ROSSI INVENTORY SYSTEM - PROJECT STATUS MCP

Documento de Estado del Proyecto | Model Context Protocol
Empresa: Rossi
Sistema: Inventario Integral
Fecha: Agosto 2025
Estado: 75% Completado - Backend Implementado

---

1. ESTADO ACTUAL DEL PROYECTO

📊 Resumen Ejecutivo

- **Progreso General:** ✅ 75% Completado
- **Estado Backend:** ✅ Completamente Implementado
- **Estado Frontend:** ⏳ Pendiente (25% restante)
- **Base de Datos:** ✅ Schema completo con 15+ tablas
- **APIs REST:** ✅ Totalmente funcionales
- **Autenticación:** ✅ Sistema robusto con NextAuth

📈 Métricas de Desarrollo

- **Módulos Core:** 8/8 implementados
- **Endpoints API:** 40+ funcionales
- **Sistema de Roles:** 3 niveles implementados
- **Utilidades Avanzadas:** 5/5 completadas
- **Datos de Prueba:** Poblados completamente

---

2. ARQUITECTURA Y ESTRUCTURA

🛠️ Stack Tecnológico

```
FRONTEND (Pendiente)
├── Next.js 15.4.6
├── React 18.3.1
├── TypeScript 5.7.2
├── TailwindCSS 3.4.16
├── Headless UI 2.3.0
└── Heroicons 2.2.0

BACKEND (Completado ✅)
├── Next.js API Routes
├── NextAuth 4.24.17
├── Prisma ORM 6.13.0
├── PostgreSQL
└── TypeScript

UTILIDADES
├── PDF Generation (pdf-lib)
├── QR Codes (qrcode)
├── Email (nodemailer)
├── Encryption (bcrypt)
└── Validaciones personalizadas
```

📁 Estructura de Carpetas

```
/rossi-inventory/
├── src/
│   ├── lib/                    # Utilidades core ✅
│   │   ├── auth.ts            # Autenticación NextAuth
│   │   ├── db.ts              # Conexión Prisma
│   │   ├── permissions.ts     # Sistema de permisos
│   │   ├── audit.ts           # Auditoría de acciones
│   │   ├── pdf.ts             # Generación PDFs
│   │   ├── qr.ts              # Códigos QR
│   │   ├── email.ts           # Sistema de emails
│   │   └── utils.ts           # Utilidades generales
│   └── app/                   # App Router (pendiente)
├── prisma/
│   ├── schema.prisma          # Schema completo ✅
│   └── seed.ts                # Datos de prueba ✅
└── pages/api/                 # API Routes ✅
```

🗄️ Base de Datos - Schema PostgreSQL

15+ Tablas Interconectadas:

- `users` - Usuarios del sistema
- `suppliers` - Proveedores
- `products` - Catálogo de productos
- `product_types` - Tipos de productos
- `warehouses` - Almacenes
- `employees` - Empleados
- `payment_types` - Tipos de pago
- `purchase_orders` - Órdenes de compra
- `purchase_order_items` - Ítems de órdenes

- `inventory_batches` - Lotes de inventario
- `inventory_bulk` - Inventario granel
- `audit_logs` - Registro de auditoría
- Y más tablas de relaciones...

---

3. MÓDULOS IMPLEMENTADOS

✅ APIs REST Completadas

🏢 SUPPLIERS (Proveedores)

- `GET /api/suppliers` - Listar proveedores
- `POST /api/suppliers` - Crear proveedor
- `PUT /api/suppliers/[id]` - Actualizar proveedor
- `DELETE /api/suppliers/[id]` - Eliminar proveedor
- **Funcionalidades:** CRUD completo, validaciones, auditoría

📦 PRODUCTS (Productos)

- `GET /api/products` - Listar productos
- `POST /api/products` - Crear producto
- `PUT /api/products/[id]` - Actualizar producto
- `DELETE /api/products/[id]` - Eliminar producto
- **Funcionalidades:** Gestión de catálogo, códigos QR automáticos

