DISEÑO TÉCNICO Y PROPUESTA TECNOLÓGICA

Distribuciones Sierra Norte S.L.

Claudia Nikol Lazarte Diaz

NTT DATA Soluciones IA

Contenido

1. Introducción

El presente documento de **Diseño Técnico y Propuesta Tecnológica** define la solución técnica propuesta para el proyecto de mejora de la gestión del proceso de pedidos y su seguimiento en **Distribuciones Sierra Norte S.L.**. Este documento traduce los requisitos funcionales y no funcionales previamente definidos en una arquitectura técnica concreta, estableciendo las bases para el desarrollo, despliegue y mantenimiento del sistema.

La solución propuesta tiene como objetivo principal proporcionar una **plataforma web centralizada** que permita gestionar de forma estructurada y coherente la información clave del negocio, incluyendo **pedidos, clientes y productos**, y facilitar su uso por parte de los distintos perfiles internos de la empresa (dirección, equipo comercial, almacén, reparto y administración).

El diseño técnico tiene en cuenta el contexto real de la organización, caracterizado por:

- Un volumen medio de usuarios internos.
- Procesos actualmente manuales y descentralizados.
- Presupuesto limitado.
- Necesidad de no interrumpir la actividad diaria durante la implantación.
- Usuarios con bajo nivel de formación técnica.

Asimismo, el documento establece una propuesta tecnológica alineada con estos condicionantes, basada en tecnologías estándar y ampliamente adoptadas, que permita una **implantación progresiva**, una **evolución incremental del sistema** y un **mantenimiento sencillo** a medio y largo plazo.

1.1 Documentos de referencia

Este documento se ha elaborado tomando como base los siguientes documentos, que deben considerarse complementarios y de lectura obligatoria para una correcta comprensión de la solución:

- **Análisis Funcional y de Requisitos**
- **Historias de Usuario**

Estos documentos definen el contexto del negocio, la problemática actual, los requisitos funcionales y no funcionales, y las necesidades de los distintos perfiles de usuario, sobre los que se apoya el diseño técnico descrito en las secciones posteriores.

2. Visión general de la solución

2.1 Descripción general de la solución propuesta

La solución propuesta consiste en el desarrollo de un **sistema web de gestión interna**, diseñado para centralizar y estructurar la información relacionada con los **pedidos, clientes y productos** de Distribuciones Sierra Norte S.L., y para proporcionar visibilidad y control del estado de los pedidos a lo largo de todo su ciclo de vida.

El sistema permitirá a los distintos departamentos trabajar sobre una **fuente única de información**, reduciendo duplicidades, errores y tiempos administrativos, y mejorando la coordinación entre las áreas comercial, almacén, reparto y administración. La solución está concebida como una aplicación web accesible mediante navegador, sin necesidad de instalaciones locales, y con una interfaz sencilla adaptada a usuarios no técnicos.

Desde el punto de vista técnico, la solución se apoya en una **arquitectura en capas**, que separa claramente la presentación, la lógica de negocio y la persistencia de datos, y en un conjunto de tecnologías robustas y consolidadas que facilitan su mantenimiento y evolución futura.

2.2 Objetivos técnicos de la solución

La solución técnica propuesta persigue los siguientes objetivos:

- **Centralizar la información del negocio**, proporcionando un repositorio único para pedidos, clientes y productos.
- **Garantizar la coherencia e integridad de los datos**, evitando duplicidades y pérdidas de información.
- **Aplicar un control de acceso por rol**, asegurando que cada usuario solo pueda acceder a las secciones y operaciones que le correspondan.
- **Facilitar la mantenibilidad y evolución del sistema**, mediante una arquitectura modular basada en dominios funcionales (pedidos, clientes y productos).
- **Soportar operaciones CRUD de forma eficiente**, incluso con múltiples usuarios concurrentes.
- **Permitir una implantación progresiva**, minimizando el impacto en la operativa diaria de la empresa.
- **Utilizar tecnologías estándar y consolidadas**, reduciendo riesgos técnicos y dependencias innecesarias.
- **Preparar la solución para futuras ampliaciones funcionales**, sin necesidad de rediseños estructurales.

2.3 Principios de diseño

El diseño de la solución se basa en los siguientes principios técnicos fundamentales:

Separación de responsabilidades

La arquitectura separa claramente la capa de presentación, la lógica de negocio y la capa de datos, así como los distintos dominios funcionales del sistema (pedidos, clientes y productos).

Simplicidad y claridad

El diseño prioriza soluciones técnicas sencillas, comprensibles y alineadas con las capacidades del equipo y de los usuarios finales, evitando complejidades innecesarias.

Evolución incremental

La solución está diseñada para evolucionar de forma progresiva. La arquitectura y la organización del sistema permiten añadir nuevas funcionalidades o mejoras en fases posteriores, sin comprometer la estabilidad del sistema ni requerir rediseños completos.

3. Arquitectura del sistema

La solución propuesta se basa en una **arquitectura en tres capas**, claramente diferenciadas y desacopladas, que permite organizar el sistema de forma estructurada, facilitar su mantenimiento y garantizar su evolución futura sin necesidad de rediseños complejos.

