



Guía Técnica - CitaPlanner MVP

Para Desarrolladores que Continúen el Proyecto



Índice

- [Arquitectura del Proyecto](#)
- [Patrones de Desarrollo](#)
- [Base de Datos](#)
- [Autenticación y Roles](#)
- [Componentes UI](#)
- [Estado y Datos](#)
- [API y Integraciones](#)
- [Despliegue](#)



Arquitectura del Proyecto

Estructura de Directorios

citaplanner_mvp/	
├── app/	# Aplicación Next.js 14 (App Router)
│ ├── app/	# Rutas de la aplicación
│ │ ├── admin/	# Panel administrativo
│ │ │ ├── inventory/	# COMPLETO - Gestión inventario
│ │ │ ├── clients/	# COMPLETO - Gestión clientes
│ │ │ ├── appointments/	# COMPLETO - Gestión citas
│ │ │ ├── reports/	# 75% - Reportes y analytics
│ │ │ ├── pos/	# 60% - Punto de venta
│ │ │ ├── branches/	# 50% - Gestión sucursales
│ │ │ ├── marketing/	# 40% - Marketing y campañas
│ │ │ └── settings/	# 30% - Configuraciones
│ │ ├── auth/	# COMPLETO - Autenticación
│ │ ├── dashboard/	# Dashboard por roles
│ │ ├── api/	# API Routes
│ │ └── book/	# Portal público reservas
│ ├── components/	# Componentes reutilizables
│ │ ├── ui/	# Shadcn/ui components
│ │ ├── modals/	# Modales CRUD
│ │ └── charts/	# Gráficos con Recharts
│ ├── lib/	# Utilidades y configuraciones
│ │ ├── integrations/	# Integraciones externas
│ │ └── utils.ts	# Utilidades generales
│ ├── prisma/	# Schema y configuración BD
│ └── public/	# Archivos estáticos
└── docs/	# Documentación del proyecto




Tecnologías Clave

```
{
  "framework": "Next.js 14 (App Router)",
  "language": "TypeScript (strict mode)",
  "database": "PostgreSQL + Prisma ORM",
  "auth": "NextAuth.js v4",
  "ui": "Tailwind CSS + Shadcn/ui",
  "forms": "React Hook Form + Zod",
  "state": "Zustand (preparado)",
  "charts": "Recharts",
  "notifications": "React Hot Toast"
}
```

Patrones de Desarrollo

1. Modales CRUD (Patrón Maestro)

Archivos de Referencia:

-  components/modals/inventory-modal.tsx (COMPLETO)
-  components/modals/client-modal.tsx (COMPLETO)
-  components/modals/appointment-modal.tsx (COMPLETO)

Estructura Estándar:

```

interface ModalProps {
  isOpen: boolean
  onClose: () => void
  mode: 'create' | 'edit' | 'custom' // Según necesidad
  item?: any
}

export function ExampleModal({ isOpen, onClose, mode, item }: ModalProps) {
  // 1. React Hook Form + Zod validation
  const { register, handleSubmit, formState: { errors } } = useForm({
    defaultValues: item || defaultValues,
    resolver: zodResolver(validationSchema)
  })

  // 2. Loading state
  const [isLoading, setIsLoading] = useState(false)

  // 3. Submit handler
  const onSubmit = async (data: any) => {
    setIsLoading(true)
    try {
      // API call or mock simulation
      await simulateApiCall(data)
      toast.success('Operación exitosa')
      onClose()
    } catch (error) {
      toast.error('Error en la operación')
    } finally {
      setIsLoading(false)
    }
  }

  return (
    <Dialog open={isOpen} onOpenChange={onClose}>
      <DialogContent className="max-w-2xl max-h-[90vh] overflow-y-auto">
        <DialogHeader>
          <DialogTitle>
            {mode === 'create' ? 'Crear' : 'Editar'} Elemento
          </DialogTitle>
        </DialogHeader>


        <form onSubmit={handleSubmit(onSubmit)}>
          {/* Campos del formulario */}

          <div className="flex gap-3 pt-4">
            <Button type="submit" disabled={isLoading} className="flex-1">
              {isLoading ? 'Guardando...' : 'Guardar'}
            </Button>
            <Button
              type="button"
              variant="outline"
              onClick={onClose}
              className="flex-1"
            >
              Cancelar
            </Button>
          </div>
        </form>
      </DialogContent>
    </Dialog>
  )
}

