UNIVERSIDAD TECNOLÓGICA

DE LA ZONA METROPOLITANA DE GUADALAJARA

**MEMORIA TÉCNICA REALIZADA EN:**

Corporativo Textil JSN, S. de R.L de C.V

**PROYECTO:** Módulo de Gestión de Reprocesos y Reposiciones

**PARA OBTENER EL GRADO DE:**

Técnico Superior Universitario (TSU) en:

TECNOLOGÍAS DE LA INFORMACIÓN, ÁREA DESARROLLO DE SOFTWARE MULTIPLATAFORMA

PRESENTADO POR:

Víctor Manuel Montaño Juanpedro

Eduardo Isaías Vázquez Solís

**ASESOR INDUSTRIAL** **ASESOR ACADÉMICO**

Jair Efraín Barragan Gárcia Lizbeth Noriega Gútierrez

**COORDINADOR DE CARRERA**

Mtra. Lizbeth Noriega Gútierrez

TLAJOMULCO DE ZÚÑIGA, JALISCO, AGOSTO DEL 2025

UNIVERSIDAD TECNOLÓGICA DE LA ZONA
METROPOLITANA DE GUADALAJARA
DIRECCIÓN DE DESARROLLO Y GESTIÓN DE SOFTWARE


# Módulo de Gestión de Reprocesos y Reposiciones

**MEMORIA TÉCNICA REALIZADA EN:**

Corporativo Textil JSN, S. de R.L de C.V

**PARA OBTENER EL GRADO DE:**

Técnico Superior Universitario (TSU) en:

TECNOLOGÍAS DE LA INFORMACIÓN

Área, DESARROLLO DE SOFTWARE MULTIPLATAFORMA

PRESENTADO POR:

Víctor Manuel Montaño Juanpedro

Eduardo Isaías Vázquez Solís

AGOSTO 2025

*Agradezco a todas las personas .... Agradezco a todas las personas ....*

# Índice general

# Capítulo 1

# Introducción

Esta es la introducción jshdjhsdjkfhsdjk kjsdlkfjskad jdhshdfsdflksk, jdfjsdhjjkdsjkf. Esta es la introducción jshdjhsdjkfhsdjk kjsdlkfjskad jdhshdfsdflksk, jdfjsdhjjkdsjkf. Esta es la introducción jshdjhsdjkfhsdjk kjsdlkfjskad jdhshdfsdflksk, jdfjsdhjjkdsjkf. Esta es la introducción jshdjhsdjkfhsdjk kjsdlkfjskad jdhshdfsdflksk, jdfjsdhjjkdsjkf.

Esta es la introducción jshdjhsdjkfhsdjk kjsdlkfjskad jdhshdfsdflksk, jdfjsdhjjkdsjkf.

# Capítulo 2

# Antecedentes y Descripción de la Empresa

## 2.1. Ubicación

La empresa Corporativo Textil JSN, S. de R.L de C.V se encuentra ubicada en C. Flaviano Ramos Sur 37, Centro, 45640 Tlajomulco de Zúñiga, Jal. El mapa puede ser consultado en la Figura2.1

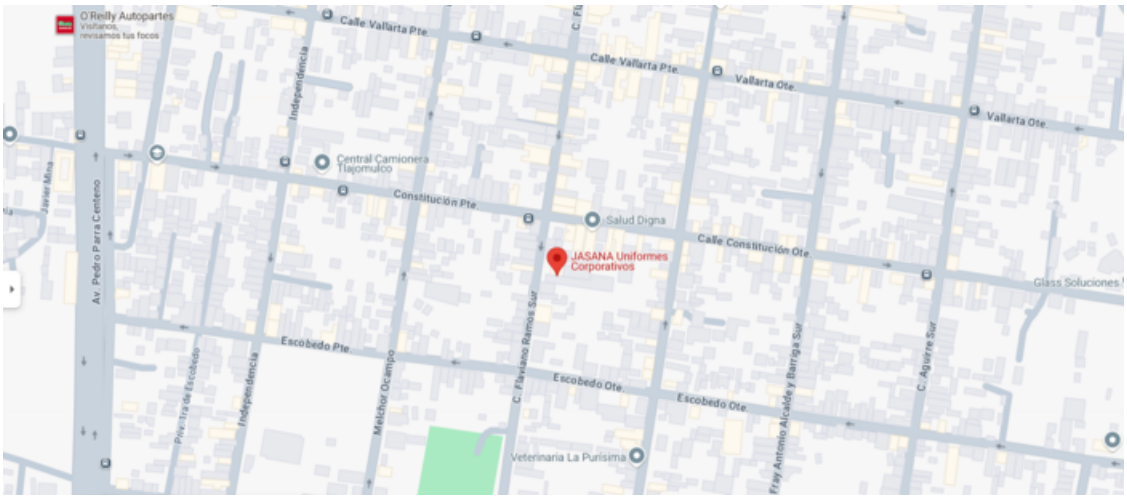


Figura 2.1: Ubicación de la empresa

## 2.2. Misión

Proyectar la identidad empresarial de sus clientes a través de la vestimenta de sus colaboradores, considerando su comodidad. Aplica diseños en tendencia, utiliza tecnologías actualizadas y apoya a la fuerza laboral conformada por familias mexicanas.

## 2.3. Visión

Consolidar una empresa transnacional en la transformación integral de textiles, estandarizando y digitalizando procesos en sus diferentes unidades de negocio para el año 2030.

## 2.4. Organigrama

A continuación en la figura 2.2 se presenta el organigrama de la empresa.

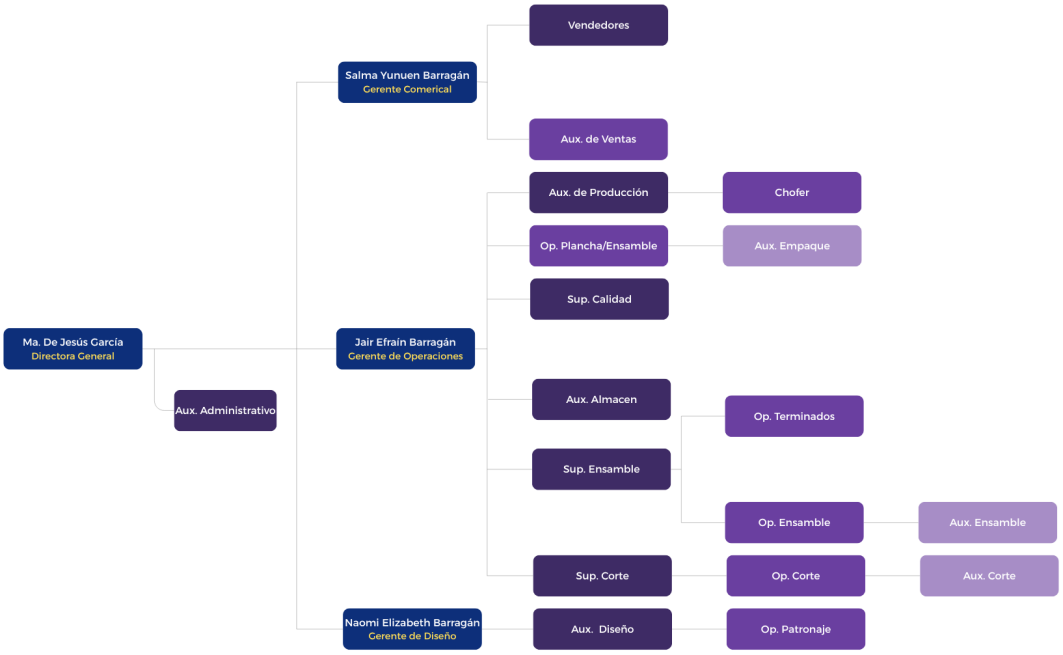


Figura 2.2: Organigrama de Textil JSN

## 2.5. Historia

En corporativo textil jsn, diseñamos y creamos prendas y textiles para hoteles y empresas de hospitalidad. Nos dedicamos a que los colaboradores de nuestros clientes estén presentables y cómodos todos los días, priorizando su seguridad e integridad. Esto lo logramos incorporando tecnologías en nuestras prendas que además aumentan su vida útil.

# Capítulo 3

# Problemática y Descripción del Proyecto

## 3.1. Problemática

La empresa **Corporativo Textil JSN**, dedicada a la fabricación de uniformes corporativos, enfrenta actualmente diversas dificultades en el control, trazabilidad y gestión eficiente de los *reprocesos y reposiciones* de piezas defectuosas o dañadas durante la producción. Aunque se utiliza el sistema ERP (Enterprise Resource Planning) Odoo, este proceso específico no se encuentra digitalizado ni debidamente integrado al entorno operativo, generando así múltiples problemáticas en distintas áreas del flujo productivo y administrativo.

Las principales problemáticas detectadas son las siguientes:

1. **Falta de trazabilidad:** No existe un registro digital estandarizado de qué piezas fueron reprocesadas, cuándo, por quién, ni en qué etapa del proceso ocurrieron los daños.

2. **Procesos manuales:** Actualmente, la información relacionada con daños, responsables y tiempos de corrección se gestiona mediante hojas físicas, lo cual incrementa el riesgo de pérdida de información y errores humanos.

3. **Descontrol en la gestión por áreas:** Las distintas áreas involucradas en la producción (Diseño, Corte, Ensamble, Planchado, entre otras) operan de forma aislada en cuanto al manejo de reprocesos, sin un sistema común que garantice la continuidad y visibilidad del flujo entre ellas.

4. **Dificultad para calcular tiempos y costos:** No se cuenta con una herramienta que permita realizar cálculos automáticos del tiempo invertido en cada reproceso, ni estimaciones precisas del costo de materiales e insumos reinvertidos.

Ante este panorama, se vuelve indispensable el desarrollo de una solución digital que permita sistematizar la gestión de reprocesos y reposiciones, garantizando control, transparencia, eficiencia operativa y toma de decisiones basada en información precisa y actualizada

## 3.2. Descripción del Proyecto

El proyecto se basa en el diseño y desarrollo de una herramienta a medida que permita gestionar de forma centralizada, digital y trazable los procesos de reposición y reproceso de piezas defectuosas en la cadena de producción de la empresa Corporativo Textil JSN.

A diferencia de un módulo integrado, esta aplicación funcionará de manera independiente al ERP Odoo que la empresa utiliza para otras operaciones dado a restricciones del propio ERP y cuestiones a nivel operacional dentro de la empresa, facilitando así una mayor flexibilidad en el desarrollo y una interfaz de usuario especializada para esta tarea. El sistema se construirá utilizando tecnologías web modernas, tales como **React** para el desarrollo de la interfaz de usuario, así como **Node.js** para el manejo del backend.

La aplicación web implementará las siguientes características clave para resolver la problemática:

- **Registro centralizado de solicitudes:** Gestión de todos los casos de reproceso y reposición con folios únicos para un seguimiento preciso.
- **Asignación de responsabilidades por área:** Flujos de trabajo que permiten asignar tareas y notificar a las áreas correspondientes (Diseño, Corte, Ensamble, etc.).
- **Cálculo automático de tiempos y costos:** Herramientas para registrar el tiempo invertido y los materiales utilizados en cada corrección.
- **Generación de reportes históricos:** Creación de informes y estadísticas para analizar tendencias, identificar cuellos de botella y fundamentar la toma de decisiones.

### Objetivo General

Desarrollar e implementar una herramienta que permita gestionar digitalmente las solicitudes de reproceso y reposición de la empresa, asegurando la trazabilidad del proceso, el control por área y la generación de reportes históricos para la toma de decisiones.

