

## 

[**1 Análisis del problema 3**](#_heading=h.2mfor2zhjqwn)

[1.1 Introducción 3](#_heading=h.tp65nozh0si)

[1.2 Objetivos 3](#_heading=h.x7mtuvsi3t5k)

[1.3 Funciones y rendimientos deseados 4](#_heading=h.pukqory0a62z)

[1.4 Planteamiento y evaluación de diversas soluciones 4](#_heading=h.m333y8fh4zd7)

[1.5 Justificación de la solución elegida 5](#_heading=h.4lod344vv6x3)

[1.6 Modelado de la solución 5](#_heading=h.4k5r4gc4kd14)

[1.6.1 Recursos humanos 5](#_heading=h.qrt2xb4tuxrk)

[1.6.2 Recursos hardware 6](#_heading=h.dh8ot0um0p5w)

[1.6.3 Recursos software 6](#_heading=h.8in64yw9n7eq)

[1.7 Planificación temporal 6](#_heading=h.3ipxi5orphmm)

[**2 Diseño e implementación del proyecto 7**](#_heading=h.lc3qxoapbwm6)

[2.1 Diagrama E/R 8](#_heading=h.mbpd40xabhmu)

[2.2 Gestión de seguridad 8](#_heading=h.6iekrx8m222v)

[2.3 Gestión de errores y logs 9](#_heading=h.4bqafs8diy88)

[**3. Fase de pruebas 10**](#_heading=h.iqwtdmazdqw9)

[**4. Documentación de la aplicación 11**](#_heading=h.v7pt7c6ckc8)

[4.1 Introducción a la aplicación (Getting Started) 11](#_heading=h.sb8i8xxwfkwf)

[4.2 Manual de Instalación 11](#_heading=h.ewn34bxn0u1y)

[4.3 Manual de usuario 11](#_heading=h.cl2fgzop27sm)

[4.4 Manual de administración 12](#_heading=h.rjnkl8y3m3qj)

[**5. Conclusiones finales 13**](#_heading=h.ey5l6sa0wnsk)

[5.1 Conclusiones 13](#_heading=h.jvdok58utbjc)

[5.2 Dificultades 13](#_heading=h.qljx292z50cn)

[**6. Bibliografía 14**](#_heading=h.6ssoxbfik46a)

[**7. Licencia 15**](#_heading=h.qeuw5112r5kb)

## 

**Documentación Técnica de RenovAuto**

## **1 Análisis del problema**

### **1.1 Introducción**

**RenovAuto** es una plataforma web moderna destinada a facilitar la compra y venta de coches. Desarrollada con tecnologías como **Symfony**, **Tailwind CSS**, **MySQL** y desplegada usando **Docker** y **Railway**, esta aplicación permite a los usuarios interactuar en un entorno intuitivo, eficiente y atractivo, uniendo así el mundo del desarrollo web con la pasión por los automóviles.

### **1.2 Objetivos**

**Objetivo principal:** Diseñar e implementar una plataforma web que permita a los usuarios consultar, publicar y administrar anuncios de vehículos, con un enfoque en usabilidad y despliegue eficiente.

**Objetivos específicos:**

* Consolidar conocimientos en frameworks y tecnologías web modernas (Symfony, Tailwind, Docker).
* Ofrecer un catálogo interactivo de vehículos con filtros y búsquedas avanzadas.
* Garantizar un diseño responsive adaptable a todos los dispositivos.
* Asegurar la escalabilidad y portabilidad del sistema mediante contenedores Docker.
* Publicar la plataforma en producción de forma automatizada usando Railway.

### **1.3 Funciones y rendimientos deseados**

* **Visualización de coches** con detalles como imagen, marca, modelo y precio.
* **Sistema de filtrado** por tipo de vehículo, marca y rango de precios.
* **Página individual por vehículo**, con toda su información.
* **Gestión de usuarios y favoritos**, incluyendo edición de perfil.
* **Mensajería entre usuarios** (sistema de chats).
* **Administración de la plataforma**, incluyendo control de anuncios, usuarios y estadísticas.
* **Despliegue local y en la nube** sin fricciones, gracias a Docker y Railway.

