

Trabajo Final

Mohamed Hamen El Majjaoui El Majjaoui

Asignatura: Integración Continua en el Desarrollo Ágil

Máster Universitario en Desarrollo Ágil de Software para la Web

Contenido

Creación del entorno de pre-producción	3
Creación del nuevo sprint (milestone)	6
Creación de las user stories (issues).....	7
Creación de las ramas necesarias	8
Implementación de la tarea REQ-1	9
Implementación de la tarea PU	12
Preparación de la rama ver-votos	13
Implementación de la tarea REQ-2	15
Implementación de la tarea PF-A	18
Implementación de la tarea PF-B	19
Merge de la rama ver-votos a master.....	20
Actualización de SonarQube	21

Creación del entorno de pre-producción

Al igual que se hizo en la práctica asociada al tema 5 de la asignatura, debemos crear una nueva App Service en la que se desplegará el nuevo trabajo que añadiremos al archivo **main.yml** del workflow.

Para ello, usaremos los mismos datos que usamos para crear el entorno de producción le asignaremos un nombre diferente:

[Inicio](#) > [App Services](#) >

Crear aplicación web ...

[Datos básicos](#) [Base de datos](#) [Implementación](#) [Redes](#) [Supervisión](#) [Etiquetas](#) [Revisar y crear](#)

App Service Web Apps le permite generar, implementar y escalar rápidamente aplicaciones empresariales web, móviles y de API que se ejecutan en cualquier plataforma. Satisfaga los estrictos requisitos de rendimiento, escalabilidad, seguridad y cumplimiento sin renunciar a una plataforma totalmente administrada para el mantenimiento de la infraestructura. [Más información](#)

Detalles del proyecto

Seleccione una suscripción para administrar los recursos implementados y los costos. Use los grupos de recursos como carpetas para organizar y administrar todos los recursos.

Suscripción * ⓘ

Grupo de recursos * ⓘ
[Crear nuevo](#)

Detalles de instancia

Nombre * [.azurewebsites.net](#)

Publicar * ☒ Código ☐ Contenedor Docker ☐ Aplicación web estática

Pila del entorno en tiempo de ejecución *

Pila de servidor web Java *

Sistema operativo * ☒ Linux ☐ Windows

Región *

i ¿No encuentra su plan de App Service? Pruebe otra región o seleccione su App Service Environment.

Planes de precios

El plan de tarifa de App Service determina la ubicación, las características, los costos y los recursos del proceso asociados a la aplicación. [Más información](#)

Plan de Linux (West Europe) * ⓘ
[Crear nuevo](#)

Plan de precios **Gratis F1** (Infraestructura compartida)

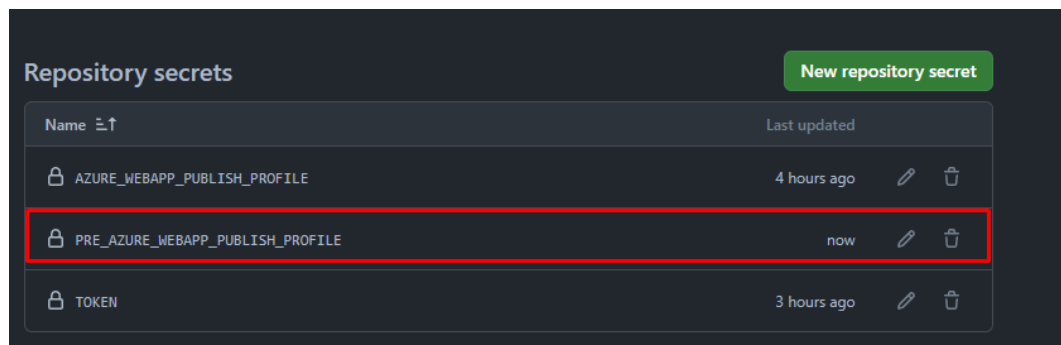
Al acabar el proceso de creación, se sumará a la lista de App Services, junto con la que ya teníamos:

App Services <small>Universidad de Alcalá (a385uah.es)</small>						
+ Crear Administrar aplicaciones eliminadas Administrar vista Actualizar Exportar a CSV Abrir consulta Asignar etiquetas Iniciar Reiniciar Detener Eliminar						
Filtrar por cualquier campo Suscripción es igual a todo Grupo de recursos es igual a todo Ubicación es igual a todo Agregar filtro						
Mostrando de 1 a 2 de 2 registros						
Nombre	Estado	Ubicación	Plan de tarifa	Plan de App Service	Suscripción	Tipo de
<input type="checkbox"/> baloncesto-majjaoui	En ejecución	West Europe	Gratis	ASP-baloncestomajjaouigroup-a168	Azure for Students	Aplicación web
<input type="checkbox"/> pre-baloncesto-majjaoui	En ejecución	West Europe	Gratis	ASP-baloncestomajjaouigroup-a168	Azure for Students	Aplicación web

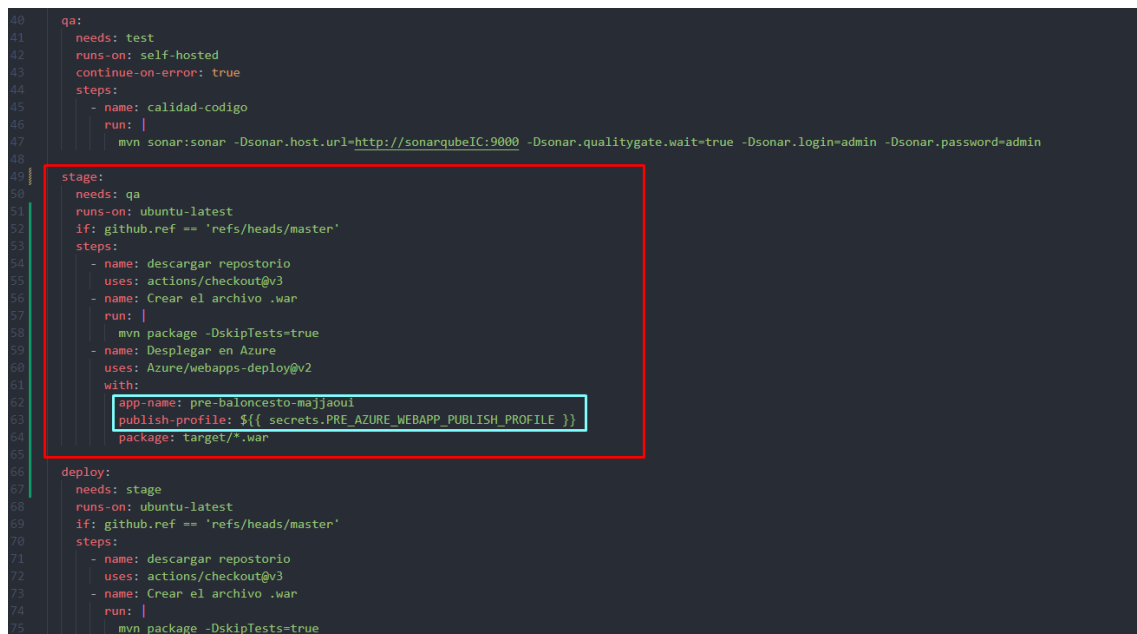
Una vez creada la nueva App Service, descargaremos su perfil de publicación para crear el token que necesitaremos para el archivo **main.yml**.



Al descargarlo, copiamos su contenido y creamos un nuevo **repository secret**, al que llamaremos **PRE_AZURE_WEBAPP_PUBLISH_PROFILE**



Con el nuevo perfil de publicación añadido al repositorio, ya podemos añadir el nuevo trabajo al workflow. Para ello, usaremos la misma estructura que el trabajo **deploy** pero eliminando una sección:



Para el trabajo **stage** eliminaremos la aprobación manual, ya que funcionaría como un entorno de pruebas.