Las capas que componen la arquitectura son:

- **Capa de presentación (Frontend)**
- **Capa de lógica de negocio (Backend)**
- **Capa de datos (Base de datos)**

3.1 Diagrama de arquitectura general

A nivel conceptual, la arquitectura del sistema se estructura en los siguientes elementos:

- **Capa de presentación (Frontend)**: Aplicación web desarrollada en Angular, responsable de la interacción con el usuario, la navegación entre vistas y la presentación de la información.
- **Capa de lógica de negocio (Backend)**: Servicios REST desarrollados en Node.js con Express.js, responsables de implementar la lógica de negocio,

las operaciones CRUD de pedidos, clientes y productos, y el control de accesos por rol.

- **Capa de datos (Base de datos)**: Base de datos relacional PostgreSQL, encargada de almacenar de forma persistente la información de usuarios, clientes, productos, pedidos y sus estados.

El flujo general de la información es el siguiente:

1. El usuario interactúa con el sistema a través del frontend.
2. El frontend envía peticiones HTTP a los servicios REST del backend.
3. El backend valida la autenticación, el rol del usuario y las reglas de negocio.
4. El backend accede a la base de datos para consultar o modificar la información.
5. El backend devuelve la respuesta al frontend para su presentación al usuario.

3.2 Descripción de capas

3.2.1 Capa de presentación (Frontend)

La capa de presentación se implementa como una aplicación web basada en Angular y está organizada en **módulos funcionales** que reflejan los dominios del negocio:

- Módulo de **Pedidos**, que incluye las vistas de listado, alta, edición, cancelación, preparación, reparto e histórico.
- Módulo de **Clientes**, que incluye las vistas de listado, alta y edición de clientes.
- Módulo de **Productos**, que incluye las vistas de listado, alta y edición de productos.
- Módulos transversales de autenticación, navegación y control de acceso.

El frontend es responsable de:

- Mostrar u ocultar secciones del sistema según el rol del usuario.
- Guiar al usuario mediante interfaces simples y claras.
- Validar de forma básica los datos introducidos antes de enviarlos al backend.
- Consumir las APIs REST para realizar las operaciones CRUD.

La lógica de negocio y las validaciones críticas se delegan siempre al backend.

3.2.2 Capa de lógica de negocio (Backend)

La capa de lógica de negocio se implementa mediante Node.js con Express.js y expone un conjunto de **APIs REST** que dan soporte a las operaciones del sistema.

El backend es responsable de:

- Gestionar la autenticación de usuarios.
- Aplicar el control de acceso por rol a todas las operaciones y secciones.
- Implementar las operaciones CRUD de pedidos, clientes y productos.
- Validar reglas de negocio, como restricciones por estado del pedido.
- Garantizar la consistencia de la información antes de persistirla.

La lógica de negocio se organiza en servicios independientes por dominio funcional (pedidos, clientes, productos), lo que facilita el mantenimiento y la evolución del sistema.

3.2.3 Capa de datos (Base de datos)

La capa de datos se basa en una base de datos relacional PostgreSQL, diseñada para almacenar de forma estructurada y consistente la información del sistema.

La base de datos incluye, entre otras, las siguientes entidades principales:

- Usuarios y roles.
- Clientes.
- Productos.
- Pedidos.
- Estados e histórico de estados de los pedidos.

La base de datos garantiza:

- Integridad referencial entre pedidos, clientes y productos.
- Uso de eliminación lógica para clientes, productos y pedidos cuando corresponda.
- Conservación del histórico de información para fines de trazabilidad.

4. Propuesta tecnológica

4.1 Tecnologías seleccionadas

Las tecnologías seleccionadas para el desarrollo del sistema son las siguientes:

- **Angular**: Framework principal para el desarrollo de la interfaz de usuario.
- **TypeScript**: Lenguaje de programación utilizado en el desarrollo del frontend, aportando tipado estático y mayor robustez al código.
- **HTML5**: Utilizado para la estructura y el marcado del contenido de la interfaz web.
- **CSS3 / SCSS / Bootstrap**: Tecnologías empleadas para el diseño visual, estilos y adaptación responsive de la interfaz.
- **Node.js con Express.js**: Plataforma y framework utilizados para la implementación de la lógica de negocio y los servicios backend.
- **PostgreSQL**: Sistema de gestión de base de datos relacional utilizado para el almacenamiento persistente de la información.

Este conjunto tecnológico es coherente con la arquitectura en tres capas definida y permite cubrir de forma adecuada los requisitos del sistema.

4.2 Justificación de la elección tecnológica

Angular y TypeScript (Frontend)

Angular ha sido seleccionado como framework de frontend por su enfoque estructurado y modular, especialmente adecuado para aplicaciones empresariales. Permite organizar la aplicación en módulos, componentes y servicios, facilitando el mantenimiento y la evolución del sistema a medio y largo plazo.

El uso de **TypeScript** aporta ventajas clave:

- Mejora la calidad del código mediante tipado estático.
- Reduce errores en tiempo de ejecución.
- Facilita el trabajo en equipo y la comprensión del código.

Estas características son especialmente relevantes en un proyecto de desarrollo interno, donde la claridad y mantenibilidad del código son factores críticos.

HTML5, CSS3, SCSS y Bootstrap

HTML5 proporciona una base estándar para la estructura del contenido, mientras que CSS3 y SCSS permiten una gestión clara y organizada de los estilos.

