🤖 INSTRUCCIONES ESPECÍFICAS PARA DEEPAGENT

Guía paso a paso para continuar desarrollo

⚠️ REGLAS CRÍTICAS - LEER PRIMERO

🚨 LO QUE NUNCA DEBE HACER DEEPAGENT:

```
❌ NUNCA ejecutar create-next-app
❌ NUNCA crear un nuevo proyecto NextJS
❌ NUNCA eliminar el directorio /home/ubuntu/sistema_erp_completo
❌ NUNCA usar npm (solo yarn)
❌ NUNCA modificar completamente .env
❌ NUNCA cambiar DATABASE_URL
❌ NUNCA hacer rm -rf en el proyecto
```

✅ LO QUE SIEMPRE DEBE HACER DEEPAGENT:

```
✅ SIEMPRE trabajar en /home/ubuntu/sistema_erp_completo/app/
✅ SIEMPRE usar yarn como package manager
✅ SIEMPRE leer archivos existentes antes de modificar
✅ SIEMPRE ejecutar yarn prisma db seed después de cambios DB
✅ SIEMPRE usar test_nextjs_project para probar
✅ SIEMPRE guardar checkpoint con build_and_save_nextjs_project_checkpoint
```

🎯 WORKFLOW OBLIGATORIO PARA DEEPAGENT

Paso 1: Verificación Inicial

```
# Confirmar ubicación del proyecto
cd /home/ubuntu/sistema_erp_completo/app
pwd  # Debe mostrar: /home/ubuntu/sistema_erp_completo/app

# Verificar estructura
ls -la  # Debe mostrar: app/, components/, lib/, prisma/, etc.
```

Paso 2: Inicialización (Si es necesario)

```
# Solo si no están instaladas las dependencias
yarn install

# Generar cliente Prisma
yarn prisma generate

# Cargar datos de prueba (CRÍTICO)
yarn prisma db seed
```

Paso 3: Desarrollo

```
# Iniciar servidor desarrollo
yarn dev  # http://localhost:3000

# En otra terminal para cambios
# Hacer modificaciones según requerimientos del usuario
```

Paso 4: Testing (OBLIGATORIO)

```
# Probar proyecto con herramienta DeepAgent
test_nextjs_project /home/ubuntu/sistema_erp_completo

# Si falla, revisar errores y corregir
# Si pasa, continuar al paso 5
```

Paso 5: Finalización (OBLIGATORIO)

```
# Guardar checkpoint
build_and_save_nextjs_project_checkpoint /home/ubuntu/sistema_erp_completo "Descrip-
ción breve del cambio"
```

📝 PLANTILLAS DE RESPUESTA PARA DEEPAGENT

Al Iniciar una Tarea:

```
Perfecto, voy a [descripción de la tarea] en el sistema ERP existente.

Primero, déjame verificar el estado actual del proyecto:
```

Durante el Desarrollo:

```
Estoy trabajando en el sistema ERP ubicado en /home/ubuntu/sistema_erp_completo/app/

He [descripción de lo que hizo]

Ahora voy a [próximo paso]
```

Al Completar una Tarea:

```
✅ He completado [descripción de la tarea]

El sistema ERP ahora incluye:
- [Lista de funcionalidades añadidas/modificadas]

Voy a probar el proyecto para asegurar que todo funcione correctamente.
```

🛠️ COMANDOS ÚTILES POR MÓDULO

Para Módulo de Clientes:

```
# Ver clientes en base de datos
yarn prisma studio
# Navegar a tabla 'Cliente'

# Verificar APIs de clientes
curl -X GET "http://localhost:3000/api/clientes?page=1&limit=10"
```

Para Cobranza Móvil:

```
# Verificar service worker
curl -I http://localhost:3000/sw.js

# Probar búsqueda de clientes offline
curl -X GET "http://localhost:3000/api/cobranza-movil/clientes/search?q=Roberto"
```

Para Productos:

```
# Verificar productos
curl -X GET "http://localhost:3000/api/productos"

# Verificar categorías
curl -X GET "http://localhost:3000/api/productos/categorias"
```

Para Autenticación:

```
# Verificar usuarios
yarn prisma studio
# Navegar a tabla 'User'

# Credenciales de prueba siempre disponibles:
# admin@sistema.com / 123456
```

