Malackathon

Reto ODS 6

Grupo de trabajo “Para Bellum”

Antonio Cañete Baena

Macorís Decena Giménez

Antonio Blas Moral Sánchez

Eulogio Quemada Torres

Alejandro Román Sánchez

ÍNDICE

1. Arquitectura del sistema, tecnologías utilizadas y mock-ups

Diagrama de despliegue del sistema

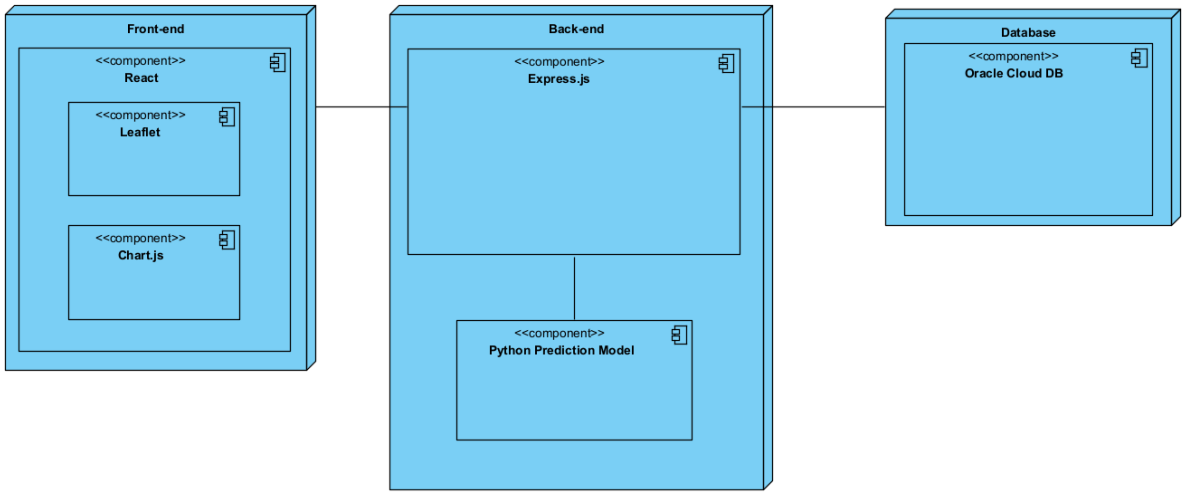

Figura 1. Diagrama de despliegue

Como se muestra en el diagrama, hemos estructurado la arquitectura mediante tres grandes bloques independientes: El front-end, el back-end y la BD.

Dentro del **front-end** está el componente correspondiente al framework empleado, React, y sus subcomponentes, relativos a los servicios externos que se usan: Leaflet para los mapas, y Chart.js para representaciones gráficas.

Dentro del **back-end** tenemos dos componentes. En primer lugar, el que usa la tecnología Express.js para comunicarse con los componentes “externos” a él: el front-end y la base de datos. Por otro lado, ese componente Express.js también se comunica con otro componente del back-end.

Para la **base de datos**, usamos la Oracle Cloud Infrastructure, donde se definen tres tablas con la siguiente información y relaciones:

