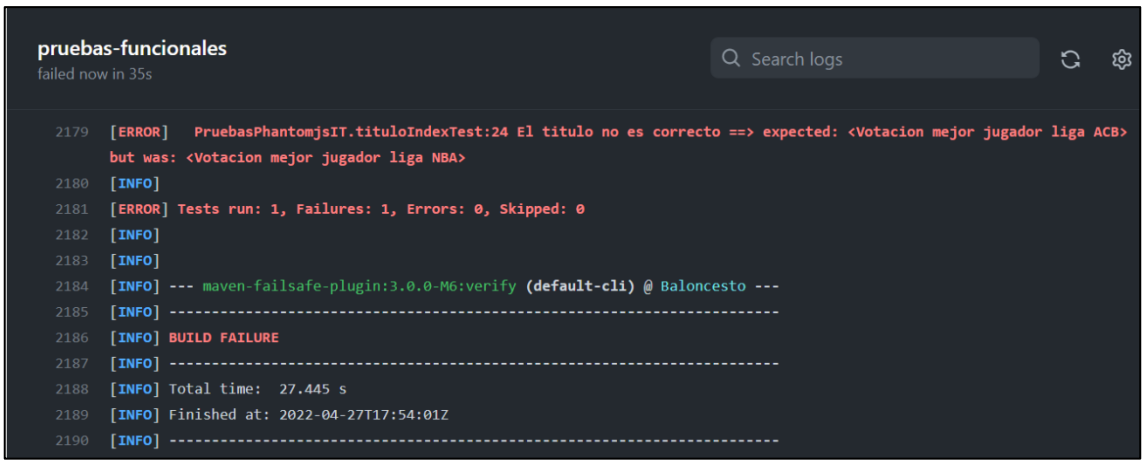
Guardamos los cuatro archivos modificados: main.yaml, pom.xml, index.html y PruebasPhantomjsIT.java, hacemos push al repositorio remoto, para ver que, al desencadenarse de nuevo el workflow, se interrumpe el trabajo pruebas-funcionales.

```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Fallo forzado en prueba funcional"
C:\Baloncesto> git push
```

Puede comprobarse en el nuevo workflow que ahora falla el trabajo pruebas-funcionales:



Si accedemos al detalle del trabajo pruebas-unitarias, puede verse que ha fallado la prueba por no coincidir el título esperado.

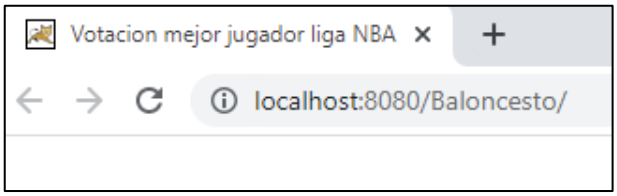


Como el error se ha generado después de instalar la aplicación en el servidor Tomcat de prueba del contenedor ubuntu-ci donde está github-actions, iniciamos el servidor Tomcat en nuestra máquina virtual, ya que Github Actions lo cierra una vez terminado el trabajo. Para activarlo usamos el comando:

```
$ sh /usr/local/tomcat/bin/catalina.sh start
```

```
root@c6900db474a2:/# sh /usr/local/tomcat/bin/catalina.sh start
Using CATALINA_BASE:   /usr/local/tomcat
Using CATALINA_HOME:   /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar
Using CATALINA_OPTS:
Tomcat started.
root@c6900db474a2:/#
```

Podemos abrirla con un navegador en http://localhost:8080/Baloncesto/ y comprobar en el título de la ventana que en efecto aparece "NBA":



6.3.2. Realizar una prueba sin errores

Para que no falle la prueba, corregimos el archivo index.html:

```
<title>Votacion mejor jugador liga NBA</title>
```

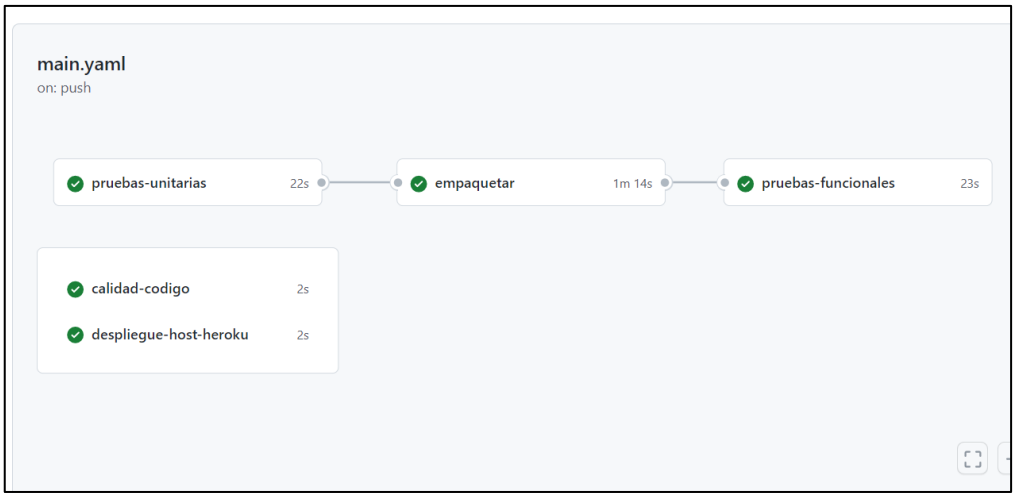
Cambiando NBA por ACB:

```
<title>Votacion mejor jugador liga ACB</title>
```

Guardamos el archivo y volvemos a subir al repositorio remoto:

```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Prueba funcional corregida"
C:\Baloncesto> git push
```

Puede comprobarse que este segundo workflow se realiza sin problemas:



Para poder usar el contenedor Ubuntu como servidor de pruebas, debemos configurar las variables de entorno de la base de datos ingresando los siguientes comandos dentro de la consola del contenedor, ya que Github Actions no almacena cambios en variables de entorno o comandos que se hagan por medio del workflow:

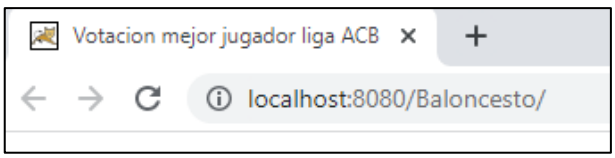
```
$ export DATABASE_HOST="jdbc:mysql://localhost"
$ export DATABASE_PORT="3306"
$ export DATABASE_NAME="baloncesto"
$ export DATABASE_USER="usuario"
$ export DATABASE_PASS="clave"
```

```
root@ab304ea19085:/# export DATABASE_HOST="jdbc:mysql://localhost"
root@ab304ea19085:/# export DATABASE_PORT="3306"
root@ab304ea19085:/# export DATABASE_NAME="baloncesto"
root@ab304ea19085:/# export DATABASE_USER="usuario"
root@ab304ea19085:/# export DATABASE_PASS="clave"
```

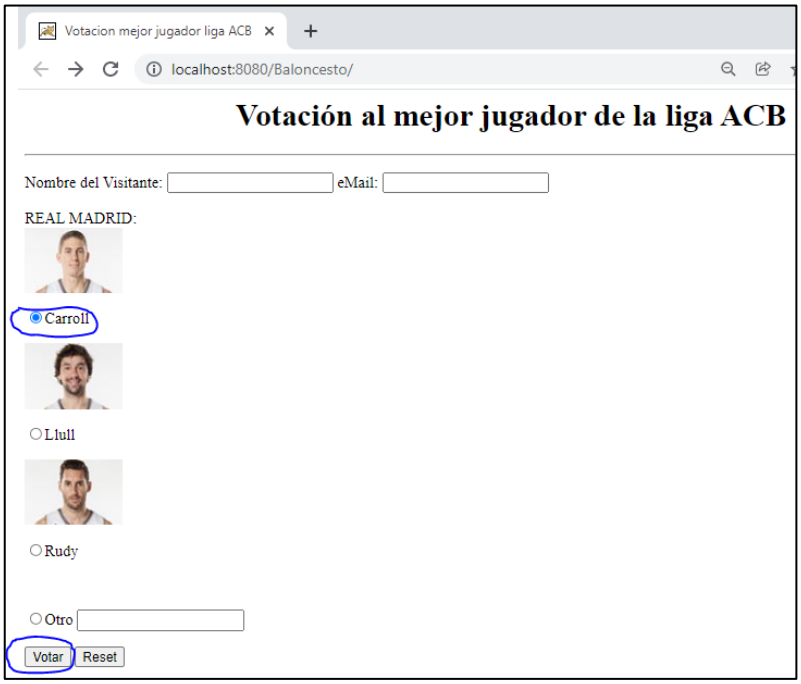
Como en el apartado anterior la aplicación queda instalada en el servidor Tomcat de pruebas del contenedor ubuntu-ci, debemos para y arrancar el servidor Tomcat para generar los cambios con el comando:

```
$ sh /usr/local/tomcat/bin/catalina.sh stop
$ sh /usr/local/tomcat/bin/catalina.sh start
```

Podemos abrir la aplicación con un navegador en http://localhost:8080/Baloncesto/ y comprobar en el título de la ventana que ya aparece el "ACB":



Podemos incluso utilizar la aplicación en el servidor de pruebas. En esta aplicación se puede votar a jugadores y las votaciones quedan registradas en una tabla "Jugadores" de la base de datos "baloncesto" en el servidor MySQL de pruebas también instalado en el contenedor ubuntu-ci. Por ejemplo, podemos votar al primer jugador:



Y comprobar que esa votación se ha registrado en la base de datos. Para lo cual, abrimos una consola del contenedor ubuntu-ci:

```
C:\Baloncesto> docker exec -it ubuntu-ci /bin/bash
```

Y ejecutamos en la consola de ubuntu-ci el comando mysql para abrir el cliente de la base de datos:

```
# mysql
mysql> use baloncesto
mysql> select * from Jugadores;
+----+---------+-------+
| id | nombre  | votos |
+----+---------+-------+
|  1 | Carroll |     1 |
|  2 | Llull   |     0 |
|  3 | Rudy    |     0 |
+----+---------+-------+
3 rows in set (0.00 sec)
```

Y vemos que el voto ha quedado registrado.

6.4. Fase qa: Trabajo calidad-codigo

Para poder realizar este trabajo, necesitamos instalar un servidor sonarqube en alguna máquina. Lo más sencillo en este caso es simularlo utilizando un contendor, a partir de la imagen oficial sonarqube disponible en docker hub: https://hub.docker.com/_/sonarqube

Para ello, debemos crear un contenedor que se llamará sonarqube con el comando:

```
C:\> docker run -d --name sonarqube -p 9000:9000 sonarqube
```

Podemos probar que el servidor sonar está arrancado, abriendo en un navegador la dirección:

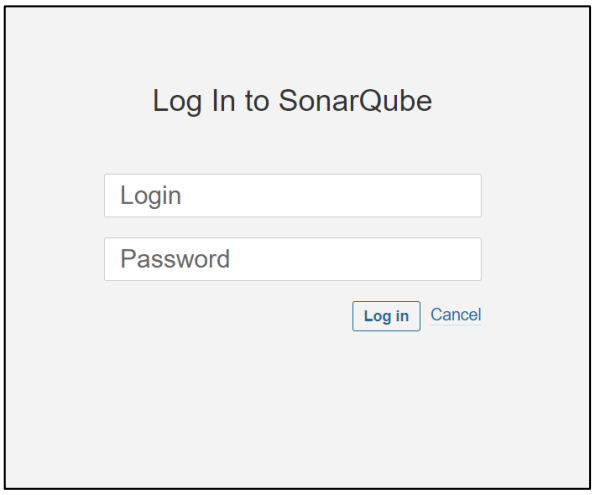
http://localhost:9000

Los datos de acceso por defecto son admin/admin.

Necesitamos que el contenedor ubuntu-ci se pueda conectar con este contenedor sonarqube, por lo que habrá que volver a crearlo, utilizando la opción link en el comando run de docker. Pero antes hay que parar y eliminar el contenedor anterior con el comando rm de docker.

```
C:\> docker stop ubuntu-ci
C:\> docker rm ubuntu-ci
C:\> docker run -it --name ubuntu-ci -p 8080:8080 -p 3306:3306 --link
sonarqube aihieuwi54/ubuntu-github-actions
```

Desde dentro del contenedor ubuntu-ci, la URL que hay que usar para acceder al servidor sonarqube, no es localhost, sino el nombre del contenedor: http://sonarqube:9000



Una vez hechos estos cambios, ya podemos definir el trabajo en el archivo main.yaml . Para evitar que errores en la calidad de código afecten nuestro despliegue a producción, ya agregar la directiva:

```
continue-on-error: true
```

Esto lo que hará es , continuar el siguiente trabajo a pasar de que el trabajo de calidad de código falle.

Archivo C:\Baloncesto\.github\workflows\main.yaml

```
calidad-codigo:
    runs-on: self-hosted
    needs: empaquetar
    continue-on-error: true
```

```
      steps:
        - name: qa
          run: |
            mvn sonar:sonar -Dsonar.host.url=http://sonarqube:9000
Dsonar.qualitygate.wait=true Dsonar.login=admin -Dsonar.password=admin
```

Hay que incluir la dependencia, porque SonarQube cuando detecta que hay archivos .java en un proyecto, exige que también estén los archivos .class, en otro caso se genera un error.

Cabe destacar que el comando:

```
mvn sonar:sonar -Dsonar.host.url=http://sonarqube:9000
Dsonar.qualitygate.wait=true Dsonar.login=admin -Dsonar.password=admin
```

Debe ir en una sola línea ya que, si lo ponemos por separado, durante el workflow se ejecutará como si fuese otro comando bash.

```
calidad-codigo:
  runs-on: self-hosted
  needs: empaquetar
  continue-on-error: true
  steps:
    - name: qa
      run: |
        mvn sonar:sonar -Dsonar.host.url=http://sonarqube:9000 -Dsonar.qualitygate.wait=true -Dsonar.login=admin -Dsonar.password=admin
```

En el comando mvn se han indicado como usuario y password "admin", si se han modificado en SonarQube hay que cambiarlos en el comando.

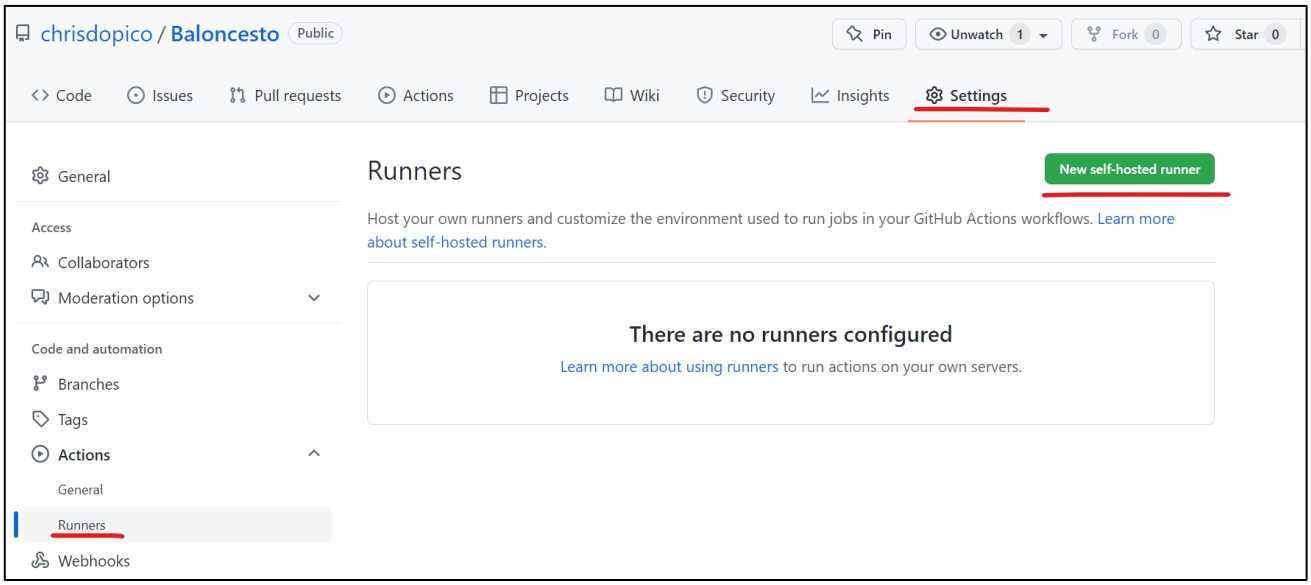
En el archivo pom.xml del proyecto hay que añadir el plugin de sonarqube.

Archivo C:\pom.xml

```
<plugin>
    <groupId>org.sonarsource.scanner.maven</groupId>
    <artifactId>sonar-maven-plugin</artifactId>
    <version>3.6.0.1398</version>
</plugin>
```

Una vez guardados los cambios hay que hacer push sobre el repositorio remoto. Pero antes hay que recordar volver a registrar y ejecutar github-actions en el contenedor ubuntu-ci, ya que este contenedor se ha creado de nuevo para conectarse con el contenedor sonarqube:

Dentro del repositorio de github nos dirigimos a la sección Settings>Actions>Runners y seleccionamos "New self-hosted runner"



Una vez seleccionado "New self-hosted runner", escogemos la opción Linux.

En la parte inferior, copiamos el token de nuestro repositorio y lo colocamos después de configurar nuestra variable de entorno:

```
$ RUNNER_ALLOW_RUNASROOT="1"
```

Por lo que, la configuración sería la siguiente:

Una vez ingresado el token de nuestro repositorio, procedemos a configurar las características de nuestro runner:

-Para el nombre de grupo, presionamos enter para seleccionar el nombre por defecto:

```
Enter the name of the runner group to add this runner to: [press Enter for
Default]
```

- Para el nombre del runner, escribiremos ubuntu:

```
Enter the name of runner: [press Enter for c6900db474a2] ubuntu
```

- Para el label, presionamos enter para evitar este paso:

```
This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]
```

- Finalmente, se selecciona el nombre de la carpeta de trabajo, para este caso presionaremos enter para seleccionar el nombre por defecto:

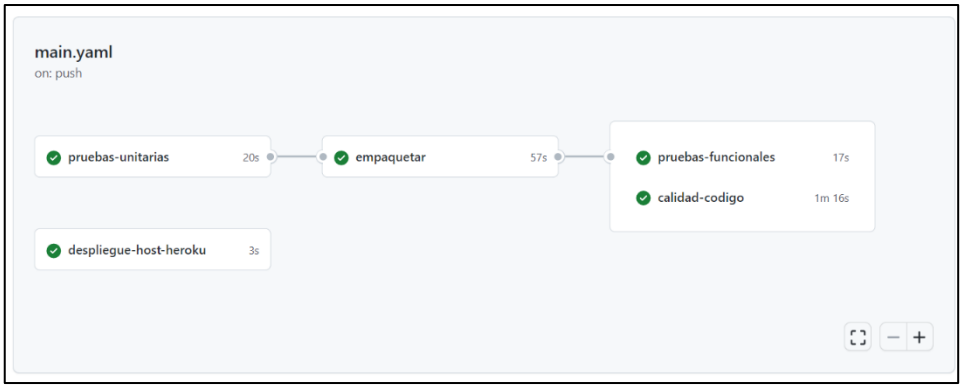
```
Enter name of work folder: [press Enter for _work]
```

```
√ Connected to GitHub

# Runner Registration

Enter the name of the runner group to add this runner to: [press Enter for Default]

Enter the name of runner: [press Enter for c6900db474a2] ubuntu

This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]

√ Runner successfully added
√ Runner connection is good

# Runner settings

Enter name of work folder: [press Enter for _work]

√ Settings Saved.

root@c6900db474a2:/#
```

Una vez hecho esto, podemos hacer push al repositorio remoto:

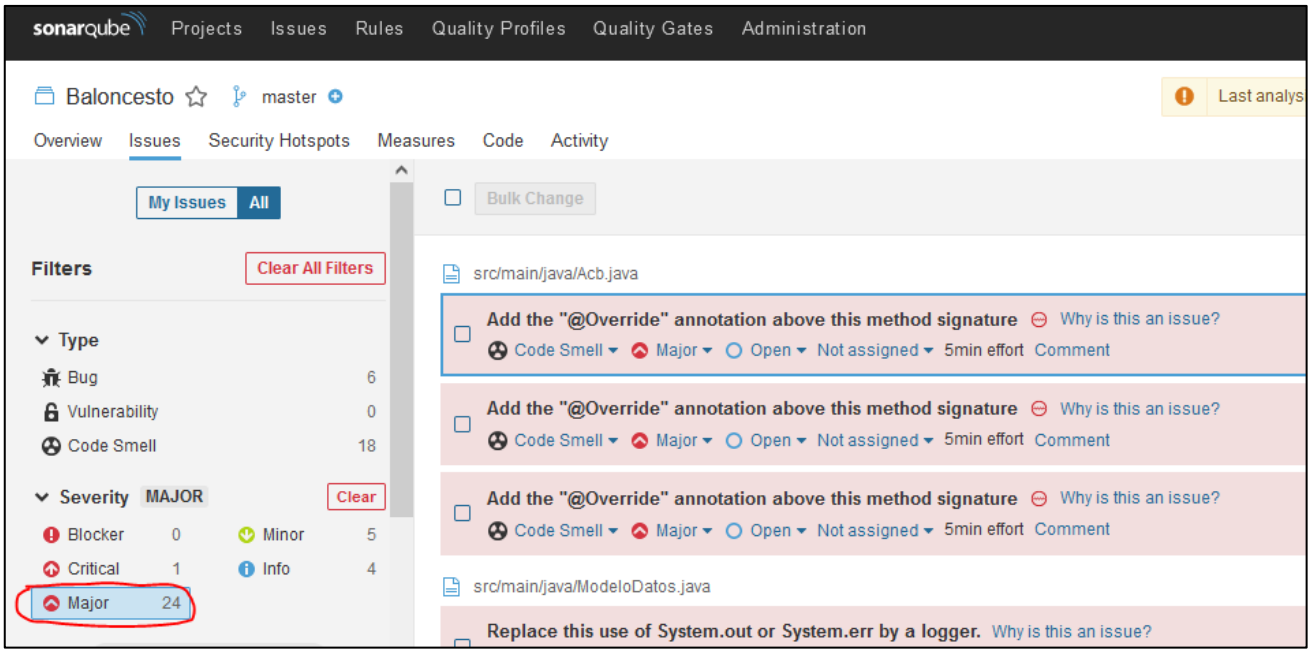
```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Creado trabajo calidad-codigo"
C:\Baloncesto> git push
```

Al ejecutarse el trabajo calidad-codigo, puede comprobarse que pasa sin problemas.



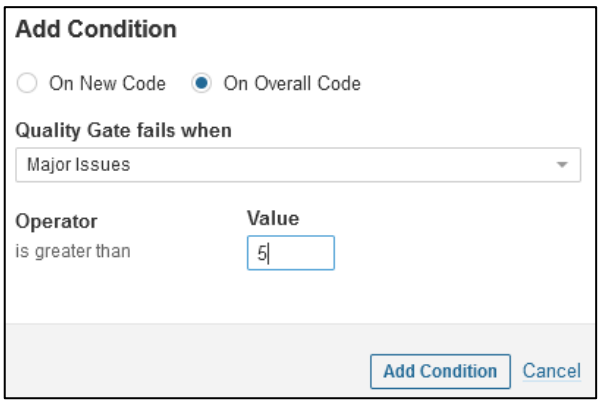
6.4.1. Aumentar la exigencia de calidad

Si accedemos al servidor sonarqube http://localhost:9000 (admin/admin), en la sección Projects > Baloncesto > Issues > Major podemos ver que hay 24 fallos de calidad importantes (major issues).

