



GOBIERNO DEL  
**ESTADO DE MÉXICO**



## UNIVERSIDAD MEXIQUENSE DEL BICENTENARIO

UNIDAD DE ESTUDIOS SUPERIORES  
SAN JOSÉ DEL RINCÓN

# SISTEMA PARA EL CONTROL VEHICULAR PARA LA EMPRESA EN TELECOMUNICACIONES ATLACOMULCO

## PROYECTO DE RESIDENCIA

CARRERA:  
**INGENIERÍA EN SISTEMAS COMPUTACIONALES**

PRESENTA:  
**IMANOL CRUZ CLEMENTE**

ASESOR:  
**GERARDO MORENO DE JESÚS**

*Noviembre de 2025.*



GOBIERNO DEL  
**ESTADO DE MÉXICO**



## UNIVERSIDAD MEXIQUENSE DEL BICENTENARIO

UNIDAD DE ESTUDIOS SUPERIORES  
SAN JOSÉ DEL RINCÓN

# SISTEMA PARA EL CONTROL VEHICULAR PARA LA EMPRESA EN TELECOMUNICACIONES ATLACOMULCO

## PROYECTO DE RESIDENCIA

CARRERA:  
**INGENIERÍA EN SISTEMAS COMPUTACIONALES**

PRESENTA:  
**IMANOL CRUZ CLEMENTE**

ASESOR:  
**GERARDO MORENO DE JESÚS**

*Noviembre de 2025.*

## ÍNDICE DE CONTENIDO

<b>INTRODUCCIÓN.....</b>	<b>XIV</b>
<b>PLANTEAMIENTO DEL PROBLEMA .....</b>	<b>XVI</b>
<b>JUSTIFICACIÓN .....</b>	<b>XVIII</b>
<b>OBJETIVOS .....</b>	<b>XIX</b>
GENERAL.....	XIX
ESPECÍFICOS.....	XIX
<b>CAPÍTULO I: ANTECEDENTES DE LA EMPRESA .....</b>	<b>20</b>
1.1 CONCEPTO .....	21
1.2 MISIÓN .....	21
1.3 VISIÓN.....	21
1.4 VALORES.....	21
1.5 CROQUIS DE UBICACIÓN.....	22
1.6 ORGANIGRAMA EMPRESARIAL .....	23
<b>CAPÍTULO II: MARCO TEÓRICO.....</b>	<b>24</b>
2.1 METODOLOGÍA.....	25
2.1.1 CASCADA (WATERFALL).....	25
2.2 LENGUAJES DE PROGRAMACIÓN .....	26
2.2.1 PYTHON.....	26
2.2.2 JAVASCRIPT (ES6+).....	26
2.3 DESARROLLO BACKEND (SERVIDOR).....	26
2.3.1 FLASK.....	27
2.3.1.1 FLASK-SQLALCHEMY .....	27
2.3.1.2 WERKZEUG .....	27
2.3.1.3 FLASK-MAIL.....	28
2.3.1.4 FLASK-CORS.....	28
2.3.1.5 PYTHON-DOTENV .....	28
2.3.1.6 BLUEPRINTS (ESTRUCTURA MODULAR EN FLASK).....	29
2.3.1.7 MIDDLEWARE .....	29

2.3.2 SERIALIZACIÓN Y DESERIALIZACIÓN DE DATOS (JSON ↔ OBJETOS PYTHON).....	29
2.3.3 MANEJO DE ERRORES Y CÓDIGOS DE ESTADO HTTP (200, 404, 500, ETC.)	29
2.3.4 AUTENTICACIÓN Y AUTORIZACIÓN (CONCEPTO).....	30
2.4 DESARROLLO FRONTEND (INTERFAZ DE USUARIO) .....	30
2.4.1 VITE .....	30
2.4.2 REACT .....	30
2.4.2.1 REACT ROUTER DOM.....	31
2.4.2.2 CSS MODULES.....	31
2.4.3 BOOTSTRAP.....	31
2.4.4 FONT AWESOME .....	32
2.4.5 FETCH API.....	32
2.4.6 COMPONENTES FUNCIONALES .....	32
2.4.6.1 PROPS Y STATE .....	32
2.4.6.2 HOOKS (USESTATE, USEEFFECT, ETC.).....	33
2.4.6.3 RENDERIZADO CONDICIONAL .....	33
2.4.6.4 RESPONSIVIDAD (MEDIA QUERIES, GRID/FLEXBOX) .....	33
2.4.6.5 SPA (SINGLE PAGE APPLICATION) .....	33
2.5 BASE DE DATOS .....	34
2.5.1 MYSQL .....	34
2.5.1.1 TABLA (TABLE).....	34
2.5.1.2 LLAVES PRIMARIAS (PRIMARY KEYS) .....	35
2.5.1.3 LLAVES FORÁNEAS (FOREIGN KEYS) .....	35
2.5.1.4 ÍNDICES (INDEXES).....	35
2.5.1.5 INTEGRIDAD REFERENCIAL .....	35
2.5.1.6 NORMALIZACIÓN (1FN, 2FN, 3FN) .....	36
2.5.1.7 CONSULTAS SQL (SELECT, INSERT, UPDATE, DELETE → CRUD) .....	36
2.5.1.8 RELACIONES (1:1, 1:N, N:N) .....	36
2.5.2 PYMYSQL / MYSQLCLIENT.....	37
2.6 ENTORNO DE DESARROLLO Y HERRAMIENTAS .....	37
2.6.1 NODE.JS Y NPM.....	37
2.6.2 PYTHON VENV Y PIP.....	38

2.6.3	MYSQL WORKBENCH .....	38
2.6.4	POSTMAN, INSOMNIA, PYTHON REQUESTS .....	38
2.6.5	GIT Y GITHUB.....	39
2.6.6	VARIABLES DE ENTORNO .....	39
2.7	ARQUITECTURA Y CONCEPTOS CLAVE .....	39
2.7.1	ARQUITECTURA CLIENTE-SERVIDOR .....	40
2.7.2	API REST .....	40
2.7.3	ORM (OBJECT-RELATIONAL MAPPING).....	40
2.7.4	ENTIDAD-RELACIÓN (ERD).....	40
<b>CAPÍTULO III: DISEÑO DE LAS INTERFACES Y MODELOS RELACIONALES .....</b>	<b>41</b>	
3.1	DIAGRAMA ENTIDAD RELACIÓN .....	42
3.1.1	SISTEMA INICIAL .....	42
3.1.2	GESTIÓN DE MANTENIMIENTO.....	42
3.1.3	AUDITORÍA Y HISTORIAL DE CAMBIOS .....	43
3.2	DIAGRAMA DE CASOS DE USO .....	44
3.2.1	FUNCIONES DEL ADMINISTRADOR (ACCESO TOTAL).....	44
3.2.2	FUNCIONES DEL USUARIO/GERENTE (OPERACIÓN Y CONSULTA) .....	45
3.2.3	FLUJO CRÍTICO Y LÓGICA AUTOMÁTICA .....	45
3.2.4	CONSULTAS Y REPORTES .....	45
3.3	MAQUETADO.....	46
3.4	CONTRASTE DE COLORES .....	67
<b>CAPÍTULO IV: DESARROLLO DEL SISTEMA .....</b>	<b>69</b>	
4.1	REQUISITOS PREVIOS .....	70
4.2	INSTALACIÓN DE MYSQL .....	71
4.3	CREACIÓN DE UN PROYECTO FRONTEND REACT .....	73
4.3.1	CREACIÓN DEL PROYECTO REACT CON VITE .....	73
4.3.2	ACCESO A LA CARPETA DEL FRONTEND .....	74
4.3.3	INSTALACIÓN DE DEPENDENCIAS.....	74
4.3.4	EJECUCIÓN DEL SERVIDOR DE DESARROLLO.....	74
4.4	CREACIÓN DEL PROYECTO BACKEND .....	75
4.4.1	CREACIÓN DE LA CARPETA DEL BACKEND .....	76

4.4.2	ACCESO A LA CARPETA DEL BACKEND .....	76
4.4.3	CREACIÓN DE UN ENTORNO VIRTUAL EN PYTHON .....	76
4.5	CONFIGURACIÓN DE LA BASE DE DATOS CON MYSQL Y CONEXIÓN AL BACKEND .....	77
4.5.1	CREACIÓN DE LA BASE DE DATOS EN MYSQL WORKBENCH .....	77
4.5.2	CREACIÓN DE LAS TABLAS MYSQL.....	77
4.5.3	CONEXIÓN DEL BACKEND CON LA BASE DE DATOS MEDIANTE SQLALCHEMY .....	78
4.5.4	DEFINICIÓN DE MODELOS CON SQLALCHEMY .....	78
4.6	IMPLEMENTACIÓN DEL SISTEMA DE AUTENTICACIÓN EN EL BACKEND..	79
4.6.1	RECUPERACIÓN DE CONTRASEÑA .....	80
4.6.2	SEGURIDAD EN EL MANEJO DE SESIONES Y TOKENS .....	81
4.6.2.1	AUTENTICACIÓN SEGURA.....	81
4.6.2.2	PREVENCIÓN DE SQL INJECTION .....	81
4.6.2.3	GESTIÓN DE TOKENS DE RECUPERACIÓN .....	82
4.6.2.4	SEGURIDAD EN EL FRONTEND .....	82
4.6.2.5	CONSIDERACIONES ADICIONALES.....	82
4.7	MANEJO DE SESIÓN EN EL FRONTEND E INTEGRACIÓN CON EL BACKEND	82
4.7.1	CONSUMO DEL ENDPOINT /LOGIN .....	82
4.7.2	PERSISTENCIA DE LA SESIÓN.....	83
4.7.3	REDIRECCIÓN POR ROL .....	83
4.7.4	CIERRE AUTOMÁTICO POR INACTIVIDAD.....	83
4.7.5	MANEJO DE ERRORES Y ESTADOS.....	84
4.7.6	INTEGRACIÓN PARCIAL DEL LOGIN EN APP.JSX .....	84
4.8	COMPONENTE HEADER.....	85
4.8.1	DEFINICIÓN E IMPORTACIONES.....	85
4.8.2	CONTROL DE VISUALIZACIÓN RESPONSIVA .....	86
4.9	COMPONENTE SIDEBAR.....	87
4.9.1	DEFINICIÓN E IMPORTACIONES.....	87
4.9.2	ESTADOS Y CONTROL DE RESPONSIVIDAD .....	87
4.9.3	ESTRUCTURA DEL JSX .....	88
4.10	INTEGRACIÓN DEL HEADER Y SIDEBAR EN APP.JSX.....	88

4.10.1	IMPORTACIONES Y ESTADOS PRINCIPALES .....	88
4.10.2	RENDERIZADO CONDICIONAL SEGÚN SESIÓN .....	89
4.11	GESTIÓN DE UNIDADES (UNIDADES.JSX) .....	89
4.11.1	IMPORTACIONES Y ESTADOS PRINCIPALES .....	90
4.11.2	OBTENCIÓN Y PAGINACIÓN DE UNIDADES .....	90
4.12	MÓDULO DE GARANTÍAS .....	91
4.12.1	REQUISITOS Y RESTRICCIONES .....	91
4.12.2	DEFINICIÓN E IMPORTACIONES FRONTEND .....	91
4.12.2.1	ESTADOS Y EFECTOS PRINCIPALES.....	91
4.12.2.2	VERIFICACIÓN Y RENOVACIÓN DESDE LA INTERFAZ .....	92
4.12.2.3	FUNCIÓN DE RENOVACIÓN Y ACTUALIZACIÓN DEL LISTADO .....	92
4.12.2.4	ESTRUCTURA DEL JSX .....	93
4.12.2.5	ESTILOS Y ADAPTABILIDAD .....	93
4.12.3	BACKEND DEL MÓDULO DE GARANTÍAS .....	93
4.12.3.1	MODELO DE DATOS.....	94
4.12.3.2	ENDPOINT PRINCIPAL DE GESTIÓN .....	94
4.12.3.3	VERIFICACIÓN DE VIGENCIA .....	94
4.12.3.4	PROCESO DE RENOVACIÓN .....	95
4.13	MÓDULO DE VERIFICACIONES VEHICULARES.....	95
4.13.1	FRONTEND: ESTRUCTURA Y FUNCIONES PRINCIPALES .....	96
4.13.1.1	OBTENCIÓN Y VISUALIZACIÓN DE VERIFICACIONES .....	96
4.13.1.2	REGISTRO Y VALIDACIÓN DE NUEVAS VERIFICACIONES .....	96
4.13.1.3	GESTIÓN DE ARCHIVOS Y RESETEO DEL FORMULARIO .....	97
4.13.2	BACKEND DEL MÓDULO DE VERIFICACIONES .....	97
4.13.2.1	GESTIÓN DE REGISTROS Y COMPROBANTES .....	98
4.13.2.2	CÁLCULO AUTOMÁTICO DE FECHAS .....	98
4.13.2.3	HISTORIAL Y RESETEO AUTOMATIZADO.....	99
4.13.2.4	CONSULTA DE DATOS .....	99
4.14	MÓDULO DE SOLICITUDES DEL CHOFER.....	100
4.14.1	DEFINICIÓN E IMPORTACIONES (FRONTEND).....	100
4.14.2	CARGA INICIAL DE DATOS.....	101
4.14.3	ENVÍO DE SOLICITUDES DE REPARACIÓN.....	101
4.14.4	REGISTRO DE FALLAS TRAS APROBACIÓN .....	102

4.14.5	RENDERIZADO CONDICIONAL Y ESTRUCTURA VISUAL .....	102
4.14.6	CREACIÓN DE SOLICITUDES DE REPARACIÓN (BACKEND).....	102
4.15	MÓDULO DE SOLICITUDES DEL ADMINISTRADOR (FRONTEND).....	103
4.15.1	CARGA Y RENDERIZADO DE SOLICITUDES .....	103
4.15.2	OBTENCIÓN DE SOLICITUDES POR CHOFER (BACKEND) .....	103
4.15.3	LISTADO GENERAL DE SOLICITUDES (ADMINISTRADOR) .....	104
4.15.4	APROBACIÓN O RECHAZO DE SOLICITUDES .....	104
4.15.5	REGISTRO DE FALLA MECÁNICA .....	105
4.16	MÓDULO DE PLACAS Y REEMPLACAMIENTOS .....	105
4.16.1	DEFINICIÓN E IMPORTACIONES (FRONTEND).....	105
4.16.2	OBTENCIÓN DE DATOS Y CARGA INICIAL .....	106
4.16.3	VERIFICACIÓN DE UNIDAD Y VALIDACIÓN DE VIGENCIA.....	106
4.16.4	REGISTRO, ACTUALIZACIÓN Y ELIMINACIÓN DE PLACAS.....	107
4.16.5	BACKEND DEL MÓDULO DE PLACAS.....	107
4.16.5.1	RUTAS PRINCIPALES.....	107
4.16.5.2	REGISTRO Y RENOVACIÓN DE PLACAS .....	108
4.16.5.3	TAREAS AUTOMÁTICAS Y ALERTAS .....	108
4.17	MÓDULO DE REFRENDO Y TENENCIA .....	109
4.17.1	CARGA INICIAL DE DATOS Y MANEJO DE ESTADOS (FRONTEND) .	109
4.17.1.1	VALIDACIÓN Y TIPO DE PAGO .....	109
4.17.1.2	REGISTRO DE PAGOS .....	110
4.17.1.3	VISUALIZACIÓN Y EDICIÓN.....	110
4.17.2	VERIFICACIÓN DE DISPONIBILIDAD DE UNIDAD (BACKEND) .....	111
4.17.2.1	FUNCIONES AUXILIARES DE ARCHIVOS .....	111
4.17.2.2	REGISTRO Y RESETEO DE PAGOS .....	112
4.17.2.3	ACTUALIZACIÓN DE PAGOS EXISTENTES .....	112
4.17.2.4	ENVÍO AUTOMÁTICO DE ALERTAS .....	113
<b>CAPÍTULO V: PRUEBAS Y RESULTADOS DEL SISTEMA</b>	.....	<b>114</b>
<b>CONCLUSIONES PRELIMINARES</b> .....		<b>CXXI</b>
<b>BIBLIOGRAFÍA</b> .....		<b>CXXIII</b>
REFERENCIAS ELECTRÓNICAS.....		CXXIII

REFERENCIAS BIBLIOGRÁFICAS.....	CXXV
<b>ANEXOS.....</b>	<b>CXXVI</b>
ALCANCE DEL SISTEMA.....	CXXVI
USUARIOS Y ROLES .....	CXXVI
DATOS CRÍTICOS .....	CXXVII

## ÍNDICE DE FIGURAS

Figura. 1 Ubicación geográfica de la empresa FIBRATEC .....	22
Figura. 2 Organigrama de la empresa.....	23
Figura. 3 Modelo en waterfall .....	25
Figura. 4 Modelo entidad relación .....	44
Figura. 5 Diagrama de caso de uso .....	46
Figura. 6 Inicio de sesión .....	47
Figura. 7 Dashboard de la aplicación .....	47
Figura. 8 Apartado de unidades vehiculares .....	48
Figura. 9 formulario de registrar automóvil .....	49
Figura. 10 Alerta de éxito en el registro.....	49
Figura. 11 Exportación de pdf .....	50
Figura. 12 Modal de ver más información .....	51
Figura. 13 Alerta de eliminación de vehículo .....	51
Figura. 14 Actualización de vehículo .....	52
Figura. 15 Alerta de actualización .....	53
Figura. 16 Apartado de mantenimientos.....	53
Figura. 17 Registro de un mantenimiento.....	54
Figura. 18 registro de mantenimiento exitoso.....	55
Figura. 19 Exportación pdf de mantenimientos .....	55
Figura. 20 Eliminación de mantenimientos .....	56
Figura. 21 Editar mantenimiento .....	57
Figura. 22 Apartado de verificaciones .....	57
Figura. 23 registro de una nueva verificación .....	58
Figura. 24 Eliminación de una verificación .....	59
Figura. 25 Actualización de verificaciones.....	59
Figura. 26 Apartado de garantías.....	60
Figura. 27 inserción de una nueva garantía .....	61

Figura. 28 Registro de garantías exitoso .....	61
Figura. 29 Modificación de las garantías .....	62
Figura. 30 Actualización exitosa de la garantía .....	63
Figura. 31 Eliminación de garantías .....	63
Figura. 32 Apartado de notificaciones .....	64
Figura. 33 Notificación correspondiente al mensaje .....	65
Figura. 34 Actualizar mantenimiento de la notificación.....	66
Figura. 35 Ver más mensajes .....	66
Figura. 36 Instalador de MySQL.....	71
Figura. 37 Tipo de instalación .....	72
Figura. 38 Instalación de dependencias adicionales .....	73
Figura. 39 Inicialización del servidor. ....	75
Figura. 40 Interfaz inicial del proyecto .....	75
Figura. 41 Inserción de tablas a la base de datos .....	78
Figura. 42 Definición de los modelos .....	79
Figura. 43 Endpoint /login .....	80
Figura. 44 Recuperación de contraseña.....	81
Figura. 45 Consumo del endpoint /login .....	83
Figura. 46 Persistencia de la sesión.....	83
Figura. 47 Redirección de roles .....	83
Figura. 48 Cierre automático .....	84
Figura. 49 Integración parcial del Login en App.jsx .....	85
Figura. 50 Definición del componente Header .....	85
Figura. 51 Control de visualización responsiva .....	86
Figura. 52 Estructura principal del Header y menú de usuario .....	86
Figura. 53 Definición del componente Sidebar .....	87
Figura. 54 Estados y control de responsividad .....	87
Figura. 55 Estructura del Sidebar con rutas y submenús .....	88
Figura. 57 Integración condicional de Header y Sidebar .....	89
Figura. 58 Estados principales para control de unidades .....	90

Figura. 59 Obtención y paginación de unidades .....	90
Figura. 60 Carpetas de guardado de documentos .....	91
Figura. 61 Definición e importaciones del componente Garantías .....	91
Figura. 62 Estados principales y carga inicial.....	92
Figura. 63 Verificación y renovación de garantía.....	92
Figura. 64 Función de renovación de garantía .....	92
Figura. 65 Estructura general del JSX.....	93
Figura. 66 Modelo de datos de garantías e historial .....	94
Figura. 67 Endpoint principal de garantías .....	94
Figura. 68 Verificación de vigencia de garantía.....	95
Figura. 69 Proceso de renovación de garantía y registro histórico .....	95
Figura. 70 Definición e importaciones del componente Verificaciones.jsx .....	96
Figura. 71 Consulta y renderizado de verificaciones en la interfaz .....	96
Figura. 72 Validación y registro de verificaciones en el formulario .....	97
Figura. 73 Reseteo de formulario y gestión de comprobantes PDF.....	97
Figura. 74 Endpoint para obtención de unidad y cálculo automático de engomado .....	98
Figura. 75 Calculo de la siguiente verificación .....	99
Figura. 76 Historial y reseteo automatizado .....	99
Figura. 77 Consulta de datos de la verificación.....	100
Figura. 78 Definición del componente e importaciones .....	100
Figura. 79 Carga inicial de catálogos y validación de solicitudes .....	101
Figura. 80 Envío de nueva solicitud de reparación .....	101
Figura. 81 Envío del formulario de falla mecánica .....	102
Figura. 82 Estructura principal del JSX .....	102
Figura. 83 Ruta para crear una nueva solicitud de falla.....	103
Figura. 84 Listado de solicitudes con opciones de gestión .....	103
Figura. 85 Ruta para obtener las solicitudes por chofer .....	104
Figura. 86 Ruta para listar todas las solicitudes .....	104
Figura. 87 Ruta para aprobar o rechazar solicitudes .....	105

Figura. 88 Ruta para registrar una falla mecánica .....	105
Figura. 89 Definición del componente y estados principales. ....	106
Figura. 90 Carga inicial de placas y unidades. ....	106
Figura. 91 Verificación de unidad y control de vigencia. ....	107
Figura. 92 Registro de nueva placa y manejo de archivos. ....	107
Figura. 93 Rutas principales del módulo de Placas. ....	108
Figura. 94 Registro y renovación de placas. ....	108
Figura. 95 Scheduler para envío automático de alertas. ....	109
Figura. 96 Carga inicial de datos y manejo de estados. ....	109
Figura. 97 4.17.2 Validación y tipo de pago .....	110
Figura. 98 Registro de pagos .....	110
Figura. 99 Visualización y edición .....	110
Figura. 100 4.17.2 Verificación de disponibilidad de unidad .....	111
Figura. 101 Funciones auxiliares de archivos de refrendo y tenencia .....	111
Figura. 102 Registro y reseteo de pagos de tenencia y refrendo .....	112
Figura. 103 Actualización de pagos existentes.....	112
Figura. 104 Envío de alertas automáticas de refrendo y tenencia .....	113

## INTRODUCCIÓN

“La gestión eficiente de flotas es clave para minimizar costos operativos y mejorar la sostenibilidad económica, mediante el control detallado de combustible, mantenimiento y capacitación de los conductores” (Casanova, Garza y Ortiz, 2012, p. 3). En este contexto, la empresa Fibratec enfrenta la necesidad de mejorar la administración de su parque vehicular, que actualmente carece de un sistema integral que centralice y automatice estas tareas.

Por ello, este proyecto tiene como objetivo desarrollar un sistema web que permita gestionar de manera integral los vehículos de Fibratec, facilitando el control de vencimientos, mantenimientos y notificaciones automáticas. Con esta solución se busca optimizar la gestión, reducir riesgos legales y garantizar la disponibilidad continua de las unidades para su operación.

El documento está organizado en los siguientes capítulos:

- **Capítulo 1 Generalidades de la empresa:** En este capítulo se presenta una descripción detallada de Fibratec, incluyendo su historia, misión, visión y estructura organizacional.
- **Capítulo 2 Marco teórico:** Se exploran las tecnologías disponibles para la administración vehicular, haciendo especial énfasis en sistemas web y la integración de APIs.
- **Capítulo 3 Diseño:** Este capítulo detalla la arquitectura del sistema web, presentando diagramas de casos de uso, diagramas de clases, modelo entidad-relación y la estructura de la base de datos. Se especifican también los requerimientos funcionales y no funcionales, así como la interfaz de usuario propuesta para asegurar una experiencia intuitiva.
- **Capítulo 4 Desarrollo:** Se documenta el proceso de construcción del sistema, describiendo las tecnologías, lenguajes de programación y herramientas utilizadas.

Se explica la integración de envío automatizado de alertas, y se describen las funcionalidades principales implementadas, tales como la gestión de información vehicular, generación de reportes y configuración de notificaciones.

- Capítulo 5 Pruebas: Finalmente, se presentan las pruebas realizadas para validar el correcto funcionamiento del sistema. Esto incluye pruebas funcionales para comprobar que cada módulo responde según lo esperado, pruebas de usabilidad para garantizar que el sistema sea amigable y fácil de usar.

## **PLANTEAMIENTO DEL PROBLEMA**

La gestión actual de la flota vehicular en Fibratec se apoya en un sistema manual que, aunque familiar, se ha vuelto obsoleto y representa un riesgo latente para la estabilidad operativa de la empresa. La dependencia en un mosaico de herramientas desconectadas, hojas de cálculo, agendas físicas y grupos de WhatsApp crea una frágil ilusión de control. En realidad, este método es altamente propenso a errores humanos, ya que la información crítica está fragmentada y no existe una fuente única de verdad, lo que impide una toma de decisiones ágil e informada.

Las consecuencias de esta falta de centralización son tangibles y costosas. Un simple descuido en el seguimiento de fechas críticas, como la renovación de una póliza de seguro o una verificación vehicular, desencadena una cadena de problemas: se inician con sanciones económicas, pueden escalar a la inmovilización de unidades clave y culminan en retrasos en el servicio al cliente, afectando directamente la reputación y la fiabilidad de Fibratec. Estos no son riesgos hipotéticos, sino incidentes que ya han generado gastos imprevistos y han puesto a prueba la capacidad de respuesta de la organización.

A su vez, esta metodología ha instaurado una cultura de mantenimiento reactiva, donde se actúa sobre las averías en lugar de prevenirlas. La ausencia de un calendario de servicio automatizado y centralizado lleva a que los mantenimientos preventivos se omitan o retrasen, lo que no solo incrementa drásticamente los costos de reparación, sino que también reduce la vida útil de los vehículos y, más importante aún, compromete la seguridad de los empleados. Cada vehículo que falla inesperadamente representa una interrupción en la operatividad y un riesgo que pudo haberse mitigado con una gestión proactiva.

Finalmente, la comunicación informal y desorganizada agrava la situación, creando un ambiente de incertidumbre administrativa donde la asignación de responsabilidades es ambigua y el seguimiento de tareas se vuelve ineficiente.

Este conjunto de factores ha atrapado a la empresa en un ciclo de desorden que genera gastos innecesarios y frena su potencial de crecimiento. Ante esta realidad, la pregunta es inevitable:

**¿Acaso la implementación de un SISTEMA PARA EL CONTROL VEHICULAR PARA LA EMPRESA EN TELECOMUNICACIONES ATLACOMULCO mejorara la gestión de vehículos en el departamento de control vehicular Fibratec?**

## **JUSTIFICACIÓN**

La implementación de este sistema de gestión vehicular no es solo una mejora tecnológica, es una decisión de negocio estratégica para Fibratec, justificada por su impacto directo en la mitigación de riesgos, la optimización de recursos y la modernización operativa. El proyecto busca transformar el paradigma reactivo actual de "apagar incendios", donde la empresa enfrenta problemas como multas y averías a medida que surgen. Mediante la automatización de alertas, el sistema actúa como un asistente de planificación y recordatorios que se anticipa a las fechas críticas; el vencimiento de un seguro o una verificación deja de ser un "ojalá no se nos olvide" para convertirse en una certeza gestionada con antelación, reduciendo drásticamente el riesgo de sanciones y la paralización de vehículos.

Además de mitigar riesgos, el sistema convierte la flota vehicular de una caja negra a un activo transparente y medible. Actualmente es difícil responder preguntas clave sobre costos y rendimiento, pero al centralizar el historial de cada unidad, desde mantenimientos hasta consumos, la plataforma genera reportes confiables que permiten a la gerencia tomar decisiones basadas en datos, no en intuición. Esto conduce a una mejor planificación de presupuestos, a optimizar la vida útil de los vehículos y a maximizar el retorno de inversión de cada activo.

Este enfoque en la optimización se complementa con una mejora directa en la eficiencia diaria, liberando al personal de tareas repetitivas y frustrantes. El tiempo que hoy se invierte buscando información en hojas de cálculo o coordinando tareas por WhatsApp se recupera gracias a una interfaz intuitiva donde cualquier dato está a solo un par de clics. La elección de tecnologías modernas como React y Flask no solo resuelve el problema actual, sino que sienta las bases para un sistema escalable que puede crecer junto con Fibratec, demostrando que esta no es solo una solución, sino una inversión en la eficiencia y capacidad de adaptación de la empresa a futuro.

## OBJETIVOS

### General

Desarrollar un sistema web de gestión vehicular para Fibratec que permita registrar, controlar y monitorear seguros, reemplaques, verificaciones, mantenimientos y notificaciones automatizadas, optimizando la administración y reduciendo riesgos operativos.

### Específicos

- Investigar los procesos actuales de gestión vehicular en Fibratec para identificar puntos críticos, definir los requerimientos funcionales y no funcionales del sistema.
- Diseñar los diagramas de la arquitectura del sistema (cliente-servidor), el diseño de la base de datos (modelo entidad-relación) y los prototipos de la interfaz de usuario (wireframes).
- Implementar la lógica del servidor con Python y Flask, incluyendo la API REST para la comunicación, el modelo de datos con SQLAlchemy y los endpoints de autenticación segura.
- Programar la interfaz de usuario interactiva utilizando React y Vite, creando componentes reutilizables para el registro de vehículos, visualización de datos y el sistema de autenticación.
- Desarrollar los módulos específicos para el registro detallado de vehículos, el historial de mantenimientos, y el sistema de alertas y notificaciones personalizables.
- Realizar pruebas integrales para validar la funcionalidad (pruebas unitarias y de integración), usabilidad y rendimiento del sistema.

## **CAPÍTULO I: ANTECEDENTES DE LA EMPRESA**

En este capítulo se presenta una descripción de Fibratec, abordando aspectos fundamentales que permiten comprender su funcionamiento. Se expone su misión, que refleja el propósito central de la organización; y su visión, que marca la dirección y metas a futuro. Finalmente, se describe su estructura organizacional, la cual muestra la forma en que se distribuyen las funciones y responsabilidades dentro de la empresa.

## 1.1 Concepto

En FIBRATEC, ofrecemos acceso a internet mediante tecnología de fibra óptica. Nuestra infraestructura moderna está diseñada para brindar conectividad tanto a hogares como a empresas, respaldada por un equipo de profesionales.

Nos enfocamos en ser más que un proveedor, actuando como un aliado comprometido. Priorizamos la transparencia en nuestro servicio y ofrecemos soporte constante para la tranquilidad de nuestros clientes. Ya sea para tu hogar o negocio, en FIBRATEC te ofrecemos una solución de internet adaptada a tus necesidades (FIBRATEC 2024).

## 1.2 Misión

Proveer servicio de acceso a internet mediante tecnología de fibra óptica para satisfacer las necesidades actuales de los usuarios. Nos comprometemos a mantener una infraestructura moderna y eficiente, que brinde un acceso adecuado para hogares y empresas. FIBRATEC se esfuerza por cumplir con las expectativas de nuestros clientes, ofreciendo soporte constante y soluciones que acompañen el crecimiento y la evolución tecnológica (FIBRATEC 2024).

