

Trabajo Final

Omar Ait Lhouss

Asignatura: Integración Continua en el Desarrollo Ágil

Máster Universitario en Desarrollo Ágil de Software para la Web

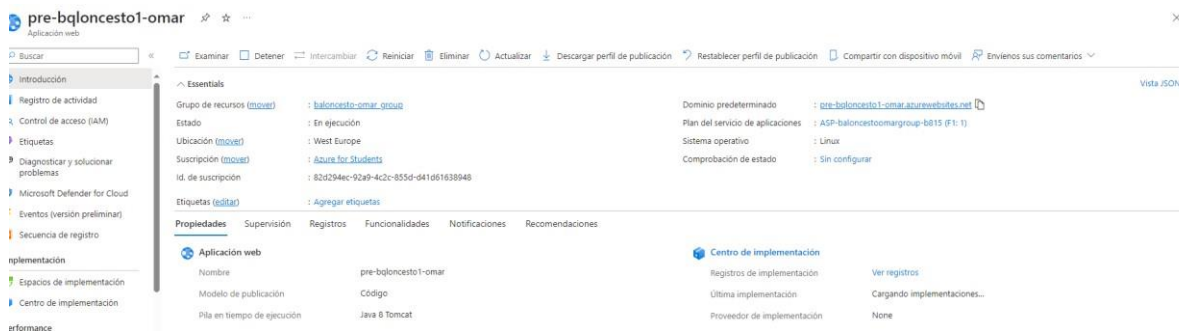
Contenido

Creación del entorno de pre-producción	3
Creación del nuevo sprint (milestone)	4
Creación de las user stories (issues)	5
Implementación de la tarea REQ-1	6
Implementación de la tarea PU	9
Implementación de la tarea REQ-2	10
Implementación de la tarea PF-A	14
Implementación de la tarea PF-B.....	15
Actualización de SonarQube	16

CREACIÓN DEL ENTORNO DE PRE-PRODUCCIÓN

Dado que se llevó a cabo una investigación relacionada con el tema 5 de la asignatura, debemos establecer una nueva aplicación servicio en la que se desplegará el trabajo que añadiremos al archivo main.yml del proceso de trabajo.

Para ello, utilizaremos los mismos datos que utilizamos para crear el entorno de producción y le asignaremos un nombre distinto.













Una vez que haya sido creado el nuevo servicio de aplicación, descargaremos su perfil de publicación para crear el token necesario para el archivo **Main.yml**.



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

Repository secrets

New repository secret

Name 	Last updated		
 AZURE_WEBAPP_PUBLISH_PROFILE	2 weeks ago		
 PRE_AZURE_WEBAPP_PUBLISH_PROFILE	2 weeks ago		
 TOKEN	2 weeks ago		

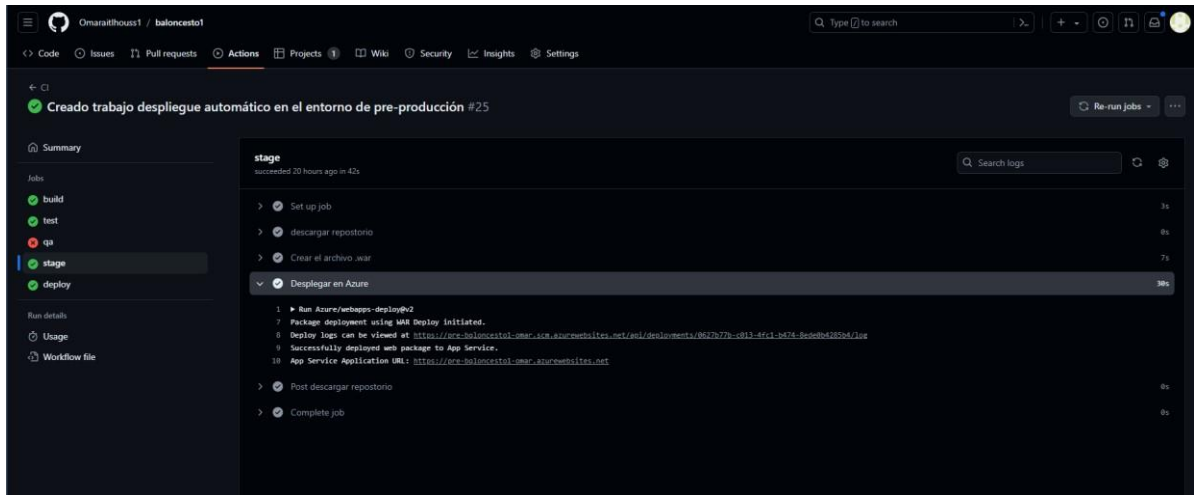
Con el nuevo perfil de publicación añadido al repositorio, ya podemos añadir el nuevo trabajo al workflow. Para ello, usaremos la misma estructura que el trabajo **deploy** pero eliminando una sección:

```
stage:
  needs: qa
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/master'
  steps:
    - name: descargar repositorio
      uses: actions/checkout@v3
    - name: Crear el archivo .war
      run: |
        mvn package -DskipTests=true
    - name: Desplegar en Azure
      uses: Azure/webapps-deploy@v2
      with:
        app-name: pre-bqloncesto1-omar
        publish-profile: ${ secrets.PRE_AZURE_WEBAPP_PUBLISH_PROFILE }
        package: target/*.war
```