### Objetivos Específicos

- Permitir el registro eficiente de solicitudes con folios y datos clave de producción.

- Asignar y seguir responsabilidades por área en el flujo del reproceso.
- Automatizar el cálculo de tiempos de reproceso y sus costos estimados.
- Generar reportes que ayuden en el análisis.

### 3.2.1. Planeación

Por cuestiones de organización y aprovechamiento eficiente del tiempo que se tiene previsto para realizar el proyectó se ha desarrollado un plan de trabajo estructurado mediante un diagrama Gantt, el cual nos facilita una mejor gestión del tiempo creando una ayuda visual para las actividades, plazos, fases clave y progreso planeados en el desarrollo de esta herramienta. El conjunto de las siguientes imágenes (3.1 a 3.3) muestra una vista general de las actividades planeadas con sus respectivos espacios de tiempo y recursos

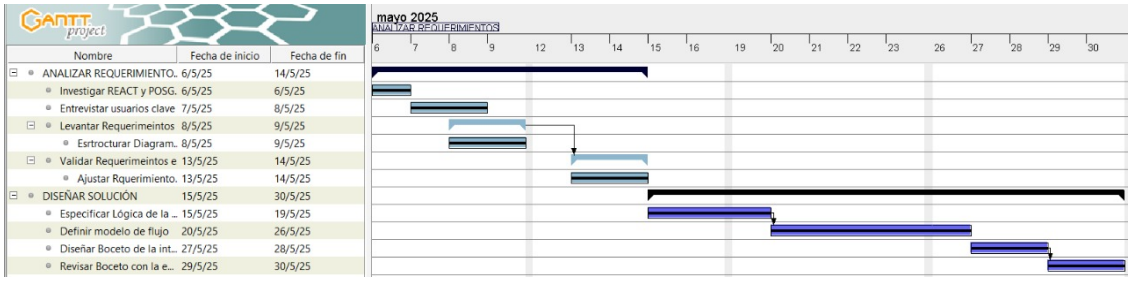


Figura 3.1: Periodo correspondientes a Mayo

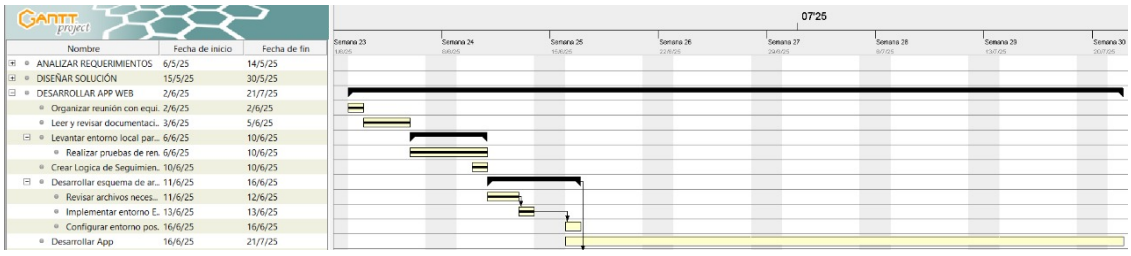


Figura 3.2: Periodo correspondiente a Junio-Julio

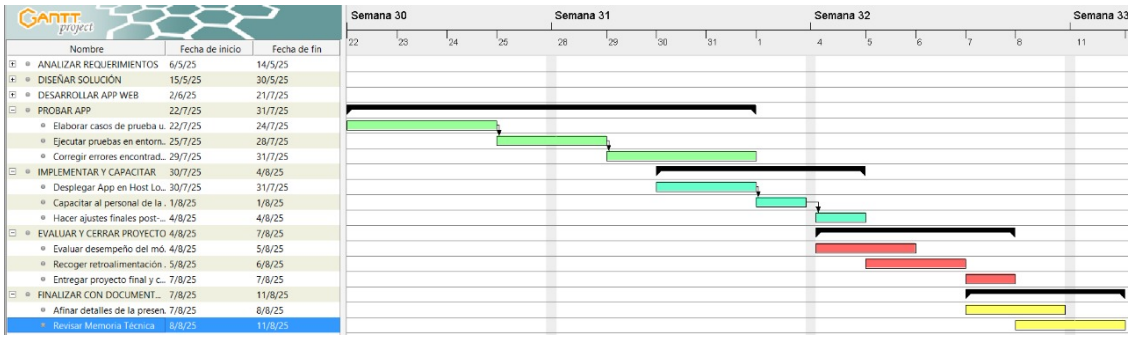


Figura 3.3: Periodo correspondiente a Julio-Agosto

# Capítulo 4

# Marco Teórico

Para comprender la construcción de este proyecto desde su núcleo, este capítulo presenta la información teórica y tecnológica indispensable. Su objetivo es exponer los conceptos que sustentan tanto el entorno del problema como la solución desarrollada, justificando las decisiones técnicas que se tomaron.

El contenido se divide en dos áreas principales. Primero, se analizará el contexto de la empresa, explorando los fundamentos de los Sistemas de Planificación de Recursos Empresariales (ERP) y la arquitectura de la plataforma Odoo. Esto es clave para entender las limitaciones que motivaron el desarrollo de una herramienta externa. En segundo lugar, el capítulo se enfocará en la solución, explicando los conceptos detrás de la arquitectura desacoplada y las tecnologías seleccionadas: React como librería para la interfaz de usuario, Vite como herramienta para optimizar el desarrollo y Node.js como el entorno para el servidor.

## 4.1. Sistemas ERP

Un sistema de Planificación de Recursos Empresariales (ERP) es mucho más que una simple herramienta tecnológica; se define como un sistema integrado que optimiza los procesos de negocio, abarcando desde la gestión financiera y de recursos humanos hasta la logística y la producción Hammouch (2024). Esta integración holística tiene como objetivo principal mejorar tanto la visibilidad de las operaciones como la eficiencia operativa, que son elementos cruciales en la gestión moderna.

La función principal de un ERP es gestionar los procesos de negocio centrales de una compañía en tiempo real, combinando diversas funciones en un único sistema unificado. Al centralizar los datos y agilizar las operaciones, los sistemas ERP conducen a una mayor eficiencia y facilitan una toma de decisiones más informada (Hammouch, 2024).

### El Legado de los Silos y la Evolución hacia la Integración

Históricamente, la mayoría de las empresas operaban con sistemas de información no integrados que soportaban únicamente las actividades de áreas funcionales individuales Monk and Wagner (2023). Esto significaba que una compañía podía tener un sistema de información para marketing, otro para producción y así sucesivamente, cada uno con su propio hardware,

software y métodos para procesar los datos. A esta configuración se le conoce como **silos** o *stovepipes*, ya que cada departamento poseía su propia pila de información, desconectada de las demás, lo que generaba ineficiencias significativas y falta de coordinación (Monk and Wagner, 2023).

La transición desde este modelo de silos hacia los sistemas integrados que hoy conocemos como ERP no fue repentina, sino una evolución gradual impulsada por la confluencia de varios factores clave. El surgimiento de los sistemas ERP actuales fue el resultado de tres elementos principales: (1) el avance de la tecnología de hardware y software, (2) el desarrollo de una visión de sistemas de información integrados, y (3) la reingeniería de las empresas para pasar de un enfoque funcional a uno centrado en los procesos de negocio (Monk and Wagner, 2023).

### 4.1.1. Las raíces en la manufactura: MRP y MRP II

Los orígenes de los sistemas de planificación de recursos empresariales se encuentran en el sector de la manufactura, donde la necesidad de gestionar eficientemente los materiales dio lugar a las primeras metodologías de planificación. Durante las décadas de 1960 y 1970, el aumento de la capacidad de cómputo permitió la creación de los primeros sistemas de **Planificación de Requerimientos de Materiales (MRP)**, cuyo objetivo era calcular las necesidades de materias primas y las fechas de sus pedidos para cumplir con los planes de producción, asegurando la disponibilidad de materiales y reduciendo los costos de inventario (Monk and Wagner, 2023; Miño-Cascante et al., 2015).

Posteriormente, estos sistemas evolucionaron hacia una planificación más integral. Esta evolución es descrita como **Planificación de Recursos de Manufactura (MRP II)**, un concepto que, a diferencia del MRP original, integraba no solo los materiales, sino también las capacidades productivas de la planta, los recursos monetarios y el personal. El MRP II fue un paso crucial hacia la integración, ya que comenzaba a coordinar diferentes áreas del ciclo de producción dentro de un marco unificado, sentando las bases para los futuros sistemas ERP (Miño-Cascante et al., 2015).

### 4.1.2. Los motores del cambio en los años 90

El verdadero nacimiento de los sistemas ERP, tal como los conocemos hoy, ocurrió a principios de los años 90, ya que los sistemas de hardware y software no fueron lo suficientemente complejos y potentes para soportarlos hasta esa década. Esta transición fue posible gracias a la confluencia de tres factores clave: el avance de la tecnología, el desarrollo de una visión de sistemas integrados y la reingeniería de las empresas hacia un enfoque en procesos (Monk and Wagner, 2023).

**Avances tecnológicos** La Ley de Moore describía el crecimiento exponencial del poder de cómputo, lo que hacía que el hardware fuera cada vez más potente y asequible. Crucialmente, la industria se movió de la costosa arquitectura *mainframe* a la más flexible y distribuida arquitectura cliente-servidor. Al mismo tiempo, el desarrollo y madurez de los sistemas de gestión de bases de datos relacionales (DBMS) durante los años 70 y 80 proporcionaron la tecnología necesaria para construir una base de datos centralizada y compartida (**?**)book.

**La visión de la integración** En el ámbito empresarial, las duras condiciones económicas de finales de los 80 y principios de los 90 impulsaron a las empresas a buscar una mayor eficiencia a través de la reorganización. Conceptos como la **Reingeniería de Procesos de Negocio (BPR)**, popularizados por autores como Michael Hammer, promovieron la idea de rediseñar radicalmente los procesos de negocio. Esto creó el ambiente perfecto para que los gerentes comenzaran a ver el software ERP como una solución a los problemas de negocio (Monk and Wagner, 2023).

**El surgimiento de SAP** La compañía alemana SAP, fundada en 1972 por ex-empleados de IBM, fue la pionera en materializar esta visión. En 1992, SAP lanzó su revolucionario software **R/3**, diseñado específicamente para la arquitectura cliente-servidor. Este sistema permitía a las empresas conectar sus diferentes áreas funcionales a una única base de datos subyacente, ofreciendo una solución integrada en tiempo real, lo que sentó las bases para el mercado ERP moderno (Monk and Wagner, 2023).

**El catalizador del Y2K y la expansión del mercado** Un evento inesperado aceleró masivamente la adopción de los sistemas ERP a finales de los 90: el **problema del año 2000 (Y2K)**. Muchas empresas utilizaban sistemas heredados (*legacy systems*) que almacenaban el año con solo dos dígitos y se enfrentaron a la decisión de reparar estos sistemas obsoletos o reemplazarlos por completo con una solución moderna e integrada. Muchas eligieron la segunda opción, lo que provocó un auge sin precedentes en las ventas de ERP, periodo durante el cual competidores como Oracle y PeopleSoft también ganaron una cuota de mercado significativa (Monk and Wagner, 2023).

A partir del nuevo milenio, los sistemas ERP continuaron evolucionando, incorporando funcionalidades de gestión de la relación con el cliente (CRM) y de la cadena de suministro (SCM), y adaptándose a nuevas tecnologías como el cómputo en la nube, el software como servicio (SaaS) y la movilidad (Monk and Wagner, 2023).