## 1.3 Visión

Ser una de las más importantes empresas proveedoras del servicio de acceso a internet, utilizando infraestructura propia. Nos proyectamos hacia un futuro donde nuestra red de fibra óptica cubra de manera extensiva las principales regiones del país. Aspiramos a ser reconocidos por nuestra capacidad de adaptación a las nuevas tendencias del mercado y por nuestra contribución al desarrollo digital, fortaleciendo nuestra posición como un socio para nuestros clientes y un referente en la industria (FIBRATEC 2024).

## 1.4 Valores

(FIBRATEC 2024). En **FIBRATEC**, nuestros valores guían cada acción y decisión, asegurando que nuestro compromiso con la calidad y el servicio se refleje en todo lo que hacemos.

- **Calidad:** Servicios adaptados a las necesidades del cliente.

- **Innovación:** Uso de tecnología avanzada para mejorar continuamente.
- **Compromiso:** Enfoque constante en cumplir con las expectativas.
- **Confianza:** Actuamos como un socio responsable.
- **Crecimiento:** Contamos con infraestructura propia.

## 1.5 Croquis de ubicación

Privada Sócrates S/N, Col. Ampliación La Garita, entre Guadalupe Monroy Cruz y Confucio, C.P. 50457, Atlacomulco, Estado de México. Ref: Edificio blanco de 3 pisos con estacionamiento de unidades rotuladas Fibratec. Tel: 55 7879 3870, Correo: recursoshumanos.fibratec@gmail.com

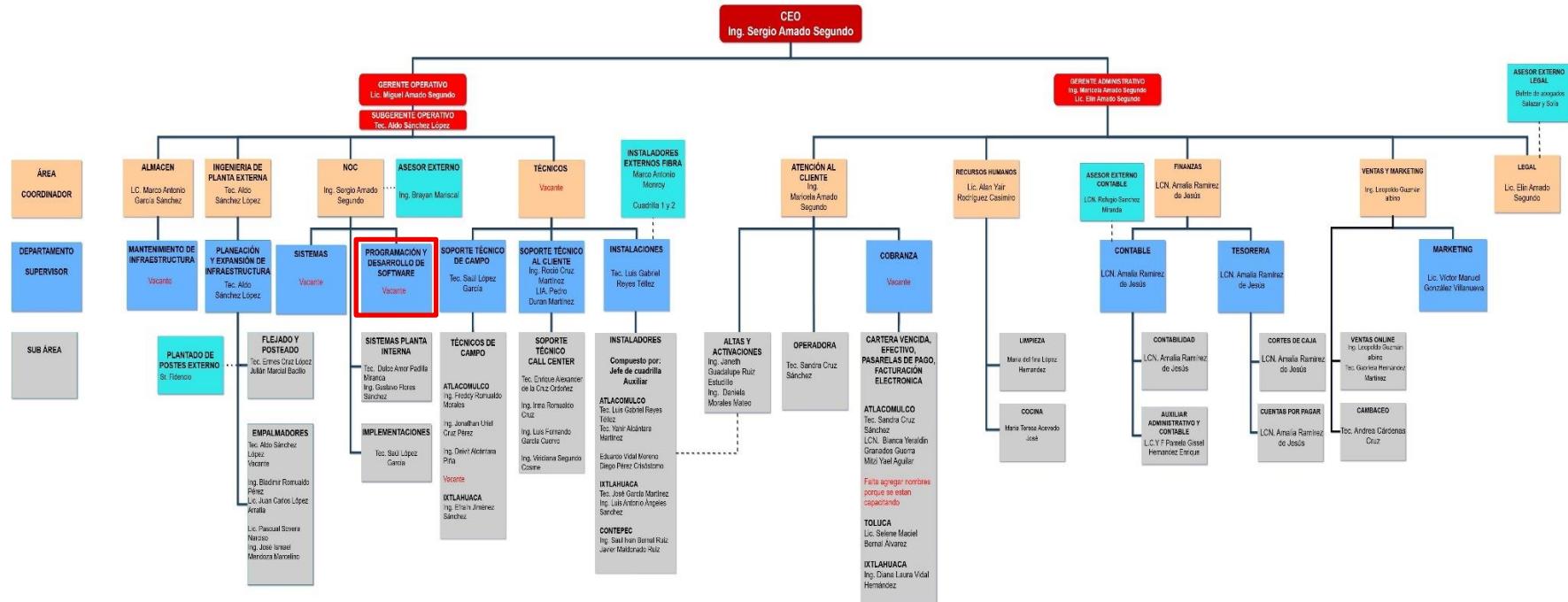


Figura. 1 Ubicación geográfica de la empresa FIBRATEC

Recuperado de [https://www.google.com.mx/maps/@19.8105463,-99.8745904,963m/data=!3m1!1e3?entry=ttu&g\\_ep=EgoyMDI1MDkyMy4wIKXMDSoASAFQAw%3D%3D44!4d-100.141752!16s%2Fg%2F11g\\_xvqy0?entry=ttu&g\\_ep=EgoyMDI1MDMxOS4xIKXMDSo ASAFQAw%3D%3D](https://www.google.com.mx/maps/@19.8105463,-99.8745904,963m/data=!3m1!1e3?entry=ttu&g_ep=EgoyMDI1MDkyMy4wIKXMDSoASAFQAw%3D%3D44!4d-100.141752!16s%2Fg%2F11g_xvqy0?entry=ttu&g_ep=EgoyMDI1MDMxOS4xIKXMDSo ASAFQAw%3D%3D)

## 1.6 Organigrama empresarial

Mi área de trabajo durante la residencia, como se muestra en el organigrama (ver Figura 2), corresponde al área de Programación y Desarrollo de Software.



**Figura. 2 Organigrama de la empresa  
Obtenido de: Fibratec**

## **CAPÍTULO II: MARCO TEÓRICO Y FUNDAMENTOS TECNOLÓGICOS**

En este capítulo se presenta una revisión general de las tecnologías empleadas en la administración vehicular. Se aborda el sistema web, el cual permite gestionar información desde cualquier computadora o dispositivo con acceso a internet, así como las APIs, que facilitan la comunicación e integración entre distintas plataformas.

## 2.1 Metodología

(Team Asana, 2025) nos dice que una metodología de gestión de proyectos es un sistema de principios, técnicas y procedimientos usados por personas que trabajan en una misma disciplina. Las principales metodologías se diferencian no solo por la manera en que están estructuradas, sino también por la naturaleza de los entregables, los flujos de trabajo e incluso por el software de gestión de proyectos.

### 2.1.1 Cascada (waterfall)

El modelo en waterfall es una metodología para gestión de proyectos que se divide en distintas fases. Cada fase comienza recién cuando ha terminado la anterior.

Este enfoque para la gestión de proyectos surgió a partir de los sectores de fabricación y construcción, en los que cada hito debe estar finalizado para poder avanzar con el proceso de producción. Por ejemplo, no puedes construir las paredes de una casa sin los cimientos. A pesar de que se inició en la fabricación, la gestión de proyectos waterfall se ha adaptado a las necesidades de muchos otros sectores diferentes, incluso al del desarrollo de software.



Figura. 3 Modelo en waterfall

Obtenido de: <https://assets.asana.biz/transform/ac9abbcb-9770-406e-a935-f64a10e17d1c/inline-project-management-project-management-methodologies-1-es-2x?io=transform:fill,width:960&format=webp>

## **2.2 Lenguajes de programación**

Un lenguaje de programación es una herramienta que permite desarrollar software o programas para computadora. Los lenguajes de programación son empleados para diseñar e implementar programas encargados de definir y administrar el comportamiento de los dispositivos físicos y lógicos de una computadora argumenta Marín Monterde, U. (2017).

Lo anterior se logra mediante la creación e implementación de algoritmos de precisión que se utilizan como una forma de comunicación humana con la computadora.

### **2.2.1 Python**

Amazon Web Services. (2023), nos dice que Python es un lenguaje de programación interpretado, de alto nivel y multiparadigma reconocido por su simplicidad y legibilidad, lo que facilita el desarrollo rápido y mantenible de aplicaciones. Su sintaxis clara y características como tipado dinámico, manejo automático de memoria y soporte para programación orientada a objetos y funcional hacen que sea una elección ideal para desarrollo web. Además, posee una amplia comunidad y un ecosistema robusto de bibliotecas que expanden sus capacidades, permitiendo abordar desde automatización hasta inteligencia artificial.

### **2.2.2 JavaScript (ES6+)**

JavaScript es el lenguaje estándar para el desarrollo frontend, capaz de crear páginas web interactivas que se ejecutan en el navegador del usuario. La versión ES6+ introduce características modernas como clases, módulos y funciones flecha que fomentan una mejor organización y mantenibilidad del código. Su naturaleza asíncrona y orientada a eventos lo hace ideal para gestionar interfaces dinámicas y comunicarse con servidores mediante tecnologías como Fetch API (y que).

## **2.3 Desarrollo backend (servidor)**

El desarrollo backend, también conocido como desarrollo del lado del servidor, es la disciplina encargada de construir y mantener la parte de una aplicación web que no es

visible para el usuario, pero que es esencial para su funcionamiento correcto y seguro. Según Arsys (2024), el backend funciona como la columna vertebral de cualquier sitio web, ya que en el servidor se ejecuta el código responsable de gestionar la lógica de negocio, procesar las solicitudes de los usuarios, interactuar con las bases de datos y garantizar que se entregue la información adecuada al frontend.

### **2.3.1 Flask**

Pallets Projects (2025) nos dice que Flask es un micro-framework para Python conocido por su flexibilidad y estructura minimalista, que permite a los desarrolladores ensamblar aplicaciones web y APIs RESTful con gran libertad arquitectónica. Provee enrutamiento, manejo de peticiones y plantillas HTML con Jinja2, simplificando la generación de contenido dinámico

#### **2.3.1.1 Flask-SQLAlchemy**

Esta extensión permite integrar el ORM (Object-Relational Mapping) SQLAlchemy a proyectos Flask, lo que facilita la manipulación de bases de datos de manera programática mediante objetos Python en vez de consultas SQL manuales. Esto mejora notablemente la productividad del desarrollador al abstraer la complejidad del manejo directo de la base de datos y permite portar el código más fácilmente a diferentes sistemas gestores (Pallets Projects, 2025). Flask-SQLAlchemy simplifica la definición y gestión de modelos, consultas, transacciones y migraciones, haciendo que trabajar con bases de datos relacionales en Flask sea más natural y coherente con la filosofía de Python.

#### **2.3.1.2 Werkzeug**

Werkzeug es un conjunto de utilidades que complementan a Flask proveyendo herramientas esenciales en la construcción de aplicaciones web. Entre sus funcionalidades más importantes está la gestión segura de contraseñas, que se realiza a través de la creación de hashes criptográficos y verificación de credenciales, lo que ayuda a proteger la información sensible de los usuarios. Además, Werkzeug ofrece funcionalidades para el manejo de requests y responses HTTP, manejo de sesiones, y depuración, siendo una pieza clave que aporta robustez y seguridad a las aplicaciones desarrolladas con Flask (Pallets Projects, 2025).

### **2.3.1.3 Flask-Mail**

Esta extensión facilita el envío de correos electrónicos desde las aplicaciones Flask, permitiendo funciones comunes como recuperación de contraseñas, confirmación de registros, notificaciones y alertas. Flask-Mail simplifica la configuración y uso de servicios SMTP, integra fácilmente plantillas para los correos y maneja correctamente respuestas y errores, lo que contribuye a mejorar la comunicación entre la aplicación y los usuarios (Pallets Projects, 2025). Su uso es fundamental en aplicaciones que requieran interacción constante vía email para validar o informar al usuario.

### **2.3.1.4 Flask-CORS**

Flask-CORS es crucial para manejar las políticas de CORS (Cross-Origin Resource Sharing), un mecanismo que controla la seguridad en la comunicación entre aplicaciones alojadas en diferentes dominios. Esta extensión permite configurar qué dominios externos pueden interactuar con la API o el backend de Flask, gestionando permisos para métodos HTTP, encabezados, y tiempo de cacheo de preflight requests. Según la documentación de Pallets Projects (2025), Flask-CORS es indispensable para evitar errores como "CORS policy" que bloquean peticiones entre frontend y backend cuando están en servidores o puertos diferentes, garantizando una comunicación segura y funcional en arquitecturas modernas con frontends desacoplados.

### **2.3.1.5 python-dotenv**

El módulo python-dotenv facilita la gestión segura y ordenada de variables de entorno en proyectos Flask, permitiendo cargar de forma automática definiciones almacenadas en un archivo .env. Esto es especialmente útil para manejar configuraciones sensibles como claves secretas, URLs de bases de datos, o configuraciones de terceros, sin exponer estos datos en el código fuente. La flexibilidad que aporta python-dotenv mejora la portabilidad y seguridad de la aplicación, permitiendo adaptar variables a distintos entornos (desarrollo, prueba, producción) de forma sencilla y segura (Pallets Projects, 2025).

### **2.3.1.6 Blueprints (estructura modular en Flask)**

Los Blueprints en Flask son elementos esenciales para organizar aplicaciones en módulos independientes, lo que mejora la mantenibilidad y escalabilidad del proyecto. Juncotic (2025) señala que permite dividir funcionalidades en componentes reutilizables y aislados. Por otro lado, Miguel Grinberg, experto en Flask, enfatiza en su libro "Flask Web Development" (2024) que la utilización de Blueprints es una buena práctica para proyectos reales que crecen en complejidad, ya que facilita el trabajo en equipo y la extensión del código sin afectar otras partes del sistema.

### **2.3.1.7 Middleware**

El middleware en Flask funge como una capa que intercepta solicitudes y respuestas, permitiendo la ejecución de código común, como autenticación o registro de eventos. GeeksforGeeks (2025) aclara que puede implementarse usando decoradores para funciones previas y posteriores a la petición. En "Flask Web Development", Grinberg (2024) explica que organizar esta lógica fuera de las vistas mejora la limpieza y el mantenimiento del código, favoreciendo la reutilización y la gestión centralizada de procesos transversales.

## **2.3.2 Serialización y deserialización de datos (JSON ↔ objetos Python)**

La serialización convierte objetos Python en strings JSON para interoperabilidad, mientras que la deserialización permite reconstruir objetos desde JSON. TutorialesProgramacionYa (s.f.) nos explica que Python tiene un módulo json robusto para estos procesos. Asimismo, Zed Shaw en su libro "Learn Python the Hard Way" (2014) destaca la importancia de manejar adecuadamente estas conversiones para trabajar con APIs y servicios web, dado que JSON es el formato estándar para intercambio de datos en aplicaciones modernas.

## **2.3.3 Manejo de errores y códigos de estado HTTP (200, 404, 500, etc.)**

Una gestión efectiva de errores HTTP es fundamental para mejorar la experiencia del usuario y facilitar el mantenimiento. Flask-es ReadTheDocs (2009) detalla que Flask provee mecanismos para personalizar respuestas según el código de error, como las páginas 404 o 500. Miguel Grinberg (2024) también explica en su libro que el uso de abort() y

controladores específicos permite crear aplicaciones robustas y manejar excepciones con respuestas claras y consistentes.

#### **2.3.4 Autenticación y autorización (concepto)**

La autenticación valida quién es el usuario, y la autorización determina qué puede hacer. DigitalOcean (2020) resalta que Flask no incluye estas funciones por defecto, pero ofrece bibliotecas como Flask-Login para su fácil implementación. En "Flask Web Development", Grinberg (2024) señala que un buen sistema de autenticación y autorización es crucial para preservar la seguridad y privacidad, evitando accesos indebidos y garantizando que los usuarios tengan permisos adecuados según su rol.

### **2.4 Desarrollo frontend (Interfaz de usuario)**

El desarrollo frontend se refiere a la práctica de construir y diseñar la interfaz de usuario de un sitio web o aplicación, abarcando la estructura visual, la presentación y la interactividad con la que los usuarios interactúan directamente. Esto implica el uso de tecnologías como HTML, CSS y JavaScript para crear páginas web visualmente atractivas, accesibles y funcionales en diferentes dispositivos y navegadores. El frontend se enfoca en ofrecer una experiencia de usuario intuitiva y fluida, integrando también frameworks y librerías como React para mejorar la escalabilidad y el rendimiento (MDN Web Docs, 2025).

#### **2.4.1 Vite**

Vite es una herramienta moderna de desarrollo frontend que ofrece un servidor de desarrollo muy rápido y eficiente, con recarga inmediata de módulos (Hot Module Replacement), optimizando el ciclo de desarrollo. Para producción, utiliza Rollup para hacer un empaquetado optimizado, aplicando técnicas como tree-shaking para eliminar código no usado y reducir el tamaño final (Vite, 2025).

#### **2.4.2 React**

React es una biblioteca de JavaScript de código abierto creada por Facebook que permite construir interfaces de usuario basadas en componentes reutilizables. Utiliza un DOM virtual

para actualizar eficientemente únicamente las partes modificadas de la interfaz, lo que mejora el rendimiento. React emplea JSX, una sintaxis que combina JavaScript y HTML permitiendo una expresión declarativa más legible del UI. React también ofrece hooks, funciones que permiten manejar el estado y efectos secundarios en componentes funcionales con gran simplicidad. Su arquitectura facilita la creación de aplicaciones escalables y mantenibles, principalmente para aplicaciones de una sola página (SPA) (ReactJS, 2025; React.dev, 2025).

#### **2.4.2.1 React Router DOM**

React Router DOM es una biblioteca esencial para aplicaciones React de una sola página (SPA), que gestiona la navegación interna sin recargar la página ni perder el estado de la aplicación. React Router permite definir rutas específicas que corresponden a diferentes componentes o vistas, facilitando la transición fluida entre ellas. Según la documentación de React Router (2025), esta librería intercepta las acciones del usuario y actualiza la interfaz de forma dinámica, manteniendo una experiencia de usuario rápida y consistente. Esto evita la recarga completa del documento HTML y permite mantener en memoria el estado de cada componente, lo cual es fundamental para aplicaciones modernas y dinámicas.

#### **2.4.2.2 CSS Modules**

CSS Modules es una técnica que permite encapsular estilos CSS en componentes específicos, evitando la contaminación global de estilos y los conflictos típicos del CSS tradicional. ReactJS (2025) explica que mediante CSS Modules, los estilos se asignan de manera local a cada componente, permitiendo que varios componentes tengan estilos similares sin interferir entre sí. Esto fomenta la modularidad y mantenibilidad en el desarrollo frontend, ya que cada componente lleva consigo su propio conjunto de reglas visuales aisladas.

#### **2.4.3 Bootstrap**

Bootstrap es un framework CSS muy popular que provee un sistema de diseño responsive, junto con componentes visuales predefinidos como botones, menús y formularios, lo que acelera el desarrollo de interfaces agradables y funcionales. Bootstrap (2025) indica que su uso garantiza consistencia en el diseño y facilita la accesibilidad, soportando varios

dispositivos y tamaños de pantalla. Combinar Bootstrap con React acelera la construcción de interfaces profesionales con poco esfuerzo.

#### **2.4.4 Font Awesome**

Font Awesome es una biblioteca de iconos vectoriales escalables que mejora la presentación visual y la usabilidad de la interfaz de usuario. Font Awesome (2025) destaca que sus iconos son personalizables en tamaño, color y animación, y se integran fácilmente en aplicaciones React para enriquecer la experiencia del usuario y facilitar la comprensión visual de acciones y estados.

#### **2.4.5 Fetch API**

Fetch API es una interfaz nativa de los navegadores que permite realizar llamadas HTTP para comunicarse con servicios RESTful desde el frontend. Según MDN Web Docs (2025), Fetch ofrece una forma moderna y basada en promesas para realizar solicitudes asíncronas, facilitando la interacción con APIs para obtener o enviar datos sin recargar la página. Es fundamental en aplicaciones web modernas que dependen de datos dinámicos desde servidores externos.

#### **2.4.6 Componentes funcionales**

Los componentes funcionales en React son funciones de JavaScript que reciben un objeto de propiedades (props) como argumento y retornan elementos de React que describen lo que debe aparecer en la interfaz. Según la documentación oficial de React (2021), esta forma de componentes es la más simple y se recomienda para construir interfaces de usuario porque es más fácil de entender, depurar y probar. Además, los componentes funcionales favorecen una mejor separación de responsabilidades y reutilización del código.

##### **2.4.6.1 Props y State**

Las props son las propiedades o parámetros que se pasan a los componentes para personalizar su contenido, mientras que el state es un objeto local que mantiene información que puede cambiar con el tiempo y afectar la renderización. React.dev (2025) explica que las props son inmutables, usadas para configurar componentes, y el estado es mutable,

gestionado dentro del propio componente para reflejar cambios en la UI en respuesta a interacciones del usuario o datos externos.

#### **2.4.6.2 Hooks (useState, useEffect, etc.)**

Los Hooks son funciones especiales que permiten usar estado y otras características de React en componentes funcionales sin escribir clases. Por ejemplo, useState permite agregar estado local, y useEffect maneja efectos secundarios como llamadas a APIs o suscripciones. MDN Web Docs (2024) señala que los Hooks simplifican el código y mejoran la legibilidad, promoviendo así prácticas modernas de desarrollo con React.

#### **2.4.6.3 Renderizado condicional**

El renderizado condicional en React implica mostrar diferentes elementos o componentes basados en condiciones lógicas. React.dev (2025) indica que esto se puede lograr con operadores ternarios o expresiones lógicas dentro del JSX, permitiendo construir interfaces dinámicas que respondan a eventos o cambios en el estado, mejorando la experiencia del usuario.

#### **2.4.6.4 Responsividad (media queries, grid/flexbox)**

La responsividad refiere a la habilidad de adaptar la UI a distintos tamaños y dispositivos. Según MDN Web Docs (2024), las media queries permiten aplicar estilos CSS según las características del dispositivo, mientras que Grid y Flexbox ofrecen sistemas flexibles para distribuir y alinear elementos en la página. Estas herramientas combinadas son la base del diseño web moderno adaptable (responsive design).

#### **2.4.6.5 SPA (Single Page Application)**

Una SPA es una aplicación web que carga una sola página HTML y realiza navegación interna sin recargar la página, ofreciendo una experiencia más rápida y fluida. React Router (2025) explica que mediante este enfoque se gestionan las rutas internamente en el cliente, reduciendo tiempos de espera y manteniendo el estado de la aplicación sin interrupciones visibles para el usuario.

## 2.5 Base de datos

Una base de datos es un sistema organizado para almacenar, gestionar y recuperar información de forma eficiente. En aplicaciones web, las bases de datos relacionales como MySQL permiten guardar datos estructurados en tablas relacionadas. El acceso a estas bases se realiza a través de consultas SQL, y para facilitar su manejo en programación se emplean herramientas como ORM (Object-Relational Mapping), que traducen datos entre el formato relacional y objetos en código. La correcta gestión de la base de datos es crucial para garantizar la integridad, seguridad y rendimiento de la aplicación (Oracle, 2025).

### 2.5.1 MySQL

MySQL es un sistema de gestión de bases de datos relacional robusto y ampliamente utilizado en aplicaciones web por su eficiencia y escalabilidad. Almacena datos estructurados en tablas relacionadas y permite consultas SQL para recuperación y manipulación de información (Oracle, 2025).

#### 2.5.1.1 Tabla (Table)

Microsoft (2025) señala que una tabla es la estructura fundamental para organizar y almacenar datos en una base de datos relacional. Está compuesta por filas y columnas donde cada fila representa un registro individual y cada columna un atributo o campo específico. Esta estructura permite ordenar grandes volúmenes de datos de forma que se facilite el acceso, la gestión y la integración. Además, las tablas pueden estar relacionadas entre sí mediante claves primarias y foráneas, lo que posibilita consultas complejas y análisis integrados que reflejan relaciones del mundo real.

La importancia de definir correctamente las tablas radica en que representan entidades concretas y constituyen la base para la integridad y el rendimiento del sistema. Por ejemplo, una tabla “Clientes” puede contener atributos como nombre, dirección y teléfono, mientras que una tabla “Pedidos” registra información sobre cada pedido realizado, vinculándola a los clientes mediante un identificador único (IBM, 2021).

### **2.5.1.2 Llaves primarias (Primary Keys)**

Las llaves primarias son atributos únicos que identifican de forma exclusiva cada registro en una tabla de una base de datos relacional. Según la documentación oficial de Microsoft SQL Server (2025), la llave primaria garantiza que no existan registros duplicados ni valores nulos en la columna o columnas designadas. Esta restricción es fundamental para mantener la integridad de los datos y permite utilizar índices que optimizan la búsqueda de información dentro de la base de datos.

### **2.5.1.3 Llaves foráneas (Foreign Keys)**

Una llave foránea es una columna que establece un vínculo entre una tabla y la llave primaria de otra, garantizando que el valor de la columna exista en la tabla referenciada. Microsoft (2025) detalla que esta relación asegura la integridad referencial, evitando inconsistencias como referencias a registros inexistentes y permite definir acciones para mantener la coherencia al actualizar o eliminar datos.

### **2.5.1.4 Índices (Indexes)**

Los índices son estructuras de datos que mejoran notablemente la velocidad de acceso a los registros en tablas extensas. En la documentación de SQL Server (2025), se explica que los índices pueden ser únicos o no, y su correcto diseño influye en el rendimiento de las consultas SELECT, así como en las operaciones de inserción, actualización y eliminación. Se recomienda un análisis cuidadoso para equilibrar velocidad y consumo de recursos.

### **2.5.1.5 Integridad referencial**

La integridad referencial es el conjunto de reglas que aseguran que las relaciones entre tablas mantengan la consistencia de los datos. Según Microsoft (2025), esta integridad se logra mediante el uso de llaves foráneas que restringen las operaciones que podrían causar datos huérfanos, permitiendo acciones en cascada y verificaciones automáticas.

### **2.5.1.6 Normalización (1FN, 2FN, 3FN)**

La normalización es un proceso para organizar datos y minimizar la redundancia. El libro Fundamentos de Bases de Datos (Elmasri & Navathe, 2016) define las tres primeras formas normales: la 1FN elimina grupos repetitivos, la 2FN elimina dependencias parciales y la 3FN elimina dependencias transitivas. Este diseño reduce problemas de inconsistencia y facilita mantenimiento y eficiencia en las bases de datos.

### **2.5.1.7 Consultas SQL (SELECT, INSERT, UPDATE, DELETE → CRUD)**

Las consultas SQL básicas forman la base para interactuar con cualquier base de datos relacional. Según la documentación oficial de Microsoft SQL Server (2025), el comando SELECT se utiliza para recuperar datos almacenados en una o varias tablas, permitiendo filtrar, ordenar y agrupar la información según sea necesario. Por otro lado, INSERT permite agregar nuevos registros, mientras que UPDATE modifica los datos existentes y DELETE elimina registros que ya no se requieren.

Estas operaciones CRUD son fundamentales para cualquier sistema que gestione información, ya que permiten crear, leer, actualizar y borrar datos en la base de datos. La documentación enfatiza que el uso eficiente y correcto de estas sentencias es clave para mantener la integridad, rendimiento y seguridad de la información.

### **2.5.1.8 Relaciones (1:1, 1:N, N:N)**

La organización y estructuración de datos en bases de datos relacionales dependen en gran medida del diseño de las relaciones entre tablas. Según Microsoft SQL Server (2025), la relación uno a uno (1:1) establece que un registro de una tabla está vinculado exclusivamente a un registro de otra, lo que generalmente se usa para dividir datos que podrían ser opcionales o contener detalles específicos.

La relación uno a muchos (1:N) es la más común y permite que un registro en una tabla principal esté asociado a múltiples registros en una tabla dependiente, ideal para modelar situaciones como clientes y sus órdenes. Finalmente, la relación muchos a muchos (N:N) requiere una tabla intermedia para mantener asociaciones múltiples entre registros de ambas tablas, permitiendo flexibilidad y normalización eficiente.

### **2.5.2 PyMySQL / mysqlclient**

Son conectores que permiten a Python comunicarse con bases de datos MySQL, facilitando a ORMs como SQLAlchemy la interacción con la base de datos y el manejo de transacciones seguras (Oracle, 2025).

## **2.6 Entorno de desarrollo y herramientas**

El entorno de desarrollo comprende el conjunto de aplicaciones, librerías y herramientas que permiten a los desarrolladores escribir, probar, depurar y gestionar el código del sistema. Herramientas como Node.js y npm facilitan la gestión de paquetes y la ejecución de JavaScript en el desarrollo frontend, mientras que los entornos virtuales en Python (venv) aíslan las dependencias del backend. Además, aplicaciones gráficas como MySQL Workbench proporcionan interfaces visuales para crear y administrar bases de datos, y herramientas como Postman permiten probar las APIs antes de integrarlas con el frontend (Node.js Foundation, 2025; Python Software Foundation, 2025; Oracle, 2025).

### **2.6.1 Node.js y npm**

Node.js es un entorno de ejecución para JavaScript que permite ejecutar este lenguaje fuera del navegador, en el servidor o en sistemas locales. Node.js destaca por ser multiplataforma y orientado a eventos, lo que facilita la construcción de aplicaciones escalables y eficientes, especialmente en tiempo real. Según la Node.js Foundation (2025), este entorno ha revolucionado la forma de desarrollar aplicaciones al llevar JavaScript más allá del navegador, permitiendo a los desarrolladores utilizar un solo lenguaje tanto para el frontend como para el backend.

Por su parte, npm (Node Package Manager) es el gestor de paquetes por excelencia que viene integrado con Node.js. Facilita la instalación, actualización y gestión de todas las dependencias necesarias para proyectos, especialmente en el desarrollo frontend moderno. Esto incluye bibliotecas populares como React y Vite, permitiendo un manejo estructurado y sencillo de las librerías que un proyecto necesita. npm administra además un amplio

ecosistema de módulos disponibles públicamente o de manera privada para acelerar y facilitar el desarrollo.

### **2.6.2 Python venv y pip**

Python ofrece herramientas fundamentales para el manejo eficiente de dependencias en proyectos. El módulo `venv` permite crear entornos virtuales aislados, de modo que cada proyecto pueda tener sus propias librerías sin interferir con otros proyectos o la instalación global de Python. Esto asegura la reproducibilidad y estabilidad del entorno de desarrollo. Junto con `venv`, `pip` es el gestor de paquetes estándar que sirve para instalar, actualizar y administrar bibliotecas dentro de estos entornos, garantizando que el proyecto cuente con las versiones específicas requeridas para funcionar correctamente (Python Software Foundation, 2025).

### **2.6.3 MySQL Workbench**

MySQL Workbench es una herramienta gráfica oficial proporcionada por Oracle que facilita el diseño, modelado y administración de bases de datos MySQL mediante interfaces visuales intuitivas. Esta aplicación proporciona funcionalidades para crear diagramas ER, gestionar esquemas, ejecutar consultas y administrar usuarios de forma centralizada y amigable. Asociado con Oracle, MySQL Workbench permite a los desarrolladores y administradores una gestión integral y eficiente de las bases de datos a través de su entorno gráfico, además de simplificar tareas complejas que de otro modo requerirían comandos manuales (Oracle, 2025).

### **2.6.4 Postman, Insomnia, Python Requests**

Estas herramientas y librerías están orientadas a la comprobación y validación de APIs RESTful antes y durante la integración con aplicaciones frontend. Postman e Insomnia son clientes REST gráficos que permiten a los desarrolladores enviar peticiones HTTP, visualizar respuestas, gestionar colecciones de pruebas y automatizar tests, facilitando la garantía de funcionamiento correcto de cada endpoint. Python Requests es una biblioteca que permite realizar solicitudes HTTP desde código Python de forma simple y poderosa,

útil para automatizar pruebas u operaciones relacionadas con APIs y servicios externos (Postman, 2025).