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

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 practica final

Due date (optional)

dd/mm/aaaa

Description

Create milestone

© 2024 GitHub, Inc.

Terms

Privacy

Security

Status

Docs

Contact

Manage cookies

Do not share my personal information

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

REQ1 Crear boton de votos a cero

Add a description

Write

Preview

H B I

Un boton para Poner votos a cero y se cambiaran a 0 los de jugadores

Markdown is supported

Paste, drop, or click to add files

Submit new issue

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Assignees

Omaraitthouss1

Labels

None yet

Projects

baloncesto1

Milestone

Sprint practica 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

🔍

Labels9

Milestones2

New issue

4 Open

✓ 19 Closed

AuthorLabelProjectsMilestonesAssigneeSort

PF-B/ Prueba para la funcionalidad de Votar

#25 opened now by Omaraitthouss1

Sprint practica fi...

Prueba para la funcionalidad de Ver votos

#24 opened 1 minute ago by Omaraitthouss1

REQ 2 Crear vista de los votos

#23 opened 2 minutes ago by Omaraitthouss1

Sprint practica fi...

REQ1 Crear boton de votos a cero

#22 opened 4 minutes ago by Omaraitthouss1

Sprint practica fi...

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

🔍

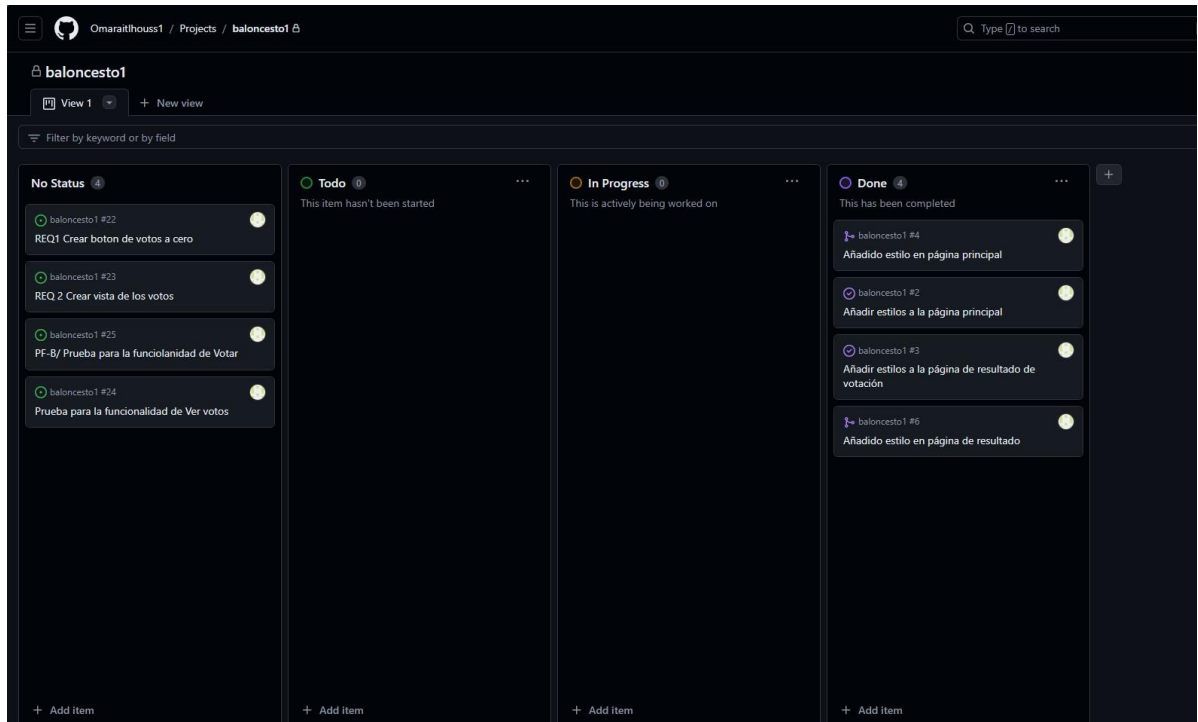
🔍

🔍

🔍

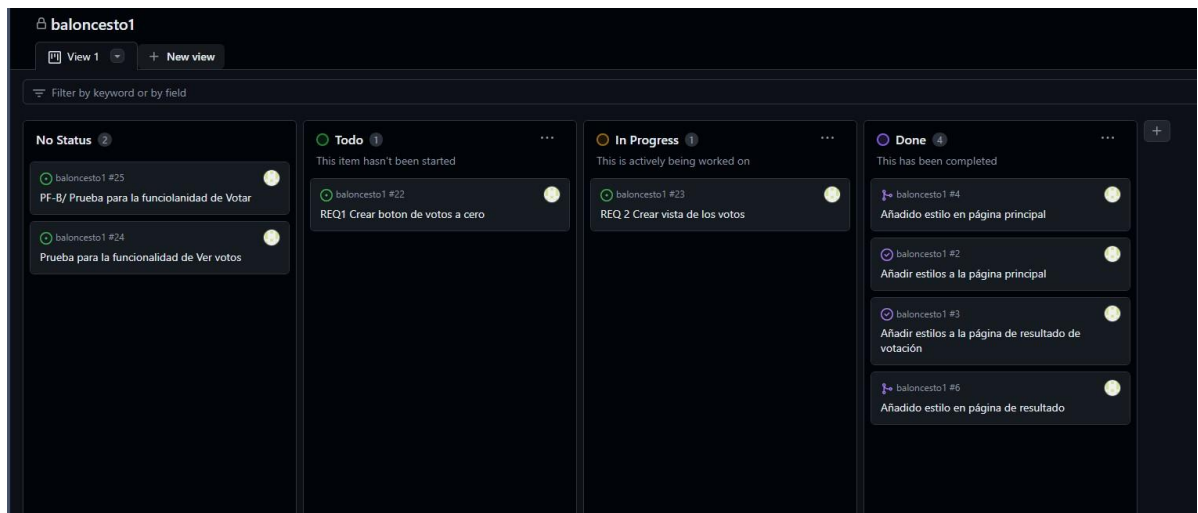
<

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



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:

```
<input id="borrarVotos" type="submit" name="resetVotes" value="Poner votos a cero" />
```

```

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

```

```

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

    if (req.getParameter("resetVotos") != null) {
        bd.resetearVotos();
        // Crear una cookie para indicar que se han eliminado los votos.
        Cookie mensajeCookie = new Cookie("mensajeVotos",
            URLEncoder.encode(s:"Todos los votos han sido eliminados.", enc:"UTF-8"));
        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.
    }

    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.
    }
}

