

# Trabajo Final ICDA

Primero creamos un nuevo trabajo en el workflow del proyecto para subir primeramente la versión del proyecto en preproducción, este trabajo seria automático como otros, es decir no necesita autorización manual.

## 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 \* ⓘ Azure para estudiantes

Grupo de recursos \* ⓘ baloncesto\_group

[Crear nuevo](#)

### Detalles de instancia

Nombre \* baloncesto-pre-stanus .azurewebsites.net

Publicar \* ☒ Código ☐ Contenedor Docker ☐ Aplicación web estática

Pila del entorno en tiempo de ejecución \* Java 8

Pila de servidor web Java \* Apache Tomcat 9.0

Sistema operativo \* ☒ Linux ☐ Windows

Región \* West Europe

**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) \* ⓘ ASP-baloncestogroup-96a7 (B1)

[Crear nuevo](#)

Plan de precios **Básico B1** (Total de ACU: 100, 1.75 GB de memoria, 1 vCPU)

### Redundancia de zona

Un plan de App Service se puede implementar como un servicio con redundancia de zona en las regiones que lo admiten. Esta es una decisión que solo se toma en el momento de la implementación. Después de la implementación, no se podrá hacer que un plan de App Service tenga redundancia de zona. [Más información](#)

Redundancia de zona

☐ **Habilitada:** Su plan de App Service y las aplicaciones que contiene tendrán redundancia de zona. El número mínimo de instancias del plan de App Service será tres.

☒ **Deshabilitada:** El plan de App Service y las aplicaciones que contiene no tendrán redundancia de zona. El número mínimo de instancias del plan de App Service será uno.

Revisar y crear




< Anterior

Siguiente: Base de datos >

Una vez creada la aplicación web en Azure creamos la variable secreta en GitHub, así como lo hemos hecho en producción.

## Repository secrets

New repository secret

Name ↕	Last updated
 AZURE_WEBAPP_PRE_PUBLISH_PROFILE	1 minute ago  

Una vez hecho esto editamos el fichero “main.yaml” del proyecto y añadimos el nuevo trabajo llamado “stage”, así como vemos en la imagen de abajo, con la variable secreta creada anteriormente:

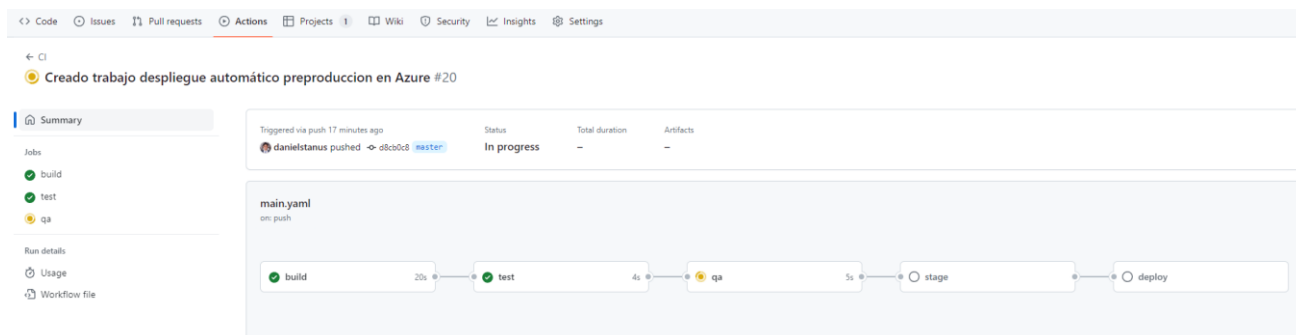
```
stage:
  needs: qa
  runs-on: ubuntu-latest
  continue-on-error: true
  steps:
    - name: Descargar repositorio
      uses: actions/checkout@v3
    - name: Crear el archivo .war
      run: |
        mvn package -DskipTests=true
    - name: Desplegar en Azure PreProducción
      uses: Azure/webapps-deploy@v2
      with:
        app-name: baloncesto-pre-stanús
        publish-profile: ${ secrets.AZURE_WEBAPP_PRE_PUBLISH_PROFILE }
        package: target/*.war
```

Después de esto hacemos un commit y push del trabajo creado con los siguientes comandos:

```
git commit -m "Creado trabajo despliegue automático preproduccion en Azure"
```

```
git push
```

Si miramos en actions de GitHub podemos ver que se están ejecutando los trabajos, en vez de tener 4 trabajos ahora tenemos uno más llamado stage:



The screenshot shows the GitHub Actions interface for a workflow named "Creado trabajo despliegue automático preproduccion en Azure #20". The workflow is triggered by a push to the master branch. The status is "In progress". The workflow consists of several jobs: build, test, qa, stage, and deploy. The "stage" job is currently running, and the "deploy" job is pending. The workflow is defined in the file "main.yaml" on the "push" event.

Con esto terminamos de crear el nuevo trabajo para la subida en preproducción de nuestro código.