### 2.6.5 Git y GitHub

Git es un sistema de control de versiones distribuido que permite registrar y gestionar cambios en el código fuente a lo largo del tiempo, habilitando la colaboración eficiente entre múltiples desarrolladores. GitHub es una plataforma basada en la nube que utiliza Git como motor para almacenar repositorios, facilitando la gestión de proyectos, revisión de código, integración continua y despliegue. Según Git SCM (2025), usar Git y GitHub es esencial para mantener un flujo de trabajo ordenado, colaborativo y seguro en el desarrollo de software moderno.

### 2.6.6 Variables de entorno

Las variables de entorno en React se almacenan en un archivo .env en la raíz del proyecto. Para que React las reconozca, deben comenzar con el prefijo REACT\_APP\_, por ejemplo:

```
REACT_APP_API_URL=https://api.example.com
```

Luego, se acceden en el código mediante process.env.REACT\_APP\_API\_URL. Es crucial reiniciar el servidor de desarrollo tras modificar el archivo para que los cambios surtan efecto. Además, el archivo .env debe ser excluido del control de versiones, para evitar exponer información sensible como credenciales.

Esta práctica permite separar la configuración del código y manejar diferentes valores según el entorno (desarrollo, producción), mejorando la seguridad y flexibilidad en la aplicación (The Lost Reference, 2021).

## 2.7 Arquitectura y conceptos clave

La arquitectura de software representa el diseño estructural fundamental de un sistema. Define cómo se organizan sus componentes, módulos, y cómo se comunican entre sí para cumplir los objetivos funcionales y no funcionales del sistema (ToGrow Agencia, 2024). Esta

estructura es esencial para garantizar que el software sea eficiente, escalable, mantenable y seguro.

### **2.7.1 Arquitectura cliente-servidor**

Modelo en el que una aplicación frontend (cliente) y backend (servidor) están separadas, comunicándose a través de solicitudes y respuestas, permitiendo escalabilidad y distribución eficiente de responsabilidades (Microsoft Docs, 2025).

### **2.7.2 API REST**

Patrón arquitectónico para la comunicación entre cliente y servidor mediante protocolos HTTP, empleando métodos estándar como GET, POST, PUT y DELETE para el intercambio de recursos, manteniendo la comunicación stateless (Fielding, 2000).

### **2.7.3 ORM (Object-Relational Mapping)**

Técnica que mapea las tablas y relaciones de bases de datos relacionales a objetos en código, permitiendo manipular la base de datos a través de código orientado a objetos, lo que facilita el mantenimiento y reduce errores (SQLAlchemy Documentation, 2024).

### **2.7.4 Entidad-Relación (ERD)**

El Diagrama Entidad-Relación (ERD) es una representación gráfica que muestra entidades (objetos, personas, conceptos), sus atributos y las relaciones que las vinculan en un sistema o base de datos. Esencial para el diseño de bases de datos, permite a analistas y desarrolladores planificar cómo se estructurarán y conectarán los datos, facilitando la identificación de dependencias y cardinalidades (uno a uno, uno a muchos, muchos a muchos). La claridad visual que ofrece ayuda a evitar redundancias y errores en el modelo de datos.

Como indica Miro (2024), los ERD usan símbolos como rectángulos para entidades, óvalos para atributos y rombos para relaciones, facilitando la comprensión y comunicación entre equipos técnicos.

## **CAPÍTULO III: DISEÑO DE LAS INTERFACES Y MODELOS RELACIONALES**

Este capítulo detalla la arquitectura del sistema web, presentando diagramas de casos de uso, modelo entidad-relación y la estructura de la base de datos. Se especifican también los requerimientos funcionales y no funcionales, así como la interfaz de usuario propuesta para asegurar una experiencia intuitiva.

### **3.1 Diagrama entidad relación**

El Modelo Entidad Relación (MER o ERD) es una herramienta para el modelado de datos que ayuda a representar entidades en una base de datos de forma visual e intuitiva (Alicante, 2023).

#### **3.1.1 Sistema inicial**

El sistema inicia con la gestión del personal. La tabla **Usuarios** es el punto de entrada, manejando las credenciales de acceso, los roles del sistema (admin, usuario). Separadamente, la tabla **Choferes** almacena la información detallada del personal de conducción, incluyendo su CURP, domicilio y la vigencia de su licencia.

El eje central de la base de datos es la tabla **Unidades**, que registra todos los datos maestros de cada vehículo de la flota (NIV, marca, modelo, motor, sucursal, etc.).

A partir de **Unidades**, la documentación legal y operativa se adjunta en relaciones uno-a-uno:

- La tabla **Garantias** se vincula a la unidad para registrar la póliza de seguro, la aseguradora y las vigencias.
- La tabla **Placas** registra el número de matrícula y sus fechas de expedición y expiración.
- La tabla **VerificacionVehicular** almacena los datos de cumplimiento, como el holograma y las fechas de los períodos de verificación.

La operación diaria se gestiona a través de la tabla **Asignaciones**, la cual conecta al personal de **Choferes** con una **Unidad** específica, registrando cuándo comienza y, opcionalmente, cuándo termina el periodo de uso.

#### **3.1.2 Gestión de mantenimiento**

La sección de mantenimiento se basa en la planificación y la ejecución.

Primero, la tabla **TiposMantenimiento** actúa como un catálogo de servicios estandarizados, definiendo la frecuencia ideal (por tiempo o kilometraje) para cada tipo de servicio. Estos tipos se aplican a cada vehículo en la tabla **MantenimientosProgramados**, la cual calcula y establece el próximo servicio pendiente para la Unidad.

Cuando el servicio se realiza, se registra en la tabla Mantenimientos, que guarda detalles como la fecha de realización, el kilometraje, el costo y el personal o empresa que lo ejecutó. Para el registro de averías (mantenimiento correctivo), se utiliza la tabla FallasMecanicas. Esta tabla detalla la avería y hace referencia a dos catálogos de inventario: la tabla Piezas (para identificar el tipo de refacción) y la tabla **MarcasPiezas** (para identificar el fabricante de la pieza instalada).

### 3.1.3 Auditoría e historial de cambios

Finalmente, el esquema incluye cinco tablas de auditoría, las cuales son la única fuente para la trazabilidad de cambios. Estas tablas de Historial (ej., HistorialPlacas, HistorialGarantias) son alimentadas automáticamente por Triggers definidos en la base de datos. Cuando un campo crítico (como el número de póliza, la vigencia de placa o el costo de un mantenimiento) se actualiza en las tablas principales, el *trigger* inserta el registro anterior en la tabla de Historial correspondiente, asegurando que siempre se conserve un registro inmutable del estado del dato antes de la modificación, ver Figura 4.

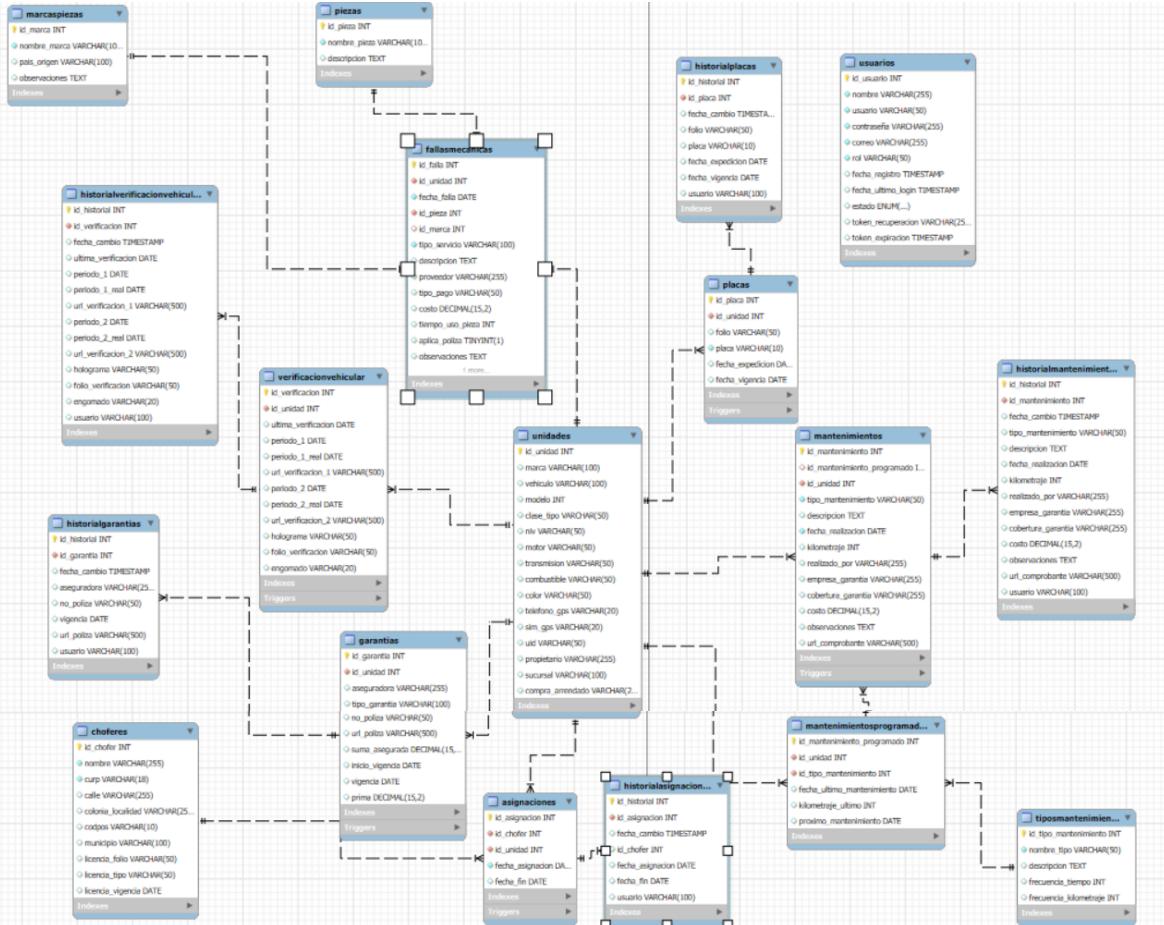


Figura. 4 Modelo entidad relación  
(Elaboración propia)

### 3.2 Diagrama de casos de uso

El Sistema de Gestión de Flota está diseñado en torno a tres actores principales: el Administrador, el Usuario/Gerente y el Sistema Historial. El acceso está segmentado estrictamente por roles.

#### 3.2.1 Funciones del administrador (acceso total)

El Administrador posee control completo (Crear, Actualizar, Eliminar) sobre la totalidad de los módulos del sistema. Es el único responsable de la Gestión de Usuarios, la Gestión de Choferes y la Gestión de Fallas y Piezas. Además, es el principal responsable de la Gestión de Unidades, donde registra información crítica como Garantías, Placas y Verificaciones

Vehiculares. Finalmente, supervisa todas las Asignaciones y tiene acceso completo a la Gestión de Mantenimientos.

### **3.2.2 Funciones del usuario/gerente (operación y consulta)**

El Usuario/Gerente tiene acceso restringido, enfocado en operaciones diarias y seguimiento. Su rol se limita a la Gestión de Mantenimientos, donde puede Registrar y Programar las tareas. Además, tiene acceso completo a todos los módulos de consulta: puede Consultar Historiales y Generar Reportes gerenciales. El Usuario/Gerente no puede eliminar, ni gestionar usuarios o choferes.

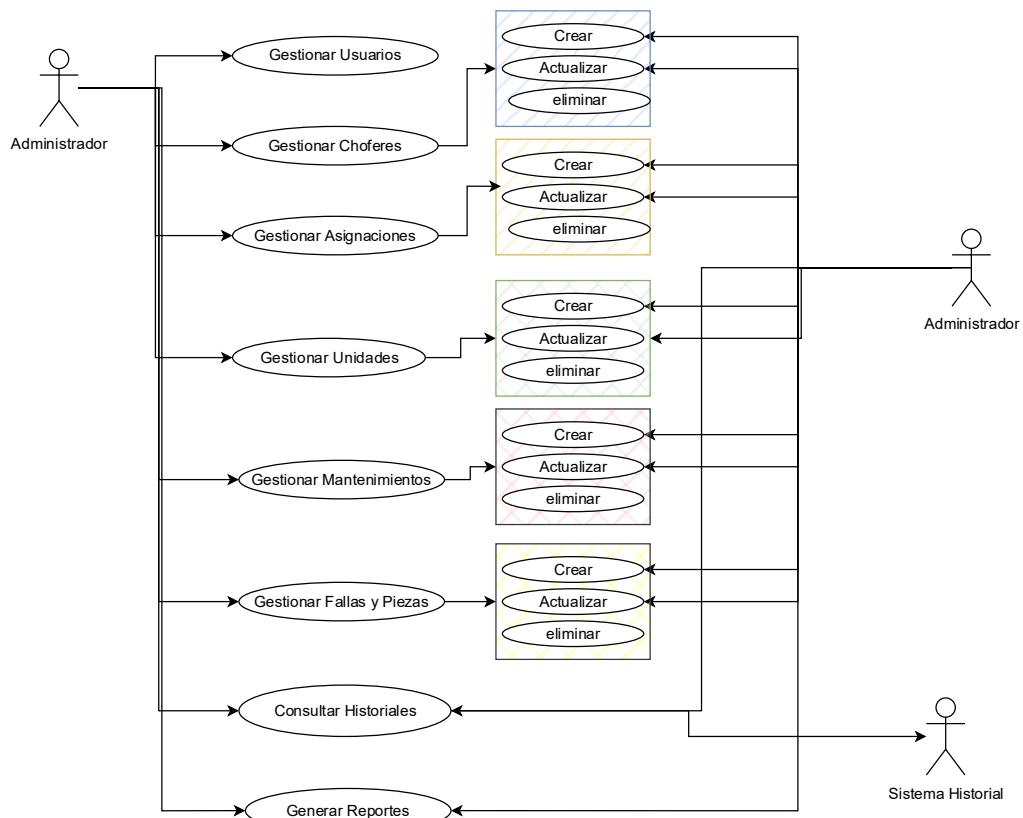
### **3.2.3 Flujo crítico y lógica automática**

El sistema incorpora procesos automáticos para la validación y trazabilidad de datos:

- Validación de Asignaciones: Al intentar Crear o Actualizar una Asignación, el sistema ejecuta obligatoriamente la función Validar Solapamiento de Fechas para prevenir conflictos de tiempo.
- Trazabilidad (Triggers): El Sistema Historial es el actor responsable de la trazabilidad. Tras cualquier acción de Creación o Actualización en módulos clave (Unidades, Mantenimientos y Asignaciones), el sistema dispara automáticamente la función Actualizar Historial, asegurando un registro completo de todos los cambios significativos.

### **3.2.4 Consultas y reportes**

Tanto el Administrador como el Usuario/Gerente obtienen información del Sistema Historial para Consultar Historiales detallados de unidades, asignaciones y mantenimientos, y para Generar Reportes operativos y gerenciales plasmados en la Figura. 5.



**Figura. 5 Diagrama de caso de uso  
(Elaboración propia)**

### 3.3 Maquetado

Esta pantalla de inicio de sesión presenta un diseño moderno, profesional y minimalista. Su tema oscuro, basado en grises y negro, le da un aspecto elegante, mientras que el rojo se utiliza como un color de acento inteligente para destacar la acción principal: el botón "Entrar". La composición es totalmente centrada y el uso de iconos claros, como el de usuario y el candado, crea una interfaz limpia e intuitiva que va directo al punto, como se muestra en la Figura. 6.

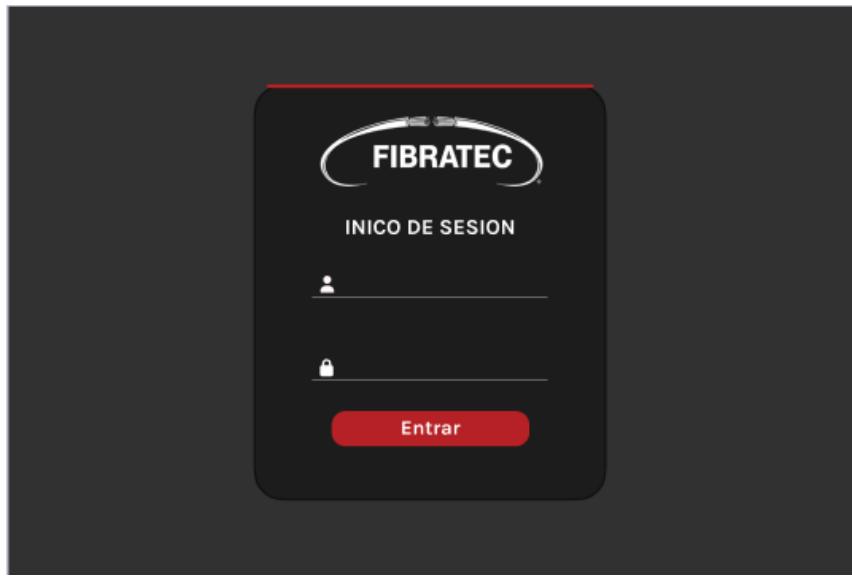


Figura. 6 Inicio de sesión  
(Elaboración propia)

La Figura. 7 muestra el panel principal de la plataforma "CAFITECH". La interfaz se divide en un menú de navegación a la izquierda, un encabezado superior con buscador y sesión de usuario, y un área de contenido principal que resume los módulos de Mantenimientos, Verificaciones y Garantías a través de tarjetas con indicadores clave. En la parte inferior, se visualiza un gráfico de barras de mantenimientos mensuales y una sección de acciones rápidas.

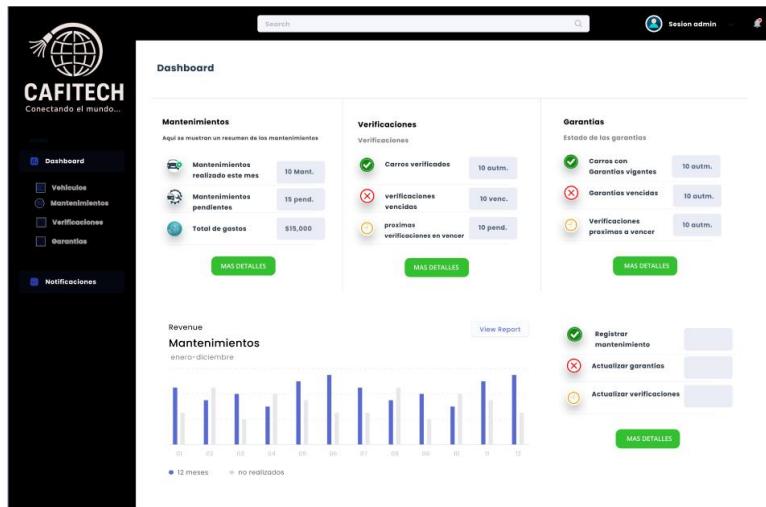


Figura. 7 Dashboard de la aplicación  
(Elaboración propia)

La imagen correspondiente a la sección Unidades Vehiculares de la plataforma 'CAFITECH' se muestra en la Figura. 8. La interfaz conserva el menú de navegación lateral izquierdo, con la opción Vehículos seleccionada. En el área principal de contenido se observa un título y una barra de filtros que permite buscar y organizar las unidades registradas. Debajo de esta, se despliega la tabla de automóviles, donde se listan los vehículos con sus detalles y estado, incluyendo botones para exportar, agregar, editar o eliminar registros.

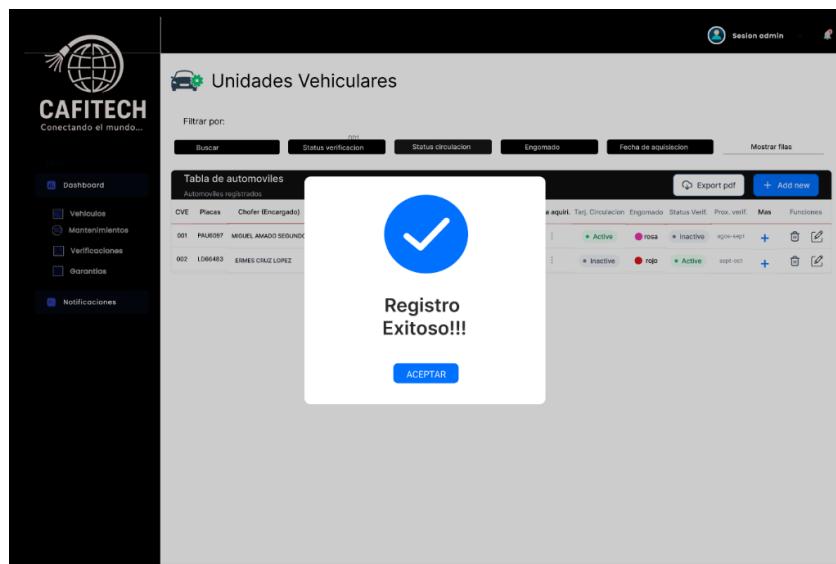
The screenshot shows the 'Unidades Vehiculares' (Vehicle Units) section of the CAFITECH platform. On the left, there's a sidebar with a globe icon and the text 'CAFITECH Conectando el mundo...'. Below it are navigation links: Dashboard, Vehículos (selected), Mantenimientos, Verificados, and Servicios. Under 'Vehículos', there are sub-links for Placas, Chofer (Encargado), Marca, Vehículo, Modelo, NIV, Fecha adq, L. Tér, Circulación, Degradado, Status Verif., Pcox. verif., and More. A 'Notificaciones' link is also present. The main content area has a title 'Unidades Vehiculares' with a blue info icon. Below it is a search bar with 'Buscar' and a magnifying glass icon, and dropdown filters for 'Status verificado', 'Status circulante', 'Fechas', 'Periodo', and 'Período de expedición'. There's also a 'Más filtros' button. A table titled 'Tabla de automóviles' (List of vehicles) displays two rows of data. The first row is for a vehicle with license plate 'B01 PAUW07', owner 'MIGUEL ANGEL ARENDON', model 'CRUISER', color 'GRIS METALICO', year '2023', and status 'Active'. The second row is for a vehicle with license plate '001 LOMR03', owner 'EMMIE CRUZ LOPEZ', model 'VOLKSWAGEN', color 'GRIS METALICO', year '2020', and status 'Inactivo'. The table includes columns for 'EVC', 'Placas', 'Chofer (Encargado)', 'Marca', 'Vehículo', 'Modelo', 'NIV', 'Fecha adq', 'L. Tér.', 'Circulación', 'Degradado', 'Status Verif.', 'Pcox. verif.', 'More', and 'Acciones'. Buttons for 'Export pdf', '+ Add new', and 'Import' are at the top right of the table. Row actions include 'Actualizar', 'Eliminar', and 'Duplicar'.

Figura. 8 Apartado de unidades vehiculares  
(Elaboración propia)

La Figura. 9 ilustra la sección Unidades Vehiculares de la plataforma CAFITECH. En esta vista, el área principal de contenido se encuentra parcialmente cubierta por el modal Registro de nuevo vehículo, el cual contiene múltiples campos de entrada distribuidos en dos columnas para capturar información detallada del vehículo, como marca, placas, motor y color, así como datos del chofer. Al final del formulario se localizan los botones de acción para Salir o Registrar el nuevo vehículo.

**Figura. 9 formulario de registrar automóvil  
(Elaboración propia)**

En la Figura. 10 se muestra la sección "Unidades Vehiculares" de la plataforma CAFITECH, evidenciando el estado posterior a la realización de una operación. El contenido principal está superpuesto por un modal de confirmación centrado en la pantalla. Este modal exhibe un ícono de verificación azul acompañado del mensaje Registro Exitoso, señalando que el nuevo vehículo se ha creado correctamente. Al pie del modal se encuentra únicamente el botón azul "ACEPTAR" para cerrar la notificación.



**Figura. 10 Alerta de éxito en el registro  
(Elaboración propia)**

En la Figura. 11 se observa la exportación de los datos de los vehículos en la plataforma CAFITECH, acompañada de la notificación de descarga. La interfaz principal está cubierta por un modal centrado, que muestra un ícono de descarga junto al mensaje "PDF descargado correctamente". En la parte inferior, se encuentra el botón "ACEPTAR" para cerrar la notificación y continuar con la operación.

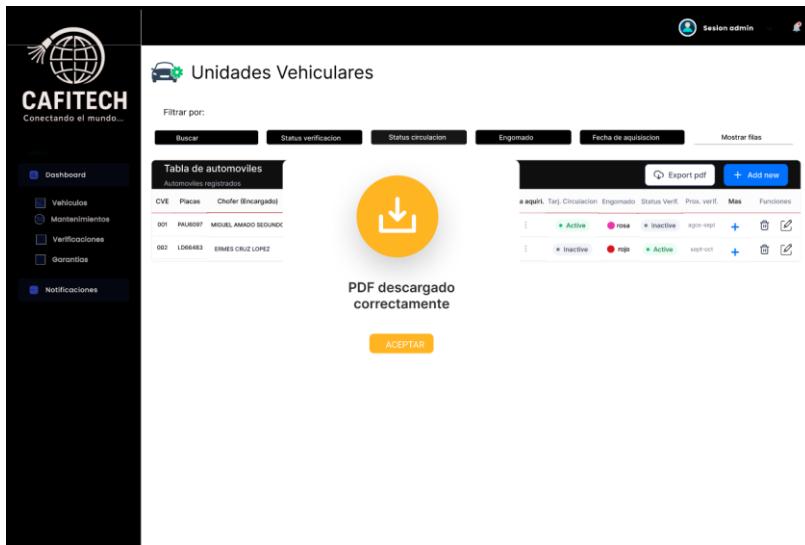
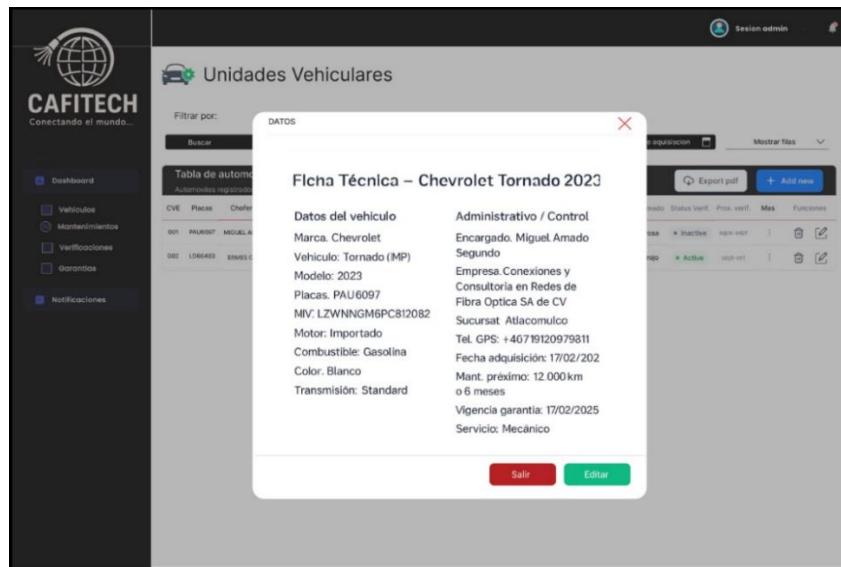


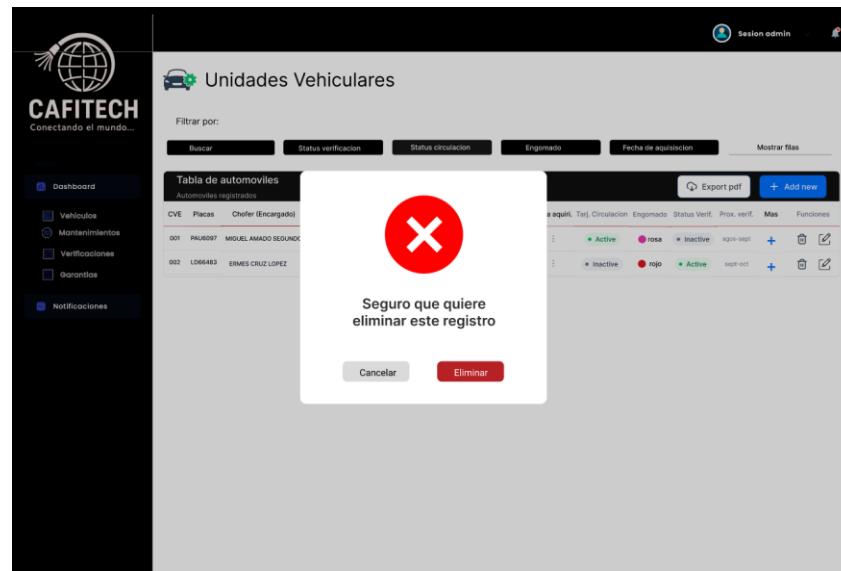
Figura. 11 Exportación de pdf  
(Elaboración propia)

Al solicitar los detalles de un vehículo, la Ficha Técnica se despliega en la plataforma CAFITECH, como se observa en la Figura 12. La interfaz principal está cubierta por un modal informativo titulado "Ficha Técnica – Chevrolet Tornado 2023". La información se distribuye en dos columnas: "Datos del vehículo", que incluye Marca, Modelo, Placas, Motor, Color y Transmisión; y "Administrativo / Control", donde se muestran el Encargado, la Empresa propietaria, la Sucursal, así como las fechas de adquisición, próximo mantenimiento y vigencia de garantía.



**Figura. 12 Modal de ver más información  
(Elaboración propia)**

La Figura. 13 ilustra la visualización del modal de confirmación al intentar eliminar un vehículo en CAFITECH. Este modal centrado presenta un ícono de error junto al mensaje de advertencia y ofrece dos botones: Cancelar para interrumpir la acción o Eliminar para completar la baja del registro.



**Figura. 13 Alerta de eliminación de vehículo  
(Elaboración propia)**

Al seleccionar la opción de editar un vehículo, se despliega el formulario de actualización, como se observa en la Figura. 14. La interfaz de CAFITECH queda cubierta por un modal

titulado "Actualizar vehículo", que contiene un formulario de doble columna mostrando los datos actuales del vehículo y del chofer. Se incluyen campos de identificación, detalles técnicos y datos administrativos, como Sucursal, Estatus de la tarjeta de circulación y Vigencia de la garantía.

The screenshot shows a web-based application interface for vehicle management. On the left, there's a sidebar with the CAFITECH logo and navigation links like Dashboard, Vehículos, Mantenimientos, Verificadores, Garantías, and Notificaciones. The main area has a title 'Unidades Ve' and a sub-section 'Actualizar vehículo'. A modal window is open, containing a form with two columns of fields. The left column includes fields for 'CVC del vehículo' (001), 'Nombre del empleado del chofer' (MIGUEL AMADO SEGUNDO), 'Marca' (CHEVROLET), 'Nombre del vehículo' (TORNADO (IMP)), 'Clase y tipo' (NIV), 'Motor' (C4), 'Tipo de transmisión' (ESTÁNDAR), 'Tipo de combustible' (GASOLINA), 'Color' (BLANCO), 'Número de teléfono gps' (+4619120979811), 'Sim gps' (89462038066007328795), 'UID' (8639405058346027), 'Propietario' (CONEXIONES Y CONSULTORIA EN REDES DE TELMA OPTICA SA DE CV), 'Folio de la tarjeta de circulación' (AU-C-17083239), 'Vigencia de la tarjeta de circulación' (10/10/2027), 'Status de la tarjeta de circulación' (ACTIVA), 'Mantenimiento por tiempo' (12 ó 6 meses), 'Status de adquisición' (comprado), 'Tipo de servicio' (Mecánico), 'Sucursal' (ATLACOMULCO), and 'Fecha de adquisición' (17/02/2022). The right column contains fields for 'Número de empleado del chofer' (001), 'Nombre del vehículo' (TORNADO (IMP)), 'Nombre del chofer' (MIGUEL AMADO SEGUNDO), 'Número de teléfono gps' (+4619120979811), 'Sim gps' (89462038066007328795), 'Propietario' (CONEXIONES Y CONSULTORIA EN REDES DE TELMA OPTICA SA DE CV), 'Vigencia de la tarjeta de circulación' (10/10/2027), 'Mantenimiento por tiempo' (12 ó 6 meses), 'Tipo de servicio' (Mecánico), and 'Fecha de adquisición' (17/02/2022). At the bottom of the modal are 'Salir' and 'Actualizar' buttons. The background shows a table titled 'Tabla de automóviles' with columns for CVE, Placas, Chofer (Encargado), and other vehicle details. The top right corner shows a user session status 'Sesión admin'.