El uso de **Bootstrap** facilita:

- La creación de interfaces consistentes.
- La adaptación del sistema a diferentes resoluciones de pantalla.
- La reducción del esfuerzo de diseño visual.

Esto contribuye directamente al cumplimiento de los requisitos no funcionales relacionados con **usabilidad**, **compatibilidad web** y **facilidad de adopción**, especialmente considerando que los usuarios finales tienen un bajo conocimiento tecnológico.

Node.js con Express.js (Backend)

Node.js ha sido seleccionado como plataforma de backend por su ligereza, eficiencia y amplia adopción en el desarrollo de APIs REST. Permite gestionar un número moderado de usuarios concurrentes con un rendimiento adecuado, alineado con el requisito de soporte para aproximadamente 100 usuarios.

El framework **Express.js** facilita:

- La creación y organización de servicios RESTful.
- La definición clara de rutas y controladores.
- La implementación de middleware para autenticación, autorización y validaciones.

Esta combinación resulta adecuada para centralizar la lógica de negocio, aplicar las reglas definidas en los requisitos funcionales y gestionar el flujo de estados del pedido de forma controlada.

PostgreSQL (Base de datos)

PostgreSQL se ha elegido como sistema de gestión de base de datos relacional por su robustez, fiabilidad y capacidad para garantizar la integridad de los datos.

Las principales razones de esta elección son:

- Soporte completo para integridad referencial mediante claves primarias y foráneas.
- Buen rendimiento en consultas relacionales.
- Adecuación para sistemas con necesidad de trazabilidad e histórico de información.

PostgreSQL permite gestionar de forma consistente la información de pedidos, usuarios, estados e históricos, cumpliendo los requisitos de fiabilidad, trazabilidad y consistencia de datos definidos en el análisis funcional.

4.3 Compatibilidad con el entorno actual

La solución propuesta es compatible con el entorno actual de la empresa, ya que:

- Se accede mediante navegador web, sin instalaciones locales.
- No requiere infraestructuras complejas.
- Se adapta a los procedimientos básicos existentes.
- Permite una implantación progresiva sin interrumpir la actividad diaria.

5. Diseño del Frontend

5.1 Estructura general de la aplicación Angular

La aplicación Angular se organiza de forma estructurada en los siguientes elementos principales:

- **Módulos funcionales**: Agrupan funcionalidades relacionadas, permitiendo una organización clara del código y una evolución controlada del sistema.
- **Componentes**: Encargados de la presentación visual y de la interacción directa con el usuario.
- **Servicios**: Responsables de la comunicación con el backend mediante APIs REST y de la gestión compartida de datos.
- **Guards de navegación**: Controlan el acceso a determinadas rutas según el perfil del usuario autenticado.

Esta estructura permite:

- Reutilizar componentes comunes.
- Centralizar la comunicación con el backend.
- Aplicar control de accesos desde el Frontend como refuerzo al control existente en el Backend.

5.2 Componentes principales

Los principales componentes del Frontend se definen en base a los requisitos funcionales y las historias de usuario documentadas:

Gestión de pedidos

- **Listado de pedidos**, con visualización del estado actual y acceso a acciones permitidas según rol.
- **Formulario de registro y edición de pedidos**, con validaciones guiadas y restricciones según estado y rol.
- **Vista de preparación de pedidos (Almacén)**, destinada exclusivamente al personal de almacén.
- **Vista de reparto**, destinada al personal de reparto para la confirmación de entregas.
- **Vista de histórico de pedidos**, de solo consulta para los roles autorizados.

Gestión de clientes

- **Listado de clientes**, con consulta de información básica.

- **Formulario de alta y edición de clientes**, con operaciones restringidas según rol.
- **Desactivación lógica de clientes**, accesible solo para roles autorizados.

Gestión de productos

- **Listado de productos**, con consulta del catálogo básico.
- **Formulario de alta y edición de productos**, con operaciones restringidas según rol.
- **Desactivación lógica de productos**, accesible solo para roles autorizados.

5.3 Gestión de navegación y rutas

La navegación de la aplicación se gestiona mediante el sistema de **rutas de Angular**, definiendo rutas claras y coherentes para cada funcionalidad.

Aspectos clave de la gestión de rutas:

- Rutas protegidas mediante guards que verifican el perfil del usuario.
- Acceso diferenciado según rol (comercial, almacén, reparto, administración, dirección).
- Redirección a vistas autorizadas en función del perfil tras el inicio de sesión.

5.4 Diseño de la interfaz de usuario

El diseño de la interfaz de usuario se basa en los siguientes principios, alineados con los requisitos no funcionales definidos:

- **Simplicidad visual**: Pantallas limpias, con información clara y bien jerarquizada.
- **Coherencia con los procesos actuales**: La organización de pantallas y acciones sigue el flujo de trabajo que ya conoce el personal.
- **Claridad en acciones y mensajes**: Botones claramente identificados, mensajes de confirmación y error comprensibles.
- **Reducción de errores**: Uso de campos obligatorios, listas desplegables y validaciones visuales para evitar entradas incorrectas.

La interfaz está específicamente orientada a **usuarios con bajo conocimiento tecnológico**, facilitando una rápida adopción del sistema y reduciendo la resistencia al cambio durante la implantación.

