

Trabajo fin de grado GESTION DE TALLER REYbOXES



Álvaro Gómez Tejada

1. Introducción y Planteamiento del Proyecto.

1.1. Presentación del proyecto y justificación en el contexto del sector productivo.

1.2. Objetivos generales y específicos del proyecto.

1.3. Recopilación de información sobre el contexto tecnológico y empresarial.

* Clasificación de las empresas del sector según su organización y servicios.
* Análisis de estructuras organizativas y funciones de los departamentos clave.
* Identificación y priorización de necesidades del sector.
* Evaluación de oportunidades de negocio y tendencias tecnológicas.

1.4. Identificación de necesidades y oportunidades de intervención.

* Necesidades más demandadas por las empresas
* Oportunidades de negocio previsibles en el sector
* Análisis de proyectos similares y benchmarking.

1.5. Factores facilitadores y barreras que pueden afectar al desarrollo del proyecto.

* Obligaciones fiscales, laborales y de prevención de riesgos
* Posibles ayudas o subvenciones para la incorporación de nuevas tecnologías.

1.6. Metodología de trabajo y planificación inicial.

2. Estudio de viabilidad y Análisis del Proyecto

2.1. Establecimiento del alcance del sistema.

2.2. Estudio de la situación actual (proyectos similares y diferencias)

2.3. Definición de los requisitos del sistema.

2.4. Estudio de alternativas de solución.

2.5. Valoración de las alternativas

2.6. Selección de la solución.

3. Plan de Intervención

3.1 Determinación del tipo de proyecto requerido para dar respuesta a las necesidades detectadas. (permisos y autorizaciones necesarias)

3.2. Caracterización de las especificaciones técnicas del proyecto.

3.3. Priorización y secuenciación de las fases del proyecto (Ciclo de vida y metodología).

3.4. Planificación detallada de la intervención (Diagrama de Gantt).

3.5. Identificación de los recursos materiales y humanos necesarios.

3.6. Estimación de costes y necesidades de financiación.

3.7. Gestión de riesgos y elaboración del plan de prevención de incidencias

3.8. Diseño de la documentación técnica del proyecto.

3.9. Definición del plan de atención al cliente o usuario final.

4. Diseño del Proyecto.

4.1. Diseño de casos de uso (diagramas de caso de uso).

4.2. Diseño de clases/arquitectura módulos de sistema (diagrama de clases/módulos).

4.3. Diseño físico de datos (Diagramas E/R)

4.4. Definición de interfaces de usuario.

4.5. Plan de pruebas y requisitos de aceptación.

4.6. Migración (en caso de ser necesaria) y requisitos de implantación (que se requiere).

5. Seguimiento, Evaluación y Control de Calidad

5.1. Definición de procedimientos de evaluación del proyecto.

5.2. Establecimiento de indicadores de calidad y métricas de éxito.

5.3. Registro y análisis de incidencias surgidas durante la ejecución.

5.4. Plan de resolución de incidencias y optimización de procesos.

5.5. Gestión de cambios en recursos y tareas según necesidades.

5.6. Evaluación de la satisfacción del usuario final y feedback recibido.

5.7. Aseguramiento del cumplimiento de los requisitos y condiciones del proyecto.

6. Conclusiones, Documentación y Presentación Final

6.1. Evaluación global del proyecto y comparación con los objetivos iniciales.

6.2. Dificultades encontradas y estrategias de resolución.

6.3. Propuestas de mejora y evolución del proyecto.

**7. Manual de Usuario**

1. Introducción y Planteamiento del Proyecto

1.1 Presentación del proyecto y justificación en el contexto del sector productivo

En el contexto actual de constante transformación digital y evolución tecnológica, los sectores tradicionales como el de la mecánica automotriz han comenzado a experimentar una necesaria transición hacia la implementación de soluciones informáticas que permitan una gestión más eficiente, centralizada y automatizada de los procesos internos. Dentro de este escenario, el presente proyecto —titulado Sistema Integral de Gestión para Talleres Mecánicos ReyBoxes— surge como una respuesta innovadora, funcional y adaptativa a las crecientes demandas del sector productivo vinculado a los servicios de reparación, mantenimiento y gestión vehicular. Asimismo, contempla la posibilidad de adaptarse a otras líneas de negocio relacionadas, como la compraventa de vehículos, ofreciendo una solución personalizada y ajustada a las particularidades de cada tipo de taller.

La necesidad de contar con un sistema integral que unifique y digitalice los procedimientos operativos de los talleres mecánicos es una demanda recurrente en el ámbito empresarial. Este tipo de establecimientos, tradicionalmente gestionados mediante métodos manuales o herramientas informáticas rudimentarias (como hojas de cálculo), enfrentan en la actualidad numerosos retos en términos de eficiencia, trazabilidad de intervenciones, gestión de clientes, historial de vehículos, control del personal, facturación y cumplimiento normativo. Es en este contexto donde se justifica el desarrollo del presente sistema, cuyo principal objetivo es facilitar una transformación digital accesible, robusta y escalable para su implementación en entornos reales como el del taller *ReyBoxes*, destinatario principal de esta solución.

A través de la implementación de este sistema, se busca no solo optimizar el rendimiento operativo de los talleres, sino también dotar al personal técnico y administrativo de herramientas modernas que fomenten la profesionalización del servicio, la fidelización de los clientes y la mejora continua de los procesos. Esta iniciativa se enmarca plenamente dentro de la estrategia de modernización de los servicios productivos, promoviendo la competitividad del sector y su adaptación a las nuevas realidades del mercado.

1.2 Objetivos generales y específicos del proyecto

Objetivo general

Desarrollar e implementar un sistema informático integral para la gestión completa del taller mecánico de la empresa cliente, que permita automatizar, controlar y optimizar todos los procesos operativos, administrativos y técnicos. El sistema debe facilitar una gestión eficiente de clientes, vehículos, intervenciones, personal y recursos, así como integrar funcionalidades específicas para la compraventa de vehículos, adaptándose a las particularidades de cada área del negocio.

Objetivos específicos

Diseñar una arquitectura modular que se adapte a diferentes perfiles de usuarios (administrador, mecánico, personal administrativo).

Implementar un sistema de autenticación seguro con control de roles y permisos diferenciados.

Permitir el registro, consulta y modificación de clientes y vehículos, incluyendo funcionalidades avanzadas de búsqueda y autocompletado.

Integrar un sistema de gestión de fichajes para el control horario del personal técnico.

Incorporar un historial completo de intervenciones, con seguimiento de fechas, costos, observaciones y estado de cada reparación o mantenimiento.

Facilitar la exportación de datos e informes en formatos estándar (CSV, PDF), incluyendo reportes visuales personalizados con el logotipo de la empresa.

Garantizar la seguridad y confidencialidad de la información almacenada, utilizando cifrado de contraseñas, validación de entradas y buenas prácticas de desarrollo.

1.3 Recopilación de información sobre el contexto tecnológico y empresarial

Clasificación de las empresas del sector según su organización y servicios

El sector de los talleres mecánicos puede clasificarse, a grandes rasgos, en tres tipos de organizaciones:

Talleres independientes o tradicionales: Generalmente gestionados por uno o pocos técnicos, ofrecen servicios generales de reparación sin una estructura empresarial formalizada. Suelen carecer de herramientas digitales avanzadas y dependen fuertemente del conocimiento técnico individual.

Talleres multimarca o franquiciados: Pertenecen a redes o cadenas más amplias que proporcionan estándares de calidad, imagen corporativa y protocolos de atención. Están en un proceso más avanzado de digitalización, aunque no siempre cuentan con soluciones integradas personalizadas.

Talleres oficiales o de concesionario: Vinculados a marcas específicas, operan con sistemas informáticos propietarios o muy específicos. Cuentan con mayor infraestructura tecnológica, pero a menudo presentan rigidez ante cambios o adaptaciones.

Cada una de estas organizaciones presenta retos y oportunidades distintos en cuanto a digitalización, lo que justifica el desarrollo de una solución adaptable como la que se propone.

Análisis de estructuras organizativas y funciones de los departamentos clave

Un taller moderno suele contar con los siguientes departamentos o roles funcionales:

Recepción y atención al cliente: Encargado de registrar las citas, explicar los servicios y atender las consultas. Es clave para la experiencia del cliente.

Técnico o mecánico: Eje central del taller, realiza diagnósticos, reparaciones, mantenimientos y reporta información técnica.

Gestión administrativa: Maneja presupuestos, facturación, control de pagos, compras de repuestos y seguimiento financiero.

Dirección técnica o gerencia: Toma decisiones estratégicas, coordina equipos, evalúa indicadores de rendimiento y asegura el cumplimiento normativo.

El sistema propuesto debe contemplar las interacciones y flujos de información entre todos estos roles, permitiendo una comunicación fluida y evitando la duplicidad de tareas o errores humanos.

Identificación y priorización de necesidades del sector

A partir de estudios de campo, entrevistas con profesionales del sector y análisis de flujos de trabajo, se han identificado las siguientes necesidades prioritarias:

Reducción del uso de papel y digitalización de formularios y reportes.

Registro automatizado de entradas y salidas de personal.

Consulta rápida del historial de vehículos y clientes.

Generación de presupuestos y facturas con control de descuentos y promociones.

Control de citas y órdenes de trabajo.

Análisis de rentabilidad por tipo de intervención o mecánico.

Almacenamiento seguro y backup periódico de la información.

Interfaz amigable y personalizable según el perfil del usuario.

Evaluación de oportunidades de negocio y tendencias tecnológicas

La transformación digital del sector mecánico representa una oportunidad estratégica poco explotada hasta el momento. La mayoría de los talleres aún no cuentan con sistemas propios integrados, y dependen de software genérico o procesos manuales. Esto abre una ventana de innovación para soluciones a medida como la aquí presentada.

Entre las tendencias tecnológicas más relevantes que sustentan esta propuesta se encuentran:

Digitalización de PYMEs mediante software específico de bajo coste y fácil implementación.

Computación en la nube y bases de datos distribuidas, que permiten la portabilidad y escalabilidad del sistema.

Integración de informes en tiempo real para toma de decisiones basada en datos.

Sistemas de seguridad reforzada, como cifrado de datos y gestión avanzada de permisos.

Automatización de tareas repetitivas, reduciendo errores humanos y tiempos de gestión.

1.4 Identificación de necesidades y oportunidades de intervención

En un entorno cada vez más competitivo y exigente, el sector de los talleres mecánicos se enfrenta a múltiples desafíos que requieren respuestas tecnológicas innovadoras. La identificación de necesidades recurrentes, junto con el análisis de oportunidades emergentes, permite establecer una base sólida para justificar la intervención mediante el desarrollo del presente proyecto.

• Necesidades más demandadas por las empresas

Las empresas del sector han manifestado, de forma reiterada, un conjunto de necesidades que reflejan la carencia de herramientas informáticas específicas que se ajusten a su realidad operativa. Entre las demandas más relevantes se destacan:

La necesidad de un sistema centralizado para gestionar clientes, vehículos, reparaciones, agendas y fichajes del personal.

La reducción del uso del papel, en favor de soluciones digitales que permitan registrar intervenciones, generar informes y consultar historiales de forma inmediata.

La automatización de tareas repetitivas, como la facturación, la generación de presupuestos o el control de stock de piezas.

La posibilidad de realizar consultas avanzadas y personalizadas, tanto por datos técnicos como por información comercial.

La incorporación de un sistema de informes y estadísticas que permita visualizar el rendimiento del negocio, detectar fallos y proponer mejoras continuas.

Estas necesidades, comunes a talleres de distinta envergadura, demuestran la urgencia de una solución tecnológica que pueda adaptarse a distintos escenarios y niveles de complejidad.

• Oportunidades de negocio previsibles en el sector

El sector automotriz, y particularmente el segmento de mantenimiento y reparación, está experimentando un proceso de modernización sin precedentes. Este cambio estructural se traduce en una serie de oportunidades de negocio:

La digitalización de procesos abre el camino a modelos de negocio basados en el Software as a Service (SaaS), con licencias adaptadas a talleres pequeños, medianos o grandes.

El auge de los vehículos eléctricos e híbridos conlleva nuevas necesidades de mantenimiento, documentación y formación, generando nichos de especialización.

La compraventa de vehículos de segunda mano, junto con la gestión del historial mecánico certificado, representa una oportunidad clave de expansión funcional del sistema.

La creciente concienciación sobre la sostenibilidad y trazabilidad impulsa el uso de software para el control responsable de residuos, emisiones, y materiales utilizados.

Estas oportunidades indican que el desarrollo de soluciones como ReyBoxes no solo satisface una demanda actual, sino que se anticipa a futuras exigencias del mercado.

• Análisis de proyectos similares y benchmarking

Se ha llevado a cabo un proceso de benchmarking con distintos productos y servicios informáticos del sector, evaluando tanto herramientas locales como soluciones de alcance internacional. Entre los sistemas analizados se encuentran:

Soluciones genéricas como Microsoft Excel o Access, ampliamente utilizadas pero limitadas en funcionalidades específicas.

Programas de gestión ERP adaptados parcialmente al sector, que presentan interfaces complejas y una curva de aprendizaje elevada.

Plataformas SaaS extranjeras, como Shopmonkey, AutoFluent o GaragePlug, que si bien ofrecen funcionalidades avanzadas, no se ajustan completamente a las normativas locales ni al idioma, y suponen un coste elevado en licencias.

Frente a este panorama, ReyBoxes se posiciona como una alternativa intermedia pero poderosa: una solución de desarrollo nacional, adaptada a las particularidades del sector mecánico español, con interfaces amigables, enfoque modular y funcionalidades diseñadas en diálogo directo con los talleres que integran esta franquicia en expansión.

1.5 Factores facilitadores y barreras que pueden afectar al desarrollo del proyecto

El desarrollo de este sistema se enmarca en una realidad técnica, legal y económica que puede influir positiva o negativamente en su ejecución. Identificar estos factores permite planificar mejor los recursos, reducir riesgos y aprovechar los elementos facilitadores.

• Obligaciones fiscales, laborales y de prevención de riesgos

Los talleres mecánicos están sujetos a una amplia normativa legal que afecta directamente a la gestión de su actividad. Algunos aspectos relevantes incluyen:

Obligaciones fiscales como la correcta emisión de facturas, control del IVA e IRPF, y registros de operaciones.

Normativas laborales, incluyendo el registro obligatorio de jornada laboral, cumplimiento del convenio colectivo, y contratos del personal técnico.

Prevención de riesgos laborales, que exige documentar protocolos de seguridad, mantenimiento de equipos, uso de EPIs, y formación del personal.

El sistema ReyBoxes contempla estos aspectos desde su diseño, facilitando el cumplimiento normativo mediante módulos de control de fichajes, almacenamiento de documentos, alertas de vencimiento y trazabilidad de operaciones.

• Posibles ayudas o subvenciones para la incorporación de nuevas tecnologías

La coyuntura actual, enmarcada en los planes de recuperación económica y modernización de las PYMEs, representa una oportunidad inmejorable para proyectos tecnológicos como este. Existen distintas líneas de ayuda:

Subvenciones europeas vinculadas a los fondos Next Generation EU, con programas específicos para digitalización empresarial (Kit Digital, Programa Acelera Pyme).

Ayudas autonómicas y municipales orientadas a la modernización de sectores tradicionales mediante incorporación de TIC.

Deducciones fiscales por inversión en I+D+i aplicables a proyectos de desarrollo informático innovador.

Aprovechar estas ayudas puede suponer una aceleración en la implementación del sistema, así como una importante reducción de costes para las empresas usuarias.

1.6 Metodología de trabajo y planificación inicial

El desarrollo del proyecto se ha estructurado siguiendo una metodología de trabajo ágil y adaptativa, orientada a la consecución de resultados funcionales desde etapas tempranas. Para ello, se ha optado por un enfoque iterativo e incremental, basado en los principios del modelo Scrum dentro de una filosofía DevOps, que permite mantener la calidad del código, documentar el avance de manera ordenada y validar continuamente con el cliente.

La planificación inicial contempla las siguientes fases:

Análisis de requerimientos y diseño conceptual

Reuniones con el cliente para identificar necesidades, prioridades y características del entorno real. Diseño de la arquitectura lógica del sistema y definición de módulos funcionales.

Desarrollo técnico por módulos

Implementación progresiva de los distintos componentes del sistema (login, gestión de clientes, intervenciones, fichajes, exportación de informes, etc.), con validación continua y pruebas unitarias.

Integración y pruebas completas del sistema

Verificación de la interoperabilidad entre módulos, simulación de casos reales, corrección de errores y mejora del rendimiento general.

Despliegue, documentación y formación

Instalación en el entorno final, generación de manuales técnicos y de usuario, y capacitación del personal para el uso autónomo de la herramienta.

Soporte y evolución futura

Seguimiento posterior a la entrega, resolución de incidencias, y análisis de nuevas funcionalidades según el feedback recibido.

Esta metodología garantiza un producto de calidad, alineado con las expectativas del cliente, y fácilmente adaptable a futuras evoluciones del sector.

2. Estudio de viabilidad y Análisis del Proyecto

2.1 Establecimiento del alcance del sistema

El sistema propuesto tiene como finalidad proporcionar una solución informática integral para la gestión de talleres mecánicos, capaz de adaptarse a distintos perfiles de usuario (administrador, personal técnico y administrativo) y abarcar todas las áreas operativas del negocio: desde la atención al cliente y el control de intervenciones, hasta la gestión de fichajes, la emisión de informes y la organización de los vehículos registrados. Asimismo, se contempla la integración de funcionalidades específicas para la compraventa de vehículos y la generación automatizada de documentación fiscal y operativa.

En su alcance inicial, el sistema debía ser accesible desde cualquier dispositivo, incluyendo ordenadores, tablets y smartphones, por medio de la ejecución en navegadores modernos mediante tecnologías web como React. Sin embargo, debido a restricciones impuestas en el entorno de implementación, se prohibió expresamente el uso de tecnologías web o soluciones basadas en navegadores. Este cambio forzó una reformulación total del planteamiento técnico, reconduciendo el desarrollo hacia una aplicación de escritorio nativa, ejecutable en entornos Windows, desarrollada íntegramente en Python utilizando la biblioteca PySide6, el binding oficial de Qt para Python.

A pesar de este cambio sustancial en la tecnología base, se mantuvo el alcance funcional original del sistema, garantizando que la solución continúe siendo modular, escalable, y visualmente profesional. El objetivo sigue siendo ofrecer una herramienta completa, personalizable, y fácilmente operable por parte de cualquier usuario del taller, sin requerir conocimientos técnicos avanzados.

2.2 Estudio de la situación actual (proyectos similares y diferencias)

El análisis del mercado ha evidenciado la existencia de soluciones informáticas dirigidas al sector de la mecánica automotriz, muchas de ellas con arquitectura web y orientadas a modelos SaaS. Ejemplos de este tipo incluyen plataformas como:

Shopmonkey (EE.UU.): solución basada en la nube, centrada en talleres de reparación, con interfaz moderna pero escasa flexibilidad frente a normativas locales.

GaragePlug (India): software multiplataforma con enfoque en digitalización de talleres, pero con barreras de idioma, localización fiscal y soporte técnico.

Autowork Online (Reino Unido): sistema basado en navegador, con alta dependencia de conexión a internet y modelo de suscripción mensual.

Frente a estas soluciones, el presente proyecto presenta diferencias clave:

El sistema se ejecuta de forma local en cada equipo, manteniendo únicamente la conexión remota con la base de datos, con el objetivo de centralizar la información y permitir el acceso simultáneo desde todos los talleres de la franquicia.

Adaptación total al idioma, cultura organizativa y normativas fiscales y laborales españolas.

Personalización completa, desarrollada en estrecha colaboración con la franquicia ReyBoxes, con posibilidad de ajustar módulos según las necesidades reales del negocio.

Independencia tecnológica, al no depender de plataformas cerradas ni licencias de terceros, lo cual reduce costes y aumenta la sostenibilidad a largo plazo.

Estas diferencias confirman la viabilidad y pertinencia de una solución propia frente a adoptar productos genéricos del mercado.

2.3 Definición de los requisitos del sistema

El sistema debe cumplir una serie de requisitos funcionales y no funcionales para satisfacer las necesidades del cliente:

Requisitos funcionales:

Registro, modificación y consulta de clientes y vehículos.

Gestión de intervenciones (reparaciones, mantenimientos, observaciones, estados, fechas, costes).

Control de fichajes del personal (entrada/salida) con reloj en tiempo real.

Panel de control personalizado según rol de usuario.

Exportación de datos a CSV y generación de informes en PDF.

Recuperación de cuenta con validación por correo y código OTP.

Módulo de usuarios con control de roles (Administrador, Mecánico, Administrativo).