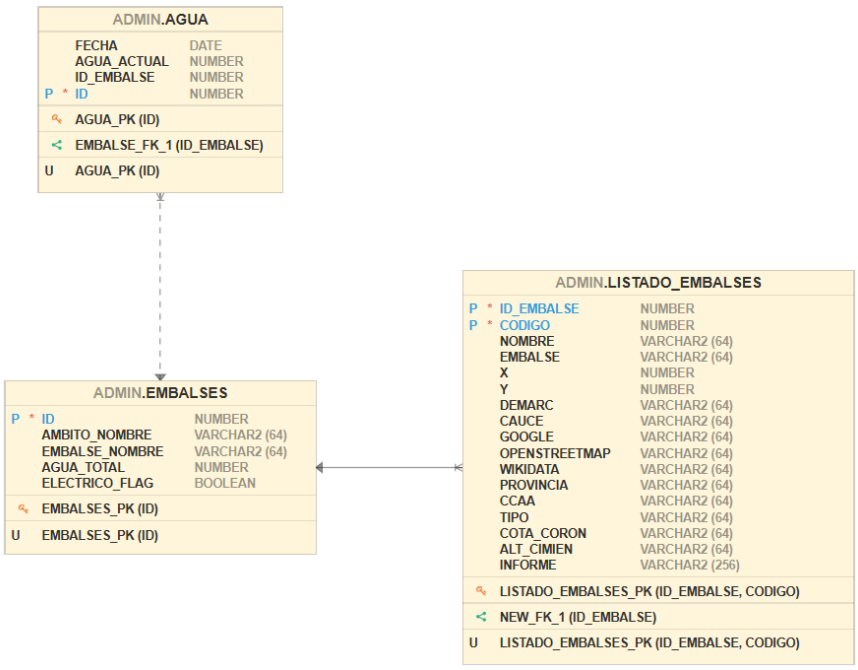

Figura 2. Diagrama de BD

Mock-ups de la interfaz del sistema

Diseñar maquetas del diseño back-end nos ha resultado muy útil para tener claro en todo momento lo que debíamos cumplir a nivel de organización estética. Echémosles un vistazo:

Figura 3. Mock-up de la página principal

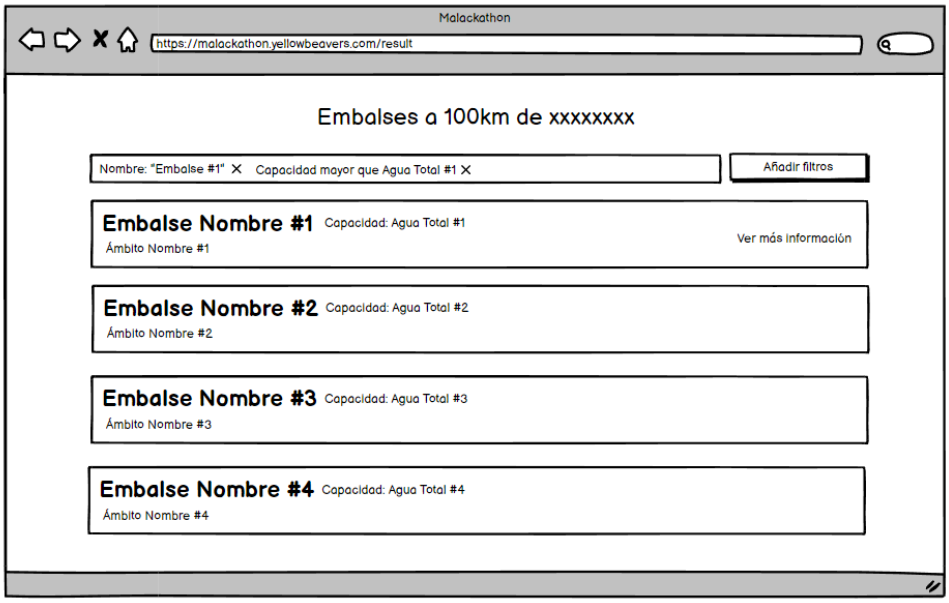

Figura 4. Mock-up del listado de embalses a un número de kilómetros

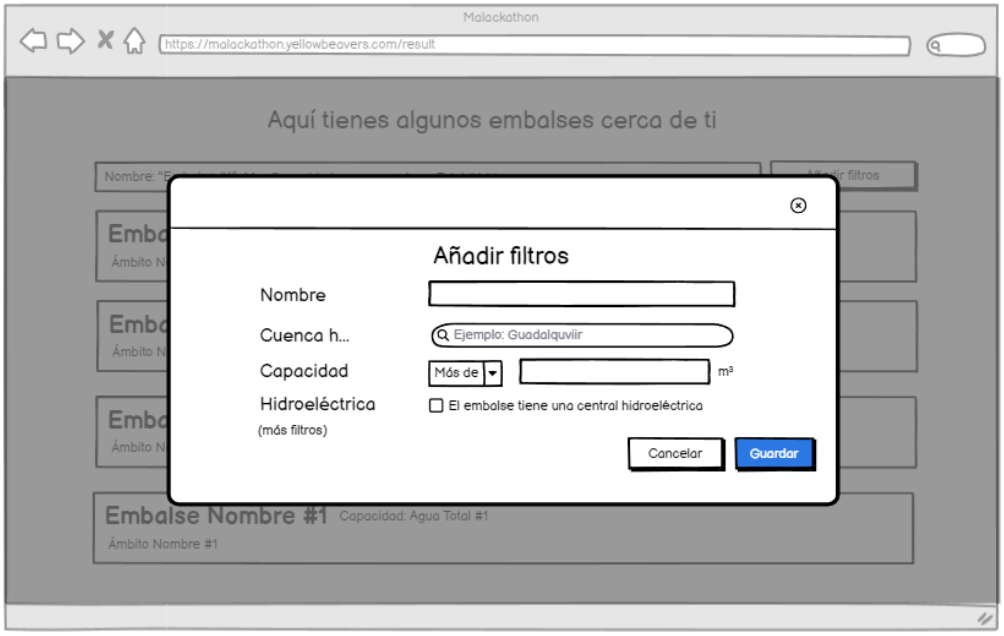

Figura 5. Mock-up del menú de filtrado de embalses

2. Desarrollo de los requisitos requeridos

A continuación se detalla cómo hemos abordado el conjunto de requisitos necesarios para satisfacer las necesidades extraídas del planteamiento del reto, incluyendo las preferencias de las distintas empresas:

Seguridad

Debido a los problemas que ha dado la base de datos inicialmente, como solución temporal dado el poco tiempo del que disponemos, hemos decidido dividir el back-end en dos partes: Express.js para asuntos de seguridad y FastApi (de Python) para la conexión a la base de datos, ya que, por algún motivo que desconocemos, la conexión a la base de datos funciona correctamente en FastApi pero no como debería en Express.

En cuanto a aspectos más directamente relacionados con la **seguridad**, hemos cubierto ciertas necesidades o posibles vulnerabilidades:

En primer lugar, hemos evitado el **web scraping** de diferentes maneras. Normalmente, al hacer web scrap con lenguajes como Python, aparecen muchos elementos comunes en la cabecera de los paquetes de red. Por ejemplo, hemos implementado una funcionalidad en el backend que detecta que, si el UserAgent es malicioso, directamente cierra la conexión con el servidor.

```
const scrapingUserAgents = [
    /scrapy/i,
    /selenium/i,
    /headlesschrome/i,
    /wget/i,
    /curl/i,
    /python-requests/i,
    /python/i,
    /web scraper/i,
    /Mozilla\/5.0 \(compatible; Scrapy\)/i,
    /Java\/1.8.0_*/i,
];

// Verifica si el User-Agent coincide con algún patrón de scraping
const isScraping = scrapingUserAgents.some(pattern => pattern.test(userAgent));

if (isScraping) {
    // Si se detecta scraping, retorna un error
    return res.status(403).json({ error: 'Acceso prohibido debido a actividad sospechosa.' });
}
```

Figura x. Primer mecanismo para evitar web scraping

Por otra parte, si por algún motivo el usuario malicioso supera esta medida de protección, hemos implementado un mecanismo que detecta si la página ha sido llamada mediante **web drivers**. En ese caso, la página redirige directamente a un reCaptcha v2 de Google, para así evitar el acceso automatizado (mediante bots) a la página.

```
if (navigator.webdriver) {
    alert("Acceso denegado: posible scraping detectado.");
    navigate('/recaptcha');
}
```

Figura x. Segundo mecanismo para evitar web scraping

Además, para evitar la **recarga constante** de una página (como utilizamos React, un usuario normal no debería necesitar recargas masivas), hemos introducido un umbral ajustable de recargas máximas en un tiempo determinado. De esta forma, si un bot intentara recargar constantemente la página, se le bloquearía y redirigiría al reCaptcha.

```
const refreshCount = localStorage.getItem('refreshCount') || 0;
if (refreshCount ≥ 3) {
    navigate('/recaptcha');
} else {
    localStorage.setItem('refreshCount', parseInt(refreshCount) + 1);
}       You, 1 hour ago • WS Secure

const resetRefreshCount = () ⇒ {
    localStorage.setItem('refreshCount', 0);
};
const timer = setTimeout(resetRefreshCount, 10000);

return () ⇒ {
    clearTimeout(timer);
};
[navigate]);
```

Figura x. Tercer mecanismo para evitar web scraping

En segundo lugar, para evitar ataques de **denegación de servicios**, hemos implementado un **límite de accesos** a la web en un tiempo determinado. Lo hemos logrado con una librería de Express llamada express-rate-limit. Con ella controlamos que, cada cierto tiempo (15 minutos), si recibe más de 100 peticiones, o bien se ralentiza o bien se deniega el acceso a la petición.

```
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // Limit each IP to 100 requests per `window` (here, per 15 minutes)
  standardHeaders: true, // Return rate limit info in the `RateLimit-*` headers
  legacyHeaders: false, // Disable the `X-RateLimit-*` headers
});

app.use(limiter);
```

