**1. Resumen General**

MediInventory es un sistema integral para la gestión de inventario farmacéutico construido con tecnologías modernas. Diseñado específicamente para cumplir con los estándares de la industria farmacéutica colombiana, el sistema garantiza trazabilidad completa, registro de códigos INVIMA y estricto control de calidad. Su arquitectura modular abarca los siguientes módulos principales:

* **Gestión de Proveedores:** Registro completo de proveedores (tipo y número de documento, nombre, contacto, actividad económica), búsqueda avanzada y control de estado (activo/inactivo).
* **Catálogo de Productos:** Mantenimiento de productos farmacéuticos con código único, descripción detallada, laboratorio y estado (activo/inactivo). Incluye búsqueda inteligente y CRUD completo con validaciones.
* **Recepción de Productos:** Registro de entradas de inventario con trazabilidad (fecha, producto, proveedor, factura, lote, registro INVIMA, fecha de vencimiento y estado de calidad). Permite añadir observaciones adicionales y enlazar automáticamente con el proveedor y el producto correspondiente.

El sistema fue desarrollado con Next.js (React) en el frontend y Supabase (PostgreSQL) en el backend, aplicando principios de seguridad como Row Level Security (RLS) en todas las tablas. Además, cuenta con una interfaz responsiva (Mobile First) y una estructura de componentes reutilizables que optimizan la experiencia de usuario. MediInventory incluye mecanismos de validación de datos (frontend con Zod y React Hook Form; backend con restricciones en la base de datos) para prevenir inconsistencias y garantizar integridad.

A continuación se presentan secciones detalladas de la documentación de la API, la guía de instalación, la guía de usuario, la documentación técnica y el modelo de base de datos.

**2. Documentación de API**

MediInventory utiliza Supabase como backend, proporcionando una API REST automática basada en el esquema de PostgreSQL. La comunicación se realiza a través de los clientes de Supabase para JavaScript.

**Arquitectura de API**

La arquitectura de la API se apoya en los clientes de Supabase para manejar la comunicación con la base de datos de PostgreSQL. Se utilizan dos configuraciones principales:

**Cliente del Navegador**

// lib/supabase/client.ts

import { createBrowserClient } from '@supabase/ssr'

export const createClient = () =>

createBrowserClient(

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

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

)

Cliente del Servidor

// lib/supabase/server.ts

import { createServerClient } from '@supabase/ssr'

import { cookies } from 'next/headers'

export const createClient = () => {

const cookieStore = cookies()

return createServerClient(

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

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

{

cookies: {

get(name: string) {

return cookieStore.get(name)?.value

},

},

}

)

}

**Estructura de Servicios**

**Cliente del Navegador y Servidor**

Como se muestra en la configuración anterior, se crea un cliente de Supabase tanto para el navegador como para el servidor, lo que permite ejecutar operaciones de la base de datos en ambos entornos. Esto posibilita llamadas de datos en el frontend y en componentes del servidor (Server Components en Next.js).

**API de Proveedores (Suppliers)**

**Endpoints Disponibles**

GET - Obtener todos los proveedores

const { data, error } = await supabase

.from('suppliers')

.select('\*')

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

**Respuesta Ejemplo:**

[

{

"id": "uuid",

"id\_type": "NIT",

"id\_number": "900123456-1",

"name": "Laboratorios ABC S.A.S.",

"address": "Calle 123 #45-67, Bogotá",

"phone": "+57 1 234-5678",

"email": "contacto@lababc.com",

"contact\_person": "María González",

"economic\_activity": ["4773"],

"status": "active",

"created\_at": "2024-01-01T00:00:00Z",

"updated\_at": "2024-01-01T00:00:00Z"

}

]

GET - Obtener proveedor por ID

const { data, error } = await supabase

.from('suppliers')

.select('\*')

.eq('id', supplierId)

.single()

POST - Crear nuevo proveedor

const { data, error } = await supabase

.from('suppliers')