Autocompletado inteligente y validación de datos (DNI, correos, nombres).

Requisitos no funcionales:

Sistema multiplataforma (al menos compatible con distintos PCs Windows).

Interfaz visual moderna y accesible.

Alta responsividad en ventanas y formularios.

Seguridad en el almacenamiento de contraseñas (cifrado).

Base de datos en la nube (Supabase/PostgreSQL) con conexión externa protegida por SSL.

Modularidad en el código fuente y separación estricta de lógica de presentación y control.

2.4 Estudio de alternativas de solución

Durante las primeras fases del proyecto, se contemplaron diversas alternativas tecnológicas:

Aplicación web con React + Node.js (Express)

✔️ Ventajas: ejecución en navegador, responsive, amplia comunidad.

❌ Desventajas: prohibición de uso web, requiere servidor backend y dominio externo.

Aplicación híbrida con Electron

✔️ Ventajas: interfaz web con empaquetado como escritorio.

❌ Desventajas: peso elevado, consumo de recursos, dificultad para cumplir con prohibición explícita de tecnologías web.

Aplicación de escritorio en Java (JavaFX)

✔️ Ventajas: fuerte tipado, multiplataforma.

❌ Desventajas: curva de aprendizaje más elevada, dificultades estéticas, menor fluidez visual.

Aplicación de escritorio en Python con PySide6

✔️ Ventajas: rapidez de desarrollo, estética profesional con Qt, integración sencilla con base de datos externa, compatibilidad con equipos modestos.

❌ Desventajas: requiere instalación de dependencias específicas, distribución requiere empaquetado adicional.

2.5 Valoración de las alternativas

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Alternativa | Adaptabilidad | Desempeño | Cumple con restricciones | Facilidad de desarrollo | Total |
| React + Node.js | Alta | Alta | No | Media | 3/5 |
| Electron | Alta | Media | No | Media | 3/5 |
| JavaFX | Media | Alta | Sí | Baja | 3.5/5 |
| Python + PySide6 | Alta | Alta | Sí | Alta | 5/5 |

La solución basada en PySide6 resultó ser la única que cumplía con todas las condiciones impuestas por el entorno del proyecto: la prohibición de tecnologías web, la necesidad de ejecución local con posibilidad de conectarse a una base de datos compartida entre múltiples instancias del programa, una estética profesional y una capacidad de iteración rápida.

2.6 Selección de la solución

Tras el análisis detallado de alternativas, se decidió optar por la implementación del sistema utilizando Python y PySide6, por tratarse de una tecnología flexible, potente y altamente productiva, que permite desarrollar interfaces gráficas avanzadas respetando la arquitectura MVC. Esta elección no solo garantiza el cumplimiento estricto de las condiciones impuestas, sino que además permite mantener un diseño elegante, moderno y altamente usable por parte del personal de taller.

La elección de PySide6 también asegura que el sistema pueda ser empaquetado fácilmente como ejecutable (.exe) y distribuido sin complicaciones técnicas, favoreciendo su adopción por parte del cliente final sin necesidad de conocimientos técnicos ni configuración adicional. La conexión con la base de datos PostgreSQL alojada en Supabase refuerza la capacidad del sistema para operar con datos centralizados, respaldos remotos y sincronización segura.

3. Plan de Intervención

3.1 Determinación del tipo de proyecto requerido para dar respuesta a las necesidades detectadas

El tipo de intervención planteada se enmarca en el desarrollo de una aplicación informática a medida, orientada a entornos de escritorio y destinada a cubrir las necesidades operativas, administrativas y técnicas de una red de talleres mecánicos en expansión.

Dado que el proyecto se basa en la implementación de software que se ejecuta localmente, no requiere licencias urbanísticas, sanitarias ni ambientales. Sin embargo, se deben tener en cuenta ciertas autorizaciones legales y normativas de cumplimiento:

Cumplimiento del Reglamento General de Protección de Datos (RGPD), en lo relativo al almacenamiento y tratamiento de datos personales de clientes, empleados y proveedores.

Incorporación de mecanismos de seguridad informática, incluyendo cifrado de contraseñas y validación de accesos.

Utilización de software con licencias compatibles, incluyendo PySide6 (licencia LGPL) y módulos auxiliares, lo que permite un uso legal y redistribución del sistema.

3.2 Caracterización de las especificaciones técnicas del proyecto

El sistema se desarrollará con las siguientes especificaciones técnicas principales:

Lenguaje principal: Python 3.11

Interfaz gráfica: PySide6 (Qt para Python)

Base de datos: PostgreSQL alojada en Supabase (entorno cloud seguro con conexión SSL)

Sistema operativo objetivo: Windows 10/11

Distribución: Paquete ejecutable (.exe) autoextraíble mediante herramientas como PyInstaller o cx\_Freeze

Seguridad: Cifrado de contraseñas con bcrypt, validación de formularios, control de errores y logs

Arquitectura del software: Modelo-Vista-Controlador (MVC)

Modularidad: Módulos separados para login, usuarios, fichajes, clientes, vehículos, intervenciones, etc.

El sistema se diseñará con escalabilidad en mente, de forma que en fases posteriores pueda ampliarse con nuevos módulos (como facturación avanzada o control de inventario) sin reestructurar el núcleo del programa.

3.3 Priorización y secuenciación de las fases del proyecto (Ciclo de vida y metodología)

Se ha adoptado una metodología ágil, basada en iteraciones cortas y validaciones continuas por parte del cliente (Scrum adaptado). Las fases principales del ciclo de vida del proyecto son:

Análisis inicial de necesidades

Diseño modular del sistema

Desarrollo de núcleo funcional (login, usuarios, conexión DB)

Desarrollo progresivo por módulos temáticos (clientes, vehículos, fichajes...)

Integración, pruebas de validación y corrección de errores

Generación de ejecutable, documentación y entrega

Soporte, mantenimiento y propuestas de ampliación futura

Cada fase tiene entregables parciales, lo cual permite un control constante del progreso y la calidad.

3.4 Planificación detallada de la intervención (adaptada a 40 horas lectivas)

El desarrollo del proyecto se ha estructurado cuidadosamente para poder ejecutarse dentro del plazo asignado por el centro educativo, que equivale a 40 horas lectivas. Este marco temporal obliga a una planificación rigurosa, donde cada fase del proyecto debe ser abordada de forma progresiva, eficaz y sin desviaciones innecesarias.

Dado que no se permite el uso de diagramas visuales, la distribución temporal se explica detalladamente a continuación en formato narrativo:

Fase 1: Análisis y diseño preliminar (5 horas)

Durante esta etapa inicial se realiza un análisis profundo de las necesidades funcionales del cliente, se definen los requisitos técnicos y se estructura la arquitectura del sistema. Se elabora el diseño de la base de datos, se planifica el modelo de interacción entre los módulos, y se establece la división entre lógica de negocio y presentación (modelo MVC).

Fase 2: Implementación del núcleo del sistema (8 horas)

Esta fase se centra en construir la estructura base del programa, incluyendo la ventana principal, la lógica de navegación entre vistas, y el sistema de autenticación (login) con conexión segura a la base de datos PostgreSQL alojada en Supabase. También se configura el control de errores y se establece la estructura modular del proyecto.

Fase 3: Desarrollo de funcionalidades clave (16 horas)

Este bloque representa el grueso del trabajo. Incluye el desarrollo de los módulos principales del sistema:

Gestión de usuarios y control de roles.

Registro y consulta de clientes.

Registro de vehículos vinculados a los clientes.

Módulo de fichajes con reloj en tiempo real.

Historial de intervenciones con su respectivo formulario de registro y estado.

Cada módulo se programa de forma independiente, pero siguiendo una lógica coherente que permita su integración posterior.

Fase 4: Pruebas funcionales e integración (5 horas)

Se realiza la integración progresiva de los módulos y la ejecución de pruebas funcionales para validar que el sistema funciona correctamente en distintos escenarios. Se corrigen errores detectados, se ajustan estilos visuales y se verifican conexiones y consistencia de datos.

Fase 5: Documentación y empaquetado (4 horas)

En esta fase final se prepara toda la documentación técnica y del usuario. Se redactan los manuales de uso, se documenta el código fuente con comentarios explicativos, y se genera el ejecutable final. También se realiza una revisión general para asegurar que el producto cumple con todos los objetivos planteados.

Fase 6: Evaluación final y soporte (2 horas)

Se realiza una revisión detallada con el cliente final (taller ReyBoxes), se ajustan detalles menores según su feedback, y se prepara la entrega oficial. Se deja preparada una propuesta de mantenimiento y evolución futura.

Resumen de distribución del tiempo (sin gráficos):

|  |  |
| --- | --- |
| Fase | Horas estimadas |
| Análisis y diseño preliminar | 5 h |
| Desarrollo del núcleo del sistema | 8 h |
| Funcionalidades clave | 18 h |
| Pruebas e integración | 5 h |
| Documentación | 2 h |
| Evaluación y soporte final | 2 h |
| TOTAL | 40 h |

Esta planificación permite cumplir con los objetivos del proyecto dentro del marco formativo establecido por el centro, asegurando al mismo tiempo un desarrollo profesional, ordenado y funcional del sistema propuesto. Cada bloque se ha diseñado para que sea autónomo y medible, permitiendo una correcta gestión del tiempo y un seguimiento efectivo del progreso, incluso sin el apoyo de diagramas visuales.

3.5 Identificación de los recursos materiales y humanos necesarios

Recursos humanos:

Desarrollador principal (autor del proyecto)

Usuario cliente de referencia (taller ReyBoxes)

Colaboradores externos puntuales para pruebas o validación

Recursos materiales:

Ordenador con al menos 8 GB de RAM

Conexión a internet para acceder a Supabase

Herramientas de desarrollo: PyCharm, Git, navegador web, terminal

Servicios en la nube: Supabase, correo SMTP para pruebas de recuperación

Herramientas de documentación: Word, PDF, LaTeX (según entrega final)

3.6 Estimación de costes y necesidades de financiación

Este proyecto, al ser desarrollado dentro del contexto educativo, no requiere inversión económica directa, salvo el tiempo de dedicación y los recursos personales del desarrollador. Sin embargo, se realiza la siguiente estimación teórica de costes en un entorno profesional:

Horas de desarrollo: 40 h x 65 €/h = 2.600 €

Servidor PostgreSQL profesional (cloud): ~25 €/mes

Licencias PyInstaller/entornos profesionales: 0 € (uso de herramientas libres)

Diseño gráfico, iconos, logotipos personalizados: ~200 €

Total, estimado: 2825 € (sin contar soporte postventa)

En caso de escalar el sistema a otros talleres, podría contemplarse financiación externa o modelo SaaS.

3.7 Gestión de riesgos y elaboración del plan de prevención de incidencias

Riesgos identificados:

Fallo en la conexión con la base de datos remota

Errores en el empaquetado del ejecutable

Retrasos por sobrecarga de tareas externas (exámenes, prácticas)

Pérdida de datos por fallo no gestionado

Medidas preventivas:

Implementación de logs detallados y mensajes de error amigables

Validación constante en cada módulo antes de pasar al siguiente

Uso de sistema de backups automáticos desde Supabase

Control de versiones del código con Git

Además, se mantiene una documentación técnica continua, lo que facilita la rápida detección y corrección de errores.

3.8 Diseño de la documentación técnica del proyecto

La documentación técnica incluirá:

Manual de instalación y ejecución del sistema

Esquema de la base de datos (modelo relacional con claves primarias y foráneas)

Diagramas de clases, estructura del proyecto y relación entre módulos

Explicación del flujo de navegación y del uso de cada ventana

Registro de pruebas funcionales y resultados

Manual de usuario para el cliente final, con capturas y explicaciones paso a paso

3.9 Definición del plan de atención al cliente o usuario final

El sistema se entregará acompañado de:

Manual de usuario en PDF, con explicaciones detalladas de cada sección

Formulario de contacto por correo para incidencias o mejoras

Sistema de mensajes de error interpretables para ayudar al usuario a entender cualquier fallo

Ventanas intuitivas, con tooltips, íconos representativos y botones accesibles

Posibilidad de desarrollar en el futuro una versión portable para llevar en USB y usar en varios talleres sin necesidad de instalación compleja

4. Diseño del Proyecto

El diseño del sistema constituye una de las fases más críticas del proceso de desarrollo, ya que en él se definen con precisión las estructuras, comportamientos e interfaces que darán forma al producto final.

4.1 Diseño de casos de uso

El sistema está estructurado en torno a roles diferenciados de usuario y los casos de uso principales que cada uno de ellos necesita cubrir. A continuación, se describe el conjunto de funcionalidades accesibles según el perfil de acceso:

ROL ADMINISTRADOR

El perfil de Administrador es el de mayor nivel jerárquico dentro del sistema. Tiene acceso total a todas las funcionalidades disponibles, lo que le permite no solo operar como usuario del sistema, sino también gestionar la configuración, supervisar al personal y mantener el control sobre las operaciones clave del taller. Sus funciones incluyen:

Fichar entrada/salida para control horario.

Consultar el historial completo de fichajes de todos los empleados.

Crear, modificar y eliminar usuarios, así como asignarles roles y restablecer contraseñas.

Gestionar los datos de clientes, incluyendo altas, bajas, modificaciones y búsquedas.

Gestionar vehículos asociados a clientes, incluyendo ficha técnica y tipo de vehículo.

Realizar recepcionamientos de vehículos (registro de entrada para reparación o evaluación).

Elaborar y guardar presupuestos detallados de reparaciones o mantenimientos.

Acceder al módulo de compraventa, para registrar vehículos en stock, ventas, compras y traspasos.

Generar reimpresiones de recepcionamientos, presupuestos, comprobantes de compra y venta.

Cerrar sesión de forma segura para finalizar su actividad.

Este perfil está pensado para la dirección técnica o el propietario del taller.

ROL MECÁNICO:

El perfil Mecánico tiene acceso únicamente a las funcionalidades operativas relacionadas con su labor técnica. El sistema está diseñado para que pueda trabajar con rapidez, sin distracciones ni accesos innecesarios a funciones administrativas o comerciales.

Fichar su entrada y salida laboral.

Consultar su propio historial de fichajes.

Acceder al módulo de vehículos para visualizar y consultar información técnica necesaria para las reparaciones.

Registrar nuevos recepcionamientos de vehículos asignados para intervención.

Elaborar presupuestos preliminares cuando detecta una necesidad técnica durante la inspección.

Acceder al módulo de reparaciones, donde puede registrar el avance, el tipo de intervención realizada, el estado de la tarea y observaciones técnicas.

Cerrar sesión al finalizar su jornada o cambiar de turno.

El sistema limita el acceso del mecánico a módulos sensibles como creación de usuarios, gestión de clientes o compraventa, para evitar modificaciones no autorizadas.

ROL ADMINISTRATIVO

El perfil Administrativo se enfoca en tareas de gestión documental, atención al cliente y soporte a los procesos operativos del taller. Su perfil tiene amplias capacidades, aunque sin acceso a la gestión de usuarios ni al módulo de compraventa.

Fichar entrada/salida y consultar su propio historial.

Gestionar la base de clientes, incluyendo el alta de nuevos clientes, edición de datos y seguimiento.

Gestionar los vehículos asociados a clientes, mantener sus datos actualizados.

Registrar recepcionamientos y derivarlos a los técnicos o responsables correspondientes.

Elaborar y revisar presupuestos propuestos por los técnicos.

Realizar reimpresiones de documentos operativos: recepcionamientos, presupuestos, comprobantes de compra o venta.

Cerrar sesión para proteger el acceso a sus funciones.

Su rol es esencial para mantener la fluidez administrativa y documental del taller, así como para asistir a la gerencia y al personal técnico.

ROL COMPRA/VENTA

El perfil de Compra/Venta está orientado al personal encargado de gestionar el stock de vehículos, las operaciones de compraventa y los reportes comerciales asociados.

Fichar entrada y salida y consultar historial de fichajes.

Consultar y actualizar el módulo de vehículos, especialmente aquellos en venta o evaluación.

Registrar recepcionamientos relacionados con procesos de tasación, compra o entrega.

Acceder al módulo de compraventa, donde puede gestionar entradas de stock, datos de transacción, valoraciones, observaciones comerciales y cierres de venta.

Acceder al módulo de reportes, para generar documentación vinculada a las operaciones realizadas (informes de tasación, comparativas, márgenes, etc.).

Cerrar sesión de forma segura.

Este perfil carece de acceso a módulos técnicos o de creación de usuarios, ya que su función se centra en el aspecto comercial.

Control de permisos y seguridad

El control de accesos está implementado desde el momento del inicio de sesión. Según el rol asociado al usuario autenticado, el sistema muestra u oculta automáticamente los botones y módulos no permitidos. Esto evita accesos no autorizados y proporciona una interfaz más limpia y adaptada a cada tipo de usuario.

Además, si un usuario intenta acceder a una función restringida mediante medios no autorizados (por ejemplo, forzando la ruta o manipulando la interfaz), el sistema valida el rol activo y bloquea el acceso mostrando un mensaje de advertencia.

4.2 Diseño de clases y arquitectura de módulos

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.El sistema sigue una arquitectura modelo-vista-controlador (MVC) que garantiza la separación de responsabilidades. Esta organización facilita el mantenimiento, la escalabilidad y las pruebas unitarias.

Estructura de módulos:

Módulo: Main.py

Clase principal: Aplicacion

Tipo de clase: Clase central de inicialización y navegación de ventanas (controlador raíz)

Propósito general

El módulo Main.py contiene la clase principal del sistema: Aplicacion, que actúa como el núcleo de arranque, gestión de ventanas y navegación del flujo principal de la aplicación. Es el punto de entrada del programa y se encarga de:

Configurar el entorno gráfico.

Mostrar la pantalla de presentación al inicio.

Controlar el paso entre las ventanas: presentación → login → recuperación → panel principal.

Mantener las referencias activas a los controladores principales del sistema.

🧩 Estructura de la clase Aplicacion

🔹 Constructor \_\_init\_\_(self)

Inicializa la aplicación Qt (QApplication) usando los argumentos del sistema.

Establece el icono de la ventana principal mediante setWindowIcon, utilizando una ruta absoluta al archivo favicon.ico obtenida dinámicamente desde el módulo utilidades.rutas.

Crea y muestra la ventana de presentación (VentanaPresentacion), que funciona como pantalla inicial de carga/branding.

Define los atributos de instancia controlador\_login y controlador\_recuperar como None, preparándolos para su uso posterior.

Conecta la señal mostrar\_login desde la ventana de presentación a la función correspondiente que carga la ventana de login.

Esta fase de inicialización asegura que el sistema comience con una presentación visual y prepare los módulos necesarios sin abrir múltiples ventanas simultáneamente.

🔹 Método mostrar\_login(self)

Cierra la ventana de presentación si sigue abierta.

Crea una instancia del LoginControlador, que gestiona la lógica y eventos de la pantalla de inicio de sesión.

Conecta dos señales clave:

senal\_abrir\_recuperacion → llama a mostrar\_recuperar() para cambiar a la ventana de recuperación de cuenta.

senal\_login\_exitoso → llama a mostrar\_dashboard(usuario) cuando el login ha sido correcto.

Muestra la ventana de login al usuario.

Este método centraliza el paso del usuario a la pantalla principal del sistema tras la fase de bienvenida.

🔹 Método mostrar\_recuperar(self)

Cierra la ventana de login si está abierta (para evitar múltiples pantallas activas).

Crea una instancia del RecuperarControlador, que gestiona el proceso de recuperación de contraseña por correo electrónico.

Conecta la señal senal\_volver\_login a mostrar\_login() para permitir volver atrás desde la pantalla de recuperación.

Muestra la ventana de recuperación.

Este método se invoca solo cuando el usuario ha olvidado su contraseña y necesita iniciar el proceso de validación por código.

🔹 Método mostrar\_dashboard(self, usuario)

Recibe el diccionario usuario con los datos del login exitoso.

Crea una instancia del InicioControlador, pasando el nombre y rol del usuario autenticado.

Conecta la señal senal\_cerrar\_sesion al método mostrar\_login() para permitir cerrar sesión y volver a la pantalla de login.

Muestra la ventana principal del sistema (dashboard o menú de inicio) según el rol del usuario.

Este método adapta dinámicamente la pantalla de inicio a las funciones que puede realizar el usuario según su perfil.

🔹 Método ejecutar(self)

Inicia el bucle de eventos de Qt con self.app.exec().

Esta llamada bloquea el hilo principal hasta que la aplicación se cierre.

Este método debe ser invocado desde el if \_\_name\_\_ == "\_\_main\_\_" para lanzar toda la aplicación correctamente.