Figura x. Mecanismo para evitar ataque DDoS

Por último, para evitar que un atacante pueda obtener **información** sobre los **detalles de implementación** del sistema (framework utilizado, versión del framework), hemos utilizado la librería Helmet de Express.

```
app.use(helmet({
  frameguard: { action: 'DENY' },
}));

app.use(helmet.hidePoweredBy({ setTo: "PHP 4.2.0" }));
```

Figura x. Mecanismo para evitar obtención de información de desarrollo

Calidad y escalabilidad

Gracias a la buena **estructuración** del código y a su bajo acoplamiento, hemos garantizado que nuestro sistema es escalable ante posibles cambios o actualizaciones futuras. Esto es gracias a la estructura del back-end, en la que se pueden añadir nuevos end points con mucha facilidad.

Hemos empleado **buenas prácticas** de programación en todo momento, añadiendo comentarios para documentar el código, siguiendo estándares como puede ser el AAA de Python, entre otros.

Accesibilidad y usabilidad

En primer lugar en cuanto a la accesibilidad, hemos garantizado el **diseño responsivo** de la aplicación web, de tal forma que su contenido pueda visualizarse correctamente dependiendo del tipo de dispositivo desde el que se navegue, adaptando la interfaz a cada dimensión de pantalla.

Figura x. Diseño responsivo de una de las páginas visitándola desde ordenador

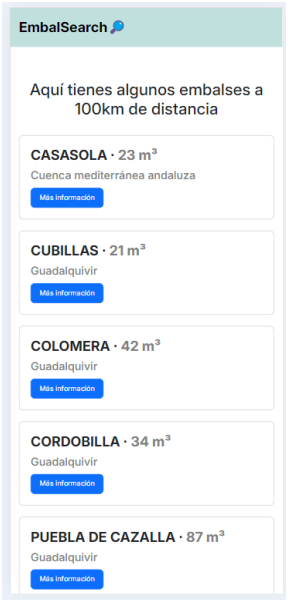

Figura x. Diseño responsivo de una de las páginas visitándola desde móvil

En términos de accesibilidad, también resulta vital garantizar la inclusión de la mayor cantidad posible de la población para que puedan navegar sin obstáculos dados por discapacidades o impedimentos físicos.

Por ello, en primer lugar, hemos garantizado un acceso a la aplicación **completamente desde teclado**, de tal modo que los usuarios que empleen teclados adaptados a sus discapacidades puedan navegar sin problemas en nuestro sistema.

Por otro lado, es importante tener en cuenta a los usuarios con problemas de visión. El estándar de accesibilidad en este aspecto se alinea con el uso de **narradores de texto**. Por ello, hemos garantizado que nuestro sistema sea compatible con narradores, usando para nuestras pruebas el narrador NVDA, de tal modo que la información de la aplicación pueda consultarse de forma auditiva.

Cabe destacar, también, que hemos utilizado herramientas como **WAVE** o **Axe Devtools** (extensiones de Chrome), que comprueban automáticamente el cumplimiento de los criterios de accesibilidad WCAG, y el resultado de los analizadores en nuestra aplicación es exitoso.

Rendimiento y eficiencia

En cuanto al rendimiento y eficiencia computacional, hemos llevado a cabo varias decisiones de diseño para **evitar una sobrecarga computacional** en nuestro servidor. Nuestra filosofía, en este sentido, es intentar recibir el mínimo procesamiento posible, siendo realistas, en nuestro servidor, es decir, procurando que los filtros y procesados simples se realicen en la parte del usuario. De esta forma, la única carga computacional de la parte del servidor sería procesar una única consulta por página.

Además, al adoptar este enfoque, se fomenta una personalización de la **experiencia del usuario y mejor capacidad de respuesta**. Los usuarios pueden aplicar filtros según sus necesidades específicas, lo que no solo optimiza la presentación de datos, sino que también les permite acceder rápidamente a la información relevante sin esperar a que el servidor procese cada solicitud. Esta arquitectura distribuida contribuye a una mejor utilización de los recursos, ya que el servidor se mantiene ágil y receptivo, lo que resulta en una aplicación más capaz de adaptarse a la demanda creciente sin comprometer su rendimiento. Además, esto solucionaría potenciales problemas momentáneos de conexión del lado del cliente si solo está haciendo filtros en vez de nuevas consultas al servidor.