6. Diseño del Backend

6.1 Estructura del proyecto Node.js

La estructura del proyecto Backend se organiza de forma modular, separando claramente responsabilidades para facilitar el mantenimiento, la evolución futura y la comprensión del código por parte del equipo técnico.

De manera conceptual, la organización del proyecto incluye los siguientes elementos:

- **Rutas (Routes)**: Definen los endpoints REST disponibles y delegan la lógica en los controladores correspondientes.
- **Controladores (Controllers)**: Gestionan las solicitudes HTTP, validan los datos de entrada y coordinan la ejecución de la lógica de negocio.
- **Servicios (Services)**: Implementan la lógica de negocio derivada de los requisitos funcionales y las historias de usuario, manteniendo los controladores ligeros.
- **Modelos (Models)**: Representan las entidades principales del sistema y su relación con la base de datos.

6.2 Gestión de la lógica de negocio

La lógica de negocio se concentra en servicios independientes para cada dominio:

- **Servicio de pedidos**, responsable de:
    - Operaciones CRUD de pedidos.
    - Gestión de estados del pedido.
    - Restricciones de modificación y cancelación según estado y rol.
- **Servicio de clientes**, responsable de:
    - Operaciones CRUD básicas de clientes.
    - Desactivación lógica de clientes.
- **Servicio de productos**, responsable de:
    - Operaciones CRUD básicas de productos.
    - Desactivación lógica de productos.

Todas las operaciones críticas se validan en el backend, independientemente de las restricciones aplicadas en el frontend.

6.3 Control de accesos y roles

El Backend implementa un **control de accesos basado en roles**, alineado con el requisito funcional **RF-08 Gestión de perfiles y permisos**.

Los perfiles definidos son:

- Dirección
- Comercial
- Almacén
- Reparto
- Administración

Para cada operación del sistema, el Backend:

- Verifica que el usuario esté autenticado.
- Comprueba el perfil asociado al usuario.
- Autoriza o rechaza la acción solicitada según los permisos definidos.

Este enfoque garantiza:

- Seguridad de la información.
- Uso correcto del sistema.
- Reducción de errores derivados de acciones no autorizadas.

7. Diseño de la base de datos

7.1 Modelo de datos conceptual

El modelo de datos conceptual identifica las entidades principales del sistema y sus relaciones, alineadas con los requisitos funcionales definidos.

Las entidades principales son:

- **Usuario**: Representa a los usuarios internos del sistema. Cada usuario dispone de un rol que determina su acceso a las funcionalidades y operaciones permitidas.
- **Cliente**: Representa a los clientes de la empresa. Se gestiona información básica de contacto y localización, utilizada como referencia en el registro de pedidos.
- **Producto**: Representa los productos comercializados por la empresa. Se gestiona un catálogo básico utilizado en el registro de pedidos, sin control avanzado de inventario.
- **Pedido**: Representa un pedido realizado por un cliente. Incluye la información del cliente asociado, los productos solicitados, las fechas relevantes y el estado del pedido.
- **Estado del pedido**: Define el estado en el que se encuentra un pedido dentro de su ciclo de vida (registrado, preparación, reparto, entregado, cancelado).

7.2 Modelo lógico de la base de datos

El modelo lógico traduce el modelo conceptual a tablas relacionales en PostgreSQL, manteniendo la coherencia con las entidades definidas.

Las tablas principales del sistema son:

- **usuarios**
  Contiene la información de autenticación y el rol del usuario.
- **clientes**
  Contiene la información básica de los clientes, incluyendo un indicador de activación lógica.
- **productos**
  Contiene el catálogo de productos disponibles, con indicador de disponibilidad lógica.
- **pedidos**
  Contiene la información del pedido, incluyendo:
  - Referencia al cliente.
  - Lista de productos asociada al pedido.

- Fechas de solicitud y entrega prevista.
- Estado del pedido.
- Motivo de cancelación cuando corresponda.

Relaciones principales

- Un **pedido** está asociado a **un único cliente**.
- Un **cliente** puede tener **varios pedidos** asociados.
- Un **pedido** contiene **uno o varios productos**, almacenados como estructura JSON para representar el detalle del pedido.
- Un **usuario**, a través de su rol, condiciona las operaciones permitidas sobre pedidos, clientes y productos.

Claves

- Todas las tablas disponen de **clave primaria**.
- Las relaciones entre pedidos y clientes se implementan mediante **claves foráneas**.
- Se aplican **restricciones de dominio** para los valores de rol de usuario y estado del pedido.
- Se utilizan campos de control (activo, disponible) para implementar **eliminación lógica**.

7.3 Consideraciones de integridad y consistencia

Para garantizar la fiabilidad de la información y evitar inconsistencias, el diseño de la base de datos incorpora las siguientes consideraciones:

- **Integridad referencial**: Todas las relaciones entre tablas se definen mediante claves foráneas, impidiendo la existencia de registros huérfanos.
- **Restricciones de obligatoriedad**: Los campos críticos (identificador del pedido, estado, fechas clave, usuario responsable) se definen como obligatorios.
- **Control de estados válidos**: El estado de un pedido solo puede tomar valores definidos en la tabla de estados, evitando estados no reconocidos.
- **Inmutabilidad de estados finales**: Los pedidos en estado “Entregado” o “Cancelado” no pueden ser modificados, conforme a las reglas de negocio.
- **Trazabilidad completa**: Cada cambio de estado queda registrado en el histórico, permitiendo auditoría y resolución de incidencias.
- **Consistencia transaccional**: Las operaciones críticas (registro de pedidos, cambio de estado, cancelaciones) deben ejecutarse de forma transaccional para evitar estados intermedios inconsistentes.