```

2. Páginas de Administración (Patrón Estándar)

Archivos de Referencia:

-  `app/admin/inventory/page.tsx` (EJEMPLO PERFECTO)
-  `app/admin/clients/page.tsx`
-  `app/admin/appointments/page.tsx`

Estructura Típica:

```

export default function AdminPage() {
  // 1. Estados locales
  const [searchTerm, setSearchTerm] = useState('')
  const [filter, setFilter] = useState('all')
  const [modal, setModal] = useState({
    isOpen: false,
    mode: 'create' as const,
    item: null
  })

  // 2. Datos (actualmente mock, fácil de reemplazar)
  const mockData = [...] // TODO: Reemplazar con API call

  // 3. Funciones de utilidad
  const filteredData = mockData.filter(item => {
    // Lógica de filtrado
  })

  const openModal = (mode: 'create' | 'edit', item?: any) => {
    setModal({ isOpen: true, mode, item })
  }

  const closeModal = () => {
    setModal({ isOpen: false, mode: 'create', item: null })
  }

  // 4. Render
  return (
    <div className="space-y-6">
      {/* Header con título y botón principal */}
      <div className="flex justify-between items-center">
        <div>
          <h1 className="text-2xl font-bold">Título de Página</h1>
          <p className="text-muted-foreground">Descripción</p>
        </div>
        <Button onClick={() => openModal('create')}>
          <Plus className="h-4 w-4 mr-2" />
          Nuevo Elemento
        </Button>
      </div>

      {/* Dashboard de estadísticas (4 cards) */}
      <div className="grid gap-4 md:grid-cols-4">
        {[1,2,3,4].map(i => (
          <Card key={i}>
            <CardContent className="p-6">
              <div className="text-2xl font-bold">{/* Métrica */</div>
              <p className="text-sm text-muted-foreground">
                {/* Descripción */}
              </p>
            </CardContent>
          </Card>
        ))}
      </div>

      {/* Filtros y búsqueda */}
      <div className="flex gap-4">
        <Input
          placeholder="Buscar..."
          value={searchTerm}
          onChange={(e) => setSearchTerm(e.target.value)}
          className="max-w-sm"
        />
      </div>
    </div>
  )
}

```

```

    />
    <Select value={filter} onValueChange={setFilter}>
      {/* Opciones de filtro */}
    </Select>
  </div>

  {/* Lista/Grid de elementos */}
  <div className="grid gap-4">
    {filteredData.map(item => (
      <Card key={item.id}>
        {/* Contenido del elemento */}
        <div className="flex gap-2">
          <Button
            variant="outline"
            onClick={() => openModal('edit', item)}
          >
            <Edit className="h-4 w-4" />
          </Button>
          {/* Más botones de acción */}
        </div>
      </Card>
    ))}
  </div>

  {/* Modal */}
  <ExampleModal
    isOpen={modal.isOpen}
    onClose={closeModal}
    mode={modal.mode}
    item={modal.item}
  />
</div>
)
}

```

3. API Routes (Patrón Básico)

Estructura Preparada:

```
// app/api/[resource]/route.ts
import { NextRequest, NextResponse } from 'next/server'

export async function GET(request: NextRequest) {
  try {
    // TODO: Implementar lógica de obtención
    // const data = await prisma.resource.findMany()

    return NextResponse.json({
      data: [],
      message: 'Success'
    })
  } catch (error) {
    return NextResponse.json(
      { error: 'Internal Server Error' },
      { status: 500 }
    )
  }
}

export async function POST(request: NextRequest) {
  try {
    const body = await request.json()

    // TODO: Validación con Zod
    // const validated = schema.parse(body)

    // TODO: Crear en BD
    // const result = await prisma.resource.create({ data: validated })

    return NextResponse.json({
      data: null,
      message: 'Created successfully'
    }, { status: 201 })
  } catch (error) {
    return NextResponse.json(
      { error: 'Bad Request' },
      { status: 400 }
    )
  }
}
```

Base de Datos

Schema Prisma (Completamente Definido)

