**Documentación Técnica**

Esta sección describe la arquitectura, configuraciones y prácticas técnicas implementadas en MediInventory.

**Arquitectura del Sistema**

El sistema sigue una **arquitectura de capas** con separación clara de responsabilidades:

┌─────────────────────────────────────┐

│ Presentation Layer │

│ (React Components + UI) │

├─────────────────────────────────────┤

│ Business Logic │

│ (Services + Hooks) │

├─────────────────────────────────────┤

│ Data Access │

│ (Supabase Client/Server) │

├─────────────────────────────────────┤

│ Database │

│ (PostgreSQL) │

└─────────────────────────────────────┘

**🔧 Configuración Técnica**

**Next.js Configuration**

// next.config.mjs

/\*\* @type {import('next').NextConfig} \*/

const nextConfig = {

experimental: {

serverActions: true,

},

images: {

domains: ['placeholder.svg'],

},

}

export default nextConfig

**TypeScript Configuration**

// tsconfig.json

{

"compilerOptions": {

"target": "ES2017",

"lib": ["dom", "dom.iterable", "es6"],

"allowJs": true,

"skipLibCheck": true,

"strict": true,

"noEmit": true,

"esModuleInterop": true,

"module": "esnext",

"moduleResolution": "bundler",

"resolveJsonModule": true,

"isolatedModules": true,

"jsx": "preserve",

"incremental": true,

"plugins": [

{

"name": "next"

}

],

"baseUrl": ".",

"paths": {

"@/\*": ["./\*"]

}

},

"include": ["next-env.d.ts", "\*\*/\*.ts", "\*\*/\*.tsx", ".next/types/\*\*/\*.ts"],

"exclude": ["node\_modules"]

}

**Esquema de Base de Datos Detallado**

**Constraints y Validaciones**

**Tabla suppliers:**

* -- Constraints de validación
* CHECK (id\_type IN ('CC', 'NIT', 'CE', 'PP'))
* CHECK (status IN ('active', 'inactive'))
* CHECK (
* economic\_activity IS NULL OR
* (
* jsonb\_typeof(economic\_activity) = 'array' AND
* (
* SELECT bool\_and(
* jsonb\_typeof(value) = 'string' AND
* length(value::text) = 6 AND
* value::text ~ '^"[0-9]{4}"$'
* )
* FROM jsonb\_array\_elements(economic\_activity) AS value
* )
* )
* )
* -- Índices
* CREATE INDEX idx\_suppliers\_id\_number ON suppliers(id\_number);
* CREATE INDEX idx\_suppliers\_status ON suppliers(status);
* CREATE INDEX idx\_suppliers\_economic\_activity ON suppliers USING GIN (economic\_activity);

**Tabla products:**

* -- Constraints de validación
* CHECK (status IN ('active', 'inactive'))
* -- Índices
* CREATE INDEX idx\_products\_code ON products(code);
* CREATE INDEX idx\_products\_status ON products(status);
* CREATE INDEX idx\_products\_name ON products(name);
* CREATE INDEX idx\_products\_laboratory ON products(laboratory);

**Tabla product\_receptions:**

* -- Constraints de validación
* CHECK (quantity > 0)
* CHECK (presentation\_state IN ('bueno', 'regular', 'malo'))
* -- Foreign Keys
* FOREIGN KEY (product\_id) REFERENCES products(id) ON DELETE CASCADE
* FOREIGN KEY (supplier\_id) REFERENCES suppliers(id) ON DELETE CASCADE
* -- Índices
* CREATE INDEX idx\_receptions\_product\_id ON product\_receptions(product\_id);
* CREATE INDEX idx\_receptions\_supplier\_id ON product\_receptions(supplier\_id);
* CREATE INDEX idx\_receptions\_date ON product\_receptions(reception\_date);
* CREATE INDEX idx\_receptions\_invoice ON product\_receptions(invoice\_number);

**Row Level Security (RLS) Policies**

-- Políticas para suppliers

CREATE POLICY "suppliers\_select" ON suppliers FOR SELECT USING (true);

CREATE POLICY "suppliers\_insert" ON suppliers FOR INSERT WITH CHECK (true);

CREATE POLICY "suppliers\_update" ON suppliers FOR UPDATE USING (true);