8. Diseño de APIs y comunicación entre capas

8.1 Endpoints principales

APIs de usuarios y seguridad

- **POST /auth/login**: Autenticación de usuarios y obtención del rol asociado.
- **GET /usuarios/perfil**: Consulta de la información básica del usuario.

APIs de clientes

- **GET /clientes**: Consulta del listado de clientes.
- **GET /clientes/{id}**: Consulta de un cliente concreto.
- **POST /clientes**: Alta de un nuevo cliente.
- **PUT /clientes/{id}**: Modificación de los datos básicos de un cliente.
- **PUT /clientes/{id}/desactivar**: Desactivación lógica de un cliente.

APIs de productos

- **GET /productos**: Consulta del catálogo de productos.
- **GET /productos/{id}**: Consulta de un producto concreto.
- **POST /productos**: Alta de un nuevo producto.
- **PUT /productos/{id}**: Modificación de los datos básicos de un producto.
- **PUT /productos/{id}/desactivar**: Desactivación lógica de un producto.

APIs de pedidos

- **GET /pedidos**: Consulta del listado de pedidos.
- **GET /pedidos/{id}**: Consulta de un pedido concreto.
- **POST /pedidos**: Registro de un nuevo pedido.
- **PUT /pedidos/{id}**: Modificación de un pedido en estados permitidos.
- **PUT /pedidos/{id}/cancelar**: Cancelación de un pedido con motivo.

APIs de gestión de estados del pedido

- **PUT /pedidos/{id}/preparar**: Cambio del estado del pedido a preparación.
- **PUT /pedidos/{id}/entregar**: Cambio del estado del pedido a entregado.

9. Seguridad técnica

9.1 Autenticación de usuarios

El sistema implementa un mecanismo de **autenticación de usuarios** basado en credenciales individuales (usuario y contraseña), conforme a lo definido en los requisitos técnicos y no funcionales.

Las principales características de la autenticación son:

- Cada usuario dispone de credenciales únicas para acceder al sistema.
- La validación de credenciales se realiza exclusivamente en el Backend.
- El Frontend no almacena información sensible de autenticación.
- Tras la autenticación correcta, el sistema mantiene la sesión del usuario mediante un mecanismo de sesión o token, gestionado por el Backend.

Este enfoque permite:

- Identificar de forma inequívoca a cada usuario.
- Asociar cada acción realizada a un usuario concreto.
- Evitar accesos anónimos o no controlados al sistema.

La autenticación es un requisito previo para cualquier operación del sistema, incluyendo la consulta de pedidos.

9.2 Autorización y control de accesos

Una vez autenticado el usuario, el sistema aplica un **modelo de autorización basado en roles**, alineado con el requisito funcional **RF-08 Gestión de perfiles y permisos** y con los perfiles definidos en el análisis funcional.

Los perfiles contemplados son:

- Dirección
- Comercial
- Almacén
- Reparto
- Administración

9.3 Protección de datos

El sistema gestiona información sensible relacionada con pedidos y datos básicos de clientes, por lo que se aplican medidas técnicas orientadas a la **protección de**

datos personales, en línea con las obligaciones legales y los riesgos identificados en el análisis de riesgos.

Las principales medidas de protección de datos son:

- **Minimización de datos**: Solo se almacenan los datos estrictamente necesarios para la gestión de pedidos y su seguimiento, evitando información innecesaria.
- **Acceso restringido a la información**: Los datos solo son accesibles para usuarios autenticados y autorizados, según su perfil.
- **Separación de responsabilidades**: El Frontend no accede directamente a la base de datos; toda la información pasa por el Backend, donde se aplican validaciones y controles de seguridad.
- **Protección en la comunicación**: Se recomienda el uso de **HTTPS** para la transmisión de información entre el Frontend y el Backend, especialmente para credenciales y datos sensibles.
- **Integridad de la información**: El diseño de la base de datos incluye restricciones y claves foráneas que evitan inconsistencias y modificaciones no autorizadas.

Estas medidas permiten cumplir los requisitos de seguridad definidos sin introducir mecanismos complejos que dificulten la adopción del sistema o incrementen innecesariamente los costes del proyecto.

10. Rendimiento y escalabilidad

10.1 Consideraciones de rendimiento

El sistema deberá ofrecer un **rendimiento adecuado durante la ejecución de las operaciones habituales**, garantizando una experiencia de uso fluida para todos los perfiles implicados (equipo comercial, almacén, reparto, administración y dirección).

Las principales consideraciones de rendimiento son las siguientes:

- El sistema deberá responder a las operaciones más comunes (consulta de pedidos, registro de pedidos, cambio de estado) en un **tiempo inferior a 2 segundos**, conforme a lo establecido en los requisitos técnicos.
- El rendimiento deberá mantenerse estable con un **mínimo de 100 usuarios concurrentes** durante la jornada laboral.
- Las operaciones críticas del negocio (registro de pedidos y actualización de estados) tendrán prioridad frente a operaciones de consulta secundaria.
- El backend será responsable de centralizar las validaciones y la lógica de negocio, evitando sobrecargar la capa de presentación.