Archivo: prisma/schema.prisma

Modelos Principales:

```
// MULTI-TENANT CORE
model Tenant {
  id          String    @id @default(cuid())
  name        String
  subdomain   String    @unique
  isActive    Boolean   @default(true)
  createdAt   DateTime  @default(now())

  // Relaciones
  users       User[]
  branches    Branch[]
  clients     Client[]
  services    Service[]
  products    Product[]
  appointments Appointment[]
}

// USUARIOS Y ROLES
model User {
  id          String    @id @default(cuid())
  email       String    @unique
  firstName   String
  lastName    String
  role        UserRole
  tenantId    String
  branchId    String?

  // Relaciones
  tenant      Tenant    @relation(fields: [tenantId], references: [id])
  branch      Branch?   @relation(fields: [branchId], references: [id])
}

enum UserRole {
  SUPERADMIN    // Acceso global
  ADMIN         // Administrador de tenant
  MANAGER       // Gerente de sucursal
  PROFESSIONAL  // Profesional/Empleado
  RECEPTIONIST  // Recepcionista
  CLIENT        // Cliente
}

// GESTIÓN DE INVENTARIO (IMPLEMENTADO)
model Product {
  id          String    @id @default(cuid())
  name        String
  sku         String    @unique
  description  String?
  category    String
  supplier    String?
  price       Decimal
  cost        Decimal
  stock       Int
  minStock    Int       @default(5)
  isActive    Boolean   @default(true)
  tenantId    String

  tenant      Tenant    @relation(fields: [tenantId], references: [id])
}

// GESTIÓN DE CLIENTES (IMPLEMENTADO)
model Client {
  id          String    @id @default(cuid())
```



```

    firstName String
    lastName  String
    email     String?
    phone     String
    address   String?
    birthDate DateTime?
    notes     String?
    tenantId  String

    tenant      Tenant      @relation(fields: [tenantId], references: [id])
    appointments Appointment[]

}

// GESTIÓN DE CITAS (IMPLEMENTADO)
model Appointment {
  id          String          @id @default(cuid())
  date        DateTime
  duration     Int             // En minutos
  status       AppointmentStatus @default(PENDING)
  notes        String?
  totalAmount  Decimal?

  // Relaciones
  clientId      String
  serviceId     String
  professionalId String
  tenantId      String
  branchId      String

  client      Client      @relation(fields: [clientId], references: [id])
  service     Service     @relation(fields: [serviceId], references: [id])
  professional User       @relation(fields: [professionalId], references: [id])
  tenant      Tenant      @relation(fields: [tenantId], references: [id])
  branch      Branch      @relation(fields: [branchId], references: [id])
}

enum AppointmentStatus {
  PENDING    // Pendiente
  CONFIRMED  // Confirmada
  COMPLETED  // Completada
  CANCELLED  // Cancelada
  NO_SHOW    // No se presentó
}

// SERVICIOS (PREPARADO)
model Service {
  id          String          @id @default(cuid())
  name        String
  description  String?
  duration     Int             // En minutos
  price        Decimal
  category     String
  isActive     Boolean @default(true)
  tenantId     String

  tenant      Tenant      @relation(fields: [tenantId], references: [id])
  appointments Appointment[]
}

// SUCURSALES (PREPARADO)
model Branch {
  id          String          @id @default(cuid())
  name        String

```

```

address      String
phone        String?
email        String?
isActive     Boolean @default(true)
tenantId     String

tenant       Tenant      @relation(fields: [tenantId], references: [id])
users        User[]
appointments Appointment[]

```

Comandos de BD Importantes

```

# Ver BD en interfaz visual
yarn prisma studio

# Aplicar cambios del schema
yarn prisma db push

# Regenerar cliente Prisma
yarn prisma generate

# Reset completo de BD (CUIDADO)
yarn prisma db reset

```

Autenticación y Roles

NextAuth.js Configuración

Archivo: app/api/auth/[...nextauth]/route.ts

Roles Implementados:

```

enum UserRole {
  SUPERADMIN = 'SUPERADMIN', // Acceso total sistema
  ADMIN = 'ADMIN',           // Admin de empresa
  MANAGER = 'MANAGER',       // Gerente sucursal
  PROFESSIONAL = 'PROFESSIONAL', // Empleado/Profesional
  RECEPTIONIST = 'RECEPTIONIST', // Recepcionista
  CLIENT = 'CLIENT'          // Cliente final
}