🔧 RESOLUCIÓN DE PROBLEMAS COMUNES

Error: “Prisma Client not found”

```
cd /home/ubuntu/sistema_erp_completo/app
yarn prisma generate
yarn dev
```

Error: “Database connection failed”

```
# Verificar variables de entorno
cat .env
# Debe contener DATABASE_URL correcto

# Probar conexión
yarn prisma db push
```

Error: “No users found” en login

```
# Recargar datos de prueba
yarn prisma db seed

# Verificar en Prisma Studio
yarn prisma studio
```

Error: Build failed

```
# Limpiar y reinstalar
rm -rf node_modules yarn.lock
yarn install
yarn build
```

PWA no funciona offline

```
# Verificar service worker
ls -la public/sw.js
# Debe existir

# Verificar manifest
ls -la public/manifest.json
# Debe existir
```

📱 FUNCIONALIDADES IMPLEMENTADAS POR MÓDULO

✅ Autenticación (100%)

- Login/logout funcionando
- Roles: SUPERADMIN, ADMIN, ANALISTA, GESTOR, CLIENTE, VENTAS
- Protección de rutas
- Sesiones persistentes

✅ Clientes (100%)

- CRUD completo
- Búsqueda avanzada
- Importación CSV
- Asignación gestores/vendedores
- Modal detalles

- Vista tarjetas/tabla

✅ Cobranza Móvil (100%)

- Funcionamiento offline total
- Búsqueda clientes instantánea
- Registro pagos múltiples métodos
- Impresión Bluetooth tickets
- Geolocalización
- Sincronización automática

✅ Productos (95%)

- CRUD productos
- Sistema múltiples precios (5 precios)
- Control inventario
- Categorías/marcas
- Código barras

✅ Comunicación (90%)

- WhatsApp individual
- SMS masivo
- Plantillas personalizables

✅ Dashboard (85%)

- Métricas tiempo real
- Gráficos Chart.js
- Resúmenes ejecutivos

✅ PWA (100%)

- Instalable móviles
- Funcionalidad offline
- Service Worker
- Manifest configurado

🎯 PRÓXIMOS MÓDULOS SUGERIDOS

1. Módulo de Ventas (Alta Prioridad)

```
// Archivos a crear:
/app/(dashboard)/ventas/
  ├── page.tsx              // Lista ventas
  ├── nueva/
  │   └── page.tsx          // Nueva venta
  ├── cotizaciones/
  │   └── page.tsx          // Cotizaciones
  └── [id]/
      └── page.tsx          // Detalle venta

/components/ventas/
  ├── venta-form.tsx        // Formulario venta
  ├── cotizacion-form.tsx   // Formulario cotización
  ├── venta-detail.tsx      // Detalle venta
  └── comision-calc.tsx     // Cálculo comisiones

/app/api/ventas/
  ├── route.ts              // CRUD ventas
  ├── cotizaciones/
  │   └── route.ts          // CRUD cotizaciones
  └── comisiones/
      └── route.ts          // Cálculo comisiones

// Modelo Prisma a agregar:
model Venta {
  id              String   @id @default(cuid())
  folio           String   @unique
  clienteId       String
  cliente         Cliente  @relation(fields: [clienteId], references: [id])
  vendedorId      String
  vendedor        User     @relation("VentaCreador", fields: [vendedorId],
references: [id])
  fecha           DateTime @default(now())
  subtotal        Float
  impuestos       Float
  total           Float
  estatus         String   @default("PENDIENTE") // PENDIENTE, PAGADA, CANCELADA
  detalles        DetalleVenta[]
}

model DetalleVenta {
  id          String   @id @default(cuid())
  ventaId     String
  venta       Venta    @relation(fields: [ventaId], references: [id])
  productoId  String
  producto    Producto @relation(fields: [productoId], references: [id])
  cantidad    Float
  precio      Float
  descuento   Float    @default(0)
  subtotal    Float
}
```

2. Cuentas por Cobrar (Media Prioridad)

```
// Funcionalidades:
- Estados de cuenta por cliente
- Aging de cartera (30, 60, 90+ días)
- Recordatorios automáticos
- Reportes de cobranza
- Proyecciones de flujo

// Archivos principales:
/app/(dashboard)/cuentas-por-cobrar/
/components/cuentas/
/app/api/cuentas/
```