### **1.4 Planteamiento y evaluación de diversas soluciones**

Se consideraron diferentes tecnologías antes de definir la arquitectura final:

* **Backend**: Symfony fue elegido por su estructura robusta, facilidad de ampliación y comunidad activa.
* **Frontend**: Tailwind CSS ofrece personalización y rapidez en el diseño responsive.
* **Base de datos**: MySQL, ampliamente soportado y fácil de gestionar mediante PhpMyAdmin.
* **Contenedorización**: Docker simplifica la creación y replicación del entorno de desarrollo.
* **Despliegue**: Railway fue elegido por su integración directa con GitHub y compatibilidad con Docker.
* **React.js:** Framework elegido para desarrollar la interfaz de usuario debido a su arquitectura basada en componentes reutilizables, su ecosistema robusto y su integración sencilla con sistemas RESTful.
* **React Router Dom:** Para la navegación entre páginas del sitio web, sin recargar la aplicación y con rutas dinámicas.
* **Context API:** Para la gestión del estado global de la aplicación, como los datos del usuario, los favoritos o los anuncios visibles.
* **Framer Motion:** Utilizado para añadir animaciones suaves e interactivas entre vistas, mejorando la experiencia de usuario.
* **Chart.js:** Para la visualización de estadísticas y gráficos dinámicos en el panel de administración.

### **1.5 Justificación de la solución elegida**

La arquitectura basada en Symfony + Tailwind + Docker permite una separación clara de responsabilidades, escalabilidad y facilidad de mantenimiento. Railway complementa esta solución al permitir automatizar el despliegue continuo de cada servicio.

### **1.6 Modelado de la solución**

#### **1.6.1 Recursos humanos**

* Un desarrollador principal responsable de todo el ciclo de vida del proyecto.

#### **1.6.2 Recursos hardware**

* Equipo de desarrollo local (PC con Docker Desktop).
* Servidor cloud proporcionado por Railway.

#### **1.6.3 Recursos software**

* **Symfony** para lógica de negocio y API.
* **Tailwind CSS** para estilos frontend.
* **MySQL** como sistema gestor de bases de datos.
* **PhpMyAdmin** como herramienta de administración de bases de datos.
* **Docker** para entornos locales.
* **Railway** para entorno en producción.

### **1.7 Planificación temporal**

| **Fase** | **Duración estimada** |
| --- | --- |
| Análisis y diseño | 2 semanas |
| Desarrollo del sistema | 7 semanas |
| Pruebas y validación | 3 semanas |
| Documentación y despliegue | 3 semana |

## 

## **2 Diseño e implementación del proyecto**

El desarrollo del frontend se basó en React, permitiendo construir una SPA (Single Page Application) altamente interactiva:

* **Estructura basada en componentes:** Cada parte de la interfaz (barra de navegación, lista de coches, detalles, formularios, chat, etc.) fue encapsulada como componente reutilizable.
* **React Router:** Para gestionar rutas internas como /login, /submit\_car, /details\_car/:id, /perfil, /stadistics, entre otras.
* **Hooks personalizados:** Se usaron useState, useEffect, useContext y useRef para manejar lógica, peticiones y estado global/local.
* **Autenticación JWT:** El frontend se comunica con la API Symfony usando JWT para asegurar las rutas y controlar accesos según roles (usuario o administrador).
* **Responsive con Tailwind:** Gracias a Tailwind, la maquetación de React se adaptó fácilmente a todos los tamaños de pantalla.
* **Integración con Symfony API:** React realiza peticiones HTTP usando fetch para obtener datos de vehículos, usuarios, y enviar acciones como publicar, editar, filtrar o eliminar anuncios.

### **2.1 Diagrama E/R**

### **2.2 Gestión de seguridad**

Implementación de JWT en Symfony con Lexi JWT Authentication Bundle.

* El token se guarda en localStorage y se usa para autenticar peticiones protegidas.
* Rutas protegidas por roles (ROLE\_USER, ROLE\_ADMIN,ROLE\_BANNED).
* Encriptación de contraseñas con bcrypt.
* Protección contra XSS, CSRF y SQL Injection mediante buenas prácticas y herramientas del framework.