CREATE POLICY "suppliers\_delete" ON suppliers FOR DELETE USING (true);

-- Políticas para products

CREATE POLICY "products\_select" ON products FOR SELECT USING (true);

CREATE POLICY "products\_insert" ON products FOR INSERT WITH CHECK (true);

CREATE POLICY "products\_update" ON products FOR UPDATE USING (true);

CREATE POLICY "products\_delete" ON products FOR DELETE USING (true);

-- Políticas para product\_receptions

CREATE POLICY "receptions\_select" ON product\_receptions FOR SELECT USING (true);

CREATE POLICY "receptions\_insert" ON product\_receptions FOR INSERT WITH CHECK (true);

CREATE POLICY "receptions\_update" ON product\_receptions FOR UPDATE USING (true);

CREATE POLICY "receptions\_delete" ON product\_receptions FOR DELETE USING (true);

**Servicios de Datos**

Cada entidad tiene su propio servicio con operaciones CRUD estandarizadas. Por ejemplo, el servicio para **Suppliers**:

// lib/services/suppliers.service.ts

import { createBrowserClient } from '@supabase/ssr'

import type { Supplier } from '@/lib/types'

const supabase = createBrowserClient(

process.env.NEXT\_PUBLIC\_SUPABASE\_URL!,

process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!

)

export const suppliersService = {

async getAll(): Promise<Supplier[]> {

const { data, error } = await supabase

.from('suppliers')

.select('\*')

.order('created\_at', { ascending: false })

if (error) throw error

return data || []

},

async create(supplier: Omit<Supplier, 'id' | 'created\_at' | 'updated\_at'>):

Promise<Supplier> {

const { data, error } = await supabase

.from('suppliers')

.insert(supplier)

.select()

.single()

if (error) throw error

return data

},

// ... más métodos (getById, update, delete)

}

**Hooks Personalizados**

Los hooks encapsulan la lógica de estado y efectos secundarios para las interfaces. Ejemplo de hook para proveedores:

// hooks/use-suppliers.ts

import { useState, useEffect } from 'react'

import { suppliersService } from '@/lib/services/suppliers.service'

import type { Supplier } from '@/lib/types'

export function useSuppliers() {

const [suppliers, setSuppliers] = useState<Supplier[]>([])

const [loading, setLoading] = useState(true)

const [error, setError] = useState<string | null>(null)

const fetchSuppliers = async () => {

try {

setLoading(true)

const data = await suppliersService.getAll()

setSuppliers(data)

setError(null)

} catch (err) {

setError(err instanceof Error ? err.message : 'Error desconocido')

} finally {

setLoading(false)

}

}

useEffect(() => {

fetchSuppliers()

}, [])

return {

suppliers,

loading,

error,

refetch: fetchSuppliers

}

}

**Sistema de Componentes**

**Jerarquía de Componentes**

App Layout

├── MainNav (Navigation)

├── Page Components

│ ├── SuppliersPage

│ │ └── SuppliersList

│ │ ├── AddSupplierDialog

│ │ ├── EditSupplierDialog

│ │ └── DeleteSupplierDialog

│ ├── ProductsPage

│ │ └── ProductsList

│ │ ├── AddProductDialog

│ │ ├── EditProductDialog

│ │ └── DeleteProductDialog

│ └── ReceptionsPage

│ └── ReceptionsList

│ ├── AddReceptionDialog

│ ├── EditReceptionDialog

│ └── DeleteReceptionDialog

└── UI Components (Shadcn/ui)

├── Button, Input, Select

├── Dialog, Sheet, Tabs

├── Table, Card, Badge

└── Toast, Alert

**Patrón de Componentes de Lista**

Todos los componentes de lista siguen el mismo patrón:

interface ListComponentProps {

searchTerm: string

statusFilter: string

onSearch: (term: string) => void

onStatusFilter: (status: string) => void

}

function EntityList({ searchTerm, statusFilter, onSearch, onStatusFilter }:

ListComponentProps) {

// 1. Hook para datos

const { entities, loading, error, refetch } = useEntities()

// 2. Filtrado local

const filteredEntities = useMemo(() => {

return entities.filter(entity => {

// Lógica de filtrado

})

