UNIVERSIDAD TECNOLÓGICA

DE LA ZONA METROPOLITANA DE GUADALAJARA

MEMORIA TÉCNICA REALIZADA EN:

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

PROYECTO: Sistema de Gestión de Reprocesos y Pedidos

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

Sistema de Gestión de Reprocesos y Pedidos

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

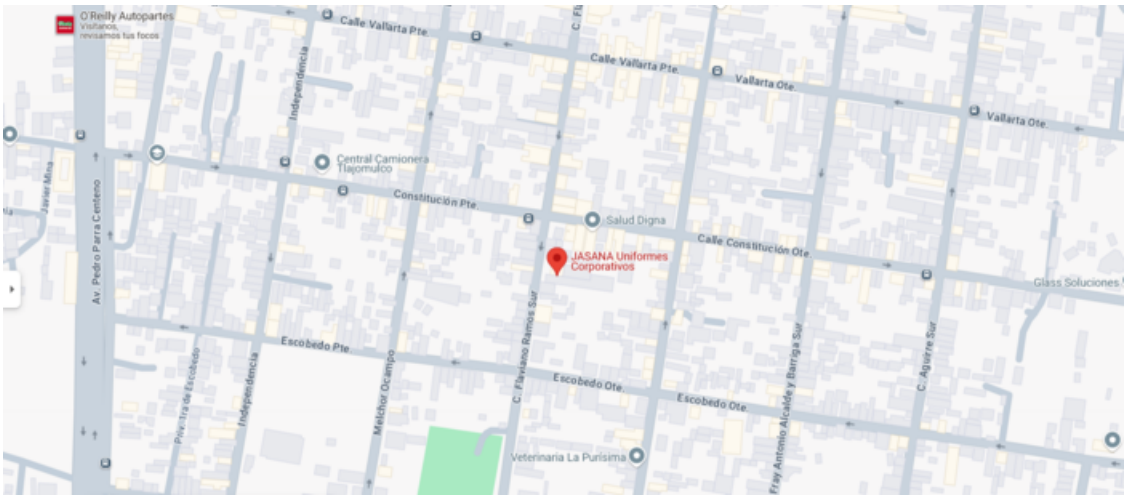

Figura 2.1: Ubicación de la empresa

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.