🏷️ PRODUCT TYPES (Tipos de Producto)

- `GET /api/product-types` - Listar tipos
- `POST /api/product-types` - Crear tipo
- `PUT /api/product-types/[id]` - Actualizar tipo
- `DELETE /api/product-types/[id]` - Eliminar tipo
- **Funcionalidades:** Categorización de productos

🏭 WAREHOUSES (Almacenes)

- `GET /api/warehouses` - Listar almacenes
- `POST /api/warehouses` - Crear almacén
- `PUT /api/warehouses/[id]` - Actualizar almacén
- `DELETE /api/warehouses/[id]` - Eliminar almacén
- **Funcionalidades:** Gestión de ubicaciones, capacidades

👥 EMPLOYEES (Empleados)

- `GET /api/employees` - Listar empleados
- `POST /api/employees` - Crear empleado
- `PUT /api/employees/[id]` - Actualizar empleado
- `DELETE /api/employees/[id]` - Eliminar empleado
- **Funcionalidades:** Gestión de personal, roles

💳 PAYMENT TYPES (Tipos de Pago)

- `GET /api/payment-types` - Listar tipos de pago
- `POST /api/payment-types` - Crear tipo de pago
- `PUT /api/payment-types/[id]` - Actualizar tipo

- `DELETE /api/payment-types/[id]` - Eliminar tipo
- **Funcionalidades:** Configuración métodos de pago

📋 PURCHASE ORDERS (Órdenes de Compra)

- `GET /api/purchase-orders` - Listar órdenes
- `POST /api/purchase-orders` - Crear orden
- `PUT /api/purchase-orders/[id]` - Actualizar orden
- `PUT /api/purchase-orders/[id]/status` - Cambiar estado
- **Estados:** Pre-orden → Emitida → Recibida → Pagada
- **Funcionalidades:** Flujo completo, ítems, totales, auditoría

📊 INVENTORY (Inventario)

- `GET /api/inventory/batches` - Inventario por lotes
- `GET /api/inventory/bulk` - Inventario granel
- `POST /api/inventory/batches` - Crear lote
- `POST /api/inventory/bulk` - Crear granel
- `PUT /api/inventory/batches/[id]` - Actualizar lote
- **Funcionalidades:** Control por lotes, códigos QR, trazabilidad

---

4. SISTEMA DE AUTENTICACIÓN

🔐 Tecnología: NextAuth.js

- **Adaptador:** Prisma
- **Estrategia:** Credenciales + Base de datos
- **Encriptación:** bcrypt para contraseñas
- **Sesiones:** JWT + Database sessions

👤 Roles Implementados

Rol	Permisos	Funcionalidades
Super Admin	Acceso total	Gestión completa del sistema, usuarios, configuraciones
Admin	Gestión operativa	Órdenes, inventario, reportes, empleados
Warehouse	Operaciones almacén	Inventario, recepción, movimientos

🛡️ Sistema de Permisos Granular

```
// Ejemplo estructura permisos
permissions: {
  suppliers: { create: true, read: true, update: true, delete: true },
  inventory: { create: true, read: true, update: true, delete: false },
  orders: { create: true, read: true, update: true, delete: false }
}
```

🔑 Credenciales de Prueba

- **Super Admin:** `admin@rossi.com` / `admin123`
- **Warehouse:** `almacen@rossi.com` / `admin123`

---

5. UTILIDADES AVANZADAS

📄 Generación de PDFs (`src/lib/pdf.ts`)

- **Librería:** pdf-lib 1.17.1
- **Funcionalidades:**
- Órdenes de compra en PDF
- Reportes de inventario
- Etiquetas para lotes
- Templates personalizados

📱 Códigos QR (`src/lib/qr.ts`)

- **Librería:** qrcode 1.5.5
- **Funcionalidades:**
- Generación automática para lotes
- QR para productos
- Códigos de seguimiento
- Integración con inventario

📧 Sistema de Emails (`src/lib/email.ts`)