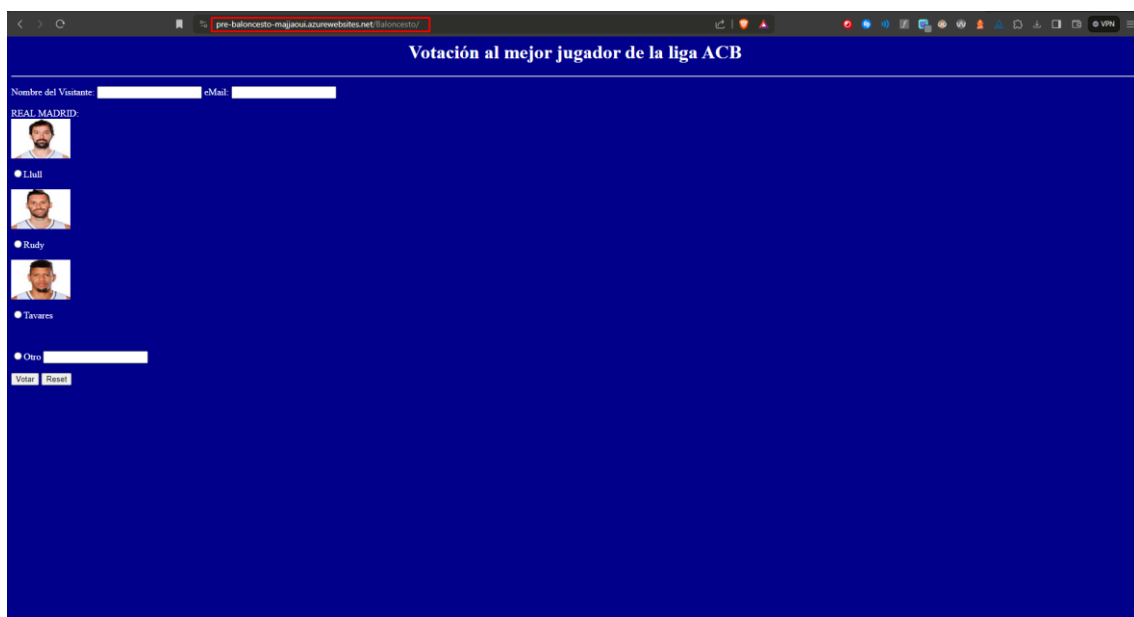
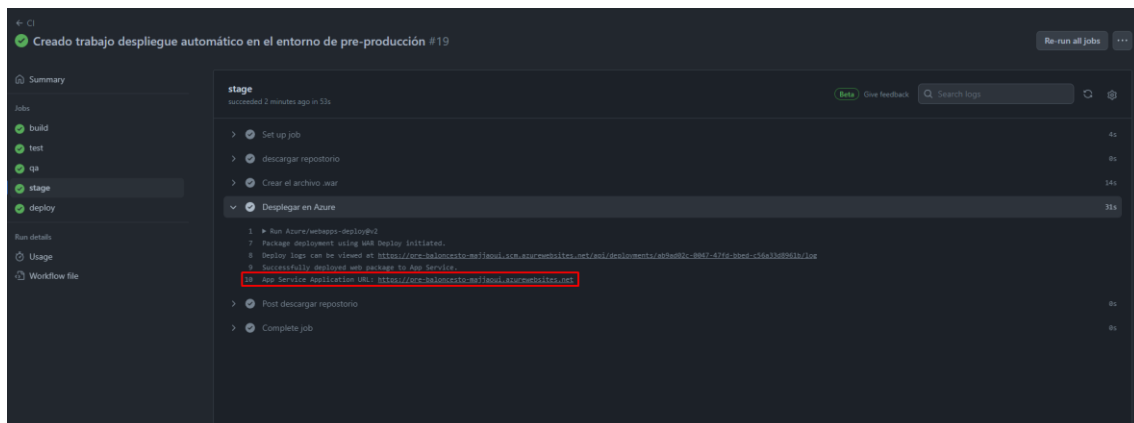
Una vez editado el archivo main.yml, lo subimos al repositorio y aprobamos el despliegue

```
PS F:\UNI\IC\repoUnidad5\baloncesto> git checkout master
Already on 'master'
Your branch is up to date with 'origin/master'.
PS F:\UNI\IC\repoUnidad5\baloncesto> git add .
PS F:\UNI\IC\repoUnidad5\baloncesto> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>.." to unstage)
        modified:   .github/workflows/main.yml

PS F:\UNI\IC\repoUnidad5\baloncesto> git commit -m "Creado trabajo despliegue automático en el entorno de pre-producción"
[master 262be75] Creado trabajo despliegue automático en el entorno de pre-producción
1 file changed, 18 insertions(+), 1 deletion(-)
PS F:\UNI\IC\repoUnidad5\baloncesto> git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 591 bytes | 591.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/mhem-space/baloncesto.git
   4a20fa4..262be75  master -> master
PS F:\UNI\IC\repoUnidad5\baloncesto>
```

Una vez se ejecute el trabajo de stage, podemos observar que se ha desplegado correctamente en la nueva App Service:



Creación del nuevo sprint (milestone)

Para crear el nuevo sprint, llamado **milestone** en GitHub, seguiremos los pasos de la práctica 5 usando los siguientes datos:

New milestone

Create a new milestone to help organize your issues and pull requests. Learn more about [milestones and issues](#).

Title

Sprint práctica final

Due date (optional)

dd/mm/aaaa

Description

Se desarrollarán las funcionalidad de "Borrar votos" y "Ver votos", además de los tests necesarios

Create milestone

Una vez creado, aparecerá en la sección de **Issues > Milestones**

mhem-space / baloncesto

Search

Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

Labels

Milestones

New milestone

1 Open 1 Closed

Sort

Sprint práctica final

No due date Last updated less than a minute ago

0% complete 0 open 0 closed

Edit Close Delete

Se desarrollarán las funcionalidad de "Borrar votos" y "Ver votos", además de los tests necesarios

Creación de las user stories (issues)

Para la creación de las historias de usuario, llamadas **issues** en GitHub, accederemos al milestone creado en el paso anterior y crearemos una issue para cada tarea a realizar rellenando los siguientes datos:

Add a title

REQ-1 | Crear botón para poner votos a cero

Add a description

Write

Preview

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.

Markdown is supported

Paste, drop, or click to add files

Submit new issue

Assignees

mhem-space

Labels

None yet

Projects

Baloncesto

Milestone

Sprint práctica final

Development

Shows branches and pull requests linked to this issue.

Helpful resources

GitHub Community Guidelines

Una vez creados los cinco issues, obtendremos esta lista en la sección **Issues**

Filters

is:issue is:open

Labels 9

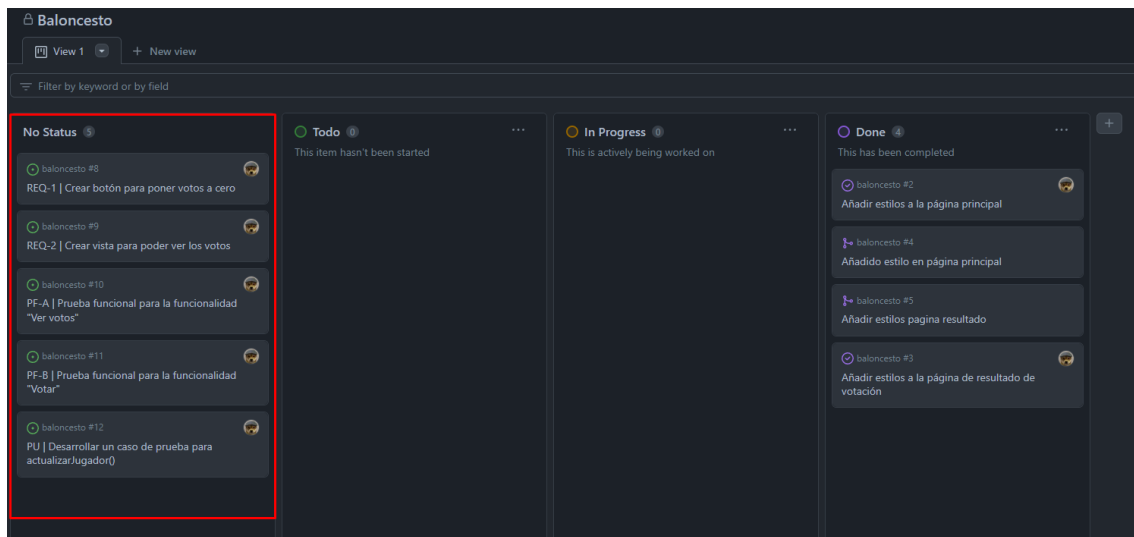
Milestones 1

New issue

<input type="checkbox"/>	5 Open	5 Closed	Author	Label	Projects	Milestones	Assignee	Sort
<input type="checkbox"/>	PU Desarrollar un caso de prueba para actualizarJugador()	#12 opened now by mhem-space Sprint práctica fi...						
<input type="checkbox"/>	PF-B Prueba funcional para la funcionalidad "Votar"	#11 opened 2 minutes ago by mhem-space Sprint práctica fi...						
<input type="checkbox"/>	PF-A Prueba funcional para la funcionalidad "Ver votos"	#10 opened 4 minutes ago by mhem-space Sprint práctica fi...						
<input type="checkbox"/>	REQ-2 Crear vista para poder ver los votos	#9 opened 9 minutes ago by mhem-space Sprint práctica fi...						
<input type="checkbox"/>	REQ-1 Crear botón para poner votos a cero	#8 opened 10 minutes ago by mhem-space Sprint práctica fi...						