```

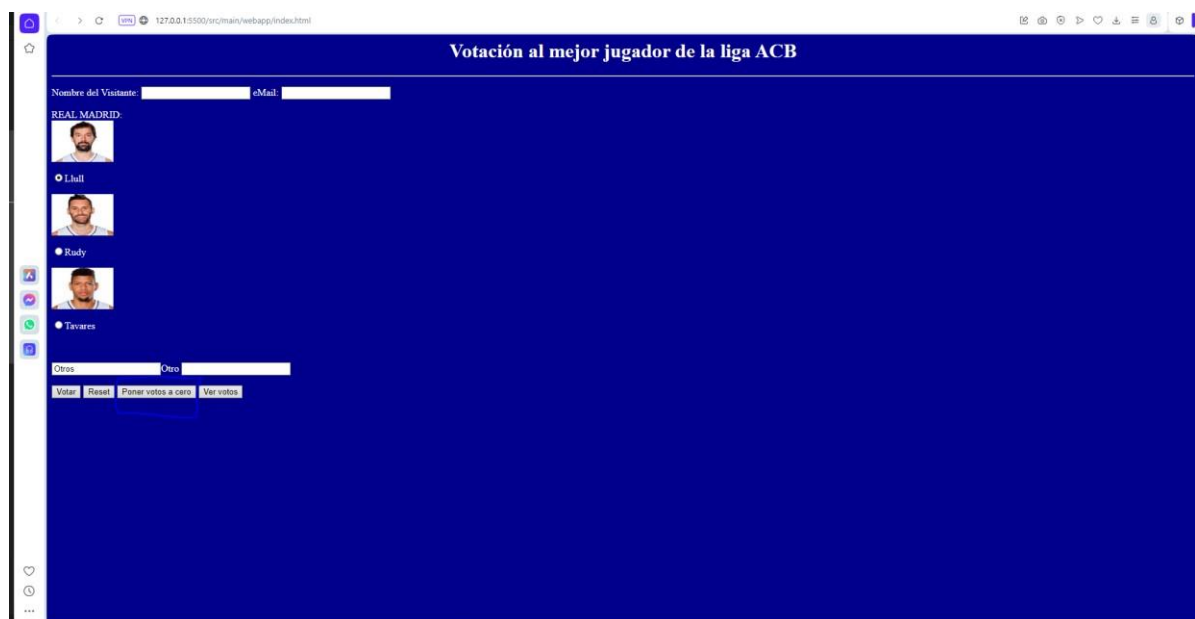
```

<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>
</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);
    }
  }
}

```

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

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.

Nos vamos a elegir el método mockito en esta tarea y agregamos 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>
```

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);

