Trabajo de Fin de Título

A-Tirma WebWare v.3: Infraestructura web para el seguimiento, monitorización, control y análisis de misiones para vehículos autónomos marinos

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Juan Pereiro González

TUTORIZADO POR:
Antonio Carlos Domínguez Brito
Jorge Cabrera Gámez

Fecha Julio/2025

ULPGC Universidad de Las Palmas de Gran Canaria | EII ESCUELA DE INGENIERÍA INFORMÁTICA

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D Juan Pereiro González, autor del trabajo A-Tirma WebWare v.3: Infraestructura web para el seguimiento, monitorización, control y análisis de misiones para vehículos autónomos marinos, correspondiente a la titulación Ingeniería Informática.

SOLICITA

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida, haciendo constar que

[x] se autoriza / [ ] no se autoriza la grabación en audio de la exposición y turno de preguntas.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[x] Se ha iniciado o hay intención de iniciarlo (defensa no pública).

[ ] No está previsto.

Y para que así conste firma la presente.

El/La estudiante

Fdo.————————

A rellenar y firmar **obligatoriamente** por el/la/los/las tutores

En relación con la presente solicitud, se informa:(firmar donde corresponda)

Positivamente (en caso de detección de copia, esta firma quedará invalidada)	Negativamente (justificación en TFT05)
Fdo.————————	Fdo.————————

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Agradecimientos

A todo el personal de la Escuela de Ingeniería Informática, por brindarnos unas condiciones y una formación a la altura de la institución.

A mis tutores del TFT, Jorge y Antonio, por su apoyo y orientación a lo largo de este proceso, siempre con profesionalismo y dedicación.

A mis amigos, por su compañía y por hacer este camino más ameno.

A mi familia, por su constante apoyo y por brindarme la libertad necesaria durante estos años.

Resumen

Este Trabajo Final de Título consiste en la implementación de una nueva iteración, concretamente la tercera versión, de la infraestructura web para la monitorización y control de los vehículos marinos autónomos de la División de Robótica y Oceanografía Computacional (ROC) del Instituto Universitario SIANI (SIANI). Los objetivos principales de esta nueva versión de la infraestructura consisten principalmente en poder incorporar varios vehículos a una misión, el adaptar y refactorizar la estructura de la base de datos para aceptar distintos tipos de vehículos, la mejora de la exportación e importación de misiones en formato .cvs, y finalmente una refactorización de la actual infraestructura para hacerla más escalable a futuros cambios y versiones.

Abstract

This Final Degree Project consists in the implementation of a new iteration, specifically the third version, of the web infrastructure for monitoring and controlling the marine autonomous vehicles at División de Robótica y Oceanografía Computacional (ROC) at Instituto Universitario SIANI (SIANI). The objectives of this new version of the infrastructure are mainly: to enable the integration of multiple vehicles into a simple mission; to adapt and refactor the data base structure to accept different types of vehicles; to improve the export and import of missions in .cvs format; and finally, to refactor the current infrastructure to make it more scalable to future changes and versions.

Índice general

Índice de Códigos

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción, Estado actual y objetivos iniciales

En este capítulo del Trabajo de Fin de Título, se tratará de proporcionar una visión resumida y desglosada del proyecto desde sus inicios, ahondando en el contexto y en la gestión de misiones de navegación, para luego presentar un análisis del estado actual de la infraestructura web, identificando áreas que requieran mejoras. Este análisis previo es determinante para fijar una línea inicial desde la cual se puedan medir las funcionalidades implementadas durante el proyecto.

1.1. Introducción

En la División de Robótica y Oceanografía Computacional del Instituto Universitario SIANI, se genera la necesidad de una gestión eficiente en las misiones de navegación de vehículos, siendo crucial para el éxito de los proyectos de investigación y exploración. Debido a la complejidad y la cantidad de datos generados durante estas misiones, se requieren herramientas, tales como una infraestructura web, que permita la visualización y gestión de estas misiones, con el objetivo de proporcionar a la División la capacidad de representar en una misión la información de varios vehículos, tanto en misiones en tiempo real como en misiones ya realizadas, ofreciendo la información adecuada a los usuarios de la web que deseen monitorizar las misiones del Instituto SIANI.

1.2. Estado inicial del proyecto

En esta sección se expondrá el estado inicial del proyecto con anterioridad a que la tercera versión haya sido instaurada, indicando los frameworks y herramientas de la infraestructura web y las funcionalidades implementadas en la versión anterior del proyecto, A-Tirma WebWare v.2 [6].

1.2.1. Frameworks y Herramientas

El sistema emplea como framework en el *frontend [1]* la herramienta *Vue 3 [44]*, y para el *backend* [8] se utiliza *Node.js [32]*.

- **Vuejs:** Framework utilizado en el frontend del sistema, atendiendo a [44]: *("Vuejs es un framework de JavaScript [22] para crear interfaces de usuario. Se basa en HTML [21], CSS [13] y JavaScript. Proporciona un modelo de programación declarativo basado en componentes que ayuda a desarrollar eficientemente interfaces de usuario de cualquier complejidad.").*
- **Vue-leaflet-rotatedmarker:** Es una extensión o plugin para la dependencia @vue-leaflet/vue-leaflet [46]. Vue-leaflet-rotatedmarker [43], se encuentra inspirada en la anterior versión mudin/vue2-leaflet-rotatedmarker [45], esta permite rotar marcadores con facilidad para vue3. Esta herramienta ha sido suprimida gracias a la implementación de un sistema de rotación más básico sin necesidad de utilizar librerías adicionales, se explica en la subsección 3.8.7.
- **Nodejs:** Este framework es usado en el backend desde el comienzo del proyecto A-Tirma WebWare. Es un entorno de ejecución de JavaScript de código abierto y multiplataforma, diseñado para construir aplicaciones del lado del servidor y redes con JavaScript. Permite ejecutar código JavaScript fuera de un navegador web, lo que lo hace útil para crear servidores, aplicaciones web, herramientas de línea de comandos y scripts.

1.2.2. Funciones

Las funciones más importantes que ya estaban implementadas en el proyecto, son las siguientes:

- **Reproducción de misiones:** El sistema es capaz de reproducir misiones tanto cargadas por ficheros CSV [14], como recibidas mediante *paquetes de telemetría* cuando las misiones se encuentran activas, i. e., en vivo o en directo.. En la versión A-Tirma-WebWare-v2 se implementaron funcionalidades que mejoran la visualización de las misiones, tales como:
  - Trayectoria del vehículo
  - *Waypoints* o puntos de paso
  - Datos recopilados
  - Mapas batimétricos
- **Recibir paquetes de telemetría de un vehículo por misión:** El sistema es capaz de recibir paquetes de telemetría con los datos del vehículo o simulador que se encuentre activo. Estos paquetes llegan al servidor backend, donde se almacenarán y se pasarán mediante un algoritmo de suscripciones al frontend para poder visualizar las misiones en directo.

- **Una gestión de permisos** que ofrece la seguridad de los datos del sistema, permitiendo visualizar o modificar las misiones a los usuarios que tengan los permisos pertinentes, existiendo dos tipos de roles para los usuarios: Administrador y Usuario.

1.2.3. Evaluación del estado inicial

Analizando el estado inicial del proyecto, se ha identificado que el principal desafío radica en el cambio de la estructura en la base de datos, puesto que uno de los objetivos iniciales del TFT consiste en añadir varios vehículos con *propiedades descriptivas* diferentes en una misión, teniendo que:

- **Refactorizar el código** del sistema para adaptarlo a la nueva arquitectura de la base de datos.
- **Modificar la reproducción de misiones** para optimizar y visualizar varios vehículos por misión .
- **Cambiar el *fichero descriptivo***, y en consecuencia el formato de la tabla de ese vehículo en la base de datos, para aceptar distintos paquetes de telemetría según el tipo de vehículo.

1.3. Objetivos iniciales

Los objetivos perseguidos en este TFT son los siguientes:

- **Refactorización de la infraestructura web de monitorización, control y seguimiento de misiones para poder visualizar varios vehículos autónomos en una misma misión.** Respecto a este objetivo, partimos de una versión donde solo se puede añadir un vehículo por misión. Debido a necesidades de la División de Robótica y Oceanográfica Computacional del Instituto Universitario SIANI, este proyecto se ha basado en permitir la posibilidad de añadir, en una misma misión, vehículos diferentes, tales como una boya, un barco a motor o un barco a vela, que tengan formatos de paquetes de telemetría distintos.
- **Exportar misiones en formato CSV.** Este apartado es importante, ya que al pasar de una versión a otra, la División de Robótica y Oceanográfica Computacional del Instituto Universitario SIANI necesitaba conservar las misiones ya realizadas. Por lo que la exportación de misiones en ficheros locales era estrictamente necesario, tanto para los registros de las misiones como para los waypoints de la misma.
- **Funcionalidad de cambiar las propiedades de cada vehículo.** De esta manera, la aplicación web podrá tener varios vehículos que reciban paquetes de telemetría distintos, así se consigue evitar el acoplamiento en los datos de los vehículos.

1.4. Estructura del Documento

El presente documento se estructura de la siguiente manera:

- **Capítulo 1. Introducción, estado y objetivo iniciales:** En el presente capítulo, se ahonda en el contexto y la justificación del estado inicial, objetivos y estructura del documento.
- **Capítulo 2. Competencias específicas y aportaciones al TFT:** Se nombran y justifican las competencias específicas citadas por la EII y las aportaciones de este proyecto.
- **Capítulo 3. Desarrollo:** Se describe el TFT realizado, incluyendo las decisiones técnicas, las funciones implementadas y las tecnologías utilizadas.
- **Capítulo 4. Conclusiones y Trabajo Futuro:** Se exponen las principales conclusiones del proyecto y se proponen nuevas implementaciones para mejorar y desarrollar el sistema.
- **Apéndice:** Se incluye un manual de despliegue de la aplicación web en un servidor y un glosario de términos técnicos utilizados a lo largo del documento.
- **Bibliografía:** Se relacionan las referencias bibliográficas consultadas para el desarrollo del proyecto.

Con este documento se pretende proporcionar una visión completa y detallada del proyecto A-Tirma WebWare v3, dándole relevancia a las aportaciones realizadas al campo de la gestión y visualización de misiones.

Capítulo 2

Competencias específicas y aportaciones del trabajo

En este capítulo se detallan las competencias específicas utilizadas durante el TFT. Cada subsección de la primera sección de este capítulo se centra en definir una competencia particular, exponiendo cómo se ha aplicado en el contexto del proyecto, permitiendo la debida comprensión del impacto del proyecto en el desarrollo profesional del autor, así como en el campo genérico de la ingeniería informática. Para finalizar, se especificará la aportación de este proyecto a la División de Robótica y Oceanografía Computacional del Instituto Universitario SIANI.

2.1. Competencias

2.1.1. CI2: Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

La tercera versión del proyecto A-Tirma WebWare destaca por la necesidad de planificar, desarrollar y dirigir un sistema web avanzado dedicado al seguimiento y análisis de misiones de vehículos autónomos marinos. Desde la planificación del proyecto, se fijaron objetivos claros para guiar la implementación eficiente del proyecto. Una de las principales contribuciones técnicas fue la implementación de un modelo de base de datos para vehículos con distintas propiedades descriptivas, desarrollado en la sección 3.2, lo cual permitió añadir distintos tipos de vehículos al sistema.

2.1.2. CI12: Conocimiento y aplicación de las características, funcionalidades y estructura de las bases de datos, que permitan su adecuado uso, y el diseño y el análisis e implementación de aplicaciones basadas en ellos.

Esta competencia específica se enfoca en el diseño y gestión de bases de datos SQL [39] (un lenguaje específico de dominio, diseñado para administrar y recuperar información de sistemas de gestión de bases de datos relacionales), para almacenar y administrar datos cruciales de misiones, destacando por su integración con XAMPP [50], herramienta que facilita la administración eficiente y el óptimo rendimiento de las bases de datos.

El rediseño de la estructura de la base de datos SQL fue crucial para cumplir con los objetivos del TFT. Además, la implementación cuidadosa garantiza no solo la integridad y la accesibilidad de los datos, sino también una operación eficiente y efectiva del sistema en su conjunto.

2.1.3. CI17: Capacidad para diseñar y evaluar interfaces persona-computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.

Esta competencia se centra en el diseño avanzado y la evaluación de interfaces de usuario para garantizar la accesibilidad y mejora de la usabilidad del sistema. El proyecto destaca por la creación de interfaces intuitivas, accesibles y fácilmente navegables, para los usuarios finales, utilizando tecnologías modernas como Vue.js y Vuetify [48], mejorando la estética del sistema y optimizando la experiencia del usuario.

2.2. Aportaciones

En esta sección se enumeran las aportaciones que este TFT proporciona tanto a la infraestructura web ya existente como a la propia División de Robótica y Oceanográfica Computacional del Instituto Universitario SIANI.

2.2.1. Aportaciones a la infraestructura web

Las aportaciones a la infraestructura web han sido:

1. **Múltiples vehículos en una misión**:
    - Implementación y mejora en el sistema para permitir añadir varios vehículos en una misma misión.
2. **Vehículos con propiedades descriptivas distintas**:

- Rediseño del sistema para aceptar varios vehículos con distintas propiedades descriptivas (boya, barco a motor, barco de vela) .

3. **Exportación de misiones**:
   - Incorporación de una funcionalidad para exportar misiones con el fin de almacenarlas como resguardo o importarlas a otras misiones.
4. **Mejora de la Interfaz de Usuario**:
   - Rediseño y optimización de la interfaz para hacerla más intuitiva, moderna y atractiva. Esto incluye la mejora del panel de gestión de los diferentes elementos (misiones, vehículos, usuarios, ficheros *CSV*) y la claridad en la reproducción de misiones.
5. **Mejora en la paginación de registros de las misiones**:
   - Optimización en la paginación de registros de las misiones para no sobrecargar el frontend de registros de los vehículos.
6. **Incremento de las Funcionalidades:**
   - Incorporación de nuevas funcionalidades y perfeccionamiento de las existentes, lo que permite a los usuarios realizar tareas de manera más rápida y eficiente.

2.2.2. Aportaciones a la División de Robótica y Oceanografía Computacional del Instituto Universitario SIANI

1. **Facilitación del Análisis de Datos:**
   - Mejora en la visualización y manejo de grandes volúmenes de datos generados durante las misiones de navegación, permitiendo a los investigadores realizar análisis más profundos y precisos.
2. **Fomento de la Colaboración Científica:**
   - Herramientas que promueven una colaboración más efectiva entre los miembros de la División de Robótica y Oceanografía Computacional, así como con investigadores externos.

Capítulo 3

Desarrollo y pruebas

En este capítulo se exponen las funcionalidades implementadas en la nueva versión del sistema desarrollado en este Trabajo de Fin de Título. Se comienza detallando las herramientas y frameworks utilizados, se continúa explicando los cambios en la arquitectura de la base de datos, la exportación e importación de misiones, las particularidades de las misiones en vivo, el nuevo método para reproducir misiones, tanto en tiempo real como históricas, y una variedad de mejoras generales implementadas en el sistema.

3.1. Frameworks y herramientas utilizadas

En esta sección se exponen todas las herramientas y frameworks que se utilizan en el TFT, tanto en las versiones anteriores, como en las utilizadas en el presente TFT.

3.1.1. XAMPP para la gestión de bases de datos SQL

XAMPP [50] es un servidor web local multiplataforma que permite la creación y prueba de páginas web u otros elementos de programación. Sus funcionalidades más destacadas consisten en la gestión de bases de datos SQL [39], a través de MySQL[30] o MariaDB[27]. Utilizando la herramienta phpMyAdmin [35] (aplicación web que sirve para administrar bases de datos de forma sencilla y con una interfaz amistosa) para la gestión de la base de datos en el ámbito de desarrollo. Según [41, 50] sus ventajas son las siguientes:

1. **Instalación y configuración sencilla:**
   XAMPP se instala fácilmente con un solo ejecutable y proporciona un entorno de servidor completo sin requerir configuraciones complejas.
2. **Entorno completo de desarrollo local:**
   Permite simular un servidor web real en la máquina local, lo que facilita el desarrollo, la prueba y la depuración de aplicaciones sin necesidad de conexión a Internet.

3. **MariaDB integrado:**
   MariaDB es una base de datos robusta, compatible con MySQL, ideal para trabajar con aplicaciones PHP. Viene incluida y configurada por defecto en XAMPP.

4. **phpMyAdmin como herramienta gráfica:**
   phpMyAdmin facilita la gestión de bases de datos MariaDB mediante una interfaz web intuitiva, evitando el uso exclusivo de la línea de comandos SQL.

5. **Multiplataforma:**
   XAMPP está disponible para Windows, Linux y macOS, permitiendo una experiencia consistente en distintos sistemas operativos.

3.1.2. Utilización de Vue.js para el desarrollo de interfaces

Vue.js [44] es un framework de JavaScript de código abierto diseñado para construir interfaces de usuario y aplicaciones web de una sola página (SPA, por sus siglas en inglés) [47]. Su principal atractivo radica en la facilidad de uso y en lo rápida que resulta su curva de aprendizaje. Según [44, 40] sus propiedades son las siguientes:

1. **Reactividad declarativa:**
   Vue detecta automáticamente los cambios en los datos y actualiza solo las partes del DOM [15] necesarias. Usa `Proxy` (Vue 3) para lograr una sincronización eficiente entre modelo y vista.

2. **DOM virtual:**
   Vue genera una copia en memoria del DOM [15] real y compara versiones para aplicar únicamente los cambios necesarios, mejorando el rendimiento y reduciendo operaciones costosas en el navegador.

3. **Componentes reutilizables:**
   Permite dividir la interfaz en componentes independientes que encapsulan su lógica, plantilla y estilos. Se comunican mediante `props` (Las props son atributos personalizados que se pueden pasar a un componente hijo desde un componente padre [37]) y eventos personalizados.

4. **Directivas personalizadas:**
   Vue amplía el HTML [21] con directivas como `v-if`, `v-for`, `v-bind` y `v-on`, permitiendo manipular el DOM de forma declarativa. También se pueden definir directivas propias.

3.1.3. Uso de Vuetify para la mejora en la UI

Vuetify [48] es una biblioteca basada en Vue.js [44] diseñada específicamente para el desarrollo de interfaces de usuario. Proporciona un amplio conjunto de componentes listos para usar, lo que facilita la construcción de aplicaciones con una apariencia coherente y profesional.

Al utilizar Vuetify [48], se garantiza que todos los elementos de la interfaz mantengan una estética uniforme a lo largo de toda la aplicación, gracias al diseño consistente de sus componentes. Entre los elementos que ofrece se incluyen tarjetas, listas, botones, menús de navegación y muchos otros componentes reutilizables que permiten acelerar el desarrollo y mejorar la experiencia de usuario.

3.1.4. Node.js para la implementación del backend

Node.js [32] es un entorno de ejecución de código abierto y multiplataforma que permite desarrollar aplicaciones del lado del servidor utilizando JavaScript [22]. Gracias a su arquitectura, los desarrolladores pueden crear todo tipo de herramientas y servicios backend de manera eficiente y escalable.

Características principales:

- **Asíncrono y orientado a eventos:** Node.js utiliza un modelo de ejecución no bloqueante, lo que significa que puede manejar múltiples operaciones al mismo tiempo sin esperar a que una termine antes de comenzar otra. Esto lo hace ideal para aplicaciones con gran cantidad de solicitudes simultáneas.
- **Multiplataforma:** Node.js es compatible con diversos sistemas operativos como Windows, macOS y Linux, lo que permite desarrollar aplicaciones portables.
- **Uso de JavaScript en el servidor:** Permite escribir código del lado del servidor con el mismo lenguaje que se usa en el navegador, lo que unifica el desarrollo frontend y backend.
- **Gestor de paquetes (npm) [33]:** Incluye npm (Node Package Manager), el sistema de gestión de paquetes más grande del mundo, que facilita la instalación y uso de librerías y herramientas desarrolladas por la comunidad.
- **Alta escalabilidad:** Su arquitectura basada en eventos y su capacidad para manejar muchas conexiones simultáneas lo hacen adecuado para aplicaciones escalables como APIs [3], chats en tiempo real y microservicios.

En el TFT, la herramienta Node.js es utilizada para implementar la lógica del servidor, conectarse con la base de datos o proveer servicios a la aplicación web mediante endpoints (punto final o endpoint es la ubicación de los recursos que una aplicación solicita a otra).

3.1.5. Lighttpd como Servidor Web

Lighttpd [26] es un servidor web diseñado para ser rápido, seguro, flexible, y fiel a los estándares. Está optimizado para entornos donde la velocidad es muy importante, y por eso consume menos CPU y memoria RAM que otros servidores. Por todo lo que ofrece, Lighttpd es apropiado para cualquier servidor que tenga problemas de carga. La información acerca de Lighttpd se ha consultado en [38]

3.1.6. Librerías Utilizadas

3.1.6.1. Backend

- **fs [18]:** Módulo de Node.js para interactuar con el sistema de archivos del servidor. De esta manera podemos modificar o borrar ficheros del servidor y evitar que el sistema se llene.
- **path [34]:** El módulo path es uno de los módulos incorporados de casa en Node.js y lo podemos usar para trabajar con rutas dentro del sistema de archivos. Es uno de los módulos más usados, sea cual sea el nivel de Node.js en el que se esté y resulta básico para muchas aplicaciones basadas en Node.
- **CORS [12]:** Es un mecanismo que permite la comunicación entre aplicaciones web que se encuentran en diferentes dominios. CORS (Cross-Origin Resource Sharing) establece reglas que permiten a un navegador acceder a recursos de un servidor externo, facilitando así la integración entre servicios alojados en distintos orígenes.
- **bcrypt [9]:** Esta librería es utilizada para cifrar las contraseñas de los usuarios antes de almacenarlas en la base de datos, garantizando su seguridad.
- **bcryptjs[10]:** Alternativa a `bcrypt` para el hashing de contraseñas en JavaScript. Ofrece una implementación completamente en JavaScript, útil en entornos donde no se pueden utilizar módulos nativos.
- **Express [17]:** Es un framework minimalista para Node.js que se utiliza en el desarrollo de aplicaciones web y APIs RESTful. Proporciona herramientas sencillas para definir rutas, gestionar solicitudes y respuestas, y utilizar middleware, lo que agiliza la creación de servidores web de manera eficiente.
- **jsonwebtoken [23]:** Es una biblioteca que permite generar y verificar JSON Web Tokens (JWT) en el lado del servidor. Los JWT son un formato compacto y seguro para intercambiar información entre diferentes partes, y se utilizan principalmente en procesos de autenticación y control de acceso en aplicaciones web actuales.
- **mysql2[31]:** Cliente MySQL para Node.js que permite conectar la base de datos MySQL con Node.js. Es una librería rápida y segura que proporciona una interfaz eficiente para realizar consultas y transacciones.
- **multer [29]:** Middleware de nodejs para subir archivos o ficheros al backend de manera segura y sencilla.

3.1.6.2. Frontend

- **mdi/font [28]:** Conjunto de iconos de Material Design fácilmente integrables en proyectos Vue.js. Ofrece una variedad de íconos que mejoran la apariencia visual y la experiencia de usuario.

- **vue-leaflet [46]:** Componente de Vue.js que permite utilizar la biblioteca Leaflet para incluir mapas interactivos, facilitando la visualización y manipulación de datos geoespaciales en tiempo real.
- **vueuse [49]:** Colección de composables reusables para Vue.js. El módulo `VueUse/Core` proporciona utilidades reactivas que simplifican la gestión del estado, eventos y el acceso al DOM.
- **axios [7]:** Cliente HTTP basado en promesas que permite realizar peticiones HTTP desde el navegador o desde Node.js de manera sencilla, ideal para interactuar con APIs.
- **core-js [11]:** Biblioteca que ofrece polyfills para funciones modernas de JavaScript, asegurando compatibilidad con navegadores antiguos y permitiendo el uso de características avanzadas del lenguaje.
- **leaflet-textpath [25]:** Complemento para Leaflet que permite mostrar texto siguiendo rutas o líneas en el mapa, útil para enriquecer la representación de datos geográficos.
- **pinia [19]:** Sistema de gestión de estados para Vue.js, sucesor de Vuex. Ofrece una API intuitiva, basada en la reactividad, para compartir datos entre componentes de manera eficiente.
- **vue-router [42]:** Sistema oficial de enrutamiento para Vue.js que permite gestionar la navegación entre vistas y componentes de forma estructurada y dinámica.
- **vuetify [48]:** Conjunto de componentes de Material Design para Vue.js. Facilita la creación de interfaces limpias, modernas y adaptables a cualquier dispositivo mediante componentes reutilizables.
- **dompurify [16]:** Herramienta de seguridad que limpia contenido HTML potencialmente malicioso, protegiendo contra ataques XSS al permitir únicamente elementos y atributos seguros.

