PROYECTO “HUNDIR LA FLOTA”

DOCUMENTACIÓN DE ARQUITECTURA GENERAL

INTRODUCCIÓN

Este documento describe la **arquitectura general del proyecto “Hundir la Flota”**, una aplicación web desarrollada para un solo jugador.
El objetivo principal es mostrar cómo se comunican las diferentes capas del sistema (Front-End, Back-End y gestión de proyecto) y cómo se organiza el flujo de información entre ellas.

El proyecto se divide en tres áreas:

- **Front-End (JavaScript):** interfaz del jugador y lógica visual.
- **Back-End (PHP):** generación de datos, validación y persistencia.
- **Gestión de proyecto (Scrum):** organización, documentación y coordinación del equipo.

OBJETIVO DEL DIAGRAMA

El diagrama UML elaborado con **PlantUML** (archivo architecture.puml) tiene como finalidad representar gráficamente la **estructura lógica del proyecto** y las **relaciones entre sus componentes**.

Permite entender:

- Qué módulos existen en el sistema.
- Qué métodos principales tiene cada uno.
- Cómo se comunican entre sí.
- Qué responsabilidades tiene cada rol del equipo.

DESCRIPCIÓN DE LOS COMPONENTES

3.1. Back-End (PHP)

El Back-End constituye el **cerebro del juego**.
Su tarea es generar la flota de barcos, gestionar las puntuaciones y devolver los datos en formato JSON al cliente (JavaScript).

Se compone de los siguientes módulos principales:

Clase / Script	Responsabilidad	Métodos principales
StartGame	Genera la flota de barcos y las posiciones aleatorias.	generateFleet(), placeShip(), validatePosition(), toJSON()
SaveScore	Guarda las puntuaciones del jugador en un archivo JSON.	receivePOST(), updateScores(), sortScores(), saveToFile()
GetScores	Devuelve el ranking de puntuaciones guardadas.	readFile(), toJSON()
ScoreFile	Gestiona la lectura y escritura del archivo scores.json.	read(), write()

Flujo básico del Back-End:

1. StartGame genera una nueva partida.
2. SaveScore recibe las puntuaciones al finalizar el juego.
3. GetScores devuelve el ranking cuando se solicita.
4. ScoreFile es utilizado por los otros tres para manipular los datos del archivo scores.json.

3.2. Front-End (JavaScript)

El Front-End gestiona toda la **interacción del jugador** y la **presentación visual** del juego.
Se comunica con el servidor mediante peticiones fetch() y funciones asíncronas (async/await), enviando y recibiendo datos en formato JSON.

Sus principales módulos son:

Clase / Script	Función	Métodos principales
Game	Controla el flujo principal del juego.	startGame(), shoot(), checkVictory()
Board	Dibuja el tablero y actualiza el estado de cada celda.	render(), updateCell()
Ship	Representa un barco, su tamaño y su estado ("tocado" o "hundido").	isSunk()
ScoreManager	Se encarga de guardar y recuperar puntuaciones desde el servidor.	saveScore(), getScores(), renderRanking()

Flujo básico del Front-End:

1. Al iniciar el juego, Game.startGame() solicita los datos de la flota al servidor (start_game.php).
2. El tablero se genera dinámicamente con Board.render().
3. Cada disparo se gestiona con Game.shoot().
4. Al finalizar, se envía la puntuación con ScoreManager.saveScore().
5. El ranking se muestra con ScoreManager.getScores().

3.3. Comunicación entre Front-End y Back-End

La comunicación se realiza mediante **peticiones HTTP** y el intercambio de **datos JSON**.

Dirección	Archivo PHP	Descripción
JavaScript → PHP (GET)	start_game.php	Solicita una nueva partida.
JavaScript → PHP (POST)	save_score.php	Envía el nombre y la puntuación del jugador.
JavaScript ← PHP (GET)	get_scores.php	Devuelve el ranking de puntuaciones.

Ejemplo de intercambio JSON:

- PHP envía un array codificado con json_encode().
- JavaScript lo interpreta con .json() al recibirlo mediante fetch().
- Para enviar datos, JS usa JSON.stringify() antes de hacer el POST.

3.4. Gestión del Proyecto (Scrum Master)

El **Scrum Master** no programa, pero se encarga de:

- Crear el repositorio y definir las ramas (main, develop, feature/).
- Coordinar al equipo Front-End y Back-End.
- Revisar las Pull Requests.
- Mantener actualizada la documentación (README, diagramas, etc.).

Sus funciones principales reflejadas en el diagrama son:

- crearRepositorio()
- definirRamas()
- coordinarEquipo()
- documentarREADME()

RELACIÓN ENTRE COMPONENTES

El flujo completo del sistema puede resumirse así:

1. El jugador abre la aplicación (Front-End).
2. El cliente JS solicita una nueva partida al servidor (start_game.php).
3. El servidor PHP genera la flota, valida posiciones y devuelve un JSON.
4. El jugador realiza disparos; el JS actualiza la interfaz.
5. Cuando termina, el cliente envía la puntuación a save_score.php.
6. El servidor actualiza el archivo scores.json.
7. El ranking se obtiene mediante get_scores.php y se muestra al jugador.

Resumen gráfico del flujo de información:

Jugador → Game.js → start_game.php → JSON ↔ save_score.php / get_scores.php

---

USO DEL DIAGRAMA UML

El archivo del diagrama (architecture.puml) se encuentra en la carpeta docs/diagrams/.
Este archivo se puede visualizar de tres formas:

Opción 1:
Abrirlo en Visual Studio Code con la extensión *PlantUML* y presionar Alt + D para ver el diagrama.

Opción 2:
Pegar el contenido en el visor online de PlantUML:
https://plantuml-editor.kkeisuke.app/

Opción 3:
Exportarlo a imagen desde VS Code (clic derecho → "Export as PNG")
y referenciarlo en el README del proyecto.

CONCLUSIÓN

El diagrama de arquitectura permite entender **cómo interactúan los diferentes componentes** del proyecto "Hundir la Flota".

- El **Back-End** se encarga de la lógica del juego, la generación de datos y la persistencia.
- El **Front-End** gestiona la visualización y la interacción del jugador.
- El **Scrum Master** coordina la comunicación y documentación del equipo.

Gracias a esta estructura modular y al uso del formato JSON para la comunicación, el proyecto mantiene una separación clara entre las capas, facilitando la colaboración y el mantenimiento del código.