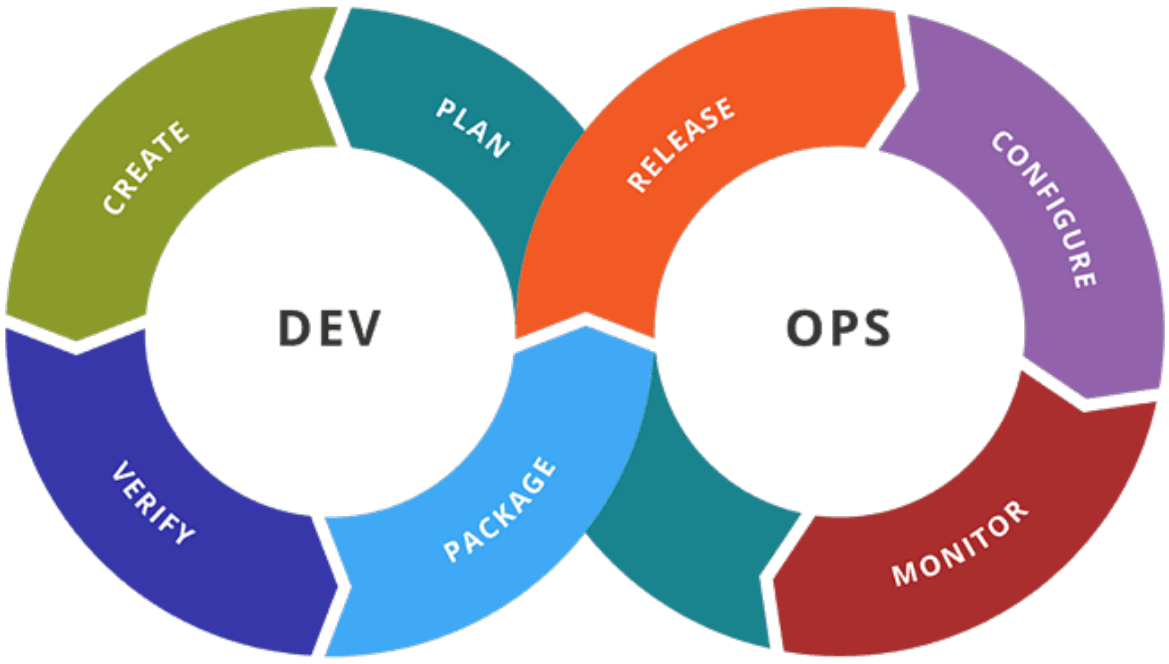
Integración Continua

Trabajo final

Patricia Coromoto Matos Meza



Universidad de Alcalá
22 de febrero de 2024

Índice

1. Crear entorno pre-productivo

Hemos creado un job entre qa y deploy, que despliega la aplicación en Azure en un entorno de staging o pre producción. Este job es dependiente de qa, a su vez, deploy es dependiente del mismo.

Para hacer este despliegue hemos creado una aplicación web en Azure llamada baloncesto-matos-stage y hemos añadido el secreto al repositorio llamado **AZURE_WEBAPP_PUBLISH_PROFILE_PRE**.