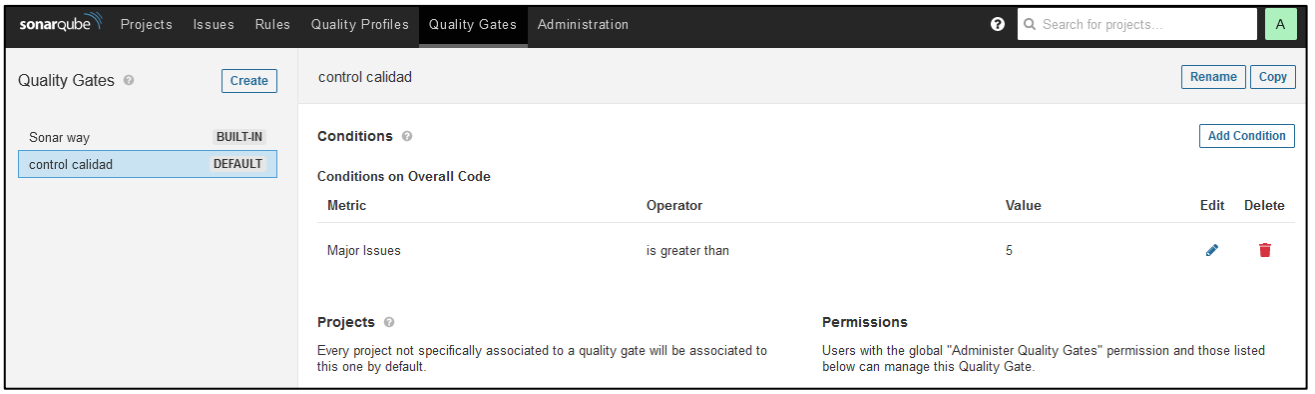


Podemos hacer que sea más exigente, y definir una regla para que no pase la prueba de calidad del código si el número de fallos de calidad importantes es mayor de una cantidad. Esto se hace en la sección:

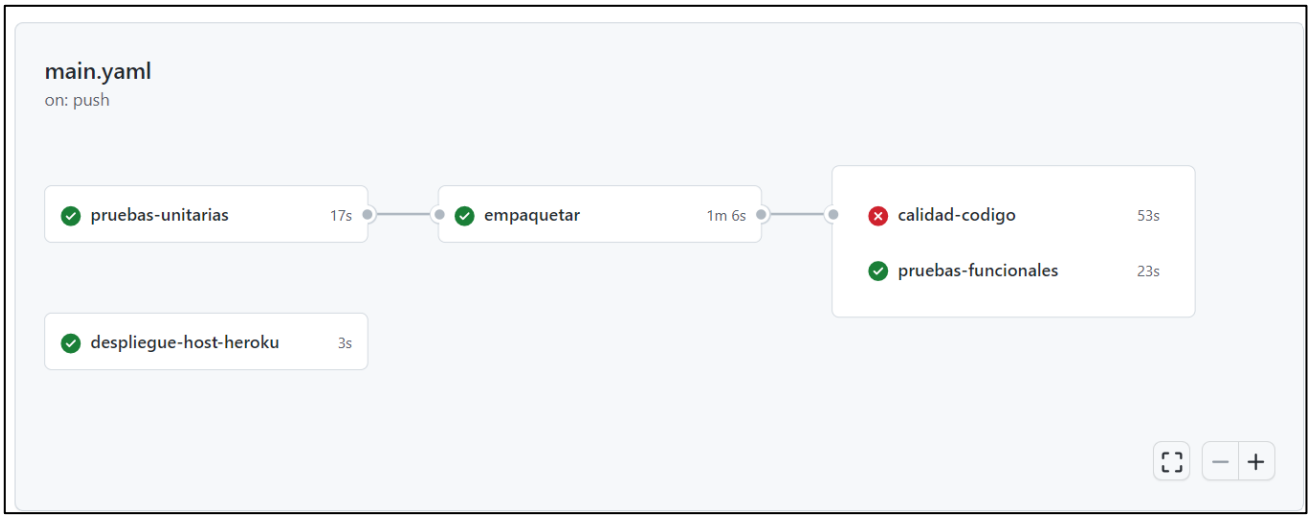
Quality Gates > Create > Name = "control calidad" > Add Condition > On Overall code > Quality Gate fails when > Major issues is greater than 5



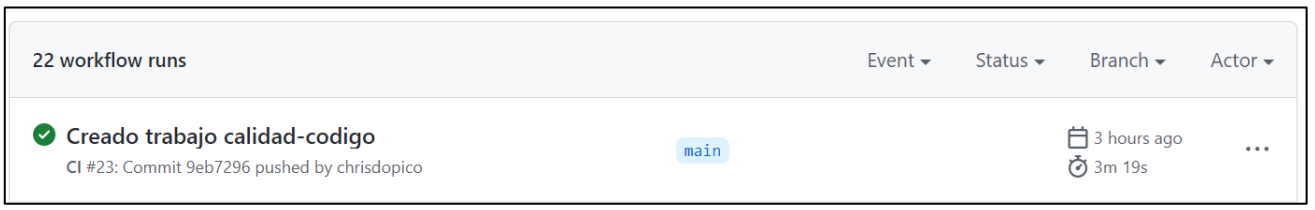
Después hay que entrar en la ventana de la condición "control calidad" y activar el botón "Set as default", para que se aplique a todos los proyectos.



Si volvemos al workflow, y seleccionamos que se repita (flechas en círculo junto al nombre del trabajo para refrescar) veremos que, aparece un error:



No se aborta el workflow porque en main.yaml hemos indicado `continue-on-error: true`.



Esto es habitual, que la calidad de código sea importante y se revise, pero que no bloquee el proceso de entrega de la aplicación, es decir, que se intente mejorar en siguientes versiones, pero que no detenga la integración o el despliegue de la aplicación.

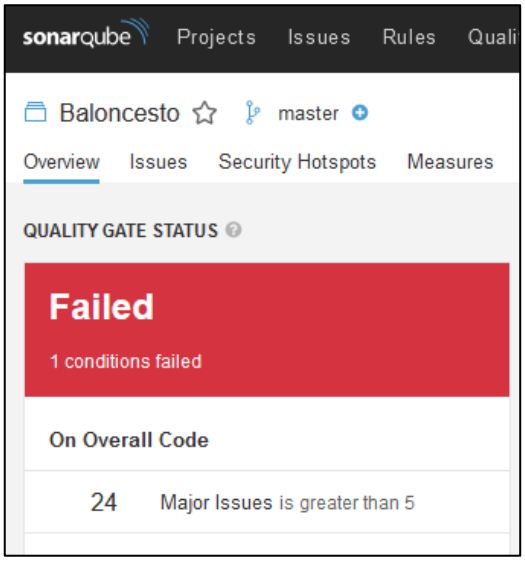
En la ventana del trabajo nos dice que ha habido un error:

```
124 [ERROR] Failed to execute goal org.sonarsource.scanner.maven:sonar-maven-plugin:3.6.0.1398:sonar (default-cli) on project
    Baloncesto: QUALITY GATE STATUS: FAILED - View details on http://sonarqube:9000/dashboard?id=IC%3ABaloncesto -> [Help 1]
125 [ERROR]
126 [ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
127 [ERROR] Re-run Maven using the -X switch to enable full debug logging.
128 [ERROR]
129 [ERROR] For more information about the errors and possible solutions, please read the following articles:
130 [ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException
131 Error: Process completed with exit code 1.
```

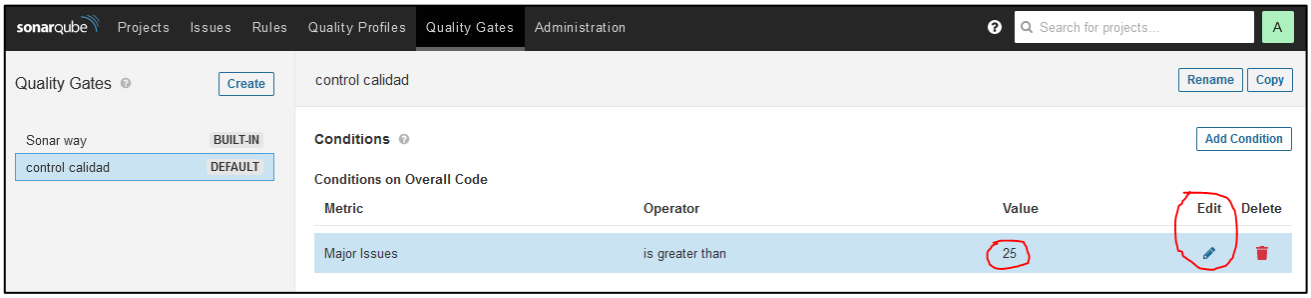
Y que el detalle está en:

`http://sonarqube:9000/dashboard?id=IC%3ABaloncesto`

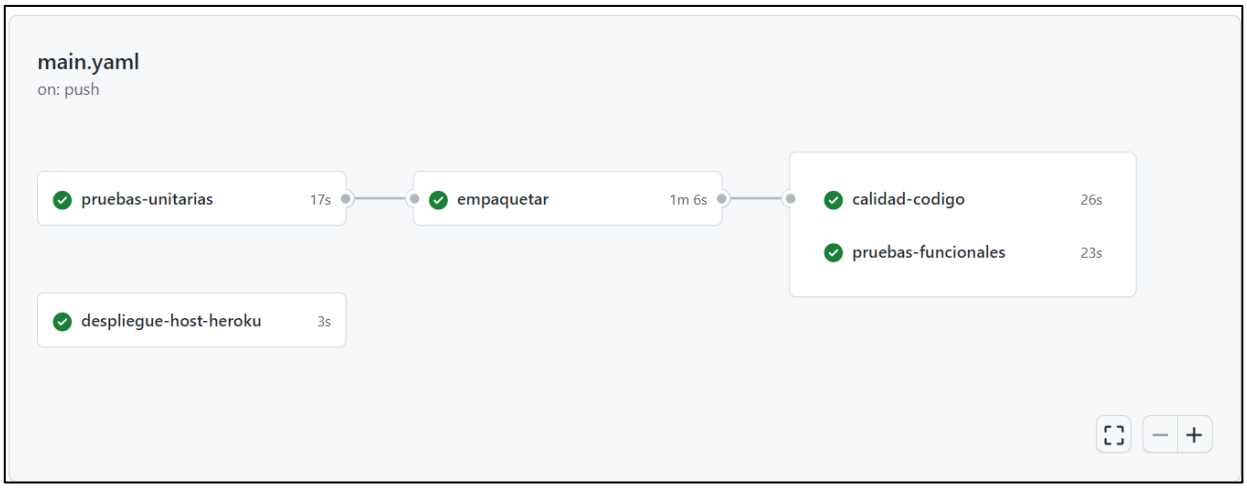
Como se ha dicho antes, esa URL es interna en el contenedor ubuntu-ci. Desde fuera del contenedor sería `http://localhost:9000/dashboard?id=IC%3ABaloncesto`. Si accedemos con usuario y contraseña, vemos que en efecto ha habido 24 errores de calidad importantes, que es mayor que el límite de 5 que habíamos definido.



Si en el servidor sonarqube aumentamos el número de fallos de calidad importantes permitidos a 25 en la condición de la Quality Gate que habíamos creado llamada "control calidad":



Y refrescamos de nuevo el trabajo calidad-código en el workflow, deja de aparecer el aviso de no superación del trabajo ya que el nivel de exigencia ahora es menor, ya que permitimos hasta 25 fallos de calidad importantes.



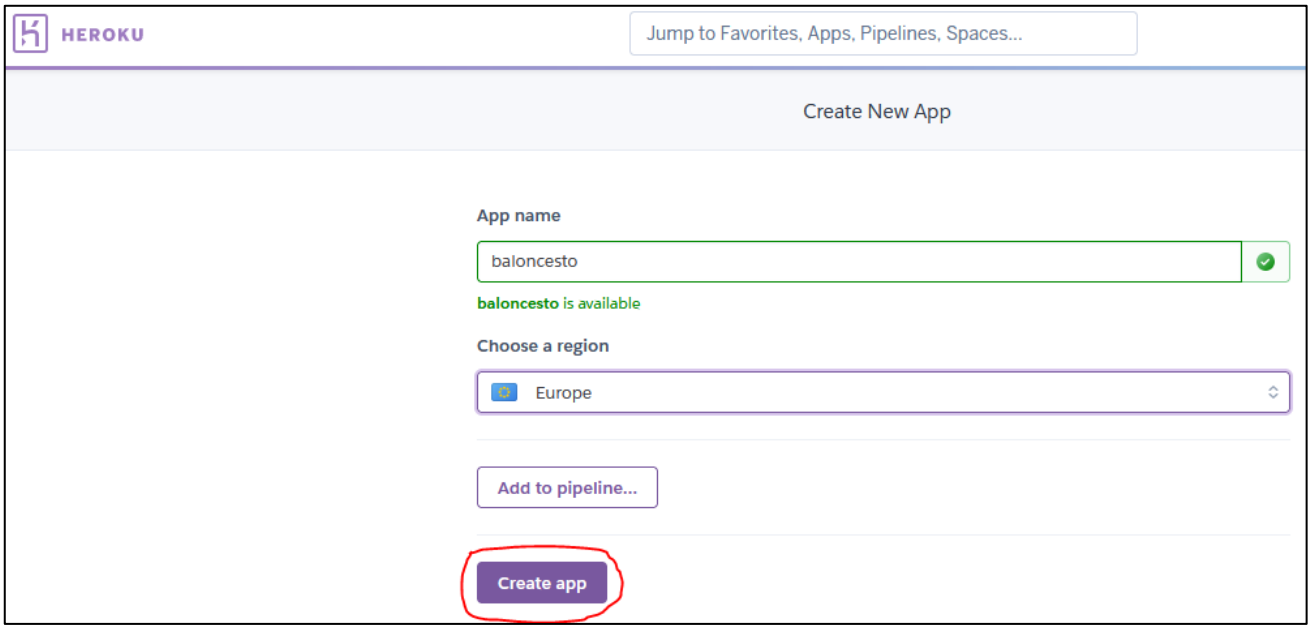
6.5. Fase deploy: Trabajo despliegue-host-heroku

Se utilizará el servicio de hosting gratuito Heroku, que permite instalar hasta 3 aplicaciones de forma gratuita.

Una vez identificados, hay que crear una aplicación vacía en:

https://dashboard.heroku.com/apps

Seleccionando New > Create new app. Con el nombre baloncesto.



En el archivo main.yaml añadimos los comandos para el despliegue usando maven.

Archivo C:\.github-ci.yml

```
    despliegue-heroku:
      runs-on: self-hosted
      needs: pruebas-funcionales
      if: github.ref == 'refs/heads/main'
      steps:
        - name: deploy
          run: |
            export HEROKU_API_KEY=${{secrets.HEROKU_API_KEY}}
            mvn     heroku:deploy-war     -Dheroku.appName=baloncesto     -
DskipTests=true
```

Desplegaremos en producción una vez se hayan aprobado el trabajo de las pruebas funcionales, para así evitar errores e2e (end to end) en nuestra aplicación.

Agregamos el condicional para establecer que solo se despliegue en producción solo cuando se haga push en la rama main.

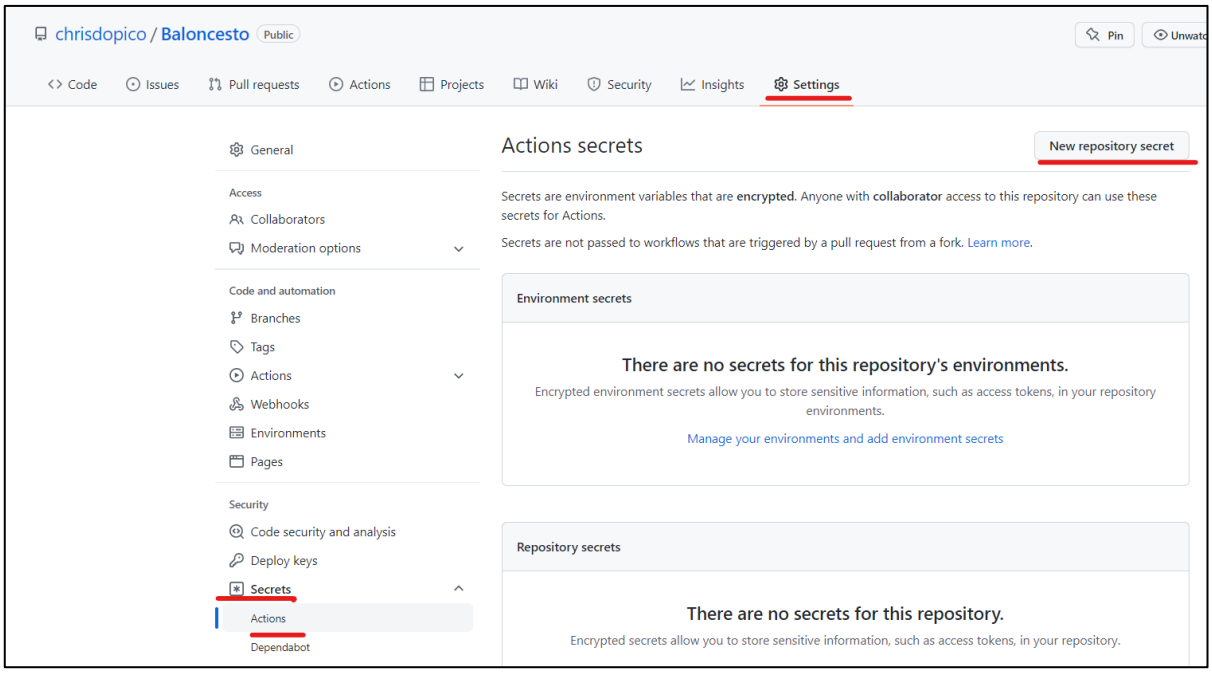
```
if: github.ref == 'refs/heads/main'
```

Para desplegar en Heroku debemos utilizar una clave (API Key) que podemos encontrar en los ajustes de nuestra cuenta de Heroku:

Account settings > API Key > Reveal

Para evitar que se vea esa clave, en el archivo main.yaml hacemos referencia a una variable de github.com, que permite crear variables cuyo valor es oculto.

Para crear esa variable, en github debemos acceder a Settings > Secrets > Actions > New repository secret.



Y creamos una con el nombre HEROKU_API_KEY y como valor el de la API Key que tenemos en Heroku.

Pulsamos "Add variable" y aparece en la lista de variables del proyecto:

En el archivo main.yaml podemos hacer referencia a las variables con el símbolo ${{secrets.variable}}.

Con el primer comando se crea en el contenedor ubuntu-ci la variable de entorno que necesita Heroku para autorizar el despliegue.

```
export HEROKU_API_KEY=${{secrets.HEROKU_API_KEY}}
```

Con el segundo comando se hace el despliegue, siguiendo las indicaciones de https://devcenter.heroku.com/articles/deploying-java-applications-with-the-heroku-maven-plugin.

Para que funcione, hay que añadir dos plugins al archivo pom.xml del proyecto:

Archivo C:\pom.xml

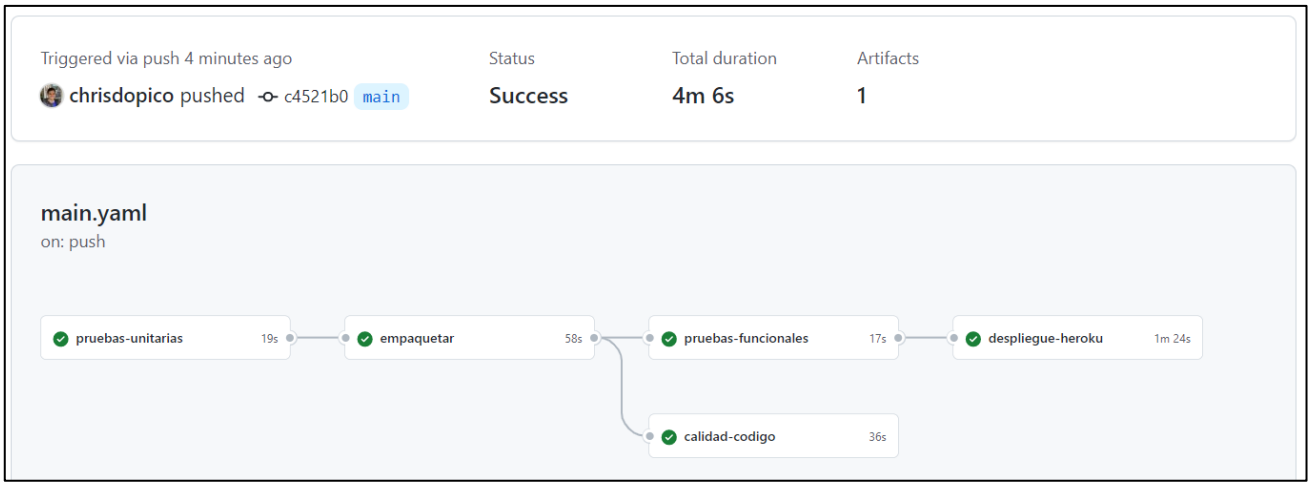
```
<plugin>
 <groupId>com.heroku.sdk</groupId>
 <artifactId>heroku-maven-plugin</artifactId>
```

```
    <version>3.0.4</version>
   </plugin>

   <plugin>
    <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-dependency-plugin</artifactId>
      <executions>
       <execution>
        <phase>package</phase>
         <goals>
          <goal>copy</goal>
         </goals>
         <configuration>
          <artifactItems>
           <artifactItem>
            <groupId>com.heroku</groupId>
            <artifactId>webapp-runner</artifactId>
            <version>9.0.30.0</version>
            <destFileName>webapp-runner.jar</destFileName>
           </artifactItem>
          </artifactItems>
         </configuration>
        </execution>
       </executions>
      </plugin>
```

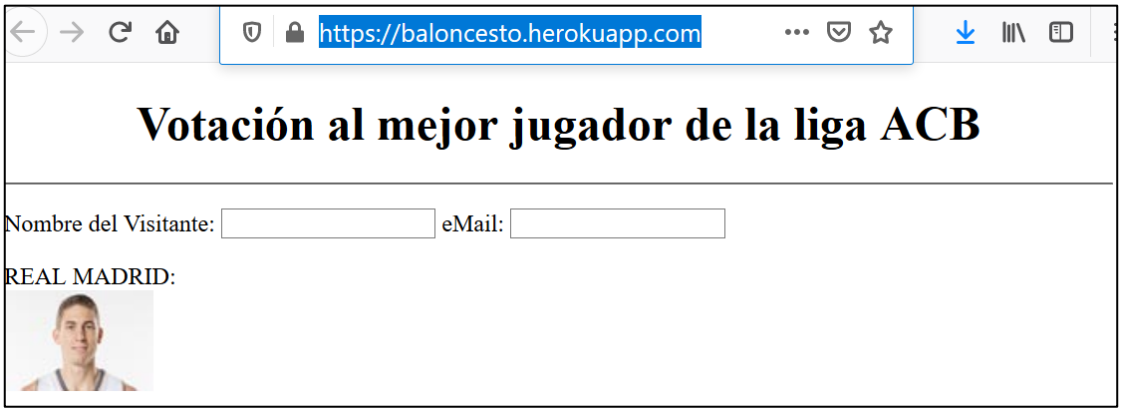
Se guarda en el repositorio local y se hace push al remoto, y se comprueba que se despliega correctamente:

```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Creado trabajo despliegue-host-heroku"
C:\Baloncesto> git push
```

Si todo ha ido bien, aparece que el trabajo ha sido un éxito.



Si abrimos la aplicación en Heroku en la URL que corresponda a nuestra aplicación, en este ejemplo en https://baloncesto.herokuapp.com/, vemos que funciona:



NOTA: Para simplificar la práctica no se ha instalado en Heroku la base de datos. Se sugiere que el alumno que esté interesado en utilizar en el futuro Heroku como hosting gratuito, investigue cómo instalar una base de datos MySQL en Heroku.