- **Librería:** nodemailer 6.9.20
- **Funcionalidades:**
- Notificaciones de órdenes
- Alertas de inventario bajo
- Confirmaciones de operaciones
- Templates HTML

✅ Validaciones (`src/lib/utils.ts`)

- Validación de formularios
- Sanitización de datos
- Formateo de fechas y números
- Validaciones de negocio

📊 Auditoría (`src/lib/audit.ts`)

- **Tabla:** `audit_logs`
- **Tracking:** Todas las acciones críticas
- **Datos:** Usuario, acción, timestamp, detalles
- **Reportes:** Trazabilidad completa

---

6. DATOS DE DEMOSTRACIÓN

🌱 Seed Data Poblado (`prisma/seed.ts`)

Usuarios de Prueba:
- 1 Super Admin
- 1 Usuario Warehouse
- Usuarios adicionales para testing

Catálogos Base:
- 10+ Proveedores de ejemplo
- 20+ Productos con códigos QR
- 5+ Tipos de productos
- 3+ Almacenes configurados
- 10+ Empleados
- Tipos de pago estándar

Datos Operativos:
- Órdenes de compra en diferentes estados
- Lotes de inventario con trazabilidad
- Inventario granel
- Registros de auditoría

Comando para poblar:

```
npm run db:seed
```

---

7. PASOS FALTANTES ORGANIZADOS POR PRIORIDAD

🔴 ALTA PRIORIDAD - Crítico para Funcionalidad Básica

Frontend Core (Estimado: 2-3 semanas)

1. **Dashboard Principal**
   - Layout base con navegación
   - Resumen ejecutivo con métricas
   - Gráficos de inventario y órdenes

2. **Páginas de Catálogos**
   - Lista de proveedores con CRUD
   - Catálogo de productos con búsqueda

- Gestión de almacenes
- Gestión de empleados

3. **Módulo de Órdenes de Compra**
   - Lista de órdenes con filtros
   - Formulario de creación/edición
   - Vista detalle con ítems
   - Cambio de estados

4. **Módulo de Inventario**
   - Vista de lotes con códigos QR
   - Inventario granel
   - Movimientos y ajustes
   - Búsqueda y filtros

5. **Autenticación UI**
   - Página de login
   - Gestión de sesiones
   - Control de acceso por roles

🟡 MEDIA PRIORIDAD - Mejoras Importantes

Módulos Adicionales (Estimado: 2-3 semanas)

1. **Payments Completo**
   - API para registrar pagos
   - Frontend para gestión de pagos
   - Reportes financieros
   - Conciliación con órdenes

2. **Production Module**
   - APIs para órdenes de producción
   - Control de materias primas
   - Productos terminados
   - Frontend para operarios

3. **Notifications System**
   - Sistema de alertas en tiempo real
   - Notificaciones push
   - Configuración de alertas por usuario
   - Dashboard de notificaciones

4. **Reportes y Analytics**
   - Reportes de inventario
   - Análisis de compras
   - KPIs operativos
   - Exportación a Excel/PDF

🟢 BAJA PRIORIDAD - Refinamientos

Mejoras y Optimizaciones (Estimado: 1-2 semanas)

1. **UX/UI Enhancements**
   - Tema dark mode
   - Responsive design completo

- Animaciones y transiciones
- Componentes reutilizables

2. **Performance & SEO**
   - Optimización de consultas
   - Caching strategies
   - Image optimization
   - Bundle size optimization

3. **Testing & Quality**
   - Unit tests para APIs
   - Integration tests
   - E2E testing
   - Error handling improvement

4. **Advanced Features**
   - Import/Export data
   - Multi-idioma (i18n)
   - Advanced search
   - Bulk operations

---

8. ROADMAP DE DESARROLLO

🗓️ Próximos Hitos

Milestone 1: Frontend Core (3-4 semanas)

- **Semana 1-2:** Layout, navegación, páginas básicas
- **Semana 3:** Integración con APIs, formularios
- **Semana 4:** Testing, refinamientos, deploy