## 4.2. Antecedentes y trabajos previos:

Para contextualizar la presente investigación y fundamentar su relevancia, es imprescindible revisar trabajos anteriores que aborden problemáticas similares o que exploren el uso de las tecnologías propuestas. A continuación, se analizan dos trabajos de investigación recientes que sirven como antecedentes directos y técnicos para el desarrollo de esta propuesta: un proyecto de mejora de la gestión docente en una universidad pública utilizando Odoo y un desarrollo técnico de una aplicación específica sobre la misma plataforma.

### 4.2.1. Antecedente en la gestión de procesos docentes universitarios

Un antecedente de alta relevancia es el trabajo de investigación de maestría titulado *Propuesta de mejora para el proceso de gestión docente de la Universidad Nacional Autónoma de Chota, Pretell Cruzado (2024)* en la Escuela de Posgrado Newman, Perú. Este estudio aborda una problemática directamente análoga a la de la presente tesis, explorando la situación de una universidad pública con el objetivo de proponer una solución tecnológica que

mejore sus procesos de gestión docente. El autor identifica que, en el contexto de las universidades públicas, existen carencias y limitaciones en los procesos académicos, especialmente los referidos a la gestión docente. Estos procesos incluyen el reclutamiento y selección, la inducción, la formación continua, la evaluación del desempeño y el reconocimiento docente. La investigación de Pretell Cruzado tuvo como resultado el diseño y la validación del software ERP Odoo v.17 - edición Community como la solución tecnológica a implementar para sistematizar dichos procesos. La conclusión de su trabajo afirma que una solución de este tipo ofrece beneficios tangibles en el marco de la transformación digital, como un mejor acceso en línea a la información, mayor transparencia y la optimización general del proceso de gestión docente.

Este trabajo es fundamental como antecedente, ya que valida tanto la problemática abordada como la elección de Odoo ERP como una herramienta tecnológica viable y pertinente para mejorar la gestión docente en el ámbito universitario público.

### 4.2.2. Antecedente en el desarrollo de interfaces modernas en Odoo

Desde una perspectiva técnica, el trabajo de fin de grado *Desarrollo de una Aplicación en el sistema de código abierto Odoo para la Gestión de Residencias Tort Carrillo (2024)*, sirve como un excelente referente. Aunque el dominio de aplicación es distinto (residencias geriátricas), su enfoque se centra en el desarrollo de una interfaz de usuario (UI) moderna sobre la plataforma Odoo.

El objetivo principal del proyecto fue la creación de una interfaz de usuario intuitiva y adaptable, utilizando específicamente el framework de JavaScript **Odoo Web Library (OWL)**. OWL es descrito como una herramienta moderna y eficiente para la creación de aplicaciones web reactivas dentro del ecosistema de Odoo. Este antecedente es técnicamente relevante porque demuestra la capacidad de Odoo para ir más allá de sus interfaces estándar, permitiendo la construcción de soluciones visuales a medida para necesidades específicas.

Asimismo, el trabajo subraya la importancia de la integración del nuevo módulo con otros módulos preexistentes en Odoo para asegurar una gestión correcta del flujo de información. Por lo tanto, este estudio valida el uso de las herramientas de frontend más modernas de Odoo (OWL) para la creación de aplicaciones especializadas y destaca la arquitectura modular e

integrada de la plataforma.

## 4.3. Arquitectura y componentes de sistemas ERP:

La arquitectura de un sistema ERP define su estructura técnica, su funcionamiento y su capacidad de adaptación. Para comprender la diversidad de enfoques en el mercado, este capítulo realiza un análisis comparativo entre dos sistemas paradigmáticos: SAP, como representante del modelo de ERP tradicional y líder del mercado, y Odoo, como exponente de los sistemas ERP modernos de código abierto.

### 4.3.1. Modelo de negocio y licenciamiento

El modelo de negocio dicta cómo una empresa adquiere, implementa y mantiene su sistema ERP, lo cual tiene un impacto directo en el costo total de propiedad (TCO).

**SAP (Modelo Propietario Tradicional)** SAP opera bajo un modelo de software propietario y de código cerrado. Históricamente, esto implica costos significativos asociados a la adquisición de licencias de software, honorarios de consultores certificados para la implementación y contratos de mantenimiento anual para recibir soporte y actualizaciones. Este modelo, aunque costoso, ofrece a las grandes corporaciones un ecosistema robusto y un soporte centralizado.

**Odoo (Modelo *Open Core*)** En contraposición, Odoo se presenta como una plataforma de planificación de recursos empresariales de código abierto Pretell Cruzado (2024) que opera bajo un modelo de negocio conocido como "Open Core". Este modelo se manifiesta en dos versiones principales, permitiendo una gran flexibilidad para las organizaciones (Tort Carrillo, 2024):

1. **Odoo Community:** Es la versión gratuita y de código abierto, mantenida gracias a una comunidad global de desarrolladores.

2. **Odoo Enterprise:** Es la versión comercial de pago, que funciona bajo un modelo

de suscripción e incluye funcionalidades adicionales y servicios de soporte directo del fabricante.

Este modelo dual ofrece un punto de entrada de bajo costo para las empresas, especialmente para las PyMEs, y una ruta de escalamiento hacia una solución más robusta y soportada a medida que estas crecen.

### 4.3.2. Arquitectura tecnológica

La pila tecnológica (o *stack*) define los componentes de software y lenguajes sobre los que se construye el sistema.

**SAP (Arquitectura Cliente-Servidor y Lenguaje Propietario)** El éxito de SAP se fundamentó en su arquitectura cliente-servidor de tres capas: presentación (SAP GUI), aplicación y base de datos. El desarrollo y la personalización se realizan utilizando su propio lenguaje de programación patentado, **ABAP (Advanced Business Application Programming)**.

**Odoo (Arquitectura Web y Pila de Código Abierto)** Odoo está construido sobre una pila tecnológica moderna y de código abierto, diseñada para ser nativa de la web. Su arquitectura también sigue un modelo de tres capas:

1. **Base de Datos:** Utiliza exclusivamente **PostgreSQL**.
2. **Servidor (Lógica):** El núcleo está escrito en **Python** y cuenta con un potente **ORM (Mapeo Objeto-Relacional)** que facilita el desarrollo.
3. **Cliente (Presentación):** La interfaz es un cliente web moderno donde las vistas se definen con **XML** y la interactividad con **JavaScript**.

### 4.3.3. Modularidad y cobertura funcional

**SAP (Módulos Funcionales Clásicos)** SAP estructura su sistema en un conjunto de módulos funcionales altamente integrados pero bien definidos, que corresponden a las áreas

clásicas de una gran empresa: **SD** (Ventas y Distribución), **MM** (Gestión de Materiales), **PP** (Planificación de la Producción), **FI** (Contabilidad Financiera), **CO** (Controlling) y **HR** (Recursos Humanos).

**Odoo (Enfoque basado en "Apps")** Odoo adopta un enfoque hiper-modular, organizando su funcionalidad en "Apps" que se pueden instalar o desinstalar fácilmente. Las aplicaciones principales se corresponden directamente con los módulos clásicos de SAP:

- La App **Ventas** de Odoo es el análogo funcional de **SAP SD**.
- Las Apps **Inventario** y **Compras** cubren las funciones de **SAP MM**.
- La App **Fabricación** de Odoo corresponde al módulo **SAP PP**.
- La App **Contabilidad** integra funcionalidades de **SAP FI y CO**.

La gran diferencia radica en su ecosistema, donde existen miles de aplicaciones adicionales para cubrir necesidades muy específicas.

### 4.3.4. Flexibilidad y personalización

**SAP (Configuración vs. Personalización)** SAP distingue claramente entre **configuración** (ajustes mediante herramientas integradas) y **personalización** (modificar el código ABAP), siendo esta última una tarea compleja, costosa y que puede dificultar las actualizaciones del sistema.

**Odoo (Flexibilidad a través del Código Abierto)** La flexibilidad de Odoo es uno de sus principales argumentos de venta. La práctica estándar no es modificar el código del núcleo, sino **crear módulos personalizados** que se instalan sobre el sistema base. Estos módulos pueden añadir o alterar funcionalidades de forma segura, asegurando que el núcleo del sistema pueda ser actualizado sin afectar las personalizaciones.

## 4.4. Desarrollo de módulos en odoo

Toda la funcionalidad en Odoo está organizada en **módulos**. Toda la funcionalidad en Odoo está organizada en módulos. Según la documentación oficial de Odoo (Odoo S.A., 2024), un módulo es técnicamente un directorio que agrupa el código y los recursos del sistema para añadir o modificar una funcionalidad específica. Esta estructura es fundamental para el desarrollo, ya que consiste en la creación o extensión de estos módulos, en lugar de modificar el código fuente del núcleo del sistema (Reis and Mader, 2022). Esta arquitectura no solo garantiza que las personalizaciones se mantengan organizadas y separadas del código fuente del núcleo, facilitando el mantenimiento y las actualizaciones, sino que también permite que el sistema sea extremadamente escalable, pudiendo activar o desactivar funcionalidades completas según las necesidades del negocio.

### 4.4.1. Estructura fundamental de un módulo

Cada módulo de Odoo es un directorio cuyo nombre técnico es el que se utilizará para referenciarlo. Dentro de él, se encuentran archivos y subcarpetas que organizan sus componentes por convención. Una estructura completa suele incluir:

- **`__manifest__.py`**: Es el archivo descriptor del módulo y el único estrictamente obligatorio. Contiene un diccionario de Python con metadatos clave como el nombre del módulo (`name`), versión (`version`), autor (`author`), categoría funcional (`category`), un resumen de su propósito (`summary`), y, de forma crucial, sus dependencias (`depends`), que aseguran que otros módulos necesarios se instalen primero. También define los archivos de datos, vistas y seguridad a cargar.

- **`models/`**: Contiene los archivos de Python (`.py`) que definen los modelos de datos y su lógica de negocio. Es el corazón del módulo.

- **`views/`**: Contiene los archivos XML (`.xml`) que definen la interfaz de usuario: las vistas, los menús que las invocan y las acciones que las conectan.

- **`controllers/`**: Alberga los archivos Python que manejan las peticiones web. Son esenciales para crear páginas para el sitio web de Odoo o para definir puntos de acceso

(endpoints) para APIs RESTful personalizadas.

- `security/`: Contiene los archivos que definen los permisos de acceso. Principalmente el archivo `ir.model.access.csv`, que especifica qué grupos de usuarios pueden realizar operaciones de lectura, escritura, creación o borrado en cada modelo.
- `data/`: Contiene archivos XML utilizados para cargar datos iniciales o de configuración que el módulo necesita para funcionar, como secuencias, categorías de productos o plantillas de correo.
- `static/`: Contiene los archivos estáticos (CSS, Sass, JavaScript, imágenes, fuentes) que se usarán en la capa de presentación, tanto en el backend como en el frontend del sitio web.

### 4.4.2. Modelos (capa de datos y lógica)

Los modelos son la columna vertebral de Odoo. Son clases de Python que heredan de `models.Model` y utilizan el **ORM (Mapeo Objeto-Relacional)** de Odoo para mapear cada clase a una tabla en la base de datos PostgreSQL y cada instancia de la clase a un registro en esa tabla. El ORM implementa un patrón de tipo *Active Record*, donde los objetos llevan consigo tanto los datos como los métodos para operar sobre ellos. Esta capa es responsable de:

1. **Definición de la estructura de datos:** Se definen los campos como atributos de la clase. Odoo ofrece una gran variedad de tipos de campo, desde los simples como `Char`, `Integer`, `Float`, `Date`, `Datetime`, `Boolean` y `Html`, hasta los relacionales que son clave para un ERP: `Many2one` (para claves foráneas, como vincular una factura a un cliente), `One2many` (la relación inversa, como ver todas las facturas de un cliente) y `Many2many` (para relaciones múltiples, como asignar varias etiquetas a un producto).
2. **Implementación de la lógica de negocio:** Se escriben métodos en Python decorados para interactuar con el ORM. Decoradores como `@api.depends` permiten crear campos cuyos valores se calculan dinámicamente a partir de otros, mientras que

`@api.constrains` define reglas de validación de datos a nivel de base de datos para garantizar la consistencia.

