Reto Malackathon 2024

Grupo APND

Integrantes:

- Pablo Márquez Benitez
- Pablo Miguel Aguilar Blanco
- Antonio Salvador Gámez Zafra
- Adrián Fernández Vera
- Jaime Ezequiel Rodríguez Rodríguez

Fase 1

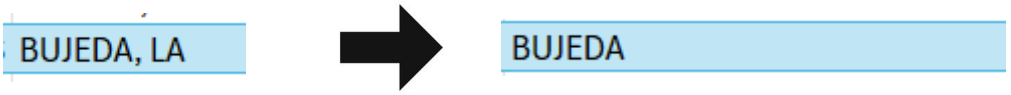
En primer lugar hemos echado un vistazo a los datos de los que disponemos, para unir la tabla **EMBALSES** con la tabla **LISTADO_EMBALSES**

Hemos observado que la columna **AMBITO_NOMBRE** (tabla EMBALSE) y **DEMARC** (tabla LISTADO_EMBALSES) que se corresponde con **cuenca hidrográfica** o **área geográfica** donde se encuentra el embalse, son iguales, pero al no ser valores únicos (ya que varios embalses pueden estar en la misma cuenca) no nos servirán para relacionar las tablas, sin embargo, la columna **Embalse_nombre** (tabla EMBALSE) y **nombre**(tabla LISTADO_EMBALSES) si son parecidos y **únicos** ya que es el nombre del propio embalse.

Pero nos encontramos con un problema, esas columnas representan la misma información pero no está escrita de la misma manera, por ejemplo hay veces que tienen un artículo como “El” a la izquierda en una tabla y a la derecha en la otra, para solucionar esto hemos **normalizado** la columna **eliminando** tanto los **artículos** (el,la,los,las…) como las **‘,’** para que los nombres sean iguales y únicos. Para casos restantes y aislados donde los nombres del mismo embalse sigan siendo diferentes tras el procesamiento se han cambiado a mano al ser pocos estos casos.

Una vez normalizadas esas columnas ya podremos unir ambas tablas. Para la visualización y procesamiento de los datos se ha usado tanto la propia web de **Oracle** como **Excel**.

Ejemplo de procesamiento:



Fase 2

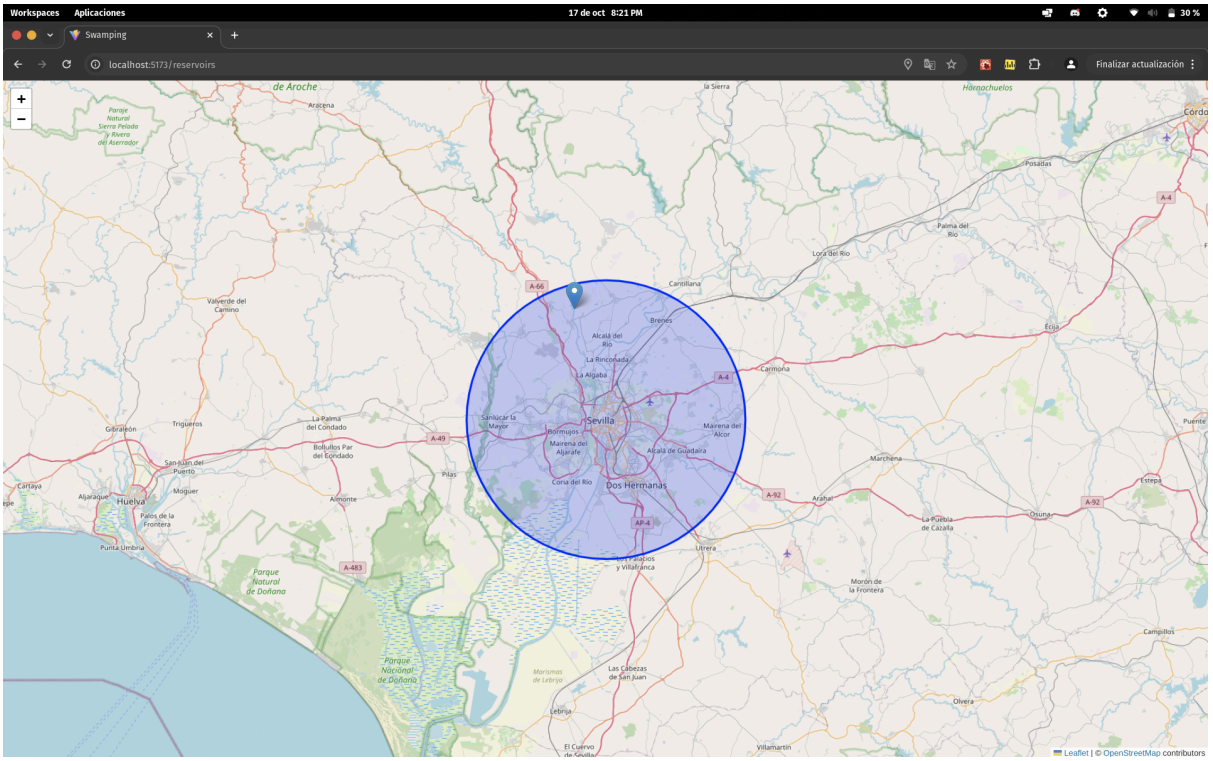
Para el desarrollo de la aplicación web, hemos optado por utilizar **React** para el **frontend**, lo que nos permite garantizar la implementación en múltiples dispositivos. En cuanto al **backend**, hemos decidido integrar directamente la **API REST** proporcionada por **Oracle**. Esta elección se debe a limitaciones de tiempo que nos impiden desarrollar un backend propio de manera adecuada.