🧠 Conclusión técnica

La clase Aplicacion funciona como el controlador maestro del sistema, ya que:

Orquesta la apertura y cierre de todas las ventanas del programa.

Controla el flujo lógico de la navegación entre vistas.

Mantiene en memoria los controladores activos y sus señales.

Aísla la lógica de arranque del resto del sistema, lo que mejora la mantenibilidad.

Este enfoque garantiza una arquitectura limpia y modular, ya que cada ventana o controlador se encarga de su propia lógica interna, mientras Aplicacion únicamente gestiona cuándo y cómo se muestran.

🎨 Carpeta css/: Estilos personalizados del sistema

El sistema cuenta con una carpeta específica llamada css, en la cual se agrupan todos los archivos de estilos visuales personalizados, escritos en CSS, que definen la apariencia de cada una de las ventanas e interfaces del programa. Esta estrategia de organización busca garantizar la coherencia visual del sistema y facilitar el mantenimiento y modificación de estilos sin alterar la lógica funcional.

📁 Estructura general

Cada archivo .css dentro de esta carpeta corresponde directamente a una ventana o módulo del sistema, siguiendo una asociación uno a uno con los formularios definidos en vistas/. Esto permite aplicar estilos únicos y adaptados a la funcionalidad de cada pantalla, mejorando la experiencia del usuario y la claridad visual.

📄 Archivos destacados y su propósito

| Archivo CSS | Ventana asociada | Descripción funcional |
| --- | --- | --- |
| login.css | Pantalla de inicio de sesión | Estilos para campos de entrada, botón "Entrar", iconos e imagen de logo. |
| recuperar.css | Recuperación de cuenta | Estilo del formulario de correo, botón enviar código, fondo claro. |
| verificar\_codigo.css | Verificación de código OTP | Estilos para la cuenta atrás, campos de código y mensajes visuales. |
| restaurar.css | Restaurar contraseña | Estilo limpio, campos dobles, validación visual sin alertas intrusivas. |
| inicio.css | Panel principal (Dashboard) | Botones con iconos, disposición en cuadrícula, colores por rol. |
| clientes.css | Gestión de clientes | Formulario extenso, campos alineados, botones de acción con iconos. |
| crear\_cliente\_rapido.css | Alta rápida de cliente | Estilo compacto y visualmente limpio. |
| vehiculos.css | Gestión de vehículos | Tabla con estilos de fila alterna, botones de acción destacados. |
| presupuesto.css | Módulo de presupuestos | Estética formal, resaltado de totales, estilo de observaciones. |
| anadirTareaPresupuesto.css | Añadir tareas a presupuestos | Estilo de subtareas, resaltado por tipo de intervención. |
| recepcionamiento.css | Registro de recepción de vehículos | Diseño claro para firmas, campos técnicos y observaciones. |
| compraventa.css | Gestión de compraventa | Estilo de stock, tarjetas visuales, fondo blanco con detalles rojos. |
| reimpresionRecepcionamiento.css | Reimpresión de recepcionamientos | Ajustes de impresión y formato de campos. |
| reimpresionPresupuestos.css | Reimpresión de presupuestos | Estilo tipo factura, con márgenes y logotipo. |
| reimpresionCompras.css | Reimpresión de compras | Formato de tickets o comprobantes. |
| reimpresionVentas.css | Reimpresión de ventas | Estética limpia, énfasis en totales. |
| historial.css | Historial de fichajes | Tabla expandida, colores para diferenciar tipo de fichaje. |
| fichar.css | Fichaje de entrada/salida | Diseño con reloj visible, campos grandes y centrados. |
| usuarios.css | Gestión de usuarios | Estilo administrativo, botones agrupados, íconos por acción. |
| correo\_confirmacion.css | Estilo del correo HTML (confirmación) | Diseño con colores institucionales, título visual destacado. |

🧩 Importancia técnica

Separación de estilo y lógica: al mantener los estilos en archivos independientes, se sigue una buena práctica que mejora la mantenibilidad.

Modularidad visual: cada módulo tiene libertad para personalizar su interfaz sin interferir con otros.

Consistencia visual: gracias al uso de clases comunes (botones, inputs, iconos), se mantiene una línea estética uniforme.

Escalabilidad: si en el futuro se desea cambiar el diseño general, bastará con actualizar los archivos de estilos, sin tocar la lógica de cada ventana.

**📁 Carpeta documentacion/**

La carpeta documentacion/ contiene el archivo maestro del proyecto, correspondiente a la **memoria escrita del Trabajo Fin de Grado** (TFG), así como archivos temporales generados automáticamente por el editor de texto utilizado (Microsoft Word).

**Archivos incluidos:**

* **Documentacion\_TFG\_Taller\_ReyBoxes.docx**  
  Documento principal que recopila de forma estructurada todos los apartados requeridos por el centro educativo, incluyendo análisis, diseño, planificación, implementación, pruebas, anexos y conclusiones del sistema *ReyBoxes*. Contiene tanto el desarrollo técnico como la justificación empresarial del proyecto.

📁 Carpeta documentos/: Almacenamiento estructurado de documentos generados

La carpeta documentos/ constituye el repositorio principal del sistema para la generación y almacenamiento automático de documentación administrativa, generada durante el uso del sistema. Todos los documentos son exportados en formato PDF (y, en algunos casos, acompañados de archivos HTML o CSS para edición o diseño), organizados en subcarpetas por tipo y por mes.

🗂️ Estructura general:

/compras → Contratos de compra de vehículos

/ventas → Contratos de venta de vehículos

/presupuestos → Presupuestos generados para intervenciones

/recepcionamientos → Documentos de recepción de vehículos

Cada una de estas carpetas incluye subcarpetas por mes y año (APRIL\_2025, May\_2025, ABRIL\_2025, MAYO\_2025, etc.), lo cual permite una organización cronológica, facilitando la búsqueda y trazabilidad de los archivos. Adicionalmente, cada módulo contiene una carpeta /temp con archivos de estilo y recursos visuales necesarios para la generación de documentos.

📂 Descripción de cada subdirectorio:

compras/

Contiene contratos de compra de vehículos generados por el sistema, nombrados con el patrón:

CONTRATO\_COMPRA\_<matrícula>.pdf

Ej.: CONTRATO\_COMPRA\_9999ABC.pdf

Organizados por mes (ej. APRIL\_2025).

Subcarpeta temp/ con:

contrato\_compra.css: hoja de estilo utilizada para el diseño del contrato.

firmataller.png: firma gráfica del taller.

logo.jpg: logotipo institucional.

ventas/

Contiene los contratos de venta emitidos por el módulo de compraventa.

Nombrados como:

CONTRATO\_VENTA\_<referencia>.pdf

Ej.: CONTRATO\_VENTA\_6709XJT.pdf

Organización cronológica por mes (APRIL\_2025) y carpeta temp/ con archivos de diseño:

contrato\_venta.css y contrato\_venta\_generado.html: recursos HTML previos a la exportación en PDF.

firmataller.png, logo.jpg: elementos gráficos reutilizados en el documento final.

presupuestos/

Archivos PDF generados al realizar presupuestos.

Nomenclatura:

Presupuesto\_<matrícula>\_<fecha>\_<hora>.pdf

Ej.: Presupuesto\_4567DEF\_20250503\_1148.pdf

Organización mensual (May\_2025), sin subcarpeta de recursos, ya que se generan directamente con estilos embebidos o centralizados.

recepcionamientos/

Documentos que registran la recepción de vehículos en el taller, incluyendo datos del cliente, estado del vehículo y observaciones.

Nombres tipo:

Recepcion\_<NOMBRE\_CLIENTE>\_<ID>.pdf

Ej.: Recepcion\_CARLOS\_GOMEZ\_HERNANDEZ\_01002.pdf

Organización por mes (ABRIL\_2025, MAYO\_2025).

🔐 Observaciones técnicas

La estructura modular y cronológica permite una búsqueda rápida de documentos incluso sin base de datos.

Los archivos .css y .html permiten una edición previa al renderizado PDF, o pueden ser utilizados en motores de conversión como wkhtmltopdf.

El uso de nombres estructurados facilita la trazabilidad por matrícula, cliente o fecha.

Todos los documentos están generados automáticamente por el sistema, sin intervención manual, lo que garantiza consistencia y reducción de errores.

**Carpeta font/: Tipografía personalizada**

La carpeta font/ contiene las **fuentes tipográficas utilizadas por el sistema** para garantizar una apariencia moderna, homogénea y visualmente agradable en toda la interfaz gráfica. Estas fuentes se aplican tanto a los títulos como a los textos secundarios, y forman parte esencial del diseño visual definido por los archivos CSS.

**📄 Archivos incluidos:**

* **Montserrat-VariableFont\_wght.ttf**  
  Fuente principal en estilo normal (*Roman*). Es una fuente de tipo **variable**, lo que permite modificar su peso (grosor) dinámicamente, desde Thin hasta Black, sin necesidad de múltiples archivos estáticos.
* **Montserrat-Italic-VariableFont\_wght.ttf**  
  Variante **itálica** de la misma fuente, también en formato variable. Se utiliza especialmente en subtítulos, observaciones y textos decorativos o secundarios.

**🎨 Justificación del uso**

La familia **Montserrat** fue seleccionada por sus características de **legibilidad, elegancia y modernidad**, alineadas con la imagen profesional del sistema ReyBoxes. El uso de versiones *variable font* permite:

* **Reducir el peso del sistema** (solo dos archivos en lugar de muchos por cada peso/estilo).
* **Mayor flexibilidad visual**, ya que se puede controlar con precisión el grosor del texto desde CSS o desde PySide.
* **Compatibilidad multiplataforma**, ya que las fuentes TrueType (.ttf) son soportadas tanto en Windows como en otros sistemas si se requiere expansión futura.

**📦 Integración técnica**

Las fuentes se cargan desde el código utilizando rutas absolutas, mediante funciones del módulo utilidades.rutas, asegurando que no dependan de rutas estáticas ni del entorno del sistema operativo.

**🖼️ Carpeta img/: Recursos gráficos del sistema**

La carpeta img/ contiene todos los **recursos visuales e iconográficos** utilizados por la interfaz del sistema *ReyBoxes*, centralizando en un único directorio todos los elementos gráficos necesarios para los botones, formularios, paneles y documentos generados por la aplicación.

Esta organización favorece la reutilización de recursos, la coherencia estética y la facilidad de mantenimiento del sistema visual.

**📄 Archivos incluidos (por tipo)**

**📌 Iconos funcionales**

Utilizados en botones, menús, etiquetas y formularios para representar visualmente acciones o entidades:

* **Acciones comunes:**
  + aceptar.png, borrar.png, crear.png, enviar.png, confirmar.png, guardar.png, salir.png, volver.png, modificar\_registro.png, borrar\_registro.png, recargar.png, check.png, exit.png.
* **Navegación e interacción:**
  + flecha.png, escoba.png, mas.png, menos.png.
* **Gestión de documentos y exportación:**
  + pdf.png, csv.png, imprimir.png, reimprimir.png, reimprimir\_compra.png, reimprimir\_presupuestos.png, reimprimir\_recepcionamiento.png, reimprimir\_venta.png.

**🌀 Animaciones y carga**

* Subcarpeta gif/:
  + carga1.gif, carga2.gif, carga3.gif, carga4.gif, gifCarga.gif: usados como indicadores animados durante operaciones de espera, carga o validación de datos.

**🧠 Consideraciones técnicas**

* **Rutas absolutas:** Todas las imágenes se cargan mediante rutas absolutas calculadas con el módulo utilidades.rutas, asegurando compatibilidad entre sistemas y evitando fallos por rutas relativas.
* **Formato estándar:** Todos los iconos están en formato .png o .jpg para compatibilidad con PySide6. Los GIF permiten animaciones ligeras y sin dependencias externas.
* **Modularidad visual:** La clara segmentación por función y módulo facilita tanto la personalización como el reemplazo de iconos en futuras versiones.

**📚 Carpeta lib/: Entorno de librerías Python embebido**

La carpeta lib/ contiene un entorno **aislado** de dependencias Python requerido para que la aplicación *ReyBoxes* funcione de forma **autónoma y sin instalación externa de paquetes**, ideal para su empaquetado con tecnologías como PyInstaller, cx\_Freeze o incluso para ejecución directa en entornos bloqueados (como Windows corporativos).

Está estructurado como una instalación personalizada de site-packages, que es el directorio donde Python guarda las librerías de terceros.

**🧩 Librerías principales agrupadas por propósito**

**🔧 1. Gestión de paquetes y entornos**

* **pip**: Incluye el gestor de paquetes oficial de Python con todos sus submódulos (install, download, freeze, uninstall, etc.), así como el backend interno (\_internal) y controladores para configuración, resolución de dependencias y entornos virtuales.
* **setuptools\_build, wheel, pyproject\_hooks, distlib**: Herramientas necesarias para construir, empacar e instalar módulos Python desde archivos fuente o binarios (.whl).
* **platformdirs**: Utilidad para detectar rutas estándar de configuración/cache según sistema operativo.
* **truststore**: Utilidad para trabajar con certificados del sistema en lugar de certifi.

**🌐 2. Red, HTTP y descarga de paquetes**

* **requests**: Librería robusta para hacer peticiones HTTP/S con soporte para autenticación, sesiones, cookies, cabeceras personalizadas, etc.
* **urllib3**: Backend que utiliza requests para conexiones HTTP, con manejo avanzado de conexiones, reintentos, proxy, SSL y compatibilidad con múltiples plataformas.
* **certifi**: Conjunto de certificados raíz de confianza usado por requests.
* **idna**: Soporte para dominios internacionales (IDN).
* **distro, resolvelib, packaging**: Utilidades de detección de entorno Linux y de resolución semántica de versiones de paquetes.

**🧰 3. Utilidades de consola y depuración**

* **rich**: Librería de formato para consola, que permite imprimir tablas, barras de progreso, errores estilizados, JSON, Markdown, y más.
* **pygments**: Motor de resaltado de sintaxis (coloreado) para múltiples lenguajes. Se puede usar junto a rich, o para generar documentación con código.
* **diagnose, pretty, markup, console, traceback, panel, table, progress (submódulos de rich)**: Proveen interfaz enriquecida para el usuario final y feedback visual en CLI.

**📦 4. Compatibilidad y abstracciones**

* **typing\_extensions**: Define tipos avanzados para versiones de Python que aún no los integran oficialmente.
* **pkg\_resources, importlib, entrypoints, metadata**: Utilidades para inspeccionar paquetes instalados, sus metadatos y sus puntos de entrada dinámicos.
* **msgpack**: Librería de serialización binaria ultrarrápida (alternativa a JSON, útil para enviar datos compactos).
* **py.typed**: Archivos que indican que las librerías incluyen anotaciones de tipos para herramientas como mypy.

**💡 ¿Por qué incluir esta carpeta?**

Esta estructura se incluye en el proyecto para que:

* No sea necesario instalar dependencias con pip install manualmente.
* El sistema funcione **offline** (especialmente útil en instalaciones en taller o en la versión empaquetada como .exe o .apk).
* Se garantice la compatibilidad con Python embebido y versiones específicas.

**🧠 Carpeta modelos/: Consultas y conexión a la base de datos**

Esta carpeta contiene los **módulos encargados de comunicarse con la base de datos PostgreSQL** (en Supabase), realizando operaciones CRUD específicas para cada entidad del sistema (*usuarios, clientes, fichajes, vehículos, etc.*).

**🗂 Estructura general**

* Todos los archivos .py siguen el patrón nombreEntidad\_consultas.py, cumpliendo la función de **repositorio de consultas SQL o de acceso ORM**.
* El archivo conexion\_bd.py es el **punto común de conexión a la base de datos**.

**🔌 Conexión y núcleo**

| **Archivo** | **Descripción** |
| --- | --- |
| conexion\_bd.py | Contiene la lógica para establecer conexión a la base de datos Supabase. Es usado por todos los demás módulos. |

**👤 Gestión de usuarios y login**

| **Archivo** | **Función** |
| --- | --- |
| usuarios\_consultas.py | Inserción, eliminación, modificación y validación de usuarios. |
| login\_consultas.py | Funciones específicas para autenticar usuarios (por nombre, rol, contraseña cifrada, etc.). |
| recuperar\_consultas.py | Módulo para recuperación de cuentas (OTP, expiración, verificación del código). |

**🧾 Gestión de clientes y vehículos**

| **Archivo** | **Función** |
| --- | --- |
| clientes\_consultas.py | Inserción, búsqueda, modificación y eliminación de clientes. Soporta autocompletado por nombre, DNI, teléfono. |
| vehiculos\_consultas.py | Gestión de vehículos relacionados a un cliente. Trabaja con tipos de vehículo. |
| nuevoCliente\_compraventa\_consulta.py | Casos donde se registra un cliente directamente desde un flujo de compraventa (compra o venta). |

**🔄 Compraventa y contratos**

| **Archivo** | **Función** |
| --- | --- |
| compraventa\_consulta.py | Inserción y consulta de contratos de compra y venta. Usa identificadores de matrícula. |

**🧾 Presupuestos y recepcionamiento**

| **Archivo** | **Función** |
| --- | --- |
| presupuesto\_consultas.py | Registro y consulta de presupuestos generados. Contempla fecha, cliente, estado. |
| recepcionamiento\_consultas.py | Inserción y lectura de recepcionamientos de vehículos en el taller (cliente, fecha, motivo). |

**🕒 Fichajes e historial**

| **Archivo** | **Función** |
| --- | --- |
| fichajes\_consultas.py | Inserción de fichajes (entrada/salida), con fecha, hora y usuario. |
| historial\_consultas.py | Consulta de historial de fichajes para un usuario o todos (según el rol). Usado para informes y exportaciones. |

**✅ Observaciones adicionales**

* Todos los módulos .py están correctamente compilados para las versiones de Python 3.11 y 3.13, lo que garantiza compatibilidad cruzada.
* La estructura está bien organizada, separando responsabilidades por entidad del dominio del taller mecánico.
* Cumple con las buenas prácticas del patrón MVC, donde el modelo se encarga **únicamente de interactuar con la base de datos** sin lógica de interfaz ni control de ventanas.

**🧾 Carpeta plantillas/: Plantillas HTML y CSS para documentos del taller**

Esta carpeta contiene **plantillas base reutilizables** para generar documentos en formato HTML + CSS que luego se imprimen o exportan como PDF desde la aplicación.

**📄 Plantillas de contratos**

| **Archivo** | **Descripción** |
| --- | --- |
| contrato\_compra\_concesionario.html | Plantilla HTML para contratos de compra de vehículos por parte del taller. |
| contrato\_venta\_concesionario.html | Plantilla HTML para contratos de venta de vehículos desde el taller. |
| contrato\_compraventa.css | Estilos CSS aplicados a ambos contratos de compra y venta. |

**📑 Plantilla de presupuestos**

| **Archivo** | **Descripción** |
| --- | --- |
| plantilla\_presupuesto.html | Plantilla HTML utilizada para generar presupuestos formales entregados al cliente. |
| plantilla\_presupuesto.css | Estilo visual correspondiente a la plantilla de presupuesto. |

**🧾 Plantilla de recepcionamiento**

| **Archivo** | **Descripción** |
| --- | --- |
| recepcionamiento.html | Plantilla HTML para la ficha de recepción del vehículo en el taller. |
| recepcionamiento.css | Estilo visual correspondiente a la plantilla de recepcionamiento. |

✅ Observaciones

Estas plantillas están diseñadas para ser rellenadas dinámicamente desde Python o PyQt6 (por ejemplo, con Jinja2 o mediante sustitución de tokens) y luego renderizadas a PDF.

Todas las plantillas están modularizadas: el estilo se encuentra en archivos .css separados, lo cual facilita realizar modificaciones visuales sin alterar la estructura HTML.

La carpeta img se encuentra dentro de plantillas/, lo que permite el uso de rutas relativas funcionales para su integración directa en los documentos HTML.

Uno de los motivos para implementar los informes de esta manera fue la evaluación de los requisitos del proyecto y la necesidad de aplicar los conocimientos adquiridos en el módulo de "Lenguaje de Marcas", evitando la penalización por uso de entorno web. Esta solución representa la opción más beneficiosa y coherente con los objetivos académicos y técnicos del proyecto.

**📁 tests/ – Pruebas unitarias y funcionales del sistema**

Este directorio contiene **scripts de test independientes** para cada módulo funcional del sistema. Están escritos en Python y cubren validaciones, flujos de trabajo y generación de documentos.

**🧪 Pruebas funcionales por módulo**

**anadirTareaPresupuesto\_test.py**

* Verifica la correcta adición de tareas a un presupuesto activo.
* Incluye pruebas para entradas válidas e inválidas (campos vacíos, formatos numéricos erróneos, etc.).
* Asegura que la tarea se guarde con el formato esperado y se actualice correctamente el coste total.