7. REALIZAR UNA SEGUNDA VERSIÓN DE LA APLICACIÓN

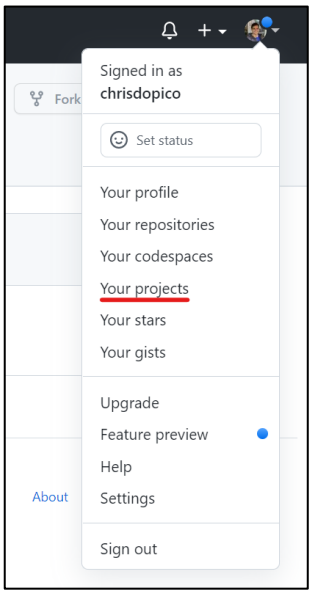
7.1. Planificación agil: crear historias de usuario (issues)

Se puede gestionar un proyecto ágil basado en Scrum con Github, teniendo en cuenta que la terminología no coincide exactamente con la que se maneja en Scrum. En esta tabla se muestra la equivalencia:

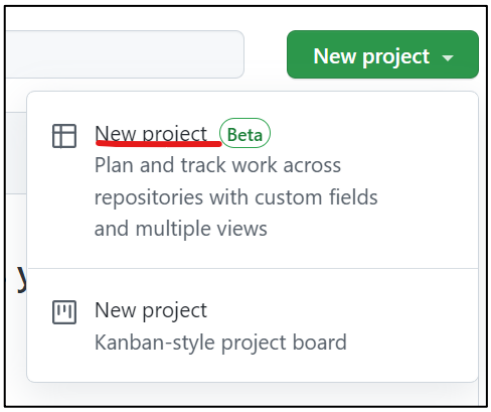
Agile (Scrum)	**Github**
User story	Issue
Sprint	Milestone
Product backlog	Issue list
Burndown chart	Burndown chart
Board	Board

7.1.1. Creación de proyecto

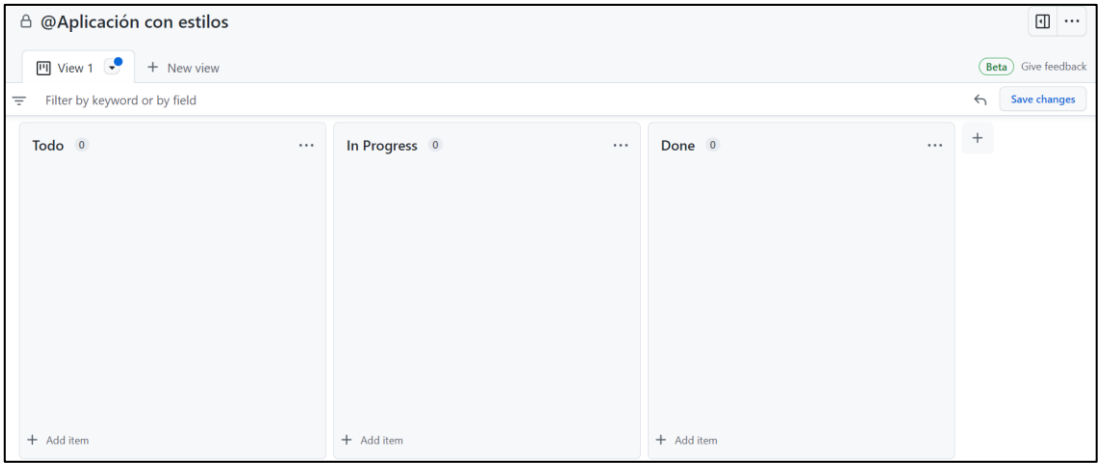
Para esto, debemos crear un proyecto en github en Menu>Your projects>New project.



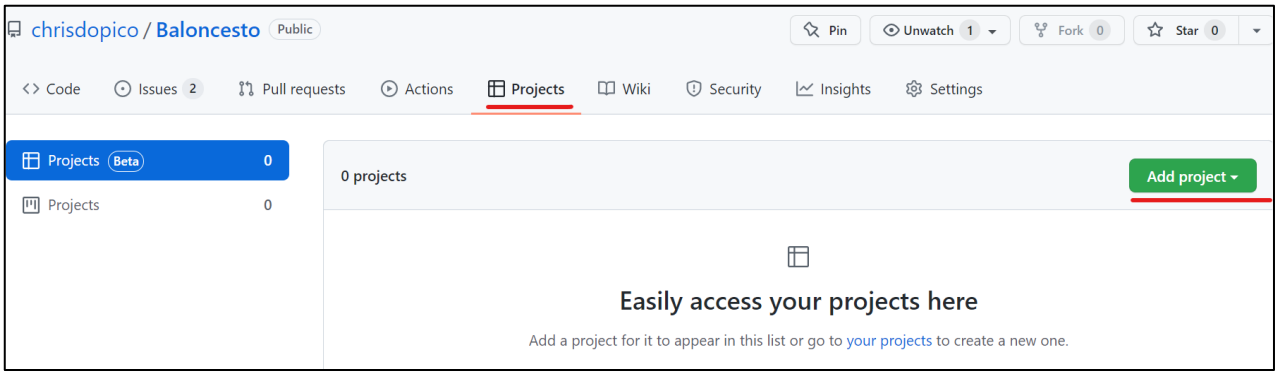
Escogemos la opción New project.



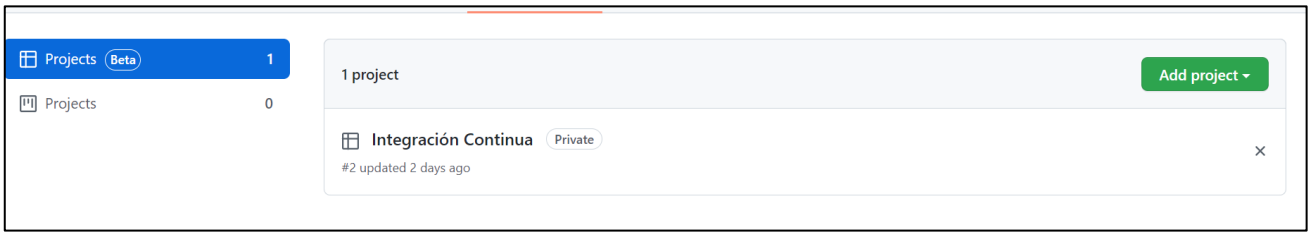
Nos queda la siguiente forma de tablero.



Ahora debemos asignar nuestro nuevo proyecto al repositorio Baloncesto. Para esto, nos dirigimos Baloncesto>Projects y seleccionamos Add project.

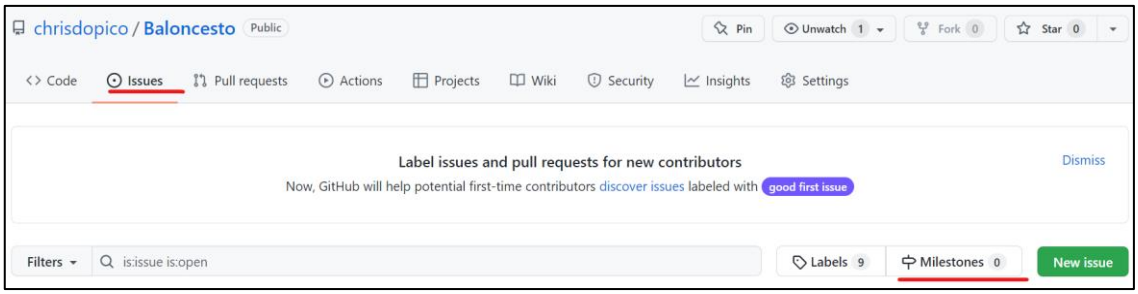


Buscamos y seleccionamos nuestro proyecto "Integración Continua".

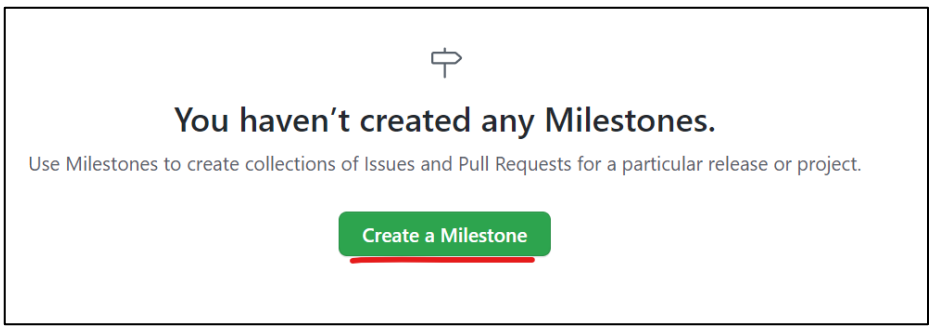


7.1.2. Creación de milestone

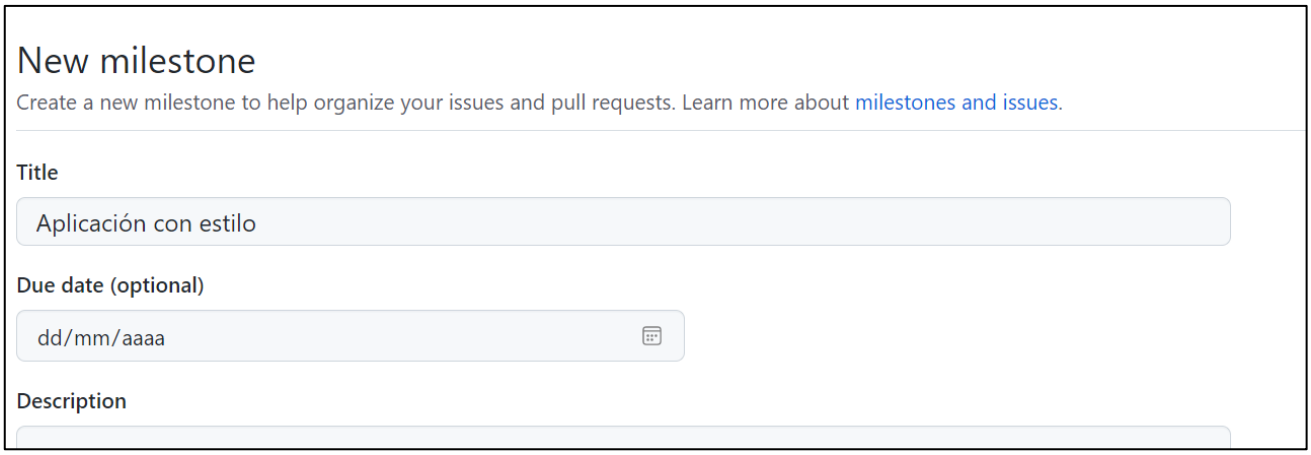
Para crear un milestone, nos dirigimos al repositorio Baloncesto>Issues>Milestones.



Seleccionamos Create a Milestone.

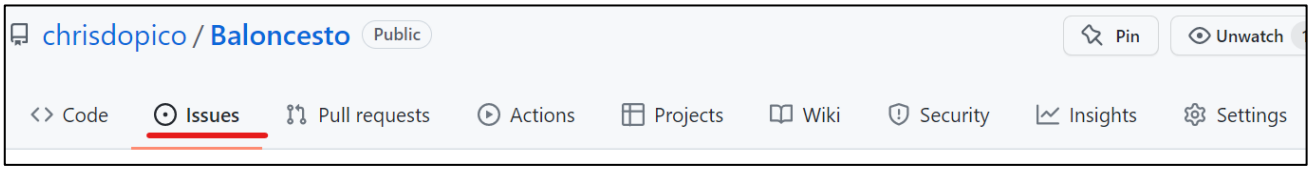


El nombre de nuestro Milestone será "Aplicación con estilo" y seleccionamos crear. Podemos seleccionar una fecha, pero para este caso, la dejaremos en blanco.



7.1.3. Creación de issues

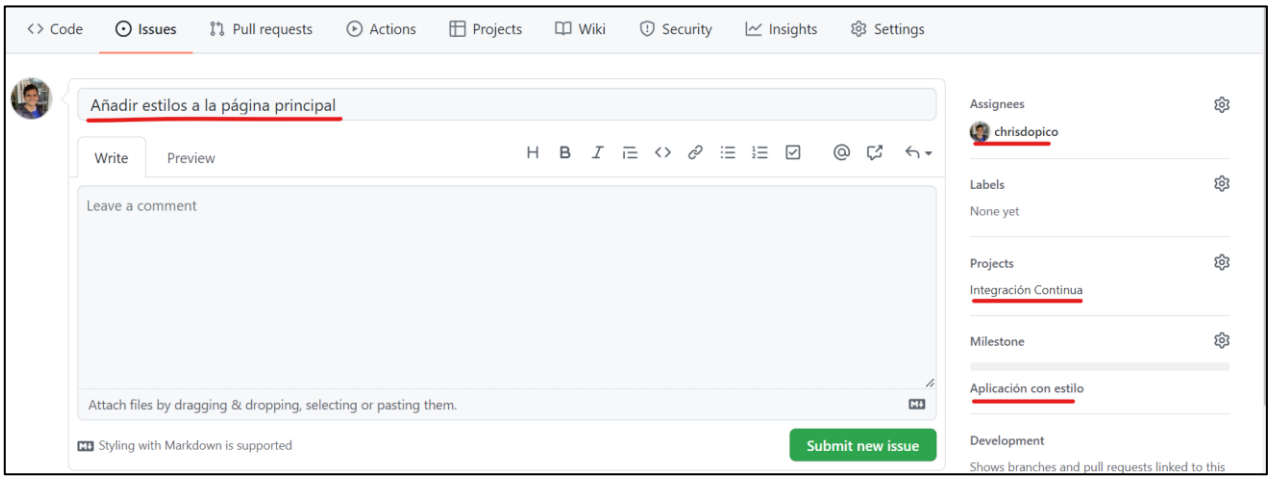
Para crear los issues, damos clic en el menú issues y seleccionamos New issue.



Los issues son los siguientes:

- Añadir estilos a la página principal: Se trata de poner color de fondo azul y color de texto blanco en la página index.html.
- Añadir estilos a la página de resultado de votación: Se trata de poner color de fondo rojo y color de texto amarillo en la página TablaVotos.jsp.

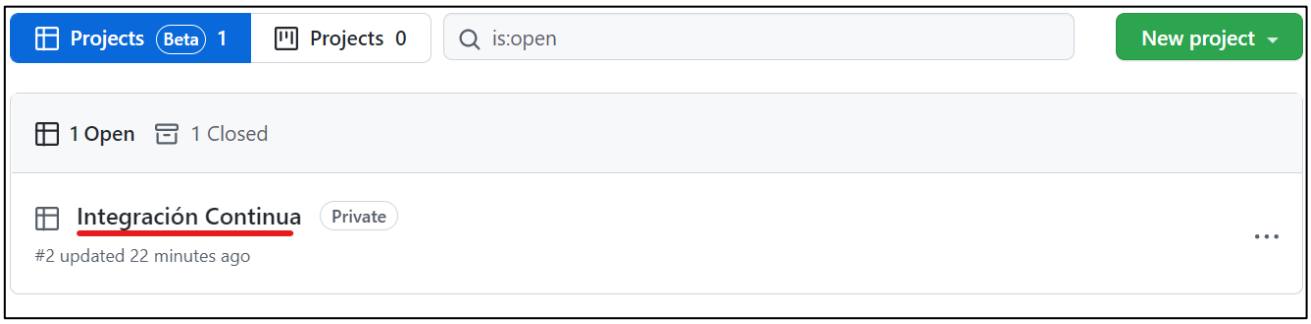
Escribimos el título de la nueva issue, su desarrollador asignado, milestone y proyecto.



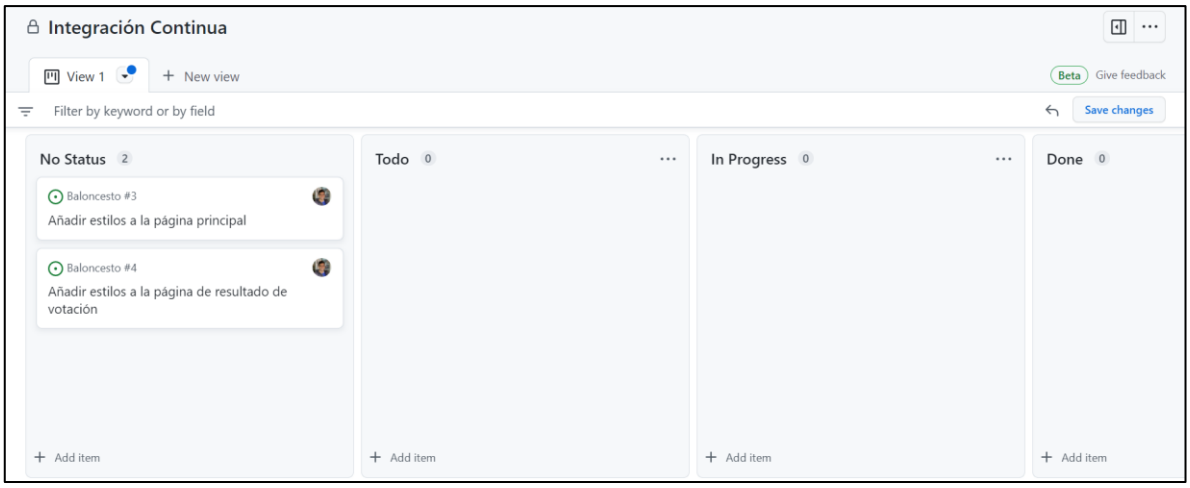
7.1.4. Visualización del backlog

En la página de Issues está la lista de issues del proyecto. Si aplicamos Scrum, representaría el product backlog o pila de producto, siendo los issues las historias de usuario. En este caso como todas las historias son del mismo sprint, representaría realmente la pila del sprint.

Para ver el tablero Kanban, en la sección de Baloncesto>Projects y seleccionamos el proyecto "Integración Continua".



Y cambiamos la vista a modo tablero.



Vamos a simular ahora cómo trabajarían dos desarrolladores diferentes con el mismo repositorio compartido. Al haber sólo un desarrollador, lo que se hará a continuación puede parecer que no tiene mucho sentido. Si se prefiere, el alumno puede crear otro usuario de Github con otro email e invitarle a ser miembro del proyecto desde la opción: Balocesto>Settings> Collaborators, asignándole el rol Developer. Y asignar uno de los issues al nuevo miembro, desde la ficha del issue, sección Asignees.

Si suponemos que solo hay un desarrollador, será quien se encargue de los dos issues, pero los va a desarrollar simultáneamente.

7.2. Crear una rama con git para cada issue (feature)

Cada historia de usuario o issue implica añadir una nueva característica o feature al código de la aplicación que se está desarrollando. Existen varias políticas de gestión de ramas. En este proyecto seguiremos la más sencilla, que consiste y es crear una rama partiendo de la rama main para cada nueva feature, y cuando esté terminada, integrar (merge) los cambios de esa rama en la rama main.

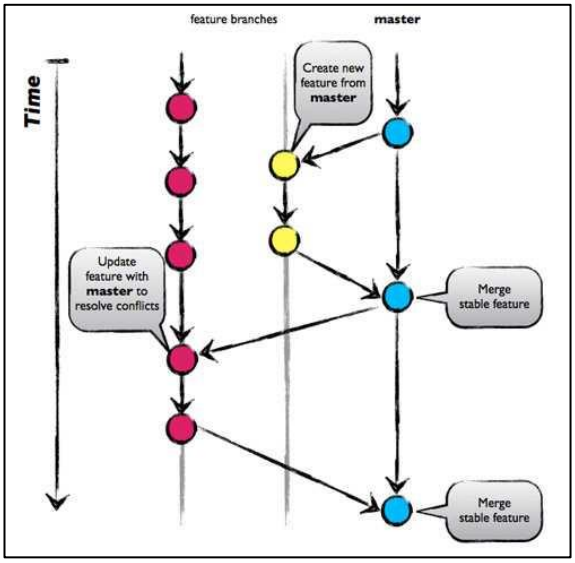


Imagen: nicoespeon

Hay que trabajar en las dos features de los dos issues, que llamaremos:

- añadir-estilos-pagina-principal
- añadir-estilos-pagina-resultado

Desde nuestro ordenador local como desarrollador encargado de la primera feature, crearemos una rama para la misma con el comando siguiente:

```
C:\Baloncesto> git branch añadir-estilos-pagina-principal
```

Pero como también vamos a desarrollar la otra issue creamos la otra rama:

```
C:\Baloncesto> git branch añadir-estilos-pagina-resultado
```

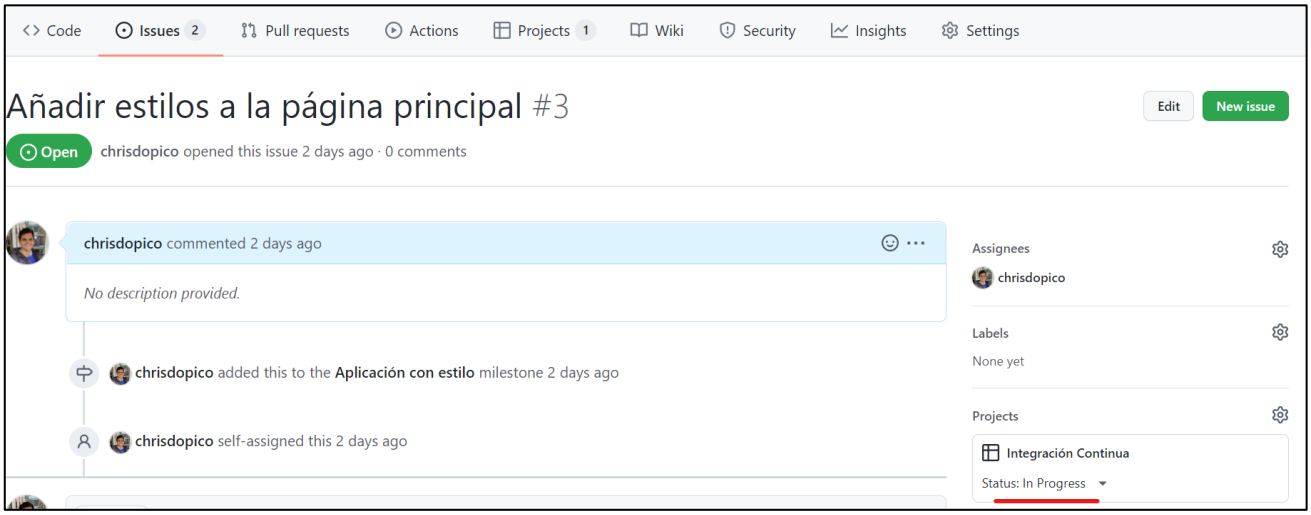
Tenemos por tanto tres ramas. Con el comando git branch puede comprobarse, indicando con * en la que nos encontramos trabajando:

```
C:\Baloncesto> git branch
  añadir-estilos-pagina-principal
  añadir-estilos-pagina-resultado
* main
```

Si usamos Visual Studio Code como editor, podemos instalar la extensión Git Graph para ver las ramas: https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph

7.2.1. Programar issue "Añadir estilo a la página principal"

Primero nos dirigimos a Issues y escogemos el issue correspondiente. En la parte derecha podemos cambiar su estado a "In Progress".

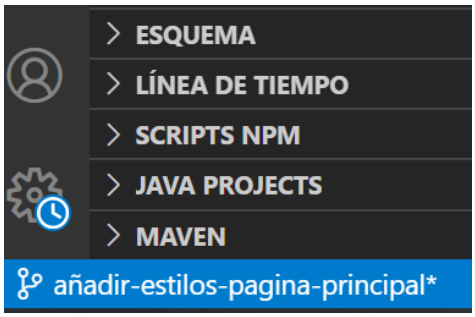


Vamos a pasarnos a la primera rama creada para que el desarrollador trabaje en la nueva característica (feature) de añadir estilos a la página principal. Para lo cual hay que cambiarse a dicha rama con el comando:

```
C:\Baloncesto> git checkout añadir-estilos-pagina-principal
Switched to branch 'añadir-estilos-pagina-principal'
```

Para saber en qué rama estamos podemos ejecutar git branch, se muestra con * la rama actual.

Si trabajamos con Visual Studio Code, en la parte inferior izquierda siempre se indica la rama en la que trabajamos, en este caso estamos en añadir-estilos-pagina-principal:



Estando en esa rama, vamos a crear un archivo `estilos.css` en la carpeta `C:\Baloncesto\src\main\webapp\`.

Archivo C:\Baloncesto\src\main\webapp\estilos.css

```
body {
    color: white;
    background-color: darkblue;
}
```

Y modificamos el archivo `index.html` de la misma carpeta, haciendo referencia al archivo de estilos desde la sección `<head>`, para que la página principal tenga fondo azul y texto blanco.