**Figura. 14 Actualización de vehículo  
(Elaboración propia)**

La Figura. 15 evidencia la notificación de éxito tras la edición de un vehículo. Un modal centrado presenta un ícono de verificación y el mensaje “Actualización exitosa”, con el botón “ACEPTAR” para cerrar la alerta

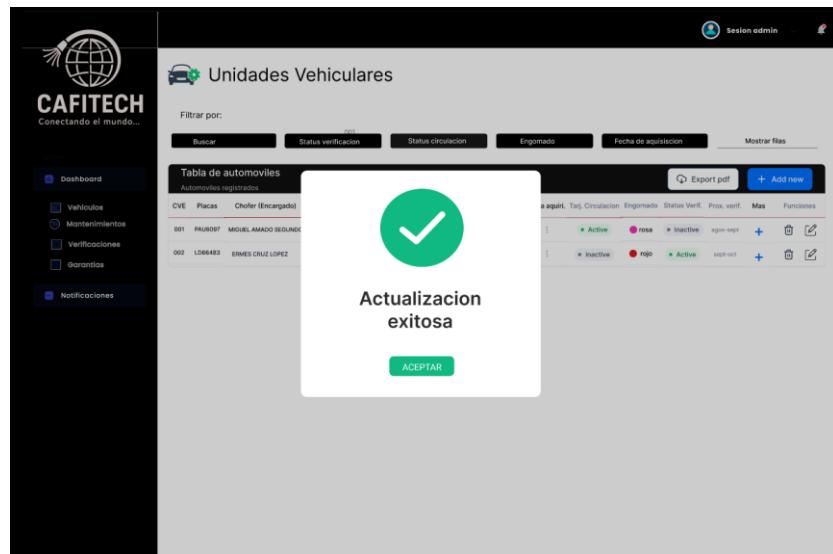


Figura. 15 Alerta de actualización  
(Elaboración propia)

Como se muestra en la siguiente Figura. 16, la imagen exhibe la sección de "Mantenimientos" de la plataforma CAFITECH. La pantalla conserva la barra de navegación lateral, donde el apartado "Mantenimientos" se encuentra activo. El área de contenido principal inicia con una barra de filtros para buscar por tipo de mantenimiento y fecha. Justo debajo, se visualiza una tabla de "Mantenimientos realizados" que enumera los servicios, detallando el vehículo, las fechas, el estatus (realizado o sin realizar) y ofrece funciones para exportar, añadir, eliminar o editar cada registro.

The screenshot shows the 'Mantenimientos' (Maintenance) section. It features a search bar and filter options for 'Tipo de mantenimiento' and 'Fecha'. Below is a table titled 'Mantenimientos realizados' (Completed Maintenance) with columns for CVE, Placas, Vehículo, Chofer (Encargado), Tipo de mantenimiento, Descripción, Fecha programada, Fecha realizada, Status, and Funciones. The table lists two entries: one for a preventive service on 10-10-15 marked as 'Sin Realizar' (Not Performed) and another for a corrective service on 10-10-15 marked as 'Realizado' (Performed).

Figura. 16 Apartado de mantenimientos  
(Elaboración propia)

En la Figura. 17 se observa la sección "Mantenimientos" de CAFITECH, donde se inicia el registro de un nuevo servicio. La interfaz está cubierta por el modal "Registro de nuevo Mantenimiento", que solicita datos como CVC del vehículo, Chofer, Tipo de mantenimiento (Preventivo), lugar de realización (Mecánico), Fechas programada y realizada, y una Descripción del mantenimiento (por ejemplo, "cambio de aceite").

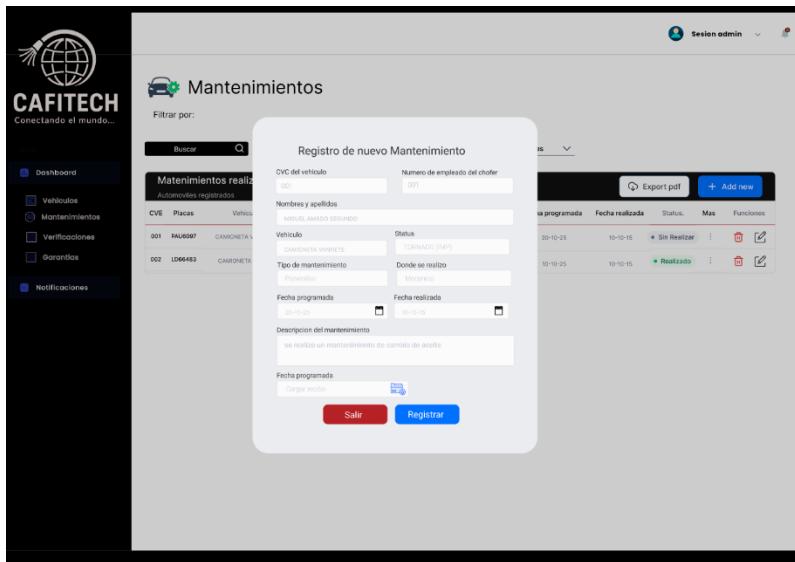


Figura. 17 Registro de un mantenimiento  
(Elaboración propia)

Como se visualiza en la Figura. 18, al finalizar el registro, se completa el mantenimiento y se muestra la notificación de éxito.

La interfaz de CAFITECH está cubierta por un modal de confirmación que presenta un gran ícono de verificación y el mensaje "Registro Exitoso". Para cerrar esta notificación y continuar, se utiliza el botón "ACEPTAR".

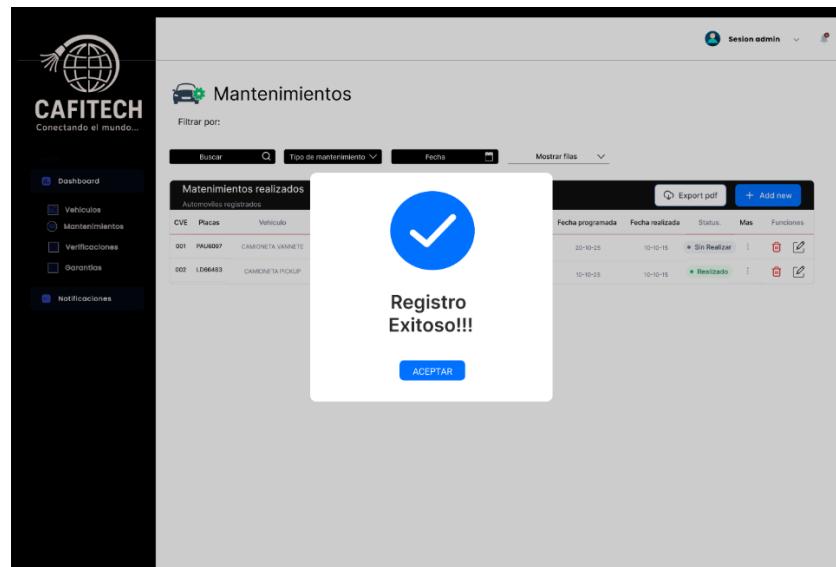


Figura. 18 registro de mantenimiento exitoso  
(Elaboración propia)

Como se puede ver en la Figura. 19, la plataforma CAFITECH despliega un modal centrado al exportar los mantenimientos, mostrando un ícono de descarga y el mensaje “PDF descargado correctamente”, que se cierra mediante el botón “ACEPTAR”.

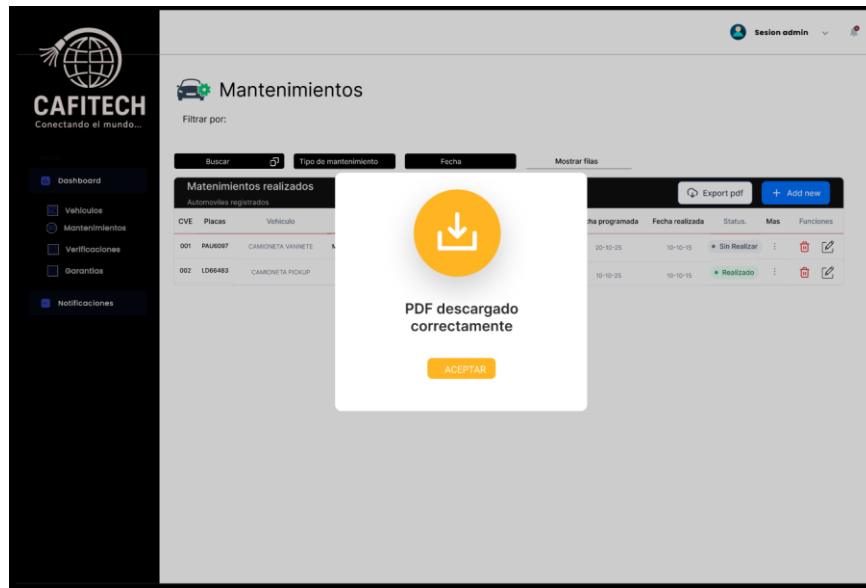


Figura. 19 Exportación pdf de mantenimientos  
(Elaboración propia)

Como se visualiza en la figura, al iniciar el proceso para eliminar un mantenimiento, se muestra la ventana de confirmación.

La interfaz de CAFITECH está cubierta por un modal de advertencia centrado en la pantalla. Este modal presenta un gran ícono de error y la pregunta de confirmación: "Seguro que quiere eliminar este registro". El usuario tiene dos opciones: el botón "Cancelar" (para abortar la acción) o el botón "Eliminar" (para confirmar la baja definitiva del registro).

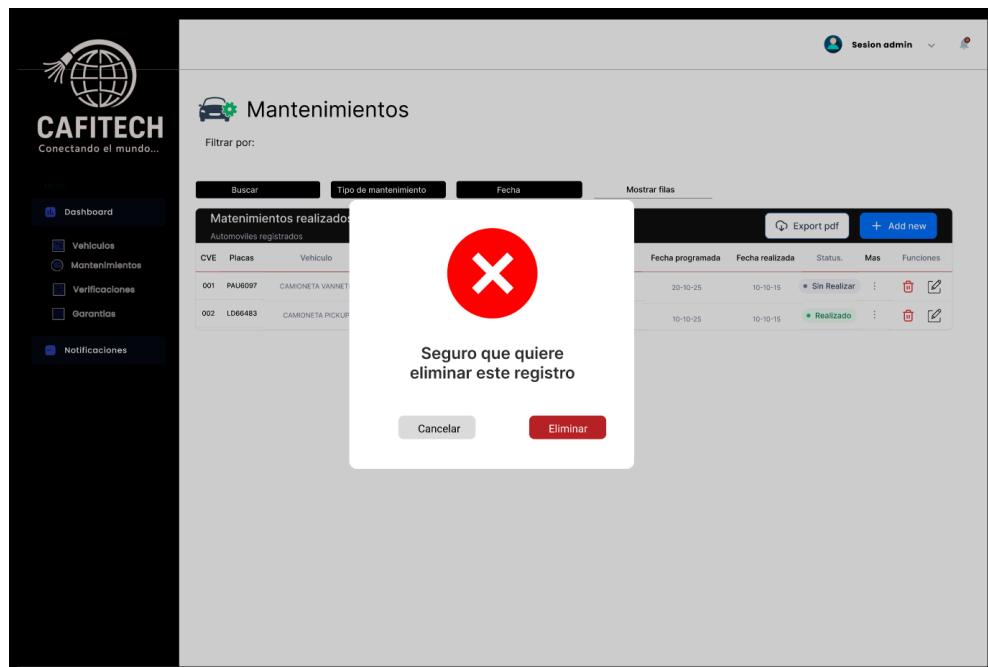


Figura. 20 Eliminación de mantenimientos  
(Elaboración propia)

Al seleccionar la opción de editar un mantenimiento, aparece el formulario para modificarlo, como se observa en la Figura. 21. La interfaz de CAFITECH queda cubierta por un modal titulado "Editar Mantenimiento", que presenta los datos actuales del vehículo y del servicio (CVC, Chofer, Tipo de mantenimiento y Ubicación) pre-cargados, incluyendo campos para Fechas programada y realizada y la Descripción del mantenimiento.

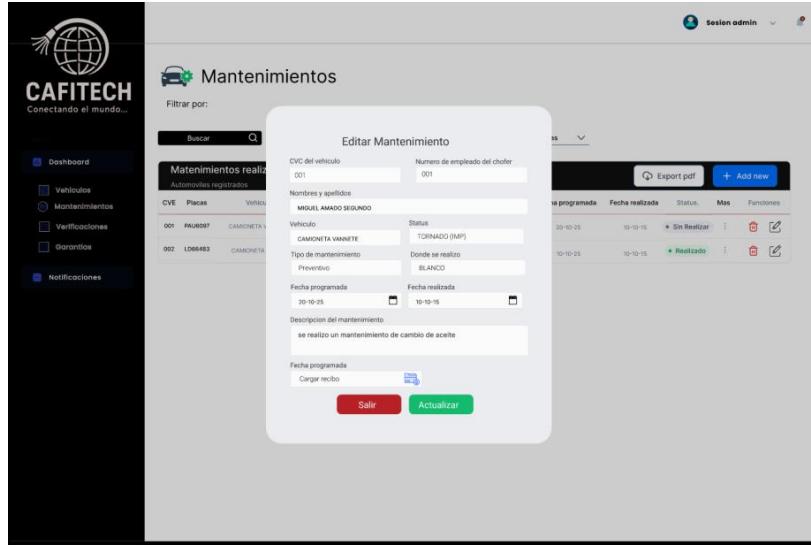


Figura. 21 Editar mantenimiento  
(Elaboración propia)

La Figura. 22 presenta la sección de "Verificaciones" en CAFITECH. La interfaz incluye el menú lateral con la opción activa de Verificaciones y el encabezado de sesión habitual. En el área principal se observa primero una barra de filtros para búsquedas por estado y periodo, seguida de la tabla de Verificaciones Vehiculares, que detalla los automóviles, su última verificación, estatus, periodos y ofrece opciones para exportar, añadir, editar o eliminar cada registro.

Figura. 22 Apartado de verificaciones  
(Elaboración propia)

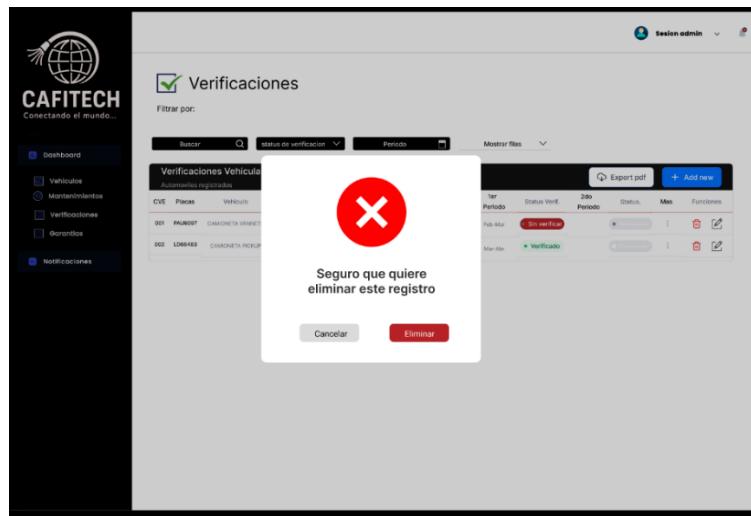
Como se aprecia, la Figura. 23 corresponde al apartado de "Verificaciones" de la plataforma CAFITECH, donde se inicia el proceso de registro de una verificación vehicular.

La interfaz está cubierta por el modal "Registro d Verificaciones". El formulario solicita datos como el CVC del vehículo, el Chofer, el Tipo de mantenimiento/verificación y dónde se realizó. También incluye campos para las Fechas programada y realizada, una descripción del servicio y una opción para subir un Comprobante digital.

The screenshot shows the CAFITECH platform's interface. On the left, there's a dark sidebar with navigation options: Dashboard, Vehículos, Mantenimientos, Verificaciones, Garantías, and Notificaciones. The main area has a title 'Verificaciones' with a checkmark icon. Below it is a search bar and a table titled 'Verificaciones Vehículo' showing two entries: 001 (PIU0097, CAMIONETA VAN) and 002 (LD66483, CAMIONETA P...). The central part of the screen is a modal window titled 'Registro d Verificaciones'. It contains several input fields: 'CVC del vehículo' (001), 'Número de empleado del chofer' (001), 'Nombres y apellidos' (MIGUEL ANGEL SERRANO), 'Vehículo' (CAMIONETA VAN), 'Status' (Sin verificar), 'Tipo de mantenimiento' (CAMBIO DE ACEITE), 'Fecha programada' (06-10-20), 'Fechas realizada' (06-10-19), 'Descripción del mantenimiento' (Se realizó un mantenimiento de cambio de aceite), and a 'Comprobante' section with a file upload button. At the bottom of the modal are 'Salir' and 'Registrar' buttons. To the right of the modal, there's a table with columns 'Status', 'Periodo', 'Status', and 'Funciones'. It shows two rows: one red row for 'Sin verificar' and one green row for 'Verificado'.

Figura. 23 registro de una nueva verificación  
(Elaboración propia)

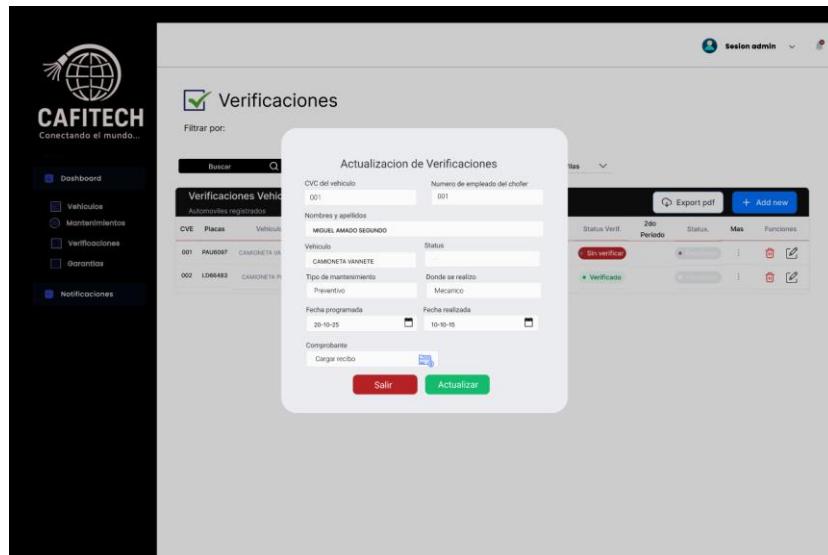
Al iniciar la eliminación de un registro de verificación, aparece la ventana de confirmación, como se observa en la Figura. 24. La interfaz de CAFITECH queda cubierta por un modal de advertencia centrado, con un ícono de error y la pregunta "Seguro que quiere eliminar este registro", ofreciendo los botones Cancelar y Eliminar.



**Figura. 24 Eliminación de una verificación  
(Elaboración propia)**

Como se aprecia en la Figura. 25, al seleccionar editar un registro de verificación, se muestra el formulario para su actualización.

La interfaz de CAFITECH está cubierta por un modal de edición titulado "Actualización de Verificaciones". El formulario presenta los datos actuales del vehículo y del servicio precargados para su modificación. Se incluyen campos para el CVC, el Chofer, el Tipo de mantenimiento/verificación, las Fechas programada y realizada, y una opción para cargar un Comprobante digital.



**Figura. 25 Actualización de verificaciones  
(Elaboración propia)**

La Figura. 26 presenta la sección "Garantías" de CAFITECH, con la opción activa en el menú lateral. El área principal incluye primero una barra de filtros para búsquedas por estado, tipo y fecha de vencimiento de la garantía, seguida de la tabla "Registro de garantías", que lista los vehículos con información de aseguradora, vigencia, estatus y botones para exportar, añadir, editar o eliminar registros.

The screenshot shows the CAFITECH application interface. On the left, there is a dark sidebar menu with the logo 'CAFITECH' and the tagline 'Conectando el mundo...'. The menu items include 'Dashboard', 'Vehículos', 'Mantenimientos', 'Verificaciones', 'Garantías' (which is highlighted in blue), and 'Notificaciones'. The main content area has a header 'Garantías' with a search icon and a 'Filtrar por:' dropdown. Below this is a toolbar with buttons for 'Buscar', 'status de garantía', 'Vencimiento', 'tipo garantía', 'Mostrar filas', 'Export.pdf', and '+ Add new'. The central part of the screen is titled 'Registro de garantías' and shows a table of registered vehicles. The table columns are: CVE, Placas, Vehículo, Marca, Modelo, NIV, Aseguradora, Tipo garantía, Suma asegurada, Inicio Vigencia, Prima, Status garantía, Coberturas y exclusiones, and Funciones. Two rows of data are visible:

CVE	Placas	Vehículo	Marca	Modelo	NIV	Aseguradora	Tipo garantía	Suma asegurada	Inicio Vigencia	Prima	Status garantía	Coberturas y exclusiones	Funciones	
001	PAU6097	CAMIONETA VANETTE	CHEVROLET	2023	LZWNWINGM6PC812082	Qualitas	Qualitas	2023	07-02-2025	07-02-2038	\$4,500	<span style="color:red">Reservado</span>		
002	1DE66483	CAMIONETA PICKUP	VOLKSWAGEN	2020	BBWK84SLBLP942729	Qualitas	Qualitas	2020	18-02-2025	18-02-2026	\$4,500	<span style="color:green">Activo</span>		

Figura. 26 Apartado de garantías  
(Elaboración propia)

La Figura. 27 presenta el inicio del registro de una nueva garantía vehicular en CAFITECH. La interfaz está cubierta por el modal "Registro de nueva Garantía", que solicita información como CVC del vehículo, Chofer, Aseguradora, Tipo de garantía y Prima, incluyendo campos para Inicio y Vigencia, y la opción de subir un Comprobante digital.

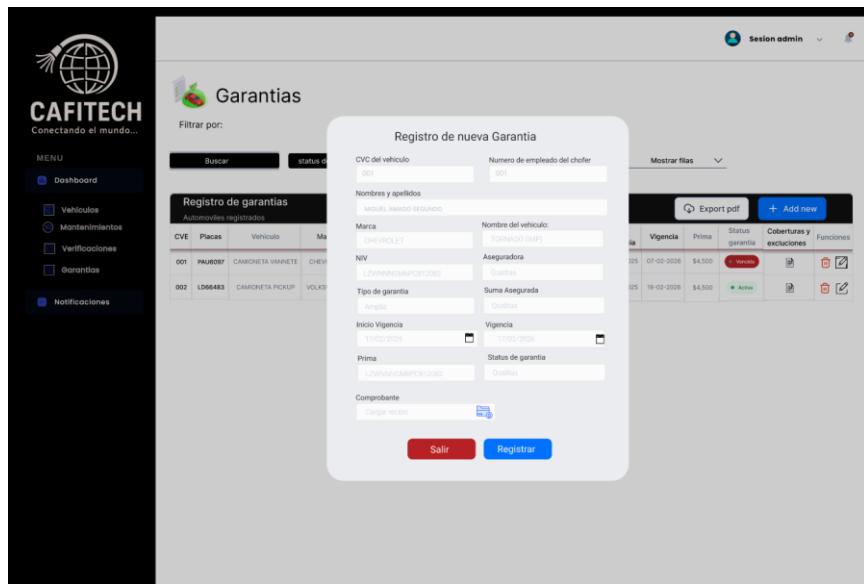


Figura. 27 inserción de una nueva garantía  
(Elaboración propia)

Como se muestra en la Figura. 28, al finalizar el registro, se completa la garantía y se despliega la notificación de éxito.

La interfaz de CAFITECH está cubierta por un modal de confirmación que presenta un gran ícono de verificación y el mensaje “¡¡¡Registro Exitoso!!!”. Para cerrar esta notificación y continuar, se utiliza el botón "ACEPTAR".

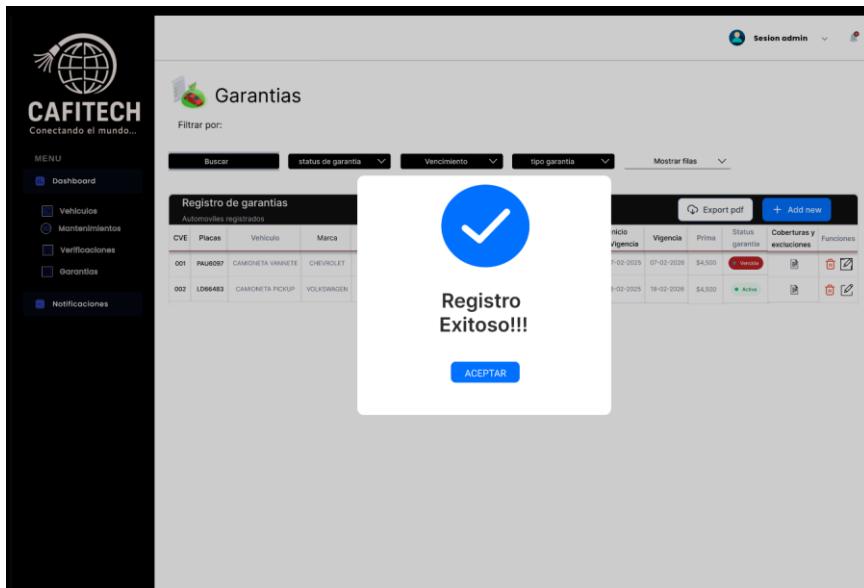


Figura. 28 Registro de garantías exitoso  
(Elaboración propia)

Al seleccionar editar una garantía, aparece el formulario para su actualización, como se observa en la Figura. 29. La interfaz de CAFITECH está cubierta por un modal titulado "Actualización de garantías", que muestra todos los datos existentes (Aseguradora, Tipo de garantía, Suma Asegurada y Prima) precargados, incluyendo Inicio, Vigencia y la opción de subir un Comprobante digital.

id	Vigencia	Prima	Status garantía	Coberturas y exclusiones	Funciones
125	07-02-2026	\$4,500	<span style="color: red;">Baja</span>	<span style="color: red;">Quilas</span>	<span style="color: red;">B</span> <span style="color: red;">D</span> <span style="color: red;">E</span>
126	18-02-2026	\$4,500	<span style="color: green;">Activa</span>	<span style="color: green;">Quilas</span>	<span style="color: green;">B</span> <span style="color: green;">D</span> <span style="color: green;">E</span>

Figura. 29 Modificación de las garantías  
(Elaboración propia)

Al finalizar la edición de un vehículo, se completa la actualización y se despliega la notificación de éxito, representada en la Figura. 30.

La interfaz de CAFITECH está cubierta por un modal de confirmación que presenta un gran ícono de verificación y el mensaje "Actualizacion exitosa". Para cerrar esta notificación y continuar, se utiliza el botón "ACEPTAR".

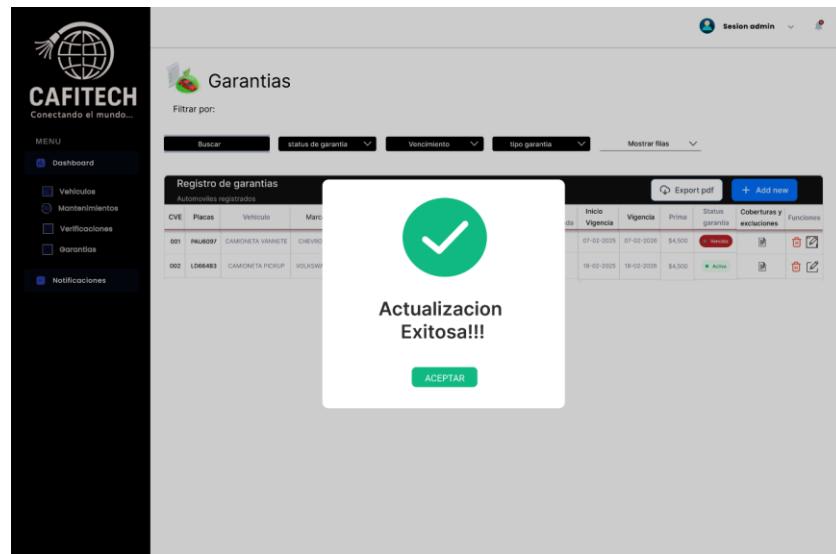


Figura. 30 Actualización exitosa de la garantía  
(Elaboración propia)

Al iniciar la eliminación de una garantía, aparece la ventana de confirmación, como se observa en la Figura. 31. La interfaz de CAFITECH queda cubierta por un modal de advertencia centrado, que incluye un ícono de error y la pregunta “Seguro que quiere eliminar este registro”, con los botones Cancelar y Eliminar para abortar o confirmar la acción.

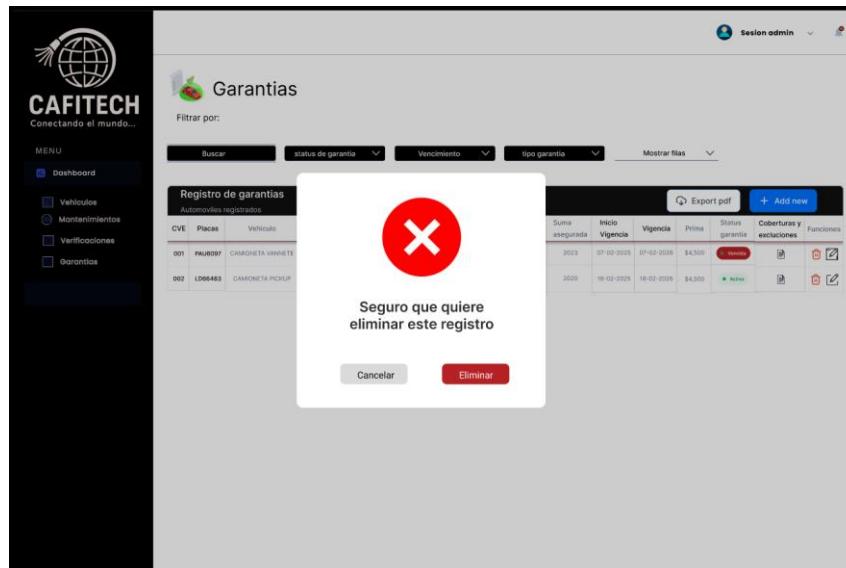


Figura. 31 Eliminación de garantías  
(Elaboración propia)

Como se muestra en la Figura. 32 corresponde a la sección "Notificaciones" de la plataforma CAFITECH, donde se visualiza el listado de mensajes del sistema.

La interfaz principal muestra el título "Mensajes" seguido de una barra de filtros que permite buscar, y filtrar por Tipo de mensaje y Fechas. El contenido principal es una Tabla de Notificaciones que enumera los mensajes del sistema.