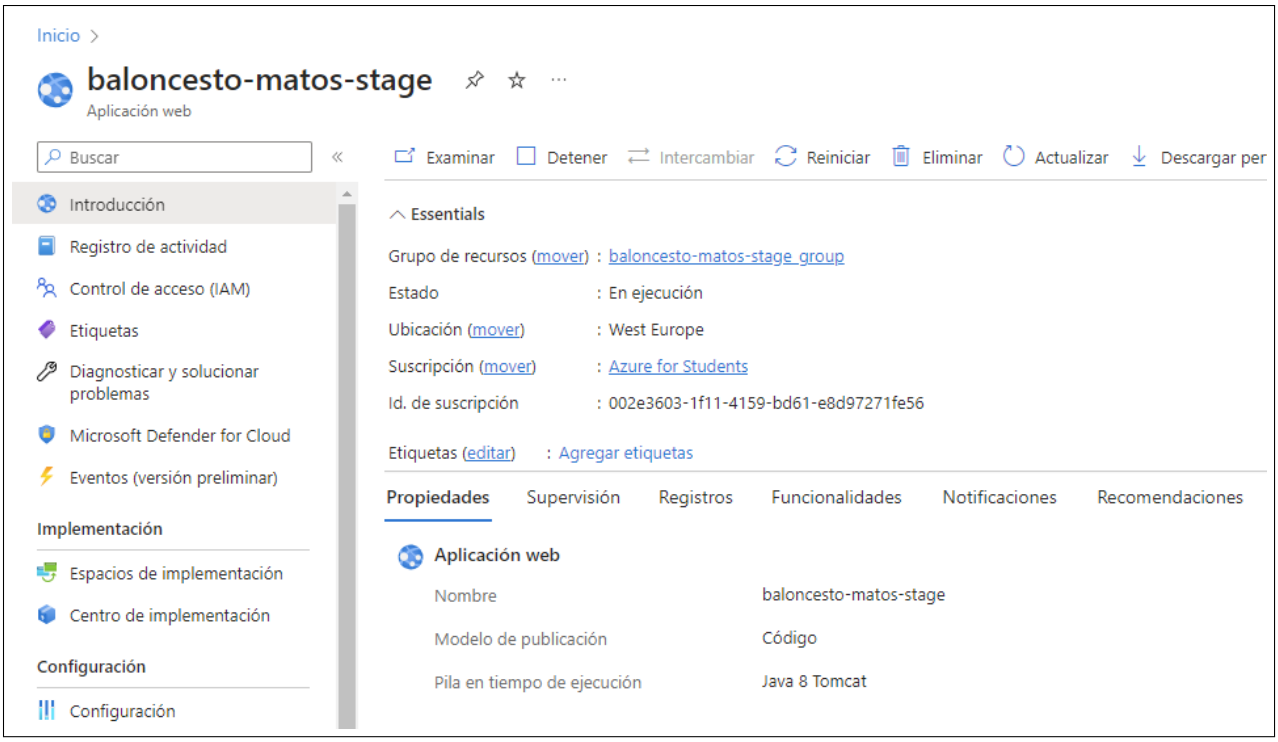


Figura 1: Pre-producción o STG

URL del sitio STG: https://baloncesto-matos-stage.azurewebsites.net/Baloncesto/.

URL del sitio PRO: https://baloncesto-matos.azurewebsites.net/Baloncesto/index.html.

Github: https://github.com/patriciamatosm/baloncesto

2. Gestión ágil

Hemos creado un milestone llamado Modificación de votos, que contiene los requisitos que nos proponen en el enunciado, como se ve en la figura 2.