El diseño de la arquitectura en tres capas contribuye directamente a este objetivo, al permitir una distribución clara de responsabilidades y evitar accesos innecesarios a la base de datos.

10.2 Escalabilidad prevista

La solución está diseñada para ofrecer una **escalabilidad básica y controlada**, adecuada al crecimiento esperado de Distribuciones Sierra Norte S.L., sin requerir rediseños estructurales ni inversiones significativas en fases iniciales.

Las principales consideraciones de escalabilidad son:

- Capacidad para **incrementar el número de usuarios** sin modificar la arquitectura del sistema.
- Posibilidad de gestionar un **mayor volumen de pedidos** manteniendo la coherencia y el rendimiento del sistema.
- Arquitectura en capas que permite:
    - Escalar el backend de forma independiente si fuera necesario.
    - Optimizar la base de datos sin afectar a la capa de presentación.
- Código organizado y modular que facilita la incorporación de nuevas funcionalidades en fases posteriores.

11. Estrategia de pruebas técnicas

11.1 Pruebas unitarias

Las pruebas unitarias tienen como finalidad verificar el **correcto funcionamiento de componentes individuales** del sistema, de forma aislada, antes de su integración con otros elementos.

En el **backend**, se realizarán pruebas unitarias sobre:

- Servicios de **clientes**, verificando altas, modificaciones y desactivaciones lógicas.
- Servicios de **productos**, validando la gestión del catálogo básico.
- Servicios de **pedidos**, comprobando:
    - Operaciones CRUD.
    - Validación de estados del pedido.
    - Restricciones de modificación y cancelación según estado.
- Lógica de control de accesos por rol.

En el **frontend**, se validarán:

- Componentes de formularios (clientes, productos y pedidos).
- Validaciones visuales y mensajes mostrados al usuario.
- Comportamiento de componentes ante estados no editables.

11.2 Pruebas de integración

Las pruebas de integración aseguran la correcta comunicación entre las distintas capas del sistema (frontend, backend y base de datos).

Se comprobarán, entre otros, los siguientes escenarios:

- Comunicación correcta entre el frontend y las APIs REST del backend.
- Persistencia correcta de datos en PostgreSQL para clientes, productos y pedidos.
- Integridad de las relaciones entre pedidos, clientes y productos.
- Aplicación coherente de las restricciones por rol en llamadas directas a los endpoints.
- Gestión correcta de errores y respuestas del backend ante operaciones no autorizadas.

Estas pruebas permiten validar los flujos completos del sistema desde la interfaz de usuario hasta la base de datos.

12. Despliegue e infraestructura

12.1 Entornos del sistema

Para asegurar la calidad del desarrollo y minimizar riesgos durante la puesta en producción, se definen los siguientes entornos:

Entorno de desarrollo

- Utilizado por el equipo técnico para el desarrollo y pruebas iniciales de funcionalidades.
- Permite validar la lógica de negocio, los flujos de estado y las interfaces antes de su despliegue.
- Puede ejecutarse en entornos locales o servidores internos destinados a desarrollo.

Entorno de pruebas (test)

- Destinado a la validación funcional y técnica del sistema antes de su puesta en producción.
- Permite realizar pruebas de integración, rendimiento básico y validación con usuarios clave.
- Replica, en la medida de lo posible, la configuración del entorno productivo.

Entorno de producción

- Entorno final donde el sistema será utilizado por los usuarios de la empresa.
- Accesible únicamente a usuarios autorizados.
- Debe garantizar estabilidad, rendimiento y disponibilidad durante la jornada laboral.

12.2 Gestión de versiones y despliegues

Control de versiones

Se utilizará **Git** como sistema de control de versiones que permite controlar cambios realizados, revertir versiones en caso de incidencias y facilitar el trabajo colaborativo del equipo técnico.

Proceso de despliegue

- Los despliegues se realizarán de forma **controlada y planificada**.
- Las nuevas versiones se validarán previamente en el entorno de pruebas.

13. Conclusión

El presente Documento de Diseño Técnico y Propuesta Tecnológica define una solución técnica coherente y alineada con las necesidades reales de Distribuciones Sierra Norte S.L., orientada a resolver las deficiencias actuales en la gestión manual y descentralizada de pedidos.

La incorporación de funcionalidades de **gestión de clientes, productos y pedidos**, junto con la implementación de **operaciones controladas por rol**, proporciona una base sólida para centralizar la información clave del negocio, mejorar la coordinación entre departamentos y reducir errores operativos derivados de datos inconsistentes o duplicados.