Archivo C:\Baloncesto\src\main\webapp\index.html

```
<head>
    <title>Votacion mejor jugador liga ACB</title>
    <link href="estilos.css" rel="stylesheet" type="text/css" />
</head>
```

Con esto terminan los cambios en la rama añadir-estilos-pagina-principal y se deben consolidar en el repositorio local antes de subirlos al repositorio remoto.

```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Añadido estilo en página principal"
```

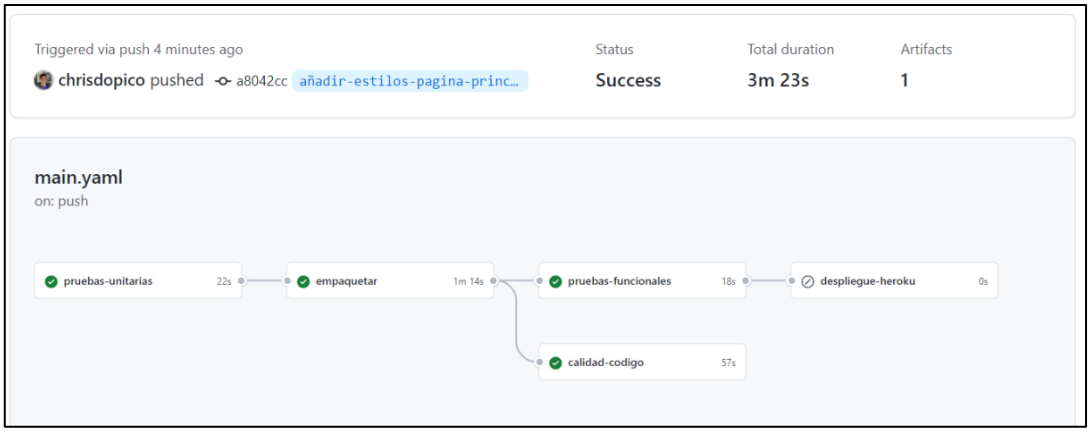
Cuando se suban los cambios al repositorio compartido en Github, se lanzarán las pruebas y si todo ha ido bien, se podrán integrar los cambios en la rama main del repositorio compartido.

Para subir los cambios locales al repositorio compartido ejecutamos:

```
C:\Baloncesto> git push origin añadir-estilos-pagina-principal
```

En el servidor de integración continua Github se crea la rama añadir-estilos-pagina-principal con el código fuente modificado, y se lanza la ejecución para esa rama de los trabajos de todas las fases del workflow del proyecto menos deploy, ya que en el archivo main.yaml se indicaba que el trabajo de la fase deploy llamado despliegue-host-heroku sólo se tenía que ejecutar cuando hubiera cambios en la rama main.

Se puede consultar el resultado en la sección Actions > All workflows

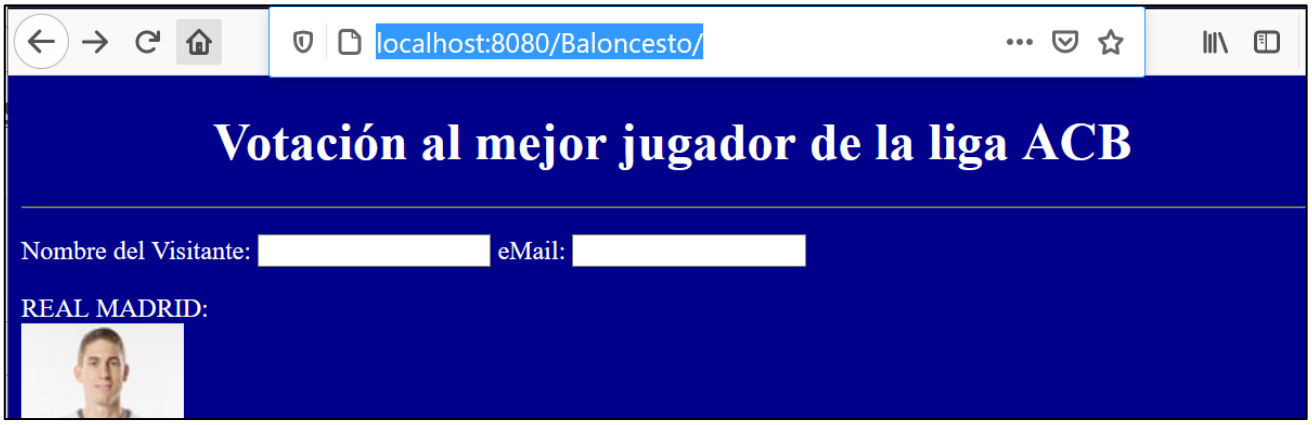


Como en el trabajo pruebas-funcionales de la fase Test se despliega aplicación modificada en el servidor web Tomcat del contenedor ubuntu-ci que se utiliza para las pruebas, podemos comprobar que ha

cambiado el color de la página principal. Para esto iniciamos el servidor tomcat en nuestra máquina virtual de Linux.

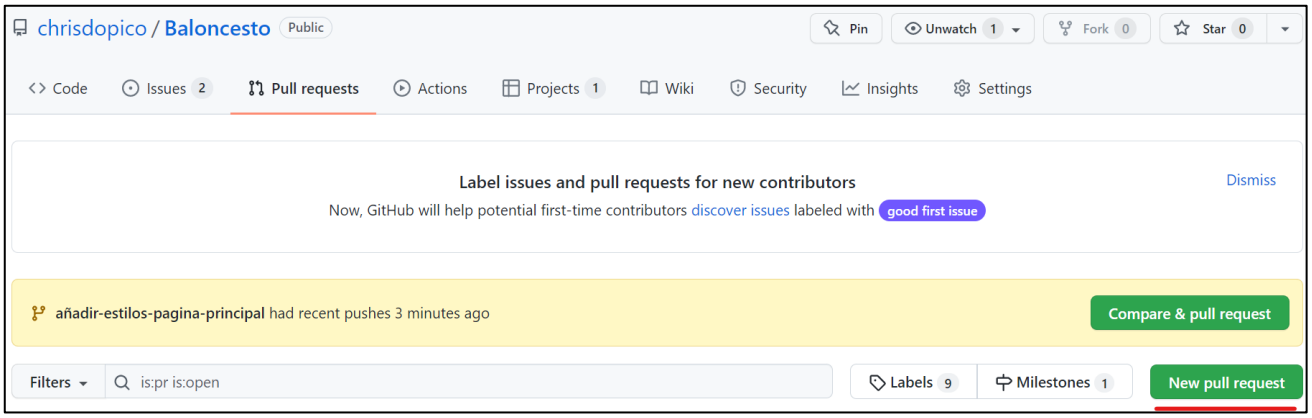
```
$ sh /usr/local/tomcat/bin/catalina.sh start
```

Accedemos a la aplicación en el servidor web de pruebas: http://localhost:8080/Baloncesto/.

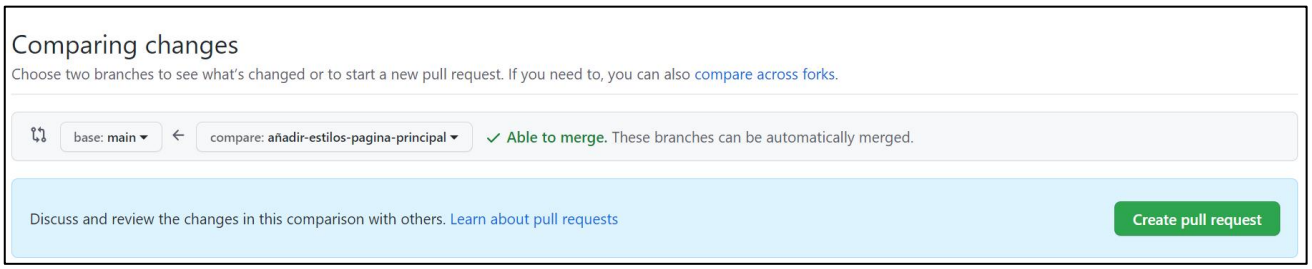


Una vez pasado el workflow con éxito, se pueden integrar los cambios en la rama main.

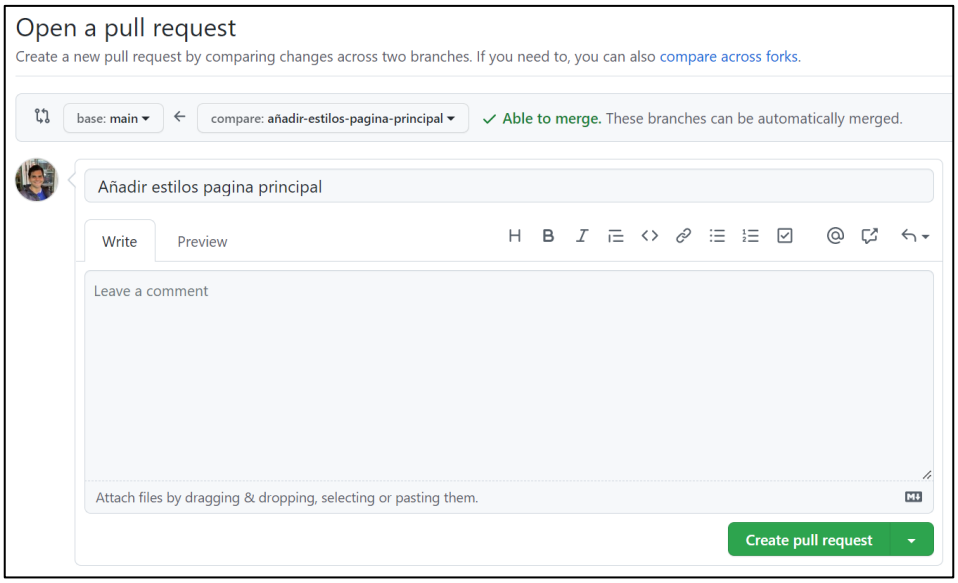
Para ello hay que crear una solicitud de integración (pull request) desde la sección del menú del repositorio Baloncesto llamada Pull request y seleccionamos el botón New pull request.



Se trata de hacer una integración desde la rama añadir-estilos-pagina-principal a la rama main. Se selecciona como rama base la rama main, y la rama a comparar la rama "Añadir-estilos-pagina-principal".

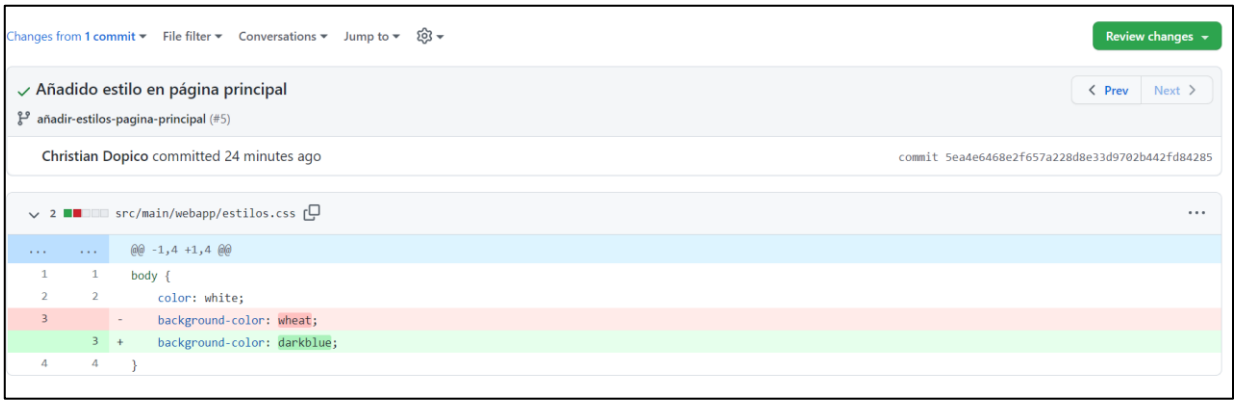


Seleccionamos Create pull request.



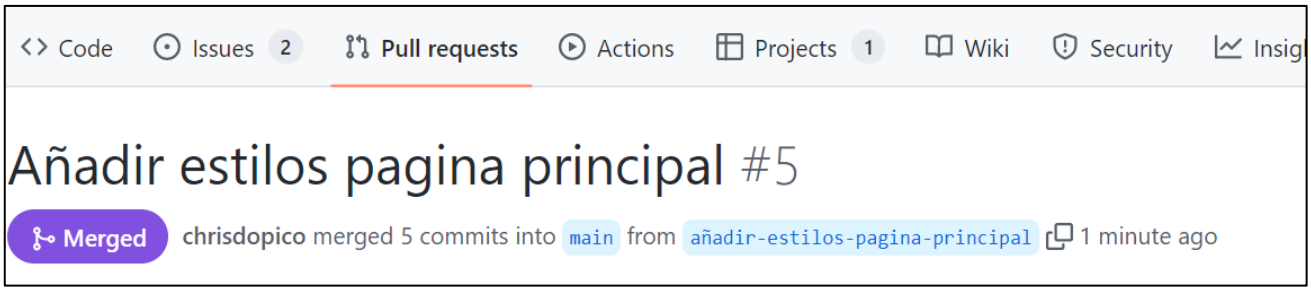
Una vez creada, en la sección Pull Request, podremos ver todas las solicitudes que tenemos abiertas.

El miembro del equipo que esté designado para aceptar los cambios (por ejemplo el product owner o quien haya decidido el equipo) puede seleccionar el pull request e ir a la opción "Files changed" para revisar los cambios en el código.

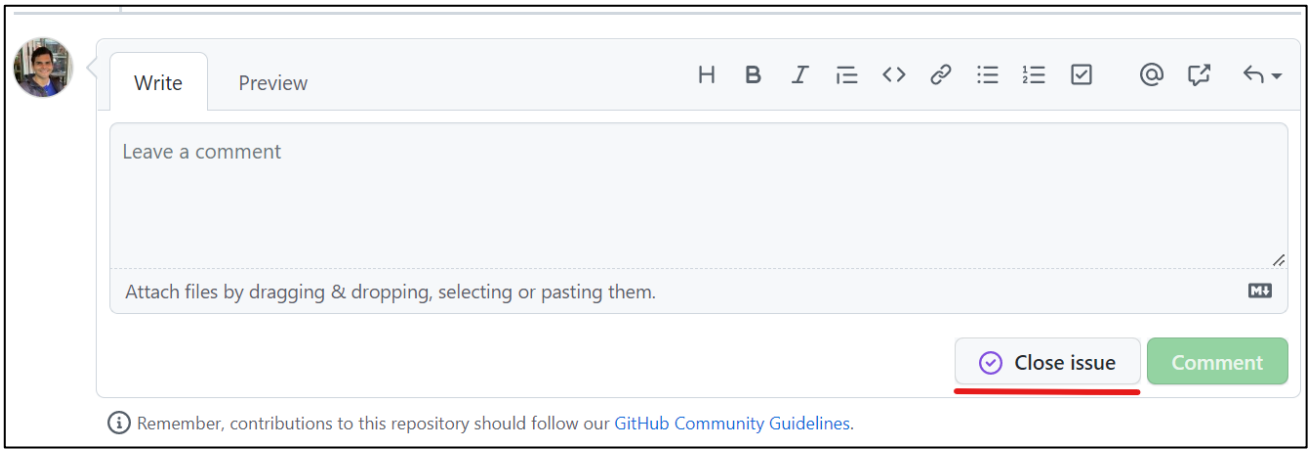


El responsable puede editar los archivos si decide que alguno de los cambios no es aceptable, o bien aceptarlos y proceder a fusionarlos en la rama main, pulsando el botón Create pull request.

Una vez que se ha creado la pull request, damos clic sobre ella y procedemos a hacer el merge, lo que procederá a cerrar nuestra pull request.

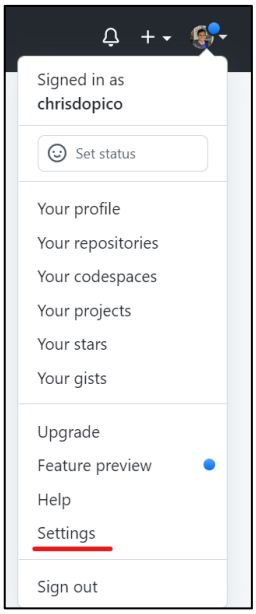


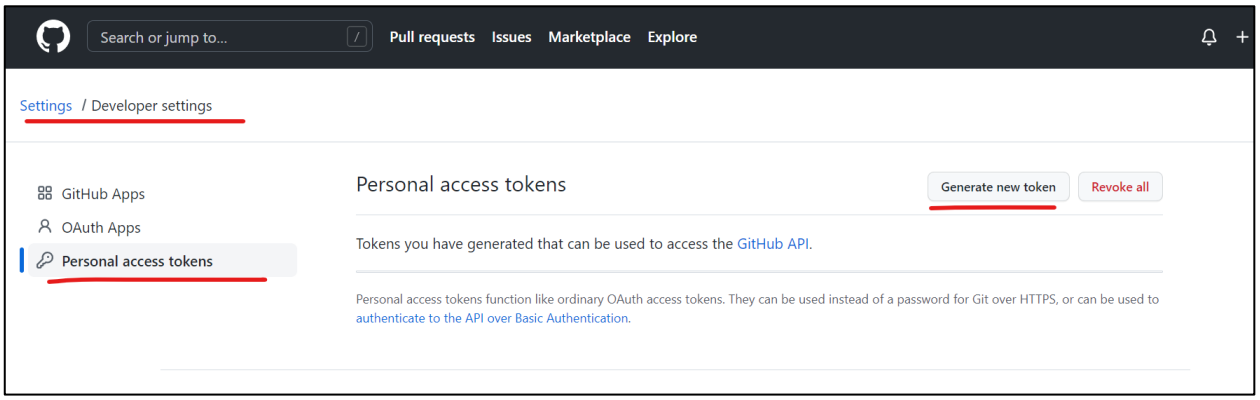
Una vez que el merge se haya cerrado, nos dirigimos a la sección issues, y procedemos a cerrar el issue "Añadir estilos página principal".



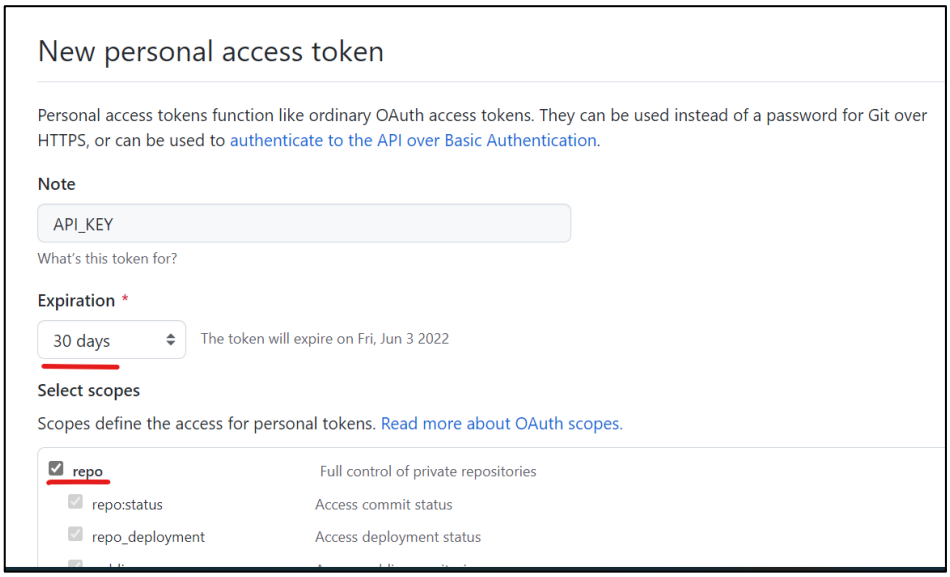
Una herramienta que se usa en metodologías ágiles es el Burndown Char, el cual nos indica en un diagrama bidimensional la relación entre la creación y/o cierre de issues en función del tiempo. Github no tiene esta herramienta integrada, sin embargo, nosotros usaremos una aplicación web externa para generarlo GitHub Burndown / Tom MacWright / Observable (observablehq.com).

Para esto, necesitamos generar un Access Token para nuestra cuenta github. Primero nos dirigimos a Settings>DeveloperSettings>Personal access tokens.



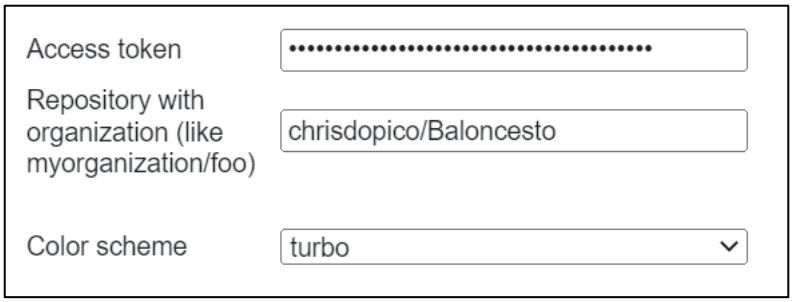


Escogemos el nombre de nuestro API key y escogemos la opción de "repo". Ahora seleccionamos el botón "Generate new token".

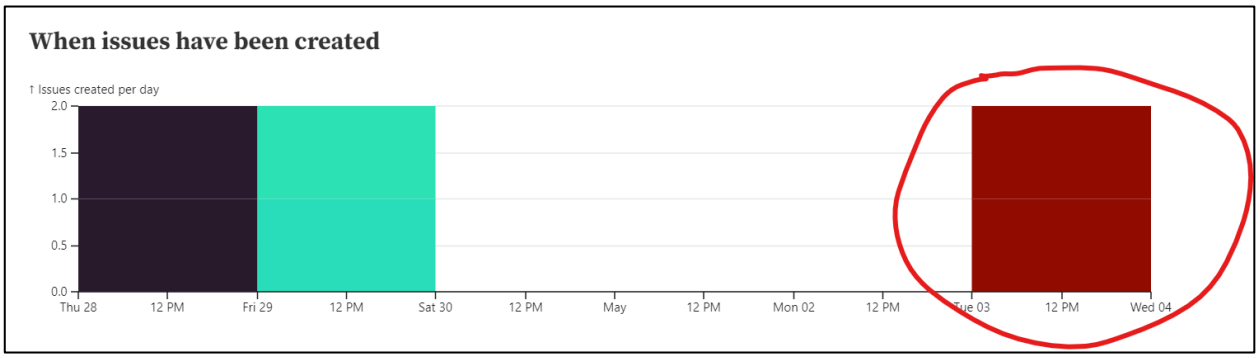


Ahora copiamos el token en un block de notas. Cabe recalcar que no debemos compartir este token.

Nos dirigimos a [GitHub Burndown / Tom MacWright / Observable (observablehq.com)] pegamos el token, escribimos nuestro nombre de usuario y el nombre del repositorio.



Nos dirigimos a los gráficos y podremos observar la siguiente imagen.



Podemos observar en la imagen en rojo, que se han creado dos issues en nuestro repositorio.

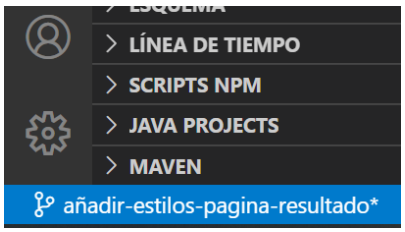
7.2.2. Programar issue "Añadir estilo a la página de resultado de votación" y resolver conflicto

La otra historia de usuario (issue) para el sprint (milestone) "Aplicación con estilos" era la titulada "Añadir estilos a la página de resultado de votación", para la cual ya se había creado una rama llamada añadir-estilos-pagina-resultado.

Para trabajar en esa rama, hay que ejecutar el comando:

```
C:\Baloncesto> git checkout añadir-estilos-pagina-resultado
Switched to branch 'añadir-estilos-pagina-resultado'
```

Si se trabaja con Visual Studio Code, puede verificarse en la parte inferior izquierda, que estamos en esa rama:



También puede comprobarse que en esa rama no existe el archivo estilos.css, porque ese archivo se creó en la otra rama (añadir-estilos-pagina-principal).

En este caso el desarrollador encargado de añadir estilos en la tabla de resultado decide crear también un archivo estilos.css:

Archivo C:\Baloncesto\src\main\webapp\estilos.css

```
body {
    color: yellow;
    background-color: darkred;
}
```

Y añadir la referencia a dicho archivo en la página de resultado de la votación TabaVotos.jsp.

Archivo C:\Baloncesto\src\main\webapp\TablaVotos.jsp

```
<head>
    <title>Votaci&oacute;n mejor jugador liga ACB</title>
    <link href="estilos.css" rel="stylesheet" type="text/css" />
</head>
```

Con esto terminan los cambios en la rama añadir-estilos-pagina-resultado y se deben consolidar en el repositorio local antes de subirlos al repositorio remoto.