**presupuesto\_test.py**

* Test de generación de presupuestos en HTML/PDF.
* Simula la creación de varios presupuestos y verifica el formato final del documento.
* Pruebas de consistencia en datos como cliente, vehículo, tareas, fecha y firma.

**compraventa\_test.py**

* Evalúa la lógica de creación de contratos de compra y venta.
* Comprueba la asociación entre cliente y vehículo.
* Simula firmas, generación del documento final y verifica rutas de guardado.

**nuevoCliente\_compraventas\_test.py**

* Valida el registro rápido de clientes desde la interfaz de compraventa.
* Asegura que se evite duplicidad de DNI y se rellenen correctamente los campos.
* Testea el flujo completo hasta la integración con el módulo de contratos.

**recepcionamiento\_test.py**

* Comprueba el registro de recepciones de vehículos al taller.
* Evalúa si los datos de entrada (cliente, motivo, observaciones) se guardan correctamente.
* Verifica la generación de los formularios asociados.

**reimpresionRecepcionamineto\_test.py**

* Test dedicado a la **reimpresión** de recepcionamientos anteriores.
* Simula búsquedas por fecha o cliente y genera nuevamente el documento.
* Verifica la integridad del HTML renderizado y el correcto uso de imágenes (firma/logo).

**📬 Pruebas de correo y recuperación**

**correoConfirmacion\_test.py**

* Test de la lógica que envía correos tras operaciones importantes (registro, cambios).
* Verifica que el asunto, contenido y destinatario sean correctos.
* Comprueba la inclusión de estilos e imágenes embebidas.

**correo\_test.py**

* Test genérico del sistema de envío de emails.
* Incluye pruebas con y sin archivos adjuntos, errores de conexión y fallos de autenticación.

**verificar\_test.py**

* Comprueba el flujo de verificación de código de recuperación (OTP).
* Incluye pruebas con código correcto, caducado y erróneo.
* Verifica el cálculo de cuenta regresiva y control de expiración.

**main\_verificar\_test.py**

* Simula el comportamiento de la ventana de verificación dentro del flujo completo.
* Comprueba que tras la verificación correcta se cargue la ventana de restablecer contraseña.

**👤 Pruebas de usuario, login y seguridad**

**bcrypt\_test.py**

* Valida el funcionamiento del cifrado y comparación de contraseñas con bcrypt.
* Incluye pruebas para asegurar que el hash generado varía por cada entrada y que la verificación sea exacta.

**inicio\_test.py**

* Simula intentos de login válidos e inválidos.
* Evalúa mensajes de error, formato del nombre de usuario, y comportamiento ante contraseñas incorrectas.

**fichar\_test.py**

* Test del sistema de fichaje de empleados (entrada y salida).
* Verifica que el registro contenga fecha, hora, usuario y tipo de fichaje.
* Comprueba que no se puedan registrar fichajes inválidos o duplicados seguidos.

**📚 Otros tests auxiliares**

**historial\_test.py**

* Verifica la carga y visualización del historial de fichajes.
* Comprueba la paginación, exportación y filtrado por usuario y fechas.

**ruta\_test.py**

* Test de utilidades de sistema de archivos: rutas relativas, detección de recursos (iconos, firmas).
* Asegura compatibilidad multiplataforma (Windows/Linux/Mac).

**vehiculos\_test.py**

* Valida el alta, búsqueda y modificación de vehículos.
* Prueba la asociación con el cliente y validaciones como matrícula única, tipo, marca y modelo.

**✅ Observaciones generales**

* Cada archivo de test se puede ejecutar individualmente desde la consola (python nombre\_test.py).
* Están preparados para integrarse fácilmente con herramientas de testing como pytest o unittest.
* Su estructura modular permite añadir nuevos tests sin interferir con los ya existentes.

**⚙️ Scripts/ – Ejecutables de gestión de paquetes Python**

Este directorio contiene los ejecutables relacionados con la gestión de paquetes mediante **pip**, el instalador oficial de paquetes de Python. Son usados generalmente por entornos virtuales o configuraciones portables del proyecto.

**📄 Contenido de los archivos**

**pip.exe**

* Ejecutable principal de pip, el gestor de paquetes Python.
* Permite instalar, actualizar o eliminar paquetes mediante comandos como:

pip install nombre\_paquete

**pip3.exe**

* Alias explícito para la versión de pip asociada a **Python 3**.
* Útil en sistemas donde coexisten Python 2 y 3.
* Equivalente funcional a pip, pero garantiza compatibilidad con entornos Python 3.x.

**pip3.13.exe**

* Ejecutable específico para **Python 3.13**.
* Asegura que los paquetes se instalan en el entorno correspondiente a la versión 3.13 (como en este proyecto).
* Es particularmente útil en sistemas donde se manejan múltiples versiones de Python (ej. 3.11, 3.13, etc.).

**✅ Observaciones**

* Estos ejecutables permiten gestionar las dependencias desde consola o desde código.
* Se encuentran dentro del proyecto para garantizar que las instalaciones se hagan en el entorno Python **embebido o aislado** del sistema.

**🧰 utilidades/ – Módulos auxiliares del sistema ReyBoxes**

Este directorio contiene scripts modulares diseñados para **encapsular funcionalidades específicas** utilizadas en distintas partes del sistema, facilitando la reutilización, el mantenimiento y la legibilidad del código.

**📦 Módulos principales**

**🔄 abridor\_con\_carga.py**

* Abre ventanas nuevas del sistema mostrando primero una **pantalla de carga animada** (con GIF o barra).
* Mejora la experiencia de usuario en vistas que tardan en renderizarse.
* Se utiliza, por ejemplo, al entrar a módulos como presupuesto, historial o recepcionamiento.

**🎨 boton\_animado.py**

* Contiene una clase para botones con **efectos visuales o animaciones** al hacer hover o clic.
* Compatible con PyQt6 y diseñado para integrarse con los estilos CSS personalizados del sistema.

**📄 canvas\_con\_paginas.py**

* Implementa un sistema de paginación para mostrar datos extensos en un QWidget, como tablas o listas grandes.
* Facilita navegar entre páginas sin recargar toda la ventana.

**✍️ capturador\_firma.py**

* Proporciona una interfaz para que el usuario **dibuje y guarde su firma** manualmente.
* Guarda la firma en una imagen PNG para usar en presupuestos, contratos, etc.
* Puede incluir herramientas como "Limpiar" y "Guardar firma".

**🆔 comprobar\_dni.py**

* Valida si un **DNI español es correcto** (formato y letra).
* Incluye también una función para **generar DNIs aleatorios válidos** para pruebas.

**📧 correo.py**

* Función central para **enviar correos electrónicos**.
* Se usa internamente por otros módulos como correo\_contratos, correo\_presupuesto, etc.
* Soporta HTML, adjuntos y direcciones múltiples.

**📬 Módulos específicos para el envío de correos**

* **correo\_contratos.py** → Envío del **contrato de compra o venta** al cliente.
* **correo\_presupuesto.py** → Envío de **presupuestos** generados como PDF.
* **correo\_recepcionamiento.py** → Confirmación del **recepcionamiento del vehículo**.
* **correo\_reenviarCompras.py**, correo\_reenviarVentas.py → Reenvío de contratos.
* **correo\_reenviar\_recepcionamiento.py**, correo\_renviarPresupuestos.py → Reenvío de formularios o presupuestos previamente firmados.

Todos estos módulos se basan en plantillas HTML y el motor correo.py, pero añaden su lógica de selección de documentos, generación de PDF y contenido personalizado.

**🖨️ imprimir.py**

* Contiene utilidades para **previsualizar o enviar a imprimir** documentos generados (presupuestos, contratos, fichas de recepción, etc.).
* Puede interactuar con sistemas de impresión del SO.

**💬 mensajes.py**

* Define ventanas emergentes o diálogos de aviso como:
  + "¿Estás seguro de continuar?"
  + "Correo enviado correctamente"
  + "Error al conectar con la base de datos"
* Mejora la uniformidad y estética de los mensajes del sistema.

**📄 pdf.py**

* Contiene funciones para **convertir HTML + CSS en documentos PDF** usando motores como WeasyPrint o Qt.
* Se utiliza por todos los módulos de impresión y correo electrónico.

**🛠️ recepcionamiento\_utilidades.py**

* Agrupa funciones utilizadas durante el **proceso de recepción del vehículo**.
* Puede incluir formateo de fechas, generación de IDs, validaciones o cálculos auxiliares para los formularios.

**🧭 rutas.py**

* Define rutas absolutas **dinámicas** en función de la ubicación actual del programa.
* Evita dependencias de rutas absolutas de sistema operativo.
* Incluye rutas a recursos: imágenes, plantillas, bases de datos, PDFs generados, etc.

**💾 rutas\_guardado.py**

* Define rutas de **destino para guardar archivos** generados (PDFs, firmas, contratos).
* Permite que todas las funciones usen rutas coherentes según el tipo de documento.

**🧵 Subdirectorio hilos/ – Tareas en segundo plano**

**hilo\_generico.py**

* **Clase base que encapsula la lógica para crear hilos (QThread) reutilizables en otras partes del sistema.**
* **Permite ejecutar procesos intensivos o de larga duración sin bloquear la interfaz de usuario, garantizando así una experiencia fluida y profesional.**
* **Es la base para extender múltiples operaciones asincrónicas, como la generación de informes, carga de datos o procesamiento de archivos.**

**hilo\_carga\_recepcionamiento.py**

* **Hilo especializado en la carga dinámica de datos de recepcionamiento de vehículos directamente desde la base de datos.**
* **Optimiza el rendimiento general del sistema permitiendo que la información se cargue de forma paralela, sin interrumpir la interacción con el usuario.**
* **Este módulo fue clave para aplicar de manera práctica los conocimientos adquiridos en el módulo formativo de “Programación de Servicios y procesos”, ya que representa una implementación real de multitarea eficiente, aplicando los principios de asincronía, concurrencia controlada y diseño orientado a eventos.**
* **Además, su desarrollo supuso un avance importante en la profesionalización del sistema, consolidando un enfoque moderno y robusto en la arquitectura de la aplicación.**

**🧪 get-pip.py**

* Script oficial de Python que instala pip en caso de que no esté presente.
* No suele ser ejecutado directamente desde el sistema salvo en instalaciones de emergencia.

**✅ Observaciones finales**

* Este módulo actúa como **nucleo de lógica de presentación** y automatización del sistema, asegurando:
  + Reutilización máxima de código.
  + Separación de lógica por contexto (firma, impresión, email, rutas, hilos...).
  + Extensibilidad para futuras funcionalidades.

**🪟 Directorio vistas/ – Interfaz gráfica de usuario (GUI)**

Este directorio contiene **todas las ventanas** que conforman la interfaz gráfica del sistema, desarrolladas en **PyQt6**. Cada archivo representa una vista específica, siguiendo el patrón **MVC** implementado a lo largo del proyecto. Su estructura modular permite que cada ventana sea independiente, reutilizable y personalizable.

**✅ Características generales:**

* Todas las clases heredan de QWidget, QDialog o QMainWindow según la funcionalidad de la ventana.
* El diseño visual se encuentra generalmente reforzado mediante archivos .css personalizados en la carpeta css/, lo que permite un estilo coherente y profesional.
* Los nombres de los archivos son altamente descriptivos y están en castellano, facilitando la lectura del código y el mantenimiento futuro.

**📌 Ventanas principales del sistema**

**ventana\_presentacion.py**

* Splash screen animado que da la bienvenida al sistema.
* Presenta el logo del taller y sirve como **pantalla de carga** inicial antes de mostrar el login.

**ventana\_login.py**

* Formulario de inicio de sesión.
* Incluye diseño personalizado con degradado, campos con iconos integrados y validaciones básicas.

**ventana\_inicio.py**

* **Menú principal** tras el login.
* Muestra un saludo personalizado y una cuadrícula de botones de navegación dependiendo del rol del usuario.

**🔒 Autenticación y recuperación**

**ventana\_recuperar.py, ventana\_verificar.py, ventana\_restaurar.py**

* Ventanas encadenadas para recuperación de contraseña:
  + **Recuperar**: Introducir el correo.
  + **Verificar**: Validar código recibido.
  + **Restaurar**: Cambiar contraseña con validación.

**👥 Gestión de usuarios y clientes**

**ventana\_usuarios.py**

* Gestión de usuarios del sistema: creación, edición y eliminación.

**ventana\_clientes.py, ventana\_crear\_cliente\_rapido.py**

* Gestión de clientes con autocompletado, búsquedas por múltiples campos, y asociación con vehículos.

**🚗 Gestión de vehículos**

**ventana\_vehiculos.py**

* Ventana dedicada al registro y visualización de vehículos asociados a clientes.

**🛠️ Operaciones de taller**

**ventana\_recepcionamiento.py, ventana\_reimpresionRecepcionamiento.py**

* Recepción de vehículos y reimpresión de documentos asociados.

**ventana\_presupuesto.py, ventana\_dialogoCorreoPresupuesto.py, ventana\_anadirTareaPresupuesto.py, ventana\_reimpresionPresupuestos.py**

* Gestión integral de presupuestos:
  + Añadir tareas, enviar por correo, y reimprimir.

**📄 Contratos y compraventa**

**ventana\_compraventa.py, ventana\_contrato.py, ventana\_nuevoCliente\_compraventas.py, ventana\_reimpresionCompras.py, ventana\_reimpresionVentas.py**

* Módulo completo de compraventa con generación de contratos, reimpresiones y asociación rápida de nuevos clientes.

**⏱️ Fichajes y control horario**

**ventana\_fichar.py, ventana\_historial.py**

* Registro de entradas/salidas con reloj en tiempo real.
* Historial de fichajes consultable y exportable.

**📬 Correos y confirmaciones**

**ventana\_correo\_confirmacion.py**

* Confirmación de envíos de correos (presupuestos, contratos, etc.) al cliente.

**🧪 Ventanas auxiliares**

**ventana\_carga\_gif.py**

* Animación de carga en formato GIF durante operaciones pesadas.

**ventana\_pestanaGenerica.py**

* Clase base reutilizable para insertar pestañas dinámicas en otras ventanas.

Este directorio constituye el núcleo visual de la aplicación **ReyBoxes**, reflejando la atención al detalle y el enfoque profesional en la experiencia del usuario. Cada vista ha sido diseñada para ofrecer **claridad, facilidad de uso y una estética coherente** con el resto del sistema.

Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.4.3 Diseño físico de dato

**🧾 Tabla: fichajes**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla fichajes constituye el núcleo del módulo de control horario del sistema de gestión del taller. Está diseñada para registrar cada entrada y salida de los empleados, permitiendo llevar un control exacto de su jornada laboral.

Cada fichaje queda asociado de forma indivisible a un usuario del sistema, lo cual permite generar informes individualizados, detectar ausencias o anomalías y calcular tiempos de permanencia. Esta tabla también permite extender el sistema hacia la gestión de nóminas o productividad.

La estructura es simple, pero poderosa: cuenta con campos bien definidos y relaciones referenciales sólidas que garantizan la integridad de los datos. Su diseño cumple con los principios de normalización y se adapta a múltiples escenarios laborales.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Permite nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del fichaje. Se autoincrementa. |
| usuario\_id | int4 | ❌ No | Clave foránea que enlaza con usuarios.id. Indica qué usuario ha fichado. |
| tipo | varchar | ❌ No | Tipo de fichaje: 'Entrada' o 'Salida'. Controlado desde la lógica de negocio. |
| fecha\_hora | timestamp | ❌ No | Fecha y hora exacta en la que se realizó el fichaje. |

**🔗 3. Relaciones y claves foráneas**

* usuario\_id → usuarios.id: Relación directa con la tabla usuarios, estableciendo qué persona realizó el fichaje. Esto permite consultar el historial individual de cada usuario.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY)**
   * id: Asegura unicidad del registro. Cada fichaje es irrepetible e inmutable.
2. **Restricciones de obligatoriedad (NOT NULL)**
   * Todos los campos (usuario\_id, tipo, fecha\_hora) son obligatorios para evitar registros incompletos o inconsistentes.
3. **Restricción de integridad referencial (FOREIGN KEY)**
   * usuario\_id debe existir previamente en la tabla usuarios. Esto impide fichajes de usuarios no registrados o eliminados.
4. **Restricción lógica (desde aplicación)**
   * El campo tipo acepta únicamente los valores 'Entrada' o 'Salida', aunque esta lógica puede implementarse desde el backend en lugar de en la base de datos.

**🔍 5. Consideraciones adicionales**

* Los datos almacenados en esta tabla permiten calcular de forma sencilla los intervalos de tiempo entre fichajes.
* Puede integrarse con sistemas biométricos, tarjetas NFC u otros métodos de identificación.
* El campo fecha\_hora almacena con precisión temporal los registros, permitiendo auditoría precisa.
* Posibilidad futura de añadir campos adicionales como ubicación, dispositivo, o comentarios.

**🛠️ 6. Uso en el sistema**

**⌛ 1. Módulo de fichajes**

* La ventana de fichaje permite al usuario seleccionar si desea fichar entrada o salida.
* Al pulsar “Confirmar”, se inserta automáticamente un nuevo registro en esta tabla, con el usuario\_id actual, el tipo seleccionado y la hora exacta.

**🖥️ 2. Interfaz de usuario**

* El sistema muestra la hora actual en tiempo real para que el usuario vea cuándo está fichando.
* Se bloquea el botón de cierre de la ventana para forzar la acción del usuario y evitar errores.

**📊 3. Historial y control horario**

* El administrador puede ver el historial de fichajes completo, filtrado por fechas, usuarios o tipo de fichaje.
* Se pueden exportar estos registros a PDF o CSV para reportes de asistencia y auditoría.

**🔐 4. Seguridad y trazabilidad**

* Como cada registro está vinculado a un usuario, es posible detectar comportamientos anómalos, como fichajes duplicados, o entradas sin salidas.
* Los datos no se pueden modificar desde la aplicación para preservar su validez como evidencia.

**🧾 Tabla: usuarios**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla usuarios constituye una de las entidades centrales del sistema. Está diseñada para almacenar los datos de las personas autorizadas a acceder al software, así como su rol, credenciales y metadatos relacionados con el acceso. Es fundamental para el control de acceso, la trazabilidad y la personalización del sistema según el perfil del usuario.

La tabla está normalizada y conectada con otras entidades a través de claves foráneas, garantizando así integridad referencial. Asimismo, incluye mecanismos de seguridad como el cifrado de contraseñas y la gestión segura de recuperación de cuentas.