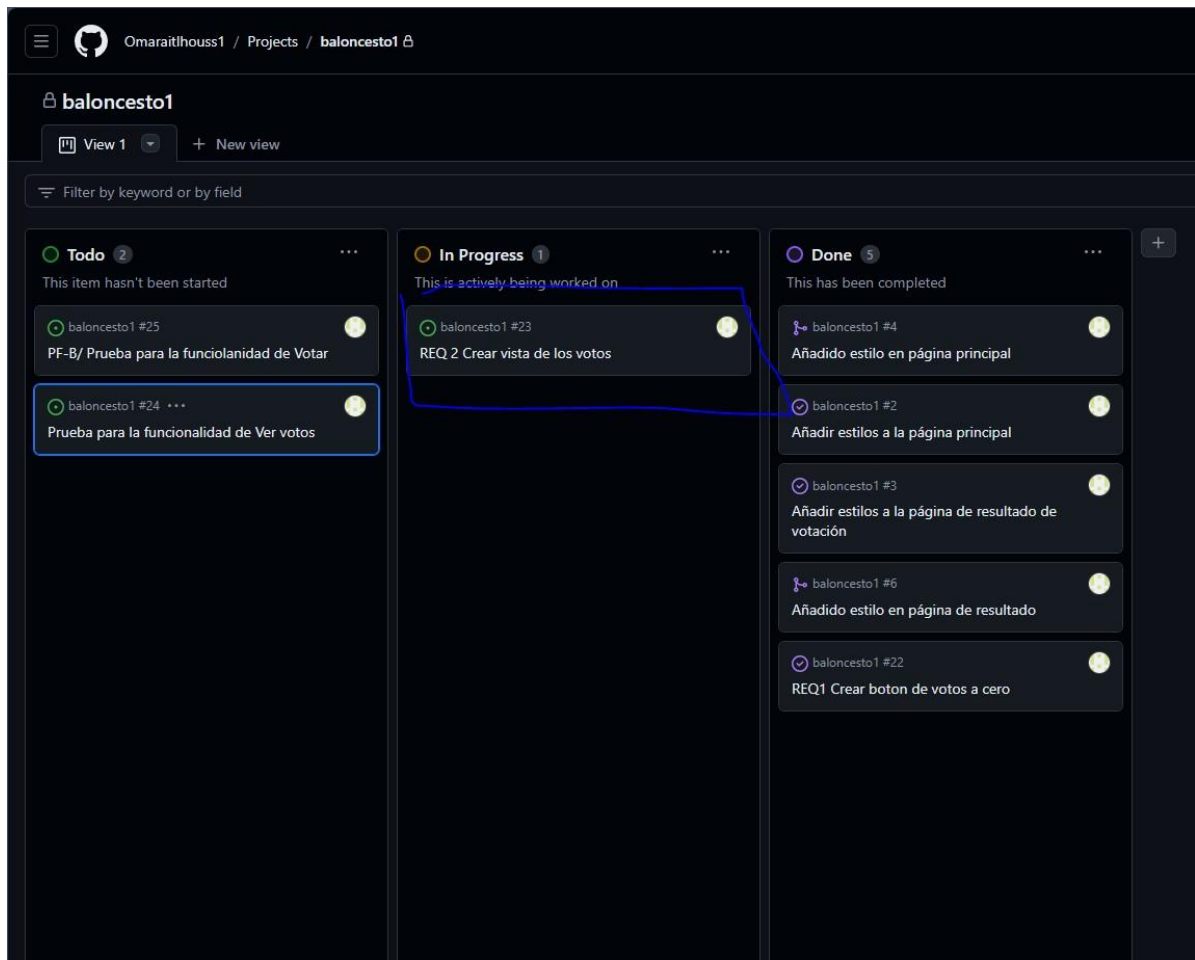
    // Verificar que el método executeUpdate fue llamado con la consulta SQL
    // esperada
    verify(mockStatement, times(1))
        .executeUpdate("UPDATE Jugadores SET votos=votos+1 WHERE nombre " + " LIKE '%" + nombreJugador + "%'");

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

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

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**:



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.