3. Inventario Avanzado (Media Prioridad)

```
// Funcionalidades:
- Control de lotes
- Fechas de vencimiento
- Transferencias entre almacenes
- Toma física de inventario
- Reportes de rotación

// Archivos principales:
/app/(dashboard)/inventario/
/components/inventario/
/app/api/inventario/
```

💡 TIPS DE DESARROLLO PARA DEEPAGENT

Al Crear Nuevos Componentes:

```
// Siempre usar TypeScript
// Siempre incluir tipos
// Usar shadcn/ui components existentes

// Ejemplo estructura:
interface ComponentProps {
  // Props tipadas
}

export function ComponentName({ prop }: ComponentProps) {
  // Lógica del componente
  return (
    <div className="classes-tailwind">
      {/* JSX */}
    </div>
  )
}
```

Al Crear Nuevas APIs:

```
// Usar NextJS 14 App Router
// Siempre validar autenticación
// Incluir manejo de errores

import { NextRequest, NextResponse } from 'next/server'
import { getServerSession } from 'next-auth'
import { authOptions } from '@/lib/auth'
import { prisma } from '@/lib/prisma'

export async function GET(request: NextRequest) {
  try {
    const session = await getServerSession(authOptions)
    if (!session) {
      return NextResponse.json({ error: 'No autorizado' }, { status: 401 })
    }

    // Lógica de la API

    return NextResponse.json({ data })
  } catch (error) {
    return NextResponse.json({ error: 'Error interno' }, { status: 500 })
  }
}
```

Al Modificar Schema Prisma:

```
# Después de cambios al schema:
yarn prisma db push
yarn prisma generate
yarn prisma db seed  # Recargar datos

# Verificar cambios:
yarn prisma studio
```

🚀 CASOS DE USO ESPECÍFICOS

Usuario solicita: “Agregar campo X al cliente”

```
// 1. Modificar schema Prisma
model Cliente {
  // ... campos existentes ...
  campoNuevo    String?  // Agregar campo
}

// 2. Actualizar formulario cliente
// /components/clientes/cliente-form.tsx
// Agregar input para nuevo campo

// 3. Actualizar API
// /app/api/clientes/route.ts
// Incluir campo en validación

// 4. Ejecutar comandos
yarn prisma db push
yarn prisma generate
yarn prisma db seed
```

Usuario solicita: “Nuevo módulo de X”

```
// 1. Crear estructura de archivos
/app/(dashboard)/modulo-x/
  ├── page.tsx
  └── components específicos

/components/modulo-x/
  ├── component1.tsx
  └── component2.tsx

/app/api/modulo-x/
  └── route.ts

// 2. Agregar al sidebar
// /components/layout/sidebar.tsx
// Incluir enlace al nuevo módulo

// 3. Crear modelo Prisma si es necesario
model ModeloX {
  // Campos del modelo
}

// 4. Ejecutar workflow completo
```

Usuario solicita: “Corregir error en Y”

```
// 1. Identificar archivo con error
// 2. Leer archivo completo
// 3. Hacer corrección específica
// 4. Probar con test_nextjs_project
// 5. Guardar checkpoint
```

📋 CHECKLIST FINAL PARA DEEPAGENT

Antes de Finalizar Cualquier Tarea:

- [ ] ¿El proyecto sigue en `/home/ubuntu/sistema_erp_completo/app/`?
- [ ] ¿Se usó yarn (no npm)?
- [ ] ¿Se ejecutó `yarn prisma db seed` si fue necesario?
- [ ] ¿Se probó con `test_nextjs_project`?
- [ ] ¿Se guardó checkpoint con `build_and_save_nextjs_project_checkpoint`?
- [ ] ¿Las credenciales de prueba siguen funcionando?
- [ ] ¿La funcionalidad offline sigue operativa?

Credenciales de Prueba SIEMPRE Disponibles:

```
Email: admin@sistema.com
Password: 123456
Rol: SUPERADMIN

Otros:
gestor1@sistema.com / password123 (GESTOR)
vendedor1@sistema.com / password123 (VENTAS)
```

Instrucciones específicas para DeepAgent - Septiembre 19, 2025
Seguir estrictamente este workflow para evitar errores