.insert({

id\_type: 'NIT',

id\_number: '900123456-1',

name: 'Nuevo Proveedor',

address: 'Dirección',

phone: '+57 1 234-5678',

email: 'contacto@proveedor.com',

contact\_person: 'Persona de Contacto',

economic\_activity: ['4773'],

status: 'active'

})

.select()

.single()

PUT - Actualizar proveedor

const { data, error } = await supabase

.from('suppliers')

.update({

name: 'Nombre Actualizado',

phone: '+57 1 999-8888'

})

.eq('id', supplierId)

.select()

.single()

DELETE - Eliminar proveedor

const { error } = await supabase

.from('suppliers')

.delete()

.eq('id', supplierId)

**Filtros y Búsquedas en Proveedores**

**Filtrar por estado**

const { data, error } = await supabase

.from('suppliers')

.select('\*')

.eq('status', 'active')

**Búsqueda por nombre**

const { data, error } = await supabase

.from('suppliers')

.select('\*')

.ilike('name', `%${searchTerm}%`)

**Búsqueda por documento**

const { data, error } = await supabase

.from('suppliers')

.select('\*')

.ilike('id\_number', `%${searchTerm}%`)

**Búsqueda combinada (nombre o documento)**

const { data, error } = await supabase

.from('suppliers')

.select('\*')

.or(`name.ilike.%${searchTerm}%,id\_number.ilike.%${searchTerm}%`)

**💊 API de Productos (Products)**

**Endpoints Disponibles**

GET - Obtener todos los productos

const { data, error } = await supabase

.from('products')

.select('\*')

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

**Respuesta Ejemplo:**

[

{

"id": "uuid",

"code": "MED001",

"name": "Acetaminofén 500mg",

"description": "Analgésico y antipirético en tabletas",

"laboratory": "Laboratorios ABC",

"status": "active",

"created\_at": "2024-01-01T00:00:00Z",

"updated\_at": "2024-01-01T00:00:00Z"

}

]

**POST - Crear nuevo producto**

const { data, error } = await supabase

.from('products')

.insert({

code: 'MED006',

name: 'Nuevo Medicamento',

description: 'Descripción del medicamento',

laboratory: 'Laboratorio XYZ',

status: 'active'

})

.select().single()

**Búsquedas Específicas en Productos**

**Búsqueda por código**

const { data, error } = await supabase

.from('products')

.select('\*')

.eq('code', productCode)

.single()

Búsqueda múltiple (por código, nombre o laboratorio)

const { data, error } = await supabase

.from('products')

.select('\*')

.or(`code.ilike.%${term}%,name.ilike.%${term}%,laboratory.ilike.%${term}%`)

**API de Recepciones (Product Receptions)**

**Endpoints con Relaciones**

GET - Obtener recepciones con datos relacionados

const { data, error } = await supabase

.from('product\_receptions')

.select(`

\*,

product:products(\*),

supplier:suppliers(\*)

`)

.order('reception\_date', { ascending: false })

**Respuesta Ejemplo:**

[

{

"id": "uuid",

"reception\_date": "2024-01-15T10:30:00Z",

"product\_id": "product-uuid",

"supplier\_id": "supplier-uuid",

"invoice\_number": "FAC-001234",

"quantity": 100,

"batch\_number": "LOTE-2024-001",

"invima\_registration": "INVIMA-123456",

"expiration\_date": "2026-01-15",

"presentation\_state": "bueno",

"notes": "Recepción normal",

"created\_at": "2024-01-15T10:30:00Z",

"updated\_at": "2024-01-15T10:30:00Z",

"product": {

"id": "product-uuid",

"code": "MED001",

"name": "Acetaminofén 500mg",

"laboratory": "Laboratorios ABC"

},

"supplier": {

"id": "supplier-uuid",

"name": "Laboratorios ABC S.A.S.",

"id\_number": "900123456-1"

}

}

]

**POST - Crear nueva recepción**

const { data, error } = await supabase

.from('product\_receptions')