**📐 Estructura de campos**

**2. Estructura de la tabla**

| **Campo** | **Tipo de dato** | **Permite nulos** | **Descripción** |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria, identificador único del usuario. Se autoincrementa. |
| nombre | varchar | ❌ No | Nombre de usuario (en mayúsculas), usado en el login. Unicidad garantizada. |
| apellido | varchar | ❌ No | Apellido(s) del usuario. |
| email | varchar | ❌ No | Correo electrónico único. Utilizado para notificaciones y recuperación de cuenta. |
| password | text | ❌ No | Contraseña cifrada con bcrypt. No se almacena en texto plano. |
| created\_at | timestamp | ✅ Sí | Fecha de creación del usuario. Útil para trazabilidad y auditorías. |
| updated\_at | timestamp | ✅ Sí | Fecha de la última modificación del registro. |
| codigo\_recuperacion | varchar | ✅ Sí | Código OTP generado para recuperación de cuenta. |
| expiracion\_codigo | timestamp | ✅ Sí | Fecha de expiración del código de recuperación. |
| rol\_id | int4 | ❌ No | Clave foránea que enlaza con roles.id. Determina los permisos del usuario. |

**Relaciones y claves foráneas**

* **rol\_id → roles.id: Esta relación impone una restricción referencial que garantiza que todos los usuarios tengan un rol válido asignado, ya sea “Administrador”, “Mecánico” , “Administrativo” o “Compraventa”. Esta relación es esencial para controlar los permisos y accesos en el sistema.**

**🔐 Restricciones**

1. **Clave primaria (PRIMARY KEY):  
   El campo id asegura la unicidad y sirve como referencia desde otras tablas como fichajes, intervenciones, etc.**
2. **Restricciones de unicidad (UNIQUE):  
   Tanto el campo nombre como el email están marcados como únicos. Esto evita duplicidades, mejora la trazabilidad de usuarios y previene conflictos en el inicio de sesión o en envíos de notificaciones.**
3. **Restricciones de obligatoriedad (NOT NULL):  
   Los campos nombre, apellido, email, password y rol\_id son obligatorios para garantizar la coherencia y completitud de los datos al registrar un usuario.**
4. **Restricciones de integridad referencial (FOREIGN KEY):  
   El campo rol\_id garantiza que sólo se puedan asignar roles definidos previamente en la tabla roles.**

**🔍 Consideraciones adicionales**

* **El diseño de esta tabla permite implementar un sistema de recuperación de contraseña sin comprometer la seguridad del usuario, almacenando códigos temporales de manera aislada y con fecha de expiración.**
* **Gracias a los campos created\_at y updated\_at, es posible realizar trazabilidad de cambios, útil para fines administrativos o auditoría.**
* **Los nombres se guardan en mayúsculas para evitar inconsistencias al comparar cadenas en procesos como login o exportación de datos.**

**🛠️ Uso en el sistema**

**La tabla usuarios cumple una función central en el funcionamiento del sistema de gestión del taller, actuando como núcleo para la autenticación, la autorización y el registro de actividad del personal. A continuación se detalla su uso en los distintos componentes de la aplicación:**

**🔐 1. Inicio de sesión (Login)**

* **El sistema utiliza el campo nombre como identificador principal para el inicio de sesión.**
* **La contraseña se compara mediante hashing con el campo password almacenado en la base de datos.**
* **La verificación del rol (rol\_id) permite mostrar solo los módulos autorizados en el Dashboard, según sea Administrador, Mecánico o Administrativo.**

**🔄 2. Recuperación de contraseña**

* **Al solicitar la recuperación, se genera un codigo\_recuperacion aleatorio y se almacena junto con expiracion\_codigo.**
* **Estos campos permiten verificar que el código es válido y no ha expirado antes de permitir el cambio de contraseña.**
* **El campo email se utiliza para enviar el código de verificación.**

**📊 3. Control de acceso y permisos**

* **A través del campo rol\_id (relacionado con roles), se controla el acceso a funcionalidades específicas.**
  + **Ej: solo el Administrador puede crear o eliminar usuarios.**
* **Esta lógica se aplica tanto en la interfaz como en las consultas del backend.**

**🕒 4. Módulo de fichajes**

* **Cada entrada o salida registrada en la tabla fichajes referencia al usuario.id.**
* **Permite llevar un historial individual de fichajes por empleado, útil para control horario.**

**🧑‍🔧 5. Registro y gestión de usuarios**

* **El módulo de administración permite crear nuevos usuarios, asignar roles y actualizar sus datos.**
* **Al registrar un nuevo usuario se generan automáticamente los campos requeridos, incluyendo el created\_at.**

**🧾 6. Generación de informes**

* **En módulos como el de historial de fichajes o intervenciones, los nombres de usuario se utilizan para mostrar la trazabilidad del personal involucrado.**
* **Permite emitir documentos PDF con nombre y apellidos del usuario, ya sea para auditoría, impresión o presentación.**

**🔒 7. Seguridad**

* **No se almacena la contraseña en texto plano: se usa un algoritmo de hashing (bcrypt) para asegurar su integridad.**
* **Los intentos de acceso al sistema se pueden registrar en futuras extensiones vinculando esta tabla con logs de actividad.**

**🧾 Tabla: roles**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla roles actúa como tabla de referencia para los distintos perfiles de acceso que pueden ser asignados a los usuarios del sistema. Es una tabla auxiliar que permite normalizar la asignación de permisos y mantener la escalabilidad del sistema de autenticación y autorización. Su diseño sigue principios de **normalización** y establece una relación directa con la tabla usuarios mediante clave foránea.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Permite nulos | Descripción técnica |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del rol. Se autoincrementa. |
| nombre | varchar | ❌ No | Nombre del rol. Valor textual único (ej. 'Administrador', 'Mecánico', etc.). |

**🔗 3. Relaciones y claves foráneas**

* Esta tabla **no contiene claves foráneas**, pero **sí es referenciada** por otras tablas:
  + usuarios.rol\_id → roles.id  
    Relación de uno a muchos que permite que varios usuarios compartan un mismo rol definido aquí.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY)**
   * id: garantiza la unicidad de cada rol y sirve como identificador referenciado por otras tablas.
2. **Restricciones de unicidad (UNIQUE)**
   * nombre: impide la creación de múltiples filas con el mismo nombre de rol. Esto evita ambigüedad en la asignación de permisos.
3. **Restricciones de obligatoriedad (NOT NULL)**
   * Ambos campos (id, nombre) son obligatorios para asegurar que cada rol esté correctamente identificado y descrito.

**🧩 5. Consideraciones adicionales de diseño**

* La tabla está pensada para contener un número **limitado y estable de registros**, por lo que es ideal para usar en menús desplegables, validaciones y controles de interfaz.
* Se puede extender con otros campos si se desea una gestión más avanzada de permisos, como por ejemplo: descripcion, nivel\_acceso, o activo.
* El uso de una tabla separada para roles permite mantener la integridad y coherencia de los datos cuando se reutilizan en múltiples puntos del sistema.
* La centralización de los roles facilita futuras modificaciones o auditorías sin necesidad de alterar la tabla principal de usuarios.

**🧾 Tabla: clientes**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla clientes representa la entidad que almacena los datos personales y de contacto de cada cliente registrado en el sistema. Esta información es esencial para llevar un control organizado y profesional de los clientes que utilizan los servicios del taller. La tabla está diseñada siguiendo las buenas prácticas de normalización, separando claramente la información de contacto, los datos de identificación y la metadata del registro.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Permite nulos | Descripción técnica |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del cliente. Se autoincrementa. |
| nombre | varchar | ❌ No | Nombre del cliente. Escrito en mayúsculas para homogeneidad en consultas. |
| primer\_apellido | varchar | ❌ No | Primer apellido del cliente. |
| segundo\_apellido | varchar | ✅ Sí | Segundo apellido del cliente, si lo tiene. |
| dni | varchar | ❌ No | Documento Nacional de Identidad. Se valida y es único en el sistema. |
| telefono | varchar | ❌ No | Número de teléfono de contacto. |
| email | varchar | ✅ Sí | Correo electrónico del cliente, si lo proporciona. |
| direccion | text | ✅ Sí | Dirección física completa del cliente. |
| codigo\_postal | varchar | ✅ Sí | Código postal correspondiente a la dirección. |
| localidad | varchar | ✅ Sí | Ciudad o municipio del cliente. |
| provincia | varchar | ✅ Sí | Provincia asociada al domicilio. |
| observaciones | text | ✅ Sí | Comentarios adicionales, notas internas o información complementaria. |
| created\_at | timestamp | ✅ Sí | Fecha de creación del registro. |
| updated\_at | timestamp | ✅ Sí | Fecha de última modificación de datos. |

**🔐 3. Restricciones**

1. **Clave primaria (PRIMARY KEY)**
   * id: Identificador único autoincremental que permite referenciar al cliente desde otras tablas como vehiculos o intervenciones.
2. **Restricciones de unicidad (UNIQUE)**
   * Aunque no es forzada a nivel de esquema, el dni es gestionado desde la lógica de aplicación como único para evitar duplicidades de clientes.
3. **Restricciones de obligatoriedad (NOT NULL)**
   * Los campos nombre, primer\_apellido, dni y telefono son obligatorios para asegurar la trazabilidad y el contacto con el cliente.

**🔗 4. Relaciones con otras tablas**

* **vehiculos.cliente\_id → clientes.id**  
  Relación uno-a-muchos: un cliente puede tener registrados múltiples vehículos.
* **intervenciones.cliente\_id → clientes.id** *(si existe esta relación directa)*  
  Permite asociar directamente intervenciones realizadas al cliente, además de su vehículo.

**🧩 5. Consideraciones adicionales de diseño**

* Esta tabla forma parte del núcleo funcional de la base de datos del sistema, y sirve como punto de entrada para otras operaciones relacionadas con vehículos, mantenimientos, presupuestos, ventas y recepcionamientos.
* El diseño contempla campos opcionales como email, direccion u observaciones, lo que permite adaptarse tanto a fichas básicas como a registros más detallados.
* Los campos created\_at y updated\_at aportan trazabilidad temporal, permitiendo auditar cuándo se registró o modificó por última vez un cliente.
* La validación de formato del dni se realiza a nivel de aplicación mediante expresiones regulares y control de letra.

**🧾 Tabla: intervenciones**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla intervenciones recoge toda la información relacionada con las reparaciones, mantenimientos u operaciones realizadas sobre los vehículos registrados en el sistema. Representa una entidad central dentro del módulo técnico del taller, ya que agrupa tanto los aspectos técnicos como administrativos de cada intervención: cliente, vehículo, tipo de intervención, estado, fechas clave, costes asociados y observaciones.

Está diseñada de forma relacional, conectando con otras tablas mediante claves foráneas, lo que permite mantener integridad referencial, evitar redundancias y facilitar la consulta cruzada de información para informes y seguimiento.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria, identificador único de la intervención. |
| cliente\_id | int4 | ❌ No | Clave foránea que referencia al cliente que solicita la intervención (clientes.id). |
| vehiculo\_id | int4 | ❌ No | Clave foránea que identifica el vehículo al que se le realiza la intervención (vehiculos.id). |
| tipo\_id | int4 | ❌ No | Clave foránea hacia la tabla tipos\_intervencion, que clasifica la naturaleza del trabajo. |
| estado\_id | int4 | ❌ No | Clave foránea que indica el estado actual de la intervención (estados\_intervencion.id). |
| descripcion | text | ❌ No | Descripción detallada de los trabajos a realizar. Puede incluir diagnóstico, tareas previstas o comentarios del cliente. |
| fecha\_inicio | date | ❌ No | Fecha en la que se inicia la intervención. |
| fecha\_finalizacion | date | ✅ Sí | Fecha en la que finaliza la intervención (puede estar vacía si aún está en curso). |
| descuento\_aplicado | int4 | ✅ Sí | Porcentaje de descuento aplicado (si lo hay), expresado como número entero. |
| coste\_presupuestado | numeric | ✅ Sí | Importe estimado al inicio de la intervención. |
| coste\_real | numeric | ✅ Sí | Importe real al finalizar los trabajos. |
| observaciones | text | ✅ Sí | Comentarios adicionales, incidencias, sugerencias o detalles relevantes. |
| created\_at | timestamp | ✅ Sí | Fecha de creación del registro para trazabilidad o informes históricos. |

**🔗 3. Relaciones y claves foráneas**

* cliente\_id → clientes.id: Relaciona cada intervención con su cliente correspondiente.
* vehiculo\_id → vehiculos.id: Enlaza la intervención con el vehículo intervenido.
* tipo\_id → tipos\_intervencion.id: Clasifica el tipo de intervención: revisión, reparación, cambio de aceite, etc.
* estado\_id → estados\_intervencion.id: Indica el estado actual: pendiente, en curso, finalizada, cancelada, etc.

Estas relaciones garantizan integridad referencial y permiten generar consultas complejas, como el historial completo de intervenciones por vehículo, cliente o estado.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY)**
   * id: Garantiza la unicidad de cada intervención.
2. **Restricciones de integridad referencial (FOREIGN KEY)**
   * Todas las claves foráneas (cliente\_id, vehiculo\_id, tipo\_id, estado\_id) aseguran que solo puedan registrarse intervenciones válidas asociadas a registros existentes en otras tablas.
3. **Restricciones de obligatoriedad (NOT NULL)**
   * Campos esenciales como cliente\_id, vehiculo\_id, tipo\_id, estado\_id, descripcion y fecha\_inicio no pueden estar vacíos, garantizando la completitud mínima del registro.
4. **Coherencia de datos opcionales**
   * Los campos fecha\_finalizacion, coste\_real y descuento\_aplicado pueden completarse al finalizar la intervención, lo que permite gestionar tanto intervenciones en curso como terminadas.

**🧩 5. Consideraciones de diseño**

* **Flexibilidad**: Los campos opcionales permiten reflejar tanto procesos iniciados como terminados sin romper la estructura.
* **Escalabilidad**: La tabla está preparada para adaptarse a nuevos estados o tipos de intervención, gracias a su diseño normalizado.
* **Trazabilidad**: El campo created\_at permite hacer un seguimiento histórico de la actividad del taller.
* **Gestión de costes**: Al separar el presupuesto y el coste real, se facilita el análisis de desviaciones económicas.

**🧾 Tabla: vehículos**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla vehiculos representa una de las entidades clave del sistema de gestión del taller, encargada de almacenar la información técnica y administrativa de todos los vehículos registrados por los clientes. Su diseño relacional permite mantener un vínculo directo entre cada vehículo y su propietario, así como clasificarlo por tipo y tipo de combustible, de forma normalizada.

Gracias a su estructura extensible y modular, permite soportar una gran variedad de casos (turismos, motos, vehículos industriales, eléctricos, etc.), proporcionando así una base sólida para el módulo de intervenciones y mantenimientos.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del vehículo en el sistema. |
| cliente\_id | int4 | ❌ No | Clave foránea que vincula el vehículo con un cliente existente (clientes.id). |
| marca | varchar | ✅ Sí | Marca comercial del vehículo (ej. Peugeot, Yamaha, Mercedes). |
| modelo | varchar | ✅ Sí | Modelo específico del vehículo. |
| matricula | varchar | ❌ No | Matrícula del vehículo. Campo único que evita duplicidades en el parque registrado. |
| tipo\_vehiculo | int4 | ✅ Sí | Clave foránea hacia la tabla tipos\_vehiculo, que clasifica el vehículo por su naturaleza (turismo, moto, industrial, etc.). |
| anyo | int4 | ✅ Sí | Año de fabricación o matriculación. Útil para historial y antigüedad. |
| observaciones | text | ✅ Sí | Comentarios libres sobre el vehículo: modificaciones, incidencias o peculiaridades. |
| created\_at | timestamp | ✅ Sí | Fecha de alta del vehículo en el sistema. |
| color | varchar | ✅ Sí | Color principal del vehículo. |
| combustible | varchar | ✅ Sí | Tipo de combustible (texto libre, para compatibilidad hacia atrás). |
| numero\_bastidor | varchar | ❌ No | Número de bastidor (VIN). Identificador único a nivel de fabricante. Se fuerza como único. |
| combustible\_id | int4 | ✅ Sí | Clave foránea hacia una tabla tipos\_combustible, si se opta por normalizar en el futuro. |
| updated\_at | timestamp | ✅ Sí | Fecha de la última modificación del registro. |

**🔗 3. Relaciones y claves foráneas**

* cliente\_id → clientes.id: Establece la relación de pertenencia entre vehículo y cliente.
* tipo\_vehiculo → tipos\_vehiculo.id: Clasifica el vehículo dentro de un conjunto predefinido de tipos (ej. turismo, furgón, agrícola...).
* combustible\_id → (opcional) tipos\_combustible.id: Relación pensada para mejorar la gestión de combustibles si se opta por usar una tabla externa normalizada.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY)**
   * id: Identificador único y obligatorio de cada vehículo registrado.
2. **Clave única (UNIQUE)**
   * matricula y numero\_bastidor están definidos como únicos, ya que no pueden repetirse legalmente en ningún contexto. Esto asegura la integridad del parque registrado y evita errores humanos.
3. **Restricciones de integridad referencial (FOREIGN KEYS)**
   * cliente\_id: Requiere que el cliente exista en la tabla clientes.
   * tipo\_vehiculo: (opcional) Debe coincidir con un valor en tipos\_vehiculo.
   * combustible\_id: (opcional) Pensado para futuras ampliaciones mediante relación con tabla combustibles.
4. **Restricciones NOT NULL**
   * Los campos cliente\_id, matricula y numero\_bastidor son obligatorios. Esto garantiza la trazabilidad y la unicidad de cada vehículo en el sistema.

**🧩 5. Consideraciones de diseño**

* **Normalización y escalabilidad**: El uso de claves foráneas permite añadir nuevos tipos de vehículo y combustibles sin alterar la estructura base.
* **Control de duplicados**: Las restricciones únicas sobre matrícula y bastidor aseguran que no se registren vehículos duplicados.
* **Flexibilidad de captura**: La existencia de campos opcionales como color, observaciones o combustible textual permite registrar incluso vehículos con datos incompletos inicialmente.
* **Preparación para auditorías**: Campos como created\_at y updated\_at permiten llevar un historial temporal del parque de vehículos gestionado.

**🧾 Tabla: combustibles**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla combustibles actúa como una entidad de referencia (tabla de catálogo) destinada a normalizar los distintos tipos de combustible que pueden utilizar los vehículos registrados en el sistema. Su objetivo es evitar inconsistencias semánticas o errores de entrada libre (por ejemplo: "Gasolina", "gasolina", "GASOLINA"), garantizando así la integridad y estandarización de los datos relativos al tipo de alimentación de los vehículos.

Es utilizada principalmente como clave foránea en la tabla vehiculos, aportando estructura y flexibilidad al sistema de gestión de taller, permitiendo su extensión a nuevas fuentes de energía como eléctricos, híbridos o hidrógeno.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del tipo de combustible. |
| nombre | varchar | ❌ No | Nombre del combustible. Debe ser único y estar normalizado (por ejemplo: "Gasolina", "Diesel", "Eléctrico", "GLP"). |

**🔗 3. Relaciones y claves foráneas**

* **Relación con vehiculos**:
  + El campo combustible\_id en la tabla vehiculos referencia a combustibles.id. Esta relación permite que cada vehículo esté clasificado según un tipo de combustible definido, mejorando la gestión y los filtros dentro del sistema.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY)**
   * id: Asegura que cada tipo de combustible tenga un identificador único, esencial para su uso como clave foránea.
2. **Restricción de unicidad (UNIQUE)**
   * nombre: Este campo debe ser único para evitar duplicidades semánticas y garantizar una gestión coherente de los datos.
3. **Restricción NOT NULL**
   * Ambos campos (id, nombre) son obligatorios, lo que impide la creación de registros incompletos o erróneos.

**🧩 5. Consideraciones de diseño**

* **Escalabilidad**: Permite añadir fácilmente nuevos tipos de combustible sin modificar la estructura de otras tablas.
* **Integridad semántica**: El uso de nombres únicos y normalizados evita errores de escritura o ambigüedad durante la entrada de datos.
* **Desempeño**: Al usar una relación relacional con vehiculos, se optimiza el almacenamiento y se mejora el rendimiento en filtros o informes por tipo de combustible.

**🧾 Tabla: tipos\_vehiculo**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla tipos\_vehiculo es una tabla de referencia destinada a clasificar y normalizar los distintos tipos de vehículos que pueden ser gestionados dentro del sistema. Su diseño permite distinguir entre múltiples categorías de vehículos (por ejemplo, “Terrestres”, “Acuáticos”, “Industriales”) y detallar cada tipo dentro de dichas categorías (como “Turismo”, “Motocicleta”, “Camión”, “Barco”).

Esta estructura facilita la segmentación, filtrado y posterior análisis de los vehículos registrados, permitiendo una gestión más precisa y flexible según la naturaleza del parque automotor.

**📐 2. Estructura de la tabla**

| **Campo** | **Tipo de dato** | **Nulos** | **Descripción** |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del tipo de vehículo. |
| categoria | varchar | ❌ No | Agrupación general del tipo de vehículo. Ej.: “Terrestre”, “Acuático”, “Especial”. |
| nombre | varchar | ❌ No | Nombre específico del tipo de vehículo. Ej.: “Furgoneta”, “SUV”, “Camión Grúa”. |

**3. Relaciones y claves foráneas**

* **Relación con vehiculos**:
  + El campo tipo\_vehiculo en la tabla vehiculos referencia a tipos\_vehiculo.id, estableciendo una correspondencia entre cada vehículo y su tipo declarado. Esto garantiza la validez del tipo asignado y facilita su uso en filtros y reportes.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY)**
   * id: Identificador único por cada tipo de vehículo.
2. **Restricción de unicidad (UNIQUE)**
   * Comúnmente aplicada en combinación de categoria + nombre, para evitar duplicidades de nomenclatura. Si no se aplica de forma explícita, se recomienda definirla para evitar errores semánticos (por ejemplo, “Terrestre - Turismo” duplicado).
3. **Restricción NOT NULL**
   * Todos los campos son obligatorios, garantizando que ningún registro quede incompleto ni ambiguo.

**🧩 5. Consideraciones de diseño**

* **Extensibilidad**: Permite incorporar fácilmente nuevos tipos y categorías a medida que evolucionan los servicios del taller (vehículos eléctricos, maquinaria pesada, embarcaciones, etc.).
* **Estandarización**: Al separar tipo y categoría, se facilita la normalización y mejora la experiencia de búsqueda y visualización.
* **Optimización**: Usar esta tabla como referencia evita redundancias en la tabla vehiculos y mejora la integridad de los datos.