3. **Herencia de Modelos:** Una de las características más potentes de Odoo es su mecanismo de herencia, que permite modificar modelos existentes de forma no intrusiva. Se puede extender un modelo existente para añadirle nuevos campos, modificar los atributos de campos existentes o sobreescribir sus métodos para adaptar su comportamiento a nuevas necesidades, todo sin tocar el código del módulo original.

### 4.4.3. Vistas (capa de presentación)

Las vistas en Odoo, definidas en **XML**, son la forma de presentar los datos de los modelos al usuario. El motor de Odoo interpreta estos archivos XML para generar dinámicamente las vistas HTML interactivas, utilizando un motor de plantillas llamado **QWeb**. Las vistas se conectan con los modelos a través de **Acciones** (`ir.actions.act_window`), que son registros en la base de datos que definen qué vistas mostrar para un modelo particular. Los tipos de vista principales son:

- **Vistas de formulario (`form`):** La vista detallada para un único registro. Su estructura puede ser altamente personalizada con elementos como pestañas, grupos y botones que ejecutan métodos del modelo.
- **Vistas de lista (`tree`):** Muestran múltiples registros en una tabla. Permiten la edición en línea y la visualización resumida de la información.
- **Vistas kanban (`kanban`):** Presentan los registros como tarjetas organizadas en columnas, ideal para visualizar flujos de trabajo basados en etapas (por ejemplo, un pipeline de ventas).
- **Vistas de búsqueda (`search`):** Definen los filtros, campos de búsqueda y opciones de agrupación (."grupar por") que el usuario puede utilizar para analizar los datos.

### 4.4.4. Controladores y workflows

Los **Controladores** son el mecanismo para crear lógica de servidor que responda a peticiones web. Heredan de la clase `http.Controller` y sus métodos se asocian a rutas (URLs) mediante un decorador. Esto permite crear desde simples páginas web con contenido dinámico hasta complejos puntos de acceso (endpoints) de API que devuelven datos en formato JSON.

El concepto de **Workflow** ha evolucionado. En versiones antiguas de Odoo (anteriores a la 8), existía un complejo sistema de workflows basado en XML que definía los flujos de documentos. Este sistema ha sido reemplazado por un enfoque más simple y flexible implementado directamente en Python. Hoy en día, un flujo de trabajo se modela típicamente con un campo de tipo `Selection` llamado `state` en el modelo (ej: 'borrador', énviado', áprobado', 'hecho'). La transición entre estados es gestionada por métodos del modelo (ej: `def action_confirm()`) que son llamados por botones en las vistas. Estos métodos contienen toda la lógica de negocio: cambiar el estado, crear otros registros, enviar correos, etc.

## 4.5. Tecnologías usadas

La plataforma Odoo se fundamenta en un modelo de despliegue y una filosofía de desarrollo que son clave para entender su flexibilidad y alcance. Estos se basan en los conceptos de Software como Servicio (SaaS) y el uso de tecnologías de código abierto (Open Source).

### 4.5.1. Modelo de despliegue: software como servicio (SaaS)

Un concepto clave en la oferta de los sistemas ERP modernos es el de Software como Servicio (SaaS). Este paradigma consiste en que un proveedor ofrece acceso a aplicaciones de software a través de internet, eliminando la necesidad de que el usuario descargue, instale o actualice programas en su propio equipo. Las ventajas de este enfoque son considerables, e incluyen ahorros significativos en costos relacionados con la adquisición de infraestructura de TI, mantenimiento y seguridad. Odoo ERP es un claro exponente de los programas que operan bajo el modelo SaaS, y su aplicación ha demostrado ser particularmente útil

para optimizar la actividad en entidades de investigación y educación, permitiendo una transformación digital acelerada (Dura et al., 2022).

### 4.5.2. Filosofía de código abierto (open source)

Además de su modelo de despliegue, Odoo se apoya en una pila tecnológica basada casi en su totalidad en tecnologías de código abierto (*open source*). El software con licencia de código abierto se caracteriza por proveer un código fuente que cualquiera puede ver, usar, modificar y distribuir de manera gratuita. Esta elección deliberada no solo reduce los cotos de licenciamiento, sino que también permite a Odoo beneficiarse de la innovación y la seguridad impulsadas por una extensa comunidad global de programadores que, de manera continua, desarrollan nuevos módulos y favorecen la adaptabilidad del sistema (Pretell Cruzado, 2024).

### 4.5.3. Backend: Python

El lenguaje de programación principal para todo el desarrollo del backend en Odoo es **Python**. La elección de este lenguaje es estratégica debido a su sintaxis limpia y legible, que reduce los costos de mantenimiento, y a su vasto ecosistema de librerías de terceros Python Software Foundation (2025). El propio framework de Odoo, desarrollado sobre Python, provee toda la infraestructura necesaria para las aplicaciones de negocio, incluyendo un ORM para la gestión de datos, un motor de plantillas y un servidor web integrado (Pretell Cruzado, 2024).

### 4.5.4. Base de Datos: PostgreSQL

Para su sistema de gestión de bases de datos (SGBD), Odoo utiliza exclusivamente **PostgreSQL**. Esta no es una simple preferencia, sino un requisito técnico estricto. La razón es que Odoo está profundamente integrado con PostgreSQL para garantizar la integridad y consistencia de los datos empresariales, aprovechando sus características avanzadas para gestionar operaciones de alta concurrencia, algo esencial en cualquier sistema ERP (The PostgreSQL Global Development Group, 2025).

### 4.5.5. Frontend: XML, JavaScript y OWL

Como lo detalla Tort Carrillo (2024), la capa de presentación que ve el usuario en Odoo es una amalgama de tecnologías web estándar y un framework propio. Esta arquitectura permite crear interfaces flexibles y dinámicas, combinando el uso declarativo de XML para la estructura, JavaScript y el framework OWL para la interactividad, y Sass para el diseño visual. A continuación, se describen estos componentes:

- **XML (Extensible Markup Language):** Su rol es puramente declarativo. Se utiliza para definir la estructura de las vistas, los menús, las acciones y los registros de datos que Odoo leerá para renderizar dinámicamente la interfaz.
- **JavaScript:** Es el motor de toda la interactividad en el navegador. Las versiones más modernas de Odoo se basan en un framework de JavaScript propio para la creación de componentes reactivos.
- **OWL (Odoo Web Library):** Es el framework de JavaScript de Odoo, inspirado en herramientas modernas como React y Vue.js. OWL es un sistema de componentes declarativo y de alto rendimiento, diseñado para construir las interfaces complejas que requiere una aplicación de negocio.
- **CSS y Sass:** Para el diseño y la apariencia visual, Odoo utiliza hojas de estilo en cascada (CSS) y se apoya en el preprocesador **Sass**, que permite escribir estilos de manera más eficiente, estructurada y mantenible.

### 4.5.6. Protocolos de Comunicación y Servidor

La comunicación entre el servidor Odoo y el mundo exterior se realiza a través de protocolos estándar. El servidor web principal que atiende las peticiones está integrado en el propio Odoo y se basa en la librería *Werkzeug* de Python desarrollada por Pallets Projects (2024). Para la integración con sistemas de terceros, Odoo expone su API a través de dos protocolos:

- **XML-RPC:** Es el protocolo histórico y más robusto, soportado desde las primeras versiones.

- **JSON-RPC:** Es una alternativa más moderna, ligera y fácil de consumir desde aplicaciones web basadas en JavaScript.

Ambas APIs están securizadas y requieren autenticación, respetando los permisos y las reglas de acceso definidos en la capa de seguridad de Odoo.

### 4.5.7. React y el Desarrollo de Interfaces de Usuario Modernas

La elección de una tecnología para la interfaz de usuario (frontend) es un factor determinante en la usabilidad, el rendimiento y la mantenibilidad de una aplicación web moderna. Si bien el ecosistema de `Odoo` provee su propia librería (`OWL`), el desarrollo de una aplicación independiente permite el uso de herramientas especializadas. En este contexto, `React` se posiciona como una de las librerías de `JavaScript` de código abierto más influyentes para la construcción de interfaces interactivas y reutilizables (Murgueytio et al., 2022).

#### Paradigma Declarativo y Arquitectura Basada en Componentes

El paradigma de `React` se fundamenta en una arquitectura basada en **componentes** y un enfoque **declarativo**. Permite a los desarrolladores construir interfaces de usuario complejas a partir de piezas de código pequeñas y aisladas que encapsulan su propia lógica y presentación. Para ello, se utiliza la extensión de sintaxis **JSX**, la cual permite escribir una estructura similar a HTML directamente en el código `JavaScript`. Este enfoque hace que el código sea más predecible y ordenado, ya que el desarrollador solo necesita describir el estado final de la interfaz, y `React` se encarga de las actualizaciones de manera eficiente (Meta and the React Team, 2025; Taipe Toapaxi and Quishpe Caizatoa, 2022).

#### El Virtual DOM y su Impacto en el Rendimiento

Una de las innovaciones técnicas más significativas de `React` es el uso de un **DOM Virtual (VDOM)**. En lugar de manipular directamente el costoso DOM del navegador, `React` mantiene una copia ligera de su estructura en memoria. Cuando el estado de un componente cambia, un algoritmo de "diferenciación"(*diffing*) calcula el conjunto mínimo de cambios

necesarios entre la nueva y la antigua versión del VDOM. Finalmente, este proceso de **reconciliación** aplica únicamente dichas diferencias al DOM real, optimizando drásticamente el rendimiento y la velocidad de la aplicación (Taipe Toapaxi and Quishpe Caizatoa, 2022; Meta and the React Team, 2025).

#### Gestión del Estado y Flujo de Datos con Hooks

En `React`, el **estado (state)** contiene los datos que describen la condición de un componente y que pueden cambiar con el tiempo. Para gestionar este estado, se introdujeron los **Hooks**, que son funciones que permiten .[en]gancharse.[a] las características de `React` desde componentes funcionales sin necesidad de escribir una clase. Los más esenciales son `useState`, para añadir estado local, y `useEffect`, para ejecutar efectos secundarios, como peticiones a una API, después del renderizado del componente (Meta and the React Team, 2025).

#### Ecosistema y Herramientas de Construcción: Vite

El desarrollo moderno depende de **herramientas de construcción** (*build tools*) que empaquetan y optimizan el código para producción. **Vite** es una herramienta de nueva generación que busca proporcionar una experiencia de desarrollo notablemente más rápida. A diferencia de empaquetadores tradicionales, aprovecha los **módulos ES nativos (ESM)** del navegador durante el desarrollo para servir archivos bajo demanda, lo que resulta en un inicio del servidor y una Recarga de Módulos en Caliente (HMR) casi instantáneos, mejorando así la productividad del desarrollador (DX) (Evan You and Vite Contributors, 2025).

### 4.5.8. Arquitectura Desacoplada y el Rol del Backend con Node.js

La decisión de construir una aplicación independiente a `Odoo` implica la adopción de un patrón arquitectónico específico, justificado por las limitaciones de los sistemas monolíticos en contextos que demandan alta flexibilidad.

