

# Documentación Técnica - DHérmica Estética

## Resumen Ejecutivo

**DHérmica Estética** es una aplicación web progresiva (PWA) diseñada para la gestión de citas y tratamientos estéticos. La aplicación permite a los clientes registrarse, agendar citas y ver su historial, mientras que los profesionales pueden gestionar turnos de manera simple e intuitiva.

## Objetivos Principales

- Digitalizar el proceso de agendamiento de citas
  - Simplificar la gestión para profesionales no técnicos
  - Brindar transparencia a los clientes sobre sus tratamientos
  - Validar restricciones médicas automáticamente
- 

## Arquitectura del Sistema

### Stack Tecnológico

- **Frontend:** Next.js 14 (App Router) - **JavaScript (NO TypeScript)**
- **Backend:** Firebase Functions (Node.js)
- **Base de Datos:** Firestore (NoSQL)
- **Autenticación:** Firebase Auth
- **Almacenamiento:** Cloudinary (imágenes)
- **UI Components:** shadcn/ui + Tailwind CSS
- **Estado Global:** Zustand
- **Validación:** Zod (validación en runtime)
- **PWA:** next-pwa
- **Deploy:** Vercel

### Arquitectura de Componentes

└─ app/	# App Router (Next.js 14)
└─ (auth)/	# Rutas protegidas
└─ (client)/	# Panel del cliente
└─ (admin)/	# Panel administrativo
└─ (public)/	# Páginas públicas
└─ components/	# Componentes reutilizables
└─ lib/	# Utilidades y configuraciones
└─ hooks/	# Custom hooks
└─ store/	# Estado global (Zustand)

## Sistema de Diseño

### Paleta de Colores

CSS

```
:root {
  --background: #f8f9fa;
  --foreground: #1a1a1a;
  --primary: #484450;
  --primary-foreground: #ffffff;
  --secondary: #466067;
  --secondary-foreground: #ffffff;
  --accent: #34baab;
  --accent-foreground: #ffffff;
  --muted: #c4c8c5;
  --muted-foreground: #6b7280;
  --success: #34baab;
  --warning: #f59e0b;
  --error: #ef4444;
  --border: #e5e7eb;
}
```

### Principios de UX

- **Mobile First:** Diseño prioritario para dispositivos móviles
- **Simplicidad:** Interfaz intuitiva para usuarios no técnicos
- **Accesibilidad:** Cumple con estándares WCAG 2.1
- **Feedback Visual:** Respuesta inmediata a todas las acciones

## Estructura de Base de Datos

### Colecciones Firestore

## 1. `/clients/{clientId}`

typescript

```
interface Client {  
  id: string  
  name: string  
  email: string  
  phone: string  
  dateOfBirth: Date  
  medicalInfo: {  
    diabetes: boolean  
    cancer: boolean  
    tattoos: boolean  
    allergies: string  
    medications: string  
    other: string  
  }  
  createdAt: Timestamp  
  updatedAt: Timestamp  
}
```

## 2. `/appointments/{appointmentId}`

typescript

```
interface Appointment {  
  id: string  
  clientId: string    // Referencia al cliente  
  clientName: string  // Para mostrar rápido en admin  
  treatmentId: string // Referencia al tratamiento  
  professionalId: string // Referencia al profesional  
  date: Timestamp    // Solo fecha (2025-08-15)  
  startTime: string   // "14:00"  
  endTime: string     // "15:30" - Calculado automáticamente  
  duration: number    // 90 minutos  
  price?: number      // Opcional, se agrega después  
  createdAt: Timestamp  
  updatedAt: Timestamp  
}
```

## 3. `/treatments/{treatmentId}`

typescript

```
interface Treatment {
  id: string
  name: string
  description: string
  duration: number           // En minutos
  basePrice: number
  category: string           // "Facial", "Corporal", etc.
  imageUrl: string
  medicalRestrictions: string[] // ['diabetes', 'cancer', 'tattoos']
  active: boolean
  createdAt: Timestamp
  updatedAt: Timestamp
}
```

#### 4. `/professionals/{professionalId}`

typescript

```
interface Professional {
  id: string
  name: string
  specialties: string[]
  workingHours: {
    monday: { start: string, end: string, active: boolean }
    tuesday: { start: string, end: string, active: boolean }
    wednesday: { start: string, end: string, active: boolean }
    thursday: { start: string, end: string, active: boolean }
    friday: { start: string, end: string, active: boolean }
    saturday: { start: string, end: string, active: boolean }
    sunday: { start: string, end: string, active: boolean }
  }
  available: boolean
  createdAt: Timestamp
  updatedAt: Timestamp
}
```