```
<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, la ejecución de la query **SELECT * FROM Jugadores**

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

- **Editar el controlador:** 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.

```
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.  
}
```

- **Creación de la clase Jugador:**

```
src > main > java > J Jugador.java > Jugador  
You, 21 hours ago | 1 author (You)  
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 }
```

You, 21 hours ago • Añadida vista para Ver Votos

- **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.

```
src > main > webapp > <?> VerVotos.jsp <?> html <?> body.verVotos <?> table#tablaDeVotos <?> tr
You, 20 hours ago | 1 author (You)
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>
```

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

Votación al mejor jugador de la liga ACB

Nombre del Visitante: eMail:

REAL MADRID:

☒ Lluís

☐ Rudy

☐ Tavares

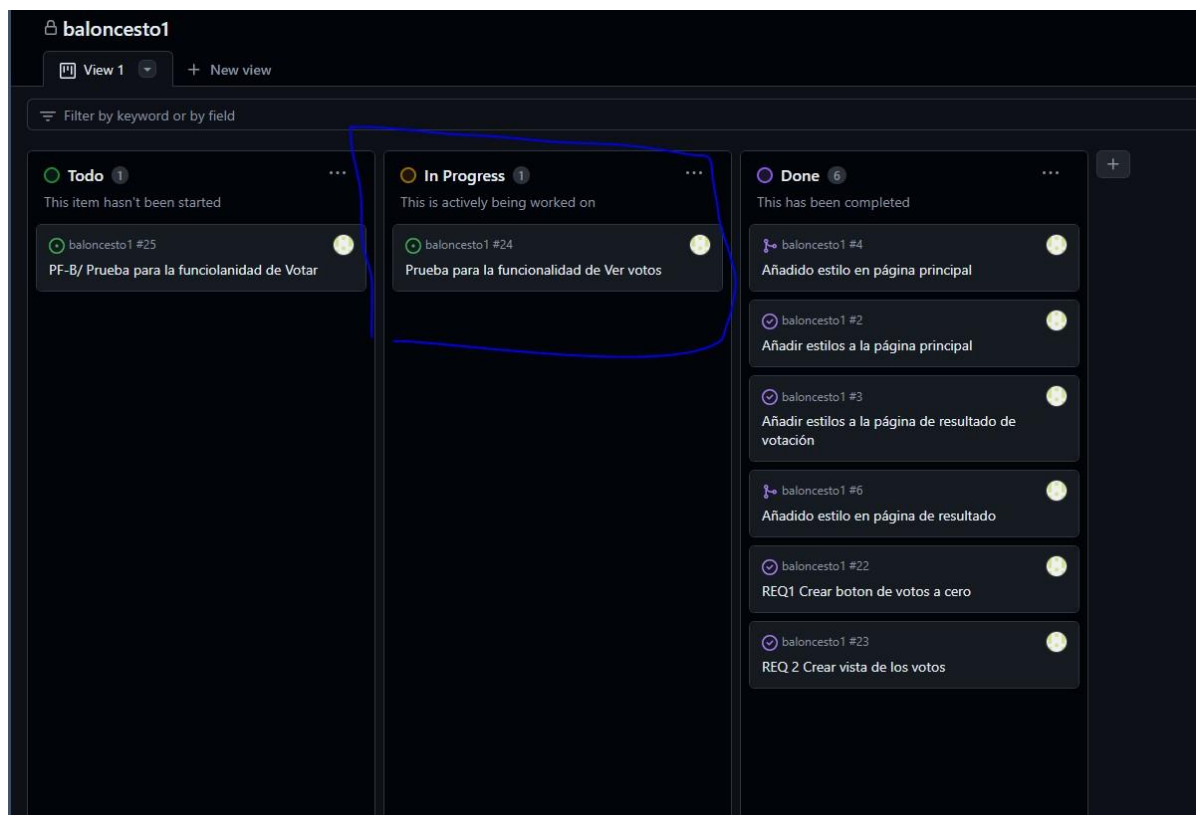
Otros Otro

Votos de los jugadores

Nombre del jugador	Número de votos
Llull	0
Rudy	1
Tavares	0
Ir a la página principal	

IMPLEMENTACIÓN DE LA TAREA PF -A

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

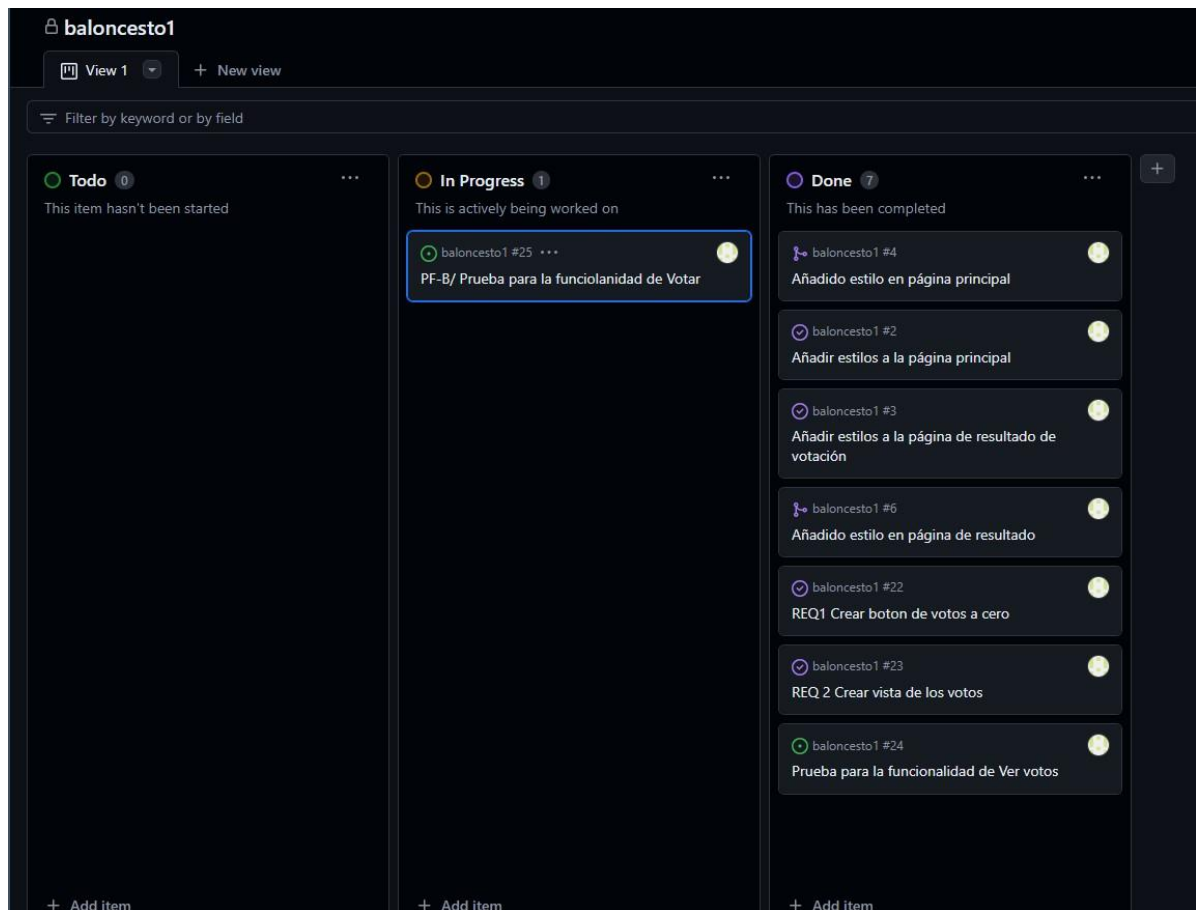


Para llevar a cabo la tarea, utilizaremos un método denominado `comprobarResetVotos()` con el funcionamiento siguiente.