### Justificación: Limitaciones de los ERP Tradicionales

Un estudio realizado por Correal Rodríguez (2021) sobre la implementación de sistemas ERP revela varias desventajas clave que validan la necesidad de soluciones externas y especializadas. Entre las más relevantes se encuentran:

- **Obsolescencia y falta de integración:** Los sistemas ERP pueden volverse obsoletos y presentar una notable falta de integración con herramientas modernas, lo que impide que la información fluya de manera oportuna.
- **Procesamiento de datos externo:** Esta falta de integración obliga a los usuarios a recurrir a herramientas externas como `Excel` para procesar los datos y convertirlos en información útil.
- **Rigidez ante el cambio:** Se requieren herramientas más dinámicas que se adapten rápidamente a las transformaciones de la empresa sin demandar largos y costosos tiempos de desarrollo.

Estas limitaciones, identificadas en el estudio de Correal Rodríguez (2021), fundamentan la decisión estratégica de optar por una arquitectura más flexible para este proyecto.

### Arquitectura Desacoplada y la API como Contrato

Frente al modelo monolítico, una arquitectura desacoplada (decoupled o headless) separa completamente el frontend (la capa de presentación) del backend (la capa de lógica de negocio y datos). La comunicación entre ambas capas se establece a través de una API (Interfaz de Programación de Aplicaciones), la cual actúa como un enlace que posibilita que las aplicaciones se comuniquen entre sí e intercambien datos (Taipe Toapaxi and Quishpe Caizatoa, 2022). La API define un contrato con un conjunto de reglas y *endpoints* que el frontend utiliza para solicitar o enviar información.

### Node.js como Entorno para el Backend

Para implementar el backend, este proyecto utiliza Node.js, definido por su documentación oficial como un entorno de ejecución de JavaScript asíncrono, impulsado por eventos,

diseñado para construir aplicaciones de red escalables (OpenJS Foundation, 2025). Su modelo de entrada/salida (I/O) sin bloqueo lo hace ideal para construir APIs de alto rendimiento que exponen una API REST. Este estilo de arquitectura utiliza los verbos del protocolo HTTP y el formato JSON para el intercambio de datos, estableciendo la comunicación con la aplicación React de una manera estandarizada y eficiente (Taipe Toapaxi and Quishpe Caizatoa, 2022).

# Capítulo 5

# Desarrollo del Proyecto

## 5.1. Requerimientos

### 5.1.1. Requerimientos funcionales y no funcionales

La fase de levantamiento de requerimientos es el pilar fundamental de cualquier proyecto de software, ya en ella se definen las capacidades y condiciones que la solución final debe cumplir. En esta sección, se detallan formalmente dichas condiciones, las cuales fueron recopiladas a partir del análisis de la problemática y las necesidades de la empresa.

Para una mayor claridad, los requerimientos se han clasificado en dos categorías: Requerimientos Funcionales, que describen qué debe hacer el sistema, y Requerimientos No Funcionales, que definen cómo debe ser el sistema en términos de calidad y operación. Ambas categorías se pueden consultar en los cuadros 5.1 al 5.10 para los requerimientos funcionales y en los cuadros 5.11 al 5.13 para los no funcionales.

Cuadro 5.1: Requerimiento Funcional: Registro de Solicitudes

| | |
|---|---|
| **ID** | RF-001 |
| **Nombre** | Registro de Solicitudes |
| **Tipo** | Requerimiento Funcional |
| **Prioridad** | Alta |
| **Descripción:** El sistema debe proporcionar un formulario intuitivo para que usuarios autorizados registren nuevas solicitudes de reproceso o reparación. El formulario debe validar los campos obligatorios antes de guardar y mostrar confirmación al usuario. Debe integrarse con el módulo de autenticación para identificar automáticamente al solicitante. | |

Cuadro 5.2: Requerimiento Funcional: Generación Automática de Folios

| | |
|---|---|
| **ID** | RF-002 |
| **Nombre** | Generación Automática de Folios |
| **Tipo** | Requerimiento Funcional (Reglas de negocio) |
| **Prioridad** | Alta |
| **Descripción:** Cada nueva solicitud debe generar un folio único con el formato: JN-REQ-MM-AA-CCC, donde: JN-REQ es prefijo fijo, MM-AA representa el mes y año actual, y CCC es un contador consecutivo de 3 dígitos que se reinicia mensualmente. El sistema debe garantizar que no existan folios duplicados. Ejemplo: JN-REQ-05-25-042. | |

Cuadro 5.3: Requerimiento Funcional: Gestión de Datos de Solicitud

| | |
|---|---|
| **ID** | RF-003 |
| **Nombre** | Gestión de Datos de Solicitud |
| **Tipo** | Requerimiento Funcional (Gestión de datos) |
| **Prioridad** | Alta |
| **Descripción:** Las solicitudes deben almacenar: Fecha de solicitud (Autogenerada), Solicitante (Obtenido del usuario autenticado), Área (Lista desplegable), Estado (Flujo definido), Costes (Campo restringido), Fecha de entrega (Editable), Tiempo total (Calculado) y Observaciones (Texto libre). | |

Cuadro 5.4: Requerimiento Funcional: Seguimiento de Reparación

| | |
|---|---|
| **ID** | RF-004 |
| **Nombre** | Seguimiento de Reparación (Registro de procesos) |
| **Tipo** | Requerimiento Funcional |
| **Prioridad** | Media |

**Descripción:** Para cada etapa del reproceso, el sistema debe permitir registrar: el área donde se realiza la operación, el operador implicado, el supervisor encargado, la fecha (autogenerada) y las horas de inicio y fin con validación. Los registros deben vincularse a la solicitud principal.

Cuadro 5.5: Requerimiento Funcional: Registro Múltiple de Piezas

| | |
|---|---|
| **ID** | RF-005 |
| **Nombre** | Registro Múltiple de Piezas |
| **Tipo** | Requerimiento Funcional (Entrada de datos) |
| **Prioridad** | Alta |

**Descripción:** El sistema debe permitir agregar N piezas a una solicitud mediante un botón .Añadir pieza"que muestre un subformulario. Cada pieza debe guardarse individualmente y asociarse al folio de la solicitud padre. El usuario debe poder editar o eliminar piezas antes de finalizar el proceso.

Cuadro 5.6: Requerimiento Funcional: Detalles de Piezas

| | |
|---|---|
| **ID** | RF-006 |
| **Nombre** | Detalles de Piezas |
| **Tipo** | Requerimiento Funcional (Gestión de datos) |
| **Prioridad** | Alta |

**Descripción:** Cada pieza debe contener: No. Solicitud (no editable), Modelo/Tela/Color (con autocompletado), Tipo de pieza (desplegable), Descripción del daño (obligatorio, con opción para adjuntar imágenes) y Responsable (opcional).

Cuadro 5.7: Requerimiento Funcional: Aprobación/Rechazo por Supervisor

| | |
|---|---|
| **ID** | RF-007 |
| **Nombre** | Aprobación/Rechazo por Supervisor |
| **Tipo** | Requerimiento Funcional (Flujo de trabajo) |
| **Prioridad** | Media |
| **Descripción:** Los supervisores deben recibir notificaciones de solicitudes En revisión. Desde su panel, podrán aprobar (cambia estado y notifica) o rechazar (requiere una razón obligatoria). Todas las acciones se registran en una bitácora. | |

Cuadro 5.8: Requerimiento Funcional: Finalización del Proceso

| | |
|---|---|
| **ID** | RF-008 |
| **Nombre** | Finalización del Proceso |
| **Tipo** | Requerimiento Funcional (Flujo de trabajo) |
| **Prioridad** | Alta |
| **Descripción:** Cuando la última operación se marca como Finalizada, el sistema debe cambiar el estado de la solicitud a Completado, calcular el tiempo total, notificar a los interesados por correo y bloquear las ediciones posteriores (modo solo lectura). | |

Cuadro 5.9: Requerimiento Funcional: Autenticación de Usuarios

| | |
|---|---|
| **ID** | RF-009 |
| **Nombre** | Autenticación de Usuarios |
| **Tipo** | Requerimiento Funcional (Seguridad) |
| **Prioridad** | Alta |
| **Descripción:** El sistema debe requerir inicio de sesión para acceder a sus funcionalidades. Las contraseñas deben almacenarse de forma encriptada y el sistema debe implementar un bloqueo temporal tras 3 intentos de inicio de sesión fallidos. | |

Cuadro 5.10: Requerimiento Funcional: Control de Permisos por Roles

| | |
|---|---|
| **ID** | RF-010 |
| **Nombre** | Control de Permisos |
| **Tipo** | Requerimiento Funcional (Seguridad) |
| **Prioridad** | Alta |
| **Descripción:** El sistema debe gestionar permisos diferenciados por roles: **Supervisores** (aprobar/rechazar, ver costes), **Operadores** (registrar solicitudes en su área) y **Administradores** (CRUD completo de solicitudes y usuarios). | |

Cuadro 5.11: Requerimiento No Funcional: Accesibilidad y Compatibilidad

| | |
|---|---|
| **ID** | RNF-001 |
| **Nombre** | Accesibilidad y Compatibilidad |
| **Tipo** | Requerimiento No Funcional |
| **Prioridad** | Alta |
| **Descripción:** El sistema debe ser accesible desde cualquier dispositivo con conexión a internet y un navegador web moderno (ej. Chrome, Firefox, Edge). | |

Cuadro 5.12: Requerimiento No Funcional: Disponibilidad y Respaldo

| | |
|---|---|
| **ID** | RNF-002 |
| **Nombre** | Disponibilidad y Respaldo |
| **Tipo** | Requerimiento No Funcional |
| **Prioridad** | Alta |
| **Descripción:** La base de datos del sistema debe ser respaldada automáticamente al menos una vez al día para garantizar la recuperación de datos en caso de una falla. | |

Cuadro 5.13: Requerimiento No Funcional: Usabilidad

| | |
|---|---|
| **ID** | RNF-003 |
| **Nombre** | Usabilidad |
| **Tipo** | Requerimiento No Funcional |
| **Prioridad** | Alta |
| **Descripción:** La interfaz de usuario debe ser intuitiva y fácil de usar, permitiendo que el personal con conocimientos básicos de computación pueda operar el sistema eficientemente con un mínimo de capacitación. | |

### 5.1.2. Casos de uso

A continuación se presenta los casos de uso junto a su respectivo diagrama (figuras 5.1 a 5.6) que se identificaron en base a los requerimientos para el desarrollo de esta solución tecnológica y qué, a su ver, ayudaron a comprender de mejor manera la construcción del sistema.

- **Caso de Uso 001: Registrar Nueva Solicitud** Permite a los Supervisores de área crear nuevas solicitudes de reproceso , generando un folio único de forma automática.
- **Caso de Uso 002: Registrar Piezas en Solicitud** Permite a Operadores y Administradores añadir, editar o eliminar múltiples piezas detallando el daño en una solicitud.
- **Caso de Uso 003: Aprobar o Rechazar Solicitud** Permite al Supervisor aprobar o rechazar las solicitudes que se encuentran **En revisión**, registrando una justificación obligatoria en caso de rechazo.
- **Caso de Uso 004: Finalizar Proceso de Reproceso** Permite al Supervisor marcar la última operación como terminada para cambiar el estado de la solicitud a **Completado**, notificando a los involucrados y bloqueando ediciones futuras.
- **Caso de Uso 005: Registrar Avance de Reparación** Permite al Supervisor registrar el avance, los tiempos y el operador de cada etapa del reproceso.

- **Caso de Uso 006: Autenticarse en el Sistema** Requiere que todos los usuarios inicien sesión para acceder a las funcionalidades permitidas para su rol.