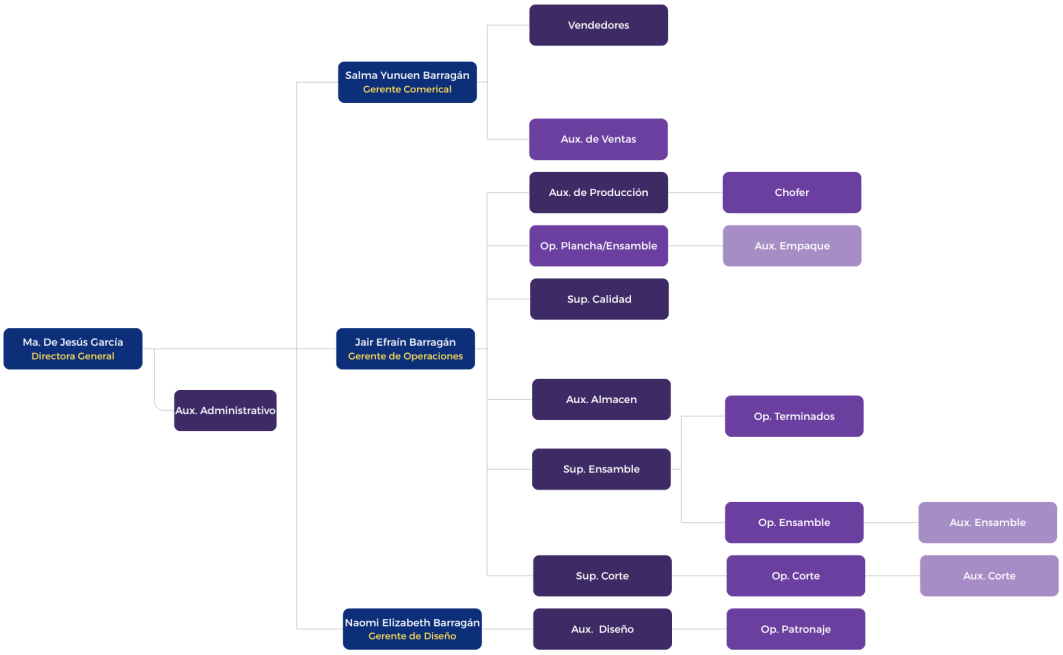

Figura 2.2: Organigrama de Textil JSN

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

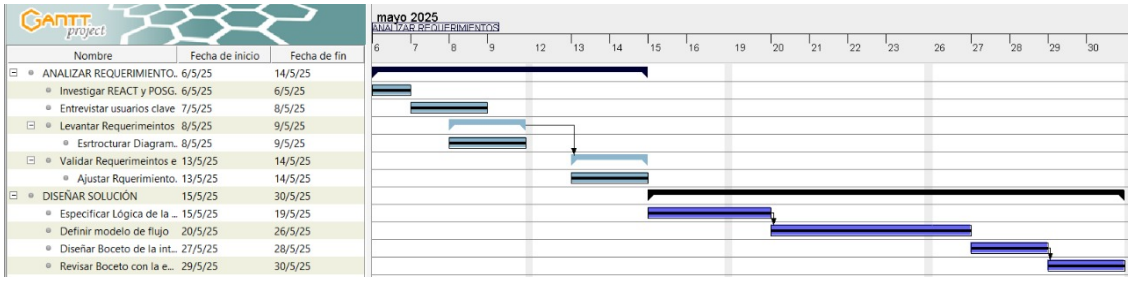

Figura 3.1: Periodo correspondientes a Mayo

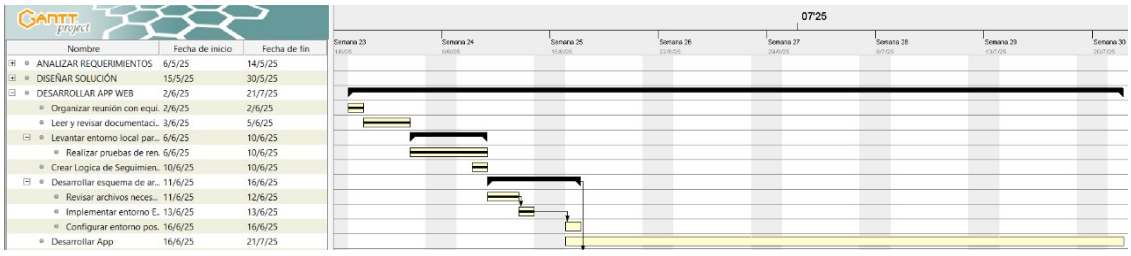

Figura 3.2: Periodo correspondiente a Junio-Julio

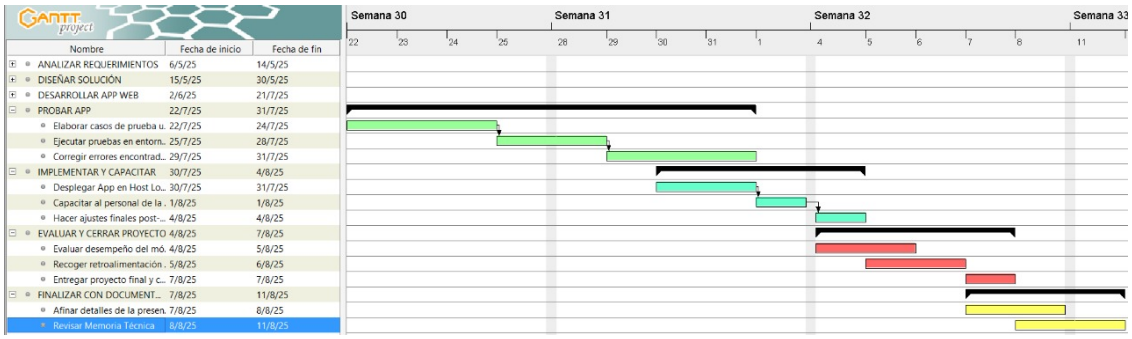

Figura 3.3: Periodo correspondiente a Julio-Agosto

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 la solución desarrollada, justificando las decisiones técnicas y metodológicas que se tomaron.

El capítulo se enfoca en 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. Se inicia con una breve contextualización del entorno empresarial basado en sistemas ERP para fundamentar la pertinencia del proyecto.