ProTip! Click a checkbox on the left to edit multiple issues at once.

Habiendo creado los issues, deberán aparecer en el tablero Kanban:



Creación de las ramas necesarias

El siguiente paso para realizar antes de comenzar a desarrollar es la creación de las ramas que usaremos. Necesitaremos dos ramas:

- **Borrar-votos:** en la que se implementarán las tareas REQ-1 y PU
- **Ver-votos:** en la que se implementarán las tareas REQ-2, PF-A y PF-B.

La creación de las ramas se hace usando el comando **git Branch <Nombre_rama>**:

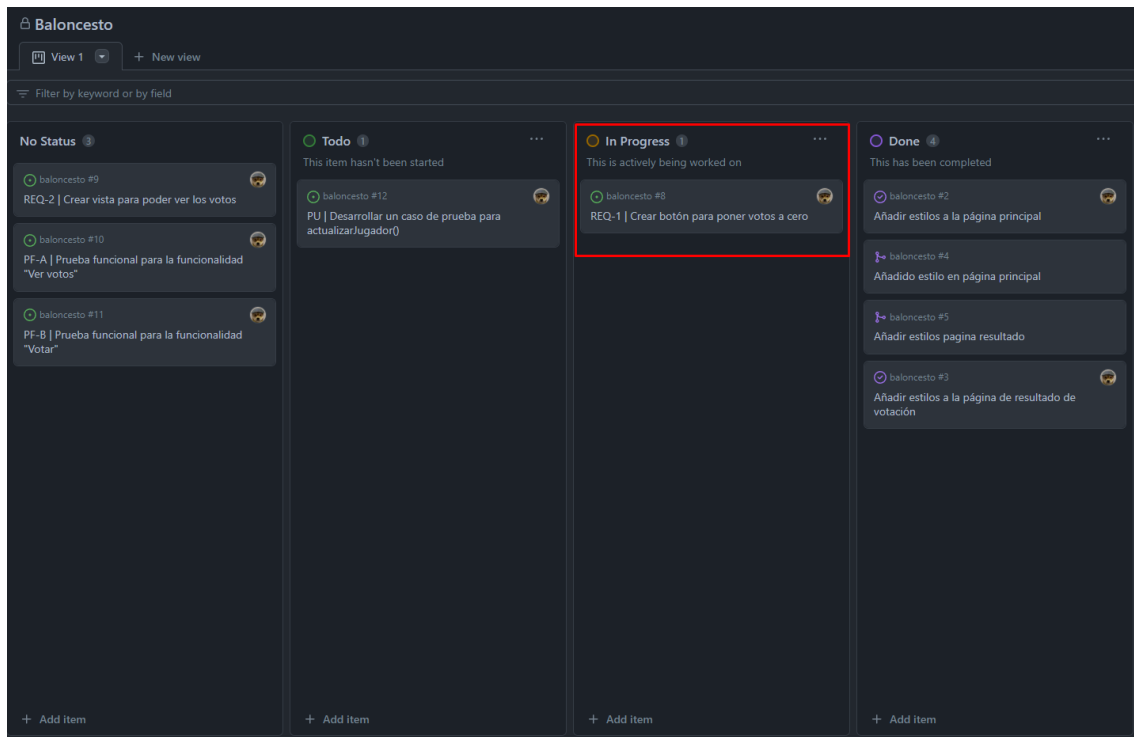
```
PS F:\UNI\IC\repoUnidad5\baloncesto> git branch borrar-votos
PS F:\UNI\IC\repoUnidad5\baloncesto> git branch ver-votos
PS F:\UNI\IC\repoUnidad5\baloncesto> git branch
añadir-estilos-pagina-principal
añadir-estilos-pagina-resultado
borrar-votos
* master
ver-votos
PS F:\UNI\IC\repoUnidad5\baloncesto>
```

Mediante el comando **git branch** podemos observar que estamos actualmente en la rama master, por lo que antes de empezar a desarrollar debemos cambiar a la rama correspondiente. La primera rama con la que trabajaremos será **borrar-votos**, por lo que ejecutaremos el siguiente comando:

```
PS F:\UNI\IC\repoUnidad5\baloncesto> git checkout borrar-votos
Switched to branch 'borrar-votos'
```


Implementación de la tarea REQ-1

Antes de comenzar a desarrollar, debemos mover en el kanban la tarea REQ-1 del estado **Todo** a **In Progress**:



Para la tarea REQ-1 debemos ofrecer al usuario un botón “Poner votos a cero” en la página principal, que al pulsarlo se pongan a 0 los votos de todos los jugadores en la base de datos.

Para ello, realizaremos los siguientes pasos:

- **Creación del nuevo botón:** en la página **index.html** crearemos un botón en el mismo formulario que ya hay creado. El botón hará una llamada al controlador **Acb**, que se encargará de ejecutar el proceso adecuado.
- **Creación del método `resetearVotos()`:** en el archivo **ModeloDatos.java**, que contiene los métodos que se comunican con la base de datos, crearemos el método `resetearVotos()` para cambiar el valor del número de votos de todos los jugadores a cero mediante la ejecución de la query **UPDATE Jugadores SET votos=0**
- **Editar el controlador:** debemos incluir en el controlador la opción de que el botón que se pulse sea el nuevo botón de reiniciar votos a cero. Para ello comprobaremos si el nombre del botón aparece en los parámetros de la petición. En caso de que el botón haya sido pulsado, llamaremos al método creado en nuestro repositorio **ModeloDatos.java**. Además, incluiremos una cookie en la respuesta, para poder avisar al usuario de que los votos han sido reiniciados.
- **Lectura de la cookie:** en el archivo **index.html**, debemos incluir una nueva sección **script** que contendrá dos funciones para obtener la cookie creada en el controlador y crear una alerta con el contenido de la cookie.

```

    public void resetearVotos() {
        try {
            set = con.createStatement();
            set.executeUpdate("UPDATE Jugadores SET votos=0");
            rs.close();
            set.close();
        } catch (Exception e) {
            // No modifica la tabla
            System.out.println("Error al resetear votos: " + e.getMessage());
        }
    }
}

```

```

public void service(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
    HttpSession s = req.getSession(true);

    if (req.getParameter("resetVotes") != null) {
        bd.resetearVotos();
        // Crear una cookie para indicar que se han eliminado los votos.
        Cookie mensajeCookie = new Cookie("mensajeVotos", "Todos los votos han sido eliminados.");
        mensajeCookie.setMaxAge(60); // Expira en 60 segundos
        res.addCookie(mensajeCookie);

        res.sendRedirect("index.html"); // Redirige de vuelta a la página principal.
        return; // Finaliza la ejecución para no procesar más código.
    }

    String nombreP = (String) req.getParameter("txtNombre");
    String nombre = (String) req.getParameter("R1");
    if (nombre.equals("Otros")) {
        nombre = (String) req.getParameter("txtOtros");
    }
    if (bd.existeJugador(nombre)) {
        bd.actualizarJugador(nombre);
    } else {
        bd.insertarJugador(nombre);
    }
}