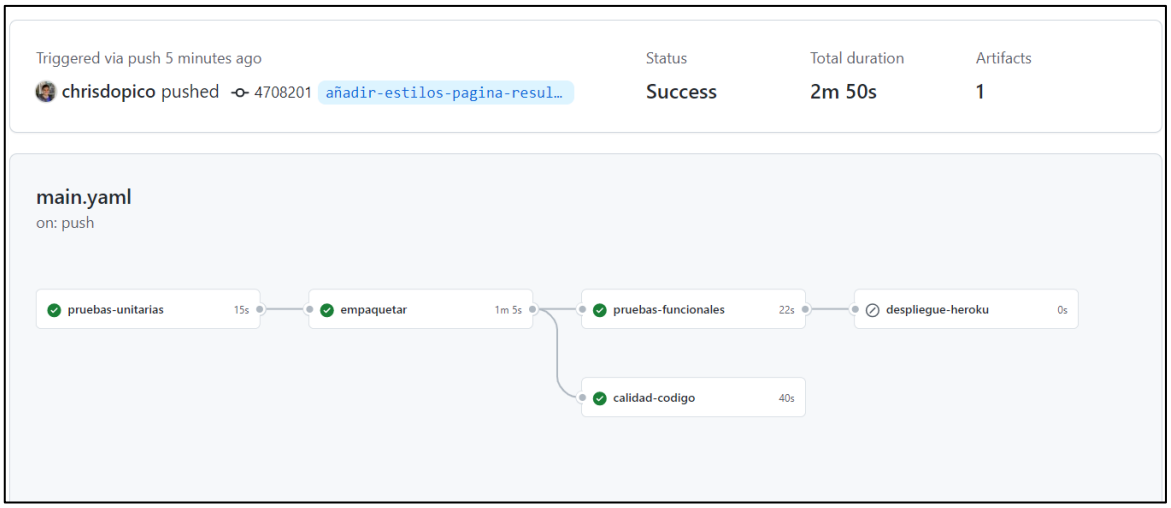
```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Añadido estilo en página de resultado"
```

Una vez hecho, desde el ordenador local se envían los cambios a esa rama del repositorio compartido en Github:

```
C:\Baloncesto> git push origin añadir-estilos-pagina-resultado
```

Se ha creado la rama añadir-estilos-pagina-resultado en el repositorio compartido y se ha lanzado para esa rama la ejecución de los trabajos del workflow a excepción del despliegue a heroku.

Si todo ha ido bien, se indica en Actions > Workflow:

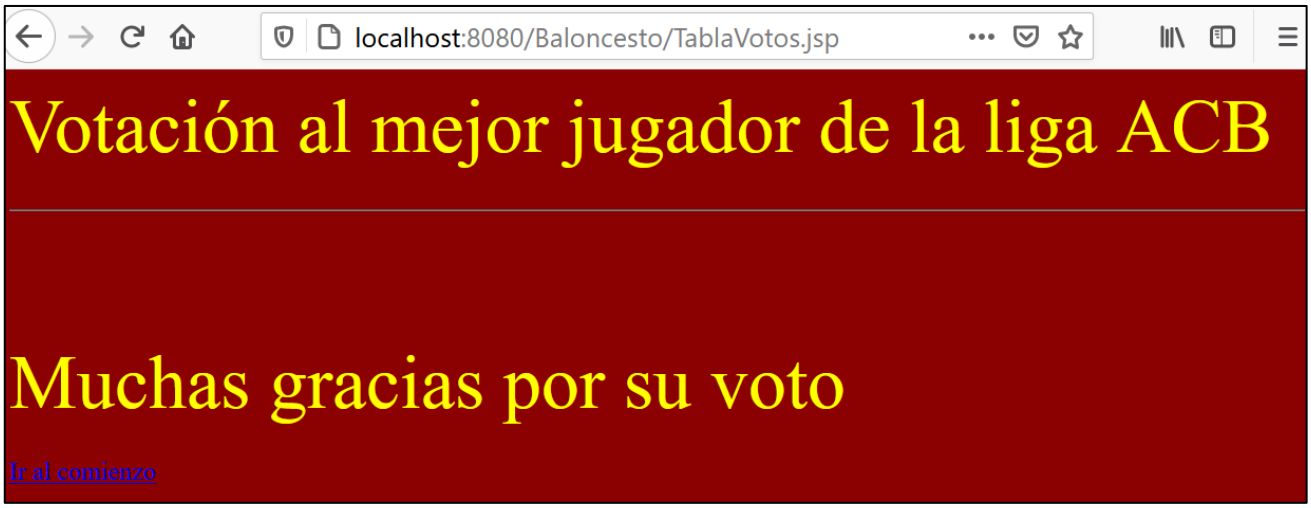


Se puede comprobar en el servidor web de pruebas que el estilo de la página de resultado de votación es el correcto iniciando nuestro servidor Tomcat.

```
$ sh /usr/local/tomcat/bin/catalina.sh start
```

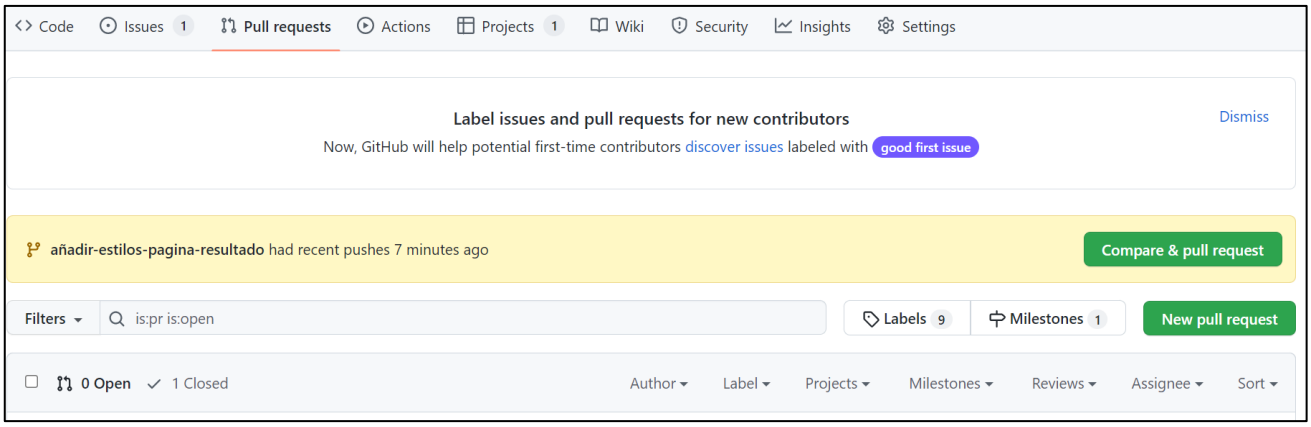
Y nos dirigimos a la página.

http://localhost:8080/Baloncesto/TablaVotos.jsp

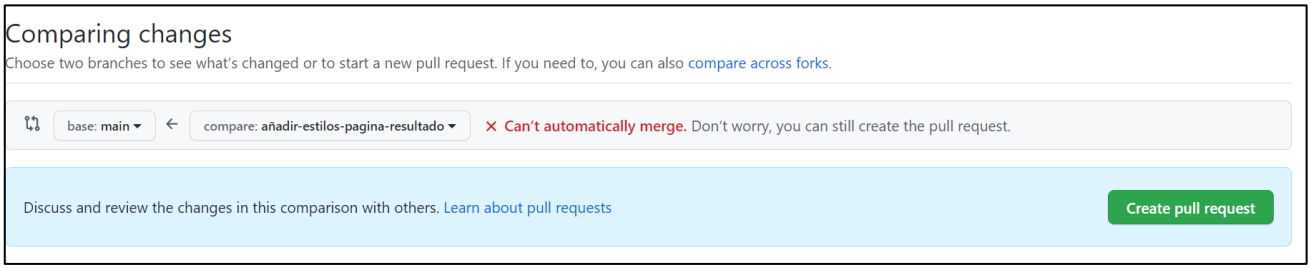


Una vez pasado el workflow con éxito, se pueden integrar los cambios en la rama main.

Para ello hay que crear una solicitud de integración (pull request) desde la sección del menú del repositorio Baloncesto>Pull request y seleccionamos el botón New pull request.



Seleccionamos las ramas correspondientes, sin embargo, nos aparece el siguiente error.



Lo que ha ocurrido es que el desarrollador encargado de este segundo issue ha creado un archivo estilos.css definiendo un estilo para body. Pero el desarrollador del issue del apartado anterior también creó ese archivo con un estilo diferente para body. Este archivo ya está integrado en la rama main, pues se hizo en el apartado anterior.

Es el desarrollador del nuevo issue el que debe hacer algo para solucionar el problema y poder integrar sus cambios.

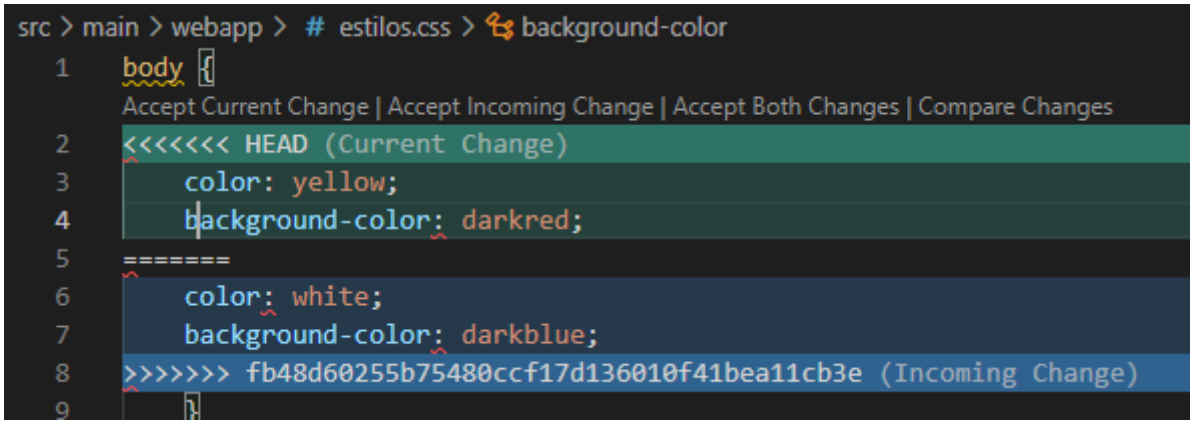
Lo fácil actualizar nuestro repositorio local usando el comando pull, que combina en un solo comando fetch y merge, de la siguiente forma:

```
C:\Baloncesto> git pull origin master
```

```
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 660 bytes | 110.00 KiB/s, done.
From https://github.com/chrisdopico/Baloncesto
 * branch            main       -> FETCH_HEAD
   c4521b0..f1fe6f4  main       -> origin/main
CONFLICT (add/add): Merge conflict in src/main/webapp/estilos.css
Auto-merging src/main/webapp/estilos.css
Auto-merging .github/workflows/main.yaml
Automatic merge failed; fix conflicts and then commit the result.
```

Nos avisa de que ha habido problemas. Si abrimos el archivo problemático, podemos ver los conflictos y borrar manualmente la sección que decidamos que hay que cambiar.

En el caso del archivo estilos.css, si lo abrimos con Visual Studio Code, nos ofrece aceptar el cambio entrante:



Realmente nos interesan ambos estilos, pero hay que cambiar el nombre del nuevo estilo para que no haya conflictos. Aceptamos ambos cambios y cambiamos el nombre del estilo de texto amarillo y fondo rojo.

Por lo que el archivo final sería:

Archivo C:\Baloncesto\src\main\webapp\estilos.css

```
body {
    color: white;
    background-color: darkblue;
}

.resultado {
    color: yellow;
    background-color: darkred;
}
```

Ahora hay que modificar el archivo TablaVotos.jsp para vincular el nuevo estilo "resultado" a la etiqueta body:

Archivo C:\Baloncesto\src\main\webapp\TablaVotos.jsp

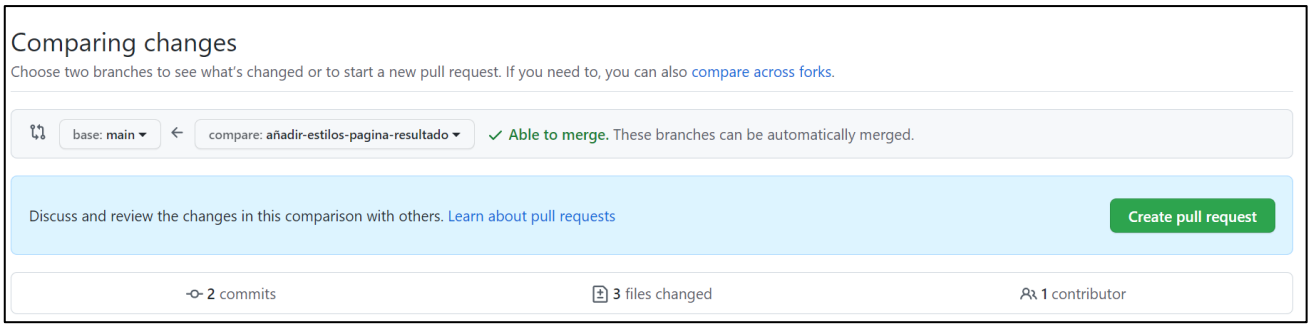
```
<html>

<head>
    <title>Votaci&oacute;n mejor jugador liga ACB</title>
    <link href="estilos.css" rel="stylesheet" type="text/css" />
</head>

<body class="resultado">
```

```
...
```

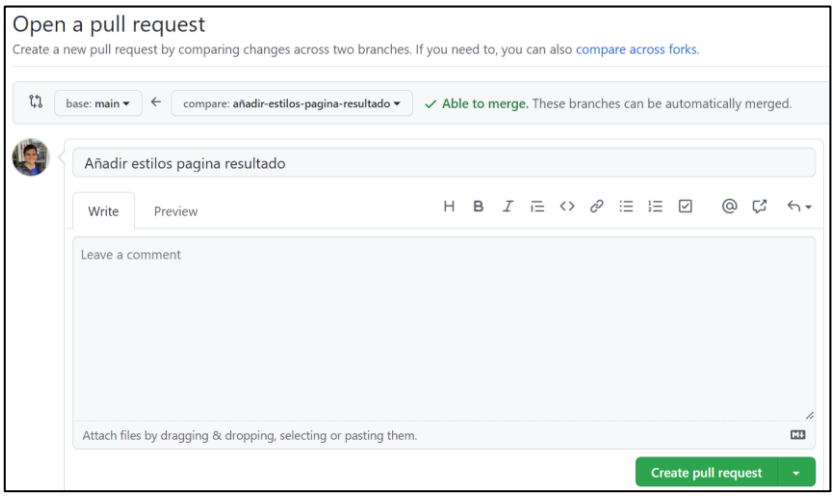
Estos cambios se deben enviar al repositorio compartido.

```
C:\Baloncesto> git add .
C:\Baloncesto> git status
C:\Baloncesto> git commit -m "Corregidos conflictos estilo en página de
resultado"
C:\Baloncesto> git push origin añadir-estilos-pagina-resultado
```

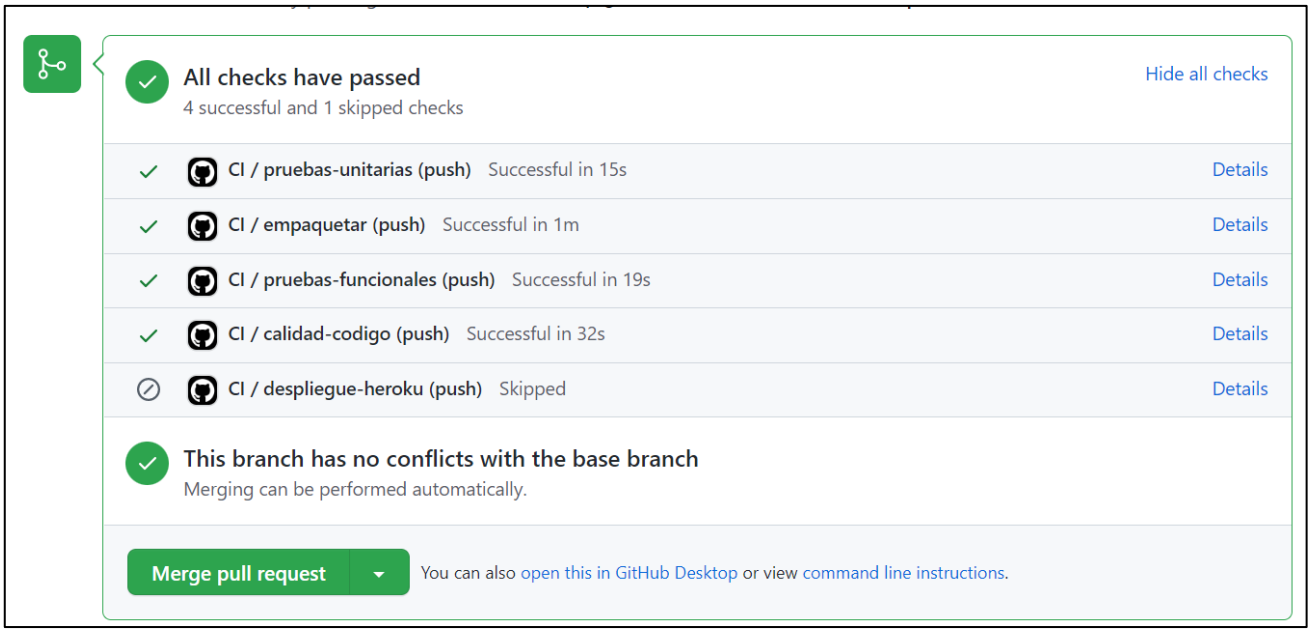
Si volvemos a Github e intentamos crear de nuevo la pull request, vemos que ya no aparece el problema anterior. Ahora pulsamos el botón Create pull request.



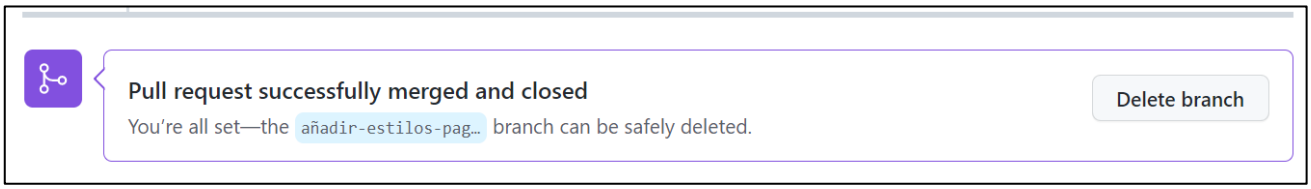
Nos aparecerá el siguiente diálogo donde crearemos la pull request.



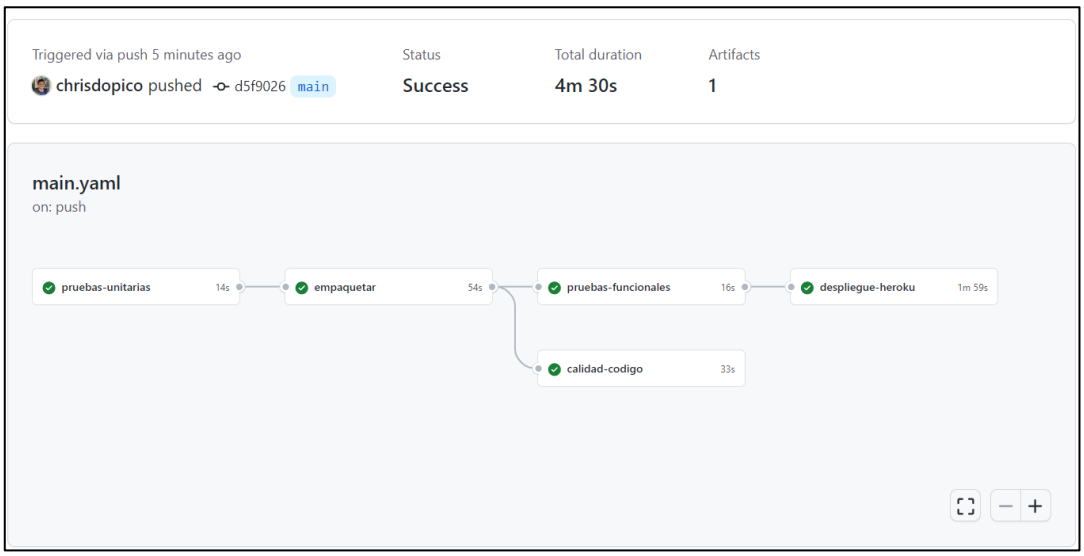
Luego pulsamos Merge pull request.



Finalmente, vemos como el merge se realizó con éxito.

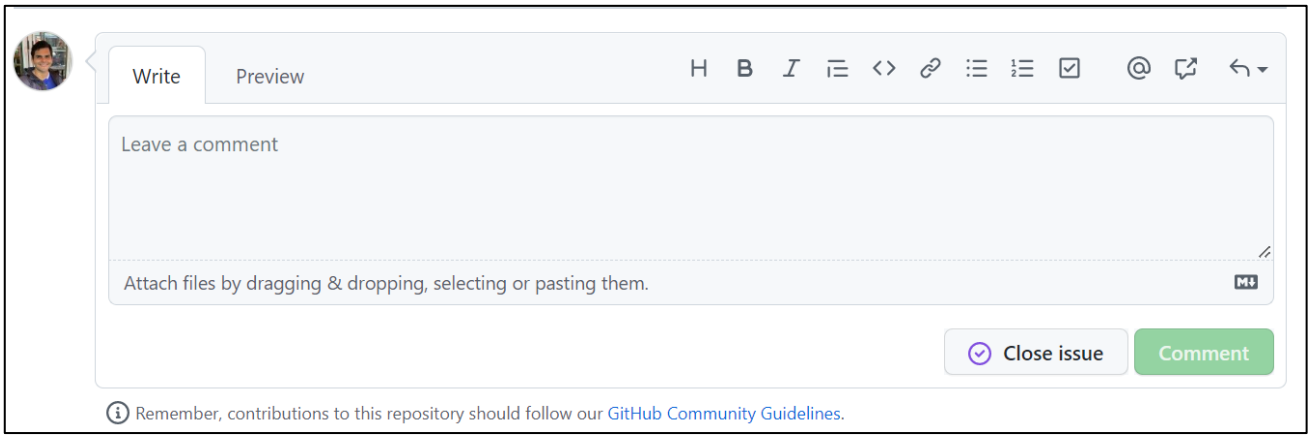


Como definimos en nuestro archivo main.yaml que el despliegue a heroku solo se hará cuando haya cambios en la rama main.

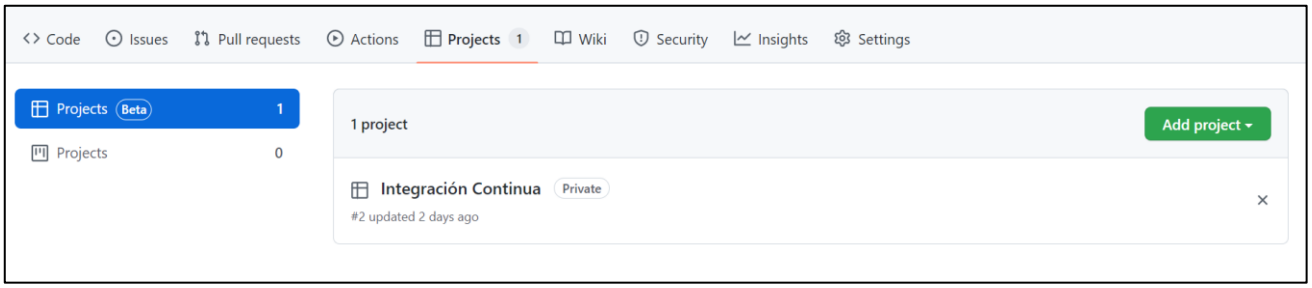


Si accedemos a https://baloncesto.herokuapp.com, vemos que los estilos de las dos páginas son correctos.