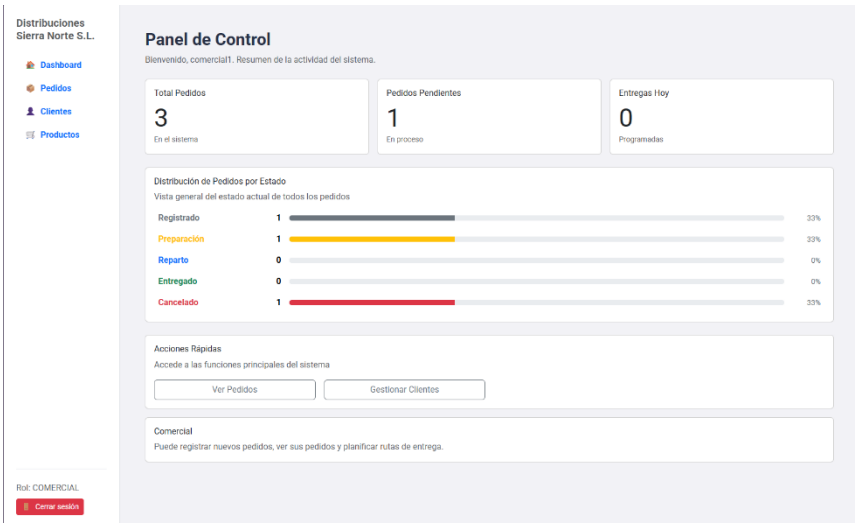
La arquitectura propuesta, basada en una separación clara entre frontend, backend y base de datos, y apoyada en tecnologías consolidadas como **Angular, Node.js con Express.js y PostgreSQL**, garantiza la mantenibilidad, escalabilidad básica y seguridad del sistema. El control de accesos por rol, aplicado de forma consistente en todas las capas, refuerza la protección de la información y asegura un uso adecuado de las funcionalidades por parte de cada perfil de usuario.

El diseño tiene en cuenta las restricciones del proyecto, especialmente el presupuesto limitado, la necesidad de no interrumpir la actividad diaria y el bajo nivel de formación técnica del personal, apostando por una **implantación progresiva** y una interfaz sencilla y guiada.

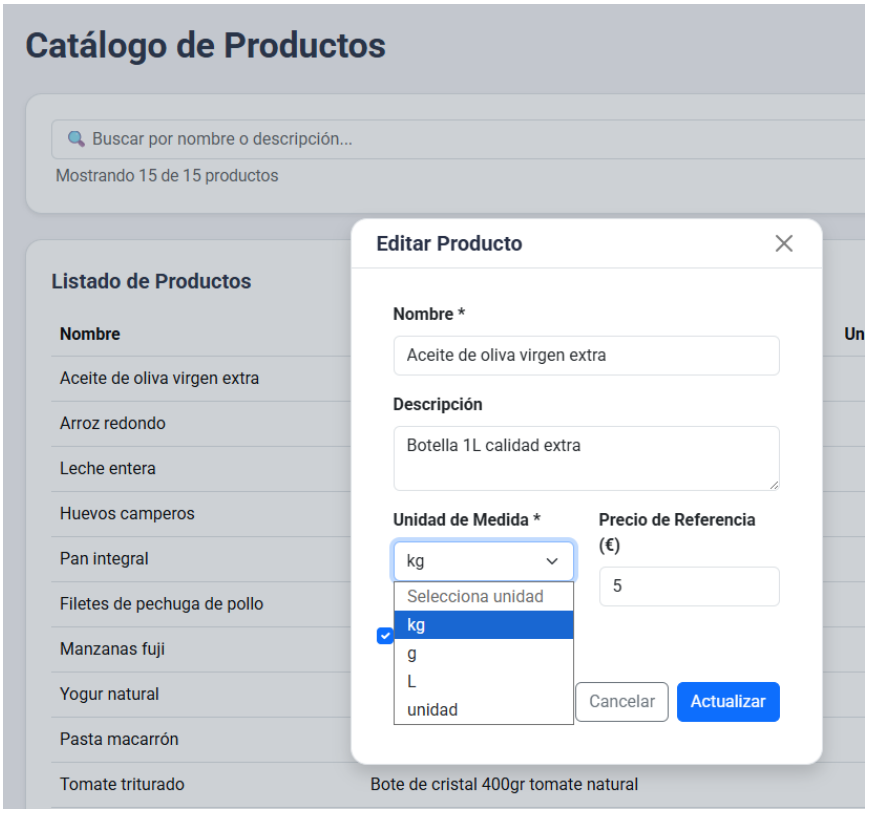
En conjunto, la solución propuesta permite a la empresa avanzar hacia un modelo de gestión más organizado, fiable y eficiente, mejorando la visibilidad del estado de los pedidos, reduciendo incidencias y liberando tiempo del personal para tareas de mayor valor. Además, deja preparada una base tecnológica estable sobre la que podrán incorporarse futuras mejoras funcionales conforme evolucionen las necesidades del negocio.