Milestone 2: Módulos Avanzados (2-3 semanas)

- **Semana 5-6:** Payments y Production modules
- **Semana 7:** Notifications y reportes

Milestone 3: Production Ready (1-2 semanas)

- **Semana 8:** Testing completo, optimizaciones
- **Semana 9:** Deploy producción, documentación

📊 Estimación Total de Completitud

- **Actual:** 75% ✅
- **Post Milestone 1:** 90%
- **Post Milestone 2:** 95%
- **Post Milestone 3:** 100% 🎯

---

9. INSTRUCCIONES TÉCNICAS

🚀 Setup del Ambiente de Desarrollo

Requisitos Previos

- Node.js 18+ ✅
- PostgreSQL ✅
- npm/yarn ✅

Comandos Importantes

```
# Instalar dependencias
npm install

# Ejecutar en desarrollo
npm run dev

# Poblar base de datos
npm run db:seed

# Generar cliente Prisma
npx prisma generate

# Migraciones
npx prisma db push

# Ver base de datos
npx prisma studio
```

URLs de Desarrollo

- **App:** http://localhost:3000
- **Prisma Studio:** http://localhost:5555

🔧 Continuación del Desarrollo

Estructura Recomendada para Frontend

```
src/app/
├── (auth)/
│   └── login/page.tsx
├── dashboard/
│   └── page.tsx
├── suppliers/
│   ├── page.tsx
│   └── [id]/page.tsx
├── products/
│   ├── page.tsx
│   └── [id]/page.tsx
├── inventory/
│   ├── batches/page.tsx
│   └── bulk/page.tsx
└── orders/
    ├── page.tsx
    └── [id]/page.tsx
```

Componentes Recomendados

```
src/components/
├── ui/          # Componentes base (Button, Input, etc.)
├── forms/       # Formularios específicos
├── tables/      # Tablas de datos
├── charts/      # Gráficos y métricas
└── layout/      # Layout y navegación
```

📝 Patterns de Desarrollo

- **App Router de Next.js 13+**
- **Server Components por defecto**
- **Client Components solo cuando necesario**
- **Hooks personalizados para lógica**
- **TypeScript estricto**
- **TailwindCSS para estilos**

---

10. CREDENCIALES Y ACCESOS

🔑 Usuarios de Prueba

Usuario	Email	Contraseña	Rol	Permisos
Admin Principal	`admin@rossi.com`	`admin123`	Super Admin	Acceso total
Usuario Almacén	`alma-cen@rossi.com`	`admin123`	Warehouse	Operaciones inventario

🗄️ Base de Datos

- **Host:** localhost
- **Puerto:** 5432
- **Base:** rossi_inventory
- **Usuario:** Configurado en `.env`

📂 Configuración Requerida (.env.local)

```
# Database
DATABASE_URL="postgresql://..."

# NextAuth
NEXTAUTH_SECRET="your-secret-key"
NEXTAUTH_URL="http://localhost:3000"

# Email (para nodemailer)
EMAIL_SERVER_HOST=smtp.gmail.com
EMAIL_SERVER_PORT=587
EMAIL_SERVER_USER=your-email@gmail.com
EMAIL_SERVER_PASSWORD=your-app-password
EMAIL_FROM=your-email@gmail.com
```

🔒 Seguridad

- Contraseñas encriptadas con bcrypt
- Sesiones seguras con NextAuth
- Validación de permisos en cada endpoint
- Logs de auditoría para acciones críticas

📞 CONTACTO Y SOPORTE

Estado del Proyecto: 75% Completado - Backend Funcional
Próximo Paso: Implementación Frontend React/NextJS
Tiempo Estimado para Completitud: 6-8 semanas adicionales

Último Actualizado: Agosto 2025
Documento Versión: 1.0
Formato: Model Context Protocol (MCP)

Este documento MCP contiene toda la información necesaria para continuar el desarrollo del Sistema de Inventario Rossi. El backend está completamente funcional y probado, listo para la implementación del frontend.