Además, hemos llevado a cabo pruebas unitarias y de carga para asegurar la calidad de nuestro software. Ambas se encuentran dentro de la carpeta Codigos_Test. Está la carpeta raíz Test con el código java y los scripts de las pruebas de carga.

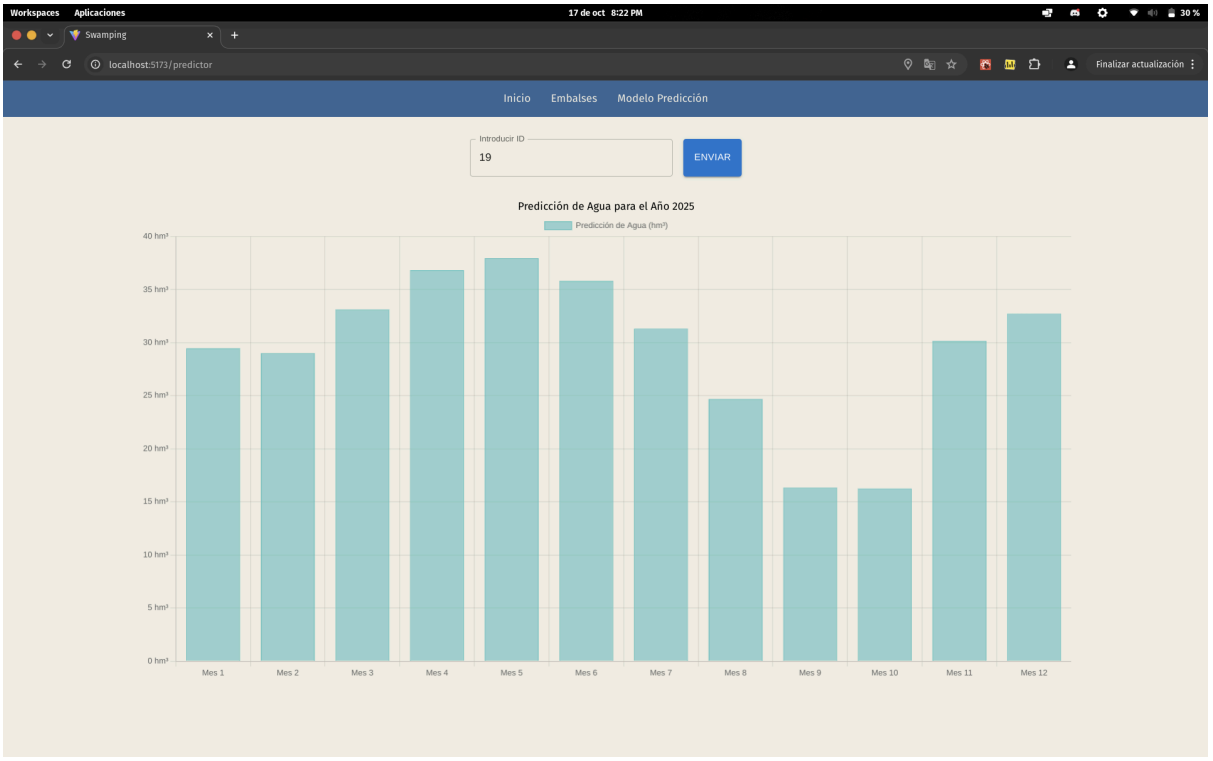
Pruebas Unitarias: Utilizamos **JUnit** para realizar verificaciones en la API de Oracle, siguiendo el enfoque **AAA** (Arrange, Act, Assert), que comprende las fases de inicialización, ejecución y verificación. A continuación, se presenta un ejemplo de prueba.