4.1. Contexto y justificación de la arquitectura

La problemática que aborda este proyecto surge dentro de una empresa cuya gestión operativa se apoya en un sistema de **Planificación de Recursos Empresariales (ERP)** bajo un modelo de **Software como Servicio (SaaS)**. Un ERP es una plataforma de software integrado que busca optimizar y centralizar los procesos de negocio de una empresa.

Aunque estos sistemas son potentes, su naturaleza monolítica puede presentar limitaciones. 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 presentar una notable falta de integración con herramientas modernas.
- **Procesamiento de datos externo:** Esta falta de integración obliga a los usuarios a recurrir a herramientas externas como Excel.
- **Rigidez ante el cambio:** Se requieren herramientas más dinámicas que se adapten rápidamente a las transformaciones de la empresa.

Estas limitaciones,fundamentan la decisión estratégica de optar por una arquitectura desacoplada para este proyecto.

4.2. Desarrollo frontend: ecosistema React

La elección de una tecnología para la interfaz de usuario (frontend) es un factor determinante en la usabilidad y el rendimiento. Para este proyecto se seleccionó **React**, una de las librerías de JavaScript de código abierto más influyentes para la construcción de interfaces interactivas y reutilizables (Murgueytio et al., 2022).

4.2.1. 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. 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, 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).

4.2.2. El Virtual DOM y su impacto en el rendimiento

Una de las innovaciones técnicas 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. Finalmente, este proceso de **reconciliación** aplica únicamente dichas diferencias al DOM real, optimizando drásticamente el rendimiento de la aplicación (Taipe Toapaxi and Quishpe Caizatoa, 2022; Meta and the React Team, 2025).

4.2.3. Gestión del estado: de Hooks a gestores globales

En **React**, el **estado (state)** contiene los datos que describen la condición de un componente. Para gestionar este estado, se introdujeron los **Hooks**, funciones que permiten "engancharse" a las características de **React** desde componentes funcionales. Los más esenciales son useState y useEffect. Para aplicaciones más complejas, el ecosistema ofrece soluciones de gestión de estado global, como la **API Context** nativa o librerías como **Redux**, que proporcionan un almacén centralizado para el estado de toda la aplicación (Meta and the React Team, 2025).

4.2.4. Ecosistema y herramientas de construcción: Vite

El desarrollo moderno depende de **herramientas de construcción** (*build tools*). **Vite** es una herramienta de nueva generación que busca proporcionar una experiencia de desarrollo notablemente más rápida. Su principal ventaja es que aprovecha los **módulos ES nativos (ESM)** del navegador durante el desarrollo. Para la fase de producción, **Vite** utiliza internamente el empaquetador Rollup para generar un paquete de código altamente optimizado (Evan You and Vite Contributors, 2025).

4.3. Arquitectura de la solución: Backend desacoplado

Frente al modelo monolítico de un ERP, una arquitectura desacoplada separa completamente el frontend del backend. 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" (Taipe Toapaxi and Quishpe Caizatoa, 2022).

4.3.1. La API como contrato y el reto de CORS

La API define un **contrato** con un conjunto de reglas y **endpoints** que el frontend utiliza para solicitar o enviar información. Este proyecto utiliza una **API REST**, un estilo de arquitectura que usa los verbos del protocolo HTTP y el formato JSON para el intercambio de datos. Un aspecto técnico indispensable en esta arquitectura es el **Intercambio de Recursos de Origen Cruzado (CORS)**. Por defecto, los navegadores web aplican una "política del mismo origen" que restringe las peticiones HTTP a un dominio diferente. Dado que el frontend y el backend operan en orígenes distintos, es crucial que el servidor implemente cabeceras CORS para indicar al navegador que las peticiones desde el frontend están permitidas y son seguras (MDN Web Docs, 2025).

4.3.2. Node.js y Express 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). Para facilitar la creación de la API REST, se utilizó **Express.js**, un framework minimalista y flexible para Node.js. Express simplifica enormemente el manejo de rutas (endpoints), la gestión de peticiones HTTP y la configuración de *middleware* (como el que gestiona CORS), permitiendo construir el backend de manera rápida y organizada (Express.js and OpenJS Foundation contributors, 2025).