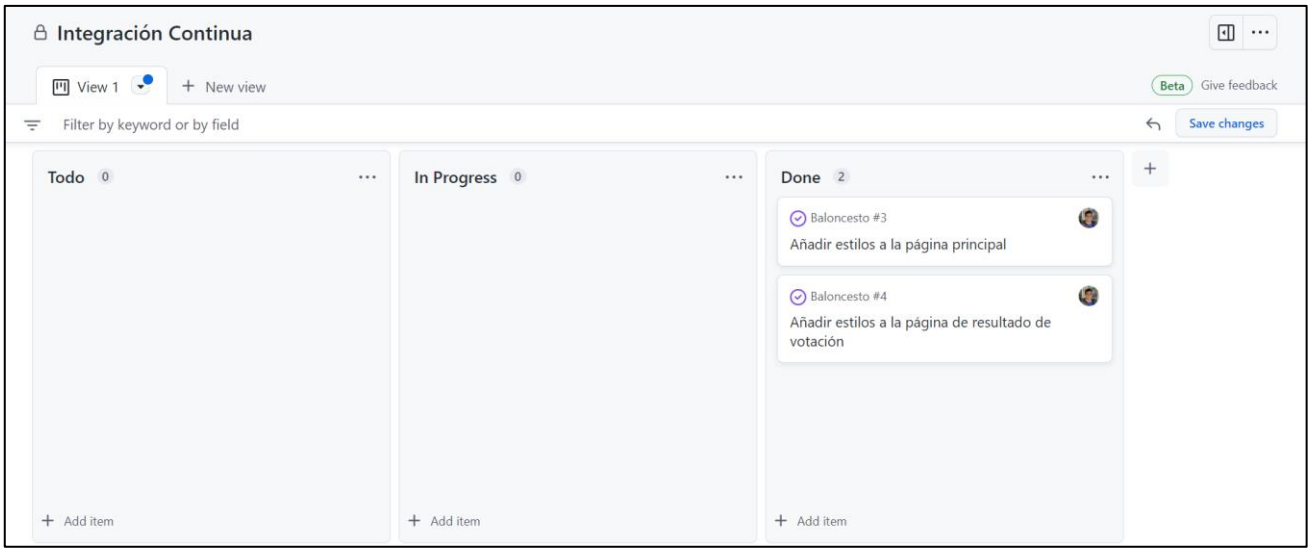
Con esto hemos terminado el issue "Añadir estilos a la página de resultado de votación", nos dirigimos a la sección issues y pulsamos el botón "Close issue".



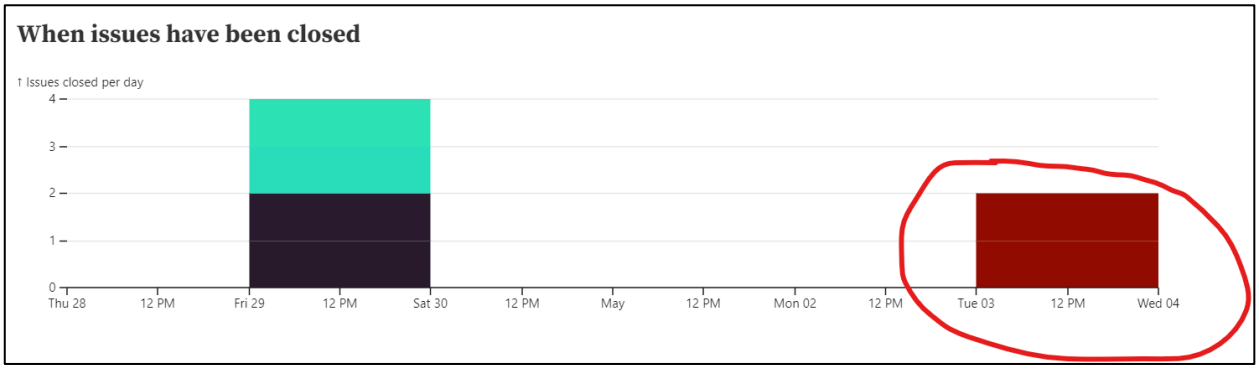
Como siguiente paso, nos dirigimos a la sección Projects, y damos click sobre nuestro proyecto "Integración Continua".



Cambiamos la vista a modo tablero y podremos observar que nuestras dos issues han pasado a la columna "Done".



Si accedemos al diagrama burndown en GitHubBurndown/TomMacWright/Observable (observablehq.com) podremos observar que se han cerrado las dos issues de nuestro proyecto.



8. REALIZAR TERCERA VERSIÓN DE LA APLICACIÓN

8.1. Requisitos

Para la tercera versión de la aplicación se deben cumplir los siguientes requisitos:

- Se va a añadir al workflow del proyecto una nueva fase, entre las fases "qa" y "deploy", llamada "release" que contenga un trabajo llamado "despliegue-host-heroku-pre", su objetivo será desplegar la aplicación en un entorno de pre-producción, que puede ser el mismo hosting Heroku, pero usando otro nombre para la aplicación, por ejemplo baloncesto-pre.
- Se creará un nuevo milestone con los siguientes requerimientos:
    - REQ-1: Como nuevo requisito de la aplicación, la página principal debe ofrecer al usuario un botón "Poner votos a cero", que al pulsarlo se pongan a 0 los votos de todos los jugadores en la base de datos.
    - REQ-2: Como nuevo requisito de la aplicación, la página principal debe ofrecer al usuario un botón "Ver votos", que al pulsarlo muestre una nueva página (por ejemplo, VerVotos.jsp), que muestre una tabla o lista con los nombres de todos los jugadores que hay en la base de datos y sus votos, y un enlace para volver a la página principal.
    - PF-A: Programar en PruebasPhantomjsIT.java una nueva prueba funcional que simule en la página principal la pulsación del botón "Poner votos a cero", después la pulsación del botón "Ver votos", y compruebe que en los votos que aparecen para cada jugador en la página VerVotos.jsp son todos cero.
    - PF-B: Programar en PruebasPhantomjsIT.java una nueva prueba funcional que introduzca en la caja de la página principal el nombre de un nuevo jugador y marque la opción "Otro", pulse el botón "Votar", vuelva a la página principal, simule la pulsación del botón "Ver votos", y compruebe que en la página VerVotos.jsp ese nuevo jugador tiene 1 voto.
    - PU: Programar en ModeloDatosTest.java un caso de prueba para el método actualizarJugador(), que compruebe, simulando una base de datos de prueba, que realmente se incrementa en 1 los votos del jugador.
    - QA: Como garantía de calidad, el código fuente completo del proyecto debe tener un límite de problemas importantes (major issues) de 20 calculados con SonarQube, en otro caso no debe poderse desplegar la aplicación a producción
- Se van a programar los issues en dos ramas:
    - Una para el requisito REQ-1. La prueba unitaria PU debe implementarse como parte de esta rama.
    - Otra para el requisito REQ-2. Este requisito debe implementarse una vez finalizado el anterior. Las pruebas funcionales deben implementarse como parte de esta rama.

8.2. Agregar fase release

En proyectos de desarrollo es común tener una fase de pre-producción donde todos los cambios se aplican antes de llegar a producción. Para realizar este proceso, debemos crear otra aplicación en nuestro host de heroku.

Una vez tenemos nuestra aplicación de heroku, procedemos a agregar el siguiente job entre los trabajos de calidad-codigo y despliegue-heroku. Otro punto a adicionar es la dependencia del trabajo despliegue-heroku del trabajo despliegue-heroku-pre.

Archivo C:\Baloncesto\.github\workflows\main.yaml

```
    despliegue-heroku-pre:
      runs-on: self-hosted
      needs: pruebas-funcionales
      if: github.ref == 'refs/heads/main'
      steps:
        - name: deploy
          run: |
            export HEROKU_API_KEY=${{secrets.HEROKU_API_KEY}}
            mvn heroku:deploy-war -Dheroku.appName=baloncestorelease -
DskipTests=true


    despliegue-heroku:
      runs-on: self-hosted
      needs: despliegue-heroku-pre
      if: github.ref == 'refs/heads/main'
      steps:
```

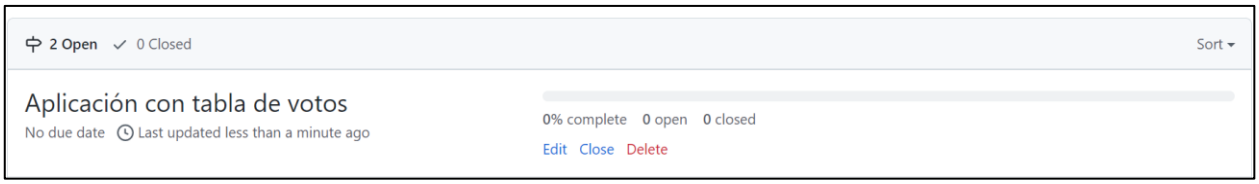
```
        - name: deploy
          run: |
            export HEROKU_API_KEY=${{secrets.HEROKU_API_KEY}}
            mvn heroku:deploy-war -Dheroku.appName=baloncestogithub -
DskipTests=true
```

Ejecutamos el workflow, una vez realizado, la nueva página de pre-produccion se nos muestra.

8.3. Crear nuevo sprint (milestone)

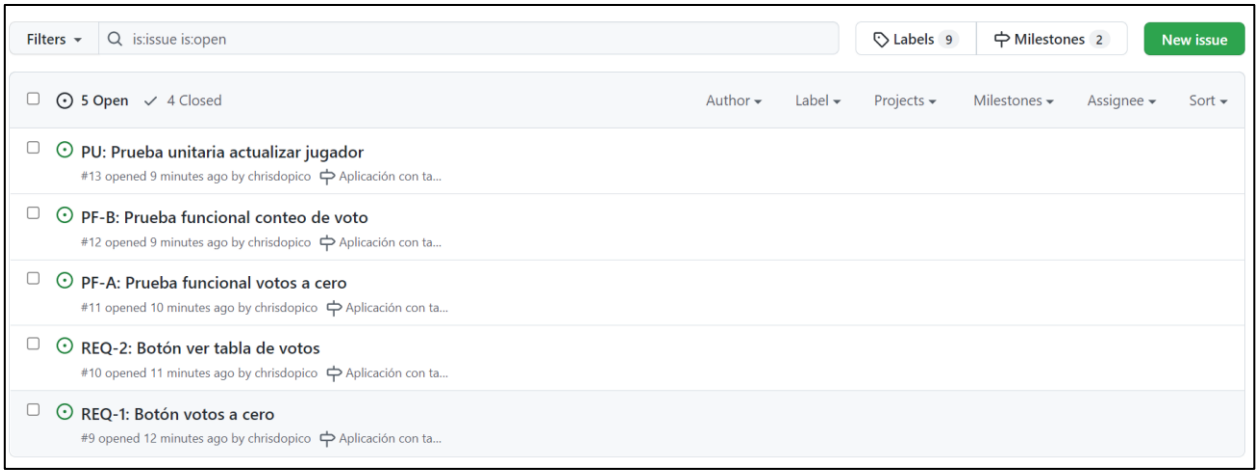
8.3.1. Creación milestone

Para crear nuestro milestone seguimos los pasos de la sección 7.1.2.

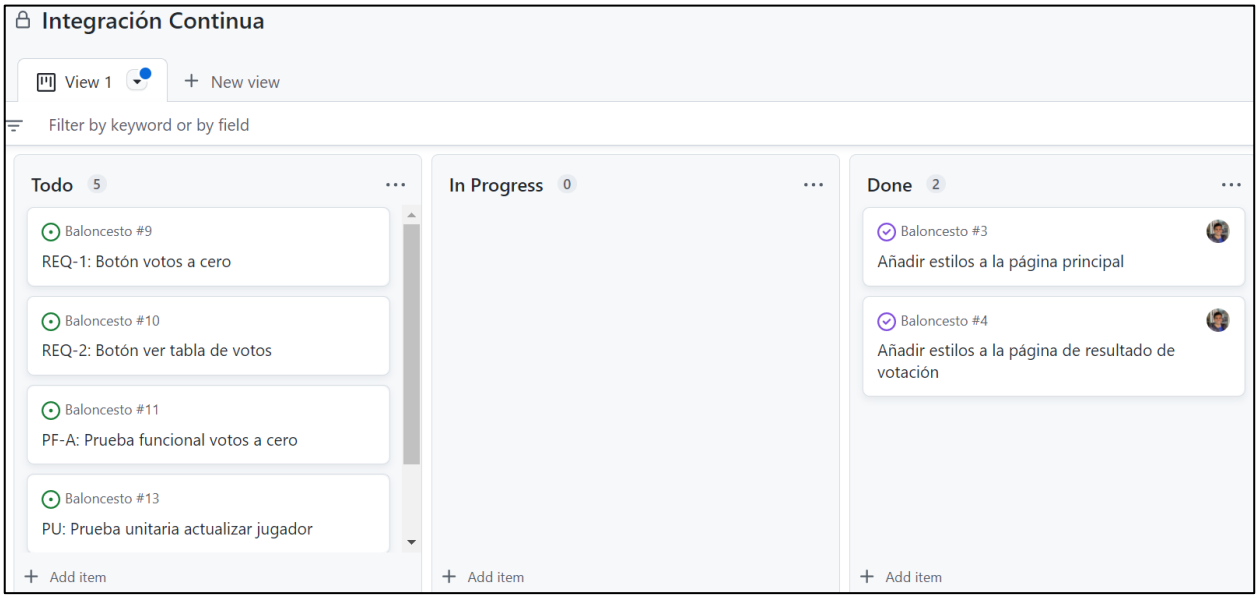


8.3.2. Creación de issues

Ahora creamos la issues para nuestro milestone.



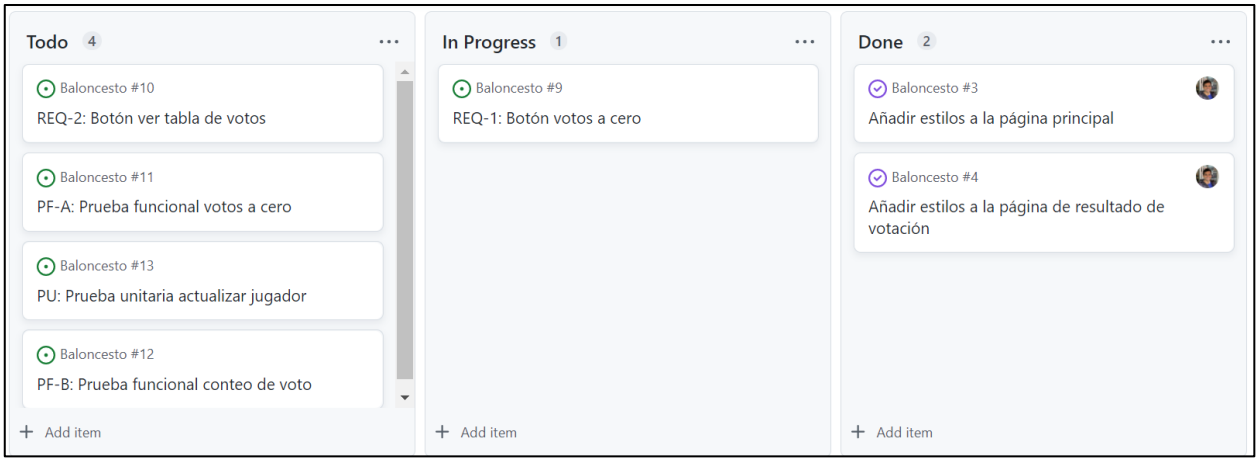
También podemos visualizar nuestro tablero del proyecto. Nuestras tareas estarán sin estado, para esto debemos moverlas a la etiqueta "To Do"



8.4. Agregar funcionalidad votos a cero

Como nuevo requisito de la aplicación, la página principal debe ofrecer al usuario un botón "Poner votos a cero", que al pulsarlo se pongan a 0 los votos de todos los jugadores en la base de datos.

Para empezar, movemos nuestra issue al estado "In Progress".



8.4.1. Creación de nueva rama

Para realizar esta issue, creamos una nueva rama desde la rama main.

```
* añadir-boton-votos-cero
  añadir-estilos-pagina-principal
  añadir-estilos-pagina-resultado
  main
```

8.4.2. Modificación de ficheros en proyecto

Debemos Modificar la clase ModeloDatos.java y agregar un nuevo método para poner los votos a cero.

Archivo C:\Baloncesto\src\main\java\ModeloDatos.java

```
import java.sql.*;

public class ModeloDatos {

    private Connection con;
    private Statement set;
    private ResultSet rs;

    public void abrirConexion() {

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
```

```
            // Con variables de entorno
            String dbHost = System.getenv().get("DATABASE_HOST");
            String dbPort = System.getenv().get("DATABASE_PORT");
            String dbName = System.getenv().get("DATABASE_NAME");
            String dbUser = System.getenv().get("DATABASE_USER");
            String dbPass = System.getenv().get("DATABASE_PASS");

            String url = dbHost + ":" + dbPort + "/" + dbName;
            con = DriverManager.getConnection(url, dbUser, dbPass);

        } catch (Exception e) {
            // No se ha conectado
            System.out.println("No se ha podido conectar");
            System.out.println("El error es: " + e.getMessage());
        }
    }

    public boolean existeJugador(String nombre) {
        boolean existe = false;
        String cad;
        try {
            set = con.createStatement();
            rs = set.executeQuery("SELECT * FROM Jugadores");
            while (rs.next()) {
                cad = rs.getString("Nombre");
                cad = cad.trim();
                if (cad.compareTo(nombre.trim()) == 0) {
                    existe = true;
                }
            }
            rs.close();
            set.close();
        } catch (Exception e) {
            // No lee de la tabla
            System.out.println("No lee de la tabla");
            System.out.println("El error es: " + e.getMessage());
        }
        return (existe);
    }

    public void actualizarJugador(String nombre) {
        try {
            set = con.createStatement();
            set.executeUpdate("UPDATE Jugadores SET votos=votos+1 WHERE
nombre " + " LIKE '%" + nombre + "%'");
            rs.close();
            set.close();
        } catch (Exception e) {
            // No modifica la tabla
            System.out.println("No modifica la tabla");
            System.out.println("El error es: " + e.getMessage());
        }
    }

    public void insertarJugador(String nombre) {
        try {
            set = con.createStatement();
            set.executeUpdate("INSERT INTO Jugadores " + " (nombre,votos)
VALUES ('" + nombre + "',1)");
            rs.close();
```

```
            set.close();
        } catch (Exception e) {
            // No inserta en la tabla
            System.out.println("No inserta en la tabla");
            System.out.println("El error es: " + e.getMessage());
        }
    }

    public void cerrarConexion() {
        try {
            con.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    public boolean votosACero() {
        boolean indicador = false;
        try {
            set = con.createStatement();
            set.executeUpdate("UPDATE Jugadores SET votos = 0;");
            indicador = true;
            set.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        return indicador;
    }

}
```

Ahora editamos nuestra clase Acb.java, donde vamos a decidir qué hacer en caso de pulsar el botón "Reset". Hemos agregado en el código condicionales para que según el botón que el usuario pulse en la pantalla, se realice una acción u otra.

Archivo C:\Baloncesto\src\main\java\Acb.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Acb extends HttpServlet {

    private ModeloDatos bd;

    public void init(ServletConfig cfg) throws ServletException {
        bd = new ModeloDatos();
        bd.abrirConexion();
    }

    public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
        HttpSession s = req.getSession(true);
        String nombreP = (String) req.getParameter("txtNombre");
        String nombre = (String) req.getParameter("R1");
        PrintWriter out = res.getWriter();

        // Cuando se pulse botón Votar
        if (req.getParameter("B1") != null) {
```

```
            if (nombre.equals("Otros")) {
                nombre = (String) req.getParameter("txtOtros");
            }
            if (bd.existeJugador(nombre)) {
                bd.actualizarJugador(nombre);
            } else {
                bd.insertarJugador(nombre);
            }
            s.setAttribute("nombreCliente", nombreP);
            // Llamada a la página jsp que nos da las gracias
            res.sendRedirect(res.encodeRedirectURL("TablaVotos.jsp"));
        }
        // Cuando se pulse el votón Reset
        else if (req.getParameter("B2") != null) {
            try {
                boolean indicador = bd.votosACero();
                if (indicador) {
                    out.print("Votos a Cero");
                } else {
                    out.print("Error");
                }

                // Llamada a la página jsp que nos avisa que se han puesto
los votos a cero
                res.sendRedirect(res.encodeRedirectURL("VotosCero.jsp"));

            } catch (NumberFormatException e) {
                out.println("Number Format Exception" + e);
            } catch (IndexOutOfBoundsException e) {
                out.println("Index out of bounds Exception" + e);
            } finally {
                out.close();
            }
        }

    }

    public void destroy() {
        bd.cerrarConexion();
        super.destroy();
    }
}
```

Creamos la jsp para el mensaje de éxito cuando los votos se ponen en cero.

Archivo C:\Baloncesto\src\webapp\VotosCero.jsp

```
<html>
    <head>
        <title>Votaci&oacute;n mejor jugador liga ACB</title>
        <link href="estilos.css" rel="stylesheet" type="text/css" />
    </head>
    <body class="resultado">
        <font size=10>
        Votaci&oacute;n al mejor jugador de la liga ACB
        <hr>
        <br>Todos los votos se han puesto en cero
        </font>
```

```
        <br>
        <br> <a href="index.html"> Ir al comienzo</a>
    </body>
</html>
```

Finalmente, editamos el archivo index.html cambiando el type del botón "Reset" a submit.

Archivo C:\Baloncesto\src\webapp\index.html

```
<html>
    <head>
        <title>Votacion mejor jugador liga ACB</title>
        <link href="estilos.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
    <center><H1>Votaci&oacute;n al mejor jugador de la liga
ACB</H1></center><hr>
    <form action="Acb" method="POST">
        <p align="left">Nombre del Visitante: <input type="text" size="20"
                                                  name="txtNombre">
            eMail: <input type="text" size="20" name="txtMail"></p> REAL
MADRID:<br>

        <img src="Carroll.jpg" width="100" height="67" alt="equipo1"/>
        <p align="left"><input type="radio" name="R1"
value="Carroll">Carroll</p>

        <img src="Llull.jpg" width="100" height="67" alt="equipo1"/>
        <p align="left"><input type="radio" name="R1"
value="Llull">Llull</p>

        <img src="Rudy.jpg" width="100" height="67" alt="equipo1"/>
        <p align="left"><input type="radio" name="R1" value="Rudy">Rudy</p>

        <br>
        <p align="left"><input type="radio" name="R1" value="Otros">Otro
            <input type="text" size="20" name="txtOtros"></p>
        <p align="left">
            <input type="submit" name="B1" value="Votar">
            <input type="submit" name="B2" value="Poner votos a cero">
        </p>
    </form>
</body>
</html>
```

8.4.3. Verificación de resultados

Una vez modificados los archivos, procedemos hacer commit en la rama "añadir-boton-votos-cero" y probamos los cambios en nuestro contenedor Ubuntu.

Base de datos antes de presionar el botón "Reset"

```
mysql> select * from Jugadores;
+----+---------+-------+
| id | nombre  | votos |
+----+---------+-------+
|  1 | Carroll |     1 |
|  2 | Llull   |     1 |
|  3 | Rudy    |     1 |
+----+---------+-------+
3 rows in set (0.00 sec)
```

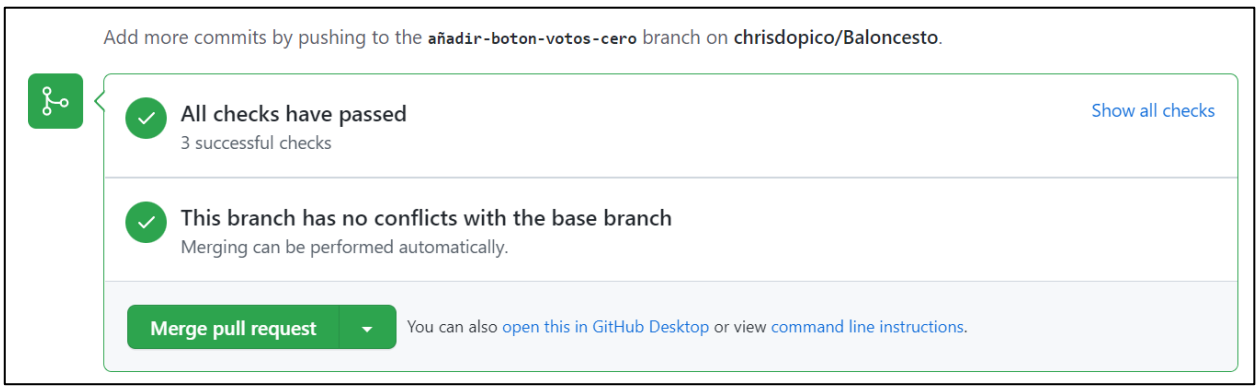
Base de datos después de presionar el botón "Reset"

```
mysql> select * from Jugadores;
+----+---------+-------+
| id | nombre  | votos |
+----+---------+-------+
|  1 | Carroll |     0 |
|  2 | Llull   |     0 |
|  3 | Rudy    |     0 |
+----+---------+-------+
3 rows in set (0.00 sec)
```

8.4.4. Merge request

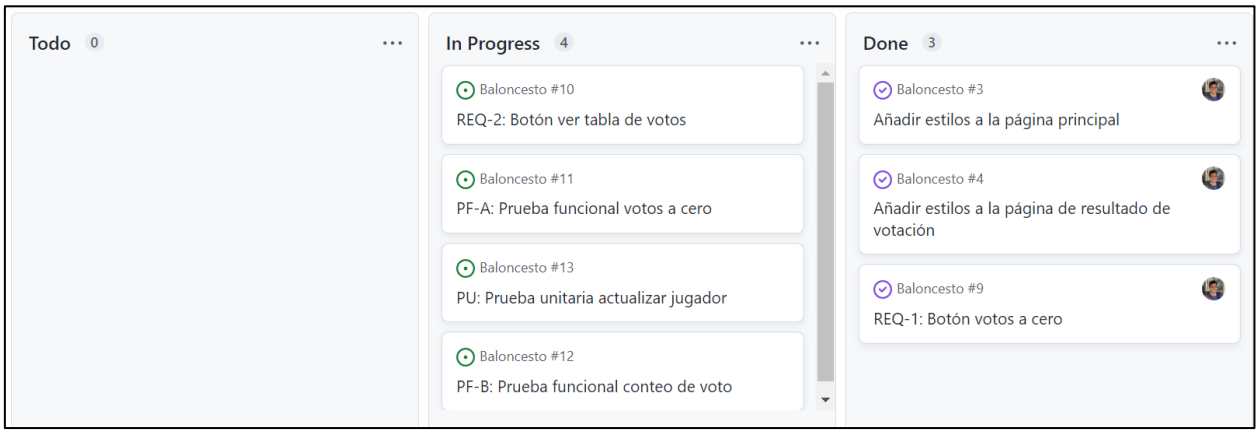
Una vez que nuestra issue ha sido completada, procedemos a hacer un merge request a la rama main y realizar le despliegue.

Primero realizamos una pull request, tal como hicimos en la sección 7.2.1.



Luego damos clic en Merge Pull Request y cerramos la issue correspondiente.

Ahora podemos ver en nuestro tablero kanban que de manera automática nuestra issue pasa a la etiqueta "Done".



8.5. Agregar funcionalidad ver tabla de votos

Como nuevo requisito de la aplicación, la página principal debe ofrecer al usuario un botón "Ver votos", que al pulsarlo muestre una nueva página (por ejemplo, VerVotos.jsp), que muestre una tabla o lista con los nombres de todos los jugadores que hay en la base de datos y sus votos, y un enlace para volver a la página principal.

8.5.1. Crear nueva rama

Para realizar esta issue, creamos una nueva rama desde la rama main.