Página de inicio.

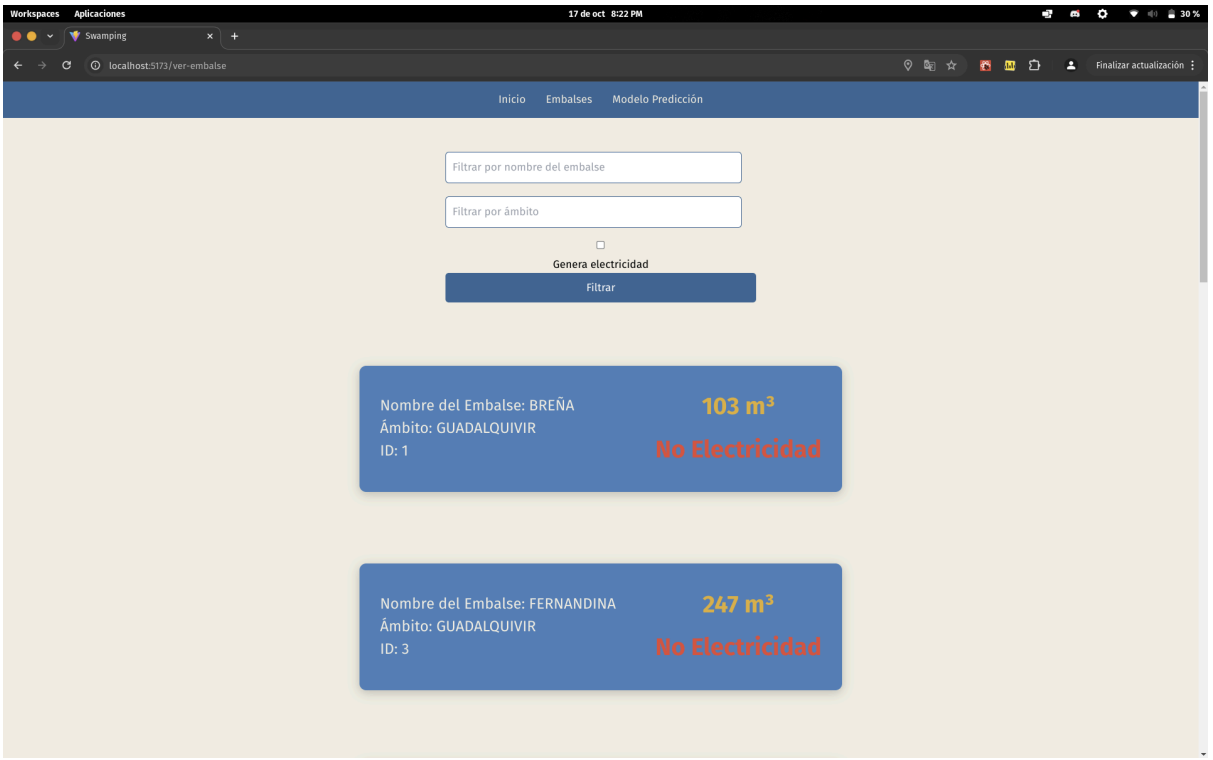
Página de mapa.



Página de predicciones.



Búsqueda de embalse por texto.