Análisis de datos y modelo de predicción

Para llevar a cabo un buen análisis de datos, primero es fundamental realizar un **preprocesado** del conjunto de datos en cuestión. Por ello, hemos usado el lenguaje R para preparar los datos de entrada y eliminar información con valores nulos o poco útiles para nuestro análisis.

Hemos recogido **datos de AEMET** de las cuencas hidrográficas españolas, concretamente, los datos fluviales y temperaturas anuales de las principales ciudades por los que pasan los ríos de dichas cuencas.

Para la **predicción** de datos futuros hemos empleado, en Python, el modelo **ARIMA**, obteniendo valores de R cuadrático bastante buenos, gracias a la antigüedad de los datos. Sabemos que ARIMA no es el mejor modelo, y por ello hemos buscado datos más actuales con los que podríamos crear nuevos modelos de predicción más precisos. Sin embargo, nos hemos quedado sin tiempo para realizar este tipo de optimizaciones.

Impacto real del sistema: optimización del agua

Para cumplir la última fase del reto, hemos seguido una **estrategia** basada en tomar la predicción de los embalses que están en el radio especificado por el usuario y sugerir un posible trasvase entre los embalses cuya capacidad se espera que crezca a los que tienen capacidad que se espera que decrezcan.

3. Metodología de gestión de proyecto

Equipo de desarrollo y roles

El equipo “Para Bellum” está conformado por los siguientes miembros:

- **Antonio Cañete Baena**: responsable de Back-end y Devops.
- **Macorís Decena Giménez**: responsable de Accesibilidad y Documentación, Scrum master.
- **Antonio Blas Moral Sánchez**: responsable de Front-end y Mock-ups.
- **Eulogio Quemada Torres**: Responsable de Back-end y Base de datos.
- **Alejandro Román Sánchez**: Responsable de Seguridad y Back-end.

Metodología Scrum adaptada

Independientemente de la escala de un proyecto de cualquier ámbito, ya sea de dos años de duración o de 12 horas, vemos un sinsentido comenzar precipitadamente sin planificarnos. Basándonos en este principio, hemos considerado adecuado basarnos en la metodología **Scrum**, con la que ya estamos familiarizados, para adaptarla a las necesidades del reto y emplearla como metodología de planificación de nuestro proyecto.

La adaptación en cuestión de la metodología Scrum que hemos planteado ha consistido en hacer una concatenación de **Scrum meetings** (a veces opcionales, dependiendo de la franja horaria) y **Sprints** de trabajo poco duraderos pero intensos en lo relativo a la eficiencia. La duración de estos Sprints puede variar, dada la naturaleza del reto, ya que resulta más adecuado usar, por ejemplo, franjas de descanso (para comer, etc.) para realizar los Scrum meetings intermedios.

Además, tal y como se suele hacer al adoptar una metodología Scrum, decidimos que usaríamos una herramienta software para planificarnos: Trello. Ya la habíamos usado antes, con lo que estábamos habituados a su interfaz y funcionamiento.

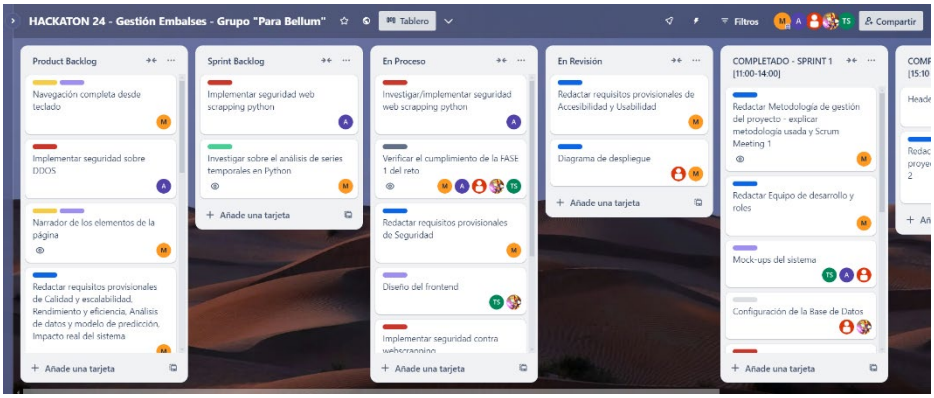

Figura x. Tablero de Trello en una parte intermedia del desarrollo

Particularmente, la planificación horaria ha resultado como se detalla a continuación. Usaremos la técnica de **storytelling** para ilustrar el proceso:

1. **Primer Scrum meeting [10:50 - 11:00]**

Tras escuchar el planteamiento del reto y hacer una breve parada en el baño, dijimos que **nadie tocaría un ordenador** hasta clarificar la **planificación** general del reto. Por ello, nos sentamos al aire libre, en una mesa del patio, y durante 10 efectivos minutos hablamos para dividir nuestros **roles**, en función de las habilidades y puntos fuertes de cada miembro, y atendiendo también a los requisitos del sistema y las preferencias de las empresas.

También tomamos las decisiones pertinentes sobre el diseño de la arquitectura y la metodología de planificación del proyecto.

2. **Primer Sprint [11:00 – 14:00]**

Durante el primer Sprint, comenzamos a crear:

- Para definir la **arquitectura** y estructura del sistema, hicimos el diagrama de despliegue.
- A nivel de **base de datos**, se construyeron las relaciones pertinentes.
- Hubo avances sustanciales en el ámbito del **back-end**, tras algunos problemas con la base de datos. Concretamente, se estructuró el back-end básico y se desarrollaron endpoints.
- Para encaminar el **front-end**, dedicamos un esfuerzo considerable a construir los mock-ups de la aplicación, con el objetivo de tener claros y definidos los diseños que debemos implementar de cara al usuario final. Creemos que es muy importante diseñar y pensar antes de hacer.
- A nivel de **datos** del dataset, se hizo únicamente un preprocesado de los datos, con el fin de “limpiarlos” y poder extraer la información útil sobre los embalses. El preprocesado es indispensable antes de cualquier análisis.
- En cuanto a la **seguridad**, se planteó todo lo que debería cubrir el sistema y se hicieron las primeras investigaciones y pruebas sobre ello.
- Para la accesibilidad, se instalaron las herramientas pertinentes y se hizo una investigación de estándares, criterios y buenas prácticas a la hora de aumentar las posibilidades de acceso a la aplicación de usuarios con discapacidades.

En resumen, este primer Sprint se podría resumir con el principio de que es esencial **hablar, planificar y pensar** antes de **hacer**.

3. **Segundo Scrum meeting [15:00 – 15:10]**

En el inicio del segundo Scrum meeting comentamos, por turnos, lo que había hecho cada uno en el primer Sprint, y lo que haríamos en el segundo.

Para terminar esta breve reunión, revisamos entre todos el tablero de Trello, por si hubiera alguna inconsistencia o faltara alguna tarea por añadir.

4. **Segundo Sprint [15:10 – 17:30]**

En el segundo Sprint del proyecto, nuestras tareas fueron las siguientes:

- Se continuó en gran medida con el desarrollo del **back-end**.
- El desarrollo del **front-end** avanzó a buen ritmo, quedando prácticamente completadas las vistas de las diferentes páginas.
- La incorporación de aspectos de **seguridad** avanzó increíblemente rápido, incluyéndose casi todas las funcionalidades que se encuentran en el sistema final.

5. **Tercer y último Scrum meeting [17:30 – 17:35]**

En el último Scrum meeting, que fue especialmente breve debido a la prisa, comentamos lo que habíamos hecho y lo último que nos quedaba por hacer a cada uno.

6. **Tercer y último Sprint [17:35 – 20:45]**

En la recta final del desarrollo, terminamos todo lo correspondiente al **backend** y **front-end**, así como los aspectos finales para garantizar la **seguridad**, **accesibilidad**, **usabilidad** y **calidad** del sistema. También iniciamos y completamos muchas tareas en cuanto al **análisis de datos**.

4. Cumplimiento de las fases del reto

Una vez llegados a los últimos minutos del desarrollo, podemos afirmar que hemos llegado a la **fase 5 del reto**, cumpliendo satisfactoriamente los requisitos que se han planteado hasta esa fase.

A pesar del **poco tiempo**, hemos preservado la intención de querer mantener una **buena calidad** en todas las fases. Estamos muy satisfechos con nuestro rendimiento.