Creamos un nuevo sprint con las siguientes issues:

Labels

Milestones

Edit milestone

New issue

## Nuevas funcionalidades

Due by January 23, 2024

0% complete

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 pá...

[Show more](#)

<input type="checkbox"/>	6 Open	0 Closed
<input type="checkbox"/>	REQ1 - Poner votos a cero <span>enhancement</span>	
	#9 opened 16 minutes ago by danielstanus	
<input type="checkbox"/>	REQ2 - Ver votos <span>enhancement</span>	
	#10 opened 10 minutes ago by danielstanus	
<input checked="" type="checkbox"/>	PFA - Prueba funcional de REQ1 y REQ2 <span>enhancement</span>	
	#11 opened 7 minutes ago by danielstanus	
<input type="checkbox"/>	PFB - Prueba funcional de REQ2 con un voto <span>enhancement</span>	
	#12 opened 6 minutes ago by danielstanus	
<input type="checkbox"/>	PU - Comprobar método "actualizarJugador()" si funciona correctamente <span>enhancement</span>	
	#13 opened 4 minutes ago by danielstanus	
<input type="checkbox"/>	QA - Comprobar código tenga menos de 20 problemas importantes <span>enhancement</span>	
	#14 opened 3 minutes ago by danielstanus	

Tip! You can use `shift + j` or `shift + k` to move items with your keyboard.

© 2024 GitHub, Inc. [Terms](#) [Privacy](#) [Security](#) [Status](#) [Docs](#) [Contact](#) [Manage cookies](#) [Do not share my personal information](#)

Para los REQ1 y REQ2 creamos 2 ramas nuevas:

- git branch poner-votos-a-cero
- git branch ver-votos

Para desarrollar la REQ1 en el index.html se ha creado un nuevo formulario con un botón:

```
<!-- Formulario para resetear votos -->
<form action="Acb" method="POST">
  <input type="submit" name="btnResetVotos" id="btnResetVotos" value="Poner votos a cero">
</form>
```

En el fichero Acb.java se ha modificado la función "service" para que ejecute una función dependiendo del botón que se pulse, en este caso cuando se pulsa el botón "btnResetVotos" se ejecutara la función "handleResetVotos", que es la encargada de llamar al "modeloDatos" para que reinicie los votos de los jugadores:

```
if (req.getParameter("btnVotar") != null) {
    handleVotar(req, session, res);
} else if (req.getParameter("btnResetVotos") != null) {
    handleResetVotos(res);
} else if (req.getParameter("btnVerVotos") != null) {
    handleVerVotos(req, session, res);
}

private void handleResetVotos(HttpServletResponse res) throws IOException {
    bd.resetVotosJugadores();
    res.sendRedirect(res.encodeRedirectURL("index.html"));
}
```

```
}
```

En la clase ModeloDatos.java he creado una función que actualiza los votos de los jugadores a 0 en la base de datos, la función se llama resetVotosJugadores():

```
public void resetVotosJugadores() {
    try {
        set = con.prepareStatement("UPDATE Jugadores SET votos = 0");
        set.executeUpdate();
    } catch (Exception e) {
        LOGGER.severe("No resetea la tabla de Jugadores");
        LOGGER.severe(MensajeError + e.getMessage());
    } finally {
        cerrarRecursos();
    }
}
```

La prueba unitaria se ha hecho de la siguiente manera:

1. Se obtiene los votos del jugador Rudy en una variable Integer
2. Se incrementa en 1 el voto del jugador Rudy en la base de datos
3. Se obtiene los votos del jugador Rudy en otra variable Integer
4. Se verifica con un assertEquals los votos después de incrementar en 1 con los votos anteriores más 1.
5. Se actualiza los votos del jugador Rudy, así como estaba antes de iniciar la prueba.

Código Java de la prueba ActualizarJugador():

```
@Test
public void testActualizarJugador() {
    System.out.println("Iniciando prueba de actualizarJugador");

    ModeloDatos modeloDatos = new ModeloDatos();
    modeloDatos.abrirConexion();

    String nombreJugador = "Rudy";
    //Obtenemos los votos del jugador antes de actualizar
    Integer votosJugadorAntes = modeloDatos.obtenerVotosJugador(nombreJugador);

    // Actualizamos los votos de jugador Rudy en 1
    modeloDatos.actualizarJugador(nombreJugador);

    //Obtenemos los votos del jugador despues de actualizar
    Integer votosJugadorDespues= modeloDatos.obtenerVotosJugador(nombreJugador);

    // Verificar que los votos del jugador Rudy han incrementado en 1
    assertEquals(votosJugadorDespues.intValue(), votosJugadorAntes+1,"Los votos del
jugador no se incrementaron correctamente");

    // Ponemos los votos como lo tenia antes el jugador
    modeloDatos.actualizarVotoJugador(nombreJugador,votosJugadorAntes);
    System.out.println("Prueba de actualizarJugador completada con éxito");
}
```

```
}
```

Para la REQ2 he creado una nueva página llamada "VerVotos.jsp" que muestra una tabla con los votos de todos los jugadores.