```

Protección de Rutas:

```

// middleware.ts (CONFIGURADO)
export { default } from "next-auth/middleware"

export const config = {
  matcher: [
    '/admin/:path*', // Solo roles administrativos
    '/dashboard/:path*', // Usuarios autenticados
    '/api/admin/:path*' // APIs protegidas
  ]
}

```

Uso en Componentes:

```

'use client'
import { useSession } from 'next-auth/react'

export function ProtectedComponent() {
  const { data: session, status } = useSession()

  if (status === 'loading') return <Loading />
  if (status === 'unauthenticated') return <Redirect />

  // Verificar roles
  const isAdmin = session?.user?.role === 'ADMIN'
  const canManageInventory = ['ADMIN', 'MANAGER'].includes(session?.user?.role)

  return (
    <div>
      {/* Contenido protegido */}
      {canManageInventory && <InventorySection />}
    </div>
  )
}

```

**Componentes UI****Biblioteca Shadcn/ui (Completa)****Componentes Disponibles:**

```

// Formularios
import { Button } from '@components/ui/button'
import { Input } from '@components/ui/input'
import { Label } from '@components/ui/label'
import { Textarea } from '@components/ui/textarea'
import { Select, SelectContent, SelectItem } from '@components/ui/select'
import { Switch } from '@components/ui/switch'
import { Checkbox } from '@components/ui/checkbox'

// Layout
import { Card, CardContent, CardHeader } from '@components/ui/card'
import { Dialog, DialogContent, DialogHeader } from '@components/ui/dialog'
import { Sheet, SheetContent, SheetTrigger } from '@components/ui/sheet'
import { Tabs, TabsContent, TabsList, TabsTrigger } from '@components/ui/tabs'

// Navegación
import { DropdownMenu, DropdownMenuItem } from '@components/ui/dropdown-menu'
import { Badge } from '@components/ui/badge'
import { Avatar, AvatarImage } from '@components/ui/avatar'

// Feedback
import { Alert, AlertDescription } from '@components/ui/alert'
import { toast } from 'react-hot-toast' // Para notificaciones

```

Convenciones de Estilo**Colores del Tema:**

```
/* Paleta principal */
primary: blue-600 /* Botones principales */
secondary: gray-100 /* Botones secundarios */
muted: gray-500 /* Texto secundario */
destructive: red-600 /* Acciones de eliminación */
success: green-600 /* Estados exitosos */
warning: orange-500 /* Alertas y avisos */
```

Espaciado Consistente:

```
className="space-y-6" // Espacio vertical entre secciones
className="space-y-4" // Espacio entre elementos
className="gap-4" // Grid gaps
className="p-6" // Padding de cards
className="px-4 py-2" // Padding de botones
```



Estado y Datos

Datos Mock (Actual)

Estructura Actual:

```
// Patrón usado en todas las páginas
const mockProducts = [
  {
    id: 1,
    name: 'Shampoo Premium',
    sku: 'SH001',
    price: 299.99,
    stock: 25,
    status: 'in_stock', // 'low_stock', 'out_of_stock'
    category: 'Cuidado Capilar'
  },
  // ... más elementos
]

// TODO: Reemplazar con:
// const { data: products } = useQuery('products', fetchProducts)
```

Zustand (Preparado para Expansión)

Configuración Básica:

```
// lib/store.ts (PREPARADO)
import { create } from 'zustand'

interface AppState {
  // Estados globales cuando se necesiten
  currentTenant: string | null
  currentBranch: string | null

  // Acciones
  setTenant: (tenantId: string) => void
  setBranch: (branchId: string) => void
}

export const useAppStore = create<AppState>((set) => ({
  currentTenant: null,
  currentBranch: null,

  setTenant: (tenantId) => set({ currentTenant: tenantId }),
  setBranch: (branchId) => set({ currentBranch: branchId })
}))
```

React Query (Recomendado para APIs)

Configuración Sugerida:

```
// lib/queries.ts (PARA IMPLEMENTAR)
import { useQuery, useMutation } from '@tanstack/react-query'

// Ejemplo para productos
export const useProducts = () => {
  return useQuery({
    queryKey: ['products'],
    queryFn: () => fetch('/api/products').then(res => res.json()),
    staleTime: 5 * 60 * 1000 // 5 minutos
  })
}

export const useCreateProduct = () => {
  return useMutation({
    mutationFn: (data: any) =>
      fetch('/api/products', {
        method: 'POST',
        body: JSON.stringify(data)
      }),
    onSuccess: () => {
      // Invalidar cache
      queryClient.invalidateQueries(['products'])
    }
  })
}
```