```
  añadir-boton-votos-cero
  añadir-estilos-pagina-principal
  añadir-estilos-pagina-resultado
* añadir-tabla-votos
  main
```

Debemos Modificar la clase ModeloDatos.java y agregar un nuevo método para poner los votos a cero.

8.5.2. Modificación ficheros del proyecto

Para poder crear la tabla, necesitamos un paquete modelo, donde almacenaremos la clase Jugador y también incluiremos en el paquete la clase ModeloDatos.java.

Archivo C:\Baloncesto\src\main\java\modelo\Jugador.java

```
package modelo;

public class Jugador {
    private int jugadorId;
    private String nombre;
    private int totalVotos;
```

```
    public Jugador() {
    }

    public Jugador(int jugadorId, String nombre, int totalVotos) {
        this.jugadorId = jugadorId;
        this.nombre = nombre;
        this.totalVotos = totalVotos;
    }

    public int getId() {
        return jugadorId;
    }

    public void setId(int jugadorId) {
        this.jugadorId = jugadorId;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getTotalVotos() {
        return totalVotos;
    }

    public void setTotalVotos(int totalVotos) {
        this.totalVotos = totalVotos;
    }

}
```

Agregamos el método getJugadores() enla clase ModeloDatos.java.

Archivo C:\Baloncesto\src\main\java\modelo\ModeloDatos.java

```
package modelo;

import java.sql.*;
import java.util.ArrayList;

public class ModeloDatos {

    private Connection con;
    private Statement set;
    private ResultSet rs;

    public void abrirConexion() {

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Con variables de entorno
            String dbHost = System.getenv().get("DATABASE_HOST");
            String dbPort = System.getenv().get("DATABASE_PORT");
```

```
            String dbName = System.getenv().get("DATABASE_NAME");
            String dbUser = System.getenv().get("DATABASE_USER");
            String dbPass = System.getenv().get("DATABASE_PASS");

            String url = dbHost + ":" + dbPort + "/" + dbName;
            con = DriverManager.getConnection(url, dbUser, dbPass);

        } catch (Exception e) {
            // No se ha conectado
            System.out.println("No se ha podido conectar");
            System.out.println("El error es: " + e.getMessage());
        }
    }

    public boolean existeJugador(String nombre) {
        boolean existe = false;
        String cad;
        try {
            set = con.createStatement();
            rs = set.executeQuery("SELECT * FROM Jugadores");
            while (rs.next()) {
                cad = rs.getString("Nombre");
                cad = cad.trim();
                if (cad.compareTo(nombre.trim()) == 0) {
                    existe = true;
                }
            }
            rs.close();
            set.close();
        } catch (Exception e) {
            // No lee de la tabla
            System.out.println("No lee de la tabla");
            System.out.println("El error es: " + e.getMessage());
        }
        return (existe);
    }

    public void actualizarJugador(String nombre) {
        try {
            set = con.createStatement();
            set.executeUpdate("UPDATE Jugadores SET votos=votos+1 WHERE
nombre " + " LIKE '%" + nombre + "%'");
            rs.close();
            set.close();
        } catch (Exception e) {
            // No modifica la tabla
            System.out.println("No modifica la tabla");
            System.out.println("El error es: " + e.getMessage());
        }
    }

    public void insertarJugador(String nombre) {
        try {
            set = con.createStatement();
            set.executeUpdate("INSERT INTO Jugadores " + " (nombre,votos)
VALUES ('" + nombre + "',1)");
            rs.close();
            set.close();
        } catch (Exception e) {
            // No inserta en la tabla
```

```
            System.out.println("No inserta en la tabla");
            System.out.println("El error es: " + e.getMessage());
        }
    }

    public void cerrarConexion() {
        try {
            con.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    public boolean votosACero() {
        boolean indicador = false;
        try {
            set = con.createStatement();
            set.executeUpdate("UPDATE Jugadores SET votos = 0;");
            indicador = true;
            set.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        return indicador;
    }


    public ArrayList<Jugador> getJugadores() {
        ArrayList<Jugador> jugadores = new ArrayList<>();
        try {
            abrirConexion();
            set = con.createStatement();
            rs = set.executeQuery("SELECT * FROM Jugadores;");
            while (rs.next()) {
                Jugador player = new Jugador();
                player.setId(rs.getInt("id"));
                player.setNombre(rs.getString("nombre"));
                player.setTotalVotos(rs.getInt("votos"));
                jugadores.add(player);
            }
            rs.close();
            set.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        return jugadores;
    }

}
```

Creamos la jsp para generar la vista de la tabla con los votos.

Archivo C:\Baloncesto\src\webapp\ListaVotos.jsp

```
<%@ page import="java.util.ArrayList"%>
<%@ page import="java.util.Iterator"%>
<%@ page import="modelo.ModeloDatos" %>
<%@ page import="modelo.Jugador" %>
```

```
<%@ page contentType="text/html" pageEncoding="UTF-8"%>

<html>
    <head><title>Ver votos</title></head>
    <link href="estilos.css" rel="stylesheet" type="text/css" />

    <body class="listado">
        <h1>Tabla de Votos</h1>
        <hr>
        <div>
            <%
                ModeloDatos bd = new ModeloDatos();
                ArrayList<Jugador> jugadores = bd.getJugadores();
                Iterator<Jugador> listado = jugadores.iterator();
            %>
            <table>
                <thead>
                    <th scope="col">Id</th>
                    <th scope="col">Nombre</th>
                    <th scope="col">Total Votos</th>
                </thead>
                <tbody>
                    <%
                        int contador = 0;
                        Jugador jugador = new Jugador();
                        while(listado.hasNext()){
                            contador++;
                            jugador = listado.next();

                    %>
                    <tr>
                        <td id="contador<%=contador%>">
                            <%=contador%>
                        </td>
                        <td id="jugador<%=contador%>">
                            <%=jugador.getNombre()%>
                        </td>
                        <td id="votes<%=contador%>">
                            <%=jugador.getTotalVotos()%>
                        </td>
                    </tr>
                    <%
                        }
                    %>
                </tbody>
            </table>
        </div>
        <br> <a href="index.html"> Ir al comienzo</a>
    </body>
</html>
```

Agregamos el estilo de la página en el fichero estilos.css.

Archivo C:\Baloncesto\src\webapp\estilos.css

```
body {
    color: white;
    background-color: darkblue;
```

```
}

.resultado {
    color: yellow;
    background-color: darkred;
}

.listado {
    color: black;
    background-color: lightblue;
}
```

Modificamos el index.html para agregar el nuevo botón "Ver Tabla".

Archivo C:\Baloncesto\src\webapp\index.html

```
<html>
    <head>
        <title>Votacion mejor jugador liga ACB</title>
        <link href="estilos.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
    <center><H1>Votaci&oacute;n al mejor jugador de la liga
ACB</H1></center><hr>
    <form action="Acb" method="POST">
        <p align="left">Nombre del Visitante: <input type="text" size="20"
                                                    name="txtNombre">
            eMail: <input type="text" size="20" name="txtMail"></p> REAL
MADRID:<br>

        <img src="Carroll.jpg" width="100" height="67" alt="equipo1"/>
        <p align="left"><input type="radio" name="R1"
value="Carroll">Carroll</p>

        <img src="Llull.jpg" width="100" height="67" alt="equipo1"/>
        <p align="left"><input type="radio" name="R1"
value="Llull">Llull</p>

        <img src="Rudy.jpg" width="100" height="67" alt="equipo1"/>
        <p align="left"><input type="radio" name="R1" value="Rudy">Rudy</p>

        <br>
        <p align="left"><input type="radio" name="R1" value="Otros">Otro
            <input type="text" size="20" name="txtOtros"></p>
        <p align="left">
            <input type="submit" name="B1" value="Votar">
            <input type="submit" name="B2" value="Poner votos a cero">
            <input type="submit" name="B3" value="Ver Tabla">
        </p>
    </form>
</body>
</html>
```

Finalmente, editamos nuestra clase Acb.java, donde vamos a decidir qué hacer en caso de pulsar el botón "Votar Tabla".

Archivo C:\Baloncesto\src\main\java\Acb.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import modelo.*;

public class Acb extends HttpServlet {

    private ModeloDatos bd;

    public void init(ServletConfig cfg) throws ServletException {
        bd = new ModeloDatos();
        bd.abrirConexion();
    }

    public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
        HttpSession s = req.getSession(true);
        String nombreP = (String) req.getParameter("txtNombre");
        String nombre = (String) req.getParameter("R1");
        PrintWriter out = res.getWriter();

        // Cuando se pulse botón Votar
        if (req.getParameter("B1") != null) {
            if (nombre.equals("Otros")) {
                nombre = (String) req.getParameter("txtOtros");
            }
            if (bd.existeJugador(nombre)) {
                bd.actualizarJugador(nombre);
            } else {
                bd.insertarJugador(nombre);
            }
            s.setAttribute("nombreCliente", nombreP);
            // Llamada a la página jsp que nos da las gracias
            res.sendRedirect(res.encodeRedirectURL("TablaVotos.jsp"));
        }
        // Cuando se pulse el botón Reset
        else if (req.getParameter("B2") != null) {
            try {
                boolean indicador = bd.votosACero();
                if (indicador) {
                    out.print("Votos a Cero");
                } else {
                    out.print("Error");
                }

                // Llamada a la página jsp que nos avisa que se han puesto
los votos a cero
                res.sendRedirect(res.encodeRedirectURL("VotosCero.jsp"));

            } catch (NumberFormatException e) {
                out.println("Number Format Exception" + e);
            } catch (IndexOutOfBoundsException e) {
                out.println("Index out of bounds Exception" + e);
            } finally {
                out.close();
            }
        }
```

```
        // Cuando se pulse el botón Ver Tabla
        else if (req.getParameter("B3") != null) {
            try {
                // Llamada a la página jsp que nos avisa que se han puesto
los votos a cero
                res.sendRedirect(res.encodeRedirectURL("ListaVotos.jsp"));
            } catch (NumberFormatException e) {
                out.println("Number Format Exception" + e);
            } catch (IndexOutOfBoundsException e) {
                out.println("Index out of bounds Exception" + e);
            } finally {
                out.close();
            }
        }

    }

    public void destroy() {
        bd.cerrarConexion();
        super.destroy();
    }
}
```

8.5.3. Verificación de resultados

Una vez modificados los archivos, procedemos hacer commit en la rama "añadir-tabla-votos" y probamos los cambios en nuestro contenedor Ubuntu.

Tabla de Votos

Id	Nombre	Total Votos
1	Carroll	0
2	Llull	0
3	Rudy	0

Si agregamos votos, podemos ver la tabla con votos.

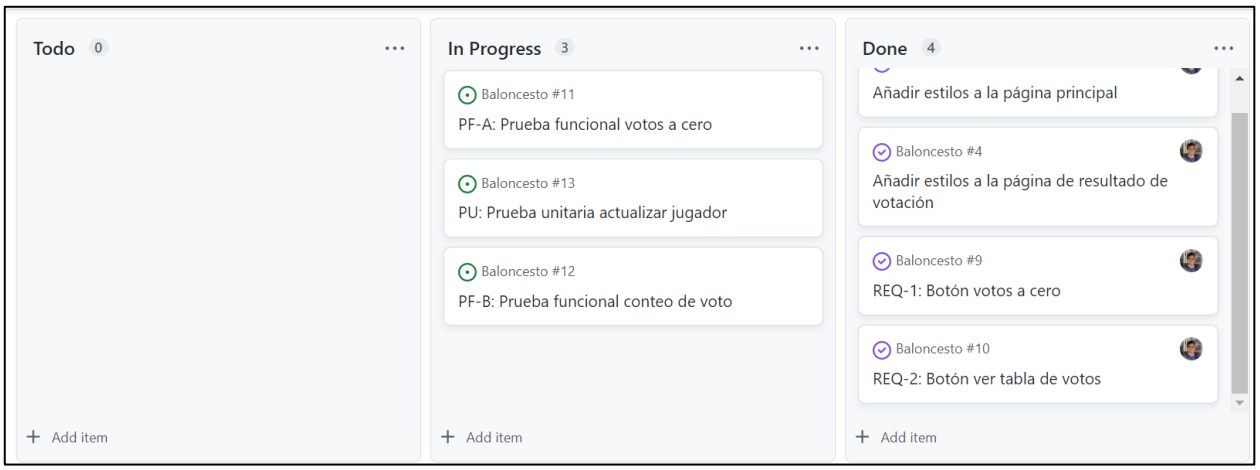
Tabla de Votos

Id	Nombre	Total Votos
1	Carroll	1
2	Llull	1
3	Rudy	0

8.5.4. Merge Request

Primero realizamos una pull request, tal como hicimos en la sección 7.2.1.

Confirmamos los cambios y seleccionamos Merge Request. Luego cerramos nuestra issue y podremos ver en el tablero que nuestra issue ha sido cerrada correctamente.



8.6. Agregar prueba funcional para votos a cero

Programar en PruebasPhantomjsIT.java una nueva prueba funcional que simule en la página principal la pulsación del botón "Poner votos a cero", después la pulsación del botón "Ver Tabla", y compruebe que en los votos que aparecen para cada jugador en la página VerVotos.jsp son todos cero.

Para esta prueba, modificamos el archivo PruebasPhantom.IT.java y agregamos el test setVotosToCeroTest().

Archivo C:\Baloncesto\src\test\java\ PruebasPhantomIT.java

```
import java.util.ArrayList;
import java.util.List;

import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.phantomjs.PhantomJSDriver;
import org.openqa.selenium.phantomjs.PhantomJSDriverService;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

class PruebasPhantomjsIT {
    private static PhantomJSDriver driver = null;

    @Test
    void tituloIndexTest() {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps.setJavascriptEnabled(true);

caps.setCapability(PhantomJSDriverService.PHANTOMJS_EXECUTABLE_PATH_PROPERTY
, "/usr/bin/phantomjs");
        caps.setCapability(PhantomJSDriverService.PHANTOMJS_CLI_ARGS,
```

```
                new String[] { "--web-security=no", "--ignore-ssl-
errors=yes" });
        driver = new PhantomJSDriver(caps);
        driver.navigate().to("http://localhost:8080/Baloncesto/");
        assertEquals("Votacion mejor jugador liga ACB", driver.getTitle(),
                "El titulo no es correcto");
        System.out.println(driver.getTitle());
        driver.close();
        driver.quit();

    }

    @Test
    void setVotosToCeroTest() {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps.setJavascriptEnabled(true);

caps.setCapability(PhantomJSDriverService.PHANTOMJS_EXECUTABLE_PATH_PROPERTY
, "/usr/bin/phantomjs");
        caps.setCapability(PhantomJSDriverService.PHANTOMJS_CLI_ARGS,
                new String[] { "--web-security=no", "--ignore-ssl-
errors=yes" });
        driver = new PhantomJSDriver(caps);
        driver.navigate().to("http://localhost:8080/Baloncesto/");
        driver.findElement(By.name("B2")).click();
        driver.findElement(By.tagName("a")).click();
        driver.findElement(By.name("B3")).click();
        WebElement tablaVotos = driver.findElement(By.tagName("table"));
        ArrayList<WebElement> rows = new
ArrayList<>(tablaVotos.findElements(By.tagName("tr")));
        rows.remove(0);
        ArrayList<WebElement> listWebElementsVotes = new ArrayList<>();
        for (int i = 0; i < rows.size(); i++) {
            List<WebElement> cells =
rows.get(i).findElements(By.tagName("td"));
            listWebElementsVotes.add(cells.get(2));
        }

        boolean expectedResult = true;
        boolean verify = true;
        int counter = 0;
        while (verify && counter < listWebElementsVotes.size()) {
            if
(!listWebElementsVotes.get(counter).getText().trim().equals("0")) {
                verify = false;
            }
            counter++;
        }

        assertEquals(expectedResult, verify);

    }
```

8.7. Agregar prueba funcional para tabla de votos

Programar en PruebasPhantomjsIT.java una nueva prueba funcional que introduzca en la caja de la página principal el nombre de un nuevo jugador y marque la opción "Otro", pulse el botón "Votar", vuelva a la página principal, simule la pulsación del botón "Ver votos", y compruebe que en la página VerVotos.jsp ese nuevo jugador tiene 1 voto.

Para esta prueba, modificamos el archivo PruebasPhantom.IT.java y agregamos el test setVoteOtherPlayerTest().

Archivo C:\Baloncesto\src\test\java\ PruebasPhantomIT.java

```
import java.util.ArrayList;
import java.util.List;

import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.phantomjs.PhantomJSDriver;
import org.openqa.selenium.phantomjs.PhantomJSDriverService;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

class PruebasPhantomjsIT {
    private static PhantomJSDriver driver = null;

    @Test
    void tituloIndexTest() {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps.setJavascriptEnabled(true);

caps.setCapability(PhantomJSDriverService.PHANTOMJS_EXECUTABLE_PATH_PROPERTY
, "/usr/bin/phantomjs");
        caps.setCapability(PhantomJSDriverService.PHANTOMJS_CLI_ARGS,
                new String[] { "--web-security=no", "--ignore-ssl-
errors=yes" });
        driver = new PhantomJSDriver(caps);
        driver.navigate().to("http://localhost:8080/Baloncesto/");
        assertEquals("Votacion mejor jugador liga ACB", driver.getTitle(),
                "El titulo no es correcto");
        System.out.println(driver.getTitle());
        driver.close();
        driver.quit();

    }

    @Test
    void setVotosToCeroTest() {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps.setJavascriptEnabled(true);

caps.setCapability(PhantomJSDriverService.PHANTOMJS_EXECUTABLE_PATH_PROPERTY
, "/usr/bin/phantomjs");
        caps.setCapability(PhantomJSDriverService.PHANTOMJS_CLI_ARGS,
                new String[] { "--web-security=no", "--ignore-ssl-
errors=yes" });
        driver = new PhantomJSDriver(caps);
```

```
        driver.navigate().to("http://localhost:8080/Baloncesto/");
        driver.findElement(By.name("B2")).click();
        driver.findElement(By.tagName("a")).click();
        driver.findElement(By.name("B3")).click();
        WebElement tablaVotos = driver.findElement(By.tagName("table"));
        ArrayList<WebElement> rows = new
ArrayList<>(tablaVotos.findElements(By.tagName("tr")));
        rows.remove(0);
        ArrayList<WebElement> listWebElementsVotes = new ArrayList<>();
        for (int i = 0; i < rows.size(); i++) {
            List<WebElement> cells =
rows.get(i).findElements(By.tagName("td"));
            listWebElementsVotes.add(cells.get(2));
        }

        boolean expectedResult = true;
        boolean verify = true;
        int counter = 0;
        while (verify && counter < listWebElementsVotes.size()) {
            if
(!listWebElementsVotes.get(counter).getText().trim().equals("0")) {
                verify = false;
            }
            counter++;
        }

        assertEquals(expectedResult, verify);

    }

    @Test
    void setVoteOtherPlayerTest() {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps.setJavascriptEnabled(true);

caps.setCapability(PhantomJSDriverService.PHANTOMJS_EXECUTABLE_PATH_PROPERTY
, "/usr/bin/phantomjs");
        caps.setCapability(PhantomJSDriverService.PHANTOMJS_CLI_ARGS,
                new String[] { "--web-security=no", "--ignore-ssl-
errors=yes" });
        driver = new PhantomJSDriver(caps);
        driver.navigate().to("http://localhost:8080/Baloncesto/");

        driver.findElement(By.id("textoOtros")).click();
        String namePlayer = "Brandon Davies";
        driver.findElement(By.name("txtOtros")).sendKeys(namePlayer);
        driver.findElement(By.name("B1")).click();
        driver.findElement(By.tagName("a")).click();
        driver.findElement(By.name("B3")).click();

        boolean expectedResult = true;
        WebElement tablaVotos = driver.findElement(By.tagName("table"));
        ArrayList<WebElement> rows = new
ArrayList<>(tablaVotos.findElements(By.tagName("tr")));
        rows.remove(0);

        int counter = 0;
        boolean verify = false;
        while (!verify && counter < rows.size()) {
```

```
            List<WebElement> cells =
rows.get(counter).findElements(By.tagName("td"));
            if (cells.get(1).getText().trim().equals(namePlayer) &&
cells.get(2).getText().trim().equals("1")) {
                verify = true;
            }
            counter++;
        }

        assertEquals(expectedResult, verify);
    }

}
```

Finalmente, podremos observar en nuestro workflow que los tres tests se han ejecutado de manera satisfactoria.

```
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 15.367 s - in PruebasPhantomjsIT
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-failsafe-plugin:3.0.0-M6:verify (default-cli) @ Baloncesto ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  29.131 s
[INFO] Finished at: 2022-05-15T12:31:45Z
[INFO] ------------------------------------------------------------------------
```

8.8. Agregar prueba unitaria para actualizar jugador

Programar en ModeloDatosTest.java un caso de prueba para el método actualizarJugador(), que compruebe, simulando una base de datos de prueba, que realmente se incrementa en 1 los votos del jugador.

Para poder crear este test, necesitamos un método que nos recupere el nombre del objeto jugador en la clase ModeloDatos.java.