Cada registro de la tabla detalla el origen de la notificación (Verificación, Mantenimiento, Garantías), el cuerpo del mensaje (ej. "El automóvil 003 está próximo..."), el Status (Relevante o Pendiente) y la Fecha y Hora de recepción. Al pie de la lista, se indica el Total de mensajes no leídos (Unread)

The screenshot shows the CAFITECH platform interface. On the left, there is a dark sidebar with the CAFITECH logo and the tagline 'Conectando el mundo...'. The sidebar includes navigation links for 'Dashboard', 'Vehículos', 'Mantenimientos', 'Verificaciones', 'Garantías', and 'Notificaciones'. The 'Notificaciones' link is highlighted. The main content area has a white background and features a title 'Mensajes' with icons for 'SMS' and 'Email'. Below the title is a search bar with placeholder 'Buscar' and a magnifying glass icon. To the right of the search bar are dropdown menus for 'Filtrar por:' (set to 'tipo de mensaje'), 'Fechas', and 'Mostrar filas' (set to '50'). A table lists seven notifications:

Origen	Mensaje	Status	Fecha/Hora
Verificación	El automóvil 003 está próximo.....	Relevante	Today , 5:50 pm
Mantenimiento	How is it going with new product ...	Pendiente	Today , 5:50 pm
Verificación	How is it going with new product ...	Relevante	Today , 5:50 pm
Garantías	How is it going with new product ...	Proxima	Today , 5:50 pm
Garantías	How is it going with new product ...	Proxima	Today , 5:50 pm
Mantenimiento	How is it going with new product ...	Pendiente	Today , 5:50 pm

At the bottom of the table, it says 'Total 55 Unread' and 'Sent : 95'. There is also a green button labeled 'Check other messages'.

Figura. 32 Apartado de notificaciones  
(Elaboración propia)

Al dar clic sobre cualquiera de estos mensajes, se despliega una ventana modal diferente que cambia su título, contenido y botones de acción dependiendo del tipo de mensaje seleccionado.

Por ejemplo:

Si el mensaje es de Mantenimiento, el modal se titula "Mantenimiento Mayor Pendiente" y ofrece los botones "Actualizar" y "Recordarme más tarde".

Si el mensaje es de Verificación, el modal también ofrece las opciones "Actualizar" y "Recordarme más tarde", pero con el texto de advertencia específico para el proceso de verificación, representado en la Figura. 33.

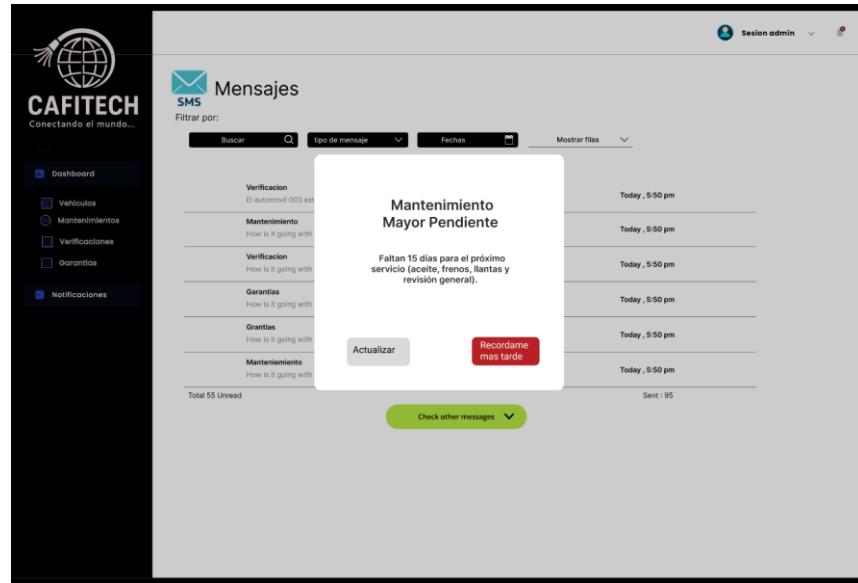


Figura. 33 Notificación correspondiente al mensaje  
(Elaboración propia)

Como se muestra, al darle "Actualizar" desde la notificación de mantenimiento, se despliega el formulario para modificar los datos del servicio pendiente.

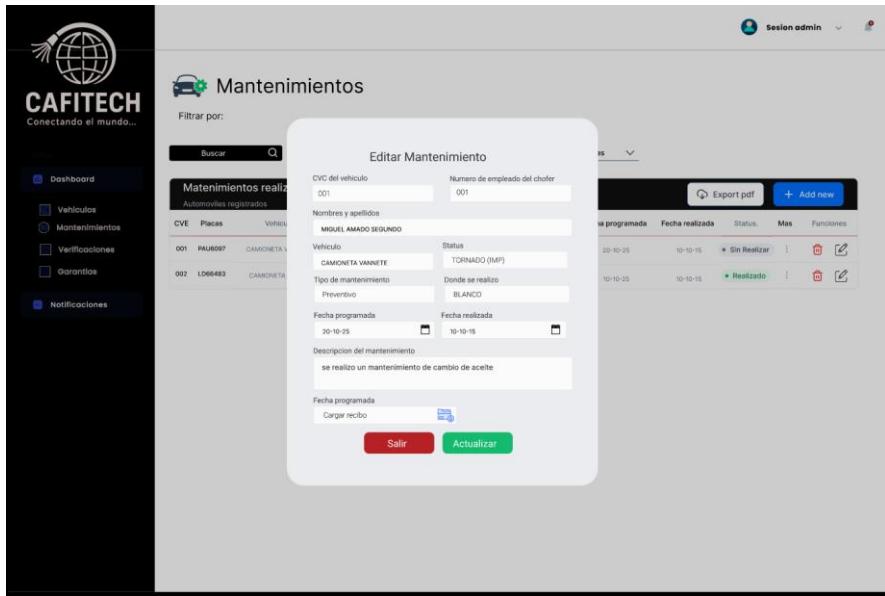


Figura. 34 Actualizar mantenimiento de la notificación  
(Elaboración propia)

Como se aprecia en la Figura. 35 de Mensajes (Notificaciones), al dar clic en el botón "Check other messages" o "Ver más mensajes" ubicado al final de la lista, el sistema cargará y mostrará los mensajes más antiguos que no se visualizan en la lista inicial.

Esta acción es un método de paginación o carga infinita que permite al usuario acceder al historial completo de notificaciones, mostrando más mensajes en la misma vista de la sección.

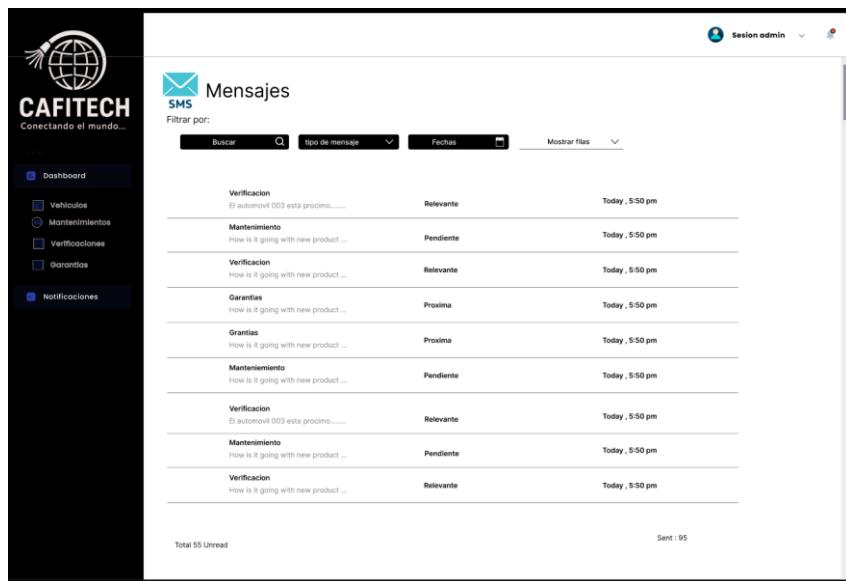


Figura. 35 Ver más mensajes  
(Elaboración propia)

### 3.4 Contraste de colores

El uso de un color de fondo negro combinado con texto en color blanco genera una proporción de contraste de 21:1, según la herramienta de Adobe Color. Este valor representa el nivel máximo de contraste, lo que asegura una legibilidad óptima(ver Figura. 36).



Figura. 36 Relación de contraste 21:1 entre colores negro y blanco.  
(Elaboración propia)

Como se muestra en la Figura. 37 el uso de un color de fondo gris combinado con texto en color negro genera una proporción de contraste de 15.44:1, según la herramienta de Adobe Color. Este valor indica un alto nivel de contraste, considerado óptimo para garantizar la legibilidad y una visualización cómoda en la página.



Figura. 37 Contraste entre fondo gris y texto negro (15.44:1).  
(Elaboración propia)

La Figura. 38 evidencia que un fondo negro combinado con texto en rojo alcanza un contraste de 5,67:1, medido con Adobe Color. Este nivel de contraste es útil para gráficos, pero no recomendable para lectura prolongada.



Figura. 38 Contraste entre fondo negro y texto rojo (5.67:1).  
(Elaboración propia)

## **CAPÍTULO IV: DESARROLLO DEL SISTEMA**

En este capítulo se presenta el proceso completo de desarrollo del sistema. Se describen las herramientas, tecnologías y metodologías utilizadas, así como la arquitectura del sistema, los módulos desarrollados y la interacción entre los distintos componentes. Además, se detallan los procedimientos para la gestión de datos, la interfaz de usuario y la integración de los distintos módulos, con el objetivo de garantizar un sistema funcional, eficiente y acorde con los requerimientos establecidos.

## 4.1 Requisitos previos

Antes de proceder con el desarrollo e implementación del sistema, es necesario contar con las siguientes herramientas instaladas en el equipo:

- **Python** – Lenguaje de programación utilizado para el backend con Flask.
- **Node.js** – Entorno de ejecución necesario para trabajar con React y Vite en el frontend.

### 4.1.1 Requisitos para despliegue

Requisitos de hardware (máquina virtual)

- **CPU:** 2 núcleos
- **Memoria RAM:** 4 GB (mínimo recomendado)
- **Almacenamiento:** 40 GB
- **Arquitectura:** x86\_64
- **Red:** Bridge con IP fija o reservada

Requisitos de software

- Requisitos de software
- Hipervisor: Proxmox VE
- Sistema operativo: Ubuntu Server 22.04 LTS
- Servidor web: Nginx
- Backend:
  - Python 3
  - Flask
  - Gunicorn
- Frontend:
  - Node.js LTS
  - React + Vite (compilado a archivos estáticos)
  - Base de datos (opcional): MySQL o PostgreSQL
  - Acceso remoto: OpenSSH
- Todo el software utilizado es open source.

Costos estimados del despliegue

Componente	Costo
Proxmox VE	\$0 MXN
Ubuntu Server	\$0 MXN
Flask	\$0 MXN
React	\$0 MXN
Nginx	\$0 MXN
Gunicorn	\$0 MXN
MySQL / PostgreSQL	\$0 MXN

## 4.2 Instalación de MySQL

Para gestionar la base de datos del sistema es necesario contar con **MySQL** instalado en el equipo. El primer paso consiste en descargar el instalador desde la página oficial de MySQL:

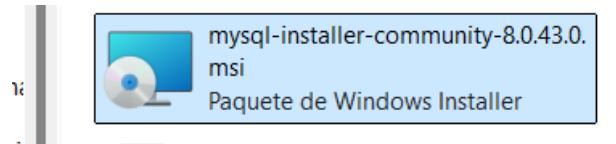
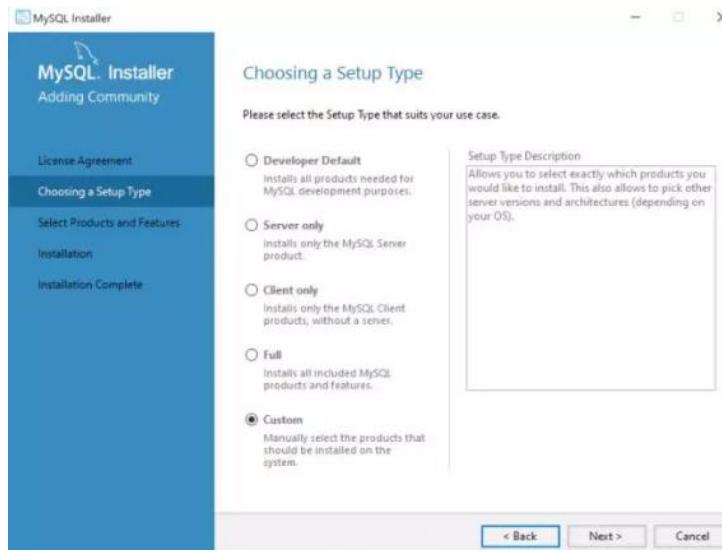


Figura. 39 Instalador de MySQL  
(Elaboración propia)

Al abrir el instalador de MySQL, se desplegará una ventana en la que se debe seleccionar el tipo de instalación. En este caso, es recomendable elegir la opción Custom, ya que permite personalizar los componentes que se instalarán, ajustando el entorno a las necesidades específicas del proyecto(ver Figura.40).



**Figura. 40 Tipo de instalación  
(Elaboración propia)**

Durante el proceso de instalación, es posible que se te presente una lista de software adicional requerido para completar la instalación de MySQL. Estas dependencias pueden variar según las aplicaciones ya instaladas en tu sistema y las características de MySQL que hayas seleccionado para instalar.

Para proceder, simplemente haz clic en "Execute" y el instalador se encargará de descargar e instalar automáticamente las dependencias necesarias, como se muestra en Figura. 41. Asegúrate de tener una conexión a Internet estable durante este proceso, ya que el instalador necesitará descargar las dependencias adicionales desde los servidores correspondientes. Una vez completada la instalación de las dependencias, podrás continuar con el proceso de configuración y personalización de MySQL según tus necesidades específicas.

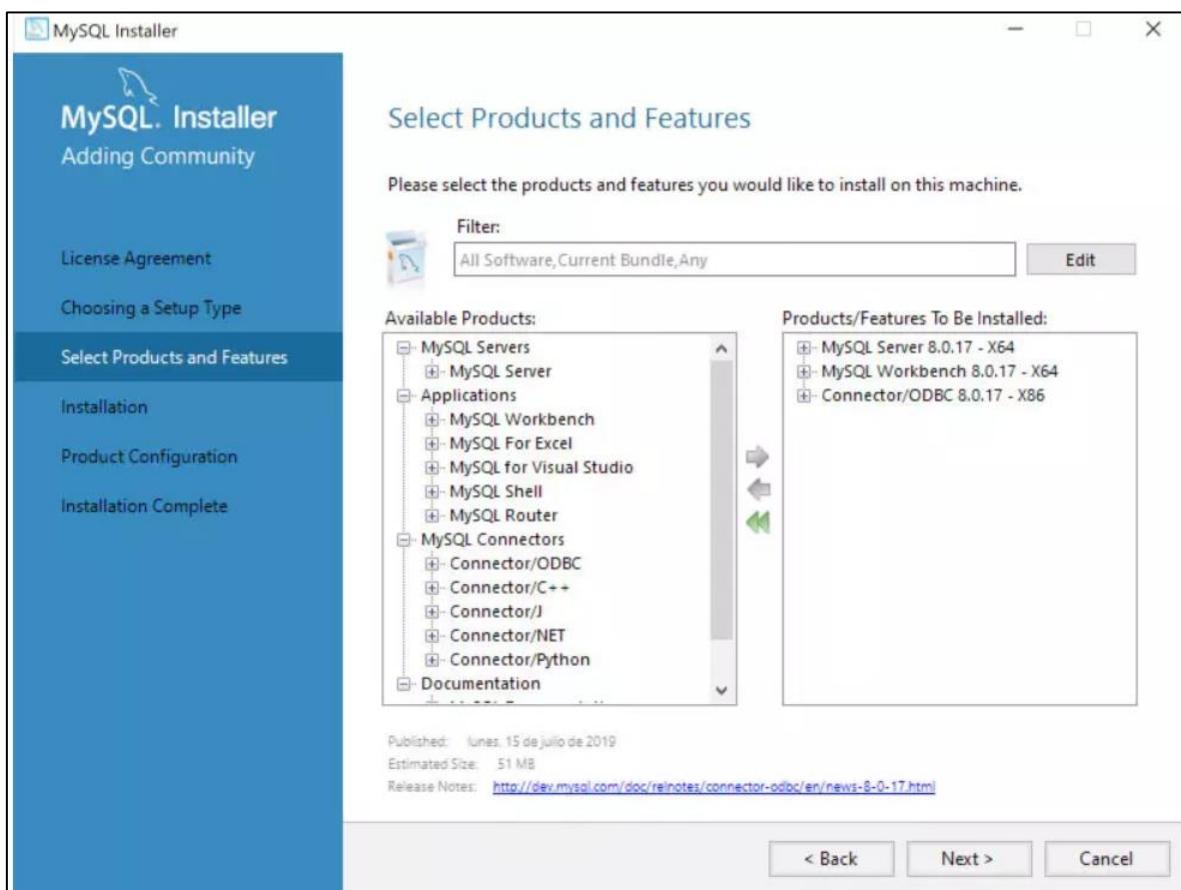


Figura. 41 Instalación de dependencias adicionales  
(Elaboración propia)

### 4.3 Creación de un proyecto Frontend React

En esta etapa se creó el proyecto Frontend con React para establecer la estructura base de la aplicación. Se configuraron los componentes iniciales, las rutas principales y los estilos generales que servirán como punto de partida para el desarrollo del sistema.

#### 4.3.1 Creación del proyecto React con Vite

La interfaz de usuario se gestionará desde un proyecto React con Vite, colocado como proyecto de nivel superior. Para ello, abra una ventana de terminal, sitúese en el directorio principal donde trabajará el sistema y ejecute el siguiente comando:

➤ `npm create vite@latest frontend`

#### **4.3.2 Acceso a la carpeta del frontend**

Una vez creado el proyecto, se debe acceder a la carpeta del frontend utilizando el siguiente comando en la terminal:

➤ `cd frontend`

Esto permite trabajar directamente dentro de la carpeta del frontend, donde se encuentran archivos esenciales como `package.json` y la carpeta `src/`, necesarios para la gestión del proyecto.

#### **4.3.3 Instalación de dependencias**

Una vez dentro de la carpeta del proyecto, se procede a instalar las dependencias necesarias para el correcto funcionamiento del entorno React. Esto se realiza ejecutando el siguiente comando en la terminal:

➤ `npm install`

Este comando descarga todas las librerías y módulos listados en `package.json` e instala los recursos necesarios en `node_modules`, garantizando que la aplicación tenga todos los elementos requeridos para funcionar correctamente.

y corremos el servidor

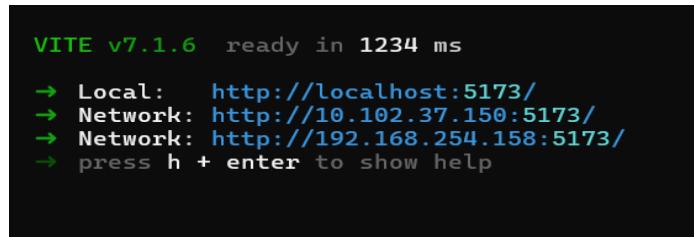
#### **4.3.4 Ejecución del servidor de desarrollo**

Inicie el servidor con:

➤ `npm run dev`

Esto activa el entorno de desarrollo proporcionado por Vite, levantando un servidor local (generalmente en `http://localhost:5173`) donde se puede visualizar la aplicación en tiempo real y probar los cambios durante el desarrollo.

El comando `npm run dev` inicia el servidor de desarrollo de Vite, lo que permite ejecutar la aplicación localmente. En la terminal, se mostrará información sobre la dirección local del servidor y otros detalles del entorno de desarrollo.



```
VITE v7.1.6 ready in 1234 ms
→ Local: http://localhost:5173/
→ Network: http://10.102.37.150:5173/
→ Network: http://192.168.254.158:5173/
→ press h + enter to show help
```

Figura. 42 Inicialización del servidor.  
(Elaboración propia)

Además, puede presionar `h` y luego `Enter` para ver la lista de comandos disponibles. Uno de los más útiles es `o + Enter`, que abre automáticamente la aplicación en su navegador predeterminado, mostrando la interfaz inicial del proyecto React.

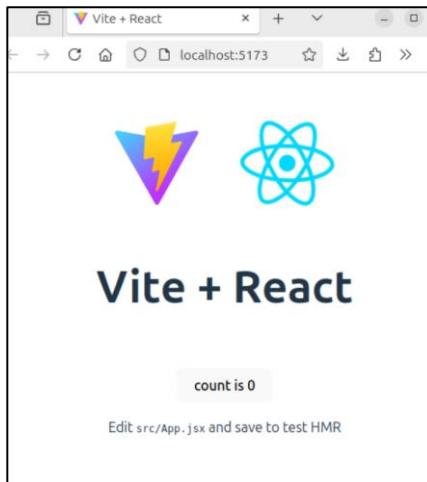


Figura. 43 Interfaz inicial del proyecto  
(Elaboración propia)

#### 4.4 Creación del proyecto Backend

En esta fase se desarrolló el proyecto Backend, encargado de gestionar la lógica del sistema y la comunicación con la base de datos. Se creó el entorno de trabajo utilizando Flask, configurando las rutas, controladores y modelos necesarios para el manejo de la información.

#### 4.4.1 Creación de la carpeta del backend

Fuera de la carpeta del frontend, se crea la carpeta donde se desarrollará el backend ejecutando:

```
➤ mkdir backend
```

Esto genera un directorio llamado backend que contendrá todos los archivos y configuraciones necesarias para el servidor y la lógica del sistema.

#### 4.4.2 Acceso a la carpeta del backend

Ingrese al directorio recién creado con:

```
➤ cd backend
```

Esto permite trabajar directamente dentro de la carpeta del backend, asegurando que todos los comandos y configuraciones se apliquen correctamente en este proyecto independiente del frontend.

#### 4.4.3 Creación de un entorno virtual en Python

Para el desarrollo del backend, es importante usar un **entorno virtual**, ya que permite instalar y gestionar las librerías necesarias de manera independiente del resto del sistema.

El entorno virtual se recomienda crear dentro del subdirectorio api del backend, de modo que todos los archivos y dependencias queden organizados en un solo lugar y sean fáciles de localizar y administrar. En la terminal, estando dentro del directorio correspondiente, se ejecuta el siguiente comando:

```
➤ python -m venv venv
```

Primero, dentro del entorno virtual creado, se instalan las librerías necesarias para nuestro servidor Flask y para manejar variables de entorno con el siguiente comando:

```
➤ (venv) $ pip install flask python-dotenv
```

## 4.5 Configuración de la base de datos con MySQL y conexión al backend

En esta etapa se realizó la configuración de la base de datos MySQL y su conexión con el proyecto Backend. Se creó la estructura de tablas necesaria para almacenar la información del sistema y se estableció la conexión mediante las credenciales del servidor en el archivo de configuración de Flask.

### 4.5.1 Creación de la base de datos en MySQL Workbench

Para gestionar la información del sistema, se utilizará MySQL como motor de base de datos. La base de datos puede crearse de manera gráfica o mediante comandos SQL en MySQL Workbench.

**Pasos para crear la base de datos mediante MySQL Workbench:**

1. **Abrir MySQL Workbench** y conectarse al servidor local con las credenciales correspondientes.
2. **Abrir una nueva pestaña de consultas SQL** para ejecutar los comandos de creación de la base de datos.
3. **Crear la base de datos** utilizando el siguiente comando:
  - `CREATE DATABASE control_vehicular;`

### 4.5.2 Creación de las tablas MySQL

Una vez creada la base de datos, se procede a la definición de las tablas que almacenarán la información del sistema. Estas tablas fueron diseñadas considerando las relaciones entre usuarios, vehículos, verificaciones y registros históricos, y se muestran en la siguiente imagen (donde puede observarse la estructura) como se muestra en la figura 41.

2025-10-16	16:58:10 CREATE TABLE HistorialAsignaciones ( `id_historial` INT AUTO_INCREMENT PRIMARY KEY, `id_asignacion` INT NOT NULL, `fecha_cambio` TIMESTAMP DEFAULT CURRENT_TIMESTAMP, ...
2025-10-15	16:58:10 CREATE TABLE HistorialPlacas ( `id_historial` INT AUTO_INCREMENT PRIMARY KEY, `id_placa` INT NOT NULL, `fecha_cambio` TIMESTAMP DEFAULT CURRENT_TIMESTAMP, `folio` VARCHAR(50), ...
2025-10-14	16:58:10 CREATE TABLE HistorialGarantias ( `id_historial` INT AUTO_INCREMENT PRIMARY KEY, `id_garantia` INT NOT NULL, `fecha_cambio` TIMESTAMP DEFAULT CURRENT_TIMESTAMP, `aseguradora` VARCHAR(50), ...
2025-10-13	16:58:10 CREATE TABLE VerificacionVehicular ( `id_verificacion` INT AUTO_INCREMENT PRIMARY KEY, `id_unidad` INT NOT NULL, `ultima_verificacion` DATE, `periodo_1` DATE, `periodo_1_real` DATE, ...
2025-10-10	16:58:10 CREATE TABLE FallasMecanicas ( `id_falla` INT AUTO_INCREMENT PRIMARY KEY, `id_unidad` INT NOT NULL, `fecha_falla` DATE NOT NULL, `id_pieza` INT NOT NULL, `id_marca` INT, `tipo` VARCHAR(50), ...
2025-10-08	16:58:10 CREATE TABLE MarcasPiezas ( `id_marca` INT AUTO_INCREMENT PRIMARY KEY, `nombre_marca` VARCHAR(100) NOT NULL UNIQUE, `pais_origen` VARCHAR(100), `observaciones` TEXT, INDEX idx_nombre_pieza (nombre_pieza))
2025-10-07	16:58:10 CREATE TABLE Piezas ( `id_pieza` INT AUTO_INCREMENT PRIMARY KEY, `nombre_pieza` VARCHAR(100) NOT NULL UNIQUE, `descripcion` TEXT, INDEX idx_nombre_pieza (nombre_pieza))
2025-10-06	16:58:10 CREATE TABLE Mantenimientos ( `id_mantenimiento` INT AUTO_INCREMENT PRIMARY KEY, `id_mantenimiento_programado` INT, `id_unidad` INT NOT NULL, `tipo_mantenimiento` VARCHAR(50), ...)
2025-10-02	16:58:10 CREATE TABLE MantenimientosProgramados ( `id_mantenimiento_programado` INT AUTO_INCREMENT PRIMARY KEY, `id_unidad` INT NOT NULL, `id_tipo_mantenimiento` INT NOT NULL, `fecha_programacion` DATE, ...)
2025-09-30	16:58:10 CREATE TABLE TiposMantenimiento ( `id_tipo_mantenimiento` INT AUTO_INCREMENT PRIMARY KEY, `nombre_tipo` VARCHAR(50) NOT NULL UNIQUE, `descripcion` TEXT, `frecuencia_temporizada` VARCHAR(50), ...)
2025-09-29	16:58:10 CREATE TABLE Asignaciones ( `id_asignacion` INT AUTO_INCREMENT PRIMARY KEY, `id_chofor` INT NOT NULL, `id_unidad` INT NOT NULL, `fecha_asignacion` DATE NOT NULL, `fecha_fin` DATE, ...)
2025-09-25	16:58:10 CREATE TABLE Placas ( `id_placa` INT AUTO_INCREMENT PRIMARY KEY, `id_unidad` INT NOT NULL, `folio` VARCHAR(50), `placa` VARCHAR(10) NOT NULL UNIQUE, `fecha_expedicion` DATE, ...)
2025-09-24	16:58:10 CREATE TABLE Garantias ( `id_garantia` INT AUTO_INCREMENT PRIMARY KEY, `id_unidad` INT NOT NULL, `aseguradora` VARCHAR(255), `tipo_garantia` VARCHAR(100), `no_poliza` VARCHAR(100), ...)
2025-09-23	16:58:10 CREATE TABLE Unidades ( `id_unidad` INT AUTO_INCREMENT PRIMARY KEY, `marca` VARCHAR(100), `vehiculo` VARCHAR(100), `modelo` INT, `clase_tipo` VARCHAR(50), `niv` VARCHAR(50), ...)
2025-09-22	16:58:10 CREATE TABLE Choferes ( `id_chofor` INT AUTO_INCREMENT PRIMARY KEY, `nombre` VARCHAR(255) NOT NULL, `cup` VARCHAR(18) NOT NULL UNIQUE, `calle` VARCHAR(255), `colonia` VARCHAR(255), ...)

Figura. 44 Inserción de tablas a la base de datos  
(Elaboración propia)

#### 4.5.3 Conexión del backend con la base de datos mediante SQLAlchemy

Para establecer la conexión entre el backend desarrollado en Flask y la base de datos MySQL, se utiliza SQLAlchemy como ORM (Object Relational Mapping), lo cual permite manipular las tablas como modelos de Python y facilita la gestión de consultas y transacciones.

```
app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv(
    'DATABASE_URL',
    'mysql+pymysql://root:admin@localhost/control_vehicular1'
)
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
```

La cadena de conexión utiliza el driver pymysql y especifica usuario, contraseña, host y nombre de la base de datos. SQLAlchemy gestiona la conexión y posteriormente permitirá mapear las tablas mediante modelos.

#### 4.5.4 Definición de modelos con SQLAlchemy

Una vez configurada la conexión con la base de datos, se definen los modelos que representan cada una de las tablas creadas en MySQL. Como se muestra en la Figura. 45

estos modelos permiten manipular los registros mediante objetos Python, facilitando la creación, consulta, modificación y eliminación de datos.

```
class Usuarios(db.Model):
    __tablename__ = 'Usuarios'
    id_usuario = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(255), nullable=False)
    usuario = db.Column(db.String(50), unique=True, nullable=False)
    contraseña = db.Column(db.String(255), nullable=False)
    correo = db.Column(db.String(255), unique=True, nullable=False)
    rol = db.Column(db.String(50), nullable=False, default='usuario')
    fecha_registro = db.Column(db.TIMESTAMP, default=datetime.utcnow)
    fecha_ultimo_login = db.Column(db.TIMESTAMP, nullable=True)
    estado = db.Column(db.Enum('activo', 'inactivo'), default='activo')
    token_recuperacion = db.Column(db.String(255), nullable=True)
    token_expiracion = db.Column(db.TIMESTAMP, nullable=True)

class Unidades(db.Model):
    __tablename__ = "Unidades"
    id_unidad = db.Column(db.Integer, primary_key=True)
    marca = db.Column(db.String(100))
    vehiculo = db.Column(db.String(100))
    modelo = db.Column(db.Integer)
    clase_tipo = db.Column(db.String(50))
    niv = db.Column(db.String(50), unique=True)
    motor = db.Column(db.String(50))
    transmision = db.Column(db.String(50))
    combustible = db.Column(db.String(50))
    color = db.Column(db.String(50))
    telefono_gps = db.Column(db.String(20))
    sim_gps = db.Column(db.String(20))
    uid = db.Column(db.String(50))
    propietario = db.Column(db.String(255))
    sucursal = db.Column(db.String(100))
    compra_arrendado = db.Column(db.String(20))
    fecha_adquisicion = db.Column(db.Date)
```

Figura. 45 Definición de los modelos  
(Elaboración propia)

## 4.6 Implementación del sistema de autenticación en el Backend

El proceso de autenticación se implementa mediante un endpoint /login, el cual valida las credenciales enviadas desde el frontend. Este proceso utiliza SQLAlchemy para consultar la base de datos y Werkzeug para verificar contraseñas cifradas, garantizando seguridad en el manejo de credenciales.