}, [entities, searchTerm, statusFilter])

// 3. Renderizado

return (

<div>

{/\* Controles de búsqueda y filtros \*/}

{/\* Tabla de datos \*/}

{/\* Dialogs para CRUD \*/}

</div>

)

}

**Validación y Tipos**

**Esquemas Zod**

// lib/validations/supplier.ts

import { z } from 'zod'

export const supplierSchema = z.object({

id\_type: z.enum(['CC', 'NIT', 'CE', 'PP']),

id\_number: z.string().min(1, 'Número de documento requerido'),

name: z.string().min(1, 'Nombre requerido'),

address: z.string().optional(),

phone: z.string().optional(),

email: z.string().email('Email inválido').optional().or(z.literal('')),

contact\_person: z.string().optional(),

economic\_activity: z.array(z.string()).optional(),

status: z.enum(['active', 'inactive']).default('active')

})

export type SupplierFormData = z.infer<typeof supplierSchema>

**Tipos TypeScript**

// lib/types.ts

export interface BaseEntity {

id: string

created\_at: string

updated\_at: string

}

export interface Supplier extends BaseEntity {

id\_type: 'CC' | 'NIT' | 'CE' | 'PP'

id\_number: string

name: string

address?: string

phone?: string

email?: string

contact\_person?: string

economic\_activity?: string[]

status: 'active' | 'inactive'

}

// Tipos utilitarios

export type CreateSupplier = Omit<Supplier, keyof BaseEntity>

export type UpdateSupplier = Partial<CreateSupplier>

**Manejo de Estados**

**Estados de Componentes**

// Estados comunes en componentes de lista

interface ListState {

entities: Entity[]

loading: boolean

error: string | null

searchTerm: string

statusFilter: string

selectedEntity: Entity | null

showAddDialog: boolean

showEditDialog: boolean

showDeleteDialog: boolean

}

**Estados de Formularios**

// Ejemplo usando React Hook Form

const form = useForm<SupplierFormData>({

resolver: zodResolver(supplierSchema),

defaultValues: {

id\_type: 'NIT',

status: 'active',

economic\_activity: []

}

})

**Optimizaciones de Rendimiento**

**Índices de Base de Datos**

* **Búsquedas frecuentes:** Índices en campos de búsqueda (idx\_suppliers\_id\_number, etc.).
* **Filtros:** Índices en campos de estado (idx\_suppliers\_status, idx\_products\_status).
* **Relaciones:** Índices en claves foráneas (idx\_receptions\_product\_id, idx\_receptions\_supplier\_id).
* **Ordenamiento:** Índices en campos de ordenamiento (idx\_receptions\_date).

**Optimizaciones Frontend**

* **useMemo:** Para filtrado de listas en memoria.
* **useCallback:** Para funciones de evento.
* **React.memo:** Para memorizar componentes que no cambian frecuentemente.
* **Lazy loading:** Para cargar componentes o librerías pesadas bajo demanda.

**Ejemplo de Optimización**

// Filtrado optimizado con useMemo

const filteredSuppliers = useMemo(() => {

return suppliers.filter(supplier => {

const matchesSearch =

supplier.name.toLowerCase().includes(searchTerm.toLowerCase()) ||

supplier.id\_number.includes(searchTerm)

const matchesStatus = statusFilter === 'all' || supplier.status === statusFilter

return matchesSearch && matchesStatus

})

}, [suppliers, searchTerm, statusFilter])

// Función de búsqueda optimizada con useCallback

const handleSearch = useCallback((term: string) => {

setSearchTerm(term)

}, [])

**Seguridad y Mejores Prácticas**

**Validación en Capas**

1. **Frontend:** Validación inmediata con Zod (campos del formulario).
2. **Backend:** Restricciones (CHECK) en la base de datos para garantizar integridad.
3. **Tipo:** TypeScript para prevenir errores en tiempo de compilación.