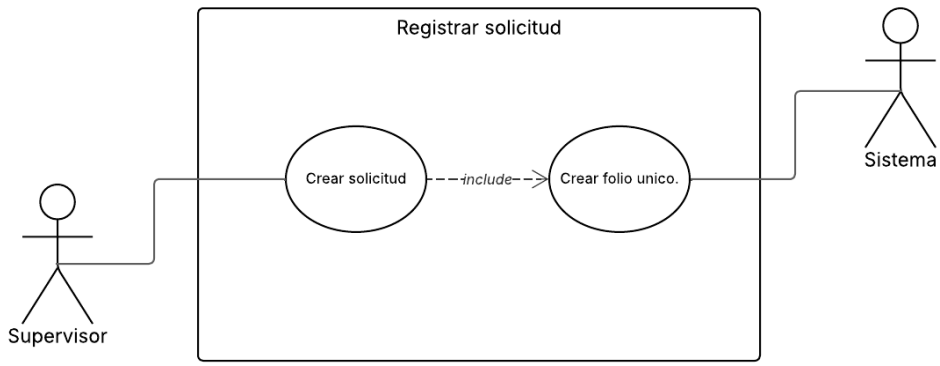


Figura 5.1: Caso de Uso 001

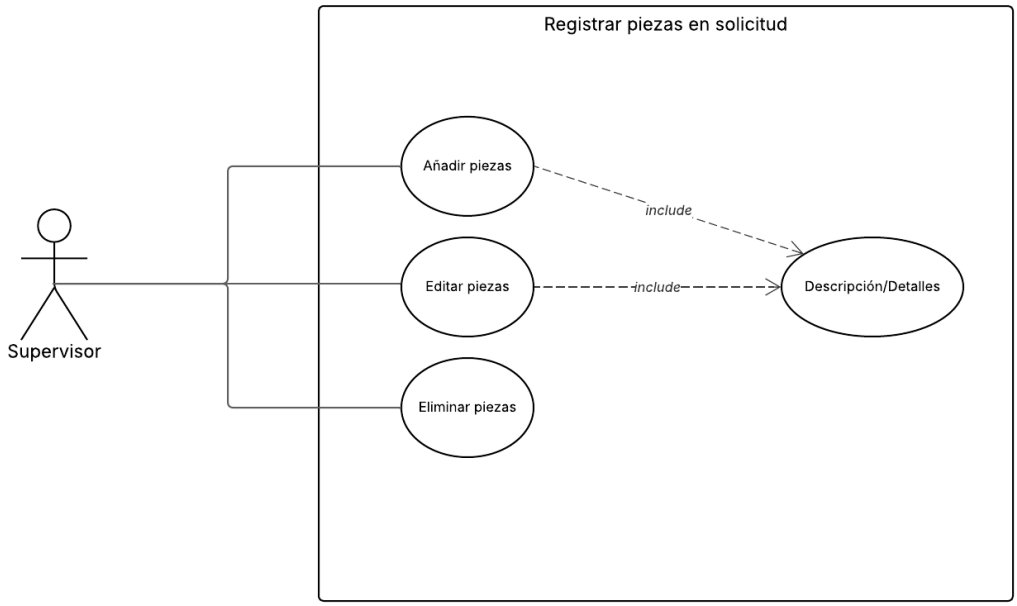


Figura 5.2: Caso de Uso 002

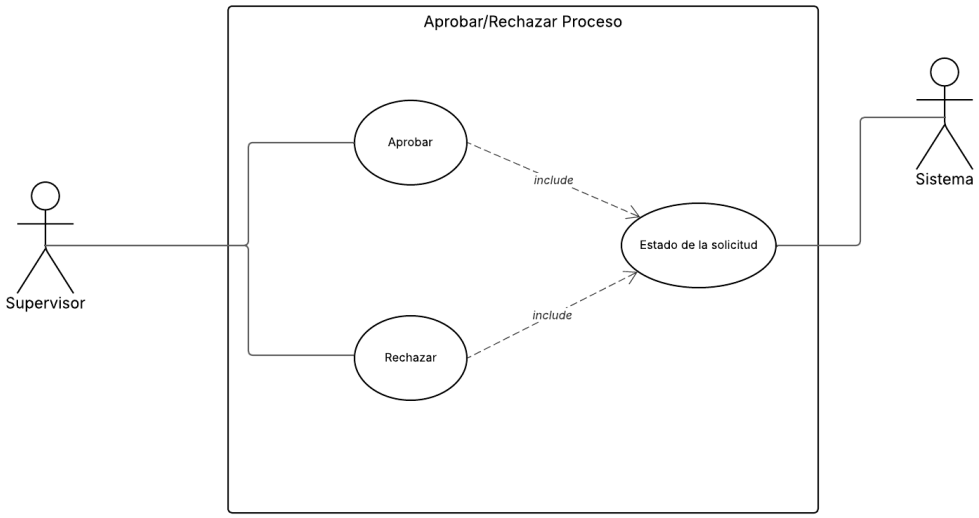


Figura 5.3: Caso de Uso 003

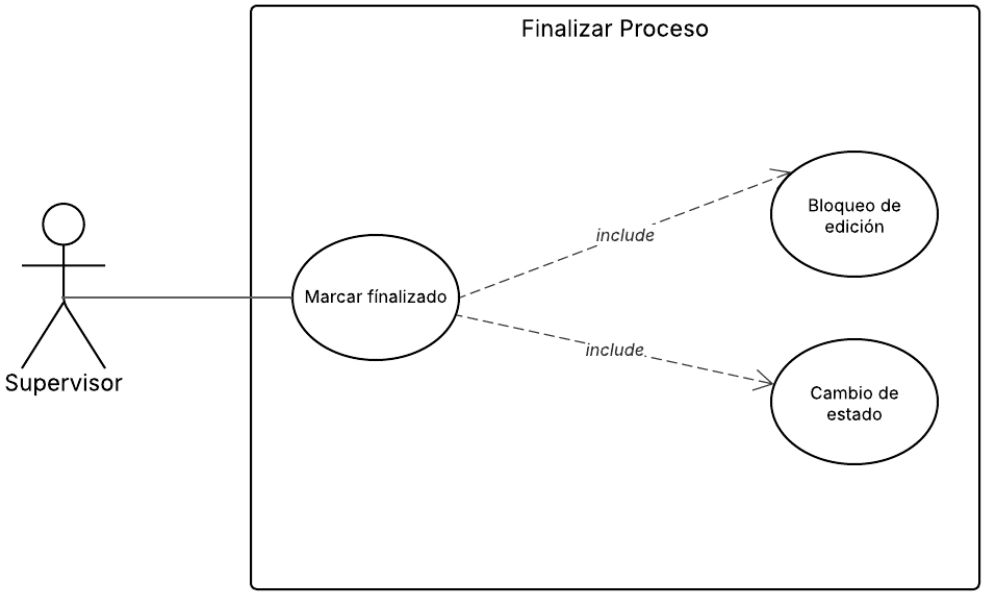


Figura 5.4: Caso de Uso 004

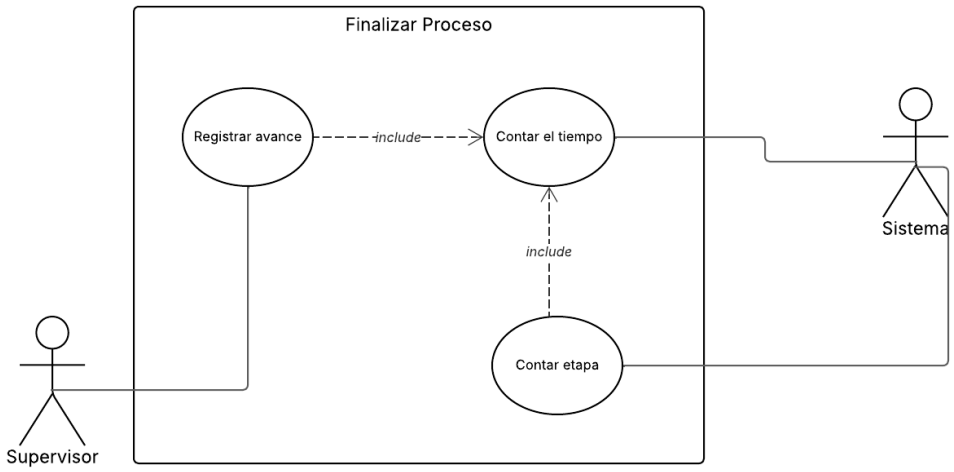


Figura 5.5: Caso de Uso 005

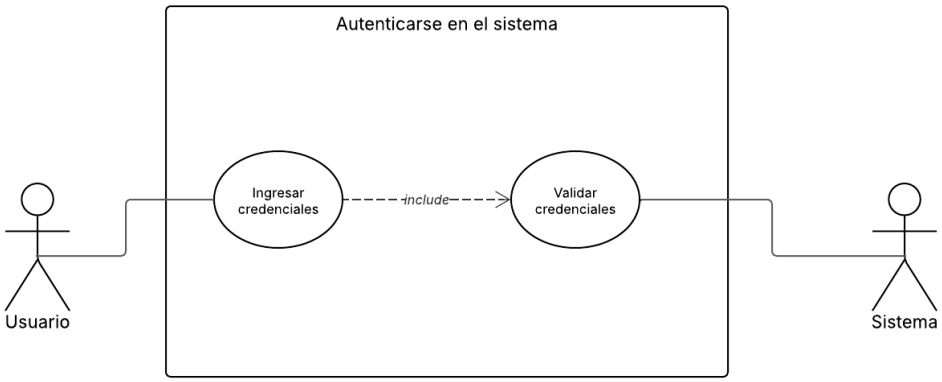


Figura 5.6: Caso de Uso 006

## 5.2. Análisis y Diseño

### 5.2.1. Elección de la Arquitectura: Aplicación Externa vs. Módulo Integrado

Tras el levantamiento y análisis de requerimientos, se procedió a la fase de diseño, donde la decisión más estratégica fue la elección de la arquitectura de software. Inicialmente, la solución más directa aparentaba ser el desarrollo de un módulo integrado directamente en la plataforma `Odoo`, dado que es el ERP en uso por la empresa. Esta opción, en teoría, permitiría centralizar la operación y aprovechar la infraestructura existente.

Sin embargo, un análisis más profundo reveló una serie de restricciones técnicas y operacionales que hacían de esta alternativa una vía no óptima para el proyecto. Como se menciona en la descripción del proyecto, existían "...restricciones del propio ERP y cuestiones a nivel operacional...". Estas se pueden detallar en los siguientes puntos:

- **Restricciones de Interfaz de Usuario (UX):** Se estableció como requisito la creación de una experiencia de usuario altamente especializada, fluida y dinámica. Lograr este nivel de interactividad con el framework nativo de `Odoo` (`OWL`) habría supuesto una curva de aprendizaje y una complejidad de desarrollo significativamente mayores en comparación con librerías de vanguardia como `React`, diseñada específicamente para este propósito.

- **Flexibilidad y Ciclo de Desarrollo:** El desarrollo de un módulo integrado está fuertemente acoplado a los ciclos y versiones de `Odoo`. El desarrollo de una aplicación externa permitía un ciclo de vida independiente, facilitando actualizaciones y mantenimientos futuros sin afectar la estabilidad del ERP principal.

- **Restricciones de Acceso y Despliegue:** Durante la fase de análisis, se identificó que las políticas administrativas de la empresa para garantizar la estabilidad del ERP limitaban el acceso para el despliegue de nuevos módulos de terceros, lo que representaba un riesgo para el cumplimiento de los plazos del proyecto.