3.2. Cambios en la Arquitectura de la Base de datos

La estructura de la base de datos y las relaciones entre sus tablas constituyen el esqueleto fundamental sobre el cual se construye todo el sistema de almacenamiento y gestión de datos. En esta sección, nos centraremos en el diseño detallado de la base de datos utilizada para almacenar la información relacionada con las misiones de navegación de vehículos de la División de Robótica y Oceanografía Computacional del Instituto Universitario SIANI.

Exploraremos en detalle las distintas tablas que componen la base de datos, identificando sus campos clave, su propósito y su relación con otras tablas dentro del esquema.

Es importante destacar que, para la implementación de esta base de datos, se ha optado por utilizar MariaDB [27], una base de datos relacional de código abierto, potente y versátil. Esta elección se debe a su compatibilidad con MySQL [30], su capacidad para manejar grandes

volúmenes de datos, así como su robustez en cuanto a seguridad y escalabilidad, lo que la convierte en una opción adecuada para el almacenamiento de datos críticos en este proyecto.

3.2.1. Estructura inicial

Tras realizar un análisis de la estructura inicial de la base de datos del proyecto, como se muestra en la ilustración 3.1 se identificaron dos tablas que fue necesario suprimir:

- La primera es la tabla **records**, se utilizaba para almacenar el identificador del *registro*, el *ID* correspondiente en la tabla `sensor_navigation`, el ID de la misión, el *timestamp* de recepción del registro, y un campo `data_source` que indicaba la procedencia del dato (por ejemplo, si provenía de una misión en vivo o si fue cargado mediante un archivo *CSV*).
- La segunda es la tabla **sensor_navigation**, que contenía los valores de telemetría asociados a cada registro. Esta tabla estaba estructurada con columnas fijas que representaban directamente las propiedades de los vehículos. Una de las razones principales para su eliminación fue la necesidad de dar soporte a distintos tipos de vehículos, cada uno con propiedades específicas. Esta variedad hacía inviable mantener una estructura estática, ya que no todos los vehículos comparten las mismas características.

Otra razón importante para reemplazar estas tablas fue el considerable crecimiento que experimentarían a lo largo del tiempo, ya que todos los datos de telemetría de vehículos y misiones se almacenaban inicialmente en ellas. El flujo consistía en guardar primero el ID del registro en `records`, y luego el paquete completo de datos en `sensor_navigation`. Esta solución no resultaba escalable ni flexible para una aplicación en constante evolución.

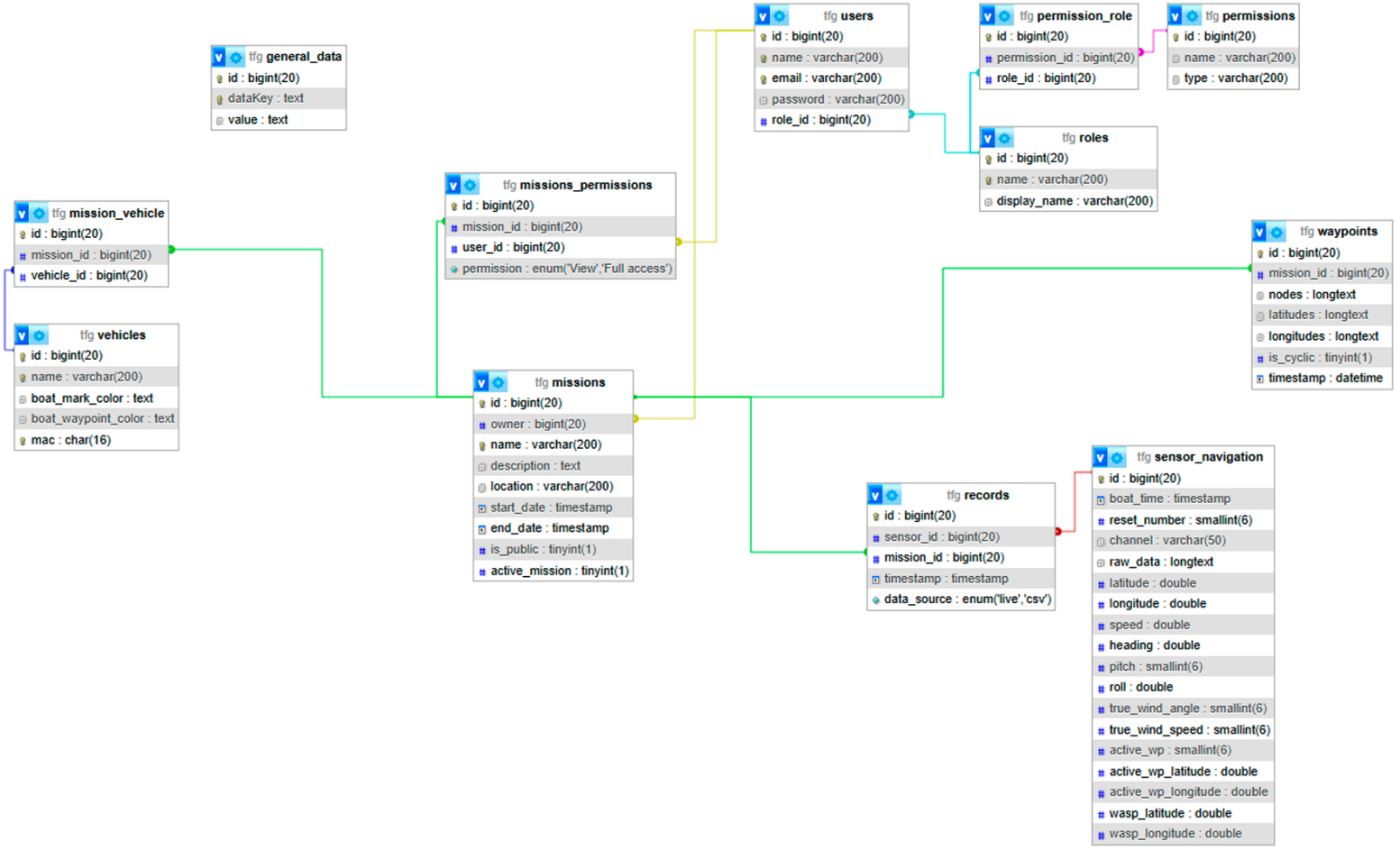


Ilustración 3.1: Estructura inicial de la base de datos

3.2.2. Tablas de registros para cada vehículo

Ante la situación explicada en el apartado 3.2.1, se ha decidido reemplazar ambas tablas por un sistema donde cada vehículo tendrá su propia tabla de registros, de esta manera, cada vehículo podrá tener columnas distintas a otros vehículos. Para ello, hemos introducido en la tabla `Vehicles` una columna para un fichero *JSON* que describirá el contenido de esta tabla. A este fichero lo llamaremos *fichero descriptivo*.

3.2.2.1. Fichero descriptivo

El fichero descriptivo es un archivo en formato JSON que se usa para definir la estructura de la tabla del vehículo. Este fichero definirá el formato de los *paquete de telemetría* que aceptará el vehículo en sus misiones.

```
{
  {
  "boat_latitude": {
    "type": "double",
    "visible": 1,
    "field": "latitude"
  },
  "boat_longitude": {
    "type": "double",
    "visible": 1,
    "field": "longitude"
  },
  "boat_speed": {
    "type": "double",
    "visible": 1
  },
  "pitch": {
    "type": "double",
    "visible": 1
  },
  "roll": {
    "type": "double",
    "visible": 1
  },
  "heading": {
    "type": "double",
    "visible": 1,
    "field": "heading"
  },
  "bearing": {
    "type": "double",
    "visible": 1
  },
  "rudder_angle_deg": {
    "type": "double",
    "visible": 1
```

```
  },
  "sail_angle_deg": {
    "type": "double",
    "visible": 1
  },
  "datetime": {
    "type": "timestamp",
    "visible": 1,
    "field": "time"
  },
  "appWindSpeedKnots": {
    "type": "double",
    "visible": 1,
    "field": "wind_speed"
  },
  "appWindAngleDeg": {
    "type": "double",
    "visible": 1,
    "field": "wind_angle"
  },
  "activeNode_latitude": {
    "type": "double",
    "visible": 1,
    "field": "wp_latitude"
  },
  "activeNode_longitude": {
    "type": "double",
    "visible": 1,
    "field": "wp_longitude"
  }
}
}
```

Código 3.1: Ejemplo de fichero descriptivo

Como se puede observar en el código 3.1, el fichero descriptivo es un archivo en formato JSON, donde cada clave representa una propiedad del paquete de telemetría del vehículo. Es importante destacar que el paquete de telemetría recibido por el backend debe contener, como mínimo, las claves definidas en dicho JSON, tales como `boat_latitude`, `boat_speed`, entre otras.

Cada clave del JSON contiene un valor asociado que, a su vez, define varios campos con la siguiente estructura:

- **Type:** Define el tipo de dato SQL que tendrá la columna correspondiente en la tabla del vehículo.
- **Visible:** Indica si la propiedad se visualizará en el bocadillo informativo que aparece sobre el paquete de telemetría del vehículo durante la reproducción de una misión.
- **Field:** Este campo señala si la propiedad es esencial para la ejecución de una misión. Los valores posibles para este campo son:

- `latitude`: Latitud del vehículo.
- `longitude`: Longitud del vehículo.
- `time`: Tiempo del registro.
- `heading`: Orientación del vehículo.
- `wind_speed`: Velocidad del viento.
- `wind_angle`: Ángulo del viento.
- `wp_latitude`: Latitud del waypoint activo.
- `wp_longitude`: Longitud del waypoint activo.

Los campos `latitude`, `longitude` y `time` son obligatorios; el sistema no permite introducir un fichero descriptivo que no los incluya. El resto de los campos son opcionales, aunque recomendables, ya que mejoran la visualización y comprensión de la misión durante su reproducción.

3.2.2.2. Creación de un vehículo

Al crear un vehículo, se lee el fichero descriptivo explicado en la sección 3.2.2.1. Los campos definidos en dicho fichero se utilizan para generar las columnas específicas de la tabla correspondiente a ese vehículo. El nombre de la tabla será `records_vehicle_`*`id`*, donde el id será el identificador que se le ha dado al vehículo en la tabla `vehicles`. Cada vehículo cuenta con un conjunto de propiedades fijas, como `id`, `mission_id`, `reset_number`, `channel`, `data_source` y `raw_data`, así como con propiedades variables, que se definen dinámicamente a partir del contenido del fichero descriptivo.

La ilustración 3.2 representa la tabla de un vehículo creado con el fichero descriptivo 3.1, de manera que, la tabla de la base de datos de este vehículo tendrá los campos fijos obligatorios y los campos variables que se definen en el fichero descriptivo.

3.2.2.3. Modificación en el fichero descriptivo

Al modificar el fichero descriptivo, existen dos opciones: cambiar únicamente el campo `Visible` de alguno de los atributos, o subir un nuevo fichero descriptivo a la base de datos. Comenzaremos explicando la primera opción.

Como se muestra en la ilustración 3.3, al pulsar el botón junto al texto `Visible`, se habilita la opción de activar o desactivar la visibilidad de los campos deseados. Cabe destacar que los campos marcados como `field` no pueden ser deshabilitados, ya que son esenciales para la correcta visualización de las misiones. Una vez aplicados los cambios, se envía una petición al backend para actualizar el fichero descriptivo.

atirma_db **records_vehicle_12**

id : int(11)
mission_id : bigint(20)
reset_number : smallint(6)
channel : varchar(50)
data_source : enum('live','log')
raw_data : longtext
boat_latitude : double
boat_longitude : double
boat_speed : double
pitch : double
roll : double
heading : double
bearing : double
rudder_angle_deg : double
sail_angle_deg : double
datetime : timestamp
appWindSpeedKnots : double
appWindAngleDeg : double
activeNode_latitude : double
activeNode_longitude : double

Ilustración 3.2: Tabla de un vehículo creado con el fichero descriptivo 3.1

Descriptive DOWNLOAD UPLOAD

Name	Type	Visible	Field
boat_latitude	double	☑	latitude
boat_longitude	double	☑	longitude
boat_speed	double	☐	
pitch	double	☑	
roll	double	☑	
heading	double	☑	heading
bearing	double	☑	
starboard_throttle_per	double	☑	
port_throttle_per	double	☑	
datetime	timestamp	☑	time
appWindSpeedKnots	double	☑	wind_speed
appWindAngleDeg	double	☑	wind_angle
activeNode_latitude	double	☑	wp_latitude
activeNode_longitude	double	☑	wp_longitude

Ilustración 3.3: Apartado en la aplicación para cambiar la visibilidad de los campos del fichero descriptivo

La segunda opción, en cambio, implica una modificación estructural de la tabla asociada al vehículo, ya que se carga un nuevo fichero descriptivo. Este proceso, más complejo, requiere los siguientes pasos para asegurar la integridad de los datos:

1. **Creación de una tabla temporal:** Se genera una nueva tabla temporal que contiene las columnas definidas en el nuevo fichero descriptivo.
2. **Copia de registros comunes:** Para preservar los datos históricos de las misiones anteriores, se copian todos los registros correspondientes a las columnas que coinciden entre la tabla antigua y la temporal.
3. **Rellenado de nuevas columnas desde `raw_data`:** El campo `raw_data` almacena el contenido original del paquete de telemetría, que puede tener más campos de los especificados explícitamente en el fichero descriptivo del vehículo. Por ello, se utilizan estos datos para completar las columnas nuevas (o modificadas) que no estaban presentes en la estructura anterior. Se extrae el valor correspondiente de `raw_data` y se inserta en la columna adecuada, garantizando así la conservación de toda la información original.
4. **Eliminación de la tabla antigua:** Una vez que todos los registros se han migrado correctamente a la nueva estructura, se elimina la tabla anterior.
5. **Renombrado de la tabla temporal:** Finalmente, se renombra la tabla temporal con el nombre estándar de las tablas de registros por vehículo, es decir, `records_vehicle_`*`id`*, donde *id* representa el identificador del vehículo registrado en la tabla `vehicles`.

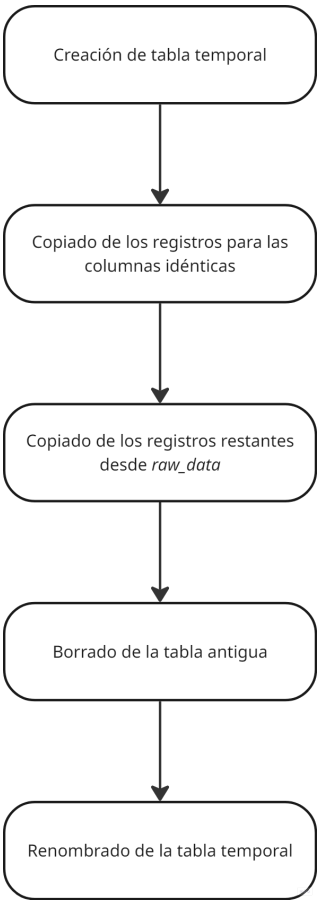


Ilustración 3.4: Gráfico demostrativo del algoritmo para modificar el fichero descriptivo

3.2.3. Estructura final

En esta sección se describe la estructura final de la base de datos del proyecto. El sistema ha pasado de tener 12 tablas a contar con 10 tablas fijas, además de las tablas específicas para cada vehículo, que se crean de manera dinámica cuando se añade un nuevo tipo de vehículo al sistema. Los cambios más relevantes se han realizado en las siguientes tablas:

- **Vehicles:** Se han añadido las siguientes columnas:
  - `descriptive`: de tipo `LONGTEXT`, almacena el fichero descriptivo en formato JSON.
  - `icon`: de tipo `VARCHAR(200)`, permite seleccionar el icono representativo del vehículo. Dado que existen distintos tipos de vehículos, se ha añadido esta funcionalidad para facilitar su diferenciación.
  - `route_color`: de tipo `TEXT`, almacena el color del vehículo en formato hexadecimal, permitiendo su representación visual personalizada en el mapa.
- **Waypoints:** Se ha añadido la siguiente columna:

- `vehicle_id`: dado que ahora es posible tener múltiples vehículos en una misma misión, cada uno puede tener sus propios *waypoints*. Por ello, se ha incluido esta columna para identificar a qué vehículo pertenece cada *waypoint*.

- **Tablas variables:** Estas son tablas específicas por vehículo y se explican en la sección 3.2.2.

3.2.3.1. Tablas

A continuación, se procederá a detallar todas las tablas que componen la base de datos del sistema.

- **Tabla `missions`** La tabla `missions` (Ilustración 3.5) almacena información sobre las misiones creadas por el usuario. Sus campos principales son:
  - **id**: Identificador único de cada misión.
  - **owner**: Identificador del propietario de la misión.
  - **name**: Nombre de la misión.
  - **description**: Descripción de la misión.
  - **location**: Ubicación de la misión.
  - **start_date**: Fecha y hora de inicio de la misión.
  - **end_date**: Fecha y hora de finalización de la misión.
  - **is_public**: Indicador de si la misión es pública o no.
  - **active_mission**: Indicador de si la misión está activa o no.

id	owner	name	description	location	start_date	end_date	is_public	active_mission
4	14	ATirma	Prueba web	LP	2024-05-31 17:57:58	2024-08-13 19:00:10	0	0
5	14	Prueba-CSV	Probando Web	LP	2021-09-24 07:38:33	2021-09-24 13:04:15	0	0

Ilustración 3.5: Estructura de la tabla *missions*

- **Tabla `mission_permissions`**

  La tabla `mission_permissions` (Figura 3.6) establece las relaciones entre las misiones y los permisos asignados a los usuarios para dichas misiones. Sus campos principales son:
  - **id**: Identificador único de cada relación.
  - **mission_id**: Identificador de la misión asociada.
  - **user_id**: Identificador del usuario asociado.
  - **permission**: Permiso asignado al usuario para la misión.

id	mission_id	user_id	permission
20	24	17	Full access

Ilustración 3.6: Estructura de la tabla *mission_permission*

- **Tabla `mission_vehicle`**

  La tabla `mission_vehicle` (Figura 3.7) establece las relaciones entre las misiones y los vehículos asignados a dichas misiones. Sus campos principales son:

  - **id**: Identificador único de cada relación.
  - **mission_id**: Identificador de la misión asociada.
  - **vehicle_id**: Identificador del vehículo asociado.

Nombre	Tipo
id	bigint(20)
mission_id	bigint(20)
vehicle_id	bigint(20)

Ilustración 3.7: Estructura de la tabla *missions_vehicle*

- **Tabla `permissions`**

  La tabla `permissions` (Figura 3.8) almacena los permisos asociados a diferentes funciones en el sistema. Sus campos principales son:

  - **id**: Identificador único de cada permiso.
  - **name**: Nombre descriptivo del permiso.
  - **type**: Tipo de permiso, que categoriza la función asociada al permiso.

id	name	type
1	show_missions	missions
2	show_vehicles	vehicles
3	show_users	users
4	show_admin_panel	-
5	show_mission	missions

Ilustración 3.8: Estructura de la tabla *permissions*

- **Tabla `roles`**

  La tabla `roles` (Figura 3.9) almacena los diferentes roles que pueden tener los usuarios en el sistema. Sus campos principales son:

- **id**: Identificador único de cada rol.
- **name**: Nombre del rol, usado internamente en el sistema.
- **display_name**: Nombre del rol mostrado a los usuarios.

id	name	display_name
1	admin	Administrador
2	user	Usuario

Ilustración 3.9: Estructura de la tabla *roles*

- **Tabla `permission_role`**

  La tabla `permission_role` (Figura 3.10) almacena la relación entre los permisos que puede tener cada rol en el sistema. Sus campos principales son:

  - **id**: Identificador único de cada permission_role.
  - **permission_id**: Identificador del permission.
  - **role_id**: Identificador del role.

1	id	bigint(20)
2	permission_id	bigint(20)
3	role_id	bigint(20)

Ilustración 3.10: Estructura de la tabla *permission_role*

- **Tabla `users`**

  La tabla `users` (Figura 3.11) almacena la información esencial de los usuarios del sistema. Sus campos principales son:

  - **id**: Identificador único de cada usuario.
  - **name**: Nombre del usuario, con índice para búsquedas rápidas.
  - **email**: Dirección de correo electrónico del usuario, también indexada.
  - **password**: Contraseña encriptada del usuario.
  - **role_id**: Identificador del rol asociado al usuario, indexado para consultas eficientes.

Nombre	Tipo
id	bigint(20)
name	varchar(200)
email	varchar(200)
password	varchar(200)
role_id	bigint(20)

Ilustración 3.11: Estructura de la tabla *users*

- **Tabla `vehicles`**

  La tabla `vehicles` almacena información sobre los vehículos utilizados en el sistema. Sus campos principales son:

  - **id**: Identificador único de cada vehículo.
  - **name**: Nombre del vehículo.
  - **boat_mark_color**: Color del marcador del barco.
  - **boat_waypoint_color**: Color de los *waypoints* del barco.
  - **route_color**: Color de la ruta del barco.
  - **mac**: Dirección MAC del vehículo.
  - **descriptive**: Fichero descriptivo en formato JSON almacenado como cadena de texto.
  - **icon**: Icono que tendrá el vehículo a la hora de ser representado en el mapa.

id	name	boat_mark_color	boat_waypoint_color	route_color	mac	descriptive ▲ 1	icon
12	Test	#002885	#002885	#002885	1:5:8:9	{"boat_latitude":{"type":"double","visible":1,"fie...	Sailboat
11	Moira	#002885	#002885	#002885	0:0:0:0	{"boat_latitude":{"type":"double","visible":1,"fie...	Sailboat
9	A-Tirma Sim (rocsim)	#004cff	#000000	#ff0000	0:0:0:2	{"boat_latitude":{"type":"double","visible":1,"fie...	Motorboat
10	A-Tirma G3	#002885	#002885	#002885	0:0:0:10	{"boat_latitude":{"type":"double","visible":1,"fie...	Sailboat

Ilustración 3.12: Estructura de la tabla *vehicles*

- **Tabla `records_vehicle_id`**

  La tabla `records_vehicle_id` representa la estructura que tendrán las tablas de registro individuales para cada vehículo. Al sustituir *id* por el identificador correspondiente de un vehículo (según la tabla `vehicles`), se obtiene el nombre concreto de la tabla. Cada una de estas tablas almacena todos los registros asociados al vehículo en cualquiera de las misiones en las que ha participado. Incluyen tanto campos variables, definidos por el fichero descriptivo, como campos fijos, que son los siguientes:

- **id**: Identificador único del registro.
- **mission_id**: Identificador de la misión a la que pertenece el registro.
- **reset_number**: Número de reinicio del sistema.
- **channel**: Canal de comunicación utilizado para recopilar datos.
- **data_source**: Fuente de procedencia del registro.
- **raw_data**: Paquete de telemetría entero en formato JSON como cadena de texto.

En la ilustración 3.13, los campos variables (definidos en el fichero descriptivo) están representados con un cuadrado verde, mientras que los campos fijos aparecen en un cuadrado rojo.