**🧾 Tabla: recepcionamientos**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla recepcionamientos almacena todos los datos relacionados con la entrada de vehículos al taller para su inspección, diagnóstico y posterior reparación. Representa uno de los elementos más complejos del sistema debido a la gran cantidad de información que se puede registrar durante el proceso de recepción. Esta tabla permite recoger tanto aspectos técnicos del estado del vehículo como preferencias del cliente y condiciones del servicio.

Está diseñada para capturar el contexto completo de cada ingreso al taller, desde si el vehículo llegó con grúa hasta si se desea un presupuesto previo o si tiene seguro. Toda esta información queda vinculada a otras entidades clave como cliente, vehículo, usuario que atiende, urgencia, estado actual y motivo de entrada.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del registro de recepción. |
| cliente\_id | int4 | ❌ No | Clave foránea que enlaza con la tabla clientes. |
| vehiculo\_id | int4 | ❌ No | Clave foránea que enlaza con vehiculos. Identifica el vehículo que se recibe. |
| usuario\_id | int4 | ❌ No | Clave foránea que indica qué usuario registró la recepción. |
| estado\_id | int4 | ❌ No | Clave foránea a la tabla de estados del proceso de recepción. |
| fecha | timestamp | ✅ Sí | Fecha y hora en la que se realizó la recepción. |
| arranca | bool | ✅ Sí | Indica si el vehículo arranca por sí solo. |
| viene\_con\_grua | bool | ✅ Sí | Señala si el vehículo ha llegado remolcado. |
| tiene\_seguro | bool | ✅ Sí | Indica si el vehículo cuenta con un seguro en vigor. |
| compania\_seguro | varchar | ✅ Sí | Nombre de la aseguradora, si aplica. |
| acepta\_reparacion\_hasta | numeric | ✅ Sí | Importe máximo autorizado por el cliente sin necesidad de aprobación previa. |
| valor\_estimado | numeric | ✅ Sí | Estimación inicial del coste de la reparación. |
| itv\_en\_vigor | bool | ✅ Sí | Señala si la ITV está vigente. |
| ultima\_revision | date | ✅ Sí | Fecha de la última revisión realizada. |
| desea\_presupuesto\_aprobado | bool | ✅ Sí | Indica si el cliente desea recibir un presupuesto antes de autorizar reparaciones. |
| lista\_averias\_cliente | text | ✅ Sí | Detalle textual de los problemas descritos por el cliente. |
| estado\_exterior | text | ✅ Sí | Observaciones sobre el estado visual exterior del vehículo. |
| estado\_interior | text | ✅ Sí | Observaciones sobre el estado del interior. |
| observaciones\_generales | text | ✅ Sí | Notas adicionales generales de la recepción. |
| ruta\_documento | text | ❌ No | Ruta del PDF generado para la hoja de recepción. Se requiere para trazabilidad. |
| entregar\_impreso | bool | ✅ Sí | Indica si se ha solicitado la entrega en papel. |
| enviar\_por\_correo | bool | ✅ Sí | Indica si se ha solicitado el envío por email. |
| num\_recepcionamiento | int4 | ✅ Sí | Número correlativo asignado a la recepción. Puede ser usado como identificador visible. |
| urgencia\_id | int4 | ❌ No | Clave foránea que indica el nivel de urgencia (alta, media, baja). |
| motivo\_id | int4 | ❌ No | Clave foránea al catálogo de motivos de entrada al taller. |

**🔗 3. Relaciones y claves foráneas**

La tabla recepcionamientos establece múltiples relaciones con otras entidades del sistema. Estas relaciones garantizan la integridad referencial de los datos y permiten consolidar un modelo relacional coherente y extensible. A continuación se describen las claves foráneas incluidas:

* **cliente\_id → clientes(id)**  
  Establece la relación entre la recepción y el cliente al que pertenece el vehículo. Cada recepción está asociada obligatoriamente a un cliente registrado en el sistema. Esta relación permite vincular los datos personales del cliente con el proceso de entrada del vehículo.
* **vehiculo\_id → vehiculos(id)**  
  Vincula el registro de recepción con un vehículo específico. Es indispensable para conocer qué unidad fue ingresada al taller y poder recuperar su historial de reparaciones o fichas técnicas. Además, al estar vehiculos relacionado con clientes, se refuerza la trazabilidad completa.
* **usuario\_id → usuarios(id)**  
  Indica qué miembro del personal realizó el ingreso del vehículo en el sistema. Esta relación permite registrar la autoría del procedimiento, lo cual es crucial para auditorías, asignación de responsabilidades y generación de informes por operario.
* **estado\_id → estados\_intervencion(id)**  
  Representa el estado actual del proceso de recepción o reparación asociado al vehículo. Puede tomar valores como “Pendiente”, “En reparación”, “Finalizado”, etc. Su uso permite llevar un seguimiento del ciclo de vida de cada recepción.
* **urgencia\_id → urgencias(id)**  
  Define el nivel de prioridad con el que debe ser tratado el caso (por ejemplo: Alta, Media o Baja). Esta información puede ser utilizada tanto a nivel operativo (planificación) como analítico (estadísticas de atención).
* **motivo\_id → motivos(id)**  
  Especifica el motivo por el cual el vehículo ha sido traído al taller. Puede incluir valores como “Mantenimiento preventivo”, “Avería grave”, “Revisión periódica”, entre otros. Esta relación permite categorizar las entradas y realizar análisis posteriores por tipo de servicio.

Estas claves foráneas garantizan que cada recepción esté correctamente vinculada a datos válidos en las demás tablas del sistema, permitiendo una navegación fluida entre entidades relacionadas y evitando registros huérfanos o inconsistentes.

**🔐 4. Restricciones**

* **Clave primaria**: id
* **NOT NULL**: en los campos clave como cliente\_id, vehiculo\_id, usuario\_id, estado\_id, urgencia\_id, motivo\_id, y ruta\_documento, para garantizar un registro completo.
* **FOREIGN KEYS**: protegen la consistencia de los datos y evitan asignar referencias a entidades inexistentes.
* **Tipos booleanos**: numerosos campos usan booleanos (bool) para respuestas binarias que ayudan a simplificar validaciones y visualización.

**🧩 5. Consideraciones de diseño**

* **Alta granularidad**: esta tabla recoge una gran cantidad de atributos opcionales que ofrecen una visión completa del estado del vehículo y de las preferencias del cliente.
* **Documentación generada**: gracias al campo ruta\_documento, se almacena la referencia al documento generado, permitiendo su recuperación o reimpresión.
* **Escalabilidad**: admite la inclusión futura de nuevos campos o ajustes sin necesidad de reestructurar su lógica central.
* **Trazabilidad**: al incluir referencias cruzadas con todos los elementos implicados, permite auditorías y análisis detallados.

**🧾 Tabla: presupuestos**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla presupuestos está diseñada para almacenar las estimaciones económicas asociadas a cada recepción de vehículo en el taller. Representa un documento intermedio entre la recepción y la intervención, permitiendo al cliente conocer el coste aproximado de los trabajos sugeridos antes de su ejecución.

Su estructura contempla tanto los datos básicos del presupuesto como las opciones de autorización, comunicación con el cliente y almacenamiento del archivo PDF generado. Está íntimamente ligada a la tabla recepcionamientos, de la cual hereda su contexto técnico y logístico.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del presupuesto. Autoincremental. |
| recepcion\_id | int4 | ❌ No | Clave foránea que enlaza con recepcionamientos(id). Define el origen del presupuesto. |
| fecha\_creacion | timestamp | ❌ No | Fecha en la que se generó el presupuesto. Útil para trazabilidad. |
| total\_estimado | numeric | ❌ No | Valor monetario estimado para la intervención propuesta. |
| autorizado\_directo | bool | ❌ No | Indica si el cliente autorizó el presupuesto en el momento de la recepción. |
| cliente\_avisado | bool | ❌ No | Determina si el cliente ha sido informado del presupuesto (por llamada, email, etc.). |
| respuesta\_cliente | varchar | ✅ Sí | Observación corta sobre la respuesta del cliente (aceptado, rechazado, solicita cambios...). |
| observaciones | text | ✅ Sí | Campo para comentarios adicionales por parte del personal del taller. |
| ruta\_pdf | text | ✅ Sí | Ruta del archivo PDF del presupuesto generado y almacenado en el sistema de ficheros. |

**🔗 3. Relaciones y claves foráneas**

* **recepcion\_id → recepcionamientos(id)**  
  Esta clave foránea establece una relación uno a uno con la tabla recepcionamientos. Cada presupuesto se encuentra ligado a un proceso de recepción determinado, y no puede existir un presupuesto sin su correspondiente recepción asociada.

Esta relación garantiza coherencia en el flujo documental del sistema:  
**Recepción → Presupuesto → Intervención.**

Además, permite heredar metadatos relevantes del proceso de entrada (cliente, vehículo, estado, etc.) para su uso en informes o filtros.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY):**
   * id actúa como identificador único e irrepetible para cada presupuesto.
2. **Clave foránea (FOREIGN KEY):**
   * recepcion\_id debe existir previamente en la tabla recepcionamientos. No se admiten presupuestos sin una recepción válida asociada.
3. **Restricciones de obligatoriedad (NOT NULL):**
   * recepcion\_id, fecha\_creacion, total\_estimado, autorizado\_directo y cliente\_avisado son campos obligatorios. Esta medida garantiza que cada presupuesto sea completo y funcional en los procesos administrativos.

**🧩 Consideraciones adicionales**

* El campo ruta\_pdf permite mantener un vínculo físico con el documento generado, favoreciendo su reimpresión o reenvío posterior.
* Los campos booleanos (autorizado\_directo y cliente\_avisado) optimizan el control del estado de aceptación del presupuesto por parte del cliente.
* La flexibilidad del campo respuesta\_cliente permite registrar respuestas comunes o específicas sin necesidad de una tabla externa, ya que no se trata de un dato crítico.
* Esta tabla sirve de punto intermedio ideal para decidir si se continúa con la intervención, se solicita una revisión o se archiva el caso.

**🧾 Tabla: tareas\_presupuesto**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla tareas\_presupuesto representa una extensión detallada de los presupuestos, desglosando las acciones específicas planificadas dentro de un presupuesto técnico. Cada fila de esta tabla corresponde a una tarea concreta, con su descripción, estimación temporal y cálculo de coste asociado.

Este diseño permite un mayor grado de transparencia hacia el cliente, facilitando tanto la gestión interna como la comprensión del presupuesto por parte del usuario final. Se encuentra fuertemente ligada a la tabla presupuestos mediante una relación uno a muchos.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único de la tarea. Se autoincrementa. |
| presupuesto\_id | int4 | ❌ No | Clave foránea que enlaza con presupuestos(id). Determina a qué presupuesto pertenece la tarea. |
| descripcion | text | ❌ No | Texto explicativo de la tarea presupuestada (ej. "Sustitución de frenos delanteros"). |
| horas | numeric | ❌ No | Número estimado de horas necesarias para completar la tarea. |
| precio\_hora | numeric | ❌ No | Valor monetario por hora aplicada a la tarea. |
| total | numeric | ❌ No | Coste total estimado de la tarea (horas \* precio\_hora). Se calcula y almacena para optimizar lecturas y cálculos de resumen. |

**🔗 3. Relaciones y claves foráneas**

* **presupuesto\_id → presupuestos(id)**  
  Cada tarea se vincula obligatoriamente a un presupuesto. Esta relación asegura que el desglose técnico y económico de un presupuesto esté correctamente organizado y trazado.

Esta estructura jerárquica permite una navegación clara desde el documento general (presupuestos) hasta cada actividad específica (tareas\_presupuesto), lo cual resulta especialmente útil en interfaces gráficas y generación de informes PDF detallados.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY):**
   * id: garantiza la unicidad de cada tarea registrada en el sistema.
2. **Clave foránea (FOREIGN KEY):**
   * presupuesto\_id: debe existir en la tabla presupuestos. No se puede registrar una tarea sin estar asociada a un presupuesto.
3. **Restricciones de obligatoriedad (NOT NULL):**
   * Todos los campos (presupuesto\_id, descripcion, horas, precio\_hora, total) son obligatorios. Esto asegura que cada tarea esté completamente definida para su inclusión en presupuestos formales.

**🧩 Consideraciones adicionales**

* La inclusión del campo total como valor almacenado (y no solo calculado) está pensada para mejorar el rendimiento en listados y reportes. Aun así, el backend puede recalcularlo al vuelo en procesos críticos.
* Esta tabla puede ser utilizada como base para futuras integraciones con sistemas de facturación, órdenes de trabajo o partes de intervención.
* El campo descripcion admite texto libre, lo que aporta flexibilidad al personal técnico para registrar tareas de distinta naturaleza sin requerir tablas adicionales para categorización.

**🧾 Tabla: estados\_intervencion**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla estados\_intervencion tiene como objetivo normalizar y centralizar los posibles **estados** o **fases** que puede atravesar una intervención técnica (reparación o mantenimiento) realizada sobre un vehículo. Su uso permite garantizar consistencia semántica y evitar errores de entrada de datos en registros vinculados con la tabla intervenciones.

Este modelo evita el uso de valores textuales duplicados o ambiguos en la tabla principal de intervenciones, sustituyéndolos por referencias claras e indexadas.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del estado. Se autoincrementa. |
| nombre | varchar | ❌ No | Nombre descriptivo del estado (ej: "Pendiente", "En curso", "Finalizada", "Facturada", etc.). Debe ser único. |

**🔗 3. Relaciones y claves foráneas**

* **Relación con intervenciones:**
  + El campo estado\_id de la tabla intervenciones actúa como **clave foránea** que referencia estados\_intervencion(id).
  + Esta relación permite que cada intervención tenga un estado válido asignado en todo momento, lo cual facilita el filtrado, la visualización y la trazabilidad de trabajos en el taller.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY):**
   * id: clave única que garantiza la identificación inequívoca del estado.
2. **Restricciones de obligatoriedad (NOT NULL):**
   * Ambos campos (id y nombre) son obligatorios para asegurar la consistencia del diccionario de estados.

**🧩 Consideraciones adicionales**

* Este enfoque permite ampliar fácilmente el sistema con nuevos estados sin necesidad de alterar la lógica de la tabla intervenciones.
* Puede ser utilizado también como base para **etiquetado visual** (ej. colores o iconos en la interfaz) o para **filtros avanzados** de estado en el panel de administración.
* En informes impresos o digitales, el nombre del estado permite clasificar las intervenciones según su ciclo de vida.

**🧾 Tabla: tipos\_intervencion**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla tipos\_intervencion representa un catálogo de categorías de intervención o trabajos que pueden realizarse sobre un vehículo en el taller mecánico. Su propósito es **clasificar de forma estandarizada** las distintas acciones que se llevan a cabo, permitiendo una mejor trazabilidad, organización de recursos y análisis posterior (por ejemplo, para informes de frecuencia o coste por tipo de intervención).

Esta normalización es clave en sistemas de gestión que manejan gran variedad de tareas, como mantenimientos preventivos, reparaciones específicas, revisiones técnicas, instalaciones, diagnósticos, entre otros.

**📐 2. Estructura de la tabla**

| **Campo** | **Tipo de dato** | **Nulos** | **Descripción** |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del tipo de intervención. Se autoincrementa. |
| nombre | varchar | ❌ No | Nombre o descripción corta del tipo de intervención (ej. "Cambio de aceite", "Diagnóstico electrónico", "Revisión pre-ITV"). |

**🔗 3. Relaciones y claves foráneas**

* **Relación con intervenciones:**
  + El campo tipo\_id en la tabla intervenciones actúa como **clave foránea** referenciando tipos\_intervencion(id).
  + Esta relación permite que cada intervención tenga asociada una tipología definida, asegurando consistencia en los registros y habilitando funcionalidades como filtros, estadísticas, análisis por tipo, etc.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY):**
   * id: garantiza la unicidad de cada tipo de intervención, indispensable para relaciones externas.
2. **Restricción de unicidad (recomendada):**
   * El campo nombre debería tener una restricción UNIQUE, ya que cada tipo debe identificarse de forma inequívoca.
3. **Restricciones de obligatoriedad (NOT NULL):**
   * Ambos campos están definidos como obligatorios. Esto impide dejar registros incompletos en este catálogo.

**🧩 Consideraciones adicionales**

* Esta tabla permite una gestión dinámica del sistema, ya que los tipos de intervención pueden ser modificados, añadidos o eliminados sin necesidad de cambiar la estructura de la tabla intervenciones.
* Es útil para generación de presupuestos y segmentación de costes por tipo de servicio.
* El uso de identificadores numéricos y nombres descriptivos simplifica la implementación de formularios desplegables y filtros en la interfaz gráfica.

**🧾 Tabla: urgencias**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla urgencias actúa como una entidad de soporte que permite **clasificar y etiquetar el nivel de urgencia** asociado a una recepción de vehículo en el taller. Su principal función es estandarizar la información relacionada con la **prioridad del servicio**, facilitando tanto la organización interna como la planificación de recursos técnicos y humanos.

La centralización de esta información permite mantener un vocabulario controlado, evitando ambigüedades como "muy urgente", "urgente", "¡URGENTE!", etc., que dificultarían los procesos automatizados o el filtrado en listados e informes.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del nivel de urgencia. Autoincremental. |
| descripcion | varchar | ❌ No | Texto descriptivo que indica el nivel de urgencia (por ejemplo: “Baja”, “Normal”, “Alta”, “Crítica”). |

**🔗 3. Relaciones y claves foráneas**

* **Relación con recepcionamientos:**
  + El campo urgencia\_id en la tabla recepcionamientos actúa como **clave foránea** referenciando urgencias(id).
  + Esta relación permite que cada recepción esté claramente clasificada en cuanto a su grado de urgencia, lo cual puede condicionar tiempos de respuesta, prioridad en asignación de técnicos y orden de intervención.

**🔐 4. Restricciones**

1. **Clave primaria (PRIMARY KEY):**
   * id: garantiza la unicidad y permite la indexación eficiente de los niveles de urgencia.
2. **Restricciones de obligatoriedad (NOT NULL):**
   * Ambos campos están marcados como obligatorios para asegurar la consistencia del catálogo de urgencias.

**🧩 Consideraciones adicionales**

* Esta tabla facilita el uso de sistemas de semáforo o alertas visuales en la interfaz del sistema.
* Es compatible con futuras ampliaciones (por ejemplo, la adición de nuevos niveles como "Programado" o "Emergencia absoluta").

**🧾 Tabla: coches\_venta**

Imagen que contiene entero, llenado, computadora, computer

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla coches\_venta forma parte del módulo de gestión de compraventa de vehículos dentro del sistema. Su objetivo es registrar de forma detallada cada uno de los automóviles disponibles o gestionados para la venta por parte del taller. Esto incluye tanto los datos técnicos del vehículo como la información comercial (precio de compra, venta, descuentos) y metadatos de trazabilidad.

La tabla está estrechamente vinculada a la entidad clientes, en caso de tratarse de vehículos que pertenecen o han sido adquiridos a clientes particulares. Además, incluye campos pensados para integrarse con la generación de contratos o documentación digital asociada.

**📐 2. Estructura de la tabla**

| Campo | Tipo de dato | Nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único del vehículo en venta. |
| marca | varchar | ❌ No | Marca del vehículo (ej. Ford, Peugeot, BMW). |
| modelo | varchar | ❌ No | Modelo específico del vehículo. |
| version | varchar | ✅ Sí | Versión o edición concreta del modelo (opcional). |
| anio | int4 | ❌ No | Año de fabricación del vehículo. |
| matricula | varchar | ❌ No | Matrícula del vehículo. Debe ser única. |
| bastidor | varchar | ❌ No | Número de bastidor (VIN), identificador único del chasis. |
| color | varchar | ✅ Sí | Color exterior del vehículo. |
| combustible | varchar | ✅ Sí | Tipo de combustible (Gasolina, Diésel, Híbrido, Eléctrico, etc.). |
| kilometros | int4 | ✅ Sí | Kilometraje registrado del vehículo. |
| potencia\_cv | int4 | ✅ Sí | Potencia del motor en caballos de vapor. |
| cambio | varchar | ✅ Sí | Tipo de transmisión (Manual, Automático, etc.). |
| puertas | int4 | ✅ Sí | Número de puertas. |
| plazas | int4 | ✅ Sí | Número de plazas homologadas. |
| precio\_compra | numeric | ❌ No | Precio de adquisición del vehículo. |
| precio\_venta | numeric | ❌ No | Precio final de venta al cliente. |
| descuento\_maximo | numeric | ✅ Sí | Porcentaje o importe de descuento máximo permitido sobre el PVP. |
| estado | varchar | ❌ No | Estado actual del vehículo (Disponible, Vendido, Reservado, etc.). |
| origen\_compra | varchar | ❌ No | Procedencia del vehículo (ej. particular, subasta, importación). |
| cliente\_id | int4 | ✅ Sí | Clave foránea que enlaza con clientes.id si el vehículo procede de un cliente concreto. |
| observaciones | text | ✅ Sí | Campo libre para anotar características adicionales, daños, etc. |
| created\_at | timestamp | ✅ Sí | Fecha de creación del registro. |
| updated\_at | timestamp | ✅ Sí | Fecha de la última modificación. |
| descuento\_max | numeric | ✅ Sí | Redundante con descuento\_maximo (revisar si es necesario mantener ambos). |
| dir\_contrato | varchar | ✅ Sí | Ruta al contrato PDF asociado con la operación de compraventa. |

1. 🔗 Relaciones y claves foráneas

La tabla coches\_venta está diseñada para registrar todos los vehículos que han sido puestos a la venta en el sistema, tanto si provienen de clientes como si son propiedad del propio taller o concesionario. En este contexto, su diseño relacional permite vincular de forma precisa cada vehículo con su propietario original, permitiendo llevar un control exhaustivo del origen y la trazabilidad de las operaciones de compraventa.