API y Integraciones

API Routes Implementadas

Estructura Actual:

app/api/	
auth/[...nextauth]/route.ts	✓ NextAuth configuration
signup/route.ts	✓ User registration
appointments/	
recent/route.ts	⚠ Mock data
dashboard/	
metrics/route.ts	⚠ Mock data
payments/	
create/route.ts	⚠ OpenPay structure
notifications/	
sms/route.ts	⚠ SMS structure
whatsapp/route.ts	⚠ WhatsApp structure

Integraciones Externas

1. OpenPay (Pagos) - Preparado

```
// lib/integrations/openpay.ts
export class OpenPayService {
  private client: any

  constructor() {
    // TODO: Configurar cliente OpenPay
    // this.client = new OpenPay(...)
  }

  async createCharge(data: ChargeData) {
    // TODO: Implementar cargo
    console.log('Creating charge:', data)
    return { id: 'mock-charge-id' }
  }
}
```

2. SMS/WhatsApp - Estructura Preparada

```
// lib/integrations/sms.ts
export class SMSService {
  async sendReminder(phone: string, message: string) {
    // TODO: Integrar con Twilio o similar
    console.log(`SMS to ${phone}: ${message}`)
  }
}

// lib/integrations/whatsapp.ts
export class WhatsAppService {
  async sendMessage(phone: string, message: string) {
    // TODO: Integrar con WhatsApp Business API
    console.log(`WhatsApp to ${phone}: ${message}`)
  }
}
```

Despliegue

Variables de Entorno Necesarias

Archivo: `.env` (usar `.env.example` como base)

```
# Base de datos
DATABASE_URL="postgresql://user:pass@host:port/db"

# NextAuth
NEXTAUTH_URL="https://tu-dominio.com"
NEXTAUTH_SECRET="clave-super-segura-32-chars"

# OpenPay (Opcional)
OPENPAY_ID="merchant-id"
OPENPAY_PRIVATE_KEY="private-key"
OPENPAY_PUBLIC_KEY="public-key"
OPENPAY_MODE="sandbox" # o "production"

# SMS/WhatsApp (Opcional)
TWILIO_ACCOUNT_SID="twilio-sid"
TWILIO_AUTH_TOKEN="twilio-token"
```

Comandos de Despliegue

Build de Producción:

```
cd app/
yarn install
yarn prisma generate
yarn build
```

Verificación Pre-despliegue:

```
# Verificar TypeScript
yarn tsc --noEmit

# Verificar build
yarn build

# Verificar base de datos
yarn prisma db push
```

Plataformas Recomendadas

1. Vercel (Más Simple):

- Conectar repositorio GitHub
- Configurar variables de entorno
- Deploy automático

2. Railway:

- Base de datos PostgreSQL incluida
- Deploy con Docker
- Variables de entorno en dashboard

3. Docker (Personalizado):

```
# Dockerfile sugerido
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN yarn install
COPY . .
RUN yarn build
EXPOSE 3000
CMD ["yarn", "start"]
```

Tips de Desarrollo

Comandos Útiles

```
# Desarrollo completo
yarn dev                # Servidor + BD en modo watch

# Base de datos
yarn prisma studio      # GUI para ver datos
yarn prisma db seed     # Poblar con datos de prueba

# Calidad de código
yarn lint               # ESLint
yarn type-check         # TypeScript

# Build
yarn build              # Producción
yarn start              # Servidor de producción
```

Debugging

```
// Logs estructurados
console.log('🔍 Debug:', { data, user, timestamp: new Date() })

// Error boundaries
try {
  await riskyOperation()
} catch (error) {
  console.error('❌ Error in operation:', error)
  toast.error('Error en la operación')
}
```

Performance

```
// Lazy loading de componentes pesados
const HeavyModal = dynamic(() => import('./heavy-modal'), {
  loading: () => <Loading />
})

// Memoización de cálculos costosos
const expensiveCalculation = useMemo(() => {
  return complexCalculation(data)
}, [data])
```

Esta guía técnica debe consultarse antes de continuar el desarrollo. Mantén la consistencia con los patrones establecidos. 🚀