### **2.3 Gestión de errores y logs**

* Redirección a página de error personalizada ante rutas inválidas.
* Sistema de notificaciones con *toasts* (éxito, error, información).
* Logs en consola del navegador para depuración.
* React gestiona los errores para evitar roturas no controladas.

### 

## **3. Fase de pruebas**

* **Pruebas unitarias**: Validación de funciones críticas en el backend con las rutas twig y servidor Symfony.
* **Pruebas funcionales**: Navegación completa por la plataforma, testeo de formularios y validaciones e inserciones y eliminación de datos.
* **Pruebas de usabilidad**: Verificación de la experiencia en múltiples navegadores y tamaños de pantalla y resolución.
* **Pruebas de integración**: Comunicación entre frontend, backend y base de datos.

## 

## **4. Documentación de la aplicación**

### **4.1 Introducción a la aplicación (Getting Started)**

Para iniciar el proyecto, es necesario clonar el repositorio y seguir las instrucciones detalladas del README. La plataforma está preparada para ejecutarse tanto en local como en producción con configuraciones predefinidas.

### 

### **4.2 Manual de Instalación**

🛠️ *Ver README original del repositorio:* [Repositorio GitHub](https://github.com/FranSalmeron/ProyectoDaw)

### **4.3 Manual de usuario**

Los usuarios registrados pueden:

* Navegar por el catálogo de vehículos con filtros personalizados.
* Consultar detalles completos de cada coche.
* Publicar anuncios con imágenes, descripción, precio y más.
* Editar o eliminar sus propios anuncios.
* Añadir coches a su lista de favoritos.
* Iniciar chats con vendedores/compradores.
* Editar su perfil (nombre, email, contraseña).
* Los usuarios baneados pueden apelar
* **Navegar sin recarga entre páginas** gracias al enrutamiento de React Router.
* **Ver animaciones de carga y transiciones** al moverse entre componentes (gracias a Framer Motion).

### **4.4 Manual de administración**

Los administradores tienen privilegios especiales para:

* Modificar, aprobar o eliminar anuncios sospechosos.
* Generar reportes de actividad y comportamiento en la plataforma.
* **Acceder a un dashboard en React con gráficos estadísticos** (usuarios activos, coches publicados, anuncios eliminados, etc.) construido con **Chart.js**.
* **Revisar en tiempo real los reportes o chats sospechosos**, donde cada hilo de conversación está disponible desde una tabla dinámica.
* **Filtrar anuncios por estado (subido, comprado, bloqueado)** desde un panel interactivo.
* **Gestionar usuarios desde un componente con paginación y búsqueda**, todo sin recarga de página.

## 

## **5. Conclusiones finales**

### **5.1 Conclusiones**

El proyecto RenovAuto ha logrado cumplir con los objetivos definidos al inicio:

* Se ha desarrollado una plataforma funcional, intuitiva y moderna.
* Se utilizaron tecnologías actuales que aseguran mantenibilidad y escalabilidad.
* El despliegue en producción mediante Railway ha permitido una entrega continua eficiente.
* Se deja la puerta abierta a futuras mejoras como pasarelas de pago, más opciones de filtrado o integración con IA para sugerencias inteligentes de vehículos.

### **5.2 Dificultades**

* Conexión de servicios y puertos en producción.
* Aprendizaje de las nuevas tecnologías y módulos como symfony, react-router, railway y más.
* Correcta configuración de jwt y seguridad en la página, certificados.
* Administrar y comprobar todo sin que falte nada debido a la magnitud del proyecto
* Aprendizaje de contextos y usarlos correctamente

### 

## **6. Bibliografía**

* Symfony:<https://symfony.com/>
* Tailwind CSS:<https://tailwindcss.com/>
* Docker:<https://www.docker.com/>
* Railway:<https://railway.app/>
* MySQL:<https://www.mysql.com/>
* PhpMyAdmin:<https://www.phpmyadmin.net/>
* Framer-motion: <https://motion.dev/docs/react-quick-start>
* Chart.js: <https://www.chartjs.org/>

## 

## **7. Licencia**

Este proyecto está licenciado bajo la GPL v3.0. Esto implica que cualquier redistribución del software debe mantenerse bajo la misma licencia. Se permite el uso, modificación y distribución, siempre que se mantenga la autoría y términos de la GPL.