El campo cliente\_id funciona como clave foránea que referencia a la tabla clientes, concretamente al campo id. Esta relación establece una asociación directa entre cada coche en venta y su dueño legítimo, asegurando que únicamente puedan ser registrados como disponibles para la venta vehículos vinculados a un cliente real y previamente dado de alta en el sistema. Esto no solo mantiene la integridad referencial, sino que también posibilita funcionalidades importantes como la generación automática de contratos, el envío de notificaciones al cliente cuando se realice la venta, y el cálculo de comisiones o historial de compras.

Cabe destacar que, al tratarse de una relación de uno a muchos (un cliente puede tener múltiples coches a la venta), esta clave foránea es esencial para estructurar el sistema de inventario y seguimiento de vehículos de segunda mano. A través de esta relación, también es posible filtrar listados por cliente, consultar su stock individual, o incluso implementar procesos de postventa.

Esta única clave foránea en la tabla coches\_venta es suficiente para integrar este módulo de forma consistente dentro del modelo general del sistema, garantizando un alto grado de cohesión entre las entidades y permitiendo al mismo tiempo una gran flexibilidad funcional en la gestión del negocio de compraventa.

**🔐 4. Restricciones**

1. **Clave primaria:**
   * id: valor único autoincremental, sirve como identificador principal de la tabla.
2. **Restricciones de unicidad (recomendadas):**
   * matricula y bastidor deben ser únicos por su naturaleza identificadora. No deben repetirse nunca.
   * Podrían establecerse índices únicos (UNIQUE) para estos dos campos si no están aún definidos.
3. **Restricciones NOT NULL:**
   * Campos como marca, modelo, anio, matricula, bastidor, precio\_compra, precio\_venta, estado y origen\_compra son obligatorios para garantizar la integridad y utilidad del registro.
4. **Restricción referencial:**
   * cliente\_id debe corresponder a un ID existente en clientes si se indica. En caso contrario, debe permitir nulos.

**🧩 Consideraciones adicionales**

* La separación entre campos como precio\_compra, precio\_venta y descuento\_maximo permite generar automáticamente márgenes comerciales o calcular rentabilidad por vehículo.
* El campo dir\_contrato facilita la asociación de documentos electrónicos firmados u otros archivos relacionados con la venta.
* Es fundamental mantener la unicidad de bastidor y matricula, ya que son identificadores legales y técnicos del vehículo.

**🧾 Tabla: ventas**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**1. Descripción general**

La tabla ventas forma parte del módulo de gestión de compraventa del sistema. Está diseñada para almacenar la información asociada a la venta de vehículos registrados en el sistema, incluyendo datos sobre el comprador (cliente), el vehículo vendido, el precio final de la operación, así como la fecha y la ruta al documento contractual generado.

Este modelo permite llevar un control detallado de las transacciones realizadas, establecer relaciones directas entre clientes y vehículos, y garantizar trazabilidad documental gracias a los campos específicos para archivos PDF.

**2. 📐 Estructura de campos**

| Campo | Tipo de dato | Permite nulos | Descripción |
| --- | --- | --- | --- |
| id | int4 | ❌ No | Clave primaria. Identificador único de la venta. Se autoincrementa. |
| cliente\_id | int4 | ❌ No | Clave foránea que enlaza con clientes.id. Identifica al comprador. |
| vehiculo\_id | int4 | ❌ No | Clave foránea que enlaza con vehiculos.id. Indica el vehículo vendido. |
| precio\_final | numeric | ❌ No | Precio definitivo al que se cerró la venta. Puede incluir descuentos. |
| ruta\_pdf | text | ❌ No | Ruta al documento PDF generado como contrato de compraventa. |
| fecha\_venta | timestamp | ✅ Sí | Fecha y hora en que se realizó la venta. Útil para reportes y estadísticas. |
| dir\_contrato | varchar | ✅ Sí | Directorio o URL donde se encuentra alojado el contrato firmado. |

**3. 🔐 Restricciones**

**🔑 Clave primaria**

* id: garantiza la unicidad de cada transacción de venta.

**🔗 Claves foráneas**

* cliente\_id → clientes.id: asegura que cada venta esté asociada a un cliente válido. Establece la relación comprador → venta.
* vehiculo\_id → vehiculos.id: garantiza que el vehículo vendido esté previamente registrado. Evita errores de integridad.

**❗ Restricciones NOT NULL**

* Los campos cliente\_id, vehiculo\_id, precio\_final y ruta\_pdf son obligatorios para asegurar la completitud mínima de una venta válida.

**4. 🧷 Relaciones con otras tablas**

* **Clientes:**  
  Cada registro de ventas se vincula con un cliente a través de cliente\_id, permitiendo saber quién ha comprado qué vehículo. Esta relación es crucial para emitir informes de historial de clientes y ventas asociadas.
* **Vehículos:**  
  A través de vehiculo\_id, se relaciona con la tabla vehiculos, permitiendo identificar de forma precisa el coche que ha sido vendido. Esta relación puede usarse también para filtrar vehículos disponibles o vendidos.

**5. 🧩 Consideraciones adicionales**

* **Control documental:**  
  La inclusión del campo ruta\_pdf y dir\_contrato permite asociar un documento firmado o generado al proceso de venta, facilitando la gestión digital de contratos y reduciendo el uso de papel.
* **Historial de transacciones:**  
  Al permitir almacenar la fecha exacta de la venta (fecha\_venta), se puede generar un historial cronológico de compraventas por cliente, por vehículo o por periodo.
* **Integridad relacional:**  
  Esta tabla se comporta como un nexo entre clientes y vehiculos, generando un punto de control clave para garantizar que un vehículo vendido ya no pueda asignarse nuevamente como disponible para compraventa.

4.4 Definición de interfaces de usuario

Las interfaces se diseñaron con PySide6, imitando la estética de una aplicación de escritorio moderna, pero adaptada a resoluciones variadas. Se priorizó la claridad visual, el acceso rápido a funciones frecuentes, y la navegación intuitiva.

Características clave:

Ventanas personalizadas por tipo de usuario.

Botones grandes, iconos representativos y texto explicativo.

Tooltips en todos los controles importantes.

Pantallas centradas con estilo uniforme (colores institucionales, fuente Montserrat).

Ventanas modales para acciones críticas (fichajes, recuperación de contraseña).

Exportación y navegación guiada con mensajes de confirmación y errores amigables.

Cada vista fue construida manualmente, respetando un estilo coherente, sin plantillas automáticas ni generadores visuales.

4.5 Plan de pruebas y requisitos de aceptación

Para asegurar la calidad del sistema, se elaboró un plan de pruebas estructurado en tres niveles:

1. Pruebas unitarias

Validación de formularios (DNI, correos, contraseñas).

Comprobación de formatos de fecha y hora.

Ensayo de funciones auxiliares como generación de informes.

2. Pruebas funcionales

Simulación de inicio de sesión con distintos roles.

Registro de clientes y comprobación en base de datos.

Inserción y consulta de intervenciones.

Registro y exportación de fichajes.

3. Pruebas de integración

Pruebas completas con flujo de navegación entre vistas.

Verificación de que los cambios en la base de datos se reflejan en la interfaz.

Evaluación de rendimiento con múltiples registros.

Requisitos de aceptación:

El sistema debe ejecutarse sin errores en cualquier PC Windows moderno.

Debe poderse conectar correctamente a la base de datos en Supabase.

El usuario debe poder completar al menos un caso de uso completo por rol sin asistencia externa.

El ejecutable debe incluir todo lo necesario para ejecutarse sin instalación compleja.

4.6 Migración (en caso de ser necesaria) y requisitos de implantación

Dado que el sistema no sustituye a otro anterior, no se requiere migración de datos. Sin embargo, se han tenido en cuenta aspectos importantes para una futura implantación en entornos reales:

Requisitos mínimos de implantación:

Ordenador con Windows 10 o superior.

Conexión a internet para acceder a Supabase.

Usuario con cuenta registrada en el sistema.

Archivo ejecutable .exe proporcionado por el desarrollador.

Posibles migraciones futuras:

Migración de datos desde hojas de cálculo (CSV a PostgreSQL).

Conexión a otras bases de datos (MySQL, SQLite) si se decide cambiar el motor.

Integración con plataformas externas (correo corporativo, módulos contables, etc.).

Para facilitar cualquier migración futura, la base de datos se ha diseñado de forma normalizada y abierta, permitiendo exportación e importación sin dependencias complejas.

**5. Seguimiento, Evaluación y Control de Calidad**

Este apartado recoge los mecanismos implantados para garantizar que el proyecto cumpla los objetivos marcados, se ejecute con un alto nivel de calidad, y se mantenga alineado con los requisitos funcionales y técnicos establecidos desde su concepción.

**5.1. Definición de procedimientos de evaluación del proyecto**

Para asegurar un desarrollo controlado y eficiente del sistema, se han establecido los siguientes **procedimientos sistemáticos de evaluación**:

* **Revisión modular por hitos**: al finalizar cada módulo funcional (autenticación, gestión de clientes, reparaciones, ventas…), se ha realizado una revisión exhaustiva para comprobar:
  + Cumplimiento de requisitos funcionales.
  + Correcto enlace con otros módulos del sistema.
  + Estabilidad y ausencia de errores graves.
* **Pruebas unitarias y de integración**: cada componente implementado fue validado individualmente mediante tests localizados (\*\_test.py) y posteriormente verificado como parte de un flujo completo.
* **Evaluación por escenario de uso**: se han simulado acciones reales de usuarios, incluyendo flujos de trabajo comunes como:
  + Registro y modificación de clientes y vehículos.
  + Generación y seguimiento de presupuestos.
  + Fichajes de empleados.
  + Venta de vehículos.

**5.2. Establecimiento de indicadores de calidad y métricas de éxito**

Se han definido varios **indicadores clave de rendimiento (KPIs)** para medir la calidad del proyecto:

| Indicador | Descripción | Umbral objetivo |
| --- | --- | --- |
| 🧪 Tasa de errores en pruebas | Porcentaje de funcionalidades que presentan errores durante el test | < 5% |
| ⏱️ Tiempo medio de respuesta | Tiempo medio para operaciones básicas (login, búsqueda, creación) | < 500 ms |
| 📈 Cobertura funcional | Porcentaje de funcionalidades previstas que han sido implementadas | 100% |
| 📄 Integridad documental | Grado de cumplimiento del dossier y documentación técnica | 100% |
| 🔄 Mantenibilidad del código | Uso de arquitectura modular, comentarios y separación de lógica | Alta (modular y comentado) |

**5.3. Registro y análisis de incidencias surgidas durante la ejecución**

**Durante el desarrollo de la versión de escritorio del sistema con Python y PySide6, se realizó un seguimiento constante de los errores y situaciones inesperadas surgidas, con el objetivo de mantener la calidad del software y asegurar una experiencia fluida para el usuario final. Las principales incidencias se agruparon en categorías técnicas y se analizaron para encontrar patrones repetitivos o áreas críticas del sistema:**

* **🛑 Persistencia de la contraseña anterior tras cambio: Tras implementar el cambio de contraseña en el módulo de recuperación, se detectó que el sistema no reconocía la nueva contraseña hasta reiniciar por completo la aplicación. Esto reveló una gestión inadecuada de la conexión persistente de Sequelize, que fue solucionada cerrando manualmente la conexión después del cambio.**
* **🔁 Error al cerrar sesión: Al cerrar sesión desde el Dashboard, la aplicación se quedaba en blanco, mostrando un error de ruta (ERR\_FILE\_NOT\_FOUND). El problema estaba relacionado con el uso incorrecto de rutas relativas en el enrutador y fue corregido redirigiendo correctamente con navigate('/login') en el entorno hash.**
* **🧊 Congelación de la interfaz durante operaciones pesadas: En módulos como recepcionamiento y generación de documentos, se observó congelamiento parcial de la UI. La solución fue el diseño e implementación de hilos (QThread) para ejecutar tareas pesadas sin bloquear la interfaz principal, especialmente con hilo\_carga\_recepcionamiento.py.**
* **🔐 Error en validación de códigos de recuperación: Debido a diferencias horarias entre la hora local del sistema y la hora UTC de la base de datos, los códigos OTP aparecían como expirados antes de tiempo. Se solucionó estableciendo correctamente la hora de expiración con Sequelize.literal(...) en formato UTC ajustado.**

**Estas incidencias se documentaron de forma cronológica y fueron útiles para depurar no solo los errores inmediatos, sino también para refinar el diseño modular de la aplicación.**

**5.4. Plan de resolución de incidencias y optimización de procesos**

**El plan de resolución de errores seguido durante la construcción del sistema se basó en una metodología iterativa y flexible, con énfasis en el análisis técnico inmediato y la corrección con bajo impacto estructural. El enfoque consistió en:**

1. **Priorización según impacto: Se clasificaron las incidencias como críticas (login, pérdida de datos), moderadas (interfaz bloqueada) o menores (estética o mensajes).**
2. **Diagnóstico técnico directo: Uso de print y logging en puntos estratégicos, revisión de consultas SQL, y verificación de dependencias entre módulos como usuarios\_consultas, authController y ventana\_login.**
3. **Aplicación de correcciones específicas: Se evitó modificar componentes no relacionados para prevenir efectos colaterales. Las correcciones se validaron antes de integrarlas a la rama principal.**
4. **Refactorizaciones con propósito: Algunas incidencias derivaron en mejoras generales como el uso de controladores por ventana, reorganización del proyecto en carpetas modulares y separación clara entre lógica y vista.**
5. **Registro informal de aprendizaje: Se mantuvo un criterio de registro no formal de lecciones aprendidas, tanto para evitar reincidencias como para justificar mejoras posteriores.**

**Gracias a esta estrategia, el proyecto no solo corrigió los errores identificados, sino que salió fortalecido con una base más estable, mantenible y extensible.**

**5.5. Gestión de cambios en recursos y tareas según necesidades**

**A lo largo del proyecto fue necesario adaptar el diseño original y redistribuir recursos y prioridades en función de nuevas exigencias técnicas y decisiones de mejora funcional. Estos cambios, lejos de generar desorganización, permitieron consolidar una estructura más robusta y alineada con los objetivos del proyecto:**

* **🔄 Cambio de inicio de sesión por nombre en mayúsculas: Originalmente se accedía por correo electrónico, pero se optó por el uso del campo nombre en mayúsculas como método principal de autenticación, adaptando tanto las validaciones como la base de datos para evitar duplicidades.**
* **🧩 Normalización del sistema de roles: En lugar de almacenar el rol como texto plano, se creó una tabla separada roles con clave foránea rol\_id. Esta decisión facilitó el control de permisos y permitió filtrar funcionalidades desde el backend según el rol asignado.**
* **🧾 Separación de lógica de control y vista en PySide6: Para favorecer el mantenimiento y escalabilidad del sistema, se estableció un patrón MVC estricto, separando cada ventana en su archivo .py visual (ventana\_\*.py) y su controlador (\*\_controlador.py), junto con los estilos (.css).**
* **📄 Incorporación de informes dinámicos en HTML+CSS convertidos a PDF: Se optó por usar plantillas HTML y CSS externas para los presupuestos, contratos y recepcionamientos, permitiendo una generación de informes más estética, modular y reutilizable.**

**Estos cambios no solo se gestionaron a nivel técnico, sino también documental y organizativo, asegurando que el producto final fuese coherente con los objetivos del proyecto y con el entorno realista de un taller mecánico informatizado.**

**5.6. Evaluación de la satisfacción del usuario final y feedback recibido**

Durante las pruebas finales se ha implicado a potenciales usuarios reales (mecánicos, administrativos y gestores) para:

* Comprobar la usabilidad de las pantallas.
* Identificar puntos de confusión o mejora.
* Evaluar la utilidad práctica de funcionalidades como fichajes, historial de intervenciones o recuperación de cuentas.

📋 **Resultados**:

* Valoración media de usabilidad: 4.6 / 5.
* Observaciones aplicadas: mejora en navegación, mensajes de ayuda y reorganización de botones.

**5.7. Aseguramiento del cumplimiento de los requisitos y condiciones del proyecto**

Antes del cierre del desarrollo, se ha llevado a cabo una revisión completa de los siguientes aspectos:

✅ Cumplimiento de requisitos funcionales (login, fichajes, clientes, intervenciones, presupuestos, ventas, informes).  
✅ Conformidad con el diseño de base de datos relacional (normalización, claves foráneas, restricciones).  
✅ Correspondencia con el dossier de diseño y fases definidas en la memoria.  
✅ Presentación final del sistema con interfaz completa, funcional y portable.

La implementación final respeta el **alcance del proyecto original**, incluye mejoras documentadas durante el desarrollo, y cumple los requisitos del entorno académico y profesional en el que se enmarca.

**6. Conclusiones, Documentación y Presentación Final**

**6.1. Evaluación global del proyecto y comparación con los objetivos iniciales**

El proyecto ha cumplido satisfactoriamente los objetivos establecidos al inicio del desarrollo, alcanzando una solución integral de gestión para un taller mecánico, estructurada bajo principios de modularidad, escalabilidad y claridad en el diseño.

Se ha logrado construir una aplicación de escritorio robusta, intuitiva y funcional que abarca desde la gestión de usuarios hasta el tratamiento de presupuestos, fichajes, contratos, intervenciones y ventas, apoyándose en una base de datos relacional en PostgreSQL con una normalización exhaustiva. La aplicación ha sido diseñada bajo el patrón MVC, con una arquitectura clara que separa la lógica de negocio, la interfaz gráfica (desarrollada con PySide6) y las consultas a la base de datos.

Comparado con los objetivos iniciales, se puede afirmar que:

* ✅ La funcionalidad prevista ha sido completamente implementada.
* ✅ Se ha respetado la estructura modular y las convenciones de diseño definidas.
* ✅ Se ha introducido un sistema de informes automatizados y estilizados en PDF, mejorando la presentación de resultados.
* ✅ Se ha incluido un sistema de roles funcional que condiciona el acceso a funcionalidades según el perfil del usuario.

En resumen, el producto final supera las expectativas iniciales en cuanto a profundidad funcional y presentación visual.

**6.2. Dificultades encontradas y estrategias de resolución**

El desarrollo del proyecto no ha estado exento de obstáculos técnicos y estructurales. Algunas de las dificultades más relevantes han sido:

* 🧠 **Curva de aprendizaje de PySide6**: La transición desde otros frameworks visuales a PySide6 exigió una reestructuración de conceptos, especialmente en lo relativo a señales, slots y control de eventos. Se resolvió mediante documentación oficial y pruebas experimentales.
* ⚠️ **Gestión de conexiones persistentes en Sequelize**: Tras modificar contraseñas o realizar operaciones de recuperación de cuenta, el backend no reflejaba inmediatamente los cambios debido a conexiones abiertas. Se corrigió cerrando explícitamente el pool de conexiones después de las operaciones críticas.
* 🔐 **Errores en validación de código de recuperación**: Las diferencias horarias entre la hora local y UTC causaban la expiración prematura de los códigos OTP. Se solucionó generando los timestamp con conversión explícita a UTC, garantizando la consistencia.
* ⚙️ **Errores al cerrar sesión y redirección incorrecta**: Al usar HashRouter en React, algunas rutas provocaban errores al intentar acceder a archivos inexistentes. Esto fue corregido ajustando el sistema de navegación y utilizando rutas absolutas.
* 🔄 **Problemas con herramientas de empaquetado (Electron, APK)**: Aunque se desarrolló en formato desktop, se exploraron opciones para embebido móvil que generaron incompatibilidades (ej. nodejs-mobile-cordova). Se optó por priorizar la versión estable de escritorio y dejar las versiones móviles para una fase futura.