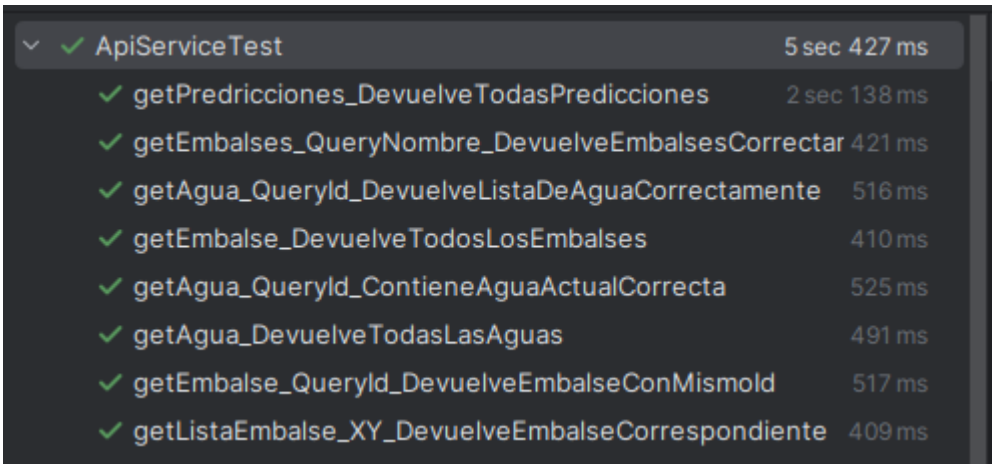


```java
@Test
@DisplayName("Hacer una peticion a la api de la tabla embalse, con una query usando el nombre, devuelve el listado de embalses correctamente")
public void getEmbalses_QueryNombre_DevuelveEmbalsesCorrectamente() {
    //Arrange
    String query = "{\"embalse_nombre\":\"FERNANDINA\"}";
    String nombreEsperado = "FERNANDINA";

    //Act
    Response response = RestAssured
            .given()
            .param( s: "q", query)
            .when()
            .get(urlEmbalses);

    List<String> nombresObtenidos = response.jsonPath().getList( path: "items.embalse_nombre");

    //Assert
    assertTrue(nombresObtenidos.contains(nombreEsperado));
}
```



Pruebas de Carga: Hemos desarrollado dos scripts utilizando la herramienta **k6**. El primero está diseñado para evaluar el rendimiento de la web bajo una **carga baja** de usuarios, mientras que el segundo tiene como objetivo identificar puntos de **rotura** manteniendo la web con alta carga.

Además, hemos implementado una prueba de tipo **"spike"** para simular un ataque DDoS, lo que nos permite analizar la capacidad de la aplicación para gestionar picos repentinos y extremos de tráfico.

La idea de estas pruebas de carga sería usarlo en la web alojada, pero por cuestiones de tiempo no pudimos subirla, así que las pruebas están hechas en local. Si alojasemos la web o montamos un servidor, sería cuestión de cambiar la url por la correcta. Además, las pruebas no están automatizadas, sería lo ideal que lo fueran, pero de nuevo, por cuestiones de tiempo, no pudimos.

Fase 3

Hemos intentado acceder a los datos del consumo de agua en España pero los datos que hemos encontrado eran demasiado pobres. Probamos a usar también los datos de *climate data store* sobre las temperaturas y las precipitaciones desde 1990 hasta 2060 pero no hemos podido conseguir los datos porque no responde.

Entrenamos el modelo para la fase 4 sin uso de datos adicionales y tenemos buenos resultados, así que decidimos no añadir más tablas a la base de datos y seguir con lo que tenemos.

Fase 4

Para obtener patrones de consumo de agua de los últimos cinco años y prever las necesidades para los próximos 12 meses en los embalses analizados, hemos elegido el método **Random Forest** como modelo de predicción. La decisión se basa en los siguientes motivos:

Rapidez y eficiencia: Random Forest es un modelo de aprendizaje supervisado relativamente ligero en comparación con otros modelos más complejos, como redes neuronales profundas, lo que lo hace ideal para aplicaciones web en tiempo real.

Implementación en el navegador: Dado que el sistema está desarrollado con **React**, es crucial garantizar tiempos de respuesta rápidos para ofrecer una experiencia fluida al usuario. Por este motivo, los resultados de las predicciones se han integrado en la **API REST de Oracle**, facilitando la accesibilidad y la gestión eficiente de los datos.

Robustez del modelo: Random Forest puede manejar datos con niveles moderados de ruido y minimiza el riesgo de sobreajuste, lo que garantiza que las predicciones sean confiables incluso ante variaciones en los datos históricos.