Nombre	Tipo
id	int(11)
mission_id	bigint(20)
reset_number	smallint(6)
channel	varchar(50)
data_source	enum('live', 'log')
raw_data	longtext
boat_latitude	double
boat_longitude	double
boat_speed	double
pitch	double
roll	double
heading	double
bearing	double
rudder_angle_deg	double
sail_angle_deg	double
datetime	timestamp
appWindSpeedKnots	double
appWindAngleDeg	double
activeNode_latitude	double
activeNode_longitude	double

Ilustración 3.13: Estructura de la tabla *records_vehicle_id*

- **Tabla `Waypoints`**

  La tabla *waypoints* (Figura 3.14) almacena información sobre los puntos de ruta definidos en las misiones. Sus campos principales son:

- **id**: Identificador único de cada paquete de puntos de ruta.
- **mission_id**: Identificador de la misión asociada al paquete de puntos de ruta.
- **vehicle_id**: Identificador del vehículo asociada al paquete de puntos de ruta.
- **nodes**: Lista de nodos que componen el paquete de puntos de ruta.
- **latitudes**: Lista de latitudes de los nodos del paquete de puntos de ruta.
- **longitudes**: Lista de longitudes de los nodos del paquete de puntos de ruta.
- **is cyclic**: Indicador de si el paquete de puntos de ruta es cíclico o no.
- **timestamp**: Marca de tiempo de cuando se creó el paquete de puntos de ruta.

id ▾ 1	mission_id	vehicle_id	nodes	latitudes	longitudes	is_cyclic	timestamp
1254	25	11	[0,1,2,3,4,5,2,6,7]	[28.1518,28.1434,28.1502,28.153999,28.1602,28.1558...	[-15.7053,-15.7173,-15.7232,-15.7174,-15.7229,-15....	0	2025-07-08 16:26:31
1253	25	11	[0,1,2,3,4,5,2,6,7]	[28.1518,28.1434,28.1502,28.153999,28.1602,28.1558...	[-15.705301,-15.717301,-15.7232,-15.7174,-15.72290...	1	2025-07-08 16:23:41
1252	25	11	[0,1,2,3,4,5,2,6,7]	[28.1518,28.1434,28.1502,28.153999,28.1602,28.1558...	[-15.705301,-15.717301,-15.7232,-15.7174,-15.72290...	1	2025-07-08 08:30:04

Ilustración 3.14: Estructura de la tabla *waypoints*

- **Tabla `general_data`**

  La tabla *general_data* (Figura 3.15) almacena información básica o de interés sobre la aplicación web. Sus campos principales son:

  - **id**: Identificador único.
  - **dataKey**: Clave para identificar el valor a guardar.
  - **value**: Cadena con el valor de la clave.

id	dataKey	value
1	aboutText	Copyright (C) 2024. **Universidad de Las Palmas de...
2	aboutTitle	**This is A-Tirma WebWare v2 web site.**

Ilustración 3.15: Estructura de la tabla *general_data*

3.2.3.2. Relaciones de tablas

Este esquema de base de datos está diseñado para gestionar y monitorear misiones de vehículos, incluyendo la asignación de permisos a usuarios, la gestión de roles y permisos, y el seguimiento detallado de los datos de navegación. La flexibilidad en las relaciones permite un manejo eficiente y seguro de la información. En la 3.16 se muestra la estructura de la base de datos, y a continuación se explicará cada una de las relaciones existentes de esta estructura.

1. **Usuarios y roles**

- **Tablas involucradas**: `users`, `roles`, `permission_role`
- **Descripción**: Esta relación define el rol que tiene cada usuario en el sistema. Los roles están definidos en la tabla `roles`, y cada usuario tiene asignado un rol específico. Mostrado en la ilustración 3.16 con una línea azul oscura.

2. **Roles y permisos**
   - **Tablas involucradas**: `roles`, `permissions`, `permission_role`
   - **Descripción**: Los permisos definen las acciones que pueden realizar los roles. La tabla `permission_role` conecta roles con permisos, permitiendo una configuración flexible de los derechos de acceso. Definido en la ilustración 3.16 con una línea naranja.
3. **Misiones y vehículos**
   - **Tablas involucradas**: `missions`, `vehicles`, `mission_vehicle`
   - **Descripción**: Esta relación permite asignar varios vehículos a una misión específica, lo que es útil para coordinar operaciones en las que se utilizan múltiples vehículos. Definido en la ilustración 3.16 con una línea verde.
4. **Misiones y usuarios**
   - **Tablas involucradas**: `missions`, `users`, `missions_permissions`
   - **Descripción**: Esta relación permite asignar permisos específicos a los usuarios sobre misiones particulares. Por ejemplo, un usuario puede tener permisos de solo vista o acceso completo a una misión. Definido en la ilustración 3.16 con una línea amarilla.
5. **Misiones y puntos de paso**
   - **Tablas involucradas**: `missions`, `waypoints`
   - **Descripción**: Los puntos de paso (`waypoints`) definen el camino o trayectoria que deben seguir los vehículos durante una misión. Cada misión puede tener un conjunto de puntos de ruta asociados. Mostrado en la ilustración 3.16 con una línea azul clara.
6. **Registros y misiones**
   - **Tablas involucradas**: `missions`, `records_vehicle_id`
   - **Descripción**: Los registros deben pertenecer a una misión. De esta manera, se crea la relación de `missions` y `records_vehicle_id`. Mostrado en la ilustración 3.16 con una línea roja.

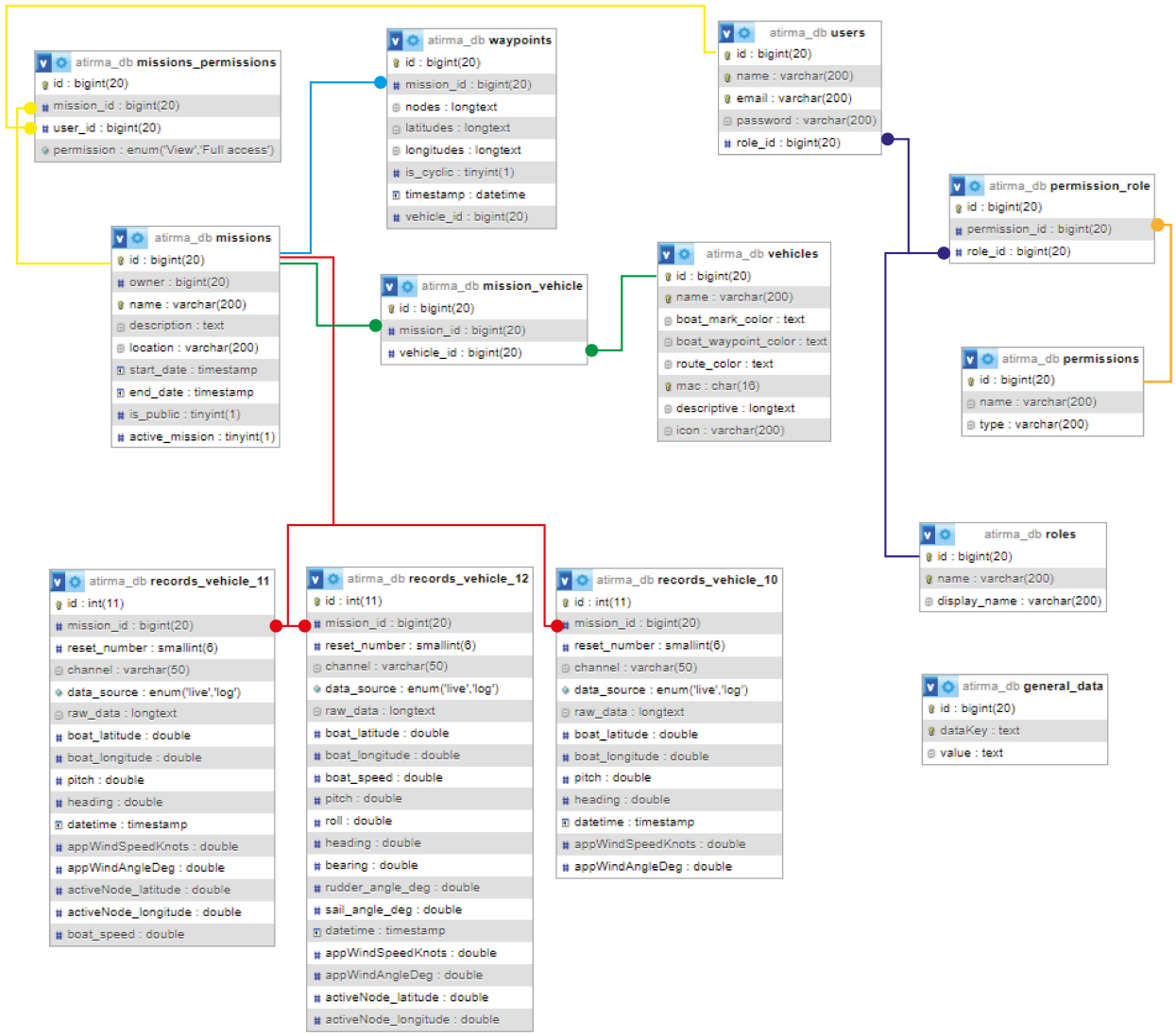


Ilustración 3.16: Nueva estructura de la base de datos

3.3. Exportación e importación de misiones

La importación y exportación de misiones se ha vuelto un componente fundamental del sistema, especialmente ante los cambios de versión y arquitectura de la base de datos. Para evitar la pérdida de misiones históricas, se ha diseñado un mecanismo que permite exportar e importar misiones de forma segura, garantizando la conservación completa de su información.

3.3.1. Importación de las misiones

Actualmente, la importación de misiones se realiza de forma distinta, ya que los registros deben insertarse directamente en la tabla de registros correspondiente a cada vehículo en la base de datos. Para ello, es necesario verificar que la misión a importar contenga todos los campos definidos en el fichero descriptivo del vehículo, lo cual garantiza la coherencia e integridad de los datos.

El proceso de importación se lleva a cabo de forma individual para cada vehículo, permitiendo además diferenciar entre la importación de registros y la importación de puntos de paso (*waypoints*).

3.3.1.1. Importación de registros

El proceso de importación de registros en la versión actual del sistema sigue una lógica similar a la utilizada en la versión 2. En dicha versión, el procedimiento consistía en subir un fichero CSV desde el front-end. Este archivo era parseado directamente en el navegador y, posteriormente, se enviaba al back-end una petición HTTP con los registros a insertar en formato JSON.

Este enfoque presentaba varios inconvenientes. En particular, cuando se intentaba importar un fichero de gran tamaño (por ejemplo, superior a 100 MB), el front-end tendía a bloquearse debido al esfuerzo de procesamiento requerido. Además, el envío de un cuerpo de petición (body) tan grande en formato JSON resultaba poco práctico e ineficiente.

Para solventar estos problemas, se implementaron las siguientes mejoras:

- **Subida del fichero CSV al back-end**: En lugar de parsear el CSV en el front-end, ahora se envía directamente el fichero al servidor utilizando la librería `multer`. Este enfoque ofrece dos ventajas principales: por un lado, permite almacenar el fichero en el servidor para su posible reutilización; por otro, evita la sobrecarga y el bloqueo del front-end.
- **Parseo en el back-end**: Una vez recibido el fichero, el procesamiento y parseo del CSV se realiza en el servidor. Esto permite aprovechar los mayores recursos del back-end, mejorando el rendimiento general del proceso.
- **Validación del fichero**: Se lleva a cabo una validación del fichero CSV para comprobar que contiene todos los campos declarados en el fichero descriptivo del vehículo. Esto

permite garantizar la integridad de los datos e impedir la inserción de columnas con valores `NULL` en la base de datos.

- **Distinción entre `LOG` y `LIVE`**: Las misiones se clasifican en dos tipos: misiones históricas (`LOG`) y misiones en tiempo real (`LIVE`). Esta distinción permite importar los registros de una misión como históricos o como datos en vivo, según corresponda. De este modo, a la hora de reproducir las misiones también será posible seleccionar el tipo de registros que se desea visualizar.
- **Validación del formulario**: Se ha implementado una validación en el front-end utilizando Vue para asegurar que se ha adjuntado un archivo y que se ha seleccionado el tipo de misión (`LOG` o `LIVE`) antes de enviar la petición al back-end para realizar la importación.

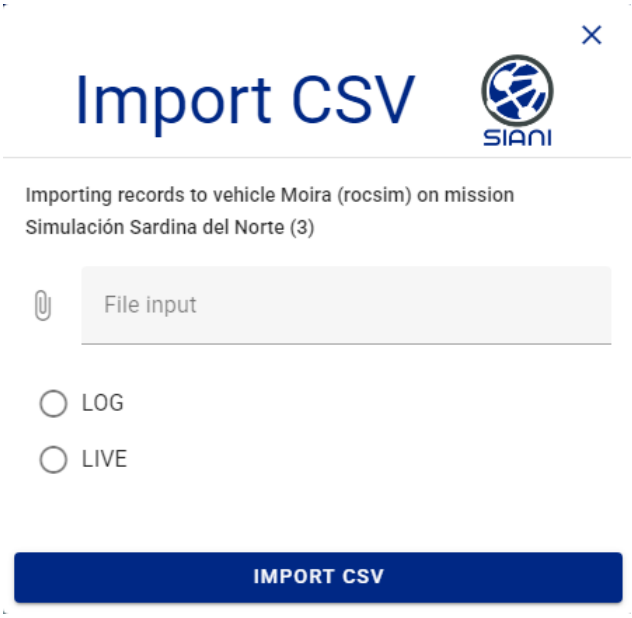


Ilustración 3.17: Diálogo para importar CSV

3.3.1.2. Importación de waypoints

Los waypoints son los puntos de ruta que debe seguir el vehículo durante una misión. El formato en el que se exportan desde el sistema se muestra en el Listado 3.2. Su estructura es la siguiente:

- **id**: Identificador del paquete de waypoints.
- **mission_id**: Identificador de la misión asociada.
- **vehicle_id**: Identificador del vehículo.
- **nodes**: Orden de paso que debe seguir el vehículo a lo largo de la ruta. El array `nodes` contiene los índices de los waypoints, lo que determina la secuencia del recorrido.

- **latitudes**: Array con las latitudes correspondientes a cada waypoint.
- **longitudes**: Array con las longitudes correspondientes a cada waypoint.
- **is_cyclic**: Indica si el recorrido es cíclico. En caso afirmativo, el vehículo volverá al primer nodo al finalizar la ruta; en caso contrario, recorrerá el camino en sentido inverso hasta el punto inicial.
- **timestamp**: Marca temporal que indica la fecha y hora en que el paquete de waypoints fue recibido en el backend.

Cabe destacar que existe una distinción especial con respecto al primer waypoint: este se considera como el punto de retorno o return to home, que el vehículo utilizará en caso de ser necesario regresar al inicio.

```
{
  "id": 95,
  "mission_id": 12,
  "vehicle_id": 15,
  "nodes": [
    0,
    1,
    2,
    3
  ],
  "latitudes": [
    28.1518,
    28.1577,
    28.152201,
    28.131001
  ],
  "longitudes": [
    -15.705301,
    -15.768,
    -15.7518,
    -15.7729
  ],
  "is_cyclic": 1,
  "timestamp": "2025-07-07T18:32:22.000Z"
}
```

Código 3.2: Ejemplo de una lista de waypoints

En la ilustración 3.18 se puede observar un conjunto de 8 waypoints. El waypoint 0, representado en color azul y ubicado más a la derecha, corresponde al punto de retorno (return home). Los demás waypoints forman un recorrido indicado por flechas, las cuales representan la dirección y el orden definidos en el array `nodes`.

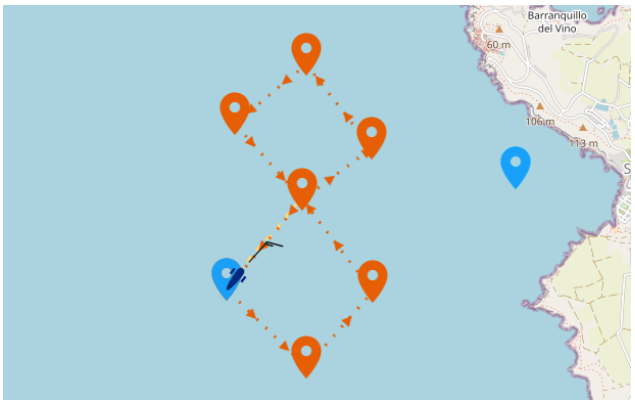


Ilustración 3.18: Paquete de waypoints en una misión en directo

Anteriormente, estos *waypoints* solo podían añadirse a la misión mediante peticiones HTTP en el contexto de misiones en vivo. Gracias a la funcionalidad de importación de *waypoints*, ahora es posible incorporarlos directamente a través del botón mostrado en la Figura 3.19.

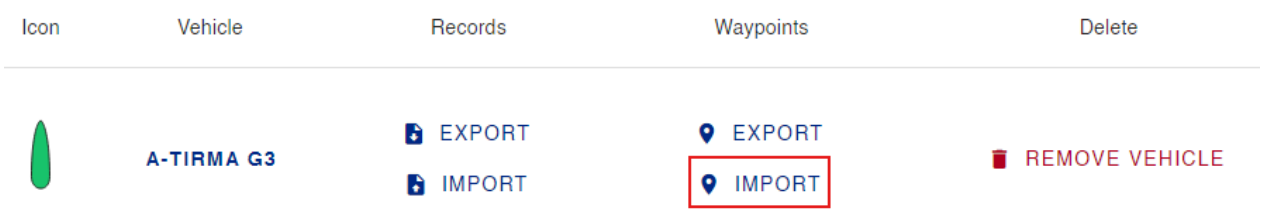


Ilustración 3.19: Apartado en los ajustes de la misión para importar registros o waypoints

3.3.2. Exportación de las misiones

La exportación de misiones en vivo ha sido una de las tareas más relevantes del Trabajo de Fin de Título, ya que la División de Robótica y Oceanografía Computacional (ROC) del Instituto Universitario SIANI (SIANI) requería preservar las misiones antiguas. Para ello, esta funcionalidad se ha dividido en dos apartados: la exportación de registros y la exportación de waypoints.

3.3.2.1. Exportación de registros

La exportación de registros de un vehículo en una misión se realiza mediante el procesamiento por lotes en la base de datos. Se recuperan los campos raw_data (información del registro recibida a través de peticiones HTTP durante una misión en vivo) de todos los registros del vehículo asociados a la misión correspondiente. A partir de estos datos, se genera un fichero CSV cuyo enlace de descarga es enviado al front-end, permitiendo al usuario descargar

la misión. De este modo, el fichero CSV puede ser descargado o eliminado posteriormente desde la pantalla de gestión de CSV, descrita en la Sección 3.7.

Además de la exportación de registros, se implementaron dos funcionalidades adicionales durante el proceso de exportación:

1. **Distinción entre `LOG` y `LIVE`**: La posibilidad de diferenciar entre los registros de misiones en vivo y misiones históricas proporciona una mayor flexibilidad a la hora de exportar los datos.
2. **Renombrado del fichero CSV**: Se permite al usuario seleccionar el nombre del fichero antes de exportar los registros de la misión. Esta funcionalidad facilita la organización de los archivos exportados. En caso de que el nombre introducido coincida con el de un fichero ya existente en el backend, dicho fichero será sobrescrito.

Ilustración 3.20: Diálogo para exportar los registros de una misión

3.3.2.2. Exportación de waypoints

Para exportar los waypoints o puntos de ruta, se realiza una petición al backend que puede devolver dos resultados:

- **Si existen waypoints para ese vehículo en la misión**: Se devuelve un array con todos los waypoints en el formato mostrado en el JSON de la Figura 3.2. El usuario podrá descargar dicho archivo con los puntos de ruta de la misión.
- **Si no existen waypoints**: Se devuelve un array vacío, y el frontend mostrará un mensaje al usuario mediante un *snackbar* de Vuetify.

En la ilustración 3.21 se muestra el botón desde el cual es posible exportar los waypoints.

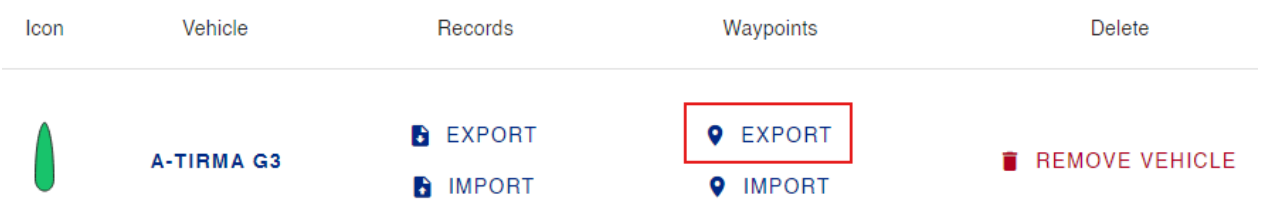


Ilustración 3.21: Apartado en los ajustes de la misión para exportar registros o waypoints

3.4. Misiones en vivo

Las misiones en vivo presentan una complejidad superior en comparación con las misiones históricas, ya que requieren una recuperación constante de datos provenientes de los vehículos. Esta funcionalidad se ha vuelto aún más compleja debido a la incorporación de múltiples vehículos por misión.

3.4.1. Suscripción a la misión

Cuando un usuario reproduce una misión en vivo, se suscribe mediante una petición al backend, donde se guarda un identificador asociado a dicha suscripción.

En la versión anterior del sistema, un usuario solo podía visualizar una única misión en directo. Esta limitación ha sido superada, permitiendo ahora la visualización simultánea de múltiples misiones en vivo. Para lograrlo, además del identificador del usuario, se ha incorporado un identificador de sesión. Es decir, el nuevo identificador de suscripción es la concatenación del nombre de usuario con un contador de sesión mantenido en el backend. Gracias a este mecanismo, un mismo usuario puede acceder a misiones en vivo desde diferentes sesiones, navegadores o pestañas.

Durante la ejecución de una misión en vivo, el frontend se suscribe al backend para recibir los registros que este va recopilando en tiempo real desde los vehículos.

El backend, al recibir registros en directo, verifica si hay usuarios con sesiones activas (`user_session`) registrados. Si es así, guarda los registros recibidos en un diccionario de JavaScript, donde la clave es el identificador `user_session` y el valor es el conjunto de registros acumulados. A medida que llegan nuevos datos, estos se van agregando al diccionario, quedando listos para ser enviados al frontend.

3.4.2. Actualización del frontend

Para mantener actualizada la misión en tiempo real, el frontend realiza peticiones al backend cada 5 segundos, solicitando los registros que hayan llegado desde la última actualización.

En cada petición, el backend consulta el diccionario asociado al identificador de sesión. Según el contenido disponible, puede devolver registros de vehículos, waypoints, puntos de paso o, en su defecto, una respuesta vacía si no hay nueva información. Tras responder a la petición, el backend limpia el valor correspondiente a esa clave en el diccionario para evitar duplicados en futuras respuestas.

Al recibir esta información, el frontend sigue la siguiente lógica:

- Si el usuario está en la última página de la misión o tiene activado el modo de visualización en directo (indicador rojo), los registros recibidos se añaden al arreglo principal de registros.
- Si no está en la paginación final, los registros recibidos se descartan para evitar inconsistencias temporales.
- Si la información recibida incluye un waypoint o un punto de paso, el sistema lo agrega directamente al arreglo correspondiente del vehículo asociado.

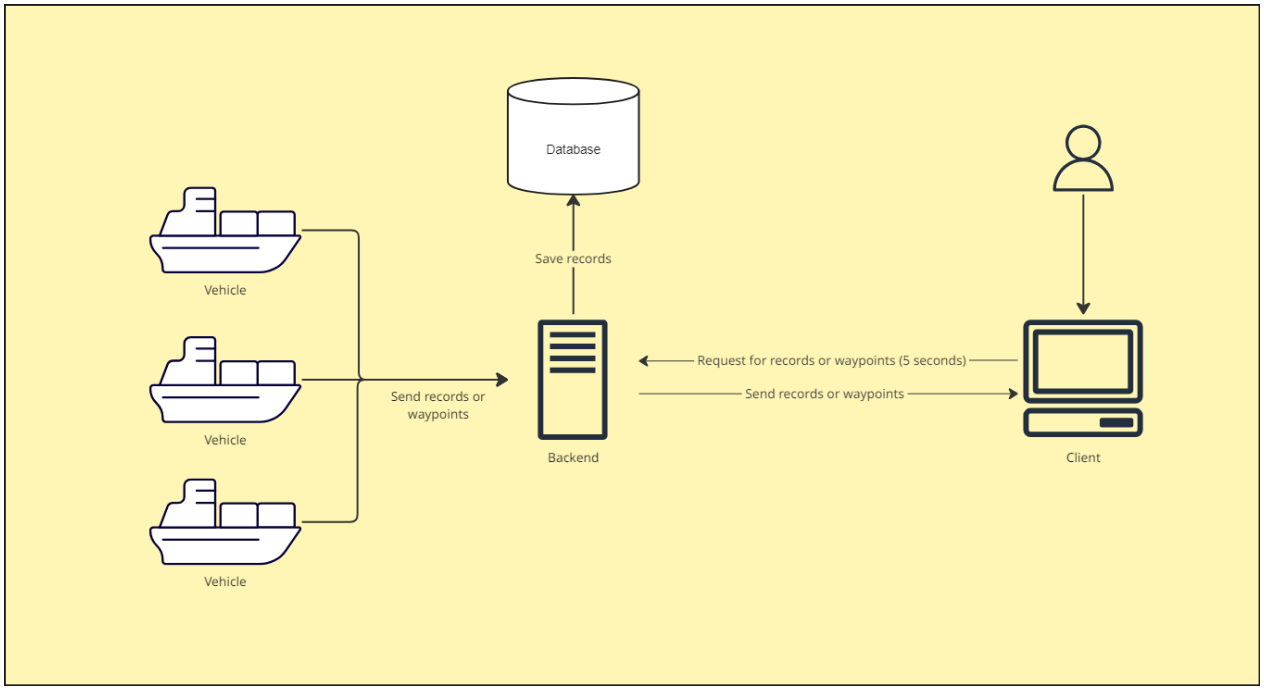


Ilustración 3.22: Diagrama del proceso de actualización en misiones en vivo

3.5. Visualización de entidades

En la versión anterior, las entidades como vehículos, misiones o usuarios se mostraban en un cuadro de diálogo donde únicamente se permitía visualizar el nombre de la entidad, añadirla o eliminarla, como se muestra en la figura 3.23.