Cuando el usuario envía su nombre de usuario y contraseña, el backend realiza las siguientes acciones:

1. Verifica que los campos no estén vacíos.
2. Busca en la base de datos un usuario activo con el nombre proporcionado.
3. **Compara la contraseña ingresada con el hash almacenado utilizando check\_password\_hash.**
4. Si las credenciales son correctas, actualiza la fecha del último inicio de sesión y retorna los datos básicos del usuario.
5. En caso contrario, devuelve un mensaje de error con código 401 Unauthorized.

A continuación, en la Figura. 46 se muestra el fragmento principal del endpoint implementado:

```
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')

    app.logger.info(f"Intento de inicio de sesión para el usuario: {username}")

    if not username or not password:
        app.logger.warning("No se proporcionó usuario o contraseña.")
        return jsonify({"error": "Usuario y contraseña son requeridos"}), 400

    user = Usuarios.query.filter_by(usuario=username, estado='activo').first()

    if user:
        app.logger.info(f"Usuario encontrado: {user.usuario}, contraseña almacenada: {user.contraseña}")
        if check_password_hash(user.contraseña, password):
            user.fecha_ultimo_login = datetime.utcnow()
            db.session.commit()
            return jsonify({
                "message": "Inicio de sesión exitoso",
                "user": {"id": user.id_usuario, "username": user.usuario, "nombre": user.nombre, "rol": user.rol}
            }), 200
        else:
            app.logger.warning("Contraseña inválida proporcionada.")
    else:
        app.logger.info("Usuario no encontrado o inactivo.")

    return jsonify({"error": "Credenciales inválidas o usuario inactivo"}), 401
```

Figura. 46 Endpoint /login  
(Elaboración propia)

#### 4.6.1 Recuperación de contraseña

El sistema incluye un mecanismo seguro para que los usuarios puedan recuperar su contraseña en caso de olvido. Este proceso utiliza **tokens temporales** enviados por correo electrónico y garantiza que las contraseñas se almacenen cifradas como se muestra en la figura .

#### Flujo de recuperación

1. El usuario accede a la opción "¿Olvidaste tu contraseña?" en el frontend (/request-reset).
2. Ingresa su correo electrónico registrado en el sistema.
3. El backend genera un **token seguro** (secrets.token\_urlsafe(32)) con expiración de **1 hora** y lo almacena junto al usuario en la base de datos.
4. Se envía un correo con un enlace al usuario para restablecer su contraseña. El enlace contiene el token generado.
5. El usuario hace clic en el enlace y es dirigido al formulario de restablecimiento (/reset-password/:token).
6. El usuario ingresa la nueva contraseña, que se envía al backend.
7. El backend valida que el token sea válido y no haya expirado, actualiza la contraseña usando generate\_password\_hash, y elimina el token de la base de datos.

8. El usuario recibe un mensaje de confirmación sobre la actualización exitosa (ver Figura 47).

```

@app.route('/request-reset', methods=['POST'])
def request_password_reset():
    """Maneja la solicitud de recuperación de contraseña."""
    data = request.get_json()
    email = data.get('email')

    if not email:
        return jsonify({"error": "Correo electrónico requerido"}), 400

    user = Usuarios.query.filter_by(correo=email).first()
    if user:
        # Generar el token de recuperación
        token = secrets.token_urlsafe(32)
        user.token_recuperacion = token
        user.token_expiracion = datetime.utcnow() + timedelta(hours=1)
        db.session.commit()

        try:
            # Crear el enlace de restablecimiento
            #reset_link = f'{os.getenv("FRONTEND_URL", "http://192.168.254.158:5173")}/reset-password/{frontend_url}={http://192.168.254.158:5173}'
            reset_link = f'{Frontend_url}/reset-password/{token}'

            msg = Message(
                'Restablecimiento de Contraseña',
                sender=app.config['MAIL_USERNAME'],
                recipients=[email]
            )
        
```

Figura. 47 Recuperación de contraseña  
(Elaboración propia)

#### 4.6.2 Seguridad en el manejo de sesiones y tokens

El sistema implementa medidas de seguridad para proteger tanto las **sesiones activas** como los **tokens de recuperación de contraseña**, garantizando la integridad de la información y evitando accesos no autorizados.

##### 4.6.2.1 Autenticación segura

- Las contraseñas se almacenan mediante **hashing seguro** (generate\_password\_hash).
- Durante el login, las contraseñas se comparan con check\_password\_hash.
- Solo los usuarios con estado activo (estado='activo') pueden autenticarse.

##### 4.6.2.2 Prevención de SQL Injection

- Todas las consultas a la base de datos se realizan usando **SQLAlchemy ORM**, evitando concatenar valores ingresados por el usuario.

Ejemplo en login

- `user = Usuarios.query.filter_by(usuario=username, estado='activo').first()`
- Esto previene que un atacante pueda ejecutar código SQL malicioso mediante el campo de usuario o contraseña.

#### 4.6.2.3 Gestión de tokens de recuperación

- Los tokens se generan con secrets.token\_urlsafe(32) y exirpan en **1 hora**.
- Solo el usuario asociado puede usar el token para restablecer la contraseña.
- Tras ser usado o exirpar, el token se elimina de la base de datos, evitando reutilización indebida.

#### 4.6.2.4 Seguridad en el frontend

- La sesión se mantiene en localStorage mediante usuarioid y isLoggedIn.
- Se implementa **auto logout por inactividad** (1 hora) monitorizando eventos de actividad (mousemove, keydown, scroll, click).
- Las rutas internas de React están protegidas y la redirección se realiza según el rol del usuario, evitando acceso a secciones no autorizadas.

#### 4.6.2.5 Consideraciones adicionales

- Mensajes de error no revelan si un correo o usuario existe en el sistema, protegiendo la información de los usuarios.
- Se recomienda usar **HTTPS** en producción para proteger los datos en tránsito.
- Mantener actualizadas las librerías de Flask, React y SQLAlchemy ayuda a evitar vulnerabilidades conocidas.

### 4.7 Manejo de sesión en el Frontend e integración con el Backend

El frontend de la aplicación, desarrollado en React, se conecta con el backend para autenticar usuarios y gestionar la sesión activa. La sesión se mantiene mediante localStorage y permite controlar el acceso a rutas protegidas según el rol del usuario.

#### 4.7.1 Consumo del endpoint /login

Cuando un usuario envía sus credenciales, el componente Login realiza una petición POST al endpoint /login del backend. El backend valida usuario y contraseña, y devuelve los datos básicos del usuario si las credenciales son correctas (ver Figura. 48).

```
const response = await fetch(`${API_URL}/login`, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({ username, password }),
});
```

Figura. 48 Consumo del endpoint /login  
(Elaboración propia)

#### 4.7.2 Persistencia de la sesión

Si la autenticación es exitosa, el usuarioid y el indicador isLoggedIn se almacenan en localStorage. Esto permite mantener la sesión activa incluso tras recargar la página como se muestra en la Figura. 49.

```
// Guardar usuarioid en localStorage
const usuarioid = data.user.id;
localStorage.setItem("usuarioid", usuarioid);
localStorage.setItem("isLoggedIn", "true");
```

Figura. 49 Persistencia de la sesión  
(Elaboración propia)

#### 4.7.3 Redirección por rol

En la Figura. 46 se visualiza que dependiendo del rol del usuario (admin o chofer), el sistema redirige automáticamente a la ruta correspondiente:

- Admin → /admin/solicitudes
- Chofer → /chofer/solicitudes

```
// Redirigir según el rol
if (data.user.rol === "admin") {
  navigate("/admin/solicitudes");
} else {
  navigate("/chofer/solicitudes");
}
```

Figura. 50 Redirección de roles  
(Elaboración propia)

#### 4.7.4 Cierre automático por inactividad

Se implementa un auto logout después de 1 hora de inactividad, monitoreando eventos como mousemove, keydown, scroll y click. Al superar este tiempo, el usuario es desconectado y se muestra una notificación (ver Figura. 46).

```

// === AUTO LOGOUT POR INACTIVIDAD ===
useEffect(() => {
  if (!isLoggedIn) return;

  const AUTO_LOGOUT_TIME = 60 * 60 * 1000; // 20 minutos
  let logoutTimer;

  const resetTimer = () => {
    clearTimeout(logoutTimer);
    logoutTimer = setTimeout(() => {
      onLogout();
      Swal.fire({
        title: "Sesión Expirada",
        text: "Has sido desconectado por inactividad.",
        icon: "warning",
        confirmButtonColor: "#dc3545",
        timer: 5000,
        timerProgressBar: true,
      });
    }, AUTO_LOGOUT_TIME);
  };

  // Escucha eventos de actividad
  window.addEventListener("mousemove", resetTimer);
  window.addEventListener("keydown", resetTimer);
  window.addEventListener("scroll", resetTimer);
  window.addEventListener("click", resetTimer);

  resetTimer(); // Inicia el contador
  return () => {
    clearTimeout(logoutTimer);
    window.removeEventListener("mousemove", resetTimer);
    window.removeEventListener("keydown", resetTimer);
    window.removeEventListener("scroll", resetTimer);
    window.removeEventListener("click", resetTimer);
  };
}, [isLoggedIn, onLogout]);

```

Figura. 51 Cierre automático

#### 4.7.5 Manejo de errores y estados

El componente Login incluye control de estados loading y error, mostrando mensajes al usuario en caso de:

- Credenciales inválidas
- Usuario inactivo
- Problemas de conexión con el servidor

Esto asegura una experiencia de usuario clara y evita intentos fallidos silenciosos.

#### 4.7.6 Integración parcial del Login en App.jsx

En esta etapa, el componente Login se integra al contenedor principal de la aplicación en App.jsx, mostrando la interfaz de inicio de sesión únicamente cuando no existe una sesión activa. La lógica de sesión se mantiene mediante los estados isLoggedIn, usuarioid y loading.

Condicional de renderizado:

- Si **isLoggedIn = true**, se prepara el contenedor principal (main-content) para mostrar posteriormente los componentes internos del sistema, como el Sidebar y las rutas protegidas.

- Si **isLoggedIn = false**, se renderiza la pantalla de Login, incluyendo rutas para recuperación de contraseña (RequestReset) y cambio de contraseña (ResetPassword) (ver Figura. 52).

```

return (
  <div className="app-container">
    {isLoggedIn ? (
      <>
        {sidebarVisible && <Sidebar />}
        <Header onLogout={onLogout} toggleSidebar={toggleSidebar} />
        <Routes>
          <Route path="/" element={<Dashboard />} />
          <Route path="/usuarios" element={<Usuarios />} />
          ...
        </Routes>
      </>
    ) : (
      <Routes>
        <Route path="/login" element={<Login onLogin={onLogin} />} />
        <Route path="/request-reset" element={<RequestReset />} />
        <Route path="/reset-password/:token" element={<ResetPassword />} />
      </Routes>
    )}
)
  
```

Figura. 52 Integración parcial del Login en App.jsx  
(Elaboración propia)

## 4.8 Componente Header

El componente Header constituye la barra superior del sistema. Permite alternar la visibilidad del menú lateral, acceder al perfil del usuario y cerrar sesión. Se desarrolló en React, utilizando useState y useEffect para controlar el estado y la adaptación a distintos tamaños de pantalla.

### 4.8.1 Definición e importaciones

El código inicia con las importaciones necesarias y la definición del componente, que recibe tres propiedades principales: onLogout, toggleSidebar y onChangePassword, como se visualiza en la Figura. 53.

```

import React, { useState, useEffect } from "react";
import "./Header.css";

const Header = ({ onLogout, toggleSidebar, onChangePassword }) => {
  
```

Figura. 53 Definición del componente Header  
(Elaboración propia)

Dentro del componente se declaran los estados y el efecto que ajusta el diseño cuando el ancho de la ventana cambia, garantizando un comportamiento responsivo.

#### 4.8.2 Control de visualización responsiva

Como se muestra en la Figura. 54 se declaran los estados y un efecto que ajusta el diseño cuando el ancho de la ventana cambia, garantizando un comportamiento responsivo.

```
const [isMobile, setIsMobile] = useState(window.innerWidth < 768);
const [showMenu, setShowMenu] = useState(false);

useEffect(() => {
  const handleResize = () => setIsMobile(window.innerWidth < 768);
  window.addEventListener("resize", handleResize);
  return () => window.removeEventListener("resize", handleResize);
}, []);
```

Figura. 54 Control de visualización responsiva  
(Elaboración propia)

#### 4.8.3 Estructura del JSX

Define la estructura visual del encabezado: un botón para alternar el Sidebar y un menú desplegable con las opciones Cambiar contraseña y Cerrar sesión (ver Figura. 55).

```
<header className="app-header">
  <div className="header-left">
    /* Este es tu botón original, mantenemos la funcionalidad */
    {isMobile && (
      <button className="main-action-btn" onClick={toggleSidebar}>
        <i className="fa-solid fa-house"></i>
      </button>
    )}
  </div>

  <div className="header-right">
    <div className="user-menu-container">
      <button
        className="user-menu-btn"
        onClick={() => setShowMenu(!showMenu)}
      >
        <i className="fa fa-user"></i>
      </button>

      {showMenu && (
        <div className="user-menu-dropdown">
          <button className="dropdown-item" onClick={onChangePassword}>
            <i className="fa fa-key" /> Cambiar contraseña
          </button>
          <button className="dropdown-item" onClick={onLogout}>
            <i className="fa fa-sign-out-alt" /> Cerrar sesión
          </button>
        </div>
      )}
    </div>
  </div>
</header>
```

Figura. 55 Estructura principal del Header y menú de usuario  
(Elaboración propia)

## 4.9 Componente Sidebar

El componente Sidebar representa el menú lateral de navegación del sistema. Permite acceder a las secciones principales, submenús y notificaciones. Está desarrollado en React, utilizando useState, useEffect y useContext para gestionar estados, adaptabilidad y notificaciones.

### 4.9.1 Definición e importaciones

Se importan librerías de React, rutas, estilos y contexto de notificaciones, y se define el componente principal como se ve en la Figura. 56.

```
import React, { useState, useEffect } from "react";
import { NavLink, useNavigate } from "react-router-dom";
import logo from "../assets/logo.jpg";
import "./Sidebar.css";
import { useContext } from "react";
import { NotificationContext } from "./NotificationContext";
```

```
const Sidebar = () => {
```

Figura. 56 Definición del componente Sidebar  
(Elaboración propia)

### 4.9.2 Estados y control de responsividad

Como se visualiza en la Figura. 57 se definen estados para abrir/cerrar el menú, submenús y detectar dispositivos móviles. Se usa useEffect para ajustar la visualización según el tamaño de pantalla.

```
const Sidebar = () => {
  const [isOpen, setIsOpen] = useState(false);
  const [isMobile, setIsMobile] = useState(window.innerWidth < 768);
  const [showSubmenu, setShowSubmenu] = useState(false);
  const navigate = useNavigate();
  const [showMantenimientos_submenu, setShowMantenimientos_submenu] = useState(false);
  const [showHistorial_submenu, setShowHistorial_submenu] = useState(false);
  const [showChofer_submenu, setShowChofer_submenu] = useState(false);
  const { pendientes } = useContext(NotificationContext);
  const [clicked, setClicked] = useState(false); // para controlar clic en submenu
```

```
useEffect(() => {
  const handleResize = () => setIsMobile(window.innerWidth < 768);
  window.addEventListener("resize", handleResize);
  return () => window.removeEventListener("resize", handleResize);
}, []);
```

Figura. 57 Estados y control de responsividad  
(Elaboración propia)

#### 4.9.3 Estructura del JSX

El **Sidebar** contiene:

- Logo y botones de apertura/cierre en móvil.
- Menú principal con rutas y submenús.
- Indicadores de notificaciones.

Mostrado en la Figura. 58.

```
<div className={ sidebar ${isMobile ? (isOpen ? "open" : "") : ""} }>
  <div className="sidebar-header">
    <img src={logo} alt="Logo" className="logo" />
    {isMobile && <button className="close-btn" onClick={toggleSidebar}></button>}
  </div>

  <nav className="sidebar-menu">
    <NavLink to="/admin/solicitudes" className={({ isActive }) => (isActive ? "active" :
      <i className="fa fa-line-chart"></i> Solicitudes
      (pendientes > 0 && <span className="notification-badge">{pendientes}</span>)
    </NavLink>

    /* Ejemplo de submenu */
    <div className="menu-item">
      <NavLink to="/mantenimientos" onClick={() => setShowMantenimientosSubmenu(!showMantenimientos)}>
        <i className="fa fa-cogs"></i> Mantenimientos
      </NavLink>
      {showMantenimientosSubmenu && (
        <div className="submenu">
          <NavLink to="/mantenimientos_menores">Mantenimientos Menores</NavLink>
          <NavLink to="/mantenimientos_mayores">Mantenimientos Mayores</NavLink>
        </div>
      )}
    </div>
  </nav>
</div>
```

Figura. 58 Estructura del Sidebar con rutas y submenús  
(Elaboración propia)

#### 4.10 Integración del Header y Sidebar en App.jsx

El contenedor principal de la aplicación (App.jsx) integra los componentes Header y Sidebar, junto con las rutas protegidas y la gestión de sesión del usuario. Esto permite una navegación centralizada y control de accesos según el rol del usuario.

##### 4.10.1 Importaciones y estados principales

Se importan React, rutas, componentes y estados de sesión. Se definen estados para controlar la visibilidad del Sidebar y la sesión activa (ver Figura. 59).

```

import React, { useState } from "react";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Sidebar from "./components/Sidebar";
import Header from "./components/Header";
import Login from "./components/Login";

function App() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [showSidebar, setShowSidebar] = useState(true);

  const toggleSidebar = () => setShowSidebar(!showSidebar);
  const handleLogout = () => setIsLoggedIn(false);
  const handleLogin = () => setIsLoggedIn(true);
}

```

**Figura. 59 Estados principales en App.jsx**  
(Elaboración propia)

#### 4.10.2 Renderizado condicional según sesión

Como se ve en Figura. 60 se renderiza Login si la sesión no está activa y, en caso contrario, se muestra el contenedor principal con Header, Sidebar y rutas protegidas.

```

return (
  <Router>
    {!isLoggedIn ? (
      <Login setLoggedIn={setLoggedIn} />
      eos\recidencias\frontend\src • Contains emphasized items
    ) : (
      <div className="app-container">
        {showSidebar && <Sidebar />}
        <div className="main-content">
          <Header toggleSidebar={toggleSidebar} onLogout={handleLogout} />
          <Routes>
            <Route path="/admin/solicitudes" element={<AdminSolicitudes />} />
            <Route path="/chofer/solicitudes" element={<ChoferSolicitudes />} />
          </Routes>
        </div>
      </div>
    )}
  </Router>
)

```

**Figura. 60 Integración condicional de Header y Sidebar**  
(Elaboración propia)

#### 4.11 Gestión de unidades (Unidades.jsx)

El componente Unidades.jsx permite listar, agregar, actualizar y eliminar unidades de transporte con sus respectivos datos y placas. Incluye paginación, modal de detalles y formularios adaptativos para pantallas grandes y pequeñas.

#### 4.11.1 Importaciones y estados principales

Se importan React, SweetAlert2, el Modal personalizado, estilos y la configuración de la API. Se definen estados para:

- Listado de unidades (unidades)
- Control de carga y errores (loading, error)
- Modal de edición o detalles (modalData, showModal)
- Unidad a editar (unidadToEdit)
- Unidad a agregar (nuevaUnidad)
- Paginación (currentPage, itemsPerPage)

```
const [unidades, setUnidades] = useState([]);
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);
const [modalData, setModalData] = useState(null);
const [showModal, setShowModal] = useState(false);
const [unidadToEdit, setUnidadToEdit] = useState(null);
const [nuevaUnidad, setNuevaUnidad] = useState({...});
const [currentPage, setCurrentPage] = useState(1);
const [itemsPerPage, setItemsPerPage] = useState(5);
const itemsPerPageOptions = [5, 10, 20];
```

Figura. 61 Estados principales para control de unidades  
(Elaboración propia)

#### 4.11.2 Obtención y paginación de unidades

Al montar el componente, se realiza una petición a la API mediante fetch dentro de useEffect, obteniendo los datos desde el backend. Posteriormente, se calcula la paginación para dividir los registros en páginas, mejorando el rendimiento y la visualización mostrada en la Figura. 62.

```
useEffect(() => {
  fetch(`${BASE_URL}/api/unidades`)
    .then(res => res.json())
    .then(data => setUnidades(data))
    .catch(err => setError(err.message))
    .finally(() => setLoading(false));
}, []);
any
const indexOfLast = currentPage * itemsPerPage;
const indexOfFirst = indexOfLast - itemsPerPage;
const currentUnidades = unidades.slice(indexOfFirst, indexOfLast);
const totalPages = Math.ceil(unidades.length / itemsPerPage);
```

Figura. 62 Obtención y paginación de unidades  
(Elaboración propia)

## 4.12 Módulo de garantías

El módulo Garantías se encarga de administrar las pólizas asociadas a cada unidad vehicular. Permite registrar, consultar, actualizar y eliminar garantías. Además, realiza validaciones sobre la vigencia de la póliza y muestra alertas cuando una garantía está próxima a vencer o puede renovarse.

### 4.12.1 Requisitos y restricciones

El sistema permite adjuntar archivos en formato PDF que representan la póliza de garantía correspondiente. Estos archivos se almacenan en la carpeta uploads/garantias, mientras que las pólizas vencidas se mueven automáticamente a uploads/historial\_garantias. Para garantizar la integridad de los datos, solo se puede registrar o renovar una póliza cuando la garantía anterior haya vencido o se encuentre dentro del mes previo a su fecha de expiración. Además, el sistema valida que el identificador de la unidad exista, que los campos numéricos sean válidos y que los archivos adjuntos no sobrepasen el tamaño permitido.

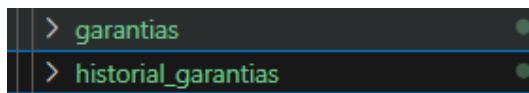


Figura. 63 Carpetas de guardado de documentos  
(Elaboración propia)

### 4.12.2 Definición e importaciones Frontend

Se importan librerías esenciales para la renderización del módulo, manejo de API, estilos y componentes de alerta (ver Figura. 58).

```
import React, { useState, useEffect } from "react";
import Swal from "sweetalert2";
import "./Garantias.css";
import { API_URL } from "../config";
```

Figura. 64 Definición e importaciones del componente Garantías  
(Elaboración propia)

#### 4.12.2.1 Estados y efectos principales

En la Figura. 65 se muestra el componente define los estados que permiten almacenar el listado de garantías, la carga de datos, el control de modales y la unidad seleccionada para edición o renovación.

useEffect ejecuta la consulta inicial al montar el componente.

```

const [garantias, setGarantias] = useState([]);
const [loading, setLoading] = useState(true);
const [unidadSeleccionada, setUnidadSeleccionada] = useState(null);

useEffect(() => {
| obtenerGarantias();
}, []);

const obtenerGarantias = async () => {
| const res = await fetch(`${API_URL}/api/garantias`);
| const data = await res.json();
| setGarantias(data);
| setLoading(false);
};

```

**Figura. 65 Estados principales y carga inicial  
(Elaboración propia)**

#### 4.12.2.2 Verificación y renovación desde la interfaz

El sistema valida si una garantía puede renovarse consultando la api. Al cumplirse la condición, se muestra un mensaje de advertencia o confirmación con sweetalert2 (ver figura. 66).

```

const verificarGarantia = async (idUnidad) => {
  const res = await fetch(`${API_URL}/api/garantias/verificar/${idUnidad}`);
  const data = await res.json();

  if (data.puede_renovar) {
    Swal.fire({
      title: "Renovar garantía",
      text: "La garantía está por vencer, ¿deseas renovarla?",
      icon: "warning",
      showCancelButton: true,
    }).then((result) => {
      if (result.isConfirmed) {
        renovarGarantia(idUnidad);
      }
    });
  } else {
    Swal.fire("Garantía vigente", "La garantía aún está activa.", "info");
  }
};

```

**Figura. 66 Verificación y renovación de garantía  
(Elaboración propia)**

#### 4.12.2.3 Función de renovación y actualización del listado

Cuando se confirma la renovación, se ejecuta una petición PUT al backend, actualizando la vigencia y registrando el evento en el historial, Figura. 67.

```

const renovarGarantia = async (idUnidad) => {
  const res = await fetch(`${API_URL}/api/garantias/renovar/${idUnidad}`, {
    method: "PUT",
  });
  const data = await res.json();
  Swal.fire("Éxito", data.msg, "success");
  obtenerGarantias();
};

```

**Figura. 67 Función de renovación de garantía  
(Elaboración propia)**

#### 4.12.2.4 Estructura del JSX

El renderizado muestra una tabla con todas las garantías registradas, indicando la unidad, fecha de inicio, vigencia y acciones disponibles.

Las filas cambian de color según la proximidad de vencimiento para facilitar su visualización como se muestra en la Figura. 68.

```
return (
  <div className="contenedor-garantias">
    <h2>Gestión de Garantías</h2>
    {loading ? (
      <p>Cargando...</p>
    ) : (
      <table className="tabla-garantias">
        <thead>
          <tr>
            <th>Unidad</th>
            <th>Fecha Inicio</th>
            <th>Vigencia</th>
            <th>Acciones</th>
          </tr>
        </thead>
        <tbody>
          {garantias.map((g) => (
            <tr key={g.id}>
              <td>{g.unidad}</td>
              <td>{g.fecha_inicio}</td>
              <td>{g.vigencia}</td>
              <td>
                <button
                  className="btn-verificar"
                  onClick={() => verificarGarantia(g.id_unidad)}
                >
                  Verificar
                </button>
              </td>
            </tr>
          )));
        </tbody>
      </table>
    );
  </div>
);
```

Figura. 68 Estructura general del JSX  
(Elaboración propia)

#### 4.12.2.5 Estilos y adaptabilidad

El archivo `Garantias.css` define un diseño responsive para pantallas de escritorio y móviles, ajustando el ancho de tabla, fuentes y botones.

El color de fondo de cada fila cambia según la proximidad de expiración, utilizando clases condicionales definidas en el JSX.

### 4.12.3 Backend del Módulo de Garantías

El backend del módulo de garantías está implementado en Flask y utiliza SQLAlchemy como ORM para la persistencia de datos.

Su objetivo es administrar los procesos de registro, consulta, actualización, renovación y eliminación de garantías asociadas a unidades de transporte.

El sistema garantiza la integridad mediante validaciones de vigencia, almacenamiento de versiones anteriores en un historial y control de respuestas JSON para su consumo por el frontend.

#### 4.12.3.1 Modelo de datos

Como se muestra en Figura. 69 el modelo Garantias define la estructura principal de la tabla de garantías, mientras que el modelo HistorialGarantias conserva registros previos cuando se realiza una renovación.

Ambos modelos están relacionados mediante el identificador de la garantía.

```
class Garantias(db.Model):
    __tablename__ = "Garantias"
    id_garantia = db.Column(db.Integer, primary_key=True, autoincrement=True)
    id_unidad = db.Column(db.Integer, db.ForeignKey("Unidades.id_unidad", onupdate="CASCADE", ondelete="RESTRICT"), nullable=False)
    aseguradora = db.Column(db.String(255))
    tipo_garantia = db.Column(db.String(50))
    no_poliza = db.Column(db.String(50), unique=True)
    url_poliza = db.Column(db.String(500))
    suma_asegurancia = db.Column(db.Numeric(15,2))
    inicio_vigencia = db.Column(db.Date)
    vigencia = db.Column(db.Date)
    prima = db.Column(db.Numeric(15,2))

    unidad = db.relationship("Unidades", backref="garantias")

class HistorialGarantias(db.Model):
    __tablename__ = "HistorialGarantias"
    id_garantia = db.Column(db.Integer, primary_key=True, autoincrement=True)
    id_unidad = db.Column(db.Integer)
    fecha_renovacion = db.Column(db.DateTime, default=datetime.now)
    aseguradora = db.Column(db.String(255))
    tipo_garantia = db.Column(db.String(100))
    no_poliza = db.Column(db.String(50))
    url_poliza = db.Column(db.String(500))
    suma_asegurancia = db.Column(db.Numeric(15,2))
    inicio_vigencia = db.Column(db.Date)
    vigencia = db.Column(db.Date)
    prima = db.Column(db.Numeric(15,2))
    usuario = db.Column(db.String(50))
```

Figura. 69 Modelo de datos de garantías e historial  
(Elaboración propia)

#### 4.12.3.2 Endpoint principal de gestión

El endpoint /api/garantias admite las operaciones CRUD básicas. La ruta se implementa con los métodos GET, POST, PUT y DELETE, retornando datos en formato JSON (ver Figura. 70).

```
@app.route('/api/garantias', methods=['POST'])
> def crear_garantia(): ...

@app.route('/api/garantias/<int:id_garantia>', methods=['PUT'])
> def actualizar_garantia(id_garantia): ...

@app.route('/garantias/verificar/<int:id_unidad>', methods=['GET'])
> def verificar_garantia(id_unidad): ...

@app.route('/api/garantias/<int:id_garantia>', methods=['DELETE'])
> def eliminar_garantia(id_garantia): ...
```

Figura. 70 Endpoint principal de garantías  
(Elaboración propia)

#### 4.12.3.3 Verificación de vigencia

La verificación determina si una garantía puede renovarse, considerando un margen de un mes antes o después del vencimiento, mostrada en la Figura. 71.

Si cumple la condición, el sistema notifica al frontend que la renovación está permitida.

```

@app.route('/garantias/verificar<int:id_unidad>', methods=['GET'])
def verificar_garantia(id_unidad):
    garantia = Garantias.query.filter_by(id_unidad=id_unidad).first()
    hoy = date.today()
    puede_renovar = False
    datos = None

    if garantia:
        vigencia = garantia.vigencia
        if vigencia:
            # Renovación permitida 1 mes antes o después del vencimiento
            fecha_límite = vigencia - timedelta(days=30)
            if hoy >= fecha_límite:
                puede_renovar = True
    datos = {
        "aseguradora": garantia.aseguradora,
        "tipo_garantia": garantia.tipo_garantia,
        "no_poliza": garantia.no_poliza,
        "suma_asegurada": garantia.suma_asegurada,
        "inicio_vigencia": garantia.inicio_vigencia.isoformat() if garantia.inicio_vigencia else None,
        "vigencia": garantia.vigencia.isoformat() if garantia.vigencia else None,
        "prima": garantia.prima
    }

```

**Figura. 71 Verificación de vigencia de garantía  
(Elaboración propia)**

#### 4.12.3.4 Proceso de renovación

Cuando una garantía está próxima a vencer y se solicita su renovación, el sistema conserva la trazabilidad mediante el guardado de la versión anterior y la actualización de los nuevos datos.