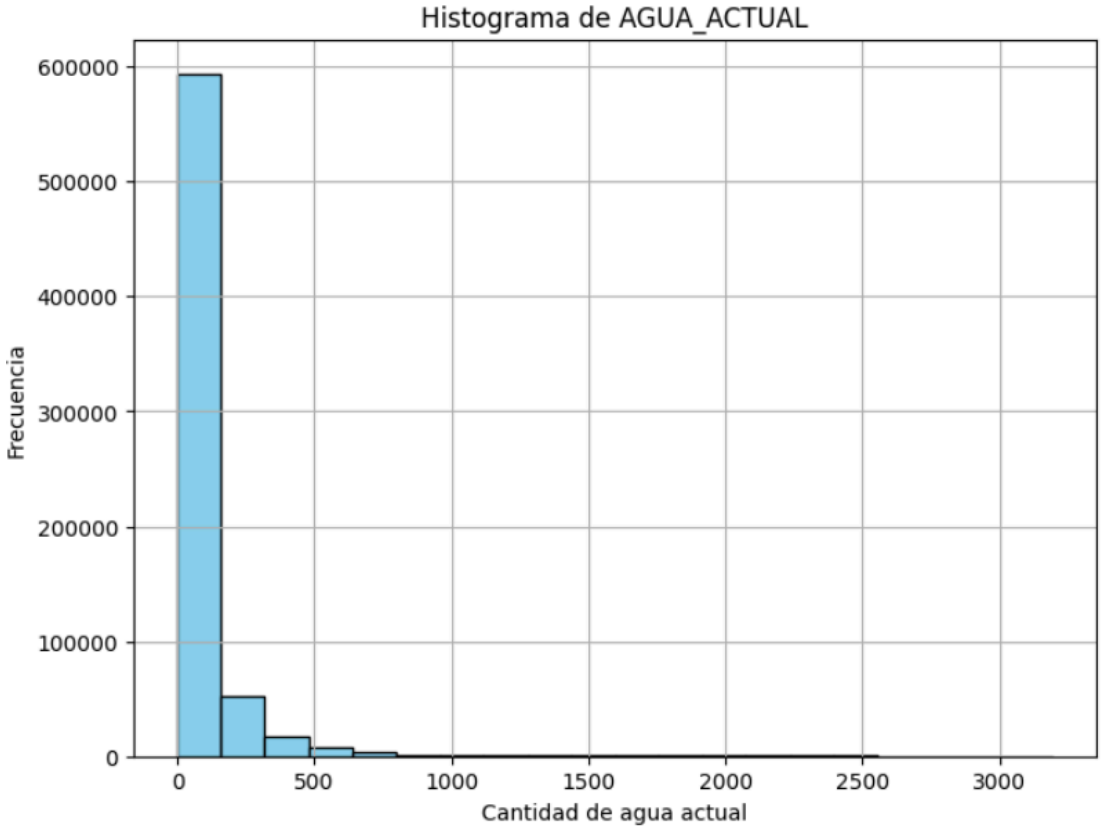
Escalabilidad y mantenimiento del sistema: Para garantizar la eficiencia operativa, hemos diseñado un esquema en el que uno de nuestros equipos generará las predicciones de consumo **mensualmente** para el conjunto de embalses. Las predicciones se actualizarán automáticamente en la página web, asegurando que los usuarios siempre dispongan de proyecciones actualizadas y precisas.

Resultados del Modelo

Podemos observar que el **coeficiente de determinación (R^2)** es muy cercano a 1, lo que indica una alta precisión en las predicciones. Cuanto más cercano a 1 sea este valor, mayor será la capacidad del modelo para explicar la variabilidad del consumo de agua y prever las necesidades con exactitud.

```
Error Cuadrático Medio (MSE): 55.689539790068046
Coeficiente de Determinación (R²): 0.9940322763431192
```

En el análisis exploratorio de los datos, hemos observado que la mayoría de los niveles de agua registrados se sitúan por debajo de 500.



Despliegue

Requisitos previos:

- Para lanzar el proyecto hay que asegurarse de tener instalado node.js en la maquina.

Pasos para realizar el despliegue a partir de la descarga del repositorio:

1. Descargar la carpeta del proyecto en github.
2. Descomprimir el archivo .zip
3. Acceder a la carpeta donde hemos descomprimido el proyecto desde una terminal.
4. Ejecutar el comando ‘npm install’
5. Hacer ‘npm run dev’
6. Acceder mediante el navegador a puerto localhost que nos indique la terminal