.insert({

reception\_date: new Date().toISOString(),

product\_id: 'product-uuid',

supplier\_id: 'supplier-uuid',

invoice\_number: 'FAC-001235',

quantity: 50,

batch\_number: 'LOTE-2024-002',

invima\_registration: 'INVIMA-789012',

expiration\_date: '2026-06-15',

presentation\_state: 'bueno',

notes: 'Recepción sin observaciones'

})

.select(`

\*,

product:products(\*),

supplier:suppliers(\*)

`)

.single()

**Consultas Avanzadas en Recepciones**

* **Recepciones por rango de fechas**

const { data, error } = await supabase

.from('product\_receptions')

.select('\*')

.gte('reception\_date', startDate)

.lte('reception\_date', endDate)

Recepciones por estado de presentación

const { data, error } = await supabase

.from('product\_receptions')

.select('\*')

.eq('presentation\_state', 'bueno')

Productos próximos a vencer

const { data, error } = await supabase

.from('product\_receptions')

.select(`

\*,

product:products(\*)

`)

.lte('expiration\_date', futureDate)

.gte('expiration\_date', new Date().toISOString().split('T')[0])

**Consultas Complejas**

**Agregaciones y Estadísticas**

* **Contar recepciones por proveedor**

const { data, error } = await supabase

.from('product\_receptions')

.select('supplier\_id, supplier:suppliers(name)')

.then(result => {

// Agrupar en el cliente

const grouped = result.data?.reduce((acc, item) => {

const supplierId = item.supplier\_id

acc[supplierId] = (acc[supplierId] || 0) + 1

return acc

}, {})

return grouped

})

* **Productos más recibidos**

const { data, error } = await supabase

.from('product\_receptions')

.select('product\_id, quantity, product:products(name)')

.then(result => {

// Agrupar y sumar cantidades

const grouped = result.data?.reduce((acc, item) => {

const productId = item.product\_id

if (!acc[productId]) {

acc[productId] = {

product: item.product,

totalQuantity: 0

}

}

acc[productId].totalQuantity += item.quantity

return acc

}, {})

return Object.values(grouped)

})

**Seguridad y Autenticación**

**Todas las tablas tienen RLS habilitado con políticas permisivas:**

**Ejemplo de política en Supabase**

CREATE POLICY "suppliers\_select" ON suppliers

FOR SELECT USING (true);

CREATE POLICY "suppliers\_insert" ON suppliers

FOR INSERT WITH CHECK (true);

**Validación de Datos**

**Validación en el Cliente (ejemplo usando Zod):**

import { z } from 'zod'

const supplierSchema = z.object({

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

id\_number: z.string().min(1),

name: z.string().min(1),

email: z.string().email().optional().or(z.literal(''))

})

// Validar antes de enviar

const validatedData = supplierSchema.parse(formData)

Constraints de Base de Datos:

CHECK (id\_type IN ('CC', 'NIT', 'CE', 'PP'))

CHECK (status IN ('active', 'inactive'))

CHECK (quantity > 0)

CHECK (presentation\_state IN ('bueno', 'regular', 'malo'))

**3. Guía de Instalación**

Esta sección describe los pasos necesarios para desplegar el módulo web de MediInventory y configurarlo adecuadamente.

**Requisitos Previos**

**Software Necesario**

* **Node.js (versión 18.0 o superior)**
* **npm o yarn (gestor de paquetes)**
* **Git (para clonar el repositorio)**
* **Navegador web moderno (Chrome, Firefox, Safari, Edge)**

**Servicios en la Nube**

* **Cuenta de Vercel (para despliegue)**
* **Cuenta de Supabase (para base de datos)**