En virtud de estas limitaciones, se tomó la decisión estratégica de optar por una **arquitectura desacoplada**. Se determinó que el desarrollo de una herramienta externa, utilizando un stack tecnológico moderno con `React` en el frontend y `Node.js` en el backend, era la solución más eficiente y robusta.

### 5.2.2. Diagrama E-R

El diseño bien logrado de una base de datos es el pilar fundamental en la arquitectura de un sistema, pues es con esto que se define el cómo se almacenará y organizará la información Para este proyecto, se ha utilizado el sistema gestor de bases de datos relacional PostgreSQL, por su robustez y compatibilidad con el ecosistema de Node.js. El siguiente diagrama Entidad-Relación (Figura 5.7) ilustra la estructura diseñada, mostrando las tablas principales y las relaciones que garantizan la integridad de los datos.

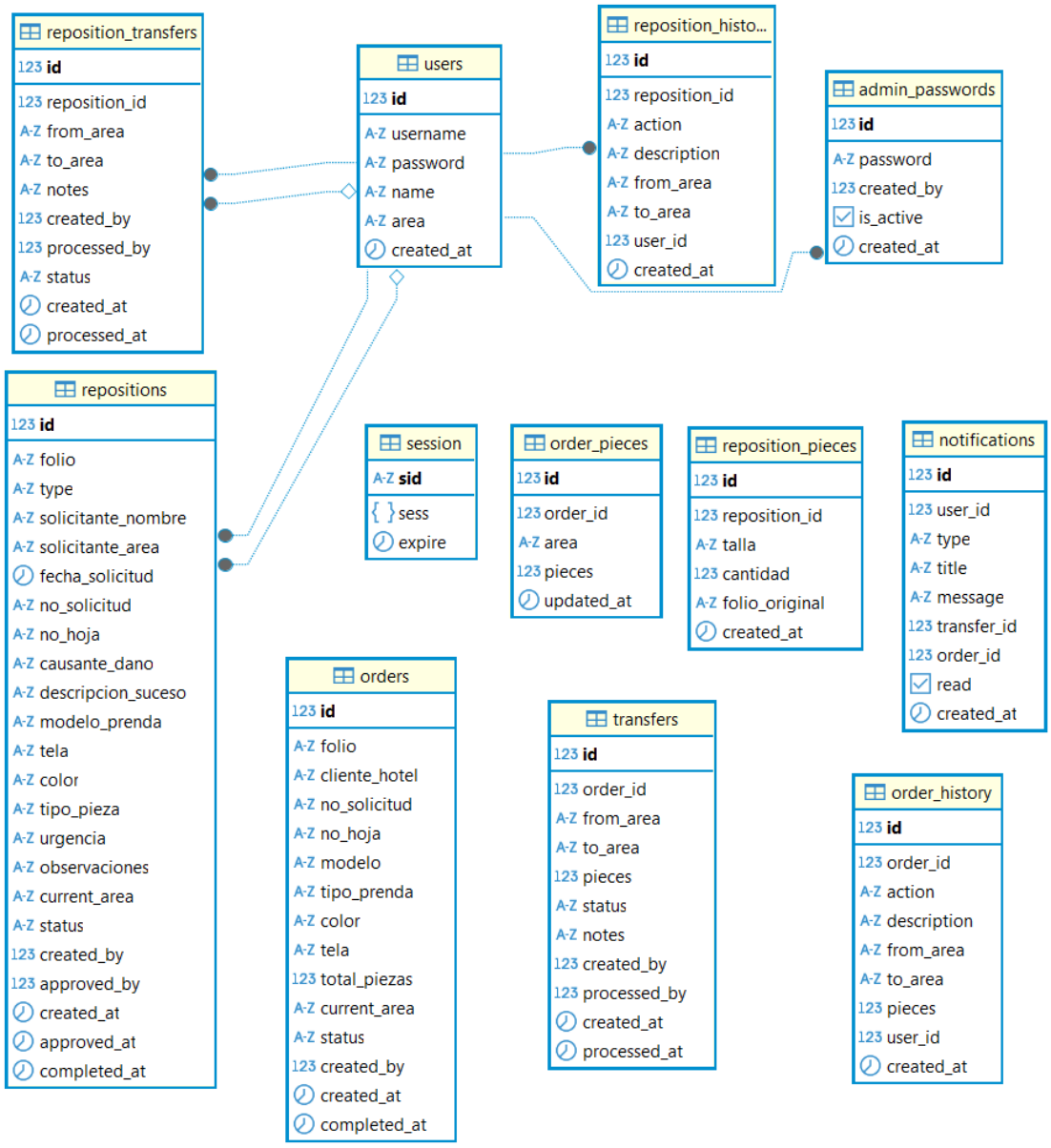


Figura 5.7: Diagrama E-R de la base de datos para el sistema de reprocesos y reposiciones generado a través del software Dbeaver

## 5.3. Implementación

La fase de implementación consiste en la traducción del análisis y diseño a código funcional. Esta sección detalla el proceso de construcción de la herramienta, utilizando el conjunto de tecnologías definido en el marco teórico: React para el frontend, Node.js para el backend

y Vite como entorno de desarrollo. Para una mayor claridad, la exposición del trabajo se ha organizado por las funcionalidades clave que se implementaron.

### 5.3.1. Tecnologías y Entorno de Desarrollo

La implementación del proyecto se realizó utilizando el siguiente conjunto de tecnologías y herramientas:

- **Frontend:** React.js (v18.2), Vite.
- **Backend:** Node.js (v20.10), Express.js.
- **Base de Datos:** PostgreSQL (v16).
- **Control de Versiones:** Git, GitHub.
- **Editor de Código:** Visual Studio Code.

### 5.3.2. Evolución de la Interfaz Principal (Dashboard)

El componente principal de la aplicación es el Dashboard o Tablero, ya que funciona como el punto central de control para los usuarios. Su diseño no fue estático, sino que evolucionó a través de un proceso de desarrollo iterativo para mejorar la experiencia de usuario y alinear la interfaz con los requerimientos funcionales. A continuación, se muestra esta progresión.

#### Versión 1.0: Estructura Inicial

La primera versión (Figura 5.8) se enfocó en establecer la funcionalidad básica: una lista de pedidos con capacidades de búsqueda. El diseño era minimalista, priorizando la correcta visualización de los datos tabulados y la estructura de navegación lateral.

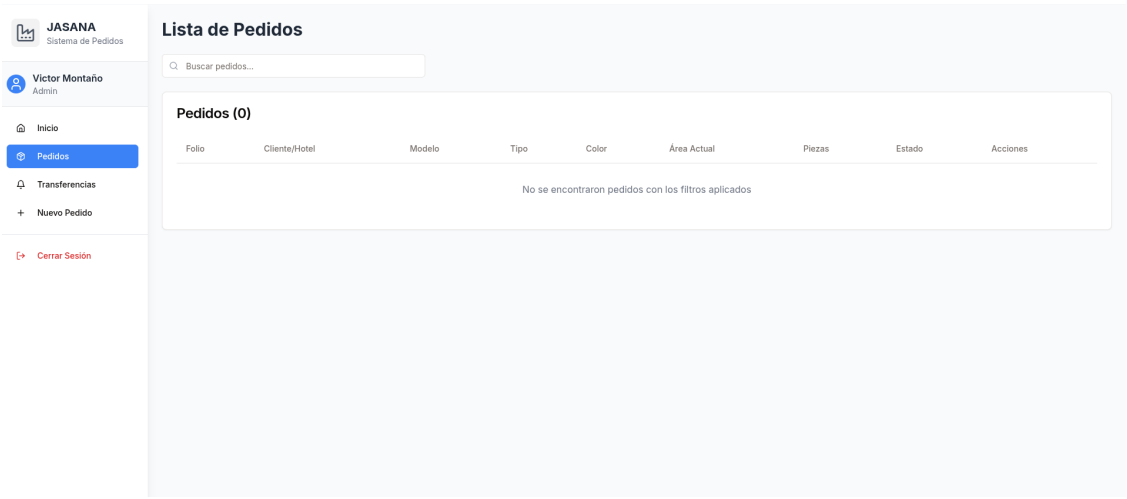


Figura 5.8: Versión 1.0 de la interfaz principal.

### Versión 2.0: Refinamiento de la Navegación

En la segunda iteración (Figura 5.9), se refinó la barra de navegación lateral, agrupando las opciones de manera más lógica (‘Inicio‘, ‘Pedidos‘, ‘Reposiciones‘) y añadiendo un botón de "Nuevo Pedido"para agilizar el flujo de trabajo del usuario. El título se actualizó a "Sistema de Pedidos"para ser más descriptivo.

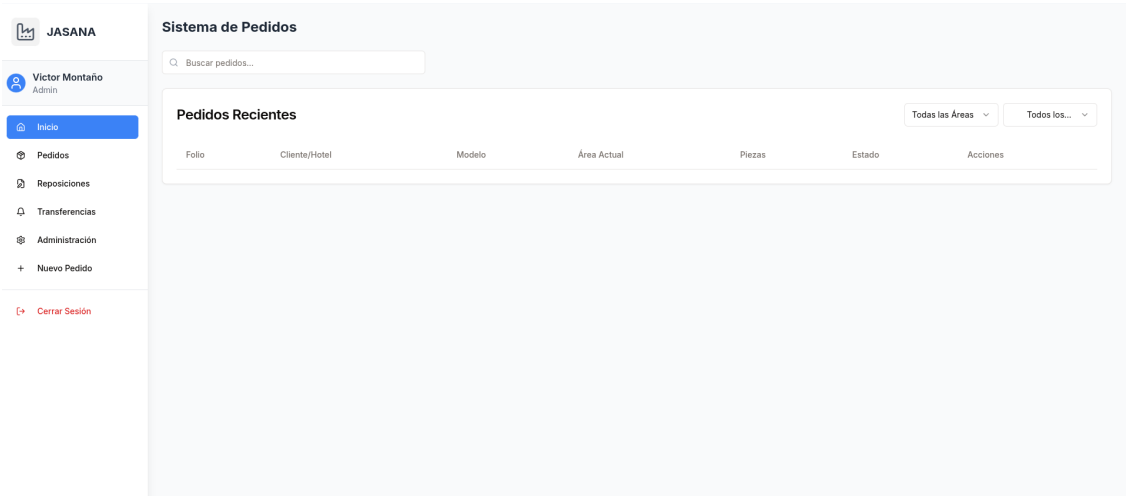


Figura 5.9: Versión 2.0 con mejoras en la navegación.

### Versión 3.0: Enfoque Gerencial

La versión 3.0 (Figura 5.10) representó un cambio de enfoque hacia una herramienta más gerencial, renombrando la aplicación a Centro de Control Empresarial. Se añadieron nuevas secciones en el menú como Reportes e Historial, anticipando la necesidad de funcionalidades de análisis y consulta.