4.3.3. Gestión de la base de datos: Drizzle ORM

Para la comunicación entre el backend y la base de datos PostgreSQL, se utilizó **Drizzle ORM**. Un ORM (Mapeo Objeto-Relacional) es una técnica que permite interactuar con la base de datos usando objetos y clases de un lenguaje de programación, en este caso TypeScript. Drizzle es un ORM "headless" o sin cabeza, diseñado para ser ligero, rápido y, sobre todo, totalmente seguro en cuanto a tipos (*typesafe*). A diferencia de otros ORMs, no abstrae por completo el lenguaje SQL, sino que proporciona un constructor de consultas que refleja la sintaxis de SQL, dando al desarrollador el control total sobre las consultas que se ejecutan y garantizando que los resultados sean compatibles con los tipos definidos en TypeScript (Drizzle Team, 2025).

4.3.4. Universalidad de las Convenciones: El Caso de Scala

La adopción de una convención de nomenclatura no es exclusiva del ecosistema de JavaScript o TypeScript, sino una práctica fundamental en la ingeniería de software en general. Para ilustrar su universalidad, se puede tomar como referencia la guía de estilo oficial del lenguaje de programación Scala, una tecnología robusta utilizada para construir sistemas de alto rendimiento.

En su documentación, se especifica una clara distinción en el uso de las capitales: se debe utilizar **UpperCamelCase** para nombrar clases y *traits* (un concepto similar a las interfaces), mientras que para nombrar métodos y valores (variables y constantes) se debe utilizar **lowerCamelCase**. Esta distinción sistemática es crucial, ya que permite a cualquier programador identificar de un vistazo la naturaleza de un identificador dentro del código, mejorando drásticamente la legibilidad y reduciendo la carga cognitiva (Scala Lang Team, 2025).

Este ejemplo demuestra que, a pesar de las diferencias sintácticas entre lenguajes, la necesidad de un código limpio y consistente lleva a la adopción de convenciones compartidas, validando la decisión de seguir un estándar estricto en este proyecto.

Capítulo 5

Desarrollo del Proyecto

5.1. Requerimientos

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.	

5.2. 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.

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 relacional

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 5.1) ilustra la estructura diseñada, mostrando las tablas principales y las relaciones que garantizan la integridad de los datos.

Figura 5.1: Diagrama relacional de la base de datos para el sistema de reprocesos y reposiciones generado a través del software Dbeaver

5.2.3. Casos de uso

A continuación se presenta los casos de uso junto a su respectivo diagrama (figuras 5.2 a 5.7) 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: Finalición 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.