Para que muestre los datos se ha creado una función en "ModeloDatos.java", llamada "obtenerTodosLosVotos()", que devuelve una lista con todos los datos de la tabla "Jugadores".

Cuando un usuario pulsa el botón "Ver votos" desde la página inicial este botón llamara al servicio "Acb.java":

```
<!-- Formulario para Ver votos -->
<form action="Acb" method="POST">
  <input type="submit" name="btnVerVotos" id="btnVerVotos" value="Ver votos">
</form>
```

Desde "Acb.Java" llamamos a la función "obtenerTodosLosVotos()" para que nos de la lista de los datos y se lo devolvemos a la página "VerVotos.jsp" en un atributo de sesión.

```
private void handleVerVotos(HttpServletRequest req, HttpSession session,
HttpServletRequest res) throws IOException, ServletException {
    List<Map<String, Object>> votos = bd.obtenerTodosLosVotos();

    session.setAttribute("listaVotos", votos);
    res.sendRedirect(res.encodeRedirectURL("VerVotos.jsp"));
}
```

Finalmente, para ver todos los votos en la página "VerVotos.jsp", iteramos sobre la lista de votos (listaVotos) utilizando un bucle for:

```
<%
    List<Map<String, Object>> votos = (List<Map<String, Object>>)
session.getAttribute("listaVotos");
    if (votos != null && !votos.isEmpty()) {
%>
<table border="1">
  <thead>
    <tr><th>Nombre</th><th>Votos</th>
    </tr>
  </thead>
  <tbody>
    <% for (Map<String, Object> voto : votos) { %>
    <tr>
      <td><%= voto.get("nombre") %></td>
      <td><%= voto.get("votos") %></td>
    </tr>
    <% } %>
  </tbody>
</table>
<% } else { %>
<p>No hay votos disponibles.</p>
<% } %>
```

La primera prueba funcional(PF-A) se ha hecho de esta forma:

1. Primero configuramos el PhantomJS
2. Después Iniciamos el navegador y cargamos "index.html"
3. Buscamos el botón "Poner votos a cero" por id y lo hacemos click sobre el.
4. Buscamos el botón " Ver votos" por id y lo hacemos click sobre el
5. En la página de "Ver votos buscamos una tabla, después el "tbody" de la tabla, una vez encontrado esto iteramos sobre los elementos "tr" del "tbody", después buscamos todos los elementos "td", una vez encontrados comprobamos que en la columna 2 del "td" sea igual a 0 con un assertEquals.

```
// Encuentra y pulsa el botón "Poner votos a cero"
WebElement botonPonerVotosACero = driver.findElement(By.id("btnResetVotos"));
botonPonerVotosACero.click();

// Encuentra y pulsa el botón "Ver votos"
WebElement botonVerVotos = driver.findElement(By.id("btnVerVotos"));
botonVerVotos.click();

// Verifica que todos los votos en la página "VerVotos.jsp" son cero
WebElement tablaVotos = driver.findElement(By.tagName("table"));
WebElement cuerpoTabla = tablaVotos.findElement(By.tagName("tbody"));
for (WebElement fila : cuerpoTabla.findElements(By.tagName("tr"))) {
    WebElement celdaVotos = fila.findElements(By.tagName("td")).get(1);
    assertEquals("0", celdaVotos.getText());
}
```

La segunda prueba funcional(PF-B) se ha hecho de esta forma:

1. Primero configuramos el PhantomJS como en la primera prueba
2. Después Iniciamos el navegador y cargamos "index.html"
3. Buscamos el elemento "txtOtros" por nombre y escribimos "JugadorNuevo"
4. Buscamos el radio button "radioButtonOtros" por id y hacemos click sobre él.
5. Buscamos el botón "Votar" por nombre y lo hacemos click sobre él.
6. Después vamos a la pagina inicial "index.html"
7. Buscamos el botón " Ver votos" por id y lo hacemos click sobre el
8. En la página de "Ver votos" buscamos una tabla, después el "tbody" de la tabla, después buscamos la fila del jugador "JugadorNuevo", una vez encontrada la fila buscamos la celda del jugador y comprobamos que la celda de los votos sea igual a 1.

```
// Verifica que el nuevo jugador tiene 1 voto en la página "VerVotos.jsp"
WebElement tablaVotos = driver.findElement(By.tagName("table"));
WebElement cuerpoTabla = tablaVotos.findElement(By.tagName("tbody"));
WebElement filaNuevoJugador = cuerpoTabla.findElement(By.xpath("//tr[contains(., 'JugadorNuevo')]"));
WebElement celdaVotosNuevoJugador =
filaNuevoJugador.findElements(By.tagName("td")).get(1);
assertEquals("1", celdaVotosNuevoJugador.getText());
```