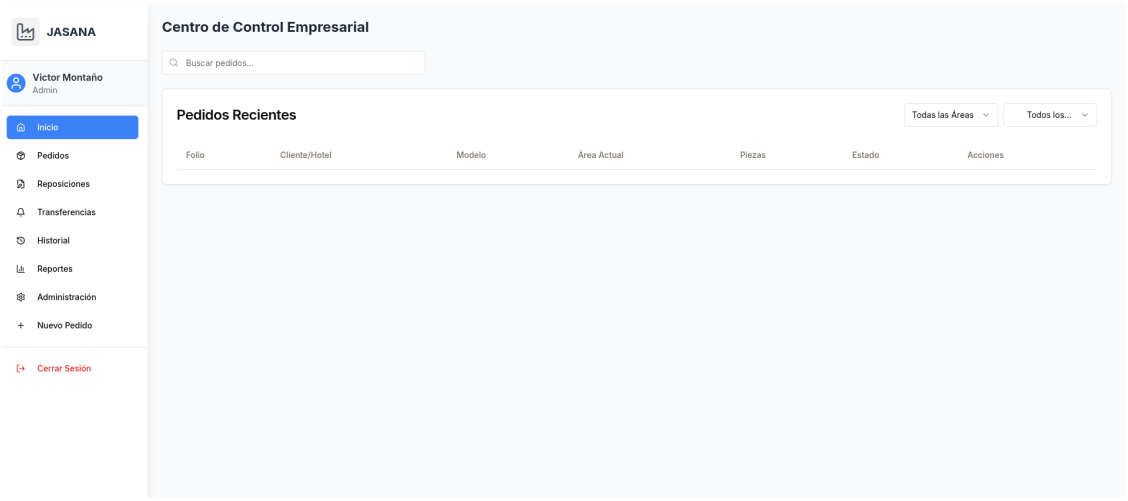


Figura 5.10: Versión 3.0 con enfoque en control y reportes.

### Versión 4.0: Tablero de Control

La versión final (Figura 5.11) consolida la evolución en un "Tableroçompleto. La innovación principal es la adición de tarjetas de indicadores clave de rendimiento en la parte superior para una visualización rápida del estado del sistema ("Pedidos Activos", "Finalizados Hoy", etc.). Además, la sección de actividad reciente ahora diferencia entre "Pedidosz Reposiciones", ofreciendo una vista más granular. Este diseño final responde a la necesidad de tener tanto una vista operativa detallada como una vista gerencial resumida en una sola pantalla.

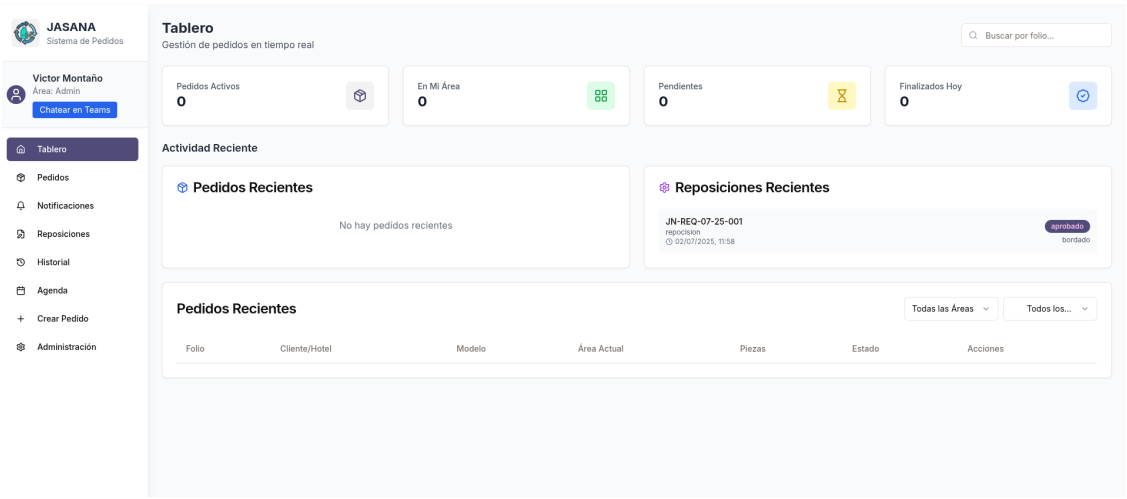


Figura 5.11: Versión final 4.0, implementada como un Tablero de Control y un logo añadido.

Este proceso iterativo fue fundamental para refinar la aplicación.

## 5.4. Pruebas

# Capítulo 6

# Resultados y Conclusiones

## 6.1. Resultados

## 6.2. Conclusiones

# Bibliografía


Correal Rodríguez, N. A. (2021). Las contribuciones de la erp de una empresa de proyectos de infraestructura para la toma de decisiones gerenciales. Master's thesis, Universidad EAFIT, Medellín.

Dura, C. C., Drigă, I., and Iordache, A. M. M. (2022). Software-as-a-service programs and project management: A case study on odoo erp. 373:00037.

Evan You and Vite Contributors (2025). Vite documentation. `https://vitejs.dev/`. Consultado el 2 de julio de 2025.

Hammouch, H. (2024). Enhancing management control through erp systems: A comprehensive literature review. *iRASD Journal of Management*, 6(3):125–133.

Meta and the React Team (2025). React documentation. `https://react.dev/`. Consultado el 2 de julio de 2025.

Miño-Cascante, M. G., Saumell-Fonseca, L. E., Toledo-Borrego, M. A., Roldan-Ruenes, D. A., and Moreno-García, D. R. R. (2015). Planeación de requerimientos de materiales por el sistema mrp. caso laboratorio farmacéutico oriente. cuba. *Chemical Technology*, 35(2):248–260.

Monk, E. and Wagner, B. (2023). *Concepts in Enterprise Resource Planning*.

Murgueytio, F. M., Galarza, P. J., and Barrientos, A. (2022). Proceso de automatización de pruebas de aplicaciones web desarrolladas con react, angular, ant y laravel. In *Memorias de la Vigésima Primera Conferencia Iberoamericana en Sistemas, Cibernética e Informática (CISCI 2022)*, pages 192–197.

Odoo S.A. (2024). Odoo 18.0 developer documentation. https://www.odoo.com/documentation/18.0/index.html. Consultado el 9 de junio de 2025.

OpenJS Foundation (2025). Node.js documentation. https://nodejs.org/en/docs. Consultado el 2 de julio de 2025.

Pallets Projects (2024). Werkzeug documentation. https://werkzeug.palletsprojects.com/en/stable/. Consultado el 5 de junio de 2025.

Pretell Cruzado, R. J. (2024). Propuesta de mejora para el proceso de gestión docente de la universidad nacional autónoma de chota, 2024. Trabajo de investigación de maestría, Escuela de Posgrado Newman, Tacna, Perú.

Python Software Foundation (2025). The python language reference. https://docs.python.org/3/reference/index.html. Consultado el 6 de junio de 2025.

Reis, D. and Mader, G. (2022). *Odoo 15 Development Essentials: Enhance your Odoo development skills to create powerful business applications*. Packt Publishing.

Taipe Toapaxi, A. I. and Quishpe Caizatoa, B. A. (2022). Desarrollo de un sistema de gestión veterinaria, mediante el modelo api rest y el framework reactjs como herramientas de software libre para el consultorio visecpro del cantón latacunga. Proyecto tecnológico de titulación, Universidad Técnica de Cotopaxi, Latacunga, Ecuador.

The PostgreSQL Global Development Group (2025). Postgresql 16 documentation. https://www.postgresql.org/docs/16/index.html. Consultado el 9 de junio de 2025.

Tort Carrillo, M. (2024). Desenvolupament dúna Aplicació en el sistema de codi obert Odoo para la Gestió de Residències. Treball de fi de grau, Escola Tècnica Superior dÉnginyeria Industrial de Barcelona (ETSEIB), Barcelona, España. Director: Manuel Moreno Eguílaz.

# Apéndice A

# Glosario

**API (Interfaz de Programación de Aplicaciones)** Conjunto de reglas y herramientas que permiten que diferentes aplicaciones de software se comuniquen entre sí, intercambiando datos e información de manera estandarizada.

**Arquitectura Desacoplada (Headless)** Patrón de diseño de software donde la capa de presentación (frontend) es completamente independiente de la capa de lógica de negocio y gestión de datos (backend). Ambas capas se comunican a través de una API.

**Backend** La parte de una aplicación de software que se ejecuta en el servidor y es invisible para el usuario final. Es responsable de la lógica de negocio, el acceso a la base de datos y la comunicación con el servidor.

**Caso de Uso** Técnica de análisis que describe una secuencia de interacciones entre un actor (usuario) y un sistema para lograr un objetivo específico. Se utiliza para definir los requerimientos funcionales de un sistema.

**Componente (React)** Pieza de código independiente y reutilizable que encapsula la lógica y la interfaz de usuario (UI) de una sección de la aplicación. Son los bloques de construcción fundamentales en React.

**Código Abierto (Open Source)** Modelo de desarrollo de software que se caracteriza por proveer un código fuente que cualquiera puede ver, usar, modificar y distribuir de manera gratuita.

**Endpoint (Punto Final)** URL específica dentro de una API a la que una aplicación cliente envía una solicitud para acceder a un recurso o ejecutar una operación concreta en el servidor.

**ERP(Enterprise Resource Planning)** Sistema integrado que permite coordinar y optimizar procesos empresariales clave, como finanzas, recursos humanos, logística y producción.

**Frontend** La parte de una aplicación de software con la que el usuario interactúa directamente. También conocida como la çapa de presentación.º ïnterfaz de usuario (UI)", se encarga de mostrar los datos y capturar las entradas del usuario.

**HMR (Hot Module Replacement)** Funcionalidad de herramientas de desarrollo como Vite que permite actualizar los módulos de una aplicación en el navegador en tiempo real, sin necesidad de recargar la página completa, agilizando el proceso de desarrollo.

**Node.js** Entorno de ejecución de JavaScript del lado del servidor, diseñado para construir aplicaciones de red escalables y rápidas. Se utiliza comúnmente para crear el backend y las APIs de las aplicaciones web.

**Odoo** Sistema ERP de código abierto y modular, utilizado para la gestión de procesos de negocio. En el contexto de esta tesis, es el sistema principal de la empresa.

**ORM (Mapeo Objeto-Relacional)** Técnica de programación que convierte los datos entre una base de datos relacional (como PostgreSQL) y un lenguaje de programación orientado a objetos (como Python). Permite al desarrollador interactuar con la base de datos usando objetos y clases en lugar de escribir consultas SQL directamente.

**PostgreSQL** Sistema de gestión de bases de datos relacional de código abierto, conocido por su robustez, escalabilidad y cumplimiento de estándares SQL.

**React** Librería de JavaScript de código abierto utilizada para construir interfaces de usuario interactivas y dinámicas, especialmente para Aplicaciones de Una Sola Página (SPA).

**Requerimiento Funcional** Describe una acción o funcionalidad específica que el sistema debe ser capaz de realizar (el "qué"del sistema).

**Requerimiento No Funcional** Describe las cualidades o restricciones del sistema, como el rendimiento, la seguridad o la usabilidad (el ”como” o "qué tan bien"funciona el sistema).

**SaaS (Software como Servicio)** Modelo de distribución de software donde las aplicaciones están alojadas en la nube por un proveedor y los usuarios acceden a ellas a través de internet, generalmente mediante una suscripción.

**Silos de Información** Metáfora usada para describir sistemas de información que operan de forma aislada dentro de diferentes departamentos de una empresa, sin compartir datos entre sí, lo que genera ineficiencia y falta de coordinación.

**Stack Tecnológico (Tech Stack)** El conjunto específico de tecnologías, lenguajes de programación, frameworks y herramientas que se utilizan en conjunto para construir y operar una aplicación de software.

**Trazabilidad** Capacidad de seguir y documentar el historial, la ubicación o la aplicación de un ítem a través de todas sus etapas.

**Virtual DOM (VDOM)** Una representación del DOM (la estructura de la página web) guardada en memoria que utiliza React. Permite optimizar el rendimiento al calcular los cambios mínimos necesarios antes de actualizar la interfaz real.

**Vite** Herramienta de construcción de frontend moderna y de alto rendimiento que se utiliza para empaquetar y servir aplicaciones web.