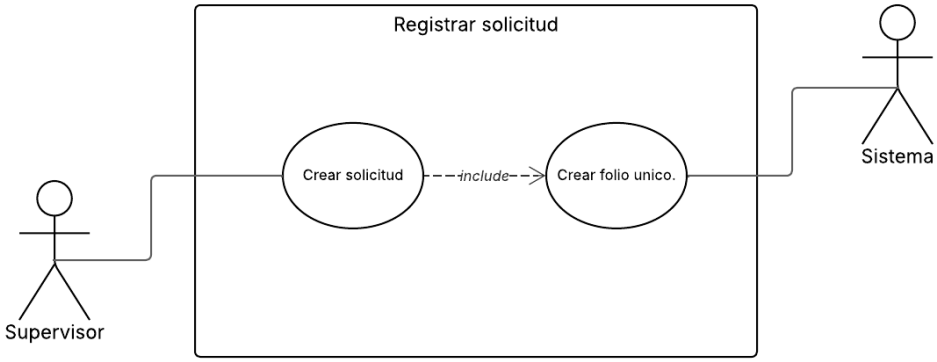

Figura 5.2: Caso de Uso 001

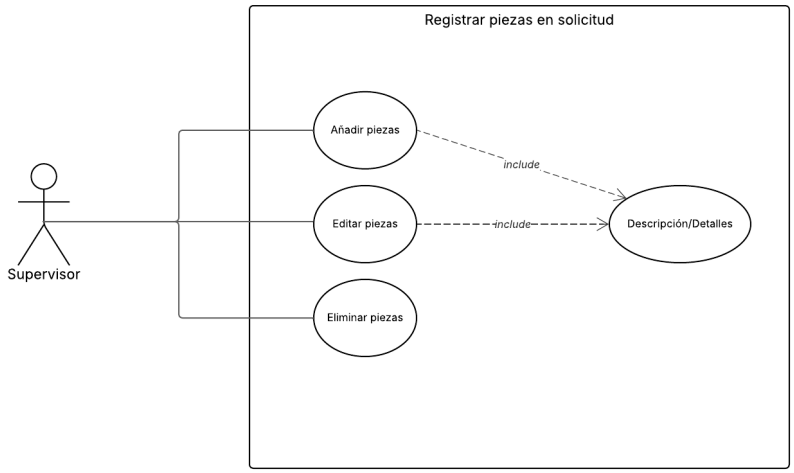

Figura 5.3: Caso de Uso 002

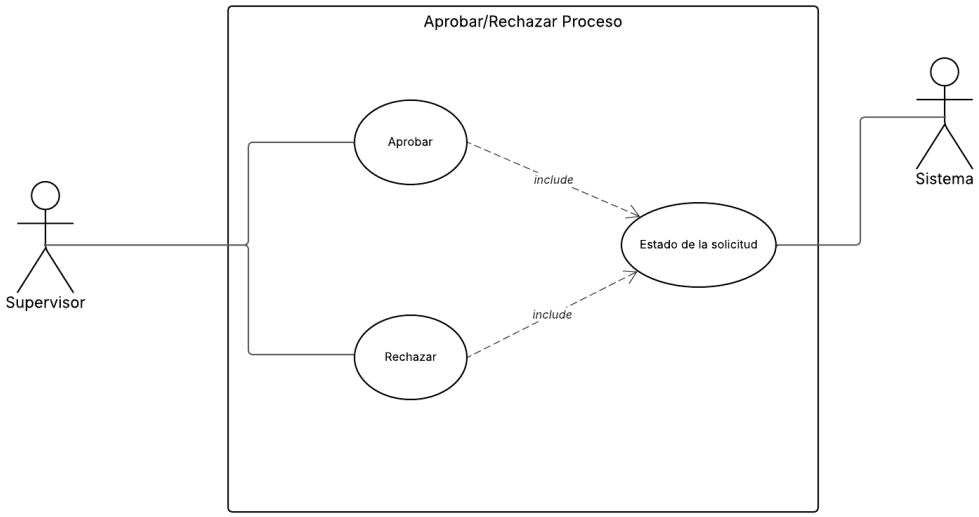

Figura 5.4: Caso de Uso 003

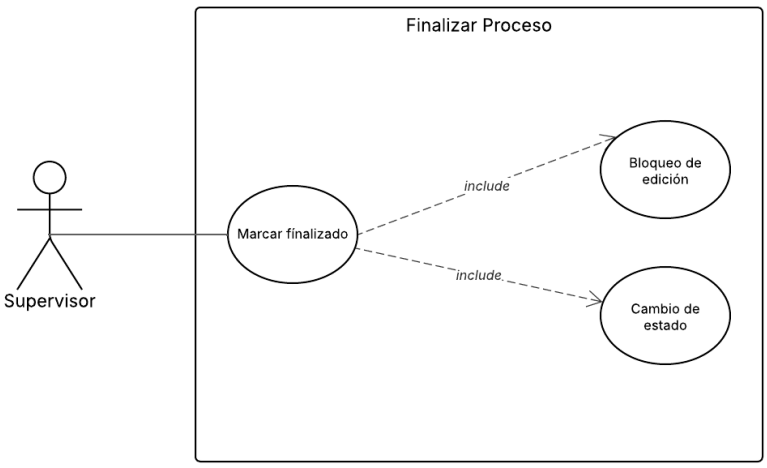

Figura 5.5: Caso de Uso 004

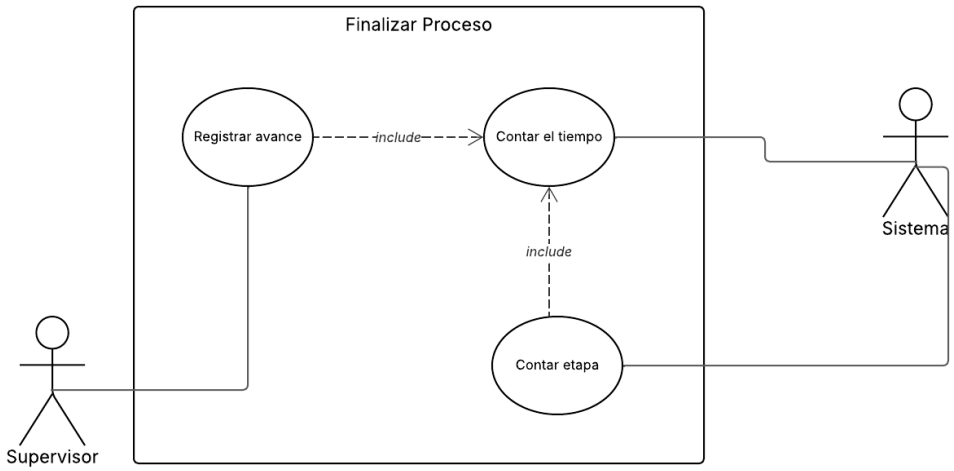

Figura 5.6: Caso de Uso 005

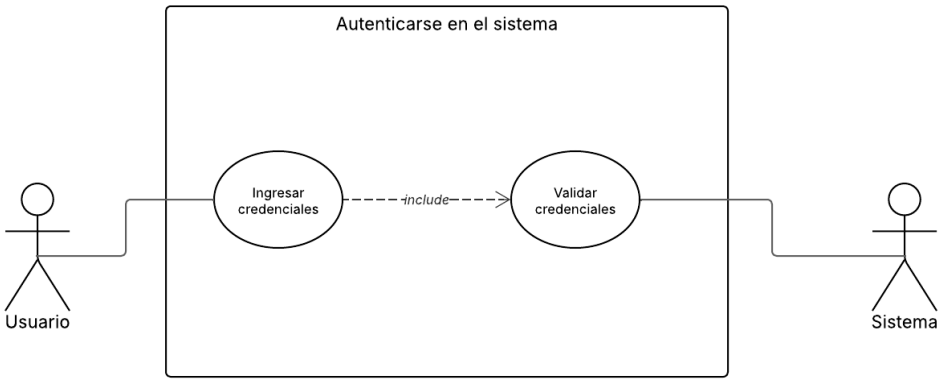

Figura 5.7: Caso de Uso 006

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.
- **Alojamiento:** Linux Mint 21.3 , Cinnamon 6.

El desarrollo y despliegue del proyecto se llevó a cabo sobre el sistema operativo **Linux Mint 21.3**, utilizando el entorno de escritorio **Cinnamon**. Esta distribución, basada en Ubuntu, fue seleccionada por su gran estabilidad, su compatibilidad con hardware diverso y por ofrecer un entorno de trabajo intuitivo y eficiente, similar en su uso a sistemas operativos tradicionales pero con toda la potencia y flexibilidad del ecosistema GNU/Linux

5.3.2. Estándares y Convenciones de Codificación

Para garantizar la legibilidad, consistencia y mantenibilidad del código, es una práctica estándar en el desarrollo de software adoptar una **convención de nomenclatura**. Esta consiste en un conjunto de reglas para nombrar los distintos identificadores (variables, funciones, clases, etc.) dentro de un proyecto.

En el ecosistema de JavaScript y, por extensión, de **TypeScript**, la convención más extendida y recomendada es **camelCase**. Esta práctica consiste en escribir los identificadores uniendo palabras sin espacios, donde la primera palabra comienza con minúscula y cada palabra subsecuente comienza con una letra mayúscula (p. ej., `nombreDeVariable`).

De forma complementaria, para los elementos que definen un tipo o una estructura, como las **clases** e **interfaces**, se utiliza la convención **PascalCase**, donde la primera letra

de cada palabra, incluida la primera, va en mayúscula (p. ej., `class GestorDePedidos`). El seguimiento de estas convenciones es una señal de código limpio y es recomendado en guías de estilo influyentes como las de Google y la comunidad de desarrolladores de MDN (**??**).

======EJEMPLO DE CÓDIGO=====

5.3.3. 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.