Primero se almacena el registro previo en la tabla HistorialGarantias, asegurando un seguimiento de cambios, y después se actualizan las fechas de inicio y vigencia de la garantía activa (ver Figura. 72).

```

# Caso: garantía existente y permite renovación
if garantia and puede_renovar:
    url_historial = mover_a_historial(garantia.url_poliza)

    historial = HistorialGarantias(
        id_garantia=garantia.id_garantia,
        id_unidad=garantia.id_unidad,
        fecha_cambio=datetime.now(),
        aseguradora=garantia.aseguradora,
        tipo_garantia=garantia.tipo_garantia,
        no_poliza=garantia.no_poliza,
        url_poliza=url_historial,
        suma_asegurada=garantia.suma_asegurada,
        inicio_vigencia=garantia.inicio_vigencia,
        vigencia=garantia.vigencia,
        prima=garantia.prima,
        usuario=data.get('usuario', 'sistema')
    )
    db.session.add(historial)

    # Actualizar con nuevos datos
    garantia.aseguradora = data.get('aseguradora')
    garantia.tipo_garantia = data.get('tipo_garantia')
    garantia.no_poliza = data.get('no_poliza')
    garantia.url_poliza = ruta_pdf
    garantia.suma_asegurada = data.get('suma_asegurada')
    garantia.inicio_vigencia = data.get('inicio_vigencia')
    garantia.vigencia = data.get('vigencia')
    garantia.prima = data.get('prima')
    print("Garantía actualizada y enviada a histórico (vencida o pre-renovación)")

```

**Figura. 72 Proceso de renovación de garantía y registro histórico  
(Elaboración propia)**

### 4.13 Módulo de verificaciones vehiculares

El módulo de Verificaciones Vehiculares permite registrar y controlar los períodos de verificación de cada unidad, considerando el engomado y holograma correspondiente. Su

objetivo es automatizar el seguimiento y emitir alertas sobre verificaciones próximas o vencidas. Incluye interacción entre frontend y backend para el manejo de archivos PDF y cálculo de fechas.

#### 4.13.1 Frontend: estructura y funciones principales

En la figura Figura. 73 el componente Verificaciones.jsx está desarrollado en React. Se importan useState, useEffect, SweetAlert2, ModalFile, Bootstrap y la configuración de la API.

Se define la estructura base del componente y los estados iniciales para el listado de verificaciones, el control de formularios y los mensajes de validación.

```
import React, { useEffect, useState } from "react";
import Swal from "sweetalert2";
import "./Unidades.css";
import { API_URL } from "../config";
import ModalFile from "../components/ModalFile"; // importa tu modal
```

Figura. 73 Definición e importaciones del componente Verificaciones.jsx  
(Elaboración propia)

##### 4.13.1.1 Obtención y visualización de verificaciones

Mediante useEffect, el componente realiza la consulta a la API para obtener los registros de verificaciones y mostrarlos en una tabla dinámica. Se integran filtros de búsqueda y estado (“EN TIEMPO”, “PENDIENTE”, “ATRASADA”).

La información se actualiza automáticamente tras cada registro o modificación (ver Figura. 74).

```
const res = await fetch(`${API_URL}/api/verificaciones`);
if (!res.ok) throw new Error(`HTTP ${res.status}`);
const data = await res.json();

const dataWithEstado = data.map((v) => {
  let estado = "PENDIENTE";
  if (v.proxima_verificacion) {
    const fechaProx = new Date(v.proxima_verificacion);
    const hoy = new Date();
    estado = hoy > fechaProx ? "ATRASADA" : "EN TIEMPO";
  }
})
```

Figura. 74 Consulta y renderizado de verificaciones en la interfaz  
(Elaboración propia)

##### 4.13.1.2 Registro y validación de nuevas verificaciones

El formulario de registro valida la existencia de la unidad, el engomado, y los períodos previos. Se utiliza FormData para el envío al backend, incluyendo el archivo PDF del

comprobante.

La función calcularFechaPorEngomado() determina automáticamente la fecha límite según el color del engomado y el semestre.

```
const formData = new FormData();
formData.append("id_unidad", idUnidad);
formData.append("periodo_${periodoSeleccionado}", fechaSugerida);
formData.append("periodo_${periodoSeleccionado}_real", periodoReal);
if (holograma) formData.append("holograma", holograma);
if (folio) formData.append("folio_verificacion", folio);
if (engomado) formData.append("engomado", engomado);
formData.append("archivo", archivo);
```

Figura. 75 Validación y registro de verificaciones en el formulario  
(Elaboración propia)

#### 4.13.1.3 Gestión de archivos y reseteo del formulario

Tras un registro exitoso, el sistema limpia los campos mediante resetForm() y muestra notificaciones con SweetAlert2. El comprobante PDF puede visualizarse directamente desde el ModalFile sin salir del componente principal, como se ve en la Figura. 77.

```
const handleIdBlur = () => checkUnidadLocal();
const handleFileChange = (e) => setArchivo(e.target.files[0] ?? null);

const resetForm = () => {
  setIdUnidad("");
  setPeriodoSeleccionado("1");
  setPeriodoReal("");
  setFechaSugerida("");
  setHolograma("");
  setFolio("");
  setEngomado("");
  setPlaca("");
  setArchivo(null);
  setUnidadExiste(false);
  setVerificacionExistente(null);
  setFormDisabled(false);
  setUsarAñoAnterior(false);
  setAñoSeleccionado(añoActual);
};
```

Figura. 76 Reseteo de formulario y gestión de comprobantes PDF  
(Elaboración propia)

#### 4.13.2 Backend del módulo de verificaciones

El backend del módulo de Verificaciones Vehiculares gestiona el ciclo completo del proceso: registro, cálculo, consulta, actualización y almacenamiento histórico. Se emplean modelos SQLAlchemy para manejar las entidades Unidades, Placas, VerificaciónVehicular y HistorialVerificaciónVehicular, junto con funciones auxiliares para el cálculo del engomado y fechas de verificación.

#### 4.13.2.1 Gestión de registros y comprobantes

El endpoint principal /api/verificaciones con método POST permite registrar o actualizar la verificación de una unidad.

Valida existencia, tipo de archivo PDF, calcula automáticamente las fechas de vigencia según holograma, y almacena físicamente los documentos en uploads/verificaciones/. Cuando ya existe una verificación activa, esta se transfiere al historial antes de registrar la nueva, preservando trazabilidad, como se muestra en la Figura. 77.

```
app.route('/api/verificaciones', methods=['POST'])
def crear_verificacion():
    UPLOAD_FOLDER = 'uploads/verificaciones'
    HISTORIAL_FOLDER = 'uploads/historial_verificaciones'

    ruta_upload_abs = os.path.join(current_app.root_path, UPLOAD_FOLDER)
    ruta_historial_abs = os.path.join(current_app.root_path, HISTORIAL_FOLDER)
    os.makedirs(ruta_upload_abs, exist_ok=True)
    os.makedirs(ruta_historial_abs, exist_ok=True)

    data = request.form
    archivo = request.files.get('archivo')

    print("---- DATA recibida ----")
    print(data)
    print("Archivo recibido:", archivo.filename if archivo else "No hay archivo")

    if not archivo or archivo.filename == '':
        return jsonify({"error": "Debe subir un archivo PDF"}), 400
    if not allowed_file(archivo.filename):
        return jsonify({"error": "Solo se permiten archivos PDF"}), 400

    unidad = Unidades.query.filter_by(id_unidad=data['id_unidad']).first()
    if not unidad:
        return jsonify({"error": "La unidad indicada no existe"}), 400
    print("Unidad encontrada:", unidad)

    placa_str = unidad.placa if unidad.placa else ""
    engomado = calcular_color_por_placa(placa_str)
```

Figura. 77 Endpoint para obtención de unidad y cálculo automático de engomado  
(Elaboración propia)

#### 4.13.2.2 Cálculo automático de fechas

La función calcular\_siguiente\_verificacion() determina la fecha de próxima verificación según el holograma y el color de engomado:

- Holograma “00”: cada 2 años.
- Holograma “0”: cada 6 meses.
- Hologramas normales: según calendario oficial de engomados.

Este proceso garantiza que las próximas verificaciones se mantengan sincronizadas con la normativa.

```

#calculo de verificaciones

def calcular_siguiente_verificacion(fecha_real, holograma, engomado):
    if not fecha_real:
        return None

    # Holograma "00" → 2 años después
    if holograma == "00":
        return fecha_real.replace(year=fecha_real.year + 2)

    # Holograma "0" → 6 meses después
    if holograma == "0":
        mes_siguiente = fecha_real.month + 6
        año = fecha_real.year
        if mes_siguiente > 12:
            mes_siguiente -= 12
            año += 1
        ultimo_dia = calendar.monthrange(año, mes_siguiente)[1]
        return fecha_real.replace(year=año, month=mes_siguiente, day=ultimo_dia)

```

Figura. 78 Calculo de la siguiente verificación  
(Elaboración propia)

#### 4.13.2.3 Historial y reseteo automatizado

El sistema incorpora una función periódica mover\_y\_resetear\_verificaciones\_vencidas(), que identifica registros caducados, los mueve a la tabla de historial y limpia el registro activo.

Esta tarea se ejecuta mediante el programador que se muestra en la Figura. 79.

```

def mover_y_resetear_verificaciones_vencidas():
    hoy = date.today()
    limite_reset = hoy + timedelta(days=60)

    verificaciones = VerificacionVehicular.query.all()

    for v in verificaciones:
        fecha_siguiente = calcular_siguiente_verificacion(
            v.ultima_verificacion, v.holograma, v.engomado
        )

        # Solo guarda en historial si está dentro del rango, pero no limpia
        if fecha_siguiente and fecha_siguiente <= limite_reset:
            historial = HistorialVerificacionVehicular(
                id_verificacion=v.id_verificacion,
                fecha_cambio=datetime.now(),
                ultima_verificacion=v.ultima_verificacion,
                periodo_1=v.periodo_1,
                periodo_1_real=v.periodo_1_real,
                url_verificacion_1=v.url_verificacion_1,
                periodo_2=v.periodo_2,
                periodo_2_real=v.periodo_2_real,
                url_verificacion_2=v.url_verificacion_2,
                holograma=v.holograma,
                folio_verificacion=v.folio_verificacion,
                engomado=v.engomado,
                usuario="sistemaAutomatico"
            )
            db.session.add(historial)

```

Figura. 79 Historial y reseteo automatizado  
(Elaboracion propia)

#### 4.13.2.4 Consulta de datos

Los endpoints /api/verificaciones y /api/verificacion-placa/<string:placa> permiten consultar el listado general o una verificación individual.

Cada resultado incluye unidad, holograma, fechas vigentes, próxima verificación y enlace al comprobante PDF (ver Figura. 80).

```

@app.route('/api/verificacion-placa/<string:placa>', methods=['GET'])
def obtener_verificacion_placa(placa):
    placa_obj = Placas.query.filter_by(placa=placa).first()
    if not placa_obj:
        return jsonify({"error": "Placa no encontrada"}), 404

    unidad = placa_obj.unidad
    verificacion = VerificacionVehicular.query.filter_by(id_unidad=unidad.id_unidad).first()
    if not verificacion:
        return jsonify({"error": "No hay verificaciones registradas"}), 404

    # Determinar la fecha base para cálculo de próxima
    fecha_base = verificacion.periodo_2_real or verificacion.periodo_1_real

    proxima = None
    if fecha_base:
        proxima = calcular_siguiente_verificacion(fecha_base, verificacion.holograma, verificacion.engomado)

    return jsonify({
        "unidad": f'{unidad.marca} ({unidad.vehiculo})',
        "placa": placa_obj.placa,
        "holograma": verificacion.holograma,
        "engomado": verificacion.engomado,
        "vigiante": verificacion.periodo_1_real,
        "proxima": proxima,
        "anterior": verificacion.periodo_1_real,
        "url_verificacion_1": verificacion.url_verificacion_1,
        "url_verificacion_2": verificacion.url_verificacion_2
    })

```

**Figura. 80 Consulta de datos de la verificación.**  
(Elaboración propia)

## 4.14 Módulo de solicitudes del chofer

El módulo de Solicitudes del Chofer permite que los conductores registren fallas mecánicas de los vehículos asignados y posteriormente documenten la reparación una vez aprobada por el administrador. Este componente gestiona el flujo completo de solicitud y reparación desde la interfaz del chofer.

### 4.14.1 Definición e importaciones (Frontend)

El componente principal SolicitudFallaPaso1y2 se define en React. Importa librerías esenciales (useState, useEffect), SweetAlert2 para las alertas, los estilos SolicitudForm.css y la constante API\_URL para las peticiones al backend, como se visualiza en la Figura. 81.

```

import React, { useState, useEffect } from "react";
import Swal from "sweetalert2";
import "./SolicitudForm.css";
import { API_URL } from "../config";

export default function SolicitudFallaPaso1y2() {
    const [unidades, setUnidades] = useState([]);
    const [piezas, setPiezas] = useState([]);
    const [marcas, setMarcas] = useState([]);
    const [lugares, setLugares] = useState([]);
    const [solicitud, setSolicitud] = useState(null);
    const [loading, setLoading] = useState(true);
}

```

**Figura. 81 Definición del componente e importaciones**  
(Elaboración propia)

#### 4.14.2 Carga inicial de datos

En el montaje del componente (useEffect), se consultan los catálogos del backend: unidades, piezas, marcas y lugares.

El sistema identifica el rol del usuario (chofer o administrador) para cargar la unidad correspondiente y verificar si existen solicitudes activas (ver Figura. 82).

```
useEffect(() => {
  const idUsuario = localStorage.getItem("id_usuario");
  const rol = localStorage.getItem("rol");

  if (rol === "chofer") {
    fetch(`${API_URL}/unidades/chofer/${idUsuario}`)
      .then(res => res.json())
      .then(data => setUnidades(data));
  } else {
    fetch(`${API_URL}/unidades`)
      .then(res => res.json())
      .then(data => setUnidades(data));
  }

  Promise.all([
    fetch(`${API_URL}/piezas`).then(r => r.json()),
    fetch(`${API_URL}/marcas`).then(r => r.json()),
    fetch(`${API_URL}/lugares`).then(r => r.json())
  ]).then(([p, m, l]) => {
    setPiezas(p);
    setMarcas(m);
    setLugares(l);
  });

  cargarSolicitudes();
}, [l]);
```

Figura. 82 Carga inicial de catálogos y validación de solicitudes  
(Elaboración propia)

#### 4.14.3 Envío de solicitudes de reparación

En la Figura. 83 el evento handleSubmitSolicitud envía la información del formulario al backend mediante una petición POST a /solicitudes.

Antes de enviar, valida campos obligatorios y muestra confirmaciones con SweetAlert2.

En caso de éxito, se limpia el formulario y se bloquea el registro hasta recibir respuesta administrativa.

```
const handleSubmitSolicitud = async (e) => {
  e.preventDefault();
  if (!nuevaSolicitud.unidad || !nuevaSolicitud.descripcion) {
    return Swal.fire("Campos incompletos", "Verifica los datos.", "warning");
  }

  const res = await fetch(`${API_URL}/solicitudes`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(nuevaSolicitud),
  });

  const data = await res.json();
  if (data.status === "ok") {
    Swal.fire("Enviada", "La solicitud fue registrada.", "success");
    setNuevaSolicitud({});
    cargarSolicitudes();
  } else {
    Swal.fire("Error", data.message, "error");
  }
};
```

Figura. 83 Envío de nueva solicitud de reparación  
(Elaboración propia)

#### 4.14.4 Registro de fallas tras aprobación

Cuando una solicitud es aprobada, el chofer puede registrar la falla y cargar el comprobante PDF del servicio.

El formulario se maneja mediante FormData para soportar archivos, incluyendo los campos de proveedor, tipo de pago, costo y observaciones (ver Figura. 84).

```
const handleSubmitFalla = async (e) => {
  e.preventDefault();
  const formData = new FormData();
  formData.append("id_solicitud", solicitud.id);
  formData.append("proveedor", falla.proveedor);
  formData.append("tipo_pago", falla.tipo_pago);
  formData.append("costo", falla.costo);
  formData.append("observaciones", falla.observaciones);
  if (archivoPDF) formData.append("pdf", archivoPDF);

  const res = await fetch(`${API_URL}/fallas`, { method: "POST", body: formData });
  const data = await res.json();

  if (data.status === "ok") {
    Swal.fire("Registrado", "Falla guardada correctamente.", "success");
    setSolicitud(null);
  } else {
    Swal.fire("Error", data.message, "error");
  }
};
```

Figura. 84 Envío del formulario de falla mecánica  
(Elaboración propia)

#### 4.14.5 Renderizado condicional y estructura visual

En la Figura. 81 se muestra el componente que renderiza dinámicamente uno de los dos formularios:

- Si no hay solicitudes activas → muestra **formulario de registro**.
- Si hay solicitud aprobada → muestra **formulario de falla**.

```
return (
  <div className="solicitud-container">
    {!solicitud ? (
      <form onSubmit={handleSubmitSolicitud}>
        <h3>Nueva Solicitud de Reparación</h3>
        <select name="unidad" onChange={handleChange}>
          <option>Selecione unidad</option>
          {unidades.map(u => <option key={u.id}>{u.nombre}</option>)}
        </select>
        <textarea name="descripcion" placeholder="Describa la falla"></textarea>
        <button type="submit">Enviar Solicitud</button>
      </form>
    ) : (
      <form onSubmit={handleSubmitFalla}>
        <h3>Registrar Falla Mecánica</h3>
        <input name="proveedor" placeholder="Proveedor" />
        <input name="costo" placeholder="Costo" />
        <input type="file" onChange={handleArchivo} />
        <button type="submit">Guardar</button>
      </form>
    )}
  </div>
);
```

Figura. 85 Estructura principal del JSX  
(Elaboración propia)

#### 4.14.6 Creación de solicitudes de reparación (Backend)

Permite a los choferes registrar solicitudes de falla mecánica, enviando datos como unidad, pieza, marca, tipo de servicio y descripción.

Los registros se almacenan en la tabla SolicitudFalla con estado inicial “pendiente”, Figura. 86.

```
@app.route('/solicitudes', methods=['POST'])
def crear_solicitud():
    data = request.json
    nueva = SolicitudFalla(
        id_unidad = data['id_unidad'],
        id_pieza = data['id_pieza'],
        id_marca = data.get('id_marca'),
        tipo_servicio = data['tipo_servicio'],
        descripcion = data.get('descripcion', ''),
        id_chofer = data['id_chofer']
    )
    db.session.add(nueva)
    db.session.commit()
    return jsonify({"msg": "Solicitud creada", "id_solicitud": nueva.id_solicitud}), 201
```

Figura. 86 Ruta para crear una nueva solicitud de falla  
(Elaboración propia)

## 4.15 Módulo de Solicitudes del Administrador (Frontend)

Este módulo gestiona las solicitudes enviadas por los choferes, permitiendo aprobar, rechazar o devolver observaciones.

Está diseñado para garantizar control sobre el flujo de mantenimiento y seguimiento.

### 4.15.1 Carga y renderizado de solicitudes

El componente ListaSolicitudes.jsx obtiene los registros desde el backend (/solicitudes) y muestra su estado actual. Se usa paginación y filtros para una navegación rápida (ver Figura. 87).

```
useEffect(() => {
  fetch(`${API_URL}/solicitudes`)
    .then(res => res.json())
    .then(data => setSolicitudes(data));
}, [ ]);

const handleAprobar = async (id) => {
  const res = await fetch(`${API_URL}/solicitudes/aprobar/${id}`, { method: "PUT" });
  const data = await res.json();
  if (data.status === "ok") {
    Swal.fire("Aprobada", "La solicitud fue aprobada.", "success");
    cargarSolicitudes();
  }
};
```

Figura. 87 Listado de solicitudes con opciones de gestión  
(Elaboración propia)

### 4.15.2 Obtención de solicitudes por chofer (Backend)

Esta ruta obtiene todas las solicitudes asociadas a un chofer específico, mostrado en la Figura. 88.

Además, valida si cada solicitud tiene una falla mecánica registrada, para marcarla como completada o pendiente.

```
@app.route('/solicitudes/chofer/<int:id_chofer>', methods=['GET'])
def solicitudes_chofer(id_chofer):
    solicitudes = SolicitudFalla.query.filter_by(id_chofer=id_chofer).all()
    resultado = []
    for s in solicitudes:
        falla_existente = FallaMecanica.query.filter_by(
            id_unidad=s.id_unidad,
            id_pieza=s.id_pieza,
            tipo_servicio=s.tipo_servicio
        ).first()
        resultado.append({
            "id_solicitud": s.id_solicitud,
            "unidad": s.id_unidad,
            "pieza": s.id_pieza,
            "marca": s.id_marca,
            "tipo_servicio": s.tipo_servicio,
            "descripcion": s.descripcion,
            "estado": s.estado,
            "completada": True if falla_existente else False
        })
    return jsonify(resultado)
```

Figura. 88 Ruta para obtener las solicitudes por chofer  
(Elaboración propia)

#### 4.15.3 Listado general de solicitudes (Administrador)

El administrador puede visualizar todas las solicitudes del sistema, ordenadas de más recientes a más antiguas, junto con su estado y chofer asignado.

Esto facilita la priorización y seguimiento de casos (ver Figura. 89).

```
@app.route('/solicitudes', methods=['GET'])
def listar_todas_solicitudes():
    solicitudes = SolicitudFalla.query.order_by(SolicitudFalla.fecha_solicitud.desc()).all()
    resultado = []
    for s in solicitudes:
        resultado.append({
            "id_solicitud": s.id_solicitud,
            "unidad": s.id_unidad,
            "pieza": s.id_pieza,
            "marca": s.id_marca,
            "tipo_servicio": s.tipo_servicio,
            "descripcion": s.descripcion,
            "estado": s.estado,
            "id_chofer": s.id_chofer,
            "fecha_solicitud": s.fecha_solicitud.isoformat()
        })
    return jsonify(resultado)
```

Figura. 89 Ruta para listar todas las solicitudes  
(Elaboración propia)

#### 4.15.4 Aprobación o rechazo de solicitudes

Esta ruta cambia el estado de una solicitud a “aprobada” o “rechazada”, según la decisión del administrador.

La acción se refleja de inmediato en la base de datos y habilita (en caso de aprobación) la creación de una falla mecánica asociada mostrado en la Figura. 90.

```

@app.route('/solicitudes/<int:id_solicitud>/aprobar', methods=['POST'])
def aprobar_solicitud(id_solicitud):
    data = request.json
    solicitud = SolicitudFalla.query.get_or_404(id_solicitud)
    solicitud.estado = 'aprobada' if data.get('aprobar') else 'rechazada'
    db.session.commit()
    return jsonify({'msg': f"Solicitud {'aprobada' if solicitud.estado=='aprobada' else 'rechazada'}"})

```

**Figura. 90 Ruta para aprobar o rechazar solicitudes  
(Elaboración propia)**

#### 4.15.5 Registro de falla mecánica

Cuando la solicitud ha sido aprobada, se puede registrar la falla mecánica correspondiente. Esta ruta recibe los datos del formulario, así como el comprobante PDF del mantenimiento, validando la extensión del archivo y guardándolo en la carpeta designada (ver Figura. 91).

```

@app.route('/fallas', methods=['POST'])
def crear_falla():
    data = request.form
    archivo = request.files.get('comprobante') # nombre del archivo en FormData

    id_solicitud = data.get('id_solicitud')
    if not id_solicitud:
        return jsonify({'error': 'falta id_solicitud'}), 400

    solicitud = SolicitudFalla.query.get_or_404(id_solicitud)
    if solicitud.estado != 'aprobada':
        return jsonify({'error': 'La solicitud no ha sido aprobada'}), 400

    url_comprobante = None
    if archivo:
        if archivo.filename.lower().endswith('.pdf'):
            filename = f'falla_{solicitud.id_solicitud}.pdf'
            carpeta = os.path.join(app.root_path, 'uploads', 'fallasmecanicas')
            os.makedirs(carpeta, exist_ok=True)
            filepath = os.path.join(carpeta, filename)
            archivo.save(filepath)
            url_comprobante = f'uploads/fallasmecanicas/{filename}'
        else:
            return jsonify({'error': 'Solo se permiten archivos PDF'}), 400

```

**Figura. 91 Ruta para registrar una falla mecánica  
(Elaboración propia)**

#### 4.16 Módulo de Placas y Reemplazamientos

El módulo Placas.jsx permite la gestión completa de las placas vehiculares asociadas a las unidades del sistema.

Incluye las operaciones de verificación, registro, edición, eliminación y visualización de documentos PDF de placas.

Su diseño está orientado a prevenir duplicidades, validar vigencias activas y controlar los reemplazos de placas con base en su fecha de caducidad.

##### 4.16.1 Definición e importaciones (Frontend)

En este apartado se importan las dependencias necesarias, incluyendo React, SweetAlert2 para notificaciones, los componentes de modales reutilizables, los estilos CSS y la constante de conexión API\_URL.

Además, se define el componente funcional principal Placas, Figura. 92.

```

import { useEffect, useState } from "react";
import Swal from "sweetalert2";
import Modal from "../components/Modal";
import ModalFile from "../components/ModalFile";
import { API_URL } from "../config";
import "./Unidades.css";
import "./Placas.css";

export default function Placas() {
  const [placas, setPlacas] = useState([]);
  const [unidades, setUnidades] = useState([]);
  const [edit, setEdit] = useState(null);
  const [form, setForm] = useState({});
  const [unidadValida, setUnidadValida] = useState(false);
  const [fileModalUrl, setFileModalUrl] = useState(null);
  const [page, setPage] = useState(1);
  const [perPage] = useState(10);
  const [total, setTotal] = useState(0);
}

```

Figura. 92 Definición del componente y estados principales.  
(Elaboración propia)

#### 4.16.2 Obtención de datos y carga inicial

En la figura Figura. 93 se implementan dos funciones asíncronas para cargar los datos desde el backend:

- fetchPlacas() obtiene las placas paginadas.
- fetchUnidades() carga las unidades disponibles para asignación.

Ambas funciones se ejecutan dentro de useEffect al montar el componente o al cambiar de página.

```

useEffect(() => {
  fetchPlacas();
  fetchUnidades();
}, [page]);

const fetchPlacas = async () => {
  try {
    const res = await fetch(`${API_URL}/placas?page=${page}&per_page=${perPage}`);
    const data = await res.json();
    setPlacas(data.placas || []);
    setTotal(data.total || 0);
  } catch (err) {
    Swal.fire("Error", "No se pudieron cargar las placas", "error");
  }
};

```

Figura. 93 Carga inicial de placas y unidades.  
(Elaboración propia)

#### 4.16.3 Verificación de unidad y validación de vigencia

Antes de registrar una nueva placa, se valida que la unidad exista y que no tenga una placa vigente con más de 180 días de duración.

La función verificarUnidad() revisa las fechas de vigencia y muestra mensajes informativos usando SweetAlert2, Figura. 94.

```

const verificarUnidad = async () => {
  if (!form.id_unidad)
    return Swal.fire("Advertencia", "Debes ingresar el ID de la unidad", "warning");

  const unidad = unidades.find(u => u.id_unidad.toString() === form.id_unidad.toString());
  if (!unidad) {
    setUnidadValida(false);
    return Swal.fire("Error", "La unidad no existe", "error");
  }

  try {
    const res = await fetch(`${API_URL}/placas?id_unidad=${form.id_unidad}`);
    const data = await res.json();
    const placasUnidad = data.placas || [];
    if (placasUnidad.length === 0) {
      setUnidadValida(true);
      return Swal.fire(
        "Unidad válida",
        "No existe placa activa, puedes registrar nueva placa",
        "success"
      );
    }
  }
}

```

**Figura. 94 Verificación de unidad y control de vigencia.  
(Elaboración propia)**

#### 4.16.4 Registro, actualización y eliminación de placas

Las operaciones CRUD se gestionan mediante fetch y FormData, permitiendo subir archivos PDF de placas frontal y trasera.

Se validan los campos requeridos y se muestran alertas en cada evento (ver Figura. 95).

```

const handleRegistro = async () => {
  if (!UnidadValida) return Swal.fire("Advertencia", "Verifica primero la unidad", "warning");
  if (!form.placa || !form.fecha_vigencia) return Swal.fire("Advertencia", "Placa y fecha de vigencia son obligatorias", "warning");

  const fd = new FormData();
  Object.keys(form).forEach((k) => {
    if (form[k] !== undefined && form[k] !== null) fd.append(k, form[k]);
  });

  try {
    const res = await fetch(`${API_URL}/placas/registrar`, { method: "POST", body: fd });
    if (!res.ok) {
      const err = await res.json();
      return Swal.fire("Error", err.error || "Error al registrar nueva placa", "error");
    }
  }
}

```

**Figura. 95 Registro de nueva placa y manejo de archivos.  
(Elaboración propia)**

#### 4.16.5 Backend del módulo de Placas

El backend está implementado en Flask, con SQLAlchemy para la gestión de datos y APScheduler para la ejecución programada de tareas.

Las funciones más relevantes son la gestión de rutas, el control automático de vencimientos y la comunicación con el módulo de correo para alertas.

##### 4.16.5.1 Rutas principales

Como se visualiza en la Figura. 96 estas rutas permiten consultar todas las placas registradas y realizar pruebas de envío de alertas.

El endpoint /placas devuelve los registros existentes en formato JSON, y /test-email permite verificar la configuración del correo institucional.

```

@app.route('/placas', methods=['GET'])
def get_placas():
    placas = Placas.query.all()
    resultado = [
        {
            "id_placa": p.id_placa,
            "numero": p.numero,
            "vigencia": p.vigencia.isoformat(),
            "id_unidad": p.id_unidad
        }
        for p in placas
    ]
    return jsonify(resultado)

@app.route('/test-email', methods=['GET'])
def test_email():
    enviar_alerta_vencimiento('test@correo.com', 'ABC-123', '2025-12-01')
    return jsonify({"msg": "Correo de prueba enviado"})

```

**Figura. 96 Rutas principales del módulo de Placas.**  
(Elaboración propia)

#### 4.16.5.2 Registro y renovación de placas

El registro de una nueva placa se realiza mediante el endpoint /placas/nueva, que valida los datos recibidos desde el frontend y almacena la información en la base de datos.

Cuando una placa llega a su fecha límite, el sistema la marca como vencida y permite su renovación con una nueva vigencia, Figura. 97.

```

@app.route("/placas/registrar", methods=["POST"])
def registrar_o_actualizar_placa():
    UPLOAD_DIR = "uploads/placas/"
    HISTORIAL_DIR = "uploads/historial_placas/"
    USUARIO_SISTEMA = "sistema"
    data = request.form
    id_unidad = data.get("id_unidad")
    nueva_placa = data.get("placa")
    fecha_expedicion = data.get("fecha_expedicion")
    fecha_vigencia = data.get("fecha_vigencia")

    if not id_unidad or not nueva_placa or not fecha_vigencia:
        return jsonify({"error": "Unidad, placa y fecha de vigencia son obligatorios"}), 400
    hoy = date.today()
    placa_activa = Placas.query.filter_by(id_unidad=id_unidad).first()

    if placa_activa:
        # Revisar días restantes
        dias_restantes = (placa_activa.fecha_vigencia - hoy).days if placa_activa.fecha_vigencia else 0
        if dias_restantes > 180:
            return jsonify({
                "error": "No se puede registrar nueva placa. Vigencia actual hasta (placa_activa.fecha_vigencia)"
            }), 400
        # Mover archivos al historial
        for field_name in ["unica", "frontal", "reverso", "placa_trasera"]:
            old_path = os.path.join(placa_activa.field_name)
            if old_path and os.path.exists(old_path):
                os.makedirs(HISTORIAL_DIR, exist_ok=True)
                ext = old_path.rsplit('.', 1)[1].lower()
                nuevo_nombre = f"{id_unidad.uuid4()}.{ext}"
                dst = os.path.join(HISTORIAL_DIR, nuevo_nombre)
                shutil.move(old_path, dst)
            # Guardar ruta en historial
            setattr(placa_activa, field_name, dst.replace("\\\\", "/"))

```

**Figura. 97 Registro y renovación de placas.**  
(Elaboración propia)

#### 4.16.5.3 Tareas automáticas y alertas

El sistema utiliza un scheduler que ejecuta periódicamente la función de verificación de vencimientos.

Si una placa está próxima a expirar, se envía un correo de alerta al responsable correspondiente.