Nos dirigimos a la página principal de nuestro sitio web, esperamos un máximo de 10 segundos para que la página se libere, seleccionamos la tabla que se muestra, recorremos sus filas y verificamos que todas las celdas están disponibles. En el caso contrario, el test no se realizará. 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

Al proseguir con la rama de borrar-datos, debemos iniciar la tarea PF-A, en la que se nos requiere 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", vuelva a la página principal, simule la pulsación del botón "Ver votos", y verifique que en la página de VerVotos.jsp ese nuevo jugador dispone de 1. Comencemos con la actualización del estado de Kanban y posteriormente nos dirigiremos al desarrollo.



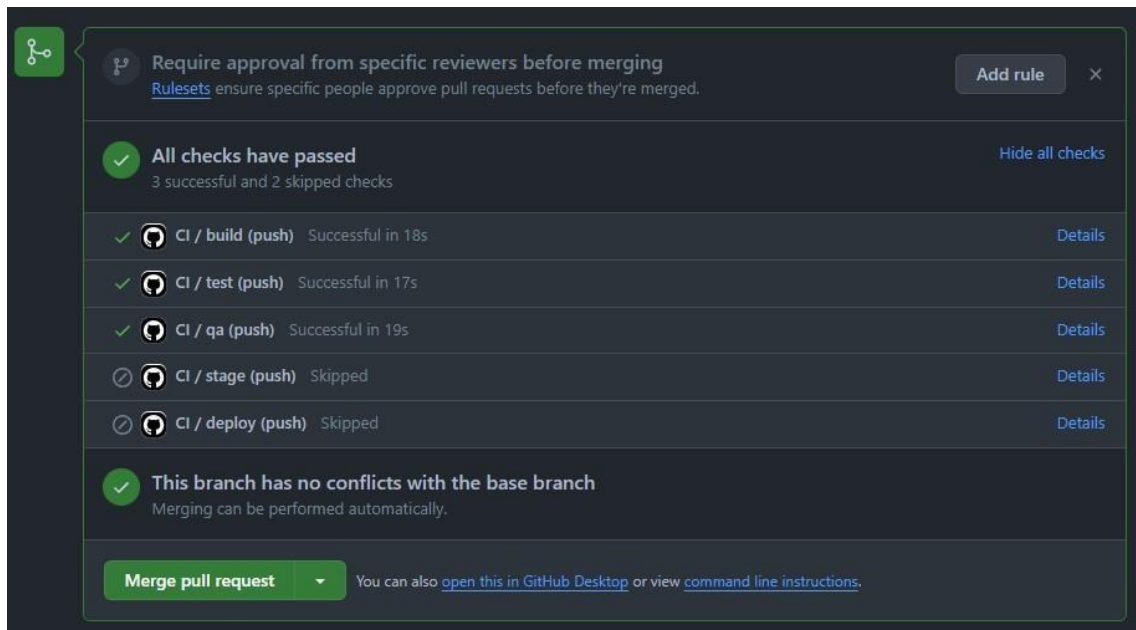
Para llevar a cabo la tarea, utilizaremos un procedimiento denominado comprobarVotoNuevoJugador () cuyo funcionamiento es el siguiente:

Nos dirigimos a la página principal de nuestro sitio web, esperamos un máximo de 10 segundos para que pueda acceder a la página web (en el caso de que sea necesario), escribimos el valor "JugadorPrueba" en el campo de texto asociado al radiobutton "Otro", pulsamos sobre el botón "Votar", esperamos que pueda acceder a la siguiente vista en la que se agradece el voto, pulsamos sobre el enlace "Ir al comienzo".

Si existe una celda con el valor "JugadorPrueba", el ensayo se llevará a cabo con éxito. En el supuesto contrario, sería un error.

Una vez que se haya elaborado el test, adjuntaremos el código en la rama de votos, cerraremos el problema y actualizaremos el tablero Kanban.

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:



ACTUALIZACIÓN DE SONARQUBE

Como garantía de calidad, el código fuente completo del proyecto debe cumplir con un límite de problemas relevantes (major problemas) con SonarQube, en el caso contrario no se podrá extender la aplicación a la producción.