```

```

<p align="left">
    <input type="submit" name="B1" value="Votar" />
    <input type="reset" name="B2" value="Reset" />
    <input type="submit" name="resetVotes" value="Poner votos a cero" />
</p>
</form>
</body>
<script>
window.onload = function () {
    var mensaje = getCookie("mensajeVotos");
    if (mensaje != "") {
        alert(mensaje.replace(/\+/g, ' '));
        // Eliminar la cookie después de mostrar el mensaje
        document.cookie = "mensajeVotos=; max-age=0";
    }
};

function getCookie(nombre) {
    var name = nombre + "=";
    var decodedCookie = decodeURIComponent(document.cookie);
    var ca = decodedCookie.split(";");
    for (var i = 0; i < ca.length; i++) {
        var c = ca[i];
        while (c.charAt(0) == " ") {
            c = c.substring(1);
        }
        if (c.indexOf(name) == 0) {
            return c.substring(name.length, c.length);
        }
    }
    return "";
}
</script>

```

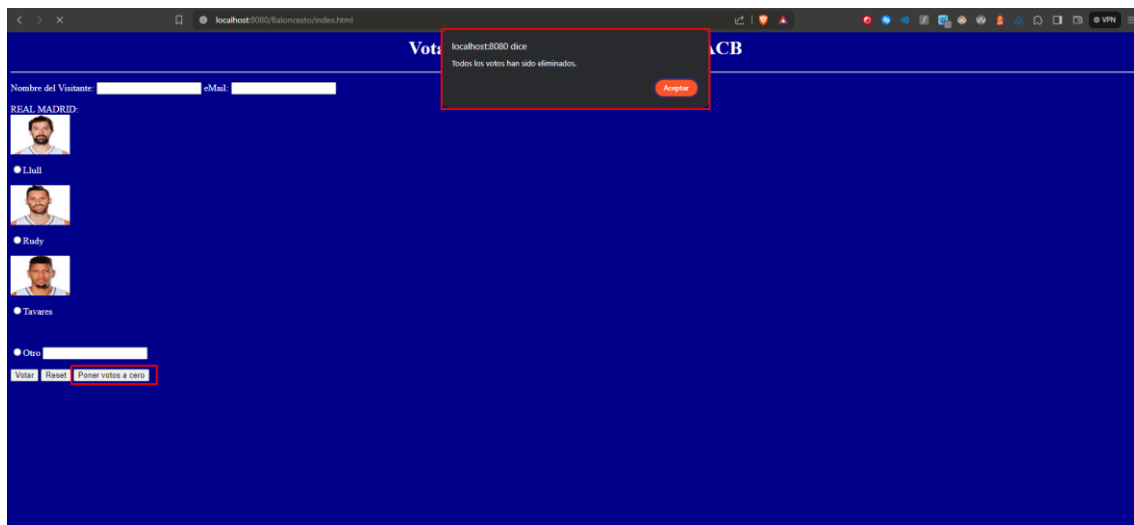
Acabado el desarrollo de la tarea, lo subiremos a la rama correspondiente del repositorio:

```
PS F:\UNI\IC\repoUnidad5\baloncesto> git add .
PS F:\UNI\IC\repoUnidad5\baloncesto> git status
On branch borrar-votos
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   src/main/java/Acb.java
    modified:   src/main/java/ModeloDatos.java
    modified:   src/main/webapp/index.html

PS F:\UNI\IC\repoUnidad5\baloncesto> git commit -m "Añadido botón para resetar los votos a cero"
[borrar-votos 17cefb8] Añadido botón para resetar los votos a cero
 3 files changed, 94 insertions(+), 35 deletions(-)
 rewrite src/main/webapp/index.html (88%)
PS F:\UNI\IC\repoUnidad5\baloncesto> git push origin borrar-votos
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 16 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 1.82 KiB | 1.82 MiB/s, done.
Total 9 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
remote:
remote: Create a pull request for 'borrar-votos' on GitHub by visiting:
remote:   https://github.com/mhem-space/baloncesto/pull/new/borrar-votos
remote:
To https://github.com/mhem-space/baloncesto.git
 * [new branch]      borrar-votos -> borrar-votos
```

Si observamos el workflow ejecutado, podemos ver que los trabajos **stage** y **deploy** no se han ejecutado debido a que no hemos insertado en la rama master.

Podemos observar los cambios realizados en nuestro entorno local:



Como hemos acabado la implementación de la tarea, debemos cerrar el issue y mover la tarea en el tablero Kanban a la sección **Done**.

Implementación de la tarea PU

Comenzaremos moviendo la tarea **PU** en el tablero Kanban a la sección **In Progress**.

Para esta tarea, debemos 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 realizar la prueba, necesitaremos realizar una simulación de la base de datos. La herramienta que hemos elegido para ello es **Mockito**, por lo que el primer paso será incluir la dependencia en el archivo **pom.xml**:

```
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-core</artifactId>
  <version>5.10.0</version>
  <scope>test</scope>
</dependency>
```

Lo siguiente, será añadir un método **setter** en nuestro repositorio **ModeloDatos.java** para poder sustituir la conexión de la instancia de la base de datos por la simulación de mockito:

```
public void setConexion(Connection con) {
    this.con = con;
}
```

Finalmente, crearemos la prueba en el archivo **ModeloDatosTest.java**:

```
@Test
public void testActualizarJugador() throws Exception {
    System.out.println("Prueba de actualizarJugador");

    // Simular las dependencias de la base de datos
    Connection mockConnection = mock(Connection.class);
    Statement mockStatement = mock(Statement.class);

    // Configurar el comportamiento simulado
    when(mockConnection.createStatement()).thenReturn(mockStatement);

    // Inyectar las dependencias simuladas en la instancia de ModeloDatos
    ModeloDatos instance = new ModeloDatos();

    // Le pasamos la conexión simulada
    instance.setConexion(mockConnection);

    instance.abrirConexion(); // Se llama al método sobrescrito que establece la conexión simulada

    String nombreJugador = "Rudy";
    // Ejecutar el método a probar
    instance.actualizarJugador(nombreJugador);

    // Verificar que el método executeUpdate fue llamado con la consulta SQL esperada
    verify(mockStatement, times(1)).executeUpdate("UPDATE Jugadores SET votos=votos+1 WHERE nombre = " + " LIKE '%" + nombreJugador + "%'");

    // Limpiar el estado de la conexión simulada
    instance.cerrarConexion();
}
```

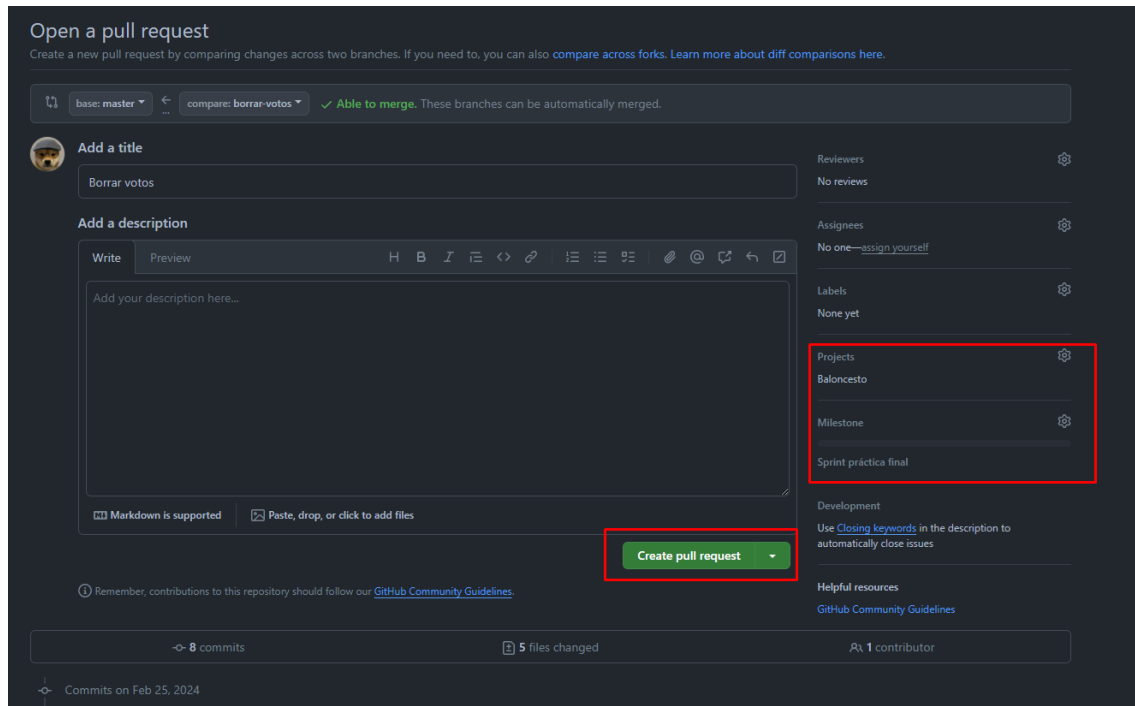
Su funcionamiento es el siguiente: configuramos la simulación de mockito, instanciamos la base de datos, sustituimos la conexión de la base de datos por la simulación de mockito, abrimos la conexión, llamamos al método **actualizarJugador()** pasándole como parámetro el valor “Rudy”, se comprueba el resultado de la query ejecutada en el método **actualizarJugador**. Si la query afecta a una fila, el test devuelve true. En caso contrario, devuelve false y, por lo tanto, el test falla.