Ilustración 3.23: Vista para gestionar los vehículos en la versión anterior

Esta implementación ha sido reemplazada por una versión más moderna e intuitiva. Gracias a las tablas de Vuetify, se ha logrado una mayor legibilidad de las entidades antes de acceder a la página de detalle correspondiente. Esta nueva visualización se muestra en la figura 3.24.

Ilustración 3.24: Vista para gestionar los vehículos en la versión actual

Además, se ha incorporado un sistema de paginación que se explicará en detalle en la sección 3.8.4.

3.6. Reproducción de misiones

En la versión anterior, solo existía un vehículo por misión, por lo que la reproducción se basaba en cargar los registros de ese único vehículo. Se realizaba una paginación de registros y estos se almacenaban en un array en el *frontend* mediante Pinia.

En la versión actual, dado que se pueden añadir varios vehículos a una misma misión, se ha modificado el planteamiento de la reproducción, ya que es posible que un vehículo no tenga registros en determinados intervalos temporales de la misión. En esta sección se explicarán los cambios realizados y el algoritmo de búsqueda de registros, que permite reproducir una misión tanto en tiempo real como de forma histórica.

3.6.0.1. Propiedades de la misión

Todas las misiones almacenan en la base de datos un *timestamp* de inicio y otro de finalización. Estos valores representan, respectivamente, el registro más antiguo y el más reciente de la misión (de cualquiera de los vehículos involucrados). En el caso de las misiones en vivo, el *timestamp* de finalización se actualiza continuamente conforme se reciben nuevos registros.

Conociendo los tiempos de inicio y finalización de la misión, se puede calcular el número máximo de intervalos que tendrá el *v-slider*. Para ello, al reproducir una misión, se selecciona un valor de *intervalo*, que define la agrupación de registros en periodos de tiempo. Por ejemplo, si el intervalo es de 60 segundos, se recuperará un único registro por minuto y por vehículo.

Este intervalo se selecciona a través del diálogo mostrado en la ilustración 3.25, donde también se puede escoger el tipo de datos o registros que se desean representar: histórica o en vivo. Para calcular el número de segmentos que tendrá el *v-slider*, utilizamos la fórmula descrita en el Algoritmo 1.

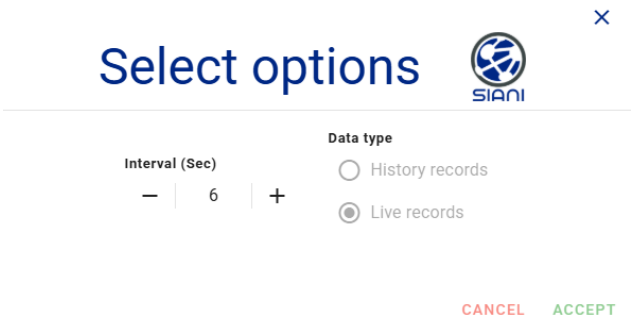


Ilustración 3.25: Diálogo que permite seleccionar el *intervalo* y el tipo de registros a recuperar. Se muestra al iniciar el proceso de reproducción de una misión.

Algoritmo 1 Cálculo del número de segmentos para el *v-slider*

1: **Entrada:**
- T_i: Tiempo inicial de la misión (timestamp)
- T_f: Tiempo final de la misión (timestamp)
- Δt: Intervalo de tiempo para agrupar los registros

2: **Salida:** Número de segmentos

3: Calcular la duración total de la misión: $D = T_f - T_i$

4: Calcular el número de segmentos: $N = \left\lfloor \frac{D}{\Delta t} \right\rfloor$

Teniendo el número de segmentos, se utiliza un cursor, al que llamaremos T, para desplazarse a lo largo de la misión. Al pulsar el botón de reproducción/pausa, mostrado en la ilustración 3.26, se activa un intervalo en JavaScript que se ejecuta con una determinada frecuencia, la cual depende de la velocidad seleccionada por el usuario para la reproducción de la misión. La función de este intervalo consiste en incrementar el valor de T de forma progresiva hasta alcanzar el final de la misión.

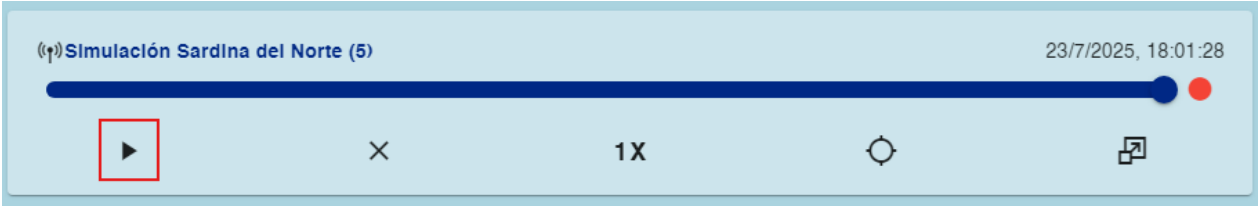


Ilustración 3.26: Botón de reproducción/pausa del menú de reproducción de una misión

3.6.0.2. Propiedades de cada vehículo

Cada vehículo perteneciente a una misión actúa de manera independiente respecto a los demás, recuperando sus registros cuando sea necesario. Lo primero que se realiza al iniciar la reproducción de una misión es una petición al *backend* para obtener los *timestamps* de los vehículos, es decir, los tiempos de inicio y fin de cada uno.

Como se muestra en la ilustración 3.27, el primer paso consiste en la recuperación de estos tiempos. Este proceso es fundamental, ya que determina el *timestamp* del primer y último registro disponibles para cada vehículo.

La respuesta a esta petición tiene el formato ilustrado en el Listado 3.3, donde se devuelve un arreglo de objetos con el `id` del vehículo, junto con los campos `start` y `end` que indican el intervalo de tiempo cubierto por sus registros.

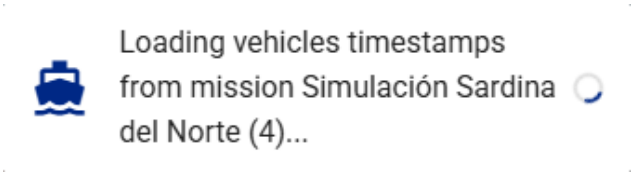


Ilustración 3.27: Recuperación de los *timestamps* de los vehículos

```
[
    {
        "vehicle_id": 15,
        "start": "2025-07-09T13:02:02.000Z",
        "end": "2025-07-13T02:55:39.000Z"
    },
    {
        "vehicle_id": 12,
        "start": "2025-07-09T13:33:30.000Z",
        "end": "2025-07-13T10:12:05.000Z"
    }
]
```

Código 3.3: Respuesta de la petición de *timestamps* de vehículos en una misión

3.6.0.3. Cálculo del registro más reciente con búsqueda binaria

Cada vehículo dispone de un cursor, al que llamaremos P, que indica la posición del registro actual a mostrar. Este cursor se actualiza en función del valor del cursor global de la misión, T, que avanza con el tiempo durante la reproducción. Para cada vehículo, se debe encontrar el registro más reciente sin que su timestamp exceda el valor de T.

Dado que los timestamps de cada vehículo están ordenados cronológicamente, se utiliza una **búsqueda binaria** (o dicotómica) para encontrar de forma eficiente el índice del registro más cercano y anterior o igual a T. Este algoritmo tiene una complejidad temporal de $\mathcal{O}(\log n)$, ya que reduce el intervalo de búsqueda a la mitad en cada iteración.

La búsqueda binaria es especialmente útil para mejorar el rendimiento cuando se trabaja con grandes cantidades de registros, ya que garantiza un número de comparaciones logarítmico en el peor de los casos. A continuación, se presenta el pseudocódigo utilizado:

Algoritmo 2 Búsqueda binaria para encontrar el índice más cercano a T

Require: *timestamps*: arreglo ordenado de tiempos
Require: T: tiempo actual de reproducción
Ensure: Índice del timestamp más cercano sin superar T

$low \leftarrow 0$
$high \leftarrow \text{longitud}(timestamps) - 1$
while $low \leq high$ **do**
    $mid \leftarrow \lfloor \frac{low+high}{2} \rfloor$
    if $timestamps[mid] = T$ **then**
        return mid
    else if $timestamps[mid] < T$ **then**
        $low \leftarrow mid + 1$
    else
        $high \leftarrow mid - 1$
    end if
end while
return $\text{máx}(0, low - 1)$

Este algoritmo devuelve el índice del último timestamp que no supera T, garantizando que se muestre siempre el registro más actualizado posible sin adelantarse al tiempo de reproducción.

3.6.0.4. Recuperación de registros del vehículo

El usuario puede mover libremente el cursor `T` del `v-slider`. Para ello, se implementa un sistema de paginación de registros que evita la carga completa de todos los datos de la misión en el frontend, lo cual sería inviable, especialmente si se trata de misiones con varios días de duración y múltiples vehículos.

Cuando el usuario avanza el cursor y el vehículo en el frontend no tiene cargada la página correspondiente (es decir, el valor de `timestamp` está fuera del rango de los registros actualmente cargados), pero aún se encuentra dentro del intervalo de tiempo definido por los `timestamp` de inicio y fin del vehículo, entonces el frontend realiza automáticamente una petición al backend para obtener los registros de dicho vehículo.

En el backend existe una variable llamada `listPerPage`, que define el número máximo de registros por página. Por tanto, cada petición para recuperar registros devuelve como máximo la cantidad especificada por `listPerPage`. La estructura de dicha petición se muestra en el código 3.4.

```
1 router.get('/getRecordsFromMission/:mision_id/:vehicle_id/:date/:start_date/:end_date/:recordsType/:
      interval', async function(req, res, next) {
2     try {
3         const t0 = performance.now();
4         const {
5             mision_id,
```

```
            vehicle_id,
            date,
            start_date,
            end_date,
            recordsType,
            interval
        } = {
            ...req.params,
            mision_id: parseInt(req.params.mision_id),
            vehicle_id: parseInt(req.params.vehicle_id),
            date: Number(req.params.date),
            start_date: Number(req.params.start_date),
            end_date: Number(req.params.end_date),
            recordsType: req.params.recordsType,
            interval: Number(req.params.interval)
        };

        res.json(await missionSearch.getRecordsFromMission(
            mision_id, vehicle_id, date, start_date, end_date, recordsType, interval
        ));

        console.log(`Recuperado registros en ${((performance.now() - t0)/1000).toFixed(2)} segundos`)
            ;
    } catch (err) {
        res.status(400).json({ message: err });
    }
});
```

Código 3.4: Petición para recuperar registros de un vehículo en un intervalo de tiempo determinado

A la petición del fragmento del código 3.4 se le adjuntan los siguientes parámetros:

- `mission_id`: Identificador de la misión.
- `vehicle_id`: Identificador del vehículo.
- `date`: Marca de tiempo para centrar la búsqueda de registros.
- `start_date`: Tiempo de inicio del vehículo.
- `end_date`: Tiempo de finalización del vehículo.
- `recordsType`: Tipo de registros a recuperar (en vivo o históricos).
- `intervalo`: *Intervalo* de tiempo definido en la misión

La función `getRecordsFromMission` calcula el rango de tiempo a consultar utilizando la variable `date`. Con los valores de `listPerPage`, `intervalo` y `date`, es posible determinar el rango de tiempo deseado: se calcula la mitad de `listPerPage` hacia atrás desde `date`, y la otra mitad hacia adelante, quedando la marca de tiempo en el centro del rango. Con este rango, se realiza la consulta de registros al backend.

Algoritmo 3 Cálculo del rango de registros a recuperar en la paginación

Require: $config.listPerPage$: número de elementos por página
Require: $interval$:
Require: $date$: tiempo actual en milisegundos
Require: $start\_date$, end_date: límites de la misión

1: $halfRange \leftarrow \left(\frac{config.listPerPage}{2} \times interval\right) \times 1000$ milisegundos
2: $minTimestamp \leftarrow \text{máx}(start\_date, date - halfRange)$
3: $maxTimestamp \leftarrow \text{mín}(end\_date, date + halfRange)$

4: **return** $[minTimestamp, maxTimestamp]$

La función `getRecordsFromMission` devuelve una respuesta con la siguiente estructura:

```
const meta = {
  pageSize: config.listPerPage,
  end: maxTimestamp.getTime() == end_date ? true : false,
  window_end: maxTimestamp.getTime(),
  start: minTimestamp.getTime() == start_date ? true : false,
  window_start: minTimestamp.getTime()
};

return {
  vehicle,
  records,
  meta
};
```

Código 3.5: Respuesta de la función getRecordsFromMission

Donde `window_start` y `window_end` son marcas de tiempo que indican el inicio y el final de la ventana temporal (o página de datos) devuelta por esta función. Esta ventana representa el rango de tiempo en el que se encuentran los registros proporcionados.

Los campos `start` y `end` son valores booleanos que indican si la ventana coincide exactamente con el inicio o el final del rango total de la misión para ese vehículo, lo cual permite saber si se ha alcanzado el límite inferior o superior de registros disponibles.

El campo `pageSize` informa del número máximo de registros contenidos en la página, definido por la variable global `listPerPage` en el sistema.

3.6.0.5. Caso: registros no encontrados en esa ventana

En el caso de que un vehículo no disponga de datos dentro de un determinado rango de tiempo, el frontend iniciará un procedimiento para recuperar el registro anterior más cercano a la marca de tiempo solicitada. Es decir, realizará peticiones iterativas en las que intentará obtener al menos un registro; en cada iteración aumentará el tamaño de la ventana de búsqueda para abarcar un mayor intervalo temporal.

El procedimiento (o algoritmo) finaliza cuando se encuentra al menos un registro correspondiente a ese vehículo. Este proceso se describe en el algoritmo 4. El *GetRecentRecords* mostrado en el algoritmo es la petición al backend solicitando los registros de esa ventana. A esta función se le pasan los siguientes parámetros:

- **mission_id:** Identificador de la misión que se está reproduciendo.
- **vehicle_id:** Identificador del vehículo del que se están buscando los registros
- **start_date:** Timestamp del primer registro del vehículo
- **timestampMin:** Timestamp o marca de tiempo desde donde se empieza a buscar los registros, en dirección hacia el start_date.
- **records_type:** Tipo de registros, históricos o en vivo.
- **interval:** *Intervalo* seleccionado para la misión.
- **factor:** Número que representa el factor de multiplicación, en el tamaño de la ventana para la búsqueda de los registros.

Algoritmo 4 Carga de registros recientes si el vehículo no tiene datos actuales

Require: $records$, $timestamp$, $start\_date$, end_date, $mission\_id$, $vehicle\_id$, $recordsType$, $interval$

if $records = \emptyset$ **then**
  $recentRecords \leftarrow \emptyset$
  $timestampMin \leftarrow \text{mín}(timestamp, end\_date)$
  $factor \leftarrow 2$
  while $timestampMin \geq start\_date$ **and** $recentRecords = \emptyset$ **do**
    $recentRecords \leftarrow$ GetRecentRecords($mission\_id$, $vehicle\_id$, $start\_date$, $timestampMin$, $recordsType$, $interval$, $factor$)
    if $recentRecords.length = \emptyset$ **then**
      if $factor \geq 10$ **then**
        $factor \leftarrow factor + 2$
      else
        $factor \leftarrow factor + 1$
      end if
    else
      $timestampMin \leftarrow recentRecords[0]$
    end if
  end while
  Insertar $recentRecords$ al inicio de $records$
end if

3.7. Gestión de los ficheros CSV

La exportación e importación de los registros de las misiones genera ficheros CSV en el servidor. Para facilitar su gestión, se ha implementado una funcionalidad que permite eliminarlos o volver a descargarlos. El acceso a la página de gestión de ficheros CSV se realiza a través del botón del menú desplegable de configuración, mostrado en la ilustración 3.28. Cabe destacar que esta página solo está disponible para los usuarios que tengan asignado el rol de administrador en el sistema.

Ilustración 3.28: Botón en el menu de configuración para entrar en la página de *CSV-FILES*

Desde esta página, cuya interfaz se muestra como ejemplo en la ilustración 3.29, es posible borrar y descargar los ficheros CSV que se hayan generado previamente. En la tabla de gestión se muestra el nombre, el tamaño y la fecha de creación de cada fichero, así como una distinción entre si han sido exportados o importados. Esta funcionalidad ha sido implementada utilizando la librería `fs` de Node.js, que permite la gestión del sistema de archivos en el backend.

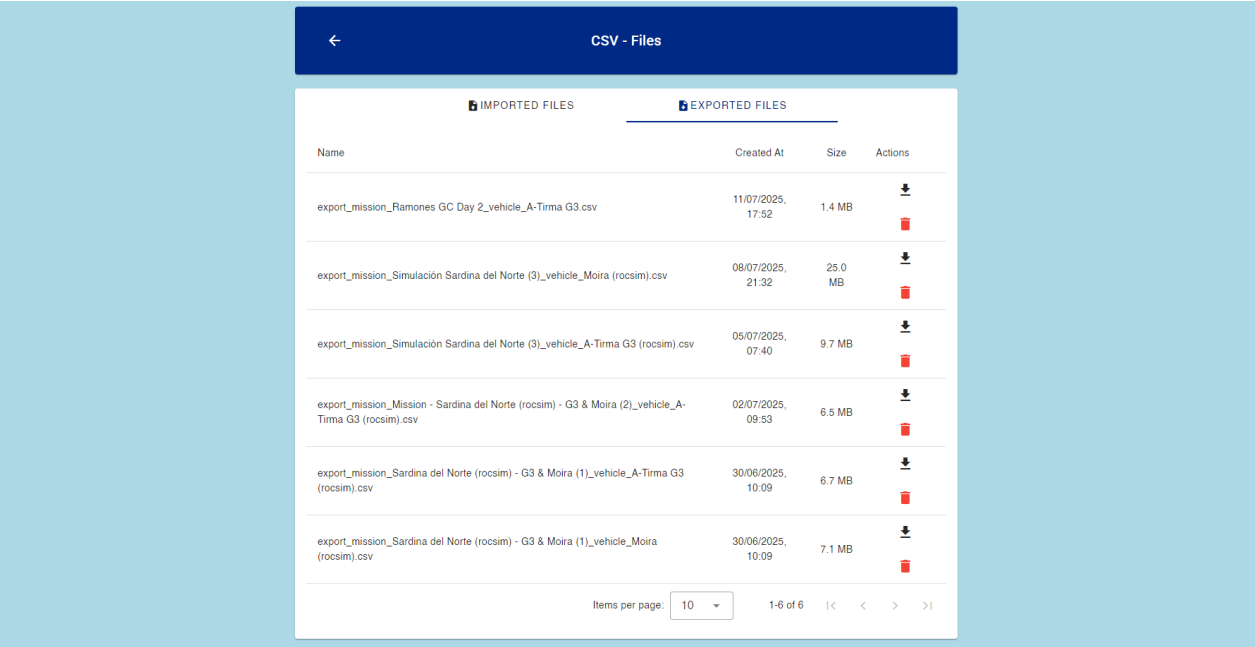


Ilustración 3.29: Página para gestionar los ficheros CSV

3.8. Mejoras

3.8.1. Iconos para los vehículos

Al introducir la capacidad de insertar diferentes vehículos con propiedades específicas —como distintos sensores— y con el objetivo de distinguir entre vehículos autónomos de vela, a motor o boyas, se ha implementado la posibilidad de asignar un icono representativo a cada vehículo.

Los iconos pueden seleccionarse tanto en el momento de creación como durante la modificación del vehículo. Actualmente, existen cuatro opciones disponibles. En la ilustración 3.30 se muestra cómo es posible seleccionar cualquiera de los iconos existentes.

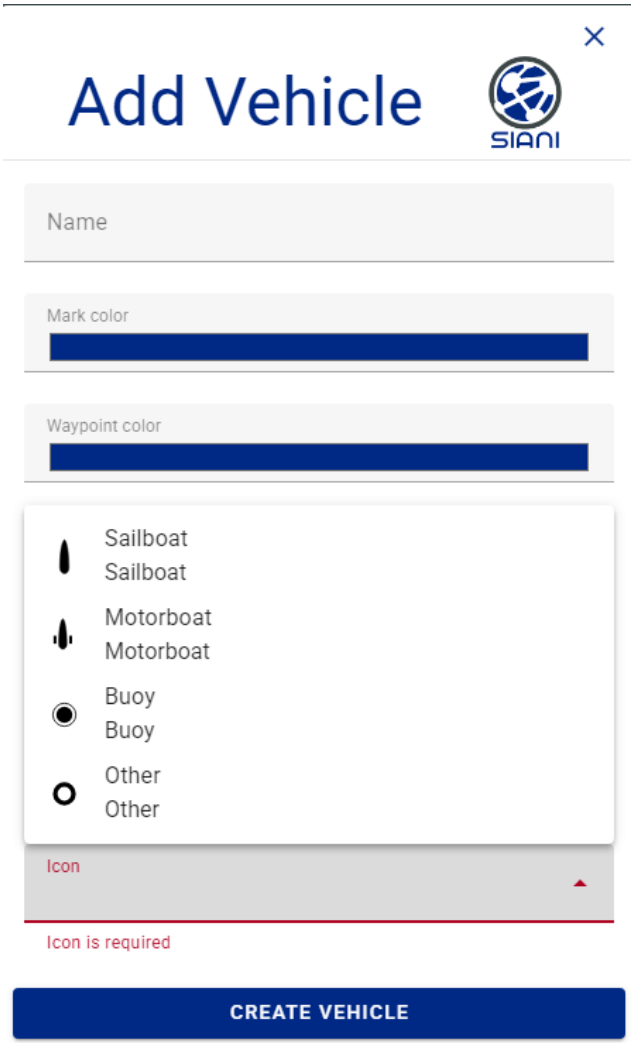


Ilustración 3.30: Formulario para crear un vehiculo nuevo

Todos los iconos han sido desarrollados utilizando estructuras básicas de `SVG` y `HTML`. Una representación visual de estos iconos puede observarse en la ilustración 3.31. Los iconos disponibles son los siguientes:

- **Buoy (Boya)**: Representa al vehículo como una boya. Se ha diseñado con dos círculos superpuestos, uno relleno y otro sin relleno.
- **Motorboat (Barco a motor)**: A partir del icono de la versión anterior, se han añadido dos rectángulos laterales que simulan los motores del vehículo.
- **Sailboat (Barco a vela)**: Este icono proviene de la versión anterior y no ha recibido modificaciones.
- **Other (Otro)**: Icono genérico para representar otros tipos de vehículos. Se ha diseñado con un círculo sin relleno.