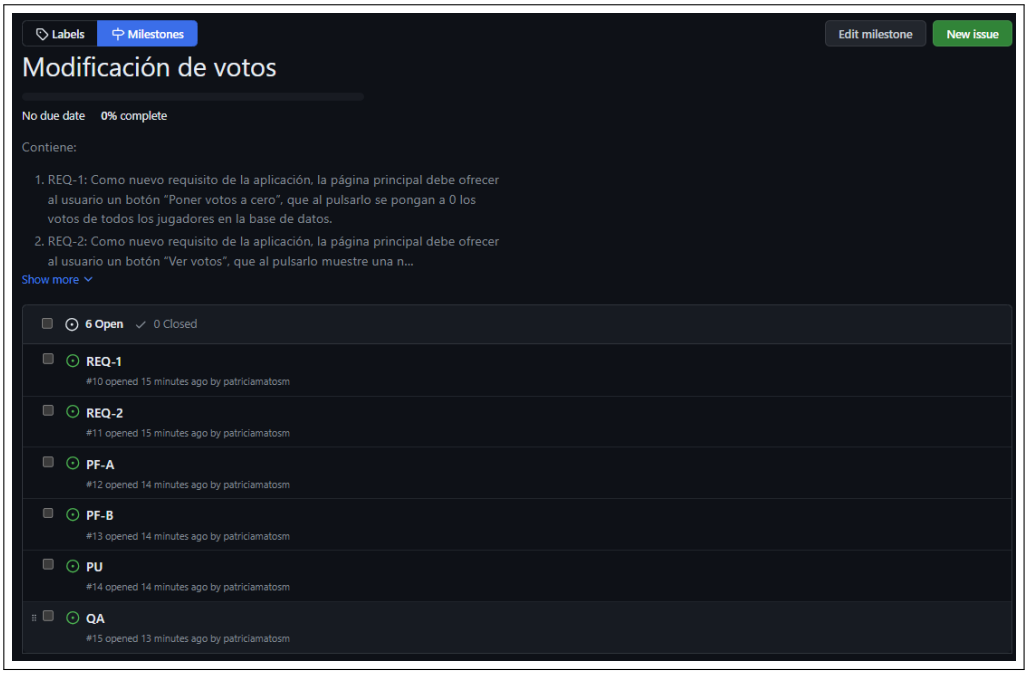


Figura 2: Milestone

Además, hemos arrancado el sprint y hemos puesto las primeras issues en in progress y el resto en To Do.

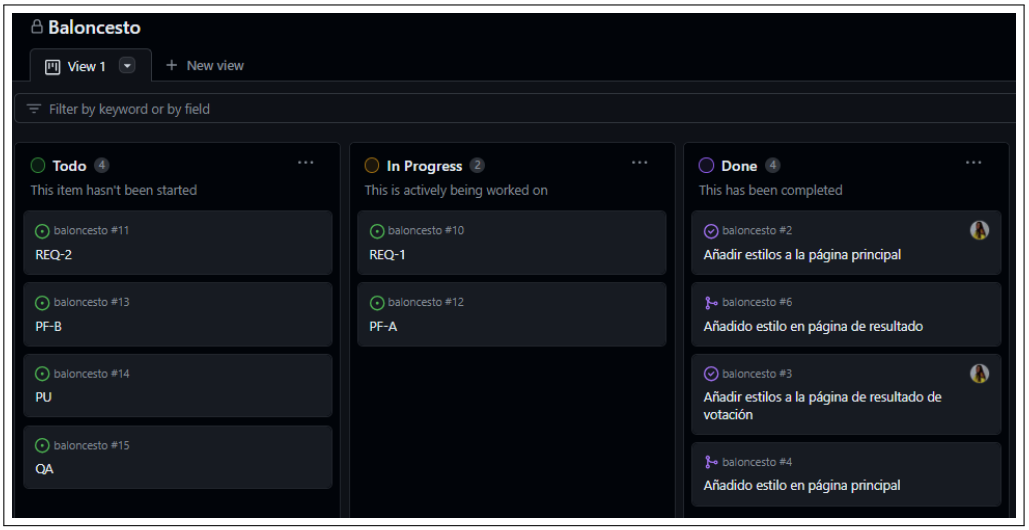


Figura 3: Tablero Kanban

3. Requisitos

A continuación, los requisitos que se han requerido:

3.1. REQ 1: Rama REQ 1

El requisito 1 nos pedía:

```
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 esto hemos editado los ficheros:

- **index.html**: Para añadir el botón nuevo con unos ids y names destacables para poder acceder a ellos desde fuera.
- **Acb.java**: Para añadir el código fuente necesario para llamar al método de ModeloDatos para poner votos a cero.
- **ModeloDatos.java** Para añadir el método votesTo0, que hace un update a la tabla Jugadores y coloca todo a 0.

Figura 4: Botón de Poner votos a 0

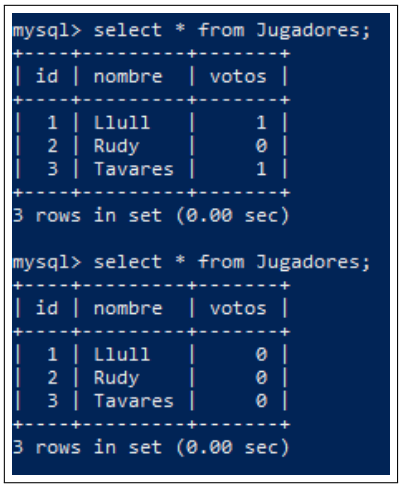


Figura 5: Antes y después de pulsar el botón

3.2. REQ 2: Rama REQ 2

El requisito 2 nos pedía:

> 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.

Para esto hemos editado los ficheros:

- **index.html**: Para añadir el botón nuevo con unos ids y names destacables para poder acceder a los votos.
- **Acb.java**: Para añadir el código fuente necesario para llamar al método de ModeloDatos para mostrar los votos por pantalla
- **ModeloDatos.java** Para añadir el método getData para obtener la info de base de datos, concatenarla en una string y enviarla al front.
- **VerVotos.jsp**: Creación de este fichero para mostrar por pantalla el resultado.

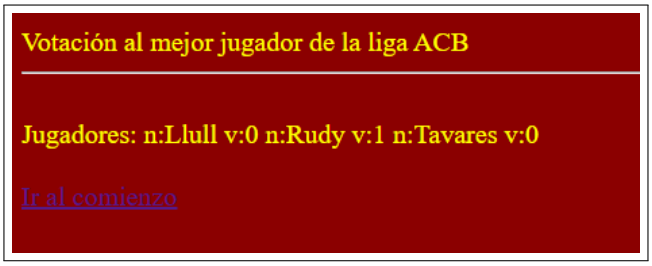


Figura 6: Botón de Poner votos a 0

3.3. PF-A, PF-B y PU: Rama Pruebas

Los requisitos de pruebas nos pedían un par de pruebas funcionales (PF-A y PF-B) y una Prueba Unitaria (PU).

Desafortunadamente no se ha podido realizar el requisito PU. Sin embargo, nuestra intención para programarlo era la siguiente:

```
    private void insertarDatosDePrueba() {
        // Inserta datos de prueba en la base de datos (simulación)
        try (Connection connection = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/baloncesto", "usuario",
                "clave")) {
            String insertQuery = "INSERT INTO Jugadores (nombre, votos) VALUES (?, ?)";
            try (PreparedStatement statement = connection.prepareStatement(insertQuery)) {
                statement.setString(1, "Llull");
                statement.setInt(2, 0);
                statement.executeUpdate();
            }
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException("Error
            al insertar datos de prueba en la base de datos");
        }
    }
    @BeforeAll
    public void setUp() throws SQLException {
        // Crea una instancia de ModeloDatos antes de cada prueba
        modeloDatos = new ModeloDatos();

        // Abre la conexión a la base de datos de prueba
        modeloDatos.abrirConexion();

        // Inserta datos de prueba en la base de datos
        insertarDatosDePrueba();
    }

    @AfterAll
    public void tearDown() throws SQLException {
        modeloDatos.cerrarConexion();
    }

    @Test
    public void testVote() throws SQLException {
        int votosAntes = modeloDatos.getVotes("Llull");

        modeloDatos.actualizarJugador("Llull");

        int votosDespues = modeloDatos.getVotes("Llull");

        assertEquals(votosAntes + 1, votosDespues, "Los votos del
        jugador no se han incrementado en 1");
    }

```

Siguiendo con los requisitos PF-A y PF-B, se ha implementeado lo siguiente:

PF-A:

```
    @Test
    public void pfATest() {
        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/");

        // Pulsar el botón "Poner votos a cero" en la página principal
        WebElement botonPonerVotosACero = driver.findElement(By.id("votos_a_cero"));
        botonPonerVotosACero.click();

        // Esperar un momento para que la página se actualice
        driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);


        // Volver a la página principal
        driver.navigate().back();

        // Esperar un momento para que la página se actualice
        driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

        // Pulsar el botón "Ver votos" en la página principal
        WebElement botonVerVotos = driver.findElement(By.id("ver_votos"));
        botonVerVotos.click();

        // Esperar un momento para que la página se actualice
        driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

        WebElement paragraphElement = driver.findElement(By.id("stringName"));
        String actualText = paragraphElement.getText();

        String expectedText = "Jugadores: n:Llull v:0 n:Rudy v:0 n:Tavares v:0";
        assertEquals(expectedText, actualText, "Text from the paragraph does
        not match the expected value");

        System.out.println(driver.getTitle());
        driver.close();
        driver.quit();
    }

```

PF-B:

```
    @Test
    public void pfBTest() {
        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/");

        WebElement nombreJugadorInput = driver.findElement(By.id("txtOtros"));
        nombreJugadorInput.sendKeys("Javier");