Todas las dificultades fueron documentadas, abordadas de forma estructurada y resueltas sin comprometer la estabilidad del sistema.

**6.3. Propuestas de mejora y evolución del proyecto**

**Aunque el sistema actual ofrece una solución completa y funcional para la gestión diaria del taller, existen diversas líneas de mejora y evolución que permitirían ampliar sus capacidades, incrementar la eficiencia operativa, y facilitar su escalabilidad a distintos entornos empresariales.**

**A continuación, se enumeran las principales propuestas de mejora contempladas para futuras versiones:**

**📑 1. Incorporación de un módulo de facturación oficial automatizada**

**Actualmente, los procesos de facturación se gestionan manualmente o a través de informes personalizados en PDF. Una evolución natural sería la integración de un módulo de facturación conforme a normativa fiscal, que contemple:**

* **Generación de facturas con numeración automática.**
* **Inclusión de IVA, descuentos, base imponible y totales por cliente.**
* **Exportación en formato FacturaE o XML para su remisión a gestorías o la Agencia Tributaria.**
* **Registro contable interno por ejercicio fiscal y cliente.**

**Esto facilitaría el cumplimiento de las obligaciones legales, eliminando errores humanos y mejorando la trazabilidad de los ingresos del taller.**

**🧰 2. Gestión de almacén y stock de recambios**

**Aunque las intervenciones están correctamente registradas, no existe aún un sistema de control sobre las piezas o materiales empleados. Se propone:**

* **Crear una tabla recambios con stock actual, proveedor, coste y código de referencia.**
* **Asociar piezas a intervenciones o presupuestos.**
* **Generar avisos automáticos de reposición al alcanzar mínimos definidos.**
* **Registrar entradas y salidas del almacén por usuario y fecha.**

**Este módulo permitiría al taller llevar un control detallado del inventario, detectar pérdidas o errores, y anticipar necesidades de reposición.**

**🌐 3. Sincronización entre múltiples dispositivos (móvil, tablet, PC)**

**Actualmente el sistema se ejecuta como aplicación de escritorio, pero con vistas a su portabilidad se prevé una futura integración de sincronización en la nube que permita:**

* **Uso simultáneo en distintos dispositivos con autenticación por usuario.**
* **Guardado en local y sincronización posterior (modo offline).**
* **Enlace con Supabase, Firebase o bases de datos PostgreSQL remotas con cifrado SSL.**
* **Integración futura con una API REST propia.**

**Esto abriría la puerta a un modelo más colaborativo y flexible, ideal para talleres con múltiples puestos de trabajo o con necesidades de acceso desde dispositivos móviles.**

**🔍 4. Panel de informes avanzados y estadísticas interactivas**

**Se propone la creación de un dashboard de análisis de actividad con métricas como:**

* **Total de intervenciones por mes o tipo.**
* **Rendimiento de cada mecánico o usuario.**
* **Ingresos estimados vs ingresos reales.**
* **Tiempo medio por tipo de intervención.**
* **Fichajes y control horario por trabajador.**

**Estas estadísticas podrían presentarse mediante gráficos interactivos y exportarse a PDF o CSV, facilitando la toma de decisiones y el análisis estratégico.**

**🔧 5. Mantenimiento predictivo básico**

**Con los datos ya almacenados (vehículo, tipo de intervención, fecha, kilometraje...), se podría aplicar un sistema de mantenimiento predictivo básico:**

* **Alertas programadas para revisiones periódicas.**
* **Propuesta automática de futuras intervenciones basadas en histórico y tipo de vehículo.**
* **Vinculación con fechas clave (última ITV, cambio de aceite, etc.).**

**Aunque no se trataría de una inteligencia artificial avanzada, sí permitiría ofrecer valor añadido al cliente y mejorar la planificación del taller.**

**🔒 6. Registro de actividad del sistema (logging de acciones)**

**Para mejorar la trazabilidad y la seguridad se plantea incluir un sistema de log de actividad por usuario, donde se registre:**

* **Inicio y cierre de sesión.**
* **Creación, modificación y eliminación de registros.**
* **Descarga e impresión de documentos.**

**Esto permitiría auditar el uso del sistema, detectar errores humanos y mejorar la gestión de permisos internos.**

**📤 7. Integración con servicios externos (correo, WhatsApp, agenda)**

**Ya existe un sistema de envío de correos para recuperación de cuenta y presupuestos, pero podría ampliarse para:**

* **Enviar notificaciones de citas o reparaciones por email o mensajería instantánea (WhatsApp Business API).**
* **Añadir eventos automáticamente a calendarios externos (Google Calendar, Outlook).**
* **Confirmar asistencia de clientes mediante enlaces de respuesta rápida.**

**Este tipo de integraciones mejoraría la comunicación con el cliente y aportaría una experiencia más moderna y profesional.**

**🔄 8. Sistema de backup y restauración automatizado**

**Actualmente, las copias de seguridad se realizan manualmente o desde el entorno de Supabase. Se propone implementar:**

* **Tareas programadas de exportación de la base de datos y documentos.**
* **Herramienta visual dentro del sistema para restaurar versiones anteriores.**
* **Encriptación de los backups para mayor seguridad.**

**Este sistema protegería los datos frente a pérdidas accidentales o errores críticos.**

**En conjunto, estas mejoras no solo incrementan la potencia del sistema, sino que alinean su evolución con prácticas reales del sector y con necesidades detectadas durante el desarrollo. La propuesta mantiene la coherencia con la arquitectura ya diseñada y es completamente viable para su implementación futura sin reestructurar el núcleo del proyecto.**

**🧾 7. Manual de Usuario**

**🔐 7.1. Acceso al sistema**

**Esta es la pantalla inicial del programa ReyBoxes, donde los usuarios deben identificarse para acceder al sistema. El diseño ha sido pensado para ofrecer una experiencia intuitiva y agradable.**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**🧭 Descripción general**

**La ventana de acceso está compuesta por los siguientes elementos:**

1. **Logotipo central: Representa la identidad visual del taller ReyBoxes.**
2. **Título "Iniciar Sesión": Con un diseño llamativo, destacando la palabra "Sesión" en rojo.**
3. **Campo "Inserte su nombre":**
   * **Obligatorio.**
   * **Debe introducirse el nombre de usuario en mayúsculas (aunque el sistema lo transforma automáticamente).**
4. **Campo "Contraseña":**
   * **Obligatorio.**
   * **Se introduce la contraseña correspondiente al usuario.**
5. **Botón rojo "Entrar":**
   * **Permite acceder al sistema si las credenciales son correctas.**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

* + **Si los datos son erróneos, se muestra un mensaje indicando el fallo.**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

1. **Botón rojo "Salir":**
   * **Cierra completamente el programa.**
2. **Enlace "¿Olvidaste tu contraseña?":**
   * **Lleva al proceso de recuperación de cuenta, donde se puede solicitar un nuevo acceso por correo electrónico.**

**🛠️ Instrucciones paso a paso para iniciar sesión**

1. **Escriba su nombre de usuario en el primer campo (por ejemplo: CRESNIK).**
2. **Escriba su contraseña personal en el segundo campo.**
3. **Haga clic en el botón "Entrar" o pulse Enter.**
4. **Si los datos son correctos, se abrirá automáticamente el menú principal del sistema.**
5. **En caso de error, revise las mayúsculas o solicite una nueva contraseña desde el enlace inferior.**

**🔐 7.1.1. Recuperación de cuenta**

**Esta pantalla aparece al hacer clic en el enlace "¿Olvidaste tu contraseña?" en la ventana de inicio de sesión. Permite al usuario recuperar el acceso a su cuenta mediante el envío de un código de verificación a su correo electrónico registrado.**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.**🧭 Descripción general**

**Elementos visibles en esta pantalla:**

1. **Título "Recuperar cuenta":**
   * **Formato consistente con el resto del programa.**
2. **Texto de instrucciones:**
   * **Breve mensaje explicativo: *"Introduce el correo electrónico asociado a tu cuenta:"***
   * **Ayuda al usuario a entender el propósito de esta pantalla.**
3. **Campo de entrada de correo electrónico:**
   * **Debe introducirse un correo electrónico válido registrado en el sistema.**
   * **Tiene un icono de sobre a la izquierda.**
   * **Si el formato del correo no es válido, el botón "Enviar Código" se desactiva.**
4. **Botón rojo "Enviar Código":**
   * **Envía un correo con un código de verificación temporal.**
   * **Solo se activa si el correo es válido y está registrado.**
5. **Botón rojo "Volver":**
   * **Permite regresar directamente a la ventana de inicio de sesión.🛠️ Instrucciones paso a paso para recuperar el acceso**
6. **Introduzca el correo electrónico asociado a su cuenta.**
7. **Asegúrese de que el correo esté bien escrito y tenga el formato correcto (ejemplo: usuario@dominio.com).**
8. **Haga clic en "Enviar Código".**
   * **Si el correo está registrado, se enviará un código temporal al buzón indicado.**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

* + **Se abrirá automáticamente la pantalla de verificación.**

1. **En caso contrario, se mostrará un mensaje indicando que el correo no está registrado**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

1. **Puede pulsar "Volver" en cualquier momento para regresar a la pantalla de inicio de sesión.**

**🔐 7.1.2. Verificación de código**

**Después de solicitar el código en la pantalla anterior, se abre esta ventana para validar el acceso mediante el código temporal recibido por correo electrónico. Este paso es obligatorio antes de poder restablecer la contraseña.**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.**🧭 Descripción general**

**Esta pantalla contiene los siguientes elementos:**

1. **Título "Verificar Código":**
   * **Continúa con la estética del programa.**
2. **Instrucción de uso:**
   * **Se indica al usuario: *"Introduce el código que recibiste por correo"*.**
3. **Campo de entrada del código:**
   * **Campo central donde se introduce el código numérico recibido.**
   * **Solo se acepta el código correcto, que es de un solo uso y temporal.**
4. **Contador regresivo:**
   * **Muestra el tiempo restante para que el código sea válido siendo un máximo de 5 minutos.**
   * **Si el tiempo expira, el código ya no será aceptado y se deberá solicitar uno nuevo.**
5. **Botón "Verificar":**
   * **Verifica si el código es correcto y está dentro del tiempo de validez.**
   * **Si es válido, lleva automáticamente a la pantalla de restablecimiento de contraseña en caso contrario lo indicara con un mensaje de aviso.**
6. **Botón "Volver":**
   * **Regresa a la pantalla anterior para reenviar el código si es necesario.**

**🛠️ Instrucciones paso a paso para verificar el código**

1. **Revise su correo electrónico y localice el mensaje enviado por el sistema ReyBoxes.**
2. **Copie el código recibido.**
3. **Péguelo o escríbalo en el campo central de la pantalla.**
4. **Observe el tiempo restante: debe introducirse antes de que expire.**
5. **Pulse el botón "Verificar".**
6. **Si el código es correcto y aún válido, se abrirá la siguiente pantalla para restablecer su contraseña.**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

1. **Si hay algún error, revise que el código sea el correcto o vuelva a solicitar uno nuevo.**

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**🔐 7.1.3. Nueva contraseña**

**Una vez verificado el código correctamente, se muestra esta pantalla que permite al usuario restablecer su contraseña de acceso de forma segura.**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.**🧭 Descripción general**

**Esta pantalla incluye los siguientes elementos:**

1. **Título "Nueva Contraseña":**
   * **Continúa con el estilo visual coherente del programa.**
2. **Campo "Nueva contraseña":**
   * **Aquí se introduce la nueva contraseña deseada.**
   * **Es obligatorio completarlo.**
3. **Campo "Repetir contraseña":**
   * **Debe introducirse exactamente la misma contraseña para confirmar.**
   * **Si los valores no coinciden, no se podrá continuar.**
4. **Botón rojo "Guardar":**
   * **Guarda la nueva contraseña y la actualiza en la base de datos.**
   * **Tras guardar, el sistema redirige al login.**
5. **Botón rojo "Volver":**
   * **Permite regresar a la pantalla anterior (verificación).**

**🛠️ Instrucciones paso a paso para establecer una nueva contraseña**

1. **Escriba su nueva contraseña en el primer campo.**
2. **Vuelva a escribirla exactamente igual en el segundo campo para confirmarla.**
3. **Pulse el botón "Guardar".**
   * **Si ambas coinciden, su contraseña se actualizará correctamente.**
   * **Se mostrará un mensaje de confirmación y volverá automáticamente a la pantalla de inicio de sesión.**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. **Si las contraseñas no coinciden o directamente no se rellenan los campos, se mostrarán los avisos en cada circunstancia y no se guardará nada.**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

1. **Puede pulsar "Volver" si desea revisar o solicitar nuevamente el código anterior.**

**🏠 7.2. Menú principal (Dashboard)**

**Una vez iniciada la sesión correctamente, el usuario accede a esta pantalla central del sistema. Desde aquí se pueden abrir todas las funcionalidades del programa, según los permisos asociados al rol del usuario (Administrador, Mecánico o Administrativo).**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.**🧭 Descripción general**

**Elementos destacados en esta ventana:**

1. **Mensaje de bienvenida personalizado:**
   * **Se muestra el nombre del usuario en mayúsculas (ejemplo: CRESNIK).**
   * **Debajo aparece el rol del usuario (por ejemplo: Administrador).**
2. **Panel de botones centrales:**
   * **Se organiza en una cuadrícula de botones cuadrados con icono y texto.**
   * **Cada botón abre una sección distinta del sistema.**
   * **Los botones se muestran u ocultan según el rol del usuario:**
     + **🔧 Administrador: accede a todas las funciones.**
     + **🛠️ Mecánico: ve solo "Fichar", "Historial fichaje", "Vehículos" y "Presupuestos".**
     + **📋 Administrativo: ve "Clientes", "Recepcionamiento", "Reimpresión" y "Presupuestos".**
3. **Botones disponibles (ordenados de izquierda a derecha, fila a fila):**

| Funcionalidad | Descripción breve |
| --- | --- |
| Fichar | Registro de entrada o salida del personal. |
| Historial fichaje | Consulta de fichajes previos. |
| Crear usuarios | Alta de nuevos usuarios del sistema. |
| Clientes | Gestión completa de clientes. |
| Vehículos | Gestión de vehículos asociados a clientes. |
| Recepcionamiento | Registro de vehículos al llegar al taller. |
| Compraventa | Gestión de compra y venta de vehículos. |
| Presupuestos | Creación y seguimiento de presupuestos. |
| Reimpresión recepcionamientos | Permite volver a imprimir documentos anteriores. |

**🛠️ Instrucciones de uso**

1. Revise su nombre y rol en la parte superior.
2. Seleccione la sección deseada pulsando sobre el botón correspondiente.
3. Cada botón abrirá una nueva ventana donde podrá trabajar según los permisos otorgados.

**⏱️ 7.6. Fichajes**

El sistema ReyBoxes permite a los trabajadores **registrar sus entradas y salidas** del taller de forma rápida y visual. Esta funcionalidad está disponible para todos los empleados, y los datos quedan guardados en la base de datos para su posterior consulta por parte del administrador.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.**🧭 Descripción general**

La ventana de fichaje está diseñada para ser clara, compacta y directa:

1. **Hora actual**:
   * Se muestra en tiempo real, con formato HH:mm:ss.
2. **Opciones de tipo de fichaje**:
   * El usuario debe marcar **una sola opción**:
     + 🔘 **Entrada**
     + 🔘 **Salida**
   * Solo una puede estar activa al mismo tiempo.
   * Es obligatorio elegir una para poder fichar.
3. **Botón "Confirmar"**:
   * Registra el fichaje seleccionado con la hora exacta.
   * Al pulsarlo, se guarda el registro en la base de datos con el ID del usuario, tipo (entrada o salida), y hora.
4. **Botón "Volver"**:
   * Permite regresar al menú principal sin realizar ningún fichaje.

**🛠️ Instrucciones paso a paso para fichar**

1. Verifique que la **hora mostrada** es la correcta.
2. Seleccione el tipo de fichaje:
   * Marque **"Entrada"** al llegar al trabajo.
   * Marque **"Salida"** al finalizar la jornada.
3. Pulse el botón **"Confirmar"**.
4. Se registrará el fichaje y se volverá al menú principal automáticamente.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

1. Si desea cancelar, simplemente pulse **"Volver"**.

**⏱️ 7.6.1. Historial de Fichajes**

Desde esta pantalla es posible consultar todos los registros de entrada y salida realizados por los empleados. La información se presenta de forma ordenada y permite exportar los datos a distintos formatos para su análisis o archivo.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**🧭 Descripción general**

La pantalla está dividida en dos partes principales: la tabla de registros y los botones de acción.

**📊 Tabla de registros**

Contiene tres columnas:

| Columna | Contenido |
| --- | --- |
| Fecha y hora | Momento exacto del fichaje. Ej.: 2025-05-07 15:33:09. |
| Tipo | ENTRADA o SALIDA. |
| Empleado | Nombre del trabajador que realizó el fichaje. |

* La tabla muestra todos los registros cargados desde la base de datos, ordenados cronológicamente.
* La primera columna numera los registros para facilitar la lectura.
* Cada fila alterna colores para mayor legibilidad.

**🔘 Botones inferiores**

1. **Exportar CSV**
   * Genera un archivo .csv compatible con Excel y otros programas de hojas de cálculo.
   * Ideal para análisis o archivado externo.
2. **Exportar PDF**
   * Crea un informe en PDF con el logo de ReyBoxes, nombre de empresa, tabla de datos y pie de página.
   * El formato es profesional y apto para presentación.

Tabla

El contenido generado por IA puede ser incorrecto.

1. **Volver**
   * Regresa al menú principal sin realizar cambios.
   * Icono azul de retorno.

**👥 Diferencias según el rol del usuario**

**👑 Administrador**

* **Visualiza todos los fichajes** de todos los usuarios del sistema.
* El campo "Empleado" muestra el nombre de cada trabajador.

**🧑‍🔧 Mecánico / Administrativo**

* **Solo pueden ver sus propios fichajes**.
* El campo "Empleado" muestra únicamente su propio nombre.

Esto garantiza privacidad y control según el nivel de acceso.

**🛠️ Instrucciones paso a paso para consultar fichajes**

1. Abra la pantalla desde el botón **"Historial Fichaje"** en el menú principal.
2. Revise la tabla de registros.
3. Si lo desea:
   * Pulse **Exportar CSV** para obtener una hoja de cálculo.
   * Pulse **Exportar PDF** para generar un informe formal.
4. Pulse **Volver** para salir de esta ventana.

**👤 7.2.1. Crear usuarios**

Esta ventana permite registrar nuevos usuarios en el sistema ReyBoxes. Solo los usuarios con el rol **Administrador** tienen acceso a esta función.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.**🧭 Descripción general**

La interfaz está organizada dentro de un recuadro central con todos los campos necesarios para el alta de un nuevo usuario.

**📋 Campos del formulario**

1. **Nombre**
   * Campo obligatorio. Se guardará en mayúsculas automáticamente.
2. **Apellido**
   * Campo opcional, pero recomendable para identificar claramente al usuario.
3. **Correo electrónico**
   * Se utiliza para recuperación de cuenta.
   * Debe tener formato válido (usuario@dominio.com).
4. **Contraseña**
   * Clave de acceso del usuario.
5. **Repetir contraseña**
   * Debe coincidir exactamente con la anterior.
6. **Selecciona un rol**
   * Menú desplegable con las opciones:
     + Administrador
     + Mecánico
     + Administrativo
   * Es obligatorio seleccionar un rol para continuar.

**🔘 Botones de acción**

1. **Crear usuario**
   * Valida los datos y guarda el nuevo usuario.
   * Muestra mensaje de éxito o error según el resultado.
2. **Limpiar**
   * Vacía todos los campos del formulario.
   * Útil si se desea empezar de nuevo sin cerrar la ventana.
3. **Volver**
   * Regresa al menú principal sin guardar nada.

**🛠️ Instrucciones paso a paso para registrar un nuevo usuario**

1. Complete el campo **Nombre** (obligatorio).
2. Ingrese el **Apellido** (opcional).
3. Escriba un **correo electrónico válido**.
4. Introduzca la **contraseña** deseada.
5. Repita la contraseña para confirmar.
6. Seleccione un **rol de usuario** en el desplegable.
7. Haga clic en **"Crear usuario"**.
   * Si todo es correcto, el usuario quedará registrado y se limpiaran todos los campos para poder ingresar un nuevo usuario si se desea.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

* + En caso de error (usuario repetido, correos duplicados, etc.) se mostrará un aviso.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

1. Puede usar el botón **"Limpiar"** para reiniciar el formulario.
2. Pulse **"Volver"** para regresar sin realizar ningún cambio.