Esta tarea se ejecuta cada cierto intervalo definido en tiempo, (ver Figura. 98).

```
# -----
# Scheduler diario
# -----
scheduler = BackgroundScheduler()
scheduler.add_job(lambda: app.app_context().push() or reseteo_y_alertas(), 'interval', days=7)
scheduler.start()
```

Figura. 98 Scheduler para envío automático de alertas.  
(Elaboración propia)

## 4.17 Módulo de Refrendo y Tenencia

El sistema de gestión de pagos de refrendo y tenencia permite registrar, administrar y controlar los pagos de las unidades vehiculares. El backend, desarrollado en **Flask**, se encarga de validar duplicados, mover los registros al historial al cambiar de año, almacenar los archivos PDF de facturas y enviar alertas automáticas por correo. El frontend, implementado en **React**, utiliza *hooks*, manejo de estado, validaciones dinámicas y componentes modales para la visualización y edición de la información.

### 4.17.1 Carga inicial de datos y manejo de estados (Frontend)

El componente carga la lista de unidades y pagos existentes al inicio mediante useEffect, empleando peticiones fetch al backend Flask.

Se gestionan los estados locales con useState, incluyendo formularios, validaciones, tipo de pago y estados de carga.

```
useEffect(() => {
  fetch(`${API_URL}/unidades`)
    .then(res => res.json())
    .then(data => setUnidades(data || []))
    .catch(() => Swal.fire("Error", "No se pudieron cargar las unidades", "error"));

  fetchPagos();
}, [ ]);
```

Figura. 99 Carga inicial de datos y manejo de estados.  
(Elaboración propia)

#### 4.17.1.1 Validación y tipo de pago

El usuario selecciona una unidad y valida si es posible registrar un nuevo pago (ver Figura. 100).

El tipo de pago se determina automáticamente con base en la fecha seleccionada:

- **REFRENDO:** hasta el 31 de marzo.

- **AMBOS (REFRENDO + TENENCIA):** posterior a esa fecha.

```
const handleFechaChange = (e, targetForm = form, setTargetForm = setForm) => {
  const fecha = e.target.value;
  setTargetForm(prev => ({ ...prev, fecha_pago: fecha }));

  const fechaPago = new Date(fecha);
  const limiteRefrendo = new Date(fechaPago.getFullYear(), 2, 31);
  setTipoPago(fechaPago > limiteRefrendo ? "AMBOS" : "REFRENDO");
};
```

Figura. 100 Validación y tipo de pago  
(Elaboración propia)

#### 4.17.1.2 Registro de pagos

Al registrar un pago se empaquetan los datos en un FormData, incluyendo archivos PDF de facturas, y se envían al servidor mediante una solicitud POST. El sistema restablece el formulario y recarga la tabla de registros al finalizar mostrada en la Figura. 101.

```
const handleRegistro = async () => {
  const fd = new FormData();
  fd.append("id_unidad", form.id_unidad);
  fd.append("fecha_pago", form.fecha_pago);
  fd.append("monto_refrendo", form.monto_refrendo || "0");
  fd.append("monto_tenencia", form.monto_tenencia || "0");
  if (form.url_factura_refrendo)
    fd.append("url_factura_refrendo", form.url_factura_refrendo);
  const res = await fetch(`${API_URL}/refrendo_tenencia`, { method: "POST", body: fd });
  const data = await res.json();
  Swal.fire("Éxito", data.message || "Pago registrado", "success");
};
```

Figura. 101 Registro de pagos  
(Elaboración propia)

#### 4.17.1.3 Visualización y edición

La tabla muestra la información consolidada de cada unidad y su pago correspondiente, con opciones para visualizar las facturas y editar los datos. El modal de edición permite actualizar montos, fechas y archivos PDF asociados (Figura. 102).

```
{showModal && editForm && (
  <Modal onClose={() => setShowModal(false)}>
    <h2>Editar Pago - Unidad {editForm.id_unidad}</h2>
    <input type="date" name="fecha_pago" value={editForm.fecha_pago}
      onChange={(e) => handleFechaChange(e, editForm, setEditForm)} />
    <button className="update-btn" onClick={handleSaveEdit}>Guardar Cambios</button>
  </Modal>
)}
```

Figura. 102 Visualización y edición  
(Elaboración propia)

#### 4.17.2 Verificación de disponibilidad de unidad (Backend)

Antes de registrar un nuevo pago, el sistema valida si la unidad ya cuenta con un registro en el año actual.

Si existe, el endpoint responde con un mensaje de advertencia; de lo contrario, autoriza el registro, visualizado en la Figura. 103.

```
@app.route("/refrendo_tenencia/check/<int:id_unidad>", methods=["GET"])
def check_unidad(id_unidad):
    hoy = date.today()
    year = hoy.year

    # Buscar si ya hay registro de este año
    pago = Refrendo_Tenencia.query.filter(
        Refrendo_Tenencia.id_unidad == id_unidad,
        db.extract("year", Refrendo_Tenencia.fecha_pago) == year
    ).first()

    if pago:
        return jsonify({
            "ok": False,
            "mensaje": "Ya existe un pago registrado para este año",
            "refrendo": pago.tipo_pago in ["REFRENDO", "AMBOS"],
            "tenencia": pago.tipo_pago in ["TENENCIA", "AMBOS"]
        })

    # Si no existe, se puede registrar
    return jsonify({
        "ok": True,
        "mensaje": "Unidad disponible para registrar pago",
        "refrendo": False,
        "tenencia": False
    })
```

Figura. 103 Verificación de disponibilidad de unidad  
(Elaboración propia)

#### 4.17.2.1 Funciones auxiliares de archivos

Se incluyen funciones para validar extensiones permitidas (pdf), guardar archivos en carpetas específicas según su tipo, y moverlos al historial cuando se renueva un pago (ver Figura. 104).

```
ALLOWED_EXTENSIONS = {"pdf"}

def allowed_file(filename):
    return "." in filename and filename.rsplit(".", 1)[1].lower() in ALLOWED_EXTENSIONS

def save_uploaded_file(file_obj, tipo, id_unidad, fecha_pago):
    if not file_obj or file_obj.filename == "":
        return None
    if not allowed_file(file_obj.filename):
        return None

    subfolder = "refrendo" if tipo == "REFRENDO" else "tenencia"
    folder = os.path.join(app.root_path, "uploads", subfolder)
    os.makedirs(folder, exist_ok=True)

    ext = os.path.splitext(file_obj.filename)[1].lower()
    filename = f'{(tipo.lower())}_{(id_unidad)}_{(fecha_pago.isoformat())}_{(uuid.uuid4().hex)}{ext}'
    safe = secure_filename(filename)
    filepath = os.path.join(folder, safe)
    file_obj.save(filepath)

    return os.path.join("uploads", subfolder, safe).replace("\\\\", "/")
```

Figura. 104 Funciones auxiliares de archivos de refrendo y tenencia  
(Elaboración propia)

#### 4.17.2.2 Registro y reseteo de pagos

El endpoint principal crea o actualiza los registros de refrendo y tenencia. Incluye tres procesos críticos:

1. Crear un nuevo registro si no existe.
2. Mover registros antiguos al historial si el año cambió.
3. Guardar los nuevos pagos y archivos PDF asociados.

```
@app.route('/refrendo_tenencia', methods=['POST'])
def registrar_pago():
    data = request.form
    try:
        id_unidad = int(data.get('id_unidad'))
        fecha_pago = datetime.strptime(data.get('fecha_pago'), '%Y-%m-%d').date()
        monto_general = float(data.get('monto_general')) or 0
        monto_refrendo = float(data.get('monto_refrendo')) or 0
        monto_tenencia = float(data.get('monto_tenencia')) or 0
        tipo_pago_input = data.get('tipo_pago', 'REFRENDO')
        usuario = data.get('usuario', 'Sistema')
    except Exception:
        return jsonify({"error": "Datos inválidos"}), 400

    unidad = Unidades.query.get(id_unidad)
    if not unidad:
        return jsonify({"error": "Unidad no encontrada"}), 404

    limite_pago = date(fecha_pago.year, 3, 31)
    registro = Refrendo_Tenencia.query.filter_by(id_unidad=id_unidad).first()
    reset_realizado = False

    # -----#
    # Si no existe registro, crearlo desde cero
    # -----#
    if not registro:
        registro = Refrendo_Tenencia(
            id_unidad=id_unidad,
            fecha_pago=fecha_pago,
            limite_pago=limite_pago,
            tipo_pago='REFRENDO' if fecha_pago <= limite_pago else 'AMBOS',
            monto_refrendo=monto_refrendo or monto_general,
            monto_tenencia=monto_tenencia or monto_general,
            monto=(monto_refrendo or monto_general) + (monto_tenencia or monto_general),
            observaciones=data.get('observaciones'),
            url_factura_refrendo=None,
            url_factura_tenencia=None
        )
        db.session.add(registro)
        db.session.commit()
        reset_realizado = True
    else:
        if fecha_pago >= limite_pago:
            registro.fecha_pago = fecha_pago
            registro.monto_refrendo = monto_refrendo
            registro.monto_tenencia = monto_tenencia
            registro.monto = monto
            registro.observaciones = data.get('observaciones')
            db.session.commit()

    return jsonify({"success": "Pago registrado con éxito"}), 201
```

Figura. 105 Registro y reseteo de pagos de tenencia y refrendo  
(Elaboración propia)

#### 4.17.2.3 Actualización de pagos existentes

El endpoint PUT permite modificar montos, observaciones y archivos PDF, registrando al usuario que realiza el cambio (ver Figura. 106).

```
@app.route('/refrendo_tenencia/<int:id_pago>', methods=['PUT'])
def actualizar_pago(id_pago):
    data = request.form

    # Ahora obtenemos el usuario desde FormData
    user_id = data.get('usuario')
    if not user_id:
        return jsonify({"error": "Usuario no proporcionado"}), 401

    registro = Refrendo_Tenencia.query.get(id_pago)
    if not registro:
        return jsonify({"error": "Registro no encontrado"}), 404

    try:
        if 'fecha_pago' in data:
            registro.fecha_pago = datetime.strptime(data['fecha_pago'], '%Y-%m-%d').date()
            registro.limite_pago = date(registro.fecha_pago.year, 3, 31)

        if 'monto_refrendo' in data:
            registro.monto_refrendo = float(data['monto_refrendo'])
        if 'monto_tenencia' in data:
            registro.monto_tenencia = float(data['monto_tenencia'])
        if 'observaciones' in data:
            registro.observaciones = data['observaciones']

        # Archivos
        file_refrendo = request.files.get('url_factura_refrendo')
        file_tenencia = request.files.get('url_factura_tenencia')
        if file_refrendo:
            registro.url_factura_refrendo = save_uploaded_file(file_refrendo, "REFRENDO", regis
```

Figura. 106 Actualización de pagos existentes  
(Elaboración propia)

#### 4.17.2.4 Envío automático de alertas

Cada semana se ejecuta una tarea automática con BackgroundScheduler, que envía correos a los usuarios con pagos pendientes según la fecha límite. También se incluye un endpoint manual para pruebas, Figura. 107.

```
# -----
# Programador semanal automático
# -----
scheduler = BackgroundScheduler()
scheduler.add_job(lambda: app.app_context().push() or enviar_alertas_refrendo_tenencia(), 'interval', days=7)
scheduler.start()
```

Figura. 107 Envió de alertas automáticas de refrendo y tenencia  
(Elaboración propia)

## **CAPÍTULO V: PRUEBAS Y RESULTADOS DEL SISTEMA**

Finalmente, se presentan las pruebas realizadas para validar el correcto funcionamiento del sistema. Esto incluye pruebas funcionales para comprobar que cada módulo responde según lo esperado, pruebas de usabilidad para garantizar que el sistema sea amigable y fácil de usar.

## 5.1 Preparación del entorno virtual en Proxmox

Para el despliegue del sistema se utiliza una máquina virtual creada en el hipervisor Proxmox. Se selecciona **Ubuntu Server 22.04 LTS** debido a su estabilidad y soporte a largo plazo, lo cual resulta adecuado para entornos productivos y académicos primero actualizamos el sistema operativo como se muestra en la figura 108.

```
imanol@fibratec:~$ sudo apt update && sudo apt upgrade -y
[sudo] password for imanol:
Hit:1 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Reading package lists... Done
```

Figura. 108 Actualización del sistema operativo.  
(Elaboración propia)

Se recomienda asignar como mínimo **cuatro núcleos de CPU y 8 GB de memoria RAM**, garantizando un desempeño adecuado tanto para el servidor web como para el backend. La interfaz de red debe configurarse en modo **Bridge**, permitiendo que la máquina virtual obtenga conectividad directa a la red local. Asimismo, es importante contar con una **dirección IP fija o reservada**, lo cual facilita el acceso constante al servicio.

## 5.2 Instalación de Python y configuración del entorno virtual

El backend del sistema está desarrollado en Python, por lo que es necesario instalar **Python 3**, el módulo de entornos virtuales y el gestor de paquetes pip. El uso de entornos virtuales permite aislar las dependencias del proyecto, evitando conflictos con otros servicios del sistema (ver figura 109).

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
imanol@fibratec:~$ sudo apt install -y python3 python3-venv python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.12.3-0ubuntu2.1).
python3 set to manually installed.
The following additional packages will be installed:
```

Figura. 109 Configuración del entorno virtual.  
(Elaboración propia)

Posteriormente, se crea un directorio estándar para alojar el proyecto dentro de /var/www, siguiendo convenciones de administración en servidores Linux, como se muestra en la figura 110:

```
No VM guests are running outdated hypervisor (QEMU) binaries on this host.  
imanol@fibratec:~$ sudo mkdir -p /var/www/control_vehicular  
imanol@fibratec:~$ ls  
imanol@fibratec:~$ sudo chown -R $imanol:$imanol /var/www/control_vehicular  
imanol@fibratec:~$ cd /var/www/control_vehicular
```

Figura. 110 Alojamiento del proyecto en la raíz  
(Elaboración propia)

### 5.3 Transferencia del proyecto al servidor

El código fuente del sistema puede transferirse al servidor mediante dos métodos. El método recomendado es el uso de un repositorio Git, ya que facilita el control de versiones y futuras actualizaciones del proyecto como se visualiza en la figura 111. Alternativamente, puede emplearse la transferencia directa de archivos desde un equipo local mediante SCP.

```
imanol@fibratec:/var/www/control_vehicular$ sudo git clone https://github.co  
m/imacr/fibratec.git  
[sudo] password for imanol:  
Cloning into 'fibratec'...  
remote: Enumerating objects: 25110, done.  
remote: Counting objects: 100% (25110/25110), done.  
remote: Compressing objects: 100% (17027/17027), done.  
remote: Total 25110 (delta 7286), reused 25110 (delta 7286), pack-reused 0 (from 0)  
Receiving objects: 100% (25110/25110), 64.22 MiB | 16.63 MiB/s, done.  
Resolving deltas: 100% (7286/7286), done.  
Updating files: 100% (33510/33510) done
```

Figura. 111 Clonamiento del proyecto a través de GitHub  
(Elaboración propia)

### 5.4 Configuración del backend con Flask y Gunicorn

Dentro del directorio del backend se crea un entorno virtual específico para el proyecto. Posteriormente como se muestra en la figura 112, se instalan las dependencias definidas en el archivo requirements.txt y el servidor de aplicaciones **Gunicorn**, el cual es recomendado para entornos productivos.

```
imanol@fibratec:/var/www/control_vehicular$ cd fibratec/
imanol@fibratec:/var/www/control_vehicular/fibratec$ cd backend/
imanol@fibratec:/var/www/control_vehicular/fibratec/backend$ python3 -m venv
venv
Error: [Errno 13] Permission denied: '/var/www/control_vehicular/fibratec/ba
ckend/venv/include'
imanol@fibratec:/var/www/control_vehicular/fibratec/backend$ sudo python3 -m
venv venv
imanol@fibratec:/var/www/control_vehicular/fibratec/backend$ source venv/bin
/activate
(venv) imanol@fibratec:/var/www/control_vehicular/fibratec/backend$ pip install gun
icorn
Collecting gunicorn
  Downloading gunicorn-23.0.0-py3-none-any.whl.metadata (4.4 kB)
Collecting packaging (from gunicorn)
  Downloading packaging-25.0-py3-none-any.whl.metadata (3.3 kB)
Downloading gunicorn-23.0.0-py3-none-any.whl (85 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━ 85.0/85.0 kB 1.0 MB/s eta 0:00:00
Downloading packaging-25.0-py3-none-any.whl (66 kB)
   ━━━━━━━━━━━━━━━━━━━━ 66.5/66.5 kB 4.0 MB/s eta 0:00:00
Installing collected packages: packaging, gunicorn
Successfully installed gunicorn-23.0.0 packaging-25.0
(venv) imanol@fibratec:/var/www/control_vehicular/fibratec/backend$ |
```

Figura. 112 se crea un entorno virtual específico para el proyecto  
(Elaboración propia)

El backend Flask se ejecuta mediante el servidor de aplicaciones Gunicorn, el cual se configura para trabajar con cuatro procesos de trabajo (workers) y enlazarse a la dirección local 127.0.0.1 a través del puerto 8000. Esta configuración permite atender múltiples solicitudes de forma concurrente y asegura que el servicio permanezca accesible únicamente desde el servidor, siendo consumido a través del proxy inverso configurado en Nginx (ver figura 113).

```
(venv) imanol@fibratec:/var/www/control_vehicular/fibratec/backend$ gunicorn
-w 4 -b 127.0.0.1:8000 main:app
[2025-12-15 20:42:03 +0000] [12232] [INFO] Starting gunicorn 23.0.0
[2025-12-15 20:42:03 +0000] [12232] [INFO] Listening at: http://127.0.0.1:8000 (12232)
[2025-12-15 20:42:03 +0000] [12232] [INFO] Using worker: sync
[2025-12-15 20:42:03 +0000] [12233] [INFO] Booting worker with pid: 12233
[2025-12-15 20:42:03 +0000] [12234] [INFO] Booting worker with pid: 12234
[2025-12-15 20:42:03 +0000] [12235] [INFO] Booting worker with pid: 12235
[2025-12-15 20:42:03 +0000] [12236] [INFO] Booting worker with pid: 12236
^C[2025-12-15 20:44:32 +0000] [12232] [INFO] Handling signal: int
[2025-12-15 20:44:32 +0000] [12233] [INFO] Worker exiting (pid: 12233)
[2025-12-15 20:44:32 +0000] [12235] [INFO] Worker exiting (pid: 12235)
[2025-12-15 20:44:32 +0000] [12234] [INFO] Worker exiting (pid: 12234)
[2025-12-15 20:44:32 +0000] [12236] [INFO] Worker exiting (pid: 12236)
[2025-12-15 20:44:32 +0000] [12232] [INFO] Shutting down: Master
```

Figura. 113 Ejecución mediante el servidor de aplicaciones Gunicorn  
(Elaboración propia)

## 5.5 Automatización del backend con systemd

Para garantizar que la aplicación Flask se ejecute de manera automática al iniciar el sistema, se configura un servicio **systemd**. Esto permite una administración adecuada del backend, así como el monitoreo de su estado.

```
r/fibratec/backend$ sudo nano /etc/systemd/system/flask_app.service
```

El servicio define el usuario de ejecución, el directorio de trabajo y el comando de inicio utilizando Gunicorn. Una vez creado, se habilita y se inicia el servicio, figura 114.

```
(venv) imanol@fibratec:/var/www/control_vehicular/fibratec/backend$ sudo sys
temctl daemon-reload
(venv) imanol@fibratec:/var/www/control_vehicular/fibratec/backend$ sudo sys
temctl enable flask_app
(venv) imanol@fibratec:/var/www/control_vehicular/fibratec/backend$ sudo sys
temctl start flask_app
(venv) imanol@fibratec:/var/www/control_vehicular/fibratec/backend$ sudo sys
temctl status flask_app
● flask_app.service - Flask App
   Loaded: loaded (/etc/systemd/system/flask_app.service; enabled; preset>
             Active: active (running) since Mon 2025-12-15 20:55:52 UTC; 2s ago
     Main PID: 12599 (gunicorn)
       Tasks: 17 (limit: 9431)
      Memory: 179.4M (peak: 180.0M)
        CPU: 833ms
      CGroup: /system.slice/flask_app.service
              └─12599 /var/www/control_vehicular/fibratec/backend/venv/bin/p>
                ├─12600 /var/www/control_vehicular/fibratec/backend/venv/bin/p>
                  |
```

Figura. 114 ejecución automática del servidor  
(Elaboración propia)

## 5.6 Construcción del frontend con React y Vite

El frontend del sistema se desarrolla con React y Vite, por lo que es necesario instalar **Node.js en su versión LTS**. Posteriormente, se compila el proyecto para generar archivos estáticos optimizados para producción (ver figura 115).

```
root@fibratec:/var/www/control_vehicular/fibratec# sudo apt install -y nodejs
s
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  nodejs
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
```

Figura. 115 Instalación de Node.js en su versión LTS  
(Elaboración propia)

Este proceso genera el directorio dist, el cual será servido directamente por el servidor web.

```
root@fibratec:/var/www/control_vehicular/fibratec/frontend# npm run build
> frontend@0.0.0 build
> vite build

vite v7.1.6 building for production...
✓ 178 modules transformed.

dist/index.html          0.52 kB | gzip:  0.33 kB
dist/assets/logo-D-fS4saK.jpg 39.27 kB
dist/assets/image-DjQF5Hfl.png 79.31 kB
dist/assets/fibra-D8ZIVScg.png 102.86 kB
dist/assets/index-CatBr0_3.css 267.65 kB | gzip: 38.79 kB
dist/assets/index-CrZ41QlS.js 758.35 kB | gzip: 223.90 kB

(!) Some chunks are larger than 500 kB after minification. Consider:
- Using dynamic import() to code-split the application
- Use build.rollupOptions.output.manualChunks to improve chunking: https://rollupjs.org/configuration-options/#output-manualchunks
```

Figura. 116 directorio dist  
(Elaboración propia)

Nginx se utiliza como servidor web y proxy inverso. Su función es servir el frontend estático y redirigir las peticiones al backend Flask mediante la ruta /api.

Se crea un archivo de configuración específico para el proyecto y se habilita dentro de Nginx. Una vez verificada la sintaxis, el servicio se reinicia para aplicar los cambios.

```
root@fibratec:/var/www/control_vehicular/fibratec/frontend# sudo apt install  
-y nginx  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done
```

Figura. 117 Nginx se utiliza como servidor web y proxy inverso  
(Elaboración propia)

## CONCLUSIONES PRELIMINARES

El desarrollo del sistema de control vehicular ha logrado una integración sólida entre sus componentes técnicos y funcionales, avanzando de manera estructurada desde las fases de análisis y diseño hasta la implementación de los módulos operativos principales. Durante las etapas iniciales se definieron los requerimientos, actores y casos de uso, lo que permitió establecer un flujo de trabajo claro y una arquitectura escalable acorde a las necesidades del sistema.

La base de datos en MySQL fue diseñada bajo un enfoque relacional, garantizando la integridad y consistencia de la información mediante el uso de claves foráneas y mecanismos de control como triggers, lo que sustenta adecuadamente los procesos de registro, actualización y seguimiento de unidades, choferes, mantenimientos, pólizas, placas, refrendos, tenencias y garantías.

El backend desarrollado en Flask (Python) se integró correctamente con el frontend implementado en React + Vite a través de servicios REST, permitiendo una comunicación eficiente entre ambas capas y facilitando la visualización y gestión de la información mediante una interfaz moderna, responsive y orientada a la usabilidad.

Entre los módulos desarrollados se incluyen el sistema de autenticación con recuperación de contraseña, la administración de unidades vehiculares, la gestión de choferes y solicitudes, el control de placas con notificaciones automáticas por correo electrónico, así como los apartados correspondientes a pólizas de seguro, refrendos, tenencias, historiales de mantenimiento, verificaciones y garantías, ampliando el alcance operativo y documental del sistema.

A pesar de que la implementación funcional del sistema se encuentra concluida, aún se encuentran pendientes las fases de **despliegue en un entorno de producción** y la **ejecución de pruebas integrales**, las cuales permitirán validar el rendimiento, la seguridad, la estabilidad y la correcta interacción entre los distintos módulos del sistema. Asimismo, estas etapas serán fundamentales para realizar ajustes finales, optimizaciones y la consolidación de la documentación técnica.

En conclusión, el proyecto presenta una estructura estable, modular y adaptable, con una implementación alineada a los objetivos planteados. Una vez completadas las fases de despliegue y pruebas, el sistema estará en condiciones óptimas para su operación en un

entorno real, contribuyendo significativamente a la optimización de la gestión vehicular y a la mejora de la trazabilidad de la información.

## BIBLIOGRAFÍA

### Referencias electrónicas

1. Alicante, V. (2023, November 17). Diagrama Entidad Relación: tipos, características y ventajas. Universidad Europea; Universidad Europea | Universidad presencial (Madrid, Valencia, Alicante, Canarias, Málaga) y Online. <https://universidadeuropea.com/blog/modelo-entidad-relacion/#que-es-mer>
2. Amazon Web Services. (2023). ¿Qué es Python? Recuperado de <https://aws.amazon.com/es/what-is/python/>
3. Arsys. (2024, enero 11). Backend: qué es y por qué tiene tanta importancia en el desarrollo web. <https://www.arsys.es/blog/backend-que-es-y-por-que-tiene-tanta-importancia-en-desarrollo-web>
4. Bootstrap. (2025). Documentación de Bootstrap. Recuperado de <https://getbootstrap.com/docs/5.0/getting-started/introduction/>
5. DigitalOcean. (2020, noviembre 12). Cómo añadir autenticación a su aplicación con Flask-Login. <https://www.digitalocean.com/community/tutorials/how-to-add-authentication-to-your-app-with-flask-login-es>
6. Fielding, R. T. (2000). Estilos arquitectónicos y el diseño de arquitecturas de software basadas en redes (Tesis doctoral). Universidad de California, Irvine. Recuperado de [https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)
7. Flask-es ReadTheDocs. (2009). Manejo de los errores de la aplicación - Flask. <https://flask-es.readthedocs.io/errorhandling/>
8. Font Awesome. (2025). Font Awesome. Recuperado de Documentos de Font Awesome | Documentos de Font Awesome
9. GeeksforGeeks. (2025, marzo 26). Flask Middlewares. <https://www.geeksforgeeks.org/python/flask-middlewares/>
10. Git SCM. (2025). Libro Pro Git. Recuperado de <https://git-scm.com/book/es/v2>
11. Juncotic. (2025, agosto 29). Blueprints en Flask. <https://juncotic.com/blueprints-en-flask/>
12. Marín Monterde, U. (2017). Lenguajes de programación. Universidad Nacional Autónoma de México. <https://repositorio>

[uapa.cuaed.unam.mx/repositorio/moodle/pluginfile.php/2655/mod\\_resource/content/1/UAPA-Lenguajes-Programacion/index.html](http://uapa.cuaed.unam.mx/repositorio/moodle/pluginfile.php/2655/mod_resource/content/1/UAPA-Lenguajes-Programacion/index.html)

13. MDN Web Docs. (2025). API Fetch. Recuperado de [https://developer.mozilla.org/es/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/es/docs/Web/API/Fetch_API)
14. MDN Web Docs. (2025). JavaScript. Recuperado de <https://developer.mozilla.org/es/docs/Web/JavaScript>
15. Microsoft Docs. (2025). Arquitectura cliente-servidor. Recuperado de Estilo de arquitectura de Event-Driven - Azure Architecture Center | Microsoft Learn
16. Microsoft SQL Server. (2025). Diseño de relaciones entre tablas. <https://learn.microsoft.com/es-es/sql/relational-databases/tables/relationships-between-tables?view=sql-server-ver17>
17. Microsoft SQL Server. (2025). Introducción a consultas Transact-SQL. <https://learn.microsoft.com/es-es/sql/t-sql/queries/select-transact-sql?view=sql-server-ver17>
18. Microsoft SQL Server. (2025). Restricciones de clave principal y clave externa. <https://learn.microsoft.com/es-es/sql/relational-databases/tables/primary-and-foreign-key-constraints?view=sql-server-ver17>
19. Node.js Foundation. (2025). Acerca de Node.js. Recuperado de <https://nodejs.org/es/about/>
20. Oracle. (2025). Documentación de MySQL. Recuperado de <https://dev.mysql.com/doc/>
21. Pallets Projects. (2025). Documentación de Flask. Recuperado de <https://flask.palletsprojects.com/en/2.1.x/>
22. Python Software Foundation. (2025). Documentación de Python. Recuperado de <https://wiki.python.org/moin/FrontPage>
23. React Router. (2025). Documentación de React Router. Recuperado de <https://reactrouter.com/start/modes>
24. ReactJS. (2025). Documentación de React. Recuperado de <https://reactjs.org/docs/getting-started.html>
25. SQLAlchemy Documentation. (2024). Tutorial de ORM SQLAlchemy. Recuperado de <https://docs.sqlalchemy.org/en/20/intro.html>
26. Sobre FIBRATEC – FIBRATEC. (2024). Fibratec.mx. <https://fibratec.mx/nosotros/>

27. Team Asana. (2025, February 19). Las 12 metodologías más populares para la gestión de proyectos [2025] Asana. <https://asana.com/es/resources/project-management-methodologies>
28. TutorialesProgramacionYa. (s.f.). Serialización y deserialización en Python (módulo json).  
[https://www.tutorialesprogramacionya.com/pythonya/detalleconcepto.php?punto=1\\_01&codigo=102&inicio=90](https://www.tutorialesprogramacionya.com/pythonya/detalleconcepto.php?punto=1_01&codigo=102&inicio=90)

## Referencias bibliográficas

1. Grinberg, M. (2024). Flask Web Development (2<sup>a</sup> ed.). O'Reilly Media.
2. Shaw, Z. A. (2014). Learn Python the Hard Way (3rd ed.). Addison-Wesley.
3. Elmasri, R., & Navathe, S. (2016). Fundamentos de Bases de Datos (6<sup>a</sup> ed.). Pearson.

## **ANEXOS**

### **Alcance del sistema**

El sistema se centrará exclusivamente en la **gestión de vehículos de la institución**, delimitando claramente lo que sí y lo que no se desarrollará:

- **Funciones incluidas**

- Registro y administración de vehículos.
- Gestión de mantenimientos programados y realizados.
- Control de verificaciones vehiculares.
- Alertas automáticas basadas en vencimientos o condiciones específicas.
- Reportes exportables en formatos comunes (PDF, Excel).
- Envío de notificaciones a responsables.
- 

- **Funciones excluidas**

- No será un sistema empresarial completo.
- No integrará rastreo GPS en tiempo real.
- No se desarrollará una aplicación móvil nativa (solo versión web).
- No se contemplará la gestión de combustible ni la logística de rutas.

### **Usuarios y roles**

Se establecen dos perfiles principales dentro del sistema:

- **Administrador**

- Acceso total a todas las funciones.
- Creación y gestión de usuarios.
- Configuración del sistema.

- Generación y consulta de reportes globales.
- **Operador**
  - Registro y actualización de vehículos.
  - Consulta y carga de mantenimientos.
  - Gestión de verificaciones.
  - Acceso a notificaciones relacionadas con su área de responsabilidad.

## Datos críticos

Los datos que sostendrán el sistema son:

- **Vehículos:** número de placas, marca, modelo, año, pólizas de seguro y vigencias.
- **Mantenimientos:** fechas programadas y realizadas, tipo de servicio (correctivo/preventivo), estatus y técnico responsable.
- **Verificaciones:** fecha de vencimiento, resultado, centro autorizado.
- **Alertas:** responsables designados, criterios de activación (ej. vencimiento próximo), medio de notificación.

**Importante: historial de eventos** para trazabilidad completa de cambios.