Con esto, acabamos la tarea y debemos cerrar el issue correspondiente y mover la tarea en el tablero Kanban a **Done**.

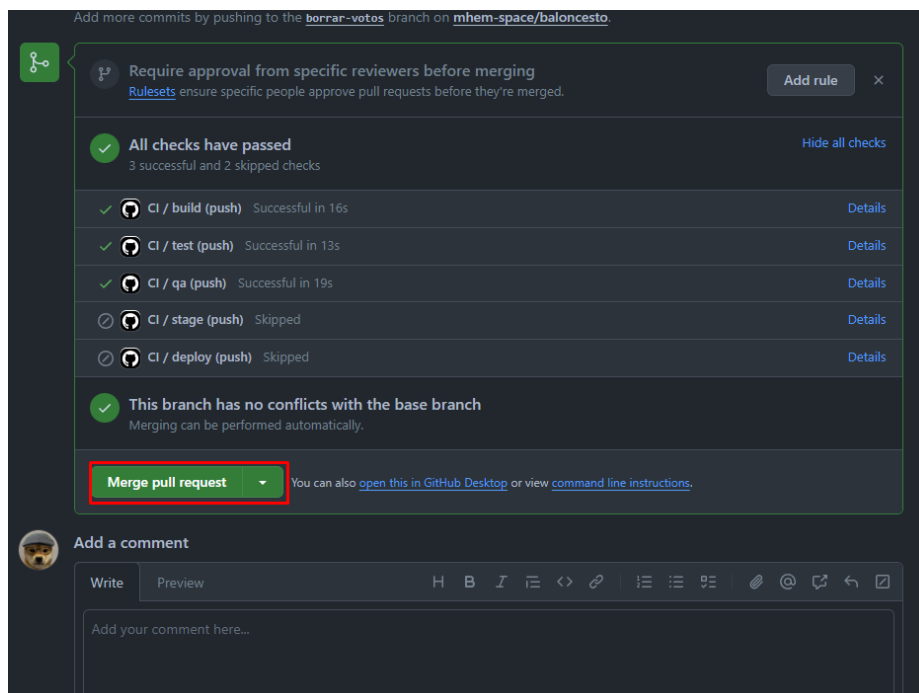
Preparación de la rama ver-votos

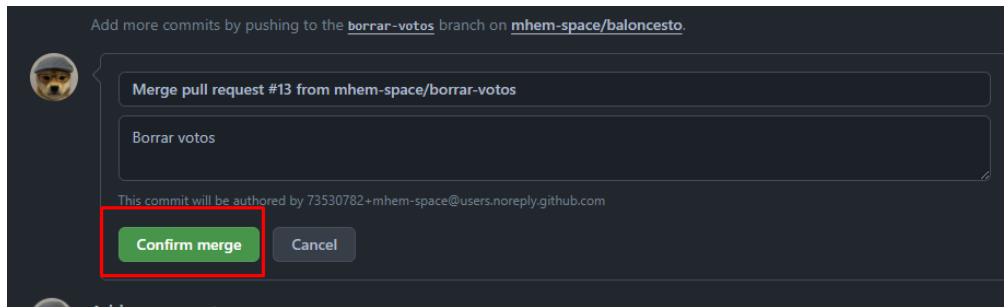
Antes de comenzar a trabajar en la otra rama, debemos fusionar la rama **borrar-votos** con **master** y luego actualizar la rama **ver-votos** para empezar a trabajar con todo el código actualizado. Aprovechando este merge, podemos solucionar algunos errores que detecta sonar en el nuevo código, como tags deprecated, duplicidad de código, alternativas más seguras a ciertas funciones, etc.

Al crear la solicitud de merge podemos observar que no se ha encontrado ningún conflicto:

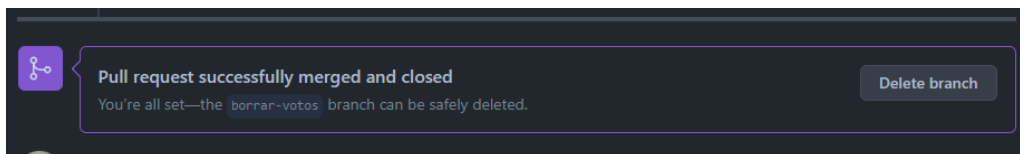


Una vez creada la pull request debemos aceptarla para que se fusionen las ramas:





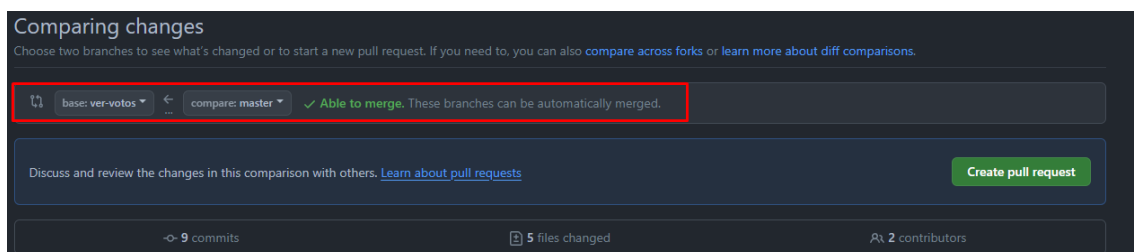
Una vez confirmado el merge, se nos indicará que la petición de pull ya ha sido cerrada:



Una vez actualizado el código de la rama master, debemos publicar la rama ver-votos y así poder hacer el merge usando la interfaz de usuario de GitHub. Para ello usaremos el comando **git push**

```
PS F:\UNI\IC\repoUnidad5\baloncesto> git push -u origin ver-votos
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'ver-votos' on GitHub by visiting:
remote:   https://github.com/mhem-space/baloncesto/pull/new/ver-votos
remote:
To https://github.com/mhem-space/baloncesto.git
 * [new branch]      ver-votos -> ver-votos
Branch 'ver-votos' set up to track remote branch 'ver-votos' from 'origin'.
```

Habiendo subido la rama, debemos repetir el proceso anterior para fusionar la rama master en la rama ver-votos:

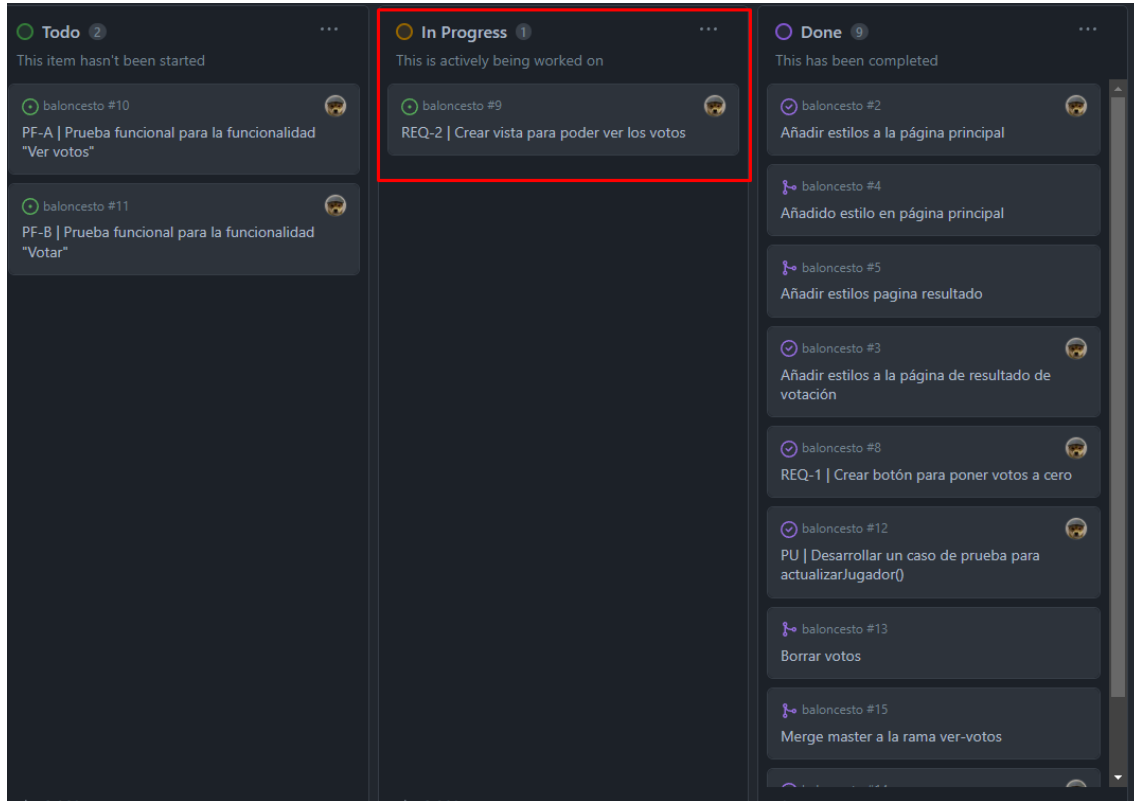


Finalizado el merge, debemos actualizar la rama ver-votos que tenemos en nuestro entorno local mediante el comando **git pull**

```
PS F:\UNI\IC\repoUnidad5\baloncesto> git pull origin ver-votos
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), 906 bytes | 453.00 KiB/s, done.
From https://github.com/mhem-space/baloncesto
 * branch                ver-votos    -> FETCH_HEAD
 262be75..bfb0e26  ver-votos    -> origin/ver-votos
Updating 262be75..bfb0e26
Fast-forward
 pom.xml                  | 6 +++
 src/main/java/Acb.java   | 14 ++++++
 src/main/java/ModeloDatos.java | 32 ++++++++-----
 src/main/webapp/index.html | 81 ++++++++++++++++++++++++++++++++++++++-----
 src/test/java/ModeloDatosTest.java | 38 ++++++++-----
 5 files changed, 137 insertions(+), 34 deletions(-)
```

Implementación de la tarea REQ-2

Comenzaremos por actualizar el tablero Kanban moviendo la tarea a la sección **In progress** y el resto de las tareas que haremos en esta rama las dejaremos en la sección **Todo**:



Como nuevo requisito de la aplicación, la página principal debe ofrecer al usuario un botón “Ver votos”, que al pulsarlo muestre una nueva página, 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.

Para ello, realizaremos los siguientes pasos:

- **Creación del nuevo botón:** en la página **index.html** crearemos un botón en el mismo formulario que ya hay creado. El botón hará una llamada al controlado Acb, que se encargará de ejecutar el proceso adecuado.

```
<p style="text-align: left;">
  <input type="submit" name="B1" value="Votar" />
  <input type="reset" name="B2" value="Reset" />
  <input id="borrarVotos" type="submit" name="resetVotes" value="Poner votos a cero" />
  <input id="ver" type="submit" name="verVotos" value="Ver votos" />
</p>
</form>
```

- **Creación del método getJugadores():** en el archivo **ModeloDatos.java**, que contiene los métodos que se comunican con la base de datos, crearemos el método getJugadores() para obtener los datos de todos los jugadores que hay en la base de datos mediante la ejecución de la query **SELECT * FROM Jugadores**

```
public List<Jugador> getJugadores() {
    List<Jugador> jugadores = new ArrayList<>();
    try {
        set = con.createStatement();
        rs = set.executeQuery("SELECT * FROM Jugadores");
        while (rs.next()) {
            jugadores.add(new Jugador(rs.getInt("id"), rs.getString("nombre"), rs.getInt("votos")));
        }
    } catch (Exception e) {
        logger.info("Error al obtener jugadores: " + e.getMessage());
    }
    return jugadores;
}
```

- **Editar el controlador:** debemos incluir en el controlador la opción de que el botón que realice la petición sea el nuevo botón creado. Para ello comprobaremos si el nombre del botón aparece en los parámetros de la petición. En caso de que el botón haya sido pulsado, llamaremos al método creado en nuestro repositorio **ModeloDatos.java**. El resultado que nos devuelva la base de datos se dividirá en dos listas (nombres y votos) que se añadirán como atributos a la respuesta del controlador.

```
mensajeCookie.setCookie(0); // Expira en 0 segundos
res.addCookie(mensajeCookie);

res.sendRedirect("index.html"); // Redirige de vuelta a la página principal.
return; // Finaliza la ejecución para no procesar más código.
}

if (req.getParameter("verVotos") != null) {
    List<String> nombres = new ArrayList<>();
    List<Integer> votos = new ArrayList<>();
    List<Jugador> datosJugadores = bd.getJugadores();

    for (Jugador jugador : datosJugadores) {
        nombres.add(jugador.getNombre());
        votos.add(jugador.getVotos());
    }

    s.setAttribute("nombres", nombres);
    s.setAttribute("votos", votos);

    res.sendRedirect("VerVotos.jsp"); // Redirige a la página correspondiente para ver los votos.
    return; // Finaliza la ejecución para no procesar más código.
}

String nombreP = (String) req.getParameter("txtNombre");
String nombre = (String) req.getParameter("R1");
if (nombre.equals("Otros")) {
    nombre = (String) req.getParameter("txtOtros");
}
```

- **Creación de la clase Jugador:** debemos crear la clase **Jugador.java** para mapear cada fila retornada por la base de datos a este tipo de dato que contendrá todos los campos de la tabla Jugador.


```

Jugador.java X
src > main > java > J Jugador.java > Jugador > getVotos()
1 public class Jugador {
2     private int id;
3     private String nombre;
4     private int votos;
5
6     public Jugador(int id, String nombre, int votos) {
7         this.id = id;
8         this.nombre = nombre;
9         this.votos = votos;
10    }
11
12    public int getId() {
13        return id;
14    }
15
16    public void setId(int id) {
17        this.id = id;
18    }
19
20    public String getNombre() {
21        return nombre;
22    }
23
24    public void setNombre(String nombre) {
25        this.nombre = nombre;
26    }
27
28    public int getVotos() {
29        return votos;
30    }
31
32    public void setVotos(int votos) {
33        this.votos = votos;
34    }
35 }
36