#### 5. `/users/{userId}` (Admin)

typescript

```
interface User {  
  id: string  
  email: string  
  role: 'admin' | 'professional'  
  professionalId?: string // Si es profesional  
  createdAt: Timestamp  
  lastLogin: Timestamp  
}
```

## Índices Compuestos Requeridos

```
javascript  
  
// Firestore Indexes  
appointments: [  
  ['professionalId', 'date'], // Para validar conflictos  
  ['clientId', 'createdAt'], // Para historial del cliente  
  ['date', 'startTime'], // Para calendario  
]
```

## Sistema de Autenticación

### Roles de Usuario

1. **Cliente (client)**: Puede ver sus citas y actualizar su perfil
2. **Administrador (admin)**: Acceso completo al sistema
3. **Profesional (professional)**: Gestión de turnos y clientes

### Flujo de Autenticación

```
typescript
```

```
// Registro de Cliente
```

```
const registerClient = async (email: string, password: string, userData: ClientData) => {  
  const userCredential = await createUserWithEmailAndPassword(auth, email, password)  
  await setDoc(doc(db, 'clients', userCredential.user.uid), userData)  
  await setDoc(doc(db, 'users', userCredential.user.uid), {  
    email,  
    role: 'client',  
    createdAt: serverTimestamp()  
  })  
}
```

```
// Verificación de Permisos
```

```
const checkUserRole = async (uid: string): Promise<string> => {  
  const userDoc = await getDoc(doc(db, 'users', uid))  
  return userDoc.data()?.role || 'client'  
}
```

## ⚡ Funcionalidades Core

### 1. Gestión de Turnos (Admin)

#### Proceso de Creación de Turno

```
typescript
```

```
interface CreateAppointmentFlow {  
  step1: 'select_professional' // Seleccionar profesional  
  step2: 'select_date' // Seleccionar fecha  
  step3: 'search_client' // Buscar cliente en agenda  
  step4: 'select_treatment' // Seleccionar tratamiento  
  step5: 'select_time' // Horarios disponibles  
  step6: 'confirm' // Confirmar y guardar  
}
```

#### Validación de Conflictos de Horarios

```
typescript
```

```

const validateAppointmentTime = async (
  professionalId: string,
  date: string,
  startTime: string,
  duration: number,
  excludeId?: string
): Promise<{ isValid: boolean; conflicts: Appointment[] }> => {

  const appointments = await getDocs(
    query(
      collection(db, 'appointments'),
      where('professionalId', '=', professionalId),
      where('date', '=', date)
    )
  )

  const newStartTime = timeToMinutes(startTime)
  const newEndTime = newStartTime + duration

  const conflicts = appointments.docs
    .filter(doc => doc.id !== excludeId)
    .filter(doc => {
      const existingStart = timeToMinutes(doc.data().startTime)
      const existingEnd = existingStart + doc.data().duration
      return !(newEndTime <= existingStart || newStartTime >= existingEnd)
    })

  return {
    isValid: conflicts.length === 0,
    conflicts: conflicts.map(doc => ({ id: doc.id, ...doc.data() }))
  }
}

```

## 2. Sistema de Clientes

### Búsqueda Inteligente de Clientes

typescript

```

const searchClients = async (searchTerm: string): Promise<Client[]> => {
  const clients = await getDocs(
    query(
      collection(db, 'clients'),
      orderBy('name'),
      limit(10)
    )
  )

  return clients.docs
    .map(doc => ({ id: doc.id, ...doc.data() }))
    .filter(client =>
      client.name.toLowerCase().includes(searchTerm.toLowerCase()) ||
      client.phone.includes(searchTerm) ||
      client.email.toLowerCase().includes(searchTerm.toLowerCase())
    )
  }

```

## Validación de Restricciones Médicas

```

typescript

const validateMedicalRestrictions = (
  clientMedicalInfo: MedicalInfo,
  treatmentRestrictions: string[]
): { isValid: boolean; warnings: string[] } => {

  const warnings: string[] = []

  treatmentRestrictions.forEach(restriction => {
    if (clientMedicalInfo[restriction]) {
      warnings.push(`Atención: El cliente tiene ${restriction}`)
    }
  })

  return {
    isValid: warnings.length === 0,
    warnings
  }
}