Anexos

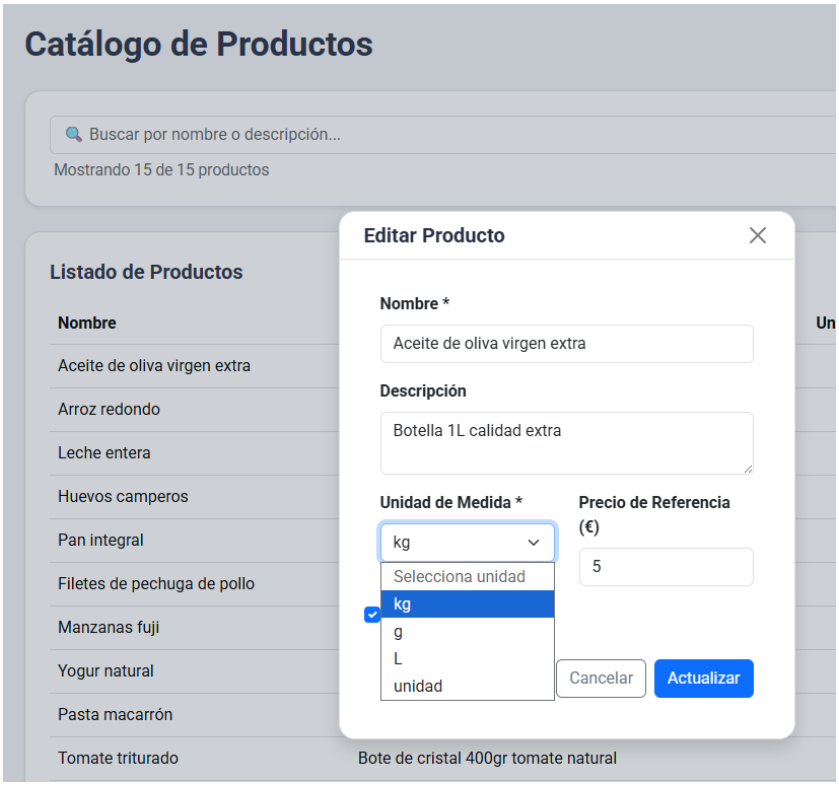
Página Principal - Dashboard



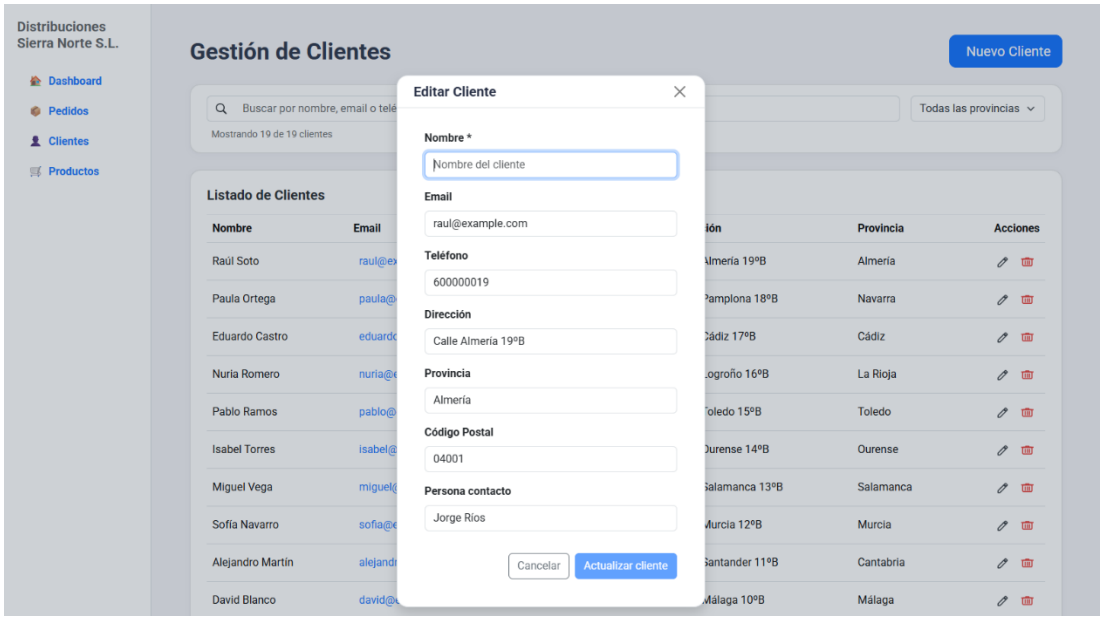
Gestión de Productos – Función de edición



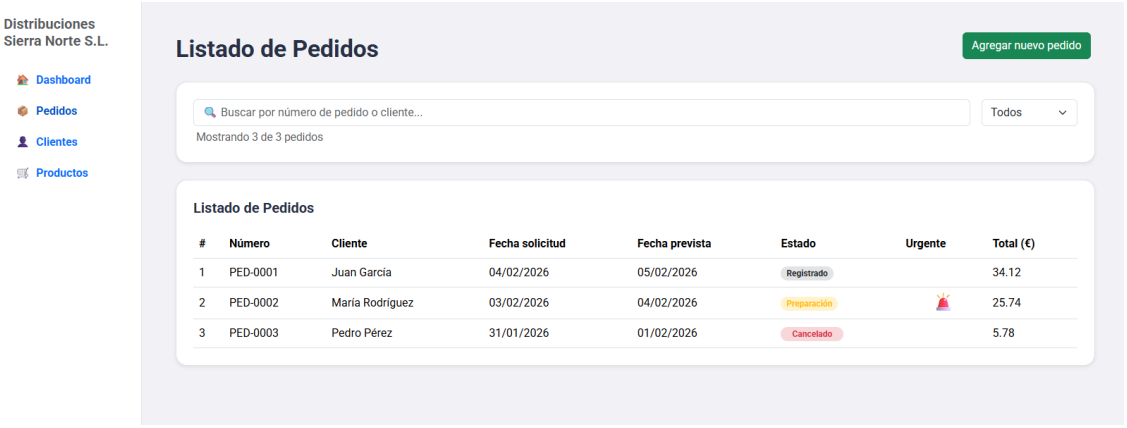
Gestión de Productos – Función de edición



Gestión de Clientes



Gestión de Productos



Mensajes de error