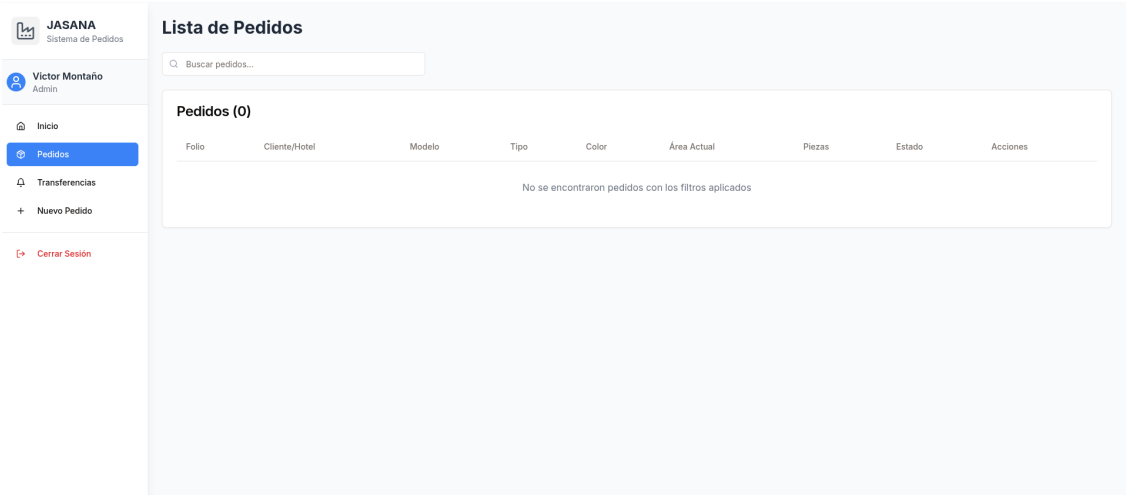

Figura 5.8: Versión 1.0 de la interfaz principal.

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.

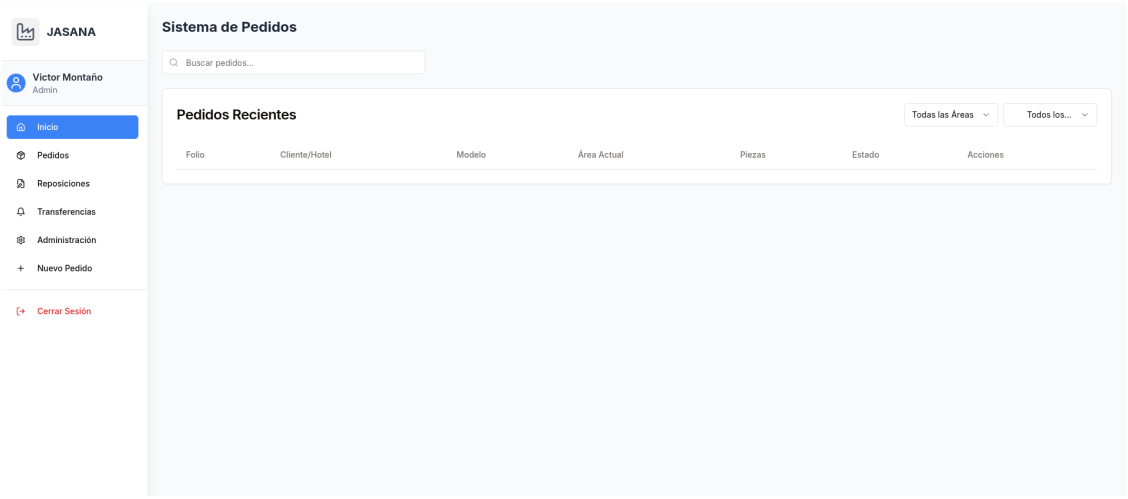

Figura 5.9: Versión 2.0 con mejoras en la navegación.

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.