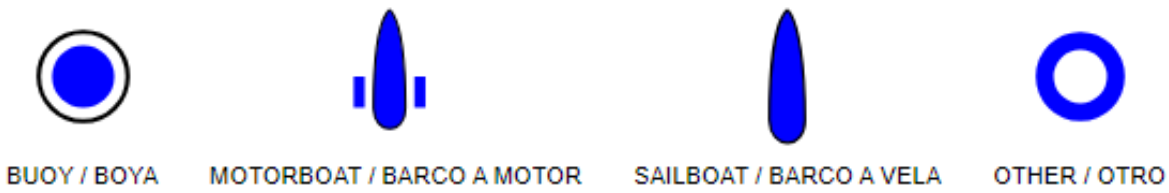


Ilustración 3.31: Iconos disponibles para los vehículos

3.8.2. Uso de vuetify

En la versión anterior del sistema se había implementado Vuetify y sus componentes, pero no se habían importado correctamente los estilos, lo que provocaba que los componentes no se visualizaran ni se comportaran como se esperaba.

Con la importación de los estilos de Vuetify, todos los componentes comenzaron a comportarse y renderizarse de forma adecuada.

Para ello, se ha añadido la línea que se muestra en el fragmento de código 3.6 dentro del archivo `main.js` del frontend desarrollado en Vue.js:

```
1 import 'vuetify/styles'
```

Código 3.6: Instrucción para importar los estilos de Vuetify en JavaScript

3.8.3. Color de la ruta del vehículo

Se ha añadido la posibilidad de personalizar el color de la ruta del vehículo. Desde el menú de edición del vehículo, el usuario puede seleccionar el color deseado para la trayectoria. Este color se mostrará al reproducir una misión, tal como se observa en la Figura 3.32.

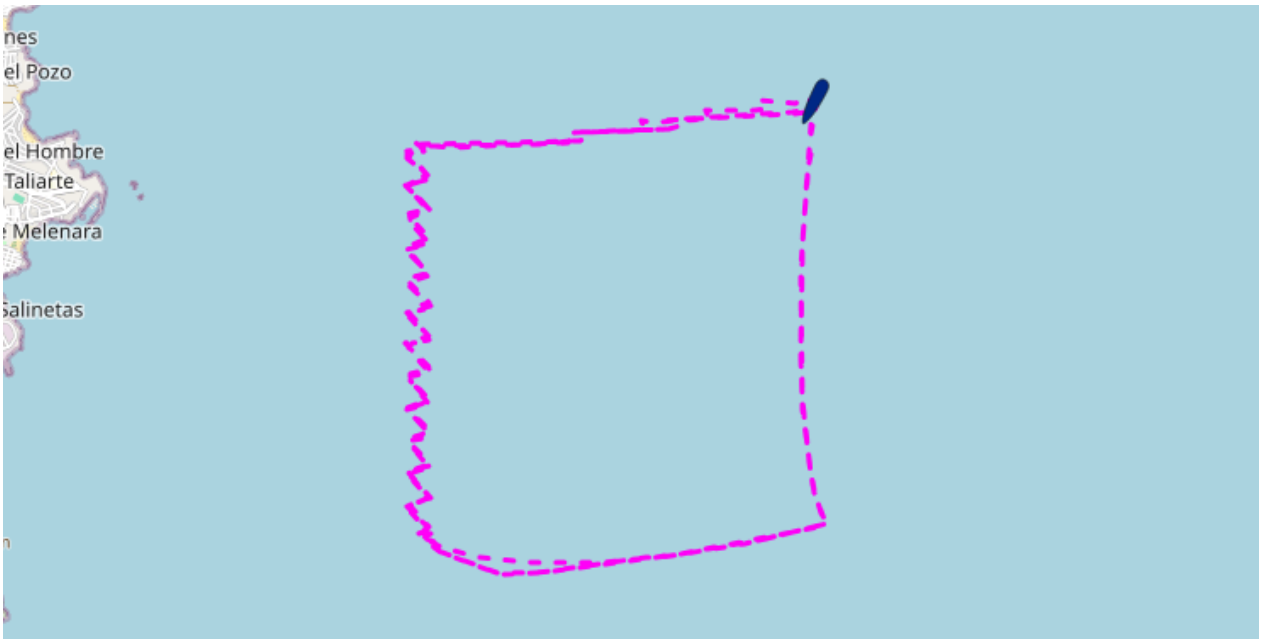


Ilustración 3.32: Representación del color de la ruta del vehículo

3.8.4. Paginación para las entidades

Se ha añadido un sistema de paginación para facilitar la visualización y gestión de las diferentes entidades del sistema. Las entidades que cuentan con esta funcionalidad son:

- **Vehicles** — Vehículos
- **Missions** — Misiones
- **Users** — Usuarios

En la ilustración 3.33 se muestra un ejemplo del diálogo proporcionado por Vuetify para visualizar todos los vehículos disponibles. En este caso, se utiliza un paginador integrado en la tabla de Vuetify que permite al usuario navegar entre las diferentes páginas de resultados, facilitando así la gestión de un gran número de registros.

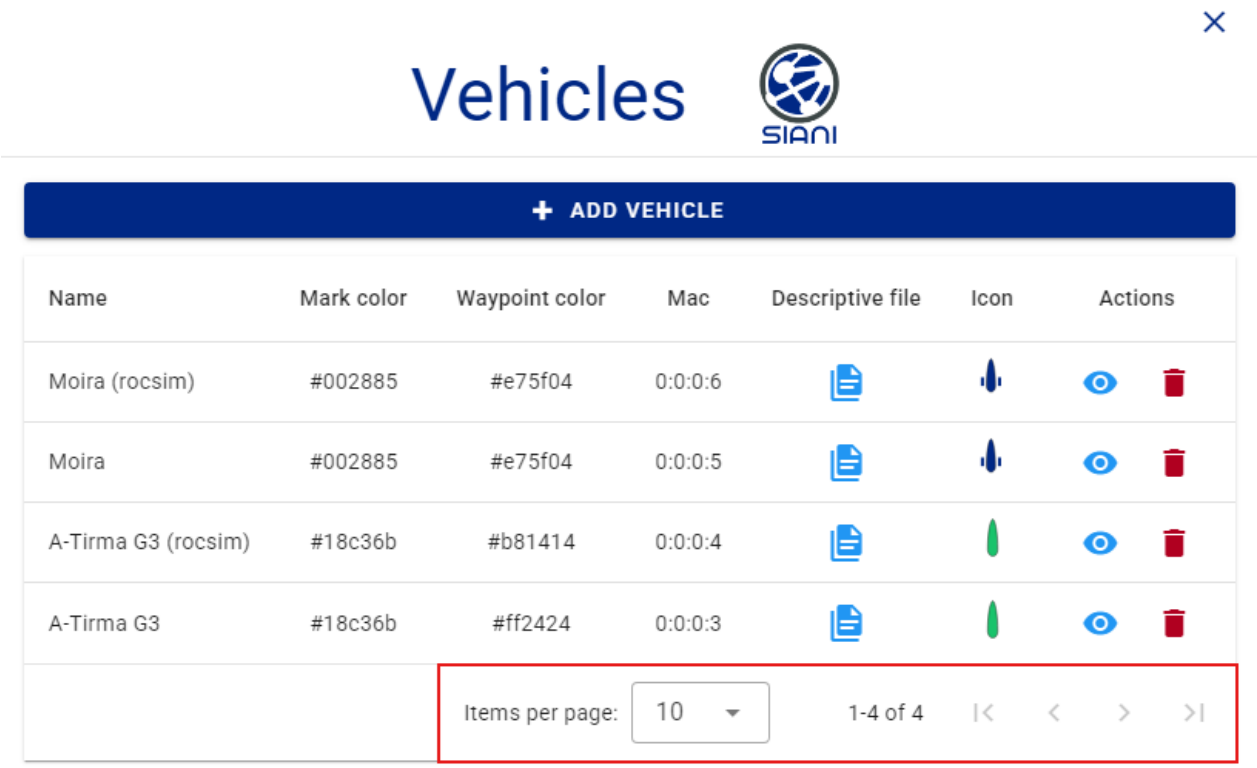


Ilustración 3.33: Diálogo para editar los vehículos con sistema de paginación

3.8.5. Mejoras en Pinia

Pinia es una biblioteca de gestión de estado para Vue.js que utiliza la API de composición de Vue 3. Aunque ya estaba implementada en la versión anterior del proyecto, en esta nueva versión se ha reorganizado su uso, separando los *stores*, (contenedor reactivo que guarda el estado y la lógica de negocio en Pinia [36]), por entidades y funcionalidades.

Gracias a esta estructura modular, cada componente importa únicamente los *stores* que necesita, reduciendo la carga innecesaria y mejorando el mantenimiento del código. Además, se ha centralizado el uso de Pinia para realizar también las peticiones al backend, lo que ha permitido refactorizar el código y reutilizar funciones entre distintos componentes.

3.8.6. Dirección de los puntos de paso

Los *waypoints* o puntos de paso representan el conjunto de coordenadas por las que debe transitar el vehículo. Para facilitar la comprensión del recorrido, se ha implementado una línea que conecta estos puntos, indicando visualmente la dirección del trayecto.

Cuando los *waypoints* son cíclicos —es decir, el recorrido finaliza conectando el último punto con el primero— se representa mediante flechas en la dirección ascendente, como muestra en la Figura 3.34.

En cambio, si los *waypoints* no son cíclicos, el trayecto finaliza en el último punto sin retornar al primero. En este caso, se representa un trayecto de ida y vuelta, ilustrado con flechas en ambos sentidos para indicar que el vehículo volverá al inicio siguiendo el camino inverso. Esto puede observarse en la Figura 3.35.

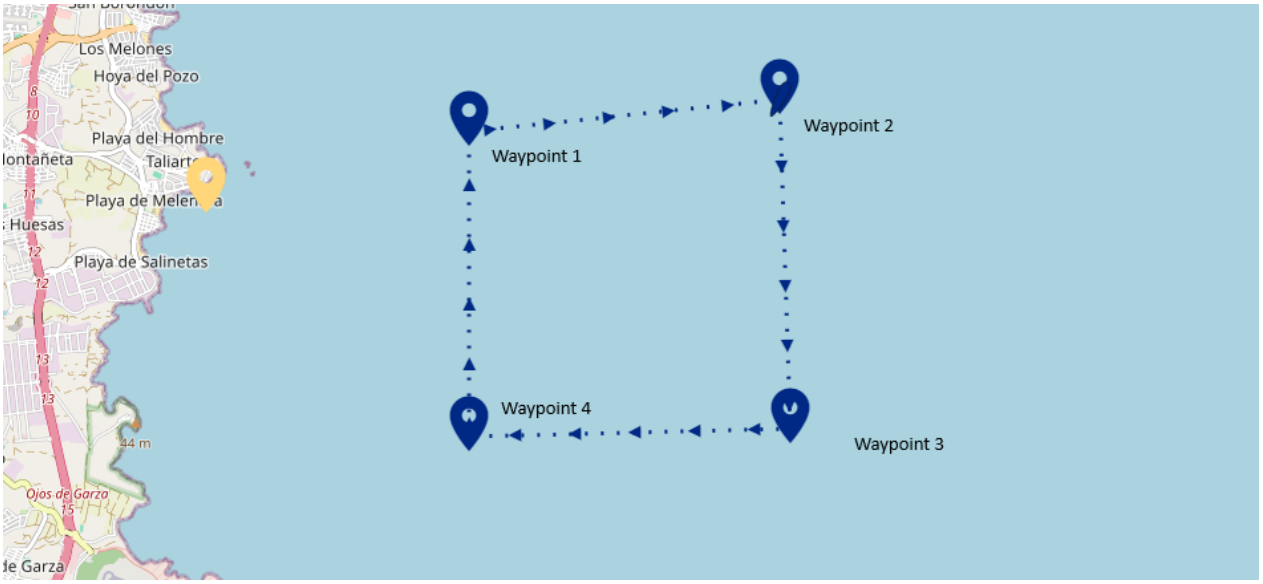


Ilustración 3.34: Trayectoria cíclica: los *waypoints* forman un bucle cerrado

Ilustración 3.35: Trayectoria no cíclica: el recorrido se realiza de ida y vuelta

3.8.7. Rumbo del vehículo

En la versión anterior del sistema, el rumbo del vehículo se implementaba mediante una librería adicional para vue-leaflet [46], concretamente la librería vue-leaflet-rotatedmarker [43]. Para evitar un uso excesivo de dependencias, esta librería ha sido eliminada y reemplazada por un mecanismo más simple para definir el rumbo del vehículo.

Actualmente, se realiza una transformación del ícono que se desea representar (ya sea una boya, un barco a motor o un velero) aplicando una rotación en grados, definida por el campo *heading* del fichero descriptivo.

```
1 <component :is="component" :color="color"
2     :style="{ width: width + 'px', height: height + 'px', transform: `rotate(${rotate}deg)` }" />
```

Código 3.7: Transformación del rumbo del vehículo

3.8.8. Viento

Debido al cambio en los íconos y el formato en que se cargan los íconos del vehículo, ha sido necesario reconfigurar la representación del viento, que ya estaba implementada en la versión anterior. La representación del viento se realiza mediante las barbas de viento, tal como en los mapas meteorológicos. Como se puede observar en la ilustración 3.36, la dirección de la flecha indica de dónde viene el viento, mientras que las pequeñas barbas o líneas en la flecha representan la velocidad.

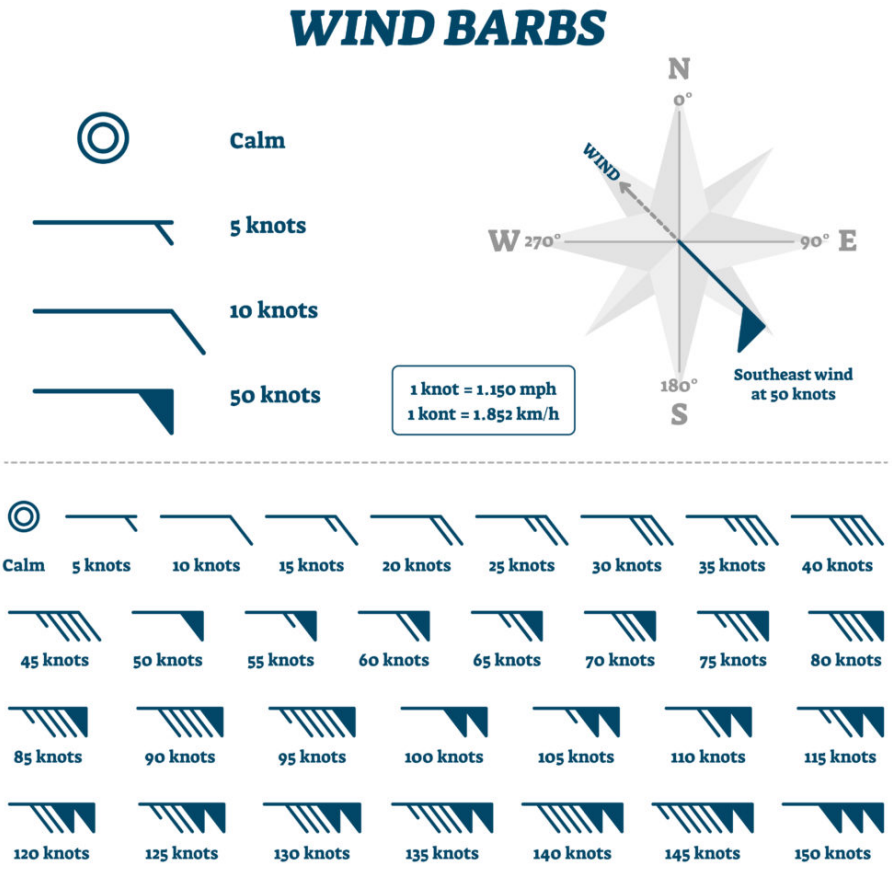


Ilustración 3.36: Representación del viento en mapas meteorológicos [24]

El principal desafío ha sido posicionar correctamente el viento alrededor del vehículo. La solución adoptada consiste en calcular la posición absoluta del viento sobre una circunferencia, es decir, calcular las coordenadas x y y de un punto sobre una circunferencia ficticia para determinar la posición que debe ocupar la representación del viento sobre el vehículo.

Es importante destacar que, además de considerar la dirección del viento, es necesario tener en cuenta la orientación del vehículo. Los vehículos utilizados por la División de Robótica y Oceanografía Computacional del Instituto Universitario SIANI disponen de un sensor que mide la dirección del viento aparente, no la dirección global. Por ello, para obtener la dirección total del viento, se debe sumar la orientación del vehículo a la dirección obtenida del sensor de viento.

Para calcular la posición del viento, es decir, las coordenadas x, y y el ángulo (ya que la línea debe apuntar hacia el barco), se utiliza trigonometría para obtener el punto correspondiente sobre la circunferencia.

Algoritmo 5 Cálculo de posición y ángulo del viento respecto al vehículo

1: $totalDirection \leftarrow (windAngle + heading)$ mód 360
2: $angleRad \leftarrow (totalDirection - 90) \times \frac{\pi}{180}$
3: $x \leftarrow RADIUS \times \cos(angleRad)$
4: $y \leftarrow RADIUS \times \sin(angleRad)$
5: **return** $(x, y, totalDirection)$

Cabe señalar que al calcular los ángulos en radianes se resta 90 grados, ya que el ángulo 0 en HTML comienza en la derecha (este) de la circunferencia, mientras que se desea que el ángulo 0 corresponda a la parte superior (norte) de la circunferencia.

Podemos observar en la ilustración 3.37 cómo se representan 36 posiciones diferentes del viento, distribuidas uniformemente cada 10 grados alrededor del vehículo. Esto significa que se han calculado puntos sobre una circunferencia imaginaria que rodea al vehículo, tomando ángulos que van desde 0° hasta 350° en incrementos de 10°.

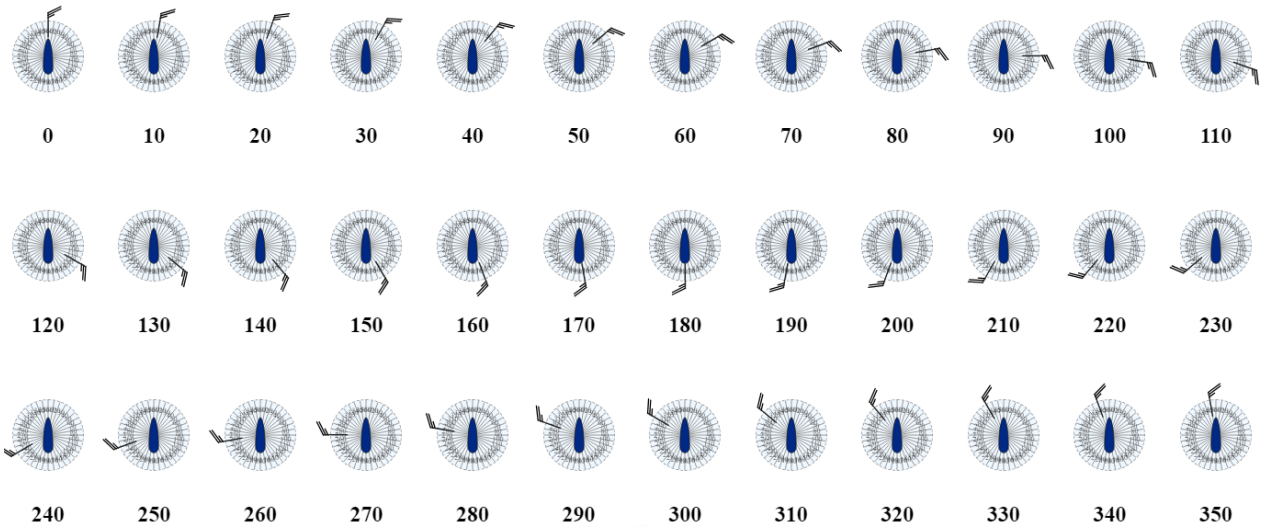


Ilustración 3.37: Ejemplo de los angulos de la dirección del viento

3.8.9. Diálogo para registros en la misión

Durante la simulación o reproducción de una misión, es posible activar un diálogo que muestra el registro actual de cada vehículo en el instante temporal correspondiente. Dado que cada vehículo puede tener campos distintos en sus registros, ya no es viable mostrar esta información de forma estática en el componente de *Vue.js*.

En su lugar, se ha implementado un sistema dinámico que visualiza los datos recibidos en función del vehículo. Para facilitar esta diferenciación, se han incorporado pestañas (*tabs*) de Vuetify, que permiten al usuario seleccionar el vehículo cuyo registro desea consultar.

Otra implementación fue la funcionalidad de aumentar la altura del diálogo para así ver todos las propiedades o valores del registro actual.

En la ilustración 3.38, se muestra un ejemplo de este diálogo informativo.

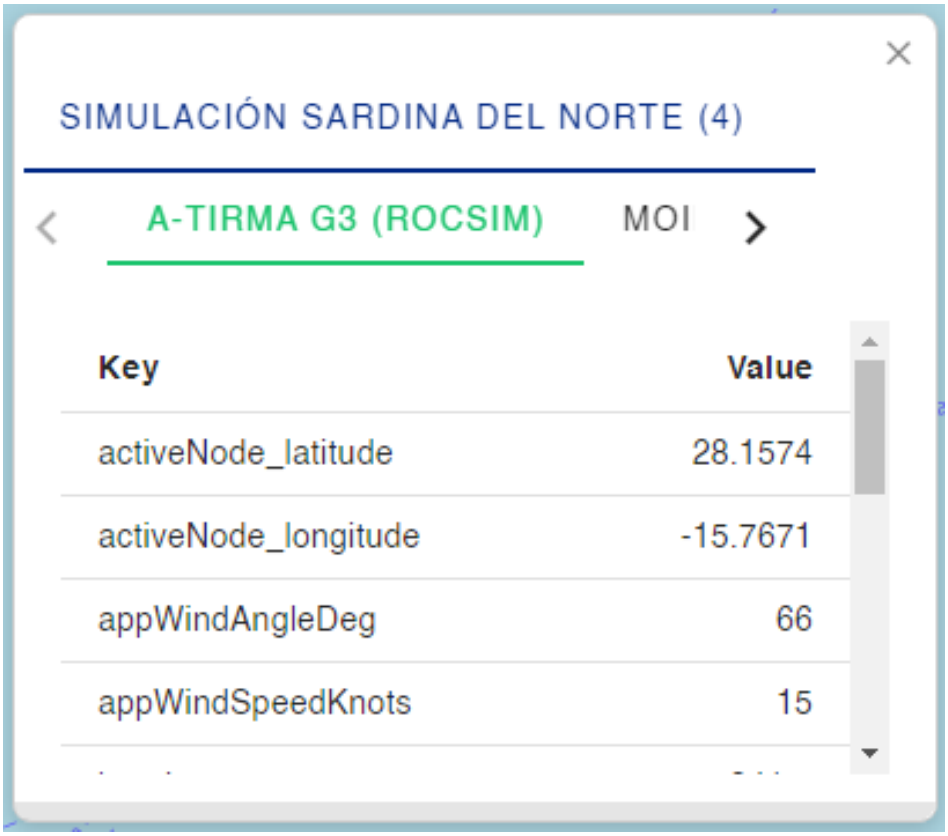


Ilustración 3.38: Diálogo para visualizar el registro actual de los vehículos durante la reproducción

3.8.10. Editar perfil de usuario

Se ha implementado una funcionalidad que permite a los usuarios editar su propio perfil, incluso si no son administradores, es decir, sin tener acceso a la gestión de otros usuarios. En la versión anterior, el usuario no podía cambiar su contraseña; sin embargo, ahora es posible configurar el perfil desde el botón del menú mostrado en la ilustración 3.39.