**Sanitización de Datos**

// Sanitización de entrada

const sanitizeInput = (input: string): string => {

return input.trim().replace(/[<>]/g, '')

}

// Validación de email

const isValidEmail = (email: string): boolean => {

return z.string().email().safeParse(email).success

}

**Manejo de Errores**

// Patrón de manejo de errores

try {

const result = await service.operation()

// Éxito

} catch (error) {

if (error instanceof Error) {

toast.error(error.message)

} else {

toast.error('Error desconocido')

}

console.error('Operation failed:', error)

}

**Monitoreo y Logging**

**Logging en Desarrollo**

// Logging condicional

const isDev = process.env.NODE\_ENV === 'development'

const log = {

info: (message: string, data?: any) => {

if (isDev) console.log(`[INFO] ${message}`, data)

},

error: (message: string, error?: any) => {

if (isDev) console.error(`[ERROR] ${message}`, error)

}

}

**Analytics**

* **Vercel Analytics:** Métricas de uso.
* **Error Tracking:** Logs de errores en producción.
* **Performance:** Métricas de rendimiento.

**Despliegue y CI/CD**

**Variables de Entorno por Ambiente**

# Desarrollo

NEXT\_PUBLIC\_SUPABASE\_URL=http://localhost:54321

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=dev\_key

# Producción

NEXT\_PUBLIC\_SUPABASE\_URL=https://your-project.supabase.co

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=prod\_key

**Pipeline de Despliegue**

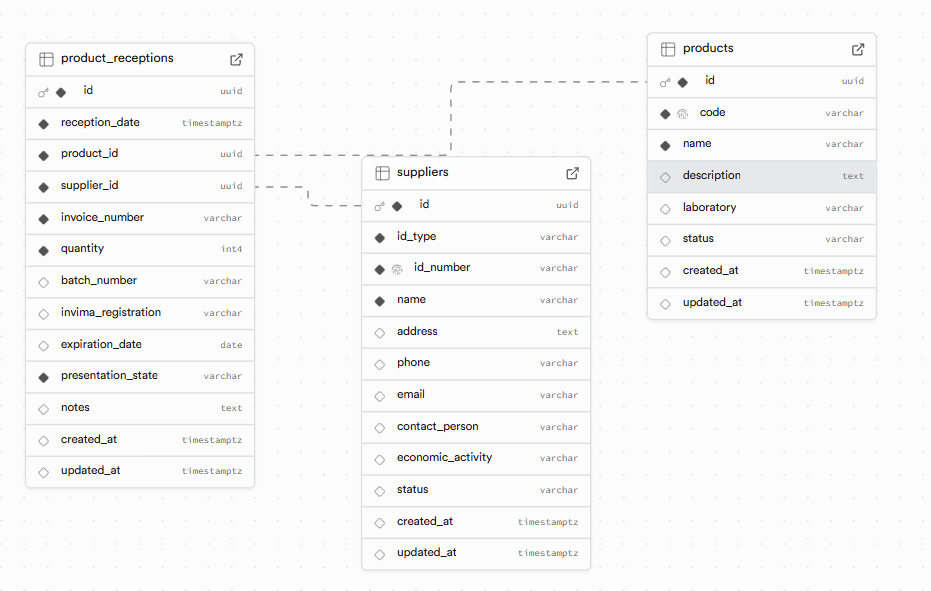
1. **Commit** → Trigger del pipeline de CI.
2. **Build** → Compilar con Next.js.
3. **Test** → Ejecutar suite de pruebas.
4. **Deploy** → Desplegar en Vercel.
5. **Verify** → Comprobaciones de salud y logs.

Esta documentación técnica proporciona una visión completa de la arquitectura, patrones y mejores prácticas implementadas en MediInventory.

**6. Modelo de Base de Datos**

Esta sección describe el esquema de la base de datos, el diccionario de datos y las relaciones del sistema.

**Diagrama del Modelo de Base de Datos**

****

**Relaciones entre Tablas**

1. **SUPPLIERS → PRODUCT\_RECEPTIONS (1:N)**
   * **Relación:** Un proveedor puede tener múltiples recepciones de productos.
   * **Clave Foránea:** product\_receptions.supplier\_id → suppliers.id
   * **Integridad:** ON DELETE CASCADE (al eliminar un proveedor, se eliminan sus recepciones).
2. **PRODUCTS → PRODUCT\_RECEPTIONS (1:N)**
   * **Relación:** Un producto puede tener múltiples recepciones.
   * **Clave Foránea:** product\_receptions.product\_id → products.id
   * **Integridad:** ON DELETE CASCADE (al eliminar un producto, se eliminan sus recepciones).