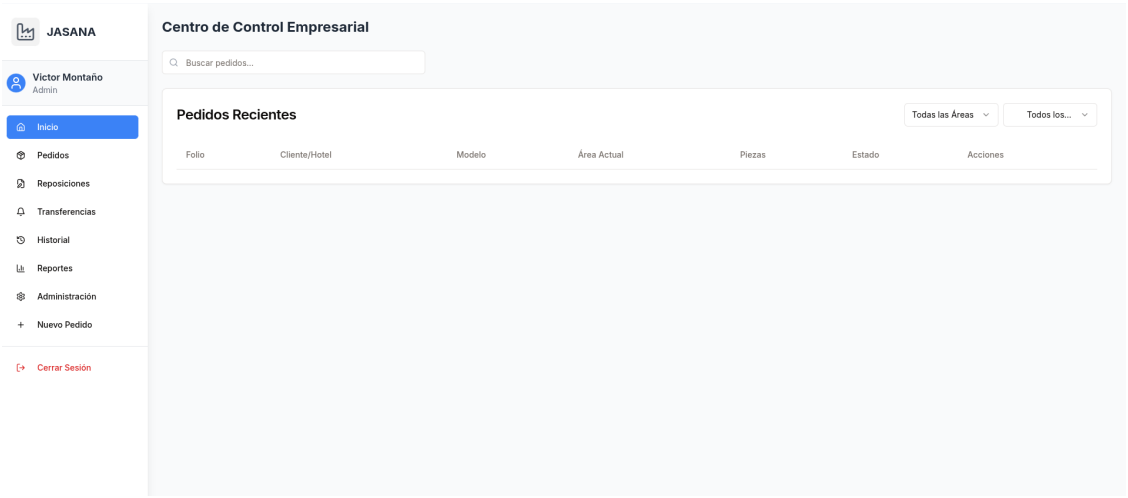

Figura 5.10: Versión 3.0 con enfoque en control y reportes.

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.

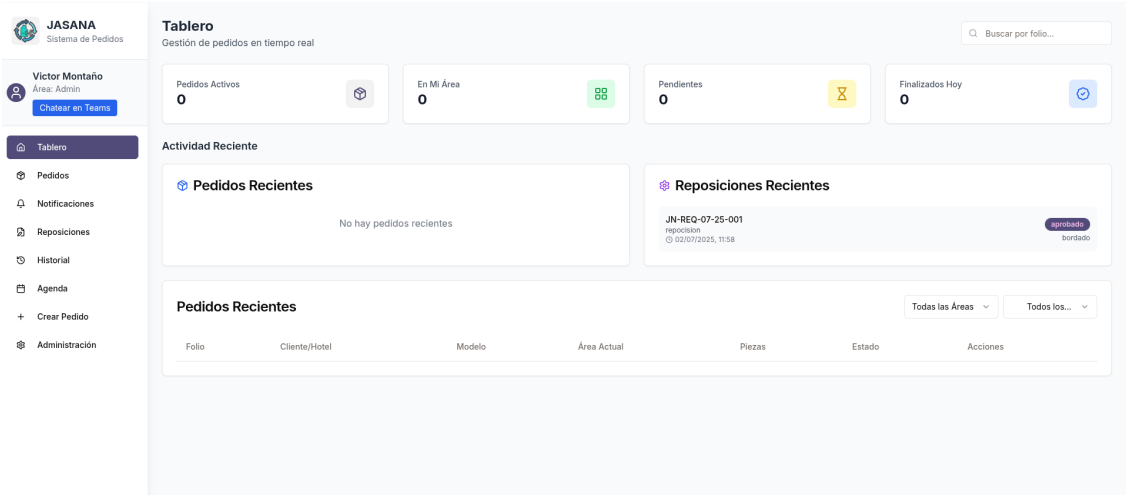

Figura 5.11: Versión final 4.0, implementada como un Tablero de Control y un logo añadido.

Este proceso iterativo fue fundamental para refinar la aplicación.

5.4. Pruebas

Como parte fundamental del proceso final del desarrollo de esta aplicación, se plantearon diferentes tipos de pruebas para corroborar la funcionalidad, fluidez, y calidad general del sistema, sobre los resultados arrojados por estas mismas se decidió reajustar y realizar correcciones y posibles mejoras para una implementación que cumpla con los requerimientos previamente levantados y así entregar un resultado con la mayor calidad posible. En las próximas imágenes se puede encontrar con más detalle cada caso de prueba ejecutado junto

a sus resultados y acciones tomadas.

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.

Drizzle Team (2025). Drizzle orm documentation - overview. `https://orm.drizzle.team/docs/overview`. Consultado el 9 de julio de 2025.

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.

Express.js and OpenJS Foundation contributors (2025). Express - node.js web application framework. `https://expressjs.com/`. Consultado el 9 de julio de 2025.

MDN Web Docs (2025). Guía cors (intercambio de recursos de origen cruzado). `https://developer.mozilla.org/es/docs/Web/HTTP/Guides/CORS`. Consultado el 9 de julio de 2025.

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

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.

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.

Scala Lang Team (2025). Style guide: Naming conventions. `https://docs.scala-lang.org/style/naming-conventions.html`. Consultado el 14 de julio de 2025.

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.

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.