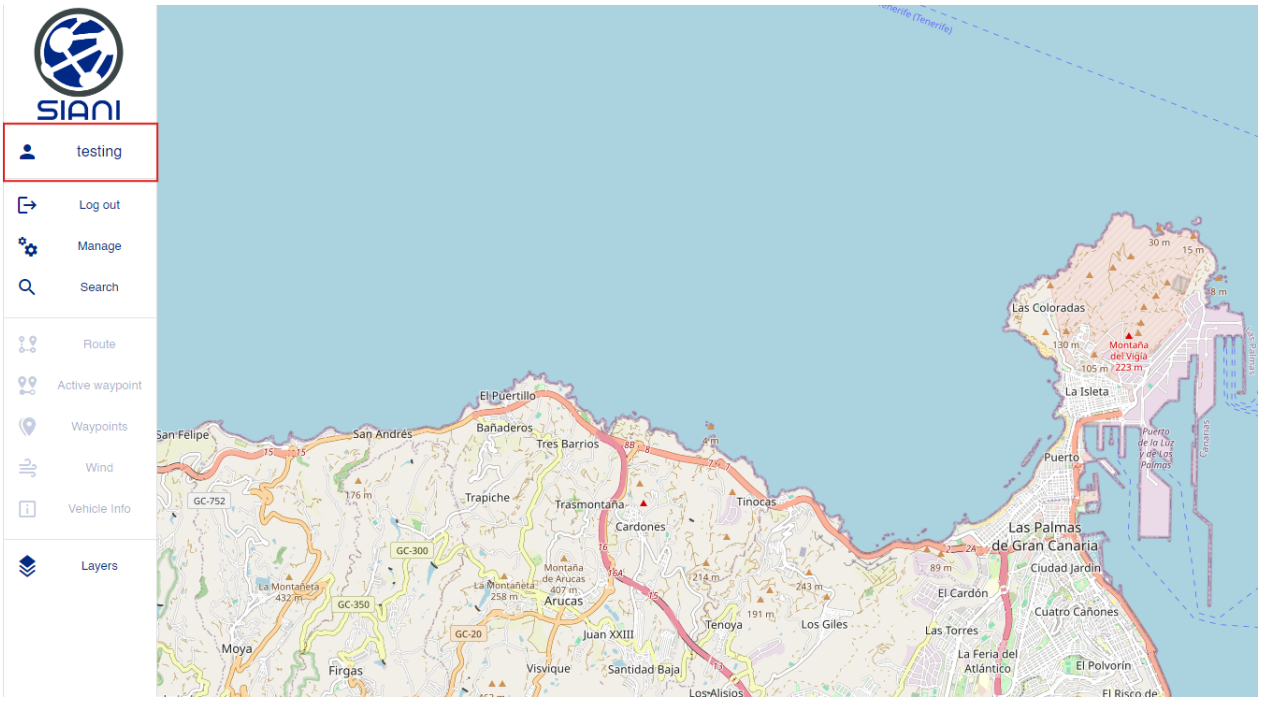


Ilustración 3.39: Botón de acceso a la configuración del perfil de usuario

En la ilustración 3.40 se muestra cómo puede modificarse el perfil de usuario, ya sea el nombre de usuario, el correo electrónico o la contraseña. No se permite configurar el rol del usuario desde este componente.

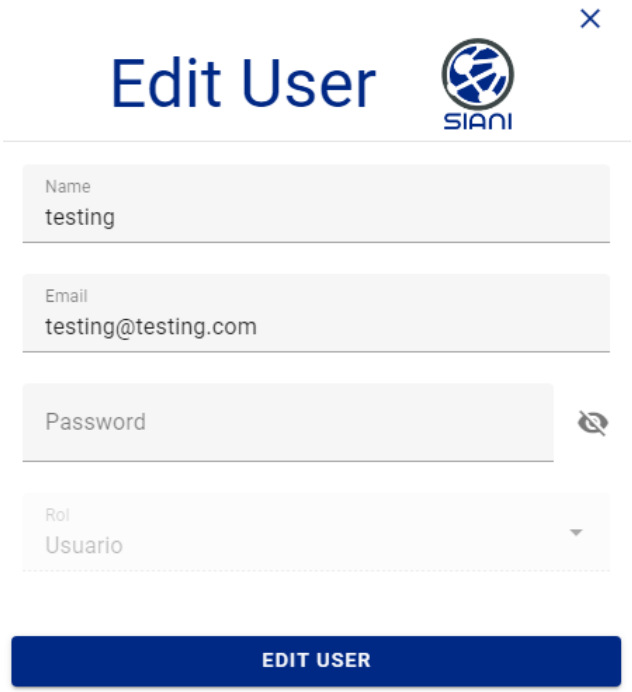


Ilustración 3.40: Diálogo para editar usuario desde el botón de perfil

3.8.11. Cambios en el menú de reproducción de misiones

En esta sección se describen las mejoras realizadas en el menú de reproducción, el cual se muestra al reproducir o simular misiones. En la ilustración 3.41 se puede observar el aspecto original del componente, mientras que en la ilustración 3.42 se muestra su apariencia tras las modificaciones implementadas en esta versión.

Ilustración 3.41: Menú de reproducción de una misión en la versión anterior

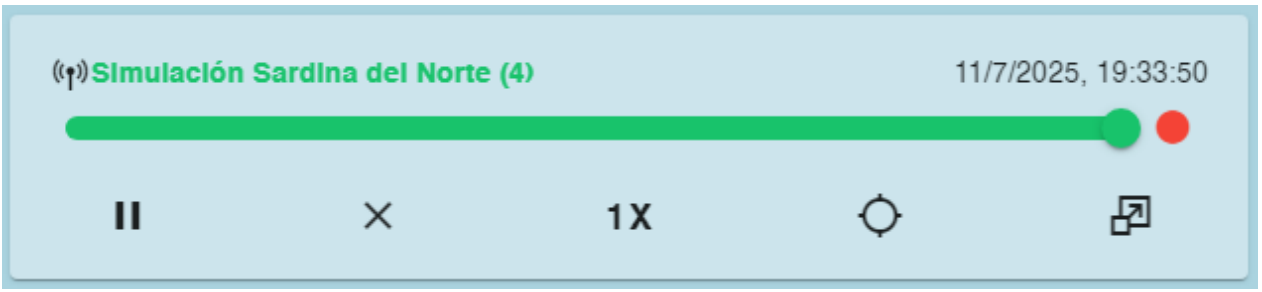


Ilustración 3.42: Menú de reproducción de una misión

3.8.11.1. Modificar el tamaño del vehículo durante la reproducción

En el menú de reproducción de la misión se ha implementado una lista que muestra cada vehículo, permitiendo modificar su tamaño. Es posible aumentar o disminuir proporcionalmente el ancho y alto del ícono que representa al vehículo en el mapa. De este modo, el usuario puede resaltar o reducir la relevancia visual de un vehículo en particular. Esta funcionalidad se muestra destacada en el recuadro rojo de la ilustración 3.43.

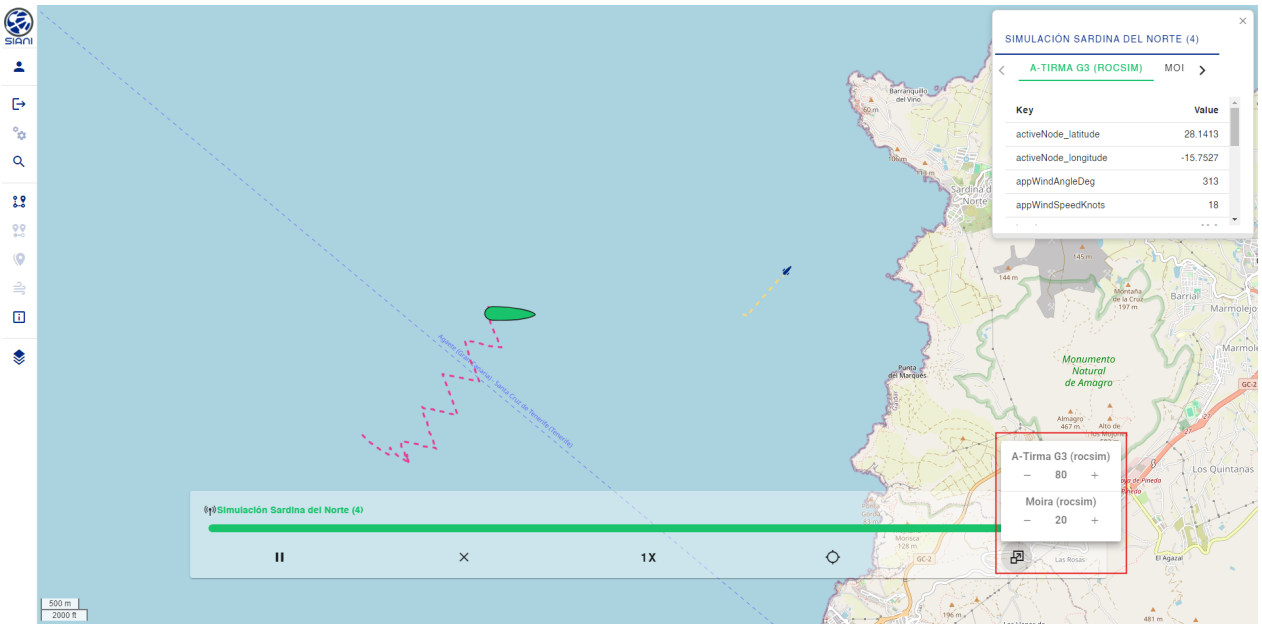


Ilustración 3.43: Opción del menú de la misión para cambiar el tamaño del vehículo seleccionado

3.8.11.2. Botón - Ir al directo

Se ha implementado una funcionalidad en el menú de reproducción para misiones en vivo que permite al usuario volver al punto actual del directo. Esta funcionalidad se representa mediante un botón cuya apariencia varía según la posición de la reproducción: si el usuario se encuentra en el último registro recibido, el botón se muestra en color rojo; en caso contrario, aparece en gris. Esta lógica simula la funcionalidad que ofrecen plataformas como YouTube en sus transmisiones en directo. Al pulsar el botón, la reproducción se sincroniza con el directo, cargando los registros correspondientes al instante actual si es necesario.

En la ilustración 3.42 se muestra el botón activado, mientras que en la ilustración 3.44 el botón aparece en gris, ya que no se está visualizando el último registro de la misión, es decir, no se está en ”directo”.

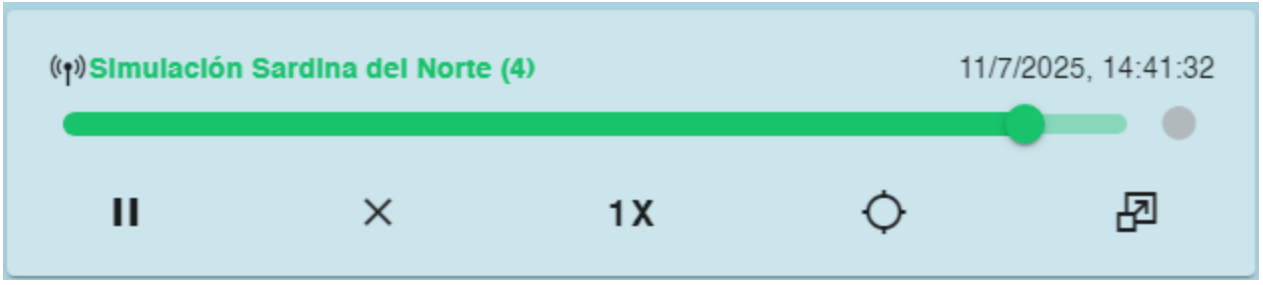


Ilustración 3.44: Menú de reproducción de una misión con el tiempo actual atrasado

3.9. Pruebas

En esta sección se presentan una serie de pruebas con el objetivo de mostrar la mayoría de las funcionalidades implementadas en la nueva versión de este Trabajo de Fin de Título. Además, se incluye el enlace en el que se encuentra desplegada la infraestructura web: https://atirma.iusiani.ulpgc.es/ a fecha de 24 de julio de 2025, donde el sistema se encuentra en producción..

3.9.1. Misión en directo con varios vehículos

Uno de los principales objetivos de este TFT consiste en permitir varios vehículos en una misma misión. A continuación, se presenta una misión con dos vehículos: *A-Tirma G3*, barco autónomo a vela, y *Moira*, un vehículo autónomo de superficie a motor. En este caso, ambos vehículos están llevando a cabo una misión utilizando dos simuladores hardware-on-the-loop, uno específico para cada uno de los vehículos mencionados. Desde el punto de vista del software de la infraestructura web desarrollada en ese proyecto, no hay diferencias desde el punto de vista de la operativa del sistema con una misión llevada a cabo con vehículos reales.

Como se ha mencionado, ambos vehículos están virtualizados mediante un simulador hardware que emula su comportamiento real. Estos dispositivos envían paquetes de telemetría cada 6 segundos al servidor backend. En el frontend, deben estar registrados con sus respectivas direcciones MAC, como se muestra en la ilustración 3.45. Es fundamental que dichas direcciones coincidan con las que se reciben en los paquetes de telemetría, ya que es la manera de relacionar los paquetes recibidos con los vehículos almacenados en el sistema.

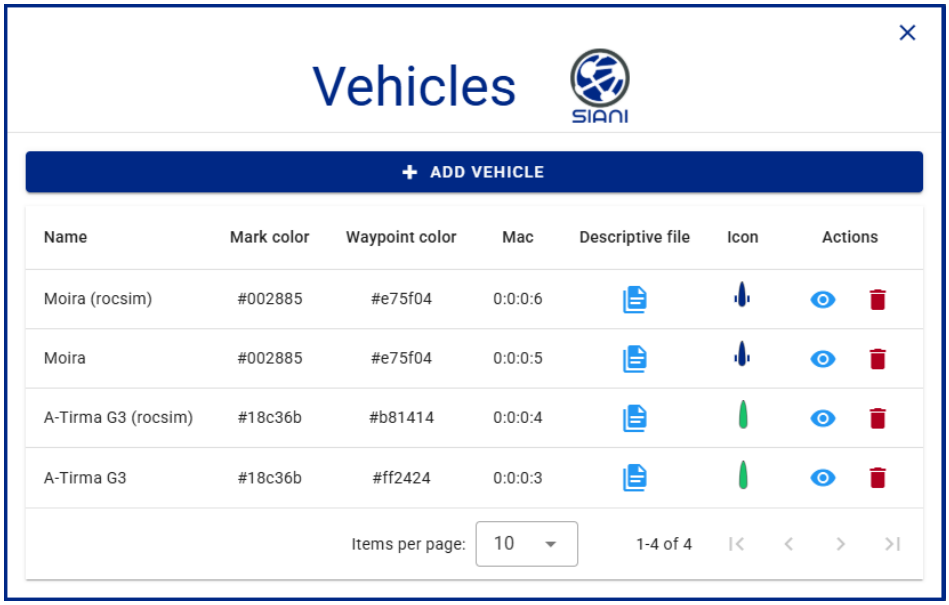


Ilustración 3.45: Diálogo de visualización de vehículos

La misión *Simulación Sardina del Norte (5)* es la que se utiliza en esta prueba. Tal como se muestra en la ilustración 3.46, esta misión incluye a los dos vehículos mencionados.

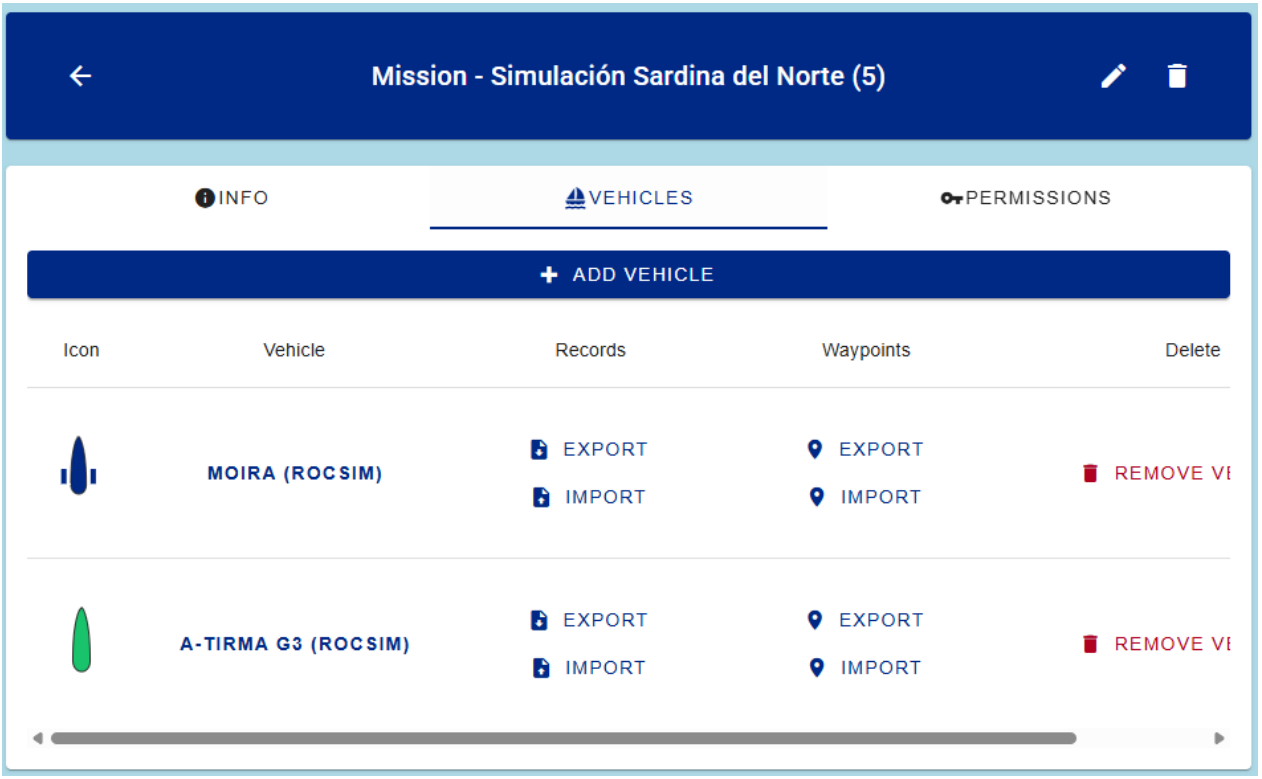
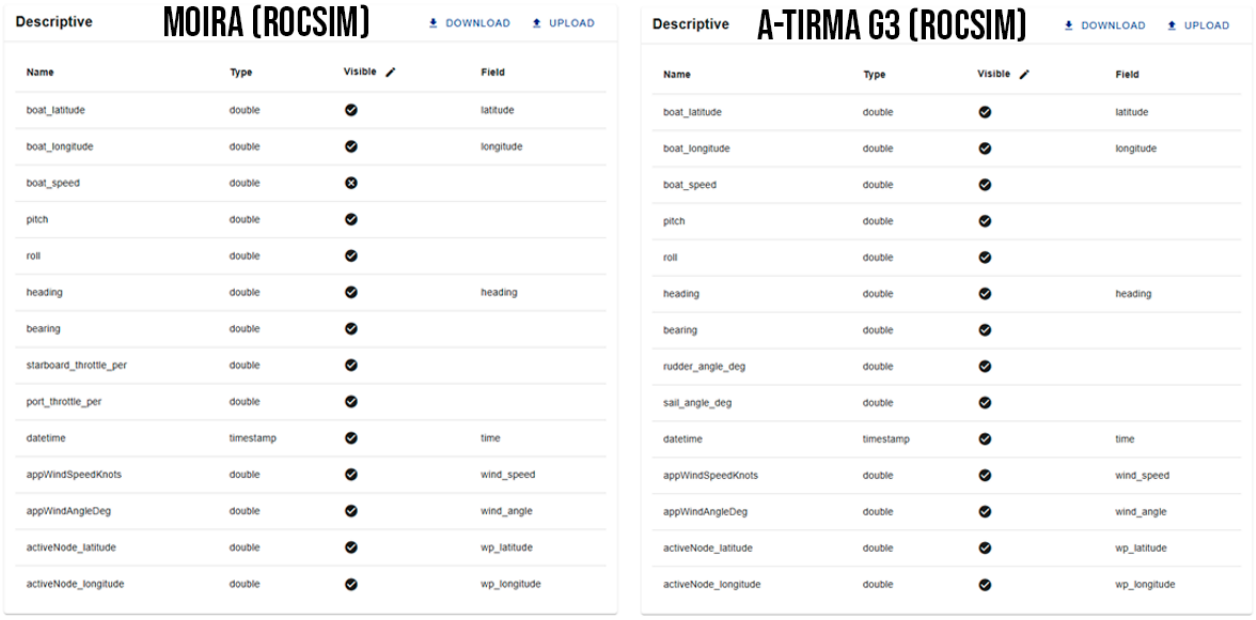


Ilustración 3.46: Vehículos asignados a la misión *Simulación Sardina del Norte (5)*

3.9.1.1. Ficheros descriptivos de los vehículos

La misión incluye dos vehículos con ficheros descriptivos distintos. En la ilustración 3.47 se muestran las propiedades específicas de cada uno. Por ejemplo, el campo `starboard_throttle_per` es exclusivo de *Moira (Rocsim)*, mientras que `sail_angle_deg` aparece únicamente en *A-Tirma G3 (Rocsim)*.



Descriptive **MOIRA (ROCSIM)** DOWNLOAD UPLOAD

Name	Type	Visible	Field
boat_latitude	double	✓	latitude
boat_longitude	double	✓	longitude
boat_speed	double	✗	
pitch	double	✓	
roll	double	✓	
heading	double	✓	heading
bearing	double	✓	
starboard_throttle_per	double	✓	
port_throttle_per	double	✓	
datetime	timestamp	✓	time
appWindSpeedKnots	double	✓	wind_speed
appWindAngleDeg	double	✓	wind_angle
activeNode_latitude	double	✓	wp_latitude
activeNode_longitude	double	✓	wp_longitude

Descriptive **A-TIRMA G3 (ROCSIM)** DOWNLOAD UPLOAD

Name	Type	Visible	Field
boat_latitude	double	✓	latitude
boat_longitude	double	✓	longitude
boat_speed	double	✓	
pitch	double	✓	
roll	double	✓	
heading	double	✓	heading
bearing	double	✓	
rudder_angle_deg	double	✓	
sail_angle_deg	double	✓	
datetime	timestamp	✓	time
appWindSpeedKnots	double	✓	wind_speed
appWindAngleDeg	double	✓	wind_angle
activeNode_latitude	double	✓	wp_latitude
activeNode_longitude	double	✓	wp_longitude

Ilustración 3.47: Propiedades descriptivas de cada vehículo

3.9.1.2. Pasos para reproducir la misión

Para reproducir la misión en tiempo real, deben seguirse los siguientes pasos:

1. **Buscar misiones:** Pulsar el botón del menú con el icono de una lupa para acceder al listado de misiones disponibles. Luego, acceder al submenú *Missions*.
2. **Filtrar por misiones activas:** Seleccionar la pestaña *Actives* en la parte superior de la vista de búsqueda.
3. **Reproducir la misión:** Pulsar en el icono de *play* (representado con un símbolo de suma) para iniciar la reproducción.
4. **Seleccionar intervalo:** Elegir el intervalo de tiempo deseado para la reproducción. Se recomienda un valor de 6 segundos, ya que es la frecuencia con la que los vehículos envían paquetes de telemetría.

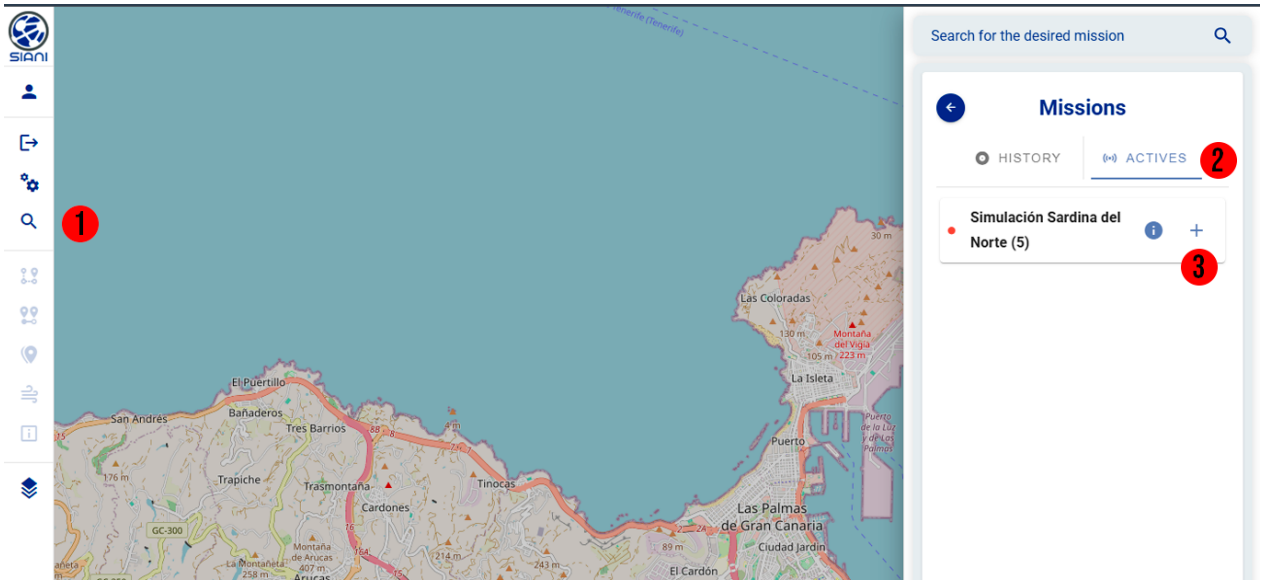


Ilustración 3.48: Pasos para reproducir una misión en tiempo real

3.9.1.3. Visualización de rutas

Para visualizar las rutas de los vehículos, debe pulsarse el botón correspondiente en el menú principal. En la ilustración 3.49 se muestra el control que permite activar o desactivar esta visualización.