**Seguridad y Políticas RLS**

Todas las tablas tienen RLS habilitado con políticas que permiten operaciones completas para usuarios autenticados.

* **SUPPLIERS:**
  + suppliers\_select: Permite SELECT a todos los usuarios.
  + suppliers\_insert: Permite INSERT a todos los usuarios.
  + suppliers\_update: Permite UPDATE a todos los usuarios.
  + suppliers\_delete: Permite DELETE a todos los usuarios.
* **PRODUCTS:**
  + products\_select: Permite SELECT a todos los usuarios.
  + products\_insert: Permite INSERT a todos los usuarios.
  + products\_update: Permite UPDATE a todos los usuarios.
  + products\_delete: Permite DELETE a todos los usuarios.
* **PRODUCT\_RECEPTIONS:**
  + receptions\_select: Permite SELECT a todos los usuarios.
  + receptions\_insert: Permite INSERT a todos los usuarios.
  + receptions\_update: Permite UPDATE a todos los usuarios.
  + receptions\_delete: Permite DELETE a todos los usuarios.

**Índices y Optimización**

**Índices Implementados**:

1. **Búsquedas por identificación**: idx\_suppliers\_id\_number.
2. **Filtros por estado**: idx\_suppliers\_status, idx\_products\_status.
3. **Búsquedas por código**: idx\_products\_code.
4. **Consultas de recepciones**: idx\_receptions\_product\_id, idx\_receptions\_supplier\_id.
5. **Filtros por fecha**: idx\_receptions\_date.

**Recomendaciones Adicionales (Ejemplo de Índices)**:

-- Índices adicionales para consultas específicas

CREATE INDEX idx\_receptions\_batch\_number ON product\_receptions(batch\_number);

CREATE INDEX idx\_receptions\_expiration\_date ON product\_receptions(expiration\_date);

CREATE INDEX idx\_products\_laboratory ON products(laboratory);

CREATE INDEX idx\_suppliers\_name ON suppliers(name);

**Triggers y Funciones Automáticas**

**Actualización Automática de Timestamps**

-- Función para actualizar updated\_at automáticamente

CREATE OR REPLACE FUNCTION update\_updated\_at\_column()

RETURNS TRIGGER AS $$

BEGIN

NEW.updated\_at = NOW();

RETURN NEW;

END;

$$ language 'plpgsql';

-- Triggers para cada tabla

CREATE TRIGGER update\_suppliers\_updated\_at

BEFORE UPDATE ON suppliers

FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();

CREATE TRIGGER update\_products\_updated\_at

BEFORE UPDATE ON products

FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();

CREATE TRIGGER update\_receptions\_updated\_at

BEFORE UPDATE ON product\_receptions

FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();

**Consultas de Ejemplo**

**Consultas Básicas**

-- Obtener todos los proveedores activos

SELECT \* FROM suppliers WHERE status = 'active';

-- Productos por laboratorio

SELECT laboratory, COUNT(\*) as total\_products

FROM products

WHERE status = 'active'

GROUP BY laboratory;

-- Recepciones del último mes

SELECT pr.\*, p.name as product\_name, s.name as supplier\_name

FROM product\_receptions pr

JOIN products p ON pr.product\_id = p.id

JOIN suppliers s ON pr.supplier\_id = s.id

WHERE pr.reception\_date >= NOW() - INTERVAL '1 month';

**Consultas Avanzadas**

-- Productos próximos a vencer (30 días)

SELECT p.name, pr.batch\_number, pr.expiration\_date, pr.quantity

FROM product\_receptions pr

JOIN products p ON pr.product\_id = p.id

WHERE pr.expiration\_date <= CURRENT\_DATE + INTERVAL '30 days'

AND pr.expiration\_date > CURRENT\_DATE

ORDER BY pr.expiration\_date;

-- Proveedores con más recepciones

SELECT s.name, COUNT(pr.id) as total\_receptions, SUM(pr.quantity) as total\_quantity

FROM suppliers s

LEFT JOIN product\_receptions pr ON s.id = pr.supplier\_id

GROUP BY s.id, s.name

ORDER BY total\_receptions DESC;