Archivo C:\Baloncesto\src\main\java\modelo\ModeloDatos.java

```
package modelo;

import java.sql.*;
import java.util.ArrayList;

public class ModeloDatos {

    private Connection con;
    private Statement set;
    private ResultSet rs;
```

```
    public void abrirConexion() {

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Con variables de entorno
            String dbHost = System.getenv().get("DATABASE_HOST");
            String dbPort = System.getenv().get("DATABASE_PORT");
            String dbName = System.getenv().get("DATABASE_NAME");
            String dbUser = System.getenv().get("DATABASE_USER");
            String dbPass = System.getenv().get("DATABASE_PASS");

            String url = dbHost + ":" + dbPort + "/" + dbName;
            con = DriverManager.getConnection(url, dbUser, dbPass);

        } catch (Exception e) {
            // No se ha conectado
            System.out.println("No se ha podido conectar");
            System.out.println("El error es: " + e.getMessage());
        }
    }

    public boolean existeJugador(String nombre) {
        boolean existe = false;
        String cad;
        try {
            set = con.createStatement();
            rs = set.executeQuery("SELECT * FROM Jugadores");
            while (rs.next()) {
                cad = rs.getString("Nombre");
                cad = cad.trim();
                if (cad.compareTo(nombre.trim()) == 0) {
                    existe = true;
                }
            }
            rs.close();
            set.close();
        } catch (Exception e) {
            // No lee de la tabla
            System.out.println("No lee de la tabla");
            System.out.println("El error es: " + e.getMessage());
        }
        return (existe);
    }

    public void actualizarJugador(String nombre) {
        try {
            set = con.createStatement();
            set.executeUpdate("UPDATE Jugadores SET votos=votos+1 WHERE
nombre " + " LIKE '%" + nombre + "%'");
            rs.close();
            set.close();
        } catch (Exception e) {
            // No modifica la tabla
            System.out.println("No modifica la tabla");
            System.out.println("El error es: " + e.getMessage());
        }
    }
```

```
    public void insertarJugador(String nombre) {
        try {
            set = con.createStatement();
            set.executeUpdate("INSERT INTO Jugadores " + " (nombre,votos)
VALUES ('" + nombre + "',1)");
            rs.close();
            set.close();
        } catch (Exception e) {
            // No inserta en la tabla
            System.out.println("No inserta en la tabla");
            System.out.println("El error es: " + e.getMessage());
        }
    }

    public void cerrarConexion() {
        try {
            con.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    public boolean votosACero() {
        boolean indicador = false;
        try {
            set = con.createStatement();
            set.executeUpdate("UPDATE Jugadores SET votos = 0;");
            indicador = true;
            set.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        return indicador;
    }


    public ArrayList<Jugador> getJugadores() {
        ArrayList<Jugador> jugadores = new ArrayList<>();
        try {
            abrirConexion();
            set = con.createStatement();
            rs = set.executeQuery("SELECT * FROM Jugadores;");
            while (rs.next()) {
                Jugador player = new Jugador();
                player.setId(rs.getInt("id"));
                player.setNombre(rs.getString("nombre"));
                player.setTotalVotos(rs.getInt("votos"));
                jugadores.add(player);
            }
            rs.close();
            set.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        return jugadores;
    }

    public Jugador getJugador(String nombreJugador) {
        Jugador jugador = new Jugador();
        try {
```

```
            abrirConexion();
            set = con.createStatement();
            rs = set.executeQuery("SELECT * FROM Jugadores WHERE nombre='" +
nombreJugador + "';");
            while (rs.next()) {
                jugador.setId(rs.getInt("id"));
                jugador.setNombre(rs.getString("nombre"));
                jugador.setTotalVotos(rs.getInt("votos"));
            }
            rs.close();
            set.close();
        } catch (Exception e) {
            System.out.println("Error " + e.getMessage());
        }

        return jugador;
    }

}
```

Agregamos nuestro test en el archivo ModeloDatosTest.java.

Archivo C:\Baloncesto\src\test\java\PruebasPhantomIT.java

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import modelo.*;

public class ModeloDatosTest {

    @Test
    public void testExisteJugador() {
        System.out.println("Prueba de existeJugador");
        String nombre = "";
        ModeloDatos instance = new ModeloDatos();
        boolean expResult = false;
        boolean result = instance.existeJugador(nombre);
        assertEquals(expResult, result);
        // fail("Fallo forzado.");
    }

    @Test
    public void testActualizarJugador() {
        System.out.println("Prueba de actualizarJugador");
        String nombre = "Rudy";
        ModeloDatos instance = new ModeloDatos();
        Jugador jugadorOriginal = instance.getJugador(nombre);
        instance.actualizarJugador(nombre);
        Jugador jugadorActualizado = instance.getJugador(nombre);
        assertEquals(jugadorOriginal.getTotalVotos() + 1,
jugadorActualizado.getTotalVotos());
    }
}
```

Finalmente, como estamos agregando un test unitario que simula una base de datos de prueba, necesitamos exportar las variables de entorno desde el job pruebas-unitarias, por lo que, debemos modificar el archivo main.yaml.

Archivo C:\Baloncesto\.github\workflows\main.yaml

```
name: CI


on:
  push:
    branches:
      - '**'



jobs:
  pruebas-unitarias:
    runs-on: self-hosted
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2.3.3
      - name: unit tests
        run:  |
          mysql -u root < db/baloncesto.sql
          export DATABASE_HOST="jdbc:mysql://localhost"
          export DATABASE_PORT="3306"
          export DATABASE_NAME="baloncesto"
          export DATABASE_USER="usuario"
          export DATABASE_PASS="clave"
          mvn test



  empaquetar:
    runs-on: self-hosted
    needs: pruebas-unitarias
    steps:
      - name: build
        run: mvn package -DskipTests=true
```

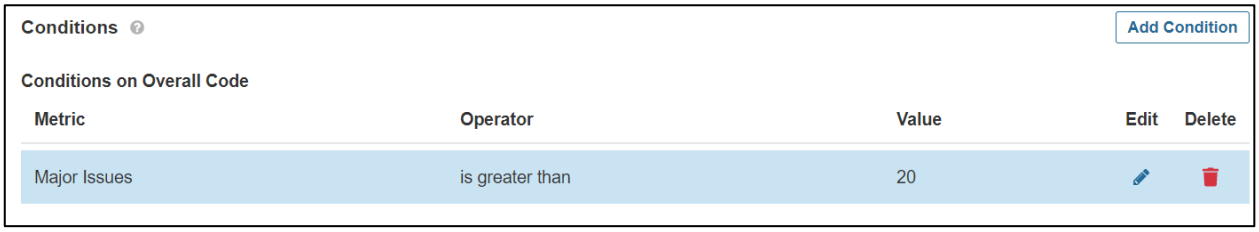
```
    pruebas-funcionales:
       runs-on: self-hosted
       needs: empaquetar
       steps:
         - name: test
           run: |
            mysql -u root < db/baloncesto.sql
            export DATABASE_HOST="jdbc:mysql://localhost"
            export DATABASE_PORT="3306"
            export DATABASE_NAME="baloncesto"
            export DATABASE_USER="usuario"
            export DATABASE_PASS="clave"
            cp -r target/Baloncesto /usr/local/tomcat/webapps
            sh /usr/local/tomcat/bin/catalina.sh stop
            sh /usr/local/tomcat/bin/catalina.sh start
            mvn failsafe:integration-test failsafe:verify



    calidad-codigo:
      runs-on: self-hosted
      needs: empaquetar
      continue-on-error: true
      steps:
        - name: qa
          run: |
            mvn sonar:sonar -Dsonar.host.url=http://sonarqube:9000 -
Dsonar.qualitygate.wait=true -Dsonar.login=admin -Dsonar.password=admin1



    despliegue-heroku-pre:
      runs-on: self-hosted
      needs: pruebas-funcionales
      if: github.ref == 'refs/heads/main'
      steps:
        - name: release
          run: |
            export HEROKU_API_KEY=${{secrets.HEROKU_API_KEY}}
```

```
            mvn heroku:deploy-war -Dheroku.appName=baloncestorelease -
DskipTests=true


    despliegue-heroku:
      runs-on: self-hosted
      needs: despliegue-heroku-pre
      if: github.ref == 'refs/heads/main'
      steps:
        - name: deploy
          run: |
            export HEROKU_API_KEY=${{secrets.HEROKU_API_KEY}}
            mvn heroku:deploy-war -Dheroku.appName=baloncestogithub -
DskipTests=true
```

8.9. Aumentar la exigencia de calidad de código

Como garantía de calidad, el código fuente completo del proyecto debe tener un límite de problemas importantes (major issues) de 20 calculados con SonarQube, en otro caso no debe poderse desplegar la aplicación a producción.

Para realizar esto, primero agregamos nuestra nueva condición en Sonarqube.



Modificamos el archivo main.yaml para pausar el despliegue cuando el job de calidad se cumpla.

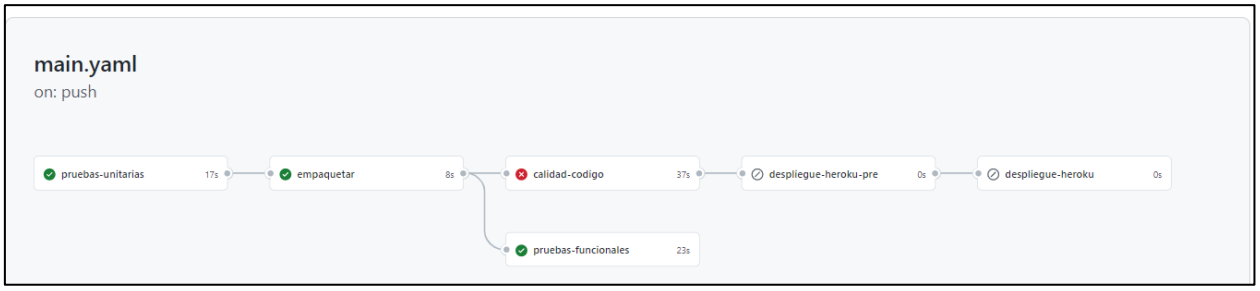
Archivo C:\Baloncesto\.github\workflows\main.yaml

```
name: CI


on:
  push:
    branches:
      - '**'
```

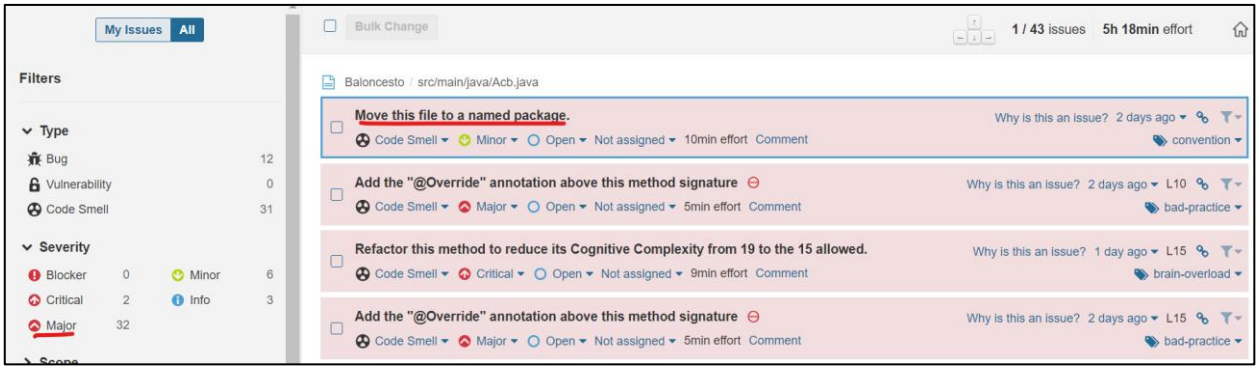
```
jobs:
  pruebas-unitarias:
    runs-on: self-hosted
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2.3.3
      - name: unit tests
        run:  |
          mysql -u root < db/baloncesto.sql
          export DATABASE_HOST="jdbc:mysql://localhost"
          export DATABASE_PORT="3306"
          export DATABASE_NAME="baloncesto"
          export DATABASE_USER="usuario"
          export DATABASE_PASS="clave"
          mvn test



  empaquetar:
    runs-on: self-hosted
    needs: pruebas-unitarias
    steps:
      - name: build
        run: mvn package -DskipTests=true


  pruebas-funcionales:
     runs-on: self-hosted
     needs: empaquetar
     steps:
       - name: test
         run: |
          mysql -u root < db/baloncesto.sql
          export DATABASE_HOST="jdbc:mysql://localhost"
          export DATABASE_PORT="3306"
          export DATABASE_NAME="baloncesto"
          export DATABASE_USER="usuario"
          export DATABASE_PASS="clave"
          cp -r target/Baloncesto /usr/local/tomcat/webapps
```

```
            sh /usr/local/tomcat/bin/catalina.sh stop
            sh /usr/local/tomcat/bin/catalina.sh start
            mvn failsafe:integration-test failsafe:verify



    calidad-codigo:
      runs-on: self-hosted
      needs: empaquetar
      continue-on-error: false
      steps:
        - name: qa
          run: |
            mvn sonar:sonar -Dsonar.host.url=http://sonarqube:9000 -
Dsonar.qualitygate.wait=true -Dsonar.login=admin -Dsonar.password=admin1



    despliegue-heroku-pre:
      runs-on: self-hosted
      needs: calidad-codigo
      if: github.ref == 'refs/heads/main'
      steps:
        - name: release
          run: |
            export HEROKU_API_KEY=${{secrets.HEROKU_API_KEY}}
            mvn heroku:deploy-war -Dheroku.appName=baloncestorelease -
DskipTests=true


    despliegue-heroku:
      runs-on: self-hosted
      needs: despliegue-heroku-pre
      if: github.ref == 'refs/heads/main'
      steps:
        - name: deploy
          run: |
            export HEROKU_API_KEY=${{secrets.HEROKU_API_KEY}}
            mvn heroku:deploy-war -Dheroku.appName=baloncestogithub -
DskipTests=true
```

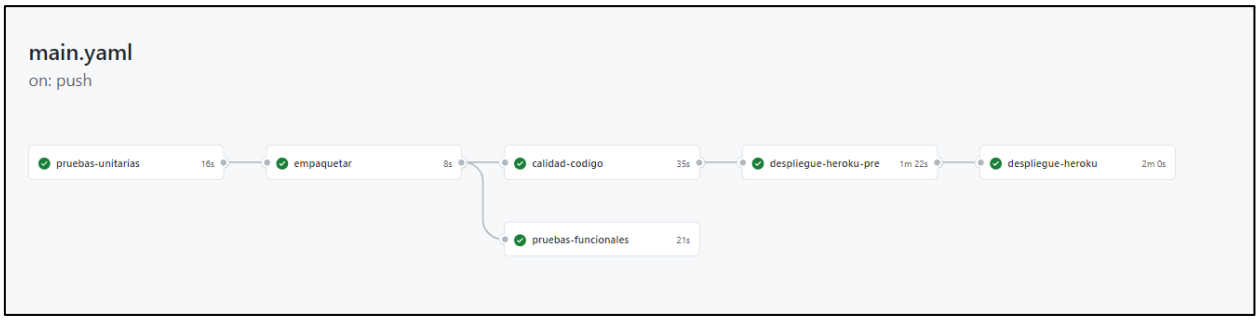
Sin embargo, Podemos ver que nuestro workflow, falla debimos a que no pasamos la prueba de calidad de código.



Nos dirigimos a sonarqube, a la sección issues, y podemos visualizar que tenemos 32 issues. Por lo que debemos editar nuestro código donde se nos indique.



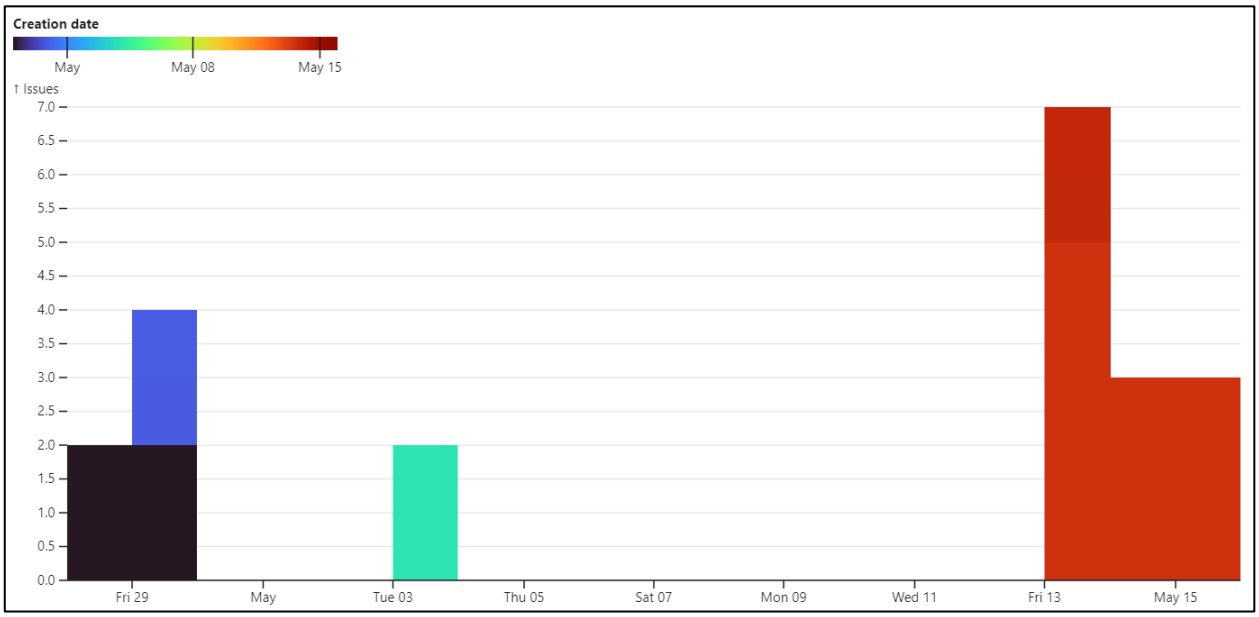
Una vez hecho los cambios, ejecutamos el worklow y vemos que se ejecuta con éxito.

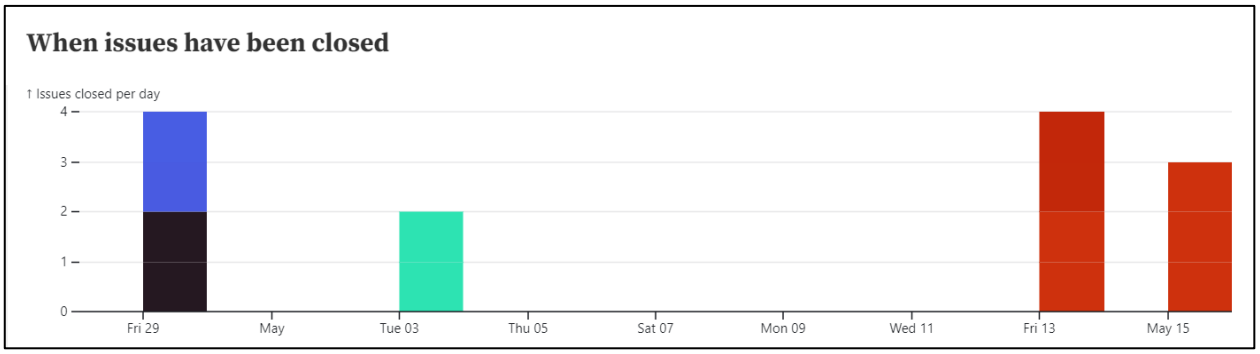


8.10. Cierre de sprint

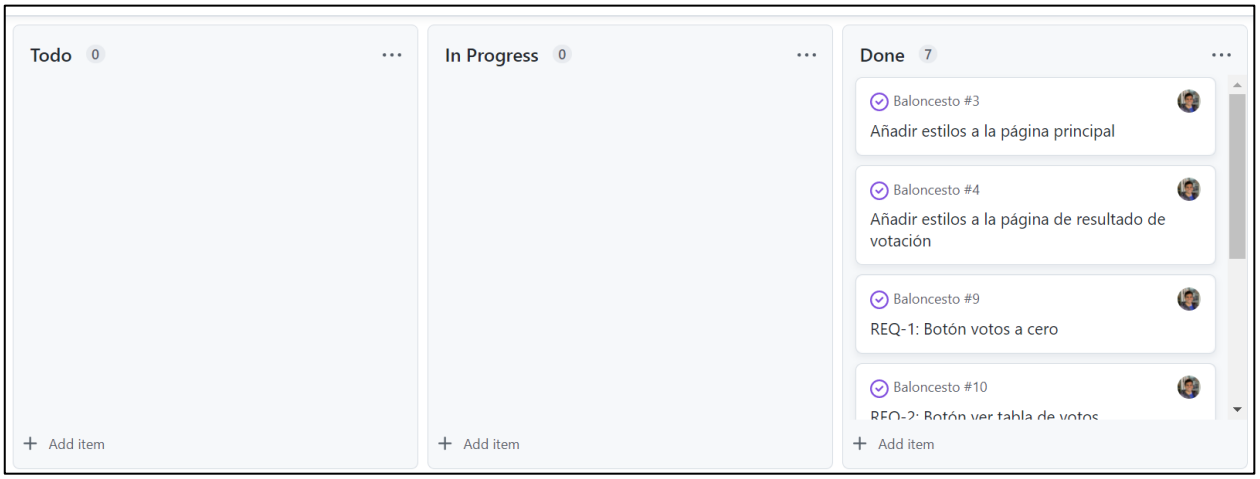
Una vez se han completado todas las issues, se procede a visualizar la aplicación en producción.

Además, procedemos a visualizar nuestro burndown chart.





También podemos observar que todas las issues de nuestro proyecto han sido cerradas.



9. CONCLUSIONES

Después de haber simulado el desarrollo de una aplicación web, se recomienda utilizar Github Actions como servidor de integración continua en la materia de Integración Continua en el Desarrollo Ágil por los siguientes motivos:

- Fácil uso, no se necesita de una gran configuración para poder usarlo.
- Integración directa con el repositorio de código.
- Todo el workflow se concentra en un solo fichero yaml.
- El workflow incluye eventos, es decir, los jobs que se activan se pueden condicionar a ciertas ramas.
- Permite usar runners propios en local o en contendedores, facilitando la prueba y la configuración de otras dependencias.

10. BIBLIOGRAFÍA

[1] Amazon Web Services, Inc. 2022. Integración continua del software | Pruebas automatizadas | AWS. [online] Available at: <https://aws.amazon.com/es/devops/continuous-integration> [Accessed 15 May 2022].

[2] Hilera, J.R., 2021. Práctica guiada. Desarrollo ágil de una aplicación web con integración continua. Universidad de Alcalá. https://github.com/josehilera/integracion-continua.

[3] n.d. [online] Available at: <https://www.java.com/es/download/help/whatis_java.html> [Accessed 15 May 2022].

[4] Code.visualstudio.com. n.d. Documentation for Visual Studio Code. [online] Available at: <https://code.visualstudio.com/docs> [Accessed 15 May 2022].

[5] Porter, B., Zyl, J. and Lamy, O., 2022. Maven – Welcome to Apache Maven. [online] Maven.apache.org. Available at: <https://maven.apache.org/> [Accessed 15 May 2022].

[6] Git-scm.com. 2022. Git. [online] Available at: <https://git-scm.com/> [Accessed 15 May 2022].

[7] Platzi. 2021. Qué es GitHub y cómo usarlo para aprovechar sus beneficios. [online] Available at: <https://platzi.com/blog/que-es-github-como-funciona/> [Accessed 15 May 2022].

[8] n.d. [online] Available at: <https://docs.github.com/es/issues/trying-out-the-new-projects-experience/about-projects> [Accessed 15 May 2022].

[9] n.d. [online] Available at: <https://docs.github.com/es/actions/learn-github-actions/understanding-github-actions?learn=getting_started&learnProduct=actions> [Accessed 15 May 2022].

[10] Docker Documentation. n.d. Get Docker. [online] Available at: <https://docs.docker.com/get-docker/> [Accessed 15 May 2022].

[11] Junit.org. n.d. JUnit – About. [online] Available at: <https://junit.org/junit4/> [Accessed 15 May 2022].

[12] Selenium. n.d. The Selenium Browser Automation Project. [online] Available at: <https://www.selenium.dev/documentation/> [Accessed 15 May 2022].

[13] n.d. [online] Available at: <https://docs.sonarqube.org/latest/> [Accessed 15 May 2022].

[14] n.d. [online] Available at: <https://www.nts-solutions.com/blog/heroku-que-es.html> [Accessed 15 May 2022].