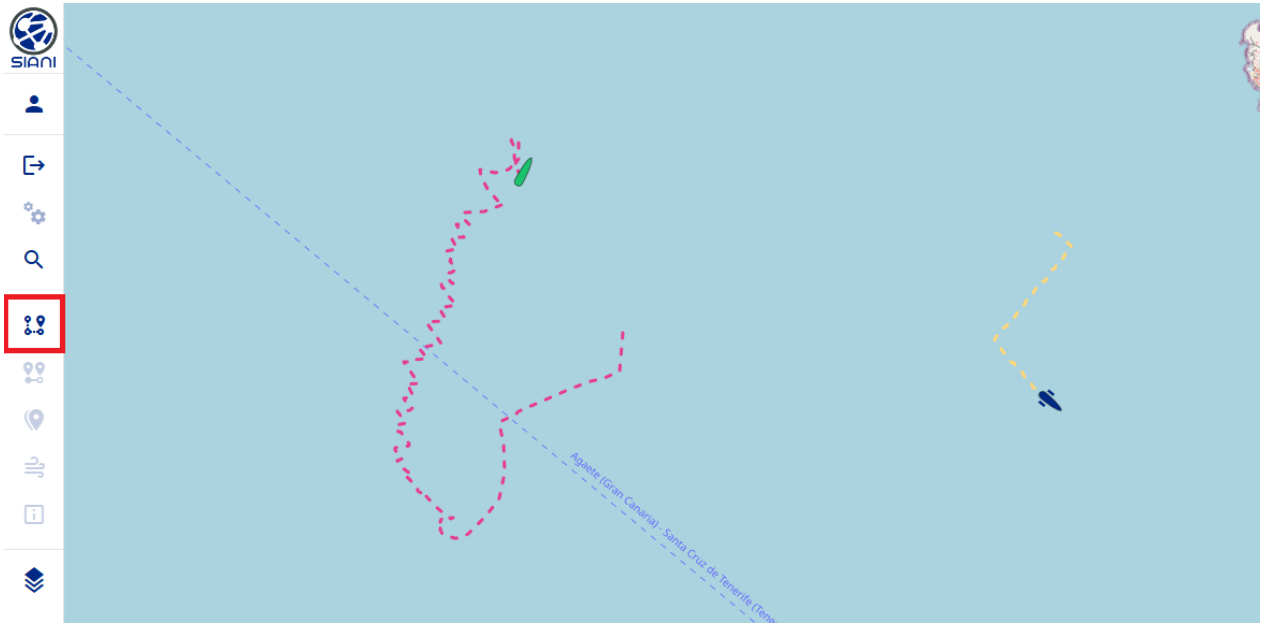


Ilustración 3.49: Visualización de rutas de los vehículos en la misión *Simulación Sardina del Norte (5)*

3.9.1.4. Waypoint activo

El *waypoint activo* es el punto de paso hacia el que se dirige actualmente un vehículo. Para visualizarlo, se debe activar el botón indicado en la ilustración 3.50.

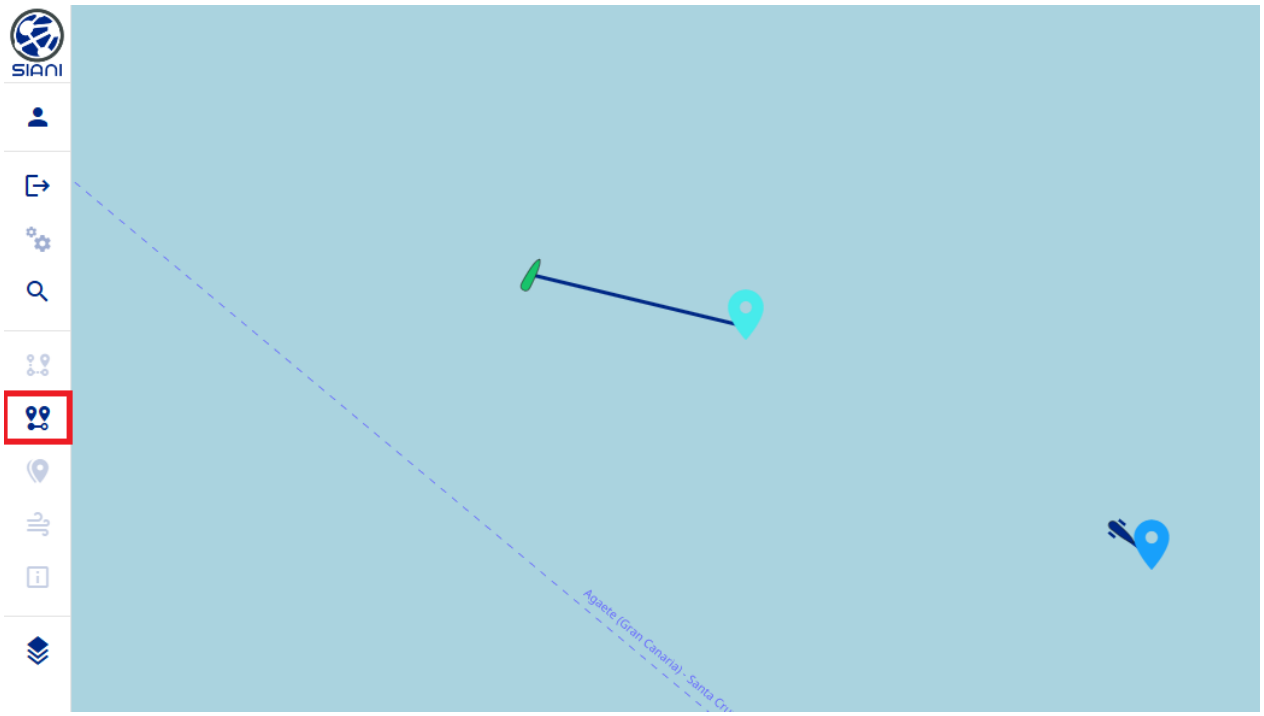


Ilustración 3.50: Visualización del *waypoint* activo en la misión *Simulación Sardina del Norte (5)*

3.9.1.5. Waypoints o puntos de paso

Para mostrar todos los *waypoints*, se debe seleccionar el botón señalado en la ilustración 3.51.

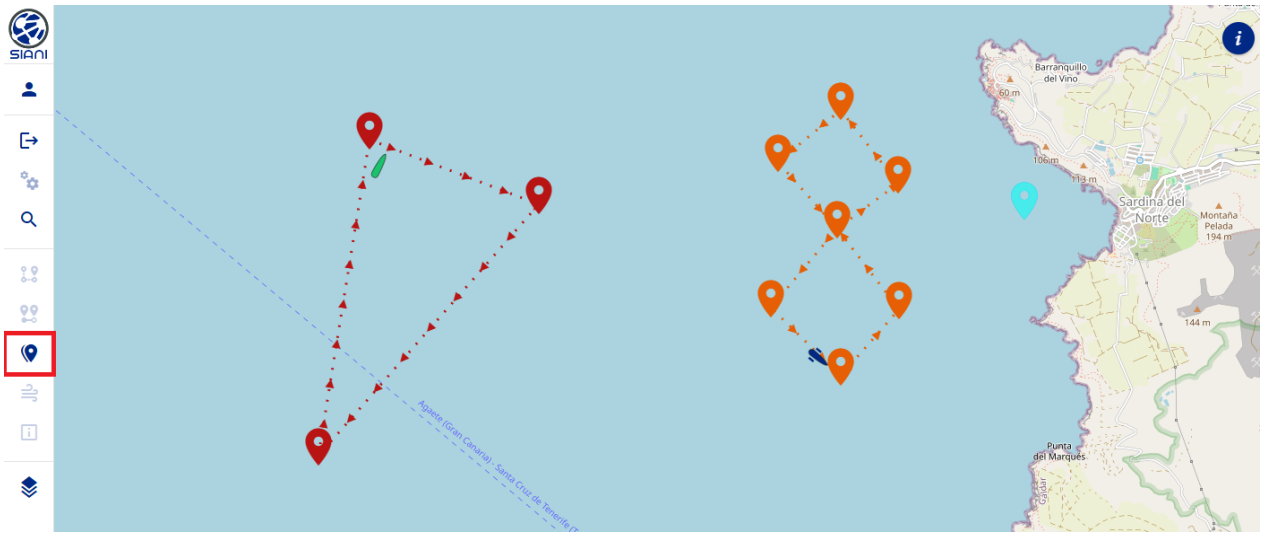


Ilustración 3.51: Waypoints de los vehículos en la misión *Simulación Sardina del Norte (5)*

3.9.1.6. Visualización del viento y registros

Para visualizar la dirección del viento, se debe activar el botón marcado con un recuadro rojo en la ilustración 3.52. Para consultar el registro actual de cada vehículo, debe pulsarse el botón señalado en verde en la misma figura.

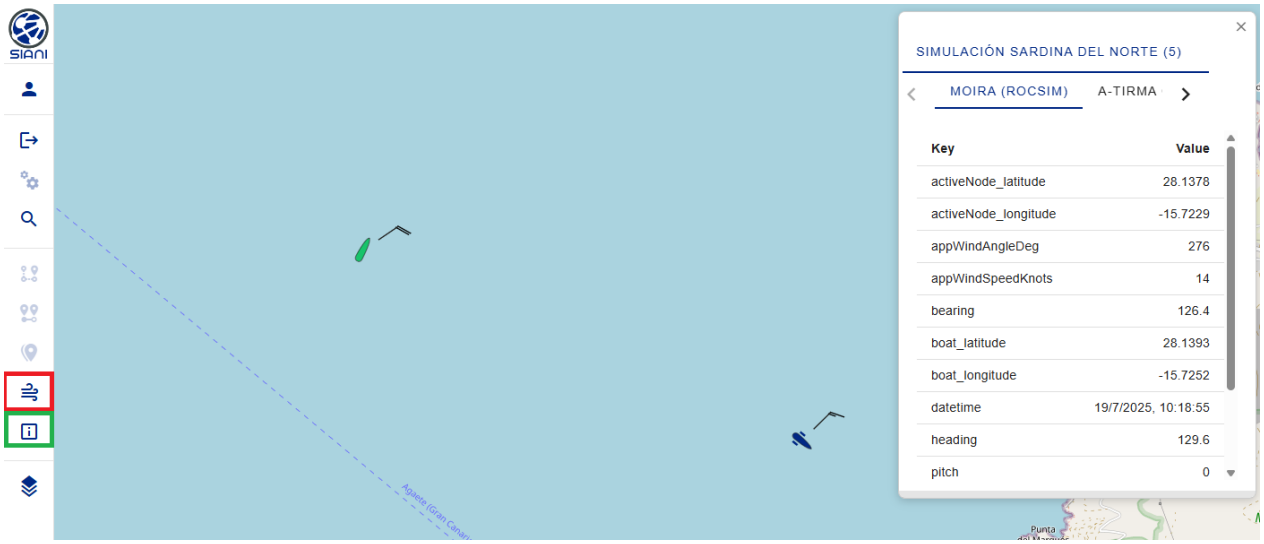


Ilustración 3.52: Visualización del viento y registros en tiempo real

3.9.1.7. Misión en directo

Una misión se considera en directo cuando el botón de *play* está activo y el indicador de *live* aparece en rojo. Esto indica que se está visualizando el último paquete de telemetría recibido.

En la ilustración 3.53 se observa la misión en directo con todos los elementos de visualización activados: rutas, *waypoints* activos, puntos de paso, dirección del viento y diálogo de registros.

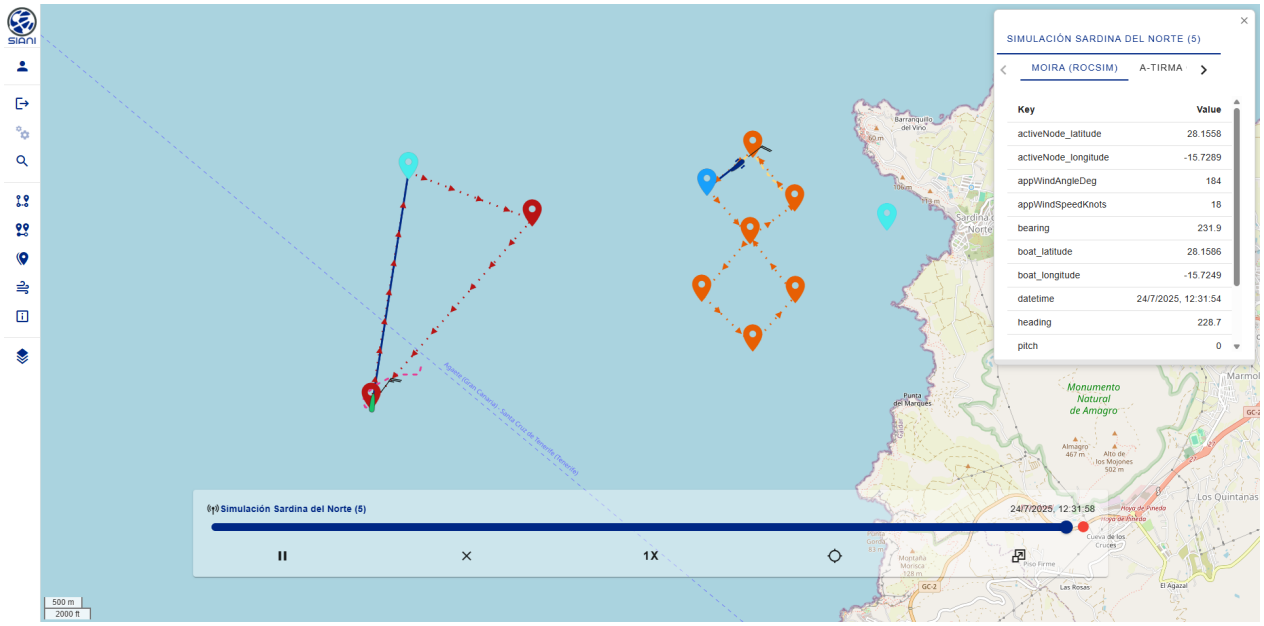


Ilustración 3.53: Misión en directo con todas las visualizaciones activadas

3.10. Gestión de ficheros CSV del backend

Desde la sección de gestión de ficheros CSV en el backend, los administradores del sistema pueden descargar o eliminar estos archivos de manera sencilla. Esto facilita la limpieza del almacenamiento y la administración eficiente del sistema. En la ilustración 3.54 se muestra la interfaz correspondiente.

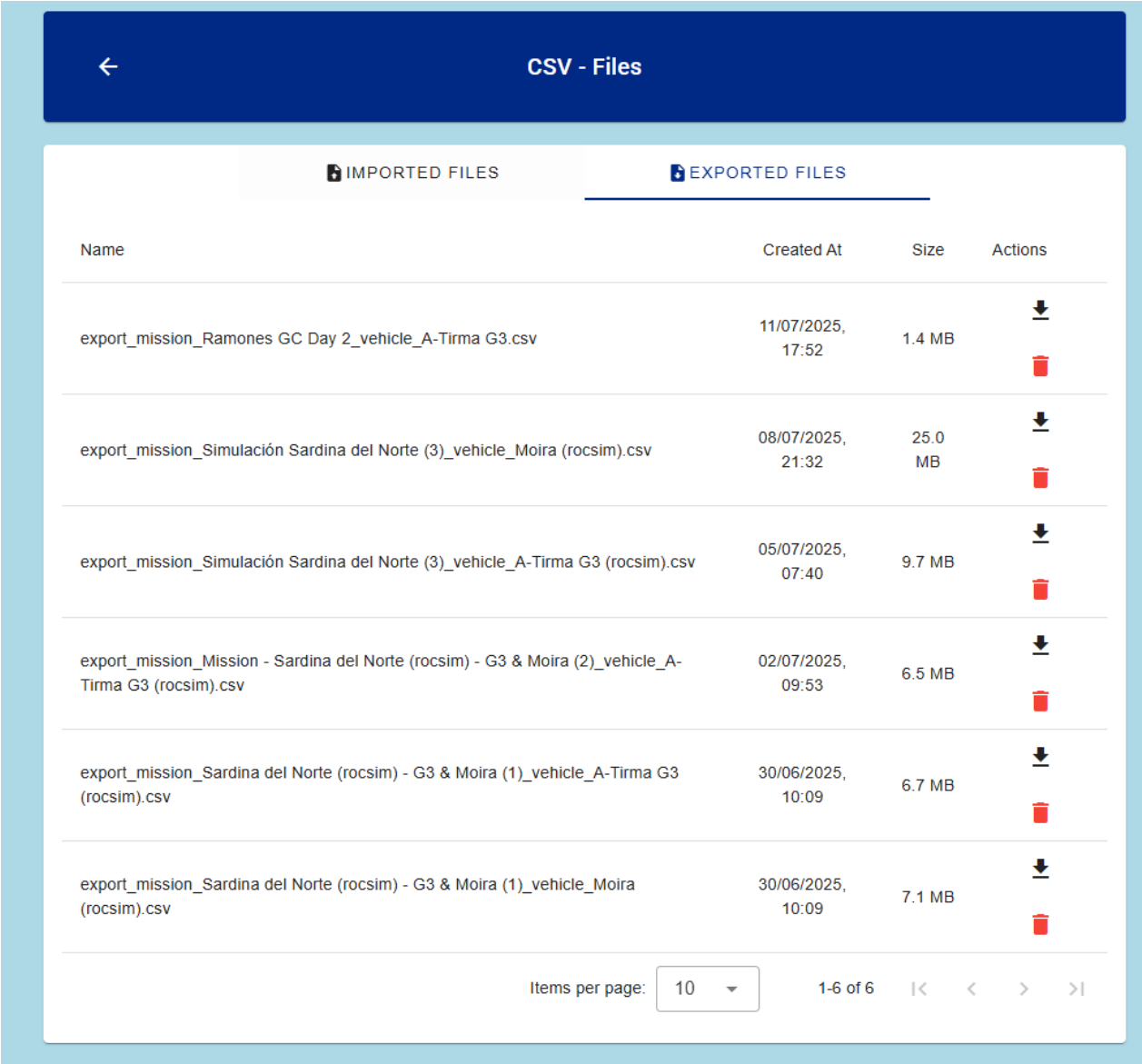


Ilustración 3.54: Gestión de ficheros CSV desde el backend

3.11. Exportar e importar misiones

La funcionalidad de exportación e importación de misiones está disponible únicamente para los propietarios o usuarios con privilegios de edición sobre una misión. Esta permite exportar o importar registros y *waypoints* en formato CSV.

Como ejemplo, se presenta una prueba de importación de registros CSV a una misión. Para ello, es necesario acceder al menú de configuración, seleccionar la sección de vehículos y pulsar el botón *Import Records* correspondiente. Esto abrirá el diálogo mostrado en la ilustración 3.55.

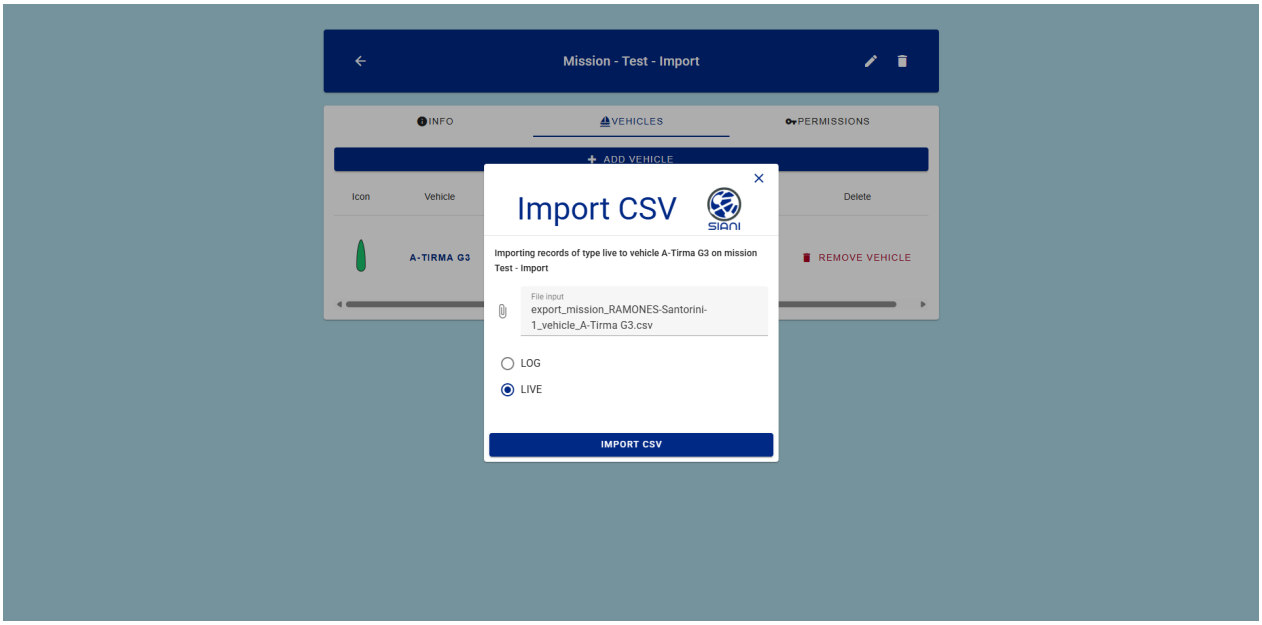


Ilustración 3.55: Diálogo para importar registros a un vehículo

Una vez seleccionado el archivo CSV y el tipo de dato a importar, se inicia el proceso. Si el formato del archivo no coincide con el fichero descriptivo del vehículo, se mostrará una notificación de error en rojo. En caso de éxito, se mostrará una notificación en verde, como se observa en la ilustración 3.56.