```

## 3. Panel del Cliente

### Vista de Citas del Cliente

```

typescript

```



```
const getClientAppointments = async (clientId: string) => {
  const appointments = await getDocs(
    query(
      collection(db, 'appointments'),
      where('clientId', '==', clientId),
      orderBy('date', 'desc')
    )
  )

  return Promise.all(
    appointments.docs.map(async (doc) => {
      const appointmentData = doc.data()
      const treatment = await getDoc(doc(db, 'treatments', appointmentData.treatmentId))
      const professional = await getDoc(doc(db, 'professionals', appointmentData.professionalId))

      return {
        id: doc.id,
        ...appointmentData,
        treatment: treatment.data(),
        professional: professional.data()
      }
    })
  )
}
```

## Estructura de Componentes

### Layout Principal

typescript

```
// app/layout.tsx
export default function RootLayout({ children }: { children: ReactNode }) {
  return (
    <html lang="es">
      <body className="min-h-screen bg-background font-sans antialiased">
        <AuthProvider>
          <ThemeProvider>
            <Toaster />
            {children}
          </ThemeProvider>
        </AuthProvider>
      </body>
    </html>
  )
}
```

## Componentes Core

### 1. AppointmentForm (Admin)

typescript

```
interface AppointmentFormProps {
  onSuccess: () => void
  editingAppointment?: Appointment
}

const AppointmentForm = ({ onSuccess, editingAppointment }: AppointmentFormProps) => {
  const [step, setStep] = useState<CreateAppointmentStep>('select_professional')
  const [selectedProfessional, setSelectedProfessional] = useState<Professional>()
  const [selectedClient, setSelectedClient] = useState<Client>()
  const [selectedTreatment, setSelectedTreatment] = useState<Treatment>()
  const [selectedDate, setSelectedDate] = useState<Date>()
  const [selectedTime, setSelectedTime] = useState<string>()
  const [availableSlots, setAvailableSlots] = useState<TimeSlot[]>([])

  // Lógica del formulario...
}
```

### 2. ClientSearch

typescript

```

interface ClientSearchProps {
  onSelectClient: (client: Client) => void
  allowNew?: boolean
}

const ClientSearch = ({ onSelectClient, allowNew = false }: ClientSearchProps) => {
  const [searchTerm, setSearchTerm] = useState("")
  const [clients, setClients] = useState<Client[]>([])
  const [isLoading, setIsLoading] = useState(false)

  // Búsqueda con debounce...
}

```

### 3. TimeSlotPicker

```

typescript

interface TimeSlotPickerProps {
  professionalId: string
  date: Date
  duration: number
  onSelectTime: (time: string) => void
  excludeAppointmentId?: string
}

const TimeSlotPicker = ({
  professionalId,
  date,
  duration,
  onSelectTime
}: TimeSlotPickerProps) => {
  // Generación de slots disponibles...
}

```

### 4. AppointmentCalendar

```

typescript

```

```

interface AppointmentCalendarProps {
  professionalId?: string
  date: Date
  appointments: Appointment[]
  onChange: (date: Date) => void
  onAppointmentClick: (appointment: Appointment) => void
}

const AppointmentCalendar = ({
  professionalId,
  date,
  appointments,
  onChange,
  onAppointmentClick
}: AppointmentCalendarProps) => {
  // Vista de calendario...
}

```

## Sistema de Componentes UI

### Componentes shadcn/ui Utilizados

```

typescript

// Componentes principales
import { Button } from "@/components/ui/button"
import { Calendar } from "@/components/ui/calendar"
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card"
import { Dialog, DialogContent, DialogHeader, DialogTitle } from "@/components/ui/dialog"
import { Form, FormControl, FormField, FormItem, FormLabel } from "@/components/ui/form"
import { Input } from "@/components/ui/input"
import { Select, SelectContent, SelectItem, SelectTrigger } from "@/components/ui/select"
import { Textarea } from "@/components/ui/textarea"
import { Badge } from "@/components/ui/badge"
import { Avatar, AvatarFallback, AvatarImage } from "@/components/ui/avatar"
import { Separator } from "@/components/ui/separator"
import { ScrollArea } from "@/components/ui/scroll-area"
import { Skeleton } from "@/components/ui/skeleton"
import { Toast } from "@/components/ui/toast"

```

### Componentes Personalizados

```

typescript

```

// components/custom/

- |— AppointmentCard.tsx # Tarjeta de cita
- |— TreatmentCard.tsx # Tarjeta de tratamiento
- |— ClientCard.tsx # Tarjeta de cliente
- |— ProfessionalCard.tsx # Tarjeta de profesional
- |— TimeSlot.tsx # Slot de tiempo
- |— MedicalWarning.tsx # Alerta médica
- |— LoadingSpinner.tsx # Spinner personalizado
- |— EmptyState.tsx # Estado vacío
- |— ErrorBoundary.tsx # Manejo de errores
- |— PWAInstallPrompt.tsx # Prompt de instalación PWA