```

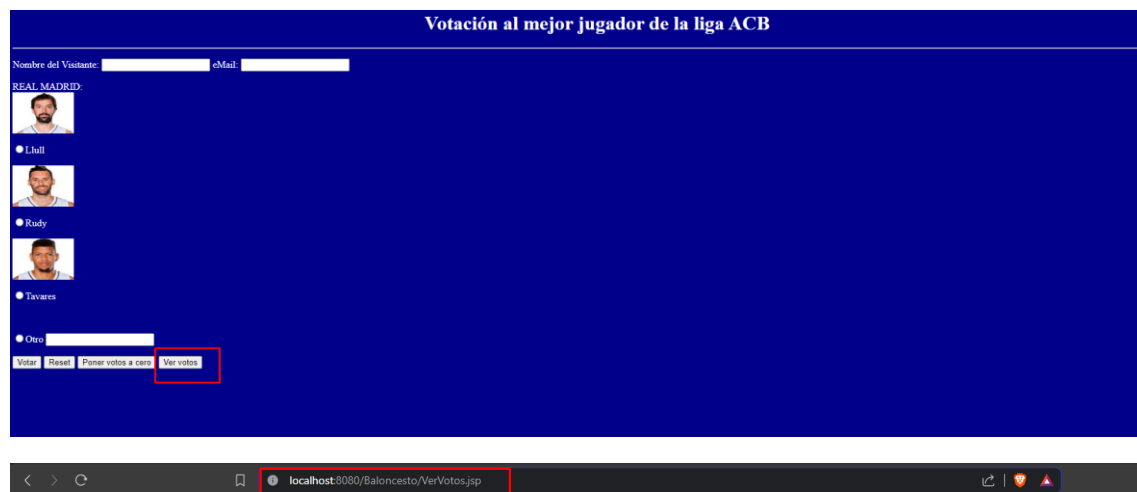
- **Creación de la nueva vista:** crearemos una nueva vista en la que se mostrará la tabla con dos columnas: nombre del jugador y número de votos. El nombre de la vista será **VerVotos.jsp**. En la vista debemos incluir un enlace para volver a la página principal.

```

VerVotos.jsp X
src > main > webapp > VerVotos.jsp
1 <%@ page import="java.util.List" %>
2 <!DOCTYPE html>
3 <html lang="es">
4 <head>
5     <title>Votos</title>
6     <link href="estilos.css" rel="stylesheet" type="text/css" />
7 </head>
8 <body class="verVotos">
9     <h1>Votos de los jugadores</h1>
10    <table id="tablaDeVotos">
11        <tr>
12            <th>Nombre del jugador</th>
13            <th>Número de votos</th>
14        </tr>
15        <% for (int i = 0; i < ((List<String>) session.getAttribute("nombres")).size(); i++) { %>
16            <tr>
17                <td><%= ((List<String>) session.getAttribute("nombres")).get(i)%></td>
18                <td style="text-align:center;"><%= ((List<Integer>) session.getAttribute("votos")).get(i) %></td>
19            </tr>
20        <% } %>
21    </table>
22    <a href="index.html"> Ir a la página principal</a>
23 </body>
24 </html>
25

```

Una vez acabado el desarrollo de la tarea debemos subirla a nuestro repositorio y poder probarla en nuestro entorno local.



Votos de los jugadores

Nombre del jugador Número de votos

Lluís	0
Rudy	1
Tavares	0

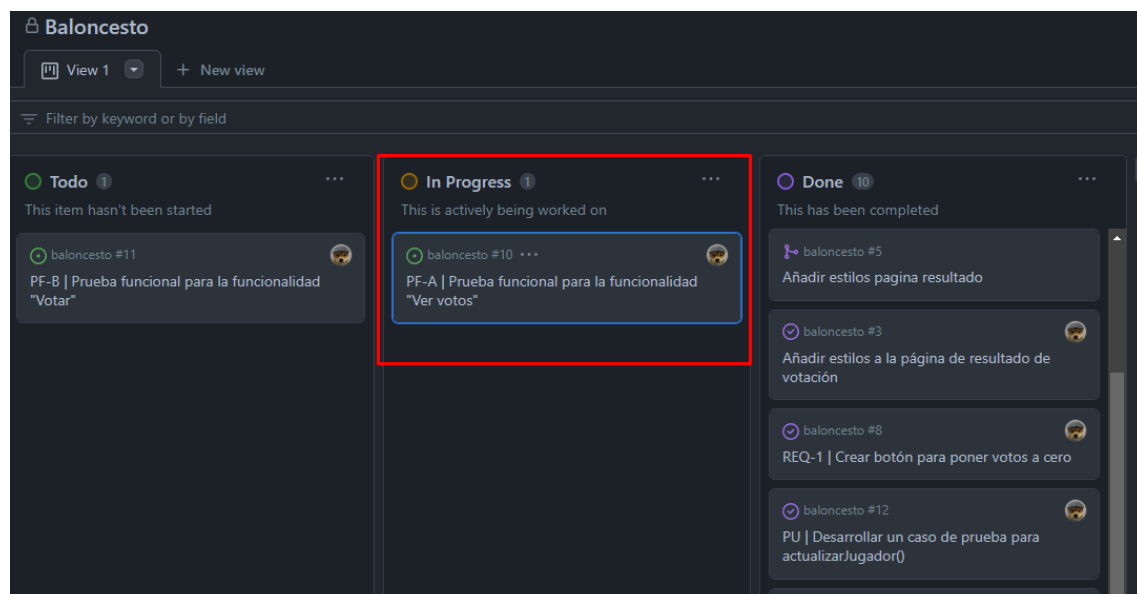
[Ir a la página principal](#)

Comprobado el funcionamiento de la tarea, cerramos el issue y actualizamos su estado en el tablero Kanban del proyecto.

Implementación de la tarea PF-A

Continuando en la rama borrar-datos, debemos desarrollar la tarea PF-A, en la que se nos pide 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.

Comenzaremos por actualizar el estado de la tarea en el tablero Kanban:



Para la desarrollar la tarea, crearemos un método llamado **comprobarResetVotos()** cuyo funcionamiento es el siguiente:

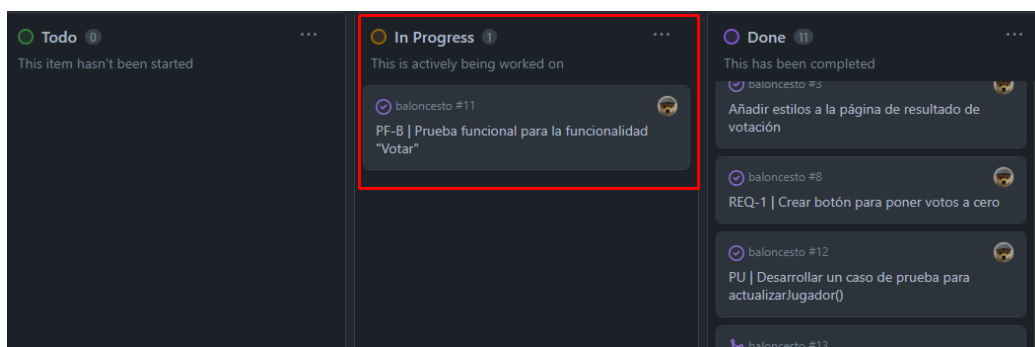
Navegamos a la página principal de nuestro sitio web, esperamos un máximo de 10 segundos a que cargue la página (en caso de que sea necesario), pulsamos sobre el botón “Poner votos a cero”, deshabilitamos la alerta que indica que los votos han sido borrados para que no interfiera con el comportamiento del test, pulsamos sobre el botón “Ver votos”, buscamos la tabla que se muestra, recorremos sus filas y comprobamos que todas las celdas de la columna que contiene el número de votos contengan el valor “0”- En caso de que todas las celdas con el número de votos estén a cero, el test se aprobará. En caso contrario, el test fallará.

Una vez desarrollado el test, subiremos el código a la rama ver-votos, cerraremos el issue y actualizaremos el tablero Kanban.

Implementación de la tarea PF-B

Continuando en la rama borrar-datos, debemos desarrollar la tarea PF-A, en la que se nos pide 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.

Comenzaremos actualizando el estado de Kanban y luego pasaremos al desarrollo:



Para la desarrollar la tarea, crearemos un método llamado **comprobarVotoNuevoJugador ()** cuyo funcionamiento es el siguiente:

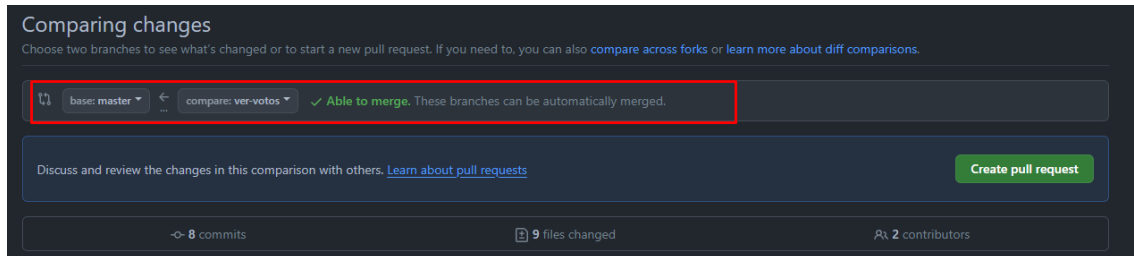
Navegamos a la página principal de nuestro sitio web, esperamos un máximo de 10 segundos a que cargue la página (en caso de que sea necesario), escribimos el valor “JugadorPrueba” en el campo de texto asociado al radiobutton “Otro”, pulsamos sobre el radiobutton “Otro”, pulsamos sobre el botón “Votar”, esperamos a que cargue la siguiente vista en la que se agradece el voto, pulsamos sobre el enlace “Ir al comienzo” para volver a la página principal, esperamos a que cargue la página principal, pulsamos el botón “Ver votos” y comprobamos que la tabla contenga un campo cuyo valor sea “JugadorPrueba”.