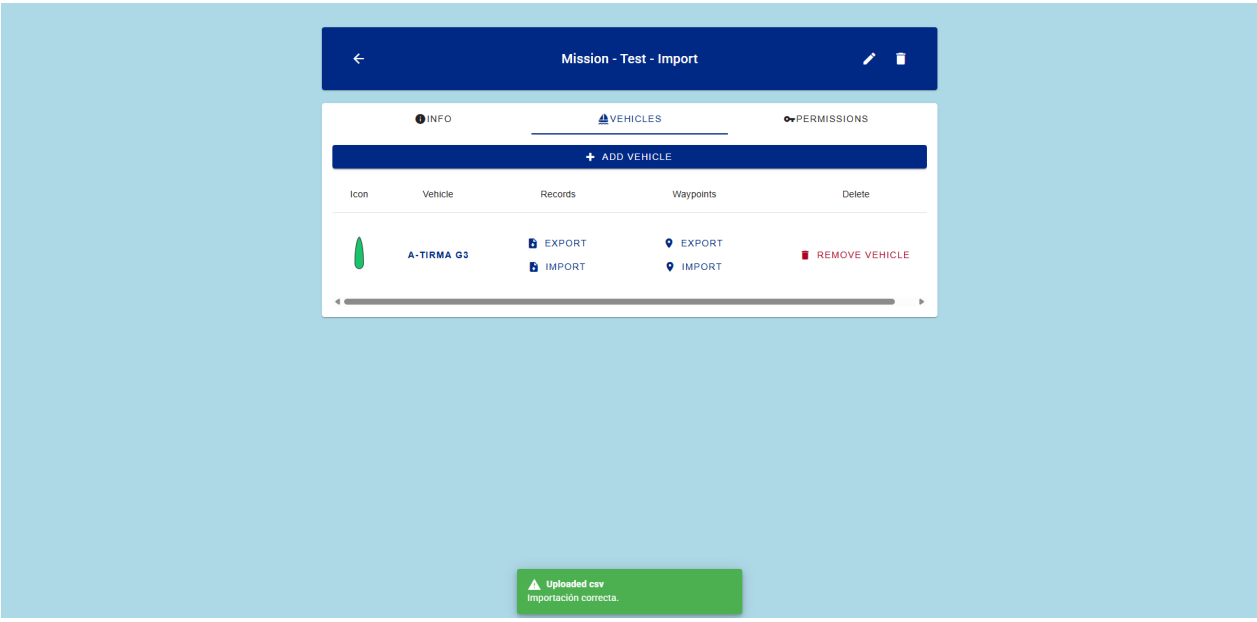


Ilustración 3.56: Notificación de importación exitosa de registros

Capítulo 4

Conclusiones y trabajo futuro

El proyecto A-Tirma WebWare v3 ha resultado en una serie de avances significativos en diversas áreas clave, fortaleciendo tanto la funcionalidad como la usabilidad de la plataforma. En este capítulo se detallan las conclusiones específicas relacionadas con cada aspecto del proyecto, destacando las mejoras implementadas y su impacto en el rendimiento y la experiencia del usuario. Además, se presentarán las oportunidades de mejora y los trabajos futuros que podrían potenciar aún más el sistema, asegurando su adaptabilidad y eficacia en un entorno en constante evolución. Esta sección proporciona una visión global de los logros alcanzados.

4.1. Conclusiones

4.1.1. Mejoras en la Interfaz de Usuario

La implementación correcta de Vuetify, ha mejorado notablemente la experiencia del usuario. La interfaz ahora es más intuitiva y eficiente, facilitando la navegación y el acceso a las funcionalidades del sistema. Esto ha resultado en una mayor satisfacción del usuario y ha simplificado la interacción con los datos de las misiones de vehículos autónomos marinos.

A continuación se enumerarán algunos de los detalles más importantes:

1. **Rediseño visual:** Con la reestructuración del flujo de navegación se han introducido accesos directos a las entidades para su visualización y edición que reduce el tiempo y esfuerzo necesarios para realizar tareas comunes.
2. **Vistas de entidades:** Mejora notable en las vistas generales de las entidades, donde ahora se muestra más información que en la versión anterior para cada una de ellas.
3. **Distintos iconos para los vehículos:** Se ha implementado distintos iconos para los vehículos del sistema. De esta manera el usuario podrá diferenciar vehículos de una manera más sencilla.

4. **Mejora en la vista de registros:** El diálogo para mostrar los registros de los vehículos en una misión se ha mejorado y refactorizado. Así convirtiéndose en un componente que se expanda en tamaño para su mayor legibilidad.

4.1.2. Vehículos con propiedades descriptivas

La funcionalidad de incorporar vehículos con propiedades descriptivas ha sido el mayor desafío de este Trabajo de Fin de Título, pero también la que ha aportado mayores ventajas al proyecto.

La posibilidad de añadir vehículos con distintos sensores permite a la División de Robótica y Oceanografía Computacional del Instituto Universitario SIANI una mayor libertad a la hora de construir los paquetes que los vehículos enviarán al servidor. Además, al contar con propiedades descriptivas variables, los vehículos definidos en el sistema pueden presentar una estructura más precisa y adaptada a sus características. En los casos en que un vehículo tenga menos campos que lo definan, la reproducción de la misión será más fluida.

En definitiva, es posible añadir al sistema, y a cualquier misión, cualquier tipo de vehículo que encaje en la tipología de 4 tipos de vehículos considerados, a saber, vehículo a vela, vehículo a motor, boya o derivador y otro, en el que incluso vehículos del mismo tipo pueden diferenciarse en los datos suministrado en sus mensajes de telemetría, siendo los datos obligatorios, únicamente los de posición, i.e., longitud y latitud, y su sello temporal. Así es posible incluir en el sistema cualquier vehículo o dispositivo marino que pueda transmitir esta información al sistema

4.1.3. Exportación e importación de misiones

La implementación de la funcionalidad de exportación e importación de misiones ha sido fundamental para preservar las misiones creadas en versiones anteriores del sistema y también permite llevarlas a otras herramientas de análisis de datos para su procesamiento posterior. Se ha añadido la posibilidad de importar tanto los registros como los waypoints, al igual que su respectiva exportación.

4.2. Trabajo Futuro

Como trabajo futuro del presente Trabajo de Fin de Título, se presentan las siguientes funcionalidades a implementar:

- **Gráficos analíticos de las misiones**: Poder visualizar los datos de una misión mediante gráficos de manera simple y rápida.
- **Exportar las misiones en otros formatos**: Tener la posibilidad de exportar una misión en formato Google Earth, video, etcétera.

4.3. Objetivos de desarrollo sostenible

Cuadro 4.1: Grado de relación del TFT con los objetivos de desarrollo sostenible.

ODS	Grado de relación			
	0 No procede	1 Bajo	2 Medio	3 Alto
1 Fin de la Pobreza	x			
2 Hambre cero	x			
3 Salud y Bienestar	x			
4 Educación de calidad	x			
5 Igualdad de género	x			
6 Agua limpia y saneamiento	x			
7 Energía Asequible y no contaminante	x			
8 Trabajo decente y crecimiento económico	x			
9 Industria, Innovación e Infraestructuras			x	
10 Reducción de las desigualdades	x			
11 Ciudades y comunidades sostenibles	x			
12 Producción y consumo sostenibles	x			
13 Acción por el clima	x			
14 Vida submarina				x
15 Vida de ecosistemas terrestres	x			
16 Paz, justicia e instituciones sólidas	x			
17 Alianzas para lograr objetivos	x			

Uno de los objetivos que tiene el Trabajo de Fin de Grado a presentar consiste en monitorizar misiones de embarcaciones autónomas, algunas de ellas se encargan de comprobar el estado de la vida submarina, escuchar grupos de cetáceos o comprobar la variedad de fauna marina. En el Cuadro 4.1 se destacan los Objetivos de Desarrollo Sostenible relacionados con el trabajo llevado a cabo en este proyecto.

Apéndice A

Apéndice de despliegue

En esta sección del Trabajo de Fin de Título se exponen los pasos necesarios para desplegar la infraestructura web del proyecto. A día 24 de julio de 2025 este TFT se encuentra en producción en el siguiente enlace https://atirma.iusiani.ulpgc.es/.

A.1. Lighttpd para el despliegue del servidor web

Como se expone en la sección 3.1.5, Lighttpd [38] es el servidor web que se utilizará para este Trabajo de Fin de Título. Se recomienda seguir los siguientes pasos para su correcta instalación en un equipo bajo GNU/Linux en una distribución compatible con Debian/Ubuntu:

- Instalar Lighttpd:

  ```
  sudo apt-get install lighttpd
  ```

- Crear los certificados correspondientes y editar el archivo de configuración del servidor web:

  ```
  sudo nano /etc/lighttpd/conf-enabled/99-atirma_proxy.conf
  ```

- Configurar el archivo con el siguiente contenido:

```
server.modules   += ( "mod_openssl",
                      "mod_proxy",
                      "mod_setenv",
)

$SERVER["socket"] == "192.168.53.9:80" {
    $HTTP["host"] == "atirma.iusiani.ulpgc.es" {
        url.redirect = ("^/.*" => "https://atirma.iusiani.ulpgc.es/%0$0")
    }
}
else $SERVER["socket"] == "192.168.53.9:443" {
```

```
    ssl.engine  = "enable"
    ssl.ca-file = "/etc/ssl/certs/ca-certificates.crt"
    ssl.pemfile = "/etc/lighttpd/ssl/atirma_web06.pem"
    ssl.cipher-list = "ECDHE-RSA-AES256-SHA384:AES256-SHA256:RC4:HIGH:!MD5:!aNULL:!EDH:!AESGCM"
    ssl.honor-cipher-order = "enable"

    $HTTP["host"] == "atirma.iusiani.ulpgc.es" {
        $HTTP["url"] =~ "^/api" {
            proxy.server = (
                "" => (( "host" => "127.0.0.1", "port" => 3000 ))
            )
        }
        else $HTTP["url"] =~ "^/live" {
            proxy.server = (
                "" => (( "host" => "127.0.0.1", "port" => 3000 ))
            )
        }
        else $HTTP["url"] =~ "^/ws" {
            setenv.add-request-header = (
                "Upgrade" => "websocket",
                "Connection" => "upgrade"
            )
            setenv.add-response-header = (
                "Upgrade" => "websocket",
                "Connection" => "upgrade"
            )
            proxy.server = (
                "" => (( "host" => "127.0.0.1", "port" => 3001 ))
            )
            proxy.header = (
                "upgrade" => "enable",
                "connect" => "enable"
            )
        }
        else {
            server.document-root = "/ruta/al/repositorio/frontend/dist"
        }
    }

    url.rewrite-if-not-found = (
        "/.*" => "/index.html"
    )
}
index-file.names := ( "index.html" )
```

Código A.1: Fichero 99-atirma_proxy.conf

- Solicitar permisos para descargar el código fuente

  El código fuente del proyecto se encuentra en el repositorio https://git.iusiani.ulpgc.es/jpereiro/a-tirma-webware-v3 del Instituto Universitario SIANI. Al tratarse de un código privado, puede solicitar acceso a los tutores de este trabako o al propio autor.

- Navegar al directorio raíz del servidor

  ```
  cd /ruta/al/repositorio
  ```

- Clonar el código fuente del proyecto

```
git clone https://git.iusiani.ulpgc.es/git/jpereiro/a-tirma-webware-v3.git
```

- Reiniciar el servicio Lighttpd para aplicar los cambios:

```
sudo systemctl restart lighttpd.service
```

A.2. Despliegue del backend

Para desplegar correctamente el backend, este se tiene que convertir en un demonio o servicio, de esta manera, el programa se ejecutará de manera continuada en el servidor.

- Crear un archivo de servicio en systemd:

```
sudo nano /etc/systemd/system/backend.service
```

- Añadir al fichero de servicio la siguiente configuración:

```
[Unit]
Description=A-Tirma Backend
After=network.target

[Service]
WorkingDirectory=/usr/local/share/atirma-app
User=root
Group=www-data
ExecStart=/usr/local/bin/node /ruta/al/repositorio/backend/index.js
EnvironmentFile=/ruta/al/repositorio/backend/.env
Restart=on-failure
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=AtirmaApp-0.1

[Install]
WantedBy=multi-user.target
```

Código A.2: Archivo de configuración para el servicio `systemd` del backend

- Recargar el servicio de systemd para aplicar los cambios:

```
sudo systemctl daemon-reload
```

- Iniciar el servicio *backend*:

```
sudo systemctl start backend.service
```

- Habilitar la opción de inicio automático del servicio en el arranque del sistema:

```
sudo systemctl enable backend.service
```

A.3. Despliegue de la base de datos

En el repositorio del proyecto se incluye un archivo modelo que define la estructura de la base de datos, denominado `default_database.sql`. Para este proyecto se utilizará **MariaDB** como sistema de gestión de bases de datos. A continuación, se detallan los pasos necesarios para su instalación y configuración[20]:

1. Instalación de MariaDB

En primer lugar, es necesario actualizar el sistema y posteriormente instalar el paquete `mariadb-server`. Para ello, se ejecutan los siguientes comandos en la terminal:

```
sudo apt-get update
sudo apt-get install mariadb-server
```

2. Configuración inicial de seguridad

Se recomienda ejecutar el script de configuración de seguridad que ofrece MariaDB tras la instalación:

```
sudo mysql_secure_installation
```

Este script permite establecer una contraseña para el usuario root, eliminar usuarios anónimos, deshabilitar el acceso remoto del usuario root y eliminar la base de datos de prueba, entre otras opciones de seguridad.

3. Creación de un usuario y asignación de privilegios

Una vez instalado y asegurado MariaDB, se debe crear un usuario para el acceso a la base de datos. Esto puede hacerse accediendo al cliente de MariaDB con privilegios administrativos y ejecutando las siguientes sentencias SQL:

```sql
CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'mypassword';
GRANT ALL PRIVILEGES ON mydatabase.* TO 'myuser'@'localhost';
FLUSH PRIVILEGES;
```

Código A.3: Creación de un usuario y asignación de privilegios

4. Importación del archivo `.sql`

Por último, se procede a importar el archivo `default_database.sql` para crear las tablas y registros iniciales definidos en el modelo. Esto se realiza con el siguiente comando:

```
mysql -u nombre_usuario -p nombre_base_datos < /ruta/al/repositorio/default_database.sql
```

Código A.4: Importación del archivo .sql

Tras introducir la contraseña del usuario correspondiente, el contenido del archivo será cargado en la base de datos especificada.

Glosario

backend Parte de una aplicación web que se ejecuta en el servidor. Normalmente es usado para proveer servicios al frontend. [1].. 2

base de datos Una base de datos es una recopilación organizada de información o datos estructurados, que normalmente se almacena de forma electrónica en un sistema informático [4]. . iv, v, 3, 5, 6, 8, 12

CSV Los archivos CSV (del inglés comma-separated values) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas [**csv**].. 2, 3, 7, 13

fichero descriptivo Un fichero descriptivo es un archivo en formato JSON que describe la estructura de un vehículo indicando sus propiedades descriptivas, la visibilidad de las mismas y su uso.. 3, 15

framework Conjunto de herramientas que proporciona una base para desarrollar aplicaciones de una manera más organizada, robusta y escalable [2]. iv, 1, 2, 8

frontend Parte de una aplicación web que se ejecuta en el navegador del cliente, suele contener una interfaz de usuario [1].. 2

ID Identificador. 13

infraestructura web Una infraestructura web expone todas las partes necesarias para que una página web sea visible y funcional. Entre las partes de una infraestructura existen bases de datos, backend, frontend, servidor de alojamiento... . 1

intervalo Un intervalo en el ámbito de este TFT es una agrupación de registros en un periodo de tiempo. Por ejemplo, si el intervalo es de 60 segundos, se recuperará un único registro por minuto. . vii, 38, 42, 44

JSON JSON (JavaScript Object Notation) es un formato basado en texto para almacenar e intercambiar datos de una manera que es legible por humanos y analizable por máquina. Como resultado, JSON es relativamente fácil de aprender y de solucionar problemas. [5].. 15

misión Una misión en el ámbito de este TFT es una agrupación de registros de uno o varios vehículos en un espacio temporal. Dividiendose las misiones, en dos tipos, en directo (aquellas que están pendientes de recibir paquetes de telemetría) y en diferido o históricas (han estado en directo o cargadas por un archivo CSV). . 2

paquete de telemetría Se define a un paquete de telemetría a una petición post que realiza el vehículo al servidor con los datos sensoriales, por ejemplo, la latitud, longitud, velocidad del viento, etcétera. . 2, 3, 15

propiedad descriptiva Se define una propiedad descriptiva de un vehículo como cualquier campo de información telemétrica que posea, ya sea latitud, velocidad del viento, orientación del vehículo, etcétera. . 3, 5, 6

registro Un registro en el ámbito del presente TFT se define como una agrupación de datos o propiedades descriptivas en un preciso instante de la misión.. 7, 13

TFT Trabajo de Fin de Título. 3, 8

timestamp La traducción literal de timestamp es ”marca de tiempo”. Como bien se aclara en dicha traducción, en informática se utiliza timestamp para referirse a un instante específico en el tiempo.. 13

vehículo Un vehículo en el ámbito de este Trabajo de Fin de Título se refiere a la representación virtual de cualquier vehículo real o simulado que disponga la División de Robótica y Oceanografía Computacional del Instituto Universitario SIANI. . 2

waypoint Un waypoint o punto de paso es una ubicación en el mapa representada mediante coordenadas, que indicará el paso o siguiente punto a seguir de un vehículo. . 2, 3

Bibliografía

[1] *¿Qué es Frontend?* Artículo introductorio sobre desarrollo frontend en Medium. 2024. URL: https://medium.com/@hanekcud/what-is-frontend-4816aedb1f50.

[2] *¿Qué es un framework en programación y para qué sirve?* Explicación de un Framework. 2025. URL: https://www.arsys.es/blog/que-es-un-framework-en-programacion-y-para-que-sirve (visitado 21-07-2025).

[3] *¿Qué es una API?* Página informativa de Amazon sobre el significado de una API. 2025. URL: https://aws.amazon.com/es/what-is/api/ (visitado 20-07-2025).

[4] *¿Que es una base de datos?* Página web de oracle explicando las bases de datos. 2025. URL: https://www.oracle.com/es/database/what-is-database/ (visitado 22-07-2025).

[5] *¿Que es una JSON?* Página web de oracle explicando el formato JSON. 2025. URL: https://www.oracle.com/es/database/what-is-json/ (visitado 22-07-2025).

[6] *A-Tirma WebWare v.2: Infraestructura web para el seguimiento, monitorización, control y análisis de misiones para vehículos autónomos marinos.* Memoria de la anterior versión del proyecto A-Tirma WebWare. 2024. URL: https://accedacris.ulpgc.es/handle/10553/132662.

[7] *axios - Librería de Node.js.* Página web de npm para descargar axios. 2025. URL: https://www.npmjs.com/package/axios.

[8] *Backend Development - Geeks for Geeks.* Introducción al desarrollo backend. 2025. URL: https://www.geeksforgeeks.org/blogs/backend-development/.

[9] *bcrypt - Librería de Node.js.* Página web de npm para descargar bcrypt. 2025. URL: https://www.npmjs.com/package/bcrypt.

[10] *bcryptjs - Librería de Node.js.* Página web de npm para descargar bcryptjs. 2025. URL: https://www.npmjs.com/package/bcryptjs.

[11] *corejs - Librería de vue.js.* Página web de npm para descargar corejs. 2025. URL: https://www.npmjs.com/package/corejs.

[12] *cors - Intercambio de recursos de origen cruzado.* Explicación del mecanismo CORS. 2025. URL: https://developer.mozilla.org/es/docs/Web/HTTP/Guides/CORS.

[13] *CSS - Guía de referencia.* Referencia de CSS en MDN Web Docs. 2025. URL: https://developer.mozilla.org/es/docs/Web/CSS.

[14] *CSV - Valores Separados por Comas*. Definición de CSV en Wikipedia. 2024. URL: https://es.wikipedia.org/wiki/Valores_separados_por_comas.

[15] *DOM - Document Object Model*. Descripción del modelo de objetos del documento en Wikipedia. 2025. URL: https://es.wikipedia.org/wiki/Document_Object_Model.

[16] *dompurify - Documentación*. Página web de npm para descargar dompurify. 2025. URL: https://www.npmjs.com/package/dompurify.

[17] *Express - Framework de Node.js*. Página oficial de Express. 2025. URL: https://expressjs.com/es/.

[18] *fs - Módulo del sistema de archivos de Node.js*. Documentación oficial del módulo fs. 2025. URL: https://nodejs.org/api/fs.html (visitado 20-07-2025).

[19] *Gestor de estados para vuejs- Pinia*. Sitio oficial de Pinia. 2025. URL: https://pinia.vuejs.org/.

[20] *How to install MariaDB on Ubuntu 22.04*. Tutorial para instalar MariaDB en Ubuntu. 2025. URL: https://www.swhosting.com/en/comunidad/manual/how-to-install-mariadb-on-ubuntu-2204 (visitado 23-07-2025).

[21] *HTML - Documentación oficial*. Referencia de HTML en MDN Web Docs. 2025. URL: https://developer.mozilla.org/es/docs/Web/HTML.

[22] *JavaScript - MDN Web Docs*. Referencia oficial de JavaScript en MDN. 2025. URL: https://developer.mozilla.org/es/docs/Web/JavaScript.

[23] *jsonwebtoken - Librería de Node.js*. Página web de npm para descargar jsonwebtoken. 2025. URL: https://www.npmjs.com/package/jsonwebtoken.

[24] *La Fuerza del Viento*. Página web sobre la fuerza del viento. 2025. URL: https://lainakai.com/la-fuerza-del-viento/ (visitado 20-06-2025).

[25] *leaflet-textpath*. Plugin para modificar polilineas de leaflet con texto. 2025. URL: https://github.com/makinacorpus/Leaflet.TextPath.

[26] *Lighttpd Web Server*. Sitio oficial del servidor Lighttpd. 2025. URL: https://www.lighttpd.net/.

[27] *MariaDB*. Web oficial de MariaDB. 2025. URL: https://mariadb.org/.

[28] *Material Design Icons*. Página web con el listado de iconos disponibles. 2025. URL: https://pictogrammers.com/library/mdi/.

[29] *Multer - Librería Node.js*. Página web para descargar multer. 2025. URL: https://www.npmjs.com/package/multer (visitado 20-07-2025).

[30] *MySQL*. Web oficial de MySQL. 2025. URL: https://www.mysql.com/.

[31] *mysql2 - Librería de Node.js*. Página web de npm para descargar mysql2. 2025. URL: https://www.npmjs.com/package/mysql2.

[32] *Node.js Server Framework*. Sitio oficial de Node.js. 2025. URL: https://nodejs.org/en.

[33] *NPM*. Sitio oficial de NPM. 2025. URL: https://www.npmjs.com/.

[34] *path - Librería de Node.js*. Página web de npm para descargar path. 2025. URL: https://www.npmjs.com/package/path.

[35] *phpMyAdmin*. Sitio oficial de phpMyAdmin. 2025. URL: https://www.phpmyadmin.net/ (visitado 20-07-2025).

[36] *Qué es y cómo el gestor de estado Pinia en Vue.js*. Página informativa sobre Pinia. 2025. URL: https://www.luisllamas.es/vuejs-gestion-estado-con-pinia/ (visitado 23-07-2025).

[37] *Qué son y cómo usar las Props en Vue.js*. Página informativa sobre las Props en Vue.js. 2025. URL: https://www.luisllamas.es/vuejs-que-son-las-props/ (visitado 23-07-2025).

[38] *Servidor Web: Lighttpd, Thttpd. Características, instalación, configuración y comprobación de un servidor web*. PDF que explica el proceso de instalación de Lighttpd. 2025. URL: https://clementecervantesbustos.wordpress.com/wp-content/uploads/2019/03/servidor-web.pdf (visitado 20-07-2025).

[39] *SQL - Lenguaje de Consulta Estructurado*. Definición de SQL en Wikipedia. 2025. URL: https://es.wikipedia.org/wiki/SQL.

[40] *Ventajas de usar Vuejs*. Consulta realizada a ChatGPT. 2025. URL: https://chatgpt.com/ (visitado 21-07-2025).

[41] *Ventajas de usar XAMPP en una aplicación web*. Consulta realizada a ChatGPT. 2025. URL: https://chatgpt.com/ (visitado 21-07-2025).

[42] *Vue Router - The official Router for Vue.js*. Documentación del enrutador oficial de vue. 2025. URL: https://router.vuejs.org/.

[43] *Vue-leaflet Rotated Marker Plugin*. Plugin para rotar marcadores en mapas de Vue Leaflet. 2024. URL: https://github.com/littleimagine/vue-leaflet-rotate-marker.

[44] *Vue.js Web Framework*. Sitio oficial de Vue.js. 2014. URL: https://vuejs.org/.

[45] *Vue2 Leaflet Rotated Marker Plugin*. Complemento para rotación de marcadores en Vue2 Leaflet. 2024. URL: https://github.com/littleimagine/vue-leaflet-rotate-marker.

[46] *Vue2-Leaflet*. Documentación de la librería Vue2-Leaflet. 2020. URL: https://vue2-leaflet.netlify.app/.

[47] *vuejs - Wikipedia*. Descripción del vuejs en Wikipedia. 2025. URL: https://es.wikipedia.org/wiki/Vue.js (visitado 21-07-2025).

[48] *Vuetify UI Library*. Sitio oficial de Vuetify. 2025. URL: https://vuetifyjs.com/en/ (visitado 20-07-2025).

[49] *VueUse - Collection of Vue Composition Utilities*. Colección de utilidades esenciales de vue. 2025. URL: https://vueuse.org/.

[50] *XAMPP*. Página oficial del entorno XAMPP. 2024. URL: https://www.apachefriends.org/es/index.html.