Documento de Arquitectura

Proyecto *Ordena*

| **Información General del Proyecto** | | | |
| --- | --- | --- | --- |
| **\*** **ID. Proyecto:** | 001 | **\* Fecha:** | 08/05/2025 |
| **\*** **Nombre del Proyecto:** | Ordena | | |
| **\*** **Director de Proyecto:** | Fabian Olate | | |
| **\*** **Preparado por:** | Área informática | | |

## Propósito del documento

Este documento describe la arquitectura del sistema *Ordena*, una aplicación web para la gestión de inventario y pedidos en una cadena de ferreterías. Su objetivo es servir como guía técnica para el desarrollo, mantenimiento y evolución del sistema.

## **Alcance**

*Ordena* permite a distintas sucursales gestionar productos, realizar solicitudes de pedido, visualizar inventario y controlar el flujo entre sucursal y bodega central.

## **Restricciones**

* El frontend debe desarrollarse con **React + Vite + TypeScript**, utilizando **Deno** cómo runtime en desarrollo.
* El backend debe construirse con **Django (Python)**, respetando el patrón RESTful para la exposición de servicios.
* La base de datos debe utilizarse desde **Supabase**, con PostgreSQL como motor.
* El sistema debe emplear **autenticación por JWT** para separar responsabilidades del cliente y proteger recursos mediante tokens.
* El sistema debe operar sobre una arquitectura **cliente-servidor con separación en capas** (presentación, lógica de negocio y persistencia).
* Toda la comunicación debe realizarse mediante **API REST**, con protección CORS habilitada.

## Vista de Contexto

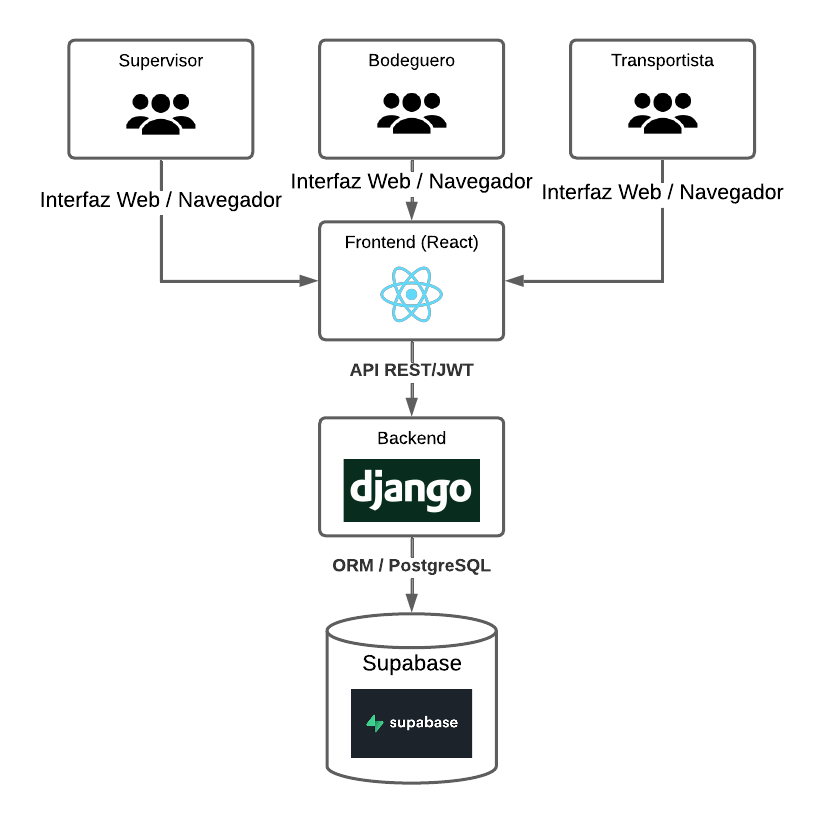
### **Actores externos**

* **Supervisor**: supervisa el estado de inventarios, pedidos, maneja empleados y flujos logísticos.
* **Bodeguero**: gestiona solicitudes de productos desde las sucursales.
* **Transportista**: encargado de realizar pedidos a sucursales.
* **Administrador** (rol técnico): gestiona usuarios, parámetros del sistema y acceso a datos críticos.
* **Supabase**: servicio externo para base de datos en la nube.
* **GitHub**: repositorio de código para colaboración y control de versiones.

### Canales de Comunicación

* **Frontend React (SPA)** se comunica con el **backend Django** mediante **peticiones HTTP (API REST)**.
* **Autenticación** mediante **JWT**, transmitido en cabeceras.
* **Django se conecta a Supabase** usando PostgreSQL a través del ORM.

### Diagrama

**

## **Estrategia de Solución**

El sistema está construido bajo una **arquitectura cliente-servidor** con separación en **capas funcionales**, lo que permite modularidad, escalabilidad y mantenibilidad.

#### **Principales componentes:**

* **Frontend**: SPA (Single Page Application) desarrollada en **React**, usando Vite como empaquetador y TypeScript como lenguaje tipado. Se gestiona el estado local con Zustand, y se estructura en componentes reutilizables según módulos funcionales.
* **Backend**: API RESTful desarrollada en **Django**, dividida por apps (productos, usuarios, pedidos). Usa el ORM de Django para interactuar con la base de datos, y aplica serialización con serializers.py para exponer datos al cliente.
* **Base de datos**: gestionada por **Supabase**, basada en PostgreSQL. Aprovecha relaciones normalizadas, roles personalizados y permite administración remota.
* **Autenticación y seguridad**: se usa **JWT** para mantener sesiones seguras desde el frontend. El backend valida cada token recibido antes de responder peticiones.
* **Separación por capas**:  
  + **Capa de presentación**: React (navegador)
  + **Capa de negocio**: Django (API + lógica de validación)
  + **Capa de datos**: Supabase (modelo relacional)

## **Vista de Contenedores**

| **Componente** | **Tecnología** | **Descripción** |
| --- | --- | --- |
| Frontend | React + Vite | Interfaz web SPA |
| Backend | Django (REST API) | Lógica de negocio, control de acceso, endpoints |
| Base de datos | Supabase PostgreSQL | Persistencia de datos, relaciones normalizadas |

## 

## 

## **Vista de Componentes**

Frontend (contenedor):

* pages/: páginas principales
* components/: componentes UI reutilizables
* store/: manejo de estado (Zustand)
* routes/: rutas protegidas, navegación
* utils/: funciones de ayuda

Backend (contenedor):

* views.py: controladores
* models.py: entidades
* serializers.py: transformadores de datos
* urls.py: rutas de la API
* admin.py: gestión desde panel de admin

## Diagrama de Arquitectura en Capas