En caso de que exista un celda con el valor “JugadorPrueba”, el test se pasará con éxito. En caso contrario, fallará.

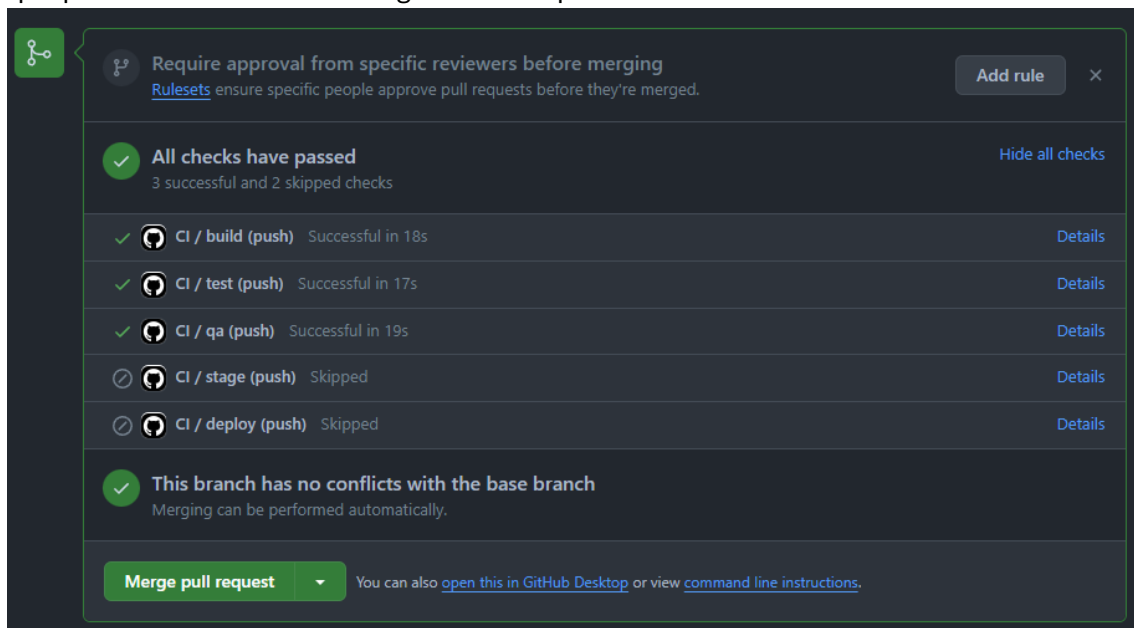
Una vez desarrollado el test, subiremos el código a la rama ver-votos, cerraremos el issue y actualizaremos el tablero Kanban.

Merge de la rama ver-votos a master

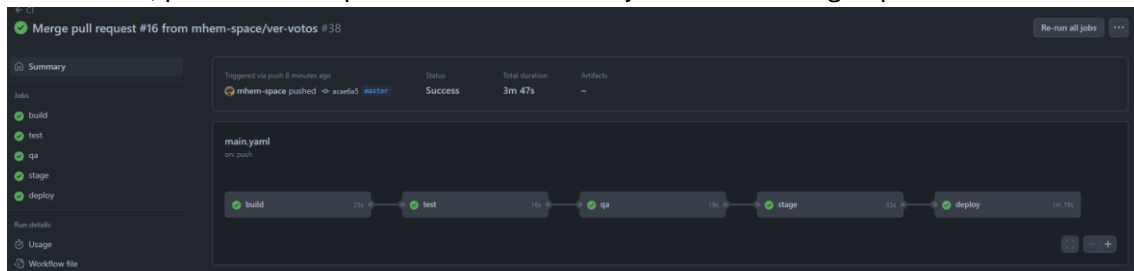
Una vez desarrolladas las últimas tareas en la rama **ver-votos** debemos incorporar el nuevo código a la rama master. Por lo que debemos solicitar el merge y a continuación aprobarlo:



Creada la petición de pull podemos observar que no se encontró ningún conflicto, por lo que podemos confirmar el merge sin tener que solucionar conflictos:



Finalmente, podemos ver que el workflow se ha ejecutado sin ningún problema:



Actualización de SonarQube

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 ello debemos actualizar la quality gate creada en la práctica del tema 5 y bajar el número de **major issues** permitidos de 25 a 20:



Ilustración 1 **Límite inicial**

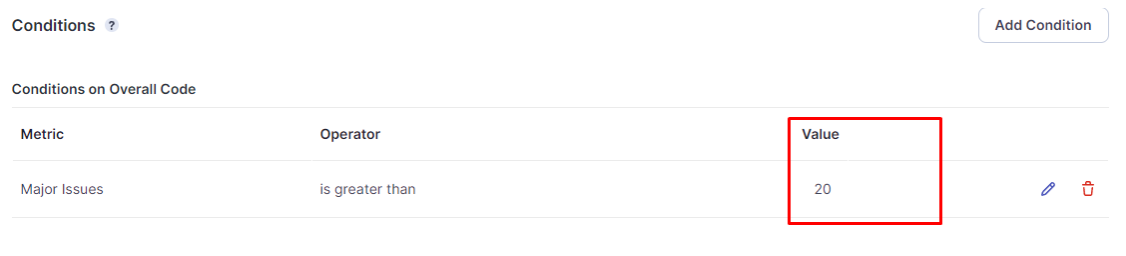
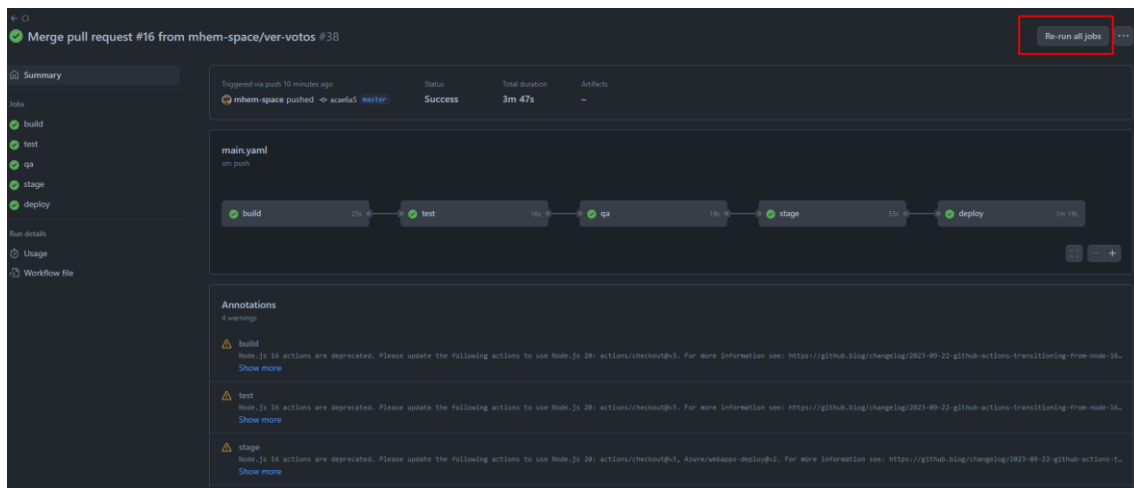


Ilustración 2 **Límite final**

Actualizado el límite de problemas permitidos, volvemos a ejecutar el último workflow:



Debido a que solamente tenemos 8 errores mayores, el workflow pasará sin problemas ya que tenemos menos de 20 problemas mayores.