        // Seleccionar la opción "Otro"
        WebElement opcionOtro = driver.findElement(By.id("otrosRadio"));
        opcionOtro.click();

        // Pulsar el botón "Votar"
        WebElement botonVotar = driver.findElement(By.id("votar"));
        botonVotar.click();


        // Volver a la página principal
        driver.navigate().back();

        // Esperar un momento para que la página se actualice
        driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

        // Pulsar el botón "Ver votos" en la página principal
        WebElement botonVerVotos = driver.findElement(By.id("ver_votos"));
        botonVerVotos.click();

        // Esperar un momento para que la página se actualice
        driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

        WebElement paragraphElement = driver.findElement(By.id("stringName"));
        String actualText = paragraphElement.getText();

        String expectedText = "Jugadores: n:Llull v:0 n:Rudy v:0 n:Tavares v:0 n:Javier v:1";
        assertEquals(expectedText, actualText, "Text from the paragraph
        does not match the expected value");

        System.out.println(driver.getTitle());
        driver.close();
        driver.quit();
    }

```

3.4. QA

Para el requisito de QA, hemos cambiado la regla en Sonarqube a 20 errores:

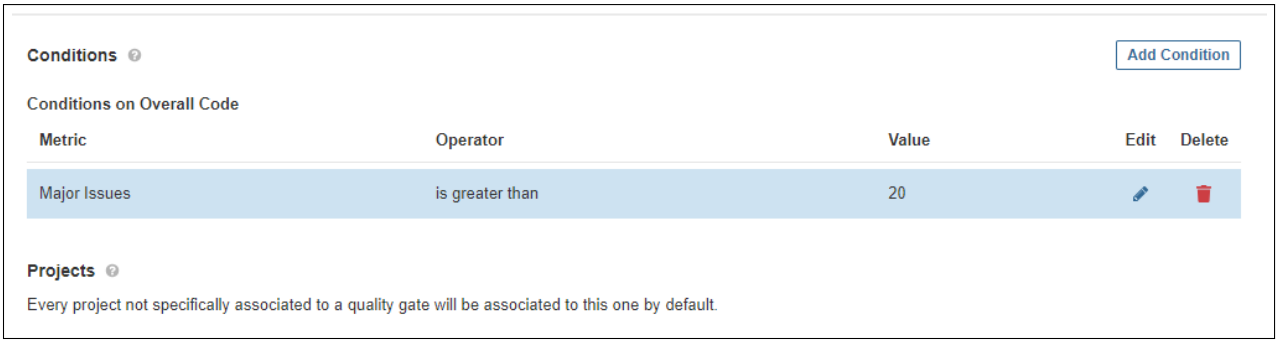


Figura 7: Rule Sonarqube

Y los resultados fueron los siguientes:

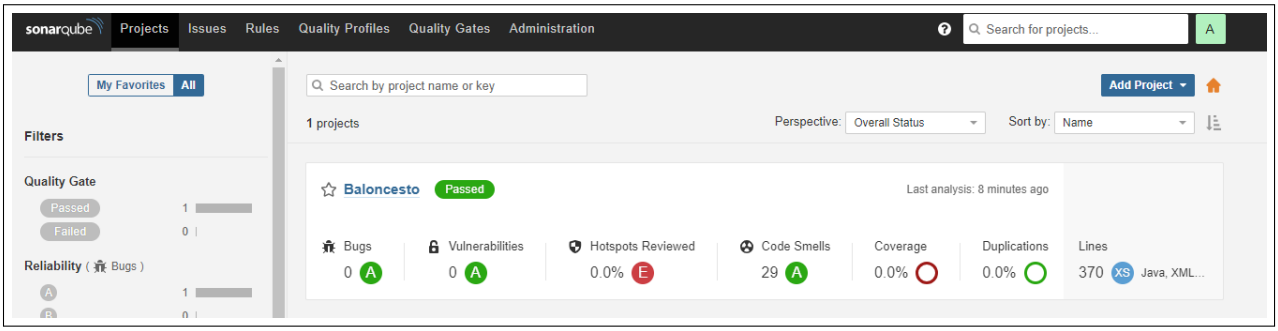


Figura 8: Resultados Sonarqube