**🚀 Instalación Paso a Paso**

**Paso 1: Preparación del Entorno**

* Verificar Node.js

node --version

npm --version

* Clonar o Descargar el Proyecto

git clone [URL\_DEL\_REPOSITORIO]

cd medicamentos-web-module

**Paso 2: Configuración de la Base de Datos (Supabase)**

1. **Crear Proyecto en Supabase**
   * Ir a supabase.com
   * Crear cuenta o iniciar sesión
   * Hacer clic en "New Project"
   * Completar los datos (Name: *MediInventory*, Password seguro, Región)
   * Esperar 2-3 minutos a que se cree el proyecto
2. **Obtener Credenciales de Supabase**
   * En el dashboard de Supabase, ir a Settings → API
   * Copiar las siguientes credenciales:
     + Project URL
     + anon public key
     + service\_role key (mantener privada)
3. **Ejecutar Scripts de Base de Datos**
   * En Supabase, ir a SQL Editor
   * Ejecutar los scripts en orden:
     + Script 1: Crear Tablas
     + -- Copiar y pegar el contenido de scripts/001\_create\_tables.sql
     + -- Hacer clic en "Run"
     + Script 2: Datos de Prueba
     + -- Copiar y pegar el contenido de scripts/002\_seed\_data.sql
     + -- Hacer clic en "Run"
     + Script 3: Actualización de Actividades Económicas
     + -- Copiar y pegar el contenido de scripts/003\_update\_economic\_activity.sql
     + -- Hacer clic en "Run"

**Paso 3: Configuración del Proyecto**

* Instalar Dependencias

npm install

yarn install

* Configurar Variables de Entorno

Crear archivo .env.local en la raíz del proyecto con contenido similar a:

* NEXT\_PUBLIC\_SUPABASE\_URL=tu\_project\_url\_aqui
* NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=tu\_anon\_key\_aqui
* SUPABASE\_SERVICE\_ROLE\_KEY=tu\_service\_role\_key\_aqui
* POSTGRES\_URL=postgresql://[usuario]:[password]@[host]:[puerto]/[database]?pgbouncer=true&connect\_timeout=15
* POSTGRES\_PRISMA\_URL=postgresql://[usuario]:[password]@[host]:[puerto]/[database]?pgbouncer=true&connect\_timeout=15
* POSTGRES\_URL\_NON\_POOLING=postgresql://[usuario]:[password]@[host]:[puerto]/[database]

**Para obtener las URLs de PostgreSQL:**

* 1. **En Supabase, ir a Settings → Database.**
  2. **Copiar la Connection string y adaptarla a las variables correspondientes.**

**Paso 4: Desarrollo Local**

**Ejecutar en Modo Desarrollo**

* npm run dev
* yarn dev

1. **Verificar Funcionamiento**
   * Abrir el navegador en http://localhost:3000.
   * Verificar que la aplicación carga correctamente.
   * Probar las funcionalidades principales:
     + Gestión de Proveedores
     + Gestión de Productos
     + Gestión de Recepciones

**Paso 5: Despliegue en Producción (Vercel)**

1. **Preparar para Despliegue**

* npm run build
* npm run start

**Desplegar en Vercel**

**Opción A: Desde Vercel Dashboard**

* + Ir a vercel.com.
  + Hacer clic en "New Project".
  + Importar repositorio de Git.
  + Configurar variables de entorno.
  + Hacer clic en "Deploy".

1. **Configurar Variables de Entorno en Vercel**
   * En el proyecto de Vercel, ir a Settings → Environment Variables.
   * Agregar todas las variables definidas en el archivo .env.local.
   * Marcar las variables públicas (NEXT\_PUBLIC\_\*) para Production, Preview y Development.

**Paso 6: Configuración Post-Despliegue**

1. **Verificar Conexión a Base de Datos**
   * Acceder a la URL de producción del módulo web.
   * Verificar que los datos se cargan correctamente.
   * Probar operaciones CRUD (crear, leer, actualizar, eliminar).
2. **Configurar Dominio Personalizado (Opcional)**
   * En Vercel, ir a Settings → Domains.
   * Agregar dominio personalizado.
   * Configurar DNS según las instrucciones de Vercel.