## Rutas y Navegación

### Estructura de Rutas

```
app/
├── page.tsx          # Landing page
├── login/page.tsx    # Página de login
├── register/page.tsx # Registro de cliente
├── (client)/
│   ├── dashboard/page.tsx # Dashboard del cliente
│   ├── appointments/page.tsx # Mis citas
│   ├── profile/page.tsx    # Mi perfil
│   └── history/page.tsx    # Historial de tratamientos
├── (admin)/
│   ├── dashboard/page.tsx # Dashboard admin
│   ├── appointments/
│   │   ├── page.tsx      # Lista de citas
│   │   ├── new/page.tsx  # Nueva cita
│   │   └── [id]/page.tsx # Editar cita
│   ├── clients/
│   │   ├── page.tsx      # Lista de clientes
│   │   ├── new/page.tsx  # Nuevo cliente
│   │   └── [id]/page.tsx # Perfil del cliente
│   ├── treatments/
│   │   ├── page.tsx      # Lista de tratamientos
│   │   ├── new/page.tsx  # Nuevo tratamiento
│   │   └── [id]/page.tsx # Editar tratamiento
│   └── professionals/
│       ├── page.tsx      # Lista de profesionales
│       ├── new/page.tsx  # Nuevo profesional
│       └── [id]/page.tsx # Editar profesional
└── treatments/
    ├── page.tsx          # Catálogo público
    └── [slug]/page.tsx    # Detalle del tratamiento
```

## Protección de Rutas

typescript

```
// middleware.ts
```

```
import { NextRequest, NextResponse } from 'next/server'
```

```
import { verifyToken } from '@lib/firebase-admin'
```

```
export async function middleware(request: NextRequest) {
```

```
  const token = request.cookies.get('auth-token')?.value
```

```
  if (request.nextUrl.pathname.startsWith('/admin')) {
```

```
    if (!token) {
```

```
      return NextResponse.redirect(new URL('/login', request.url))
```

```
    }
```

```
    const user = await verifyToken(token)
```

```
    if (user.role !== 'admin' && user.role !== 'professional') {
```

```
      return NextResponse.redirect(new URL('/dashboard', request.url))
```

```
    }
```

```
  }
```

```
  if (request.nextUrl.pathname.startsWith('/client')) {
```

```
    if (!token) {
```

```
      return NextResponse.redirect(new URL('/login', request.url))
```

```
    }
```

```
  }
```

```
  return NextResponse.next()
```

```
}
```

## Configuraciones

### Firebase Configuration

```
typescript
```

```
// lib/firebase.ts
```

```
import { initializeApp } from 'firebase/app'
```

```
import { getAuth } from 'firebase/auth'
```

```
import { getFirestore } from 'firebase/firestore'
```

```
import { getStorage } from 'firebase/storage'
```

```
const firebaseConfig = {
```

```
  apiKey: process.env.NEXT_PUBLIC_FIREBASE_API_KEY,
```

```
  authDomain: process.env.NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN,
```

```
  projectId: process.env.NEXT_PUBLIC_FIREBASE_PROJECT_ID,
```

```
  storageBucket: process.env.NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET,
```

```
  messagingSenderId: process.env.NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID,
```

```
  appId: process.env.NEXT_PUBLIC_FIREBASE_APP_ID
```

```
}
```

```
const app = initializeApp(firebaseConfig)
```

```
export const auth = getAuth(app)
```

```
export const db = getFirestore(app)
```

```
export const storage = getStorage(app)
```

## Clouinary Configuration

```
typescript
```



```
// lib/cloudinary.ts
```

```
import { v2 as cloudinary } from 'cloudinary'
```

```
cloudinary.config({  
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,  
  api_key: process.env.CLOUDINARY_API_KEY,  
  api_secret: process.env.CLOUDINARY_API_SECRET  
})
```

```
export const uploadImage = async (file: File): Promise<string> => {  
  const formData = new FormData()  
  formData.append('file', file)  
  formData.append('upload_preset', 'dhermica_preset')
```

```
  
  const response = await fetch(  
    `https://api.cloudinary.com/v1_1/${process.env.NEXT_PUBLIC_CLOUDINARY_CLOUD_NAME}/image/upload`,  
    {  
      method: 'POST',  
      body: formData  
    }  
  )
```

```
  
  const data = await response.json()  
  return data.secure_url  
}
```

## Tailwind Configuration

```
javascript
```

```
// tailwind.config.js
```

```
module.exports = {  
  content: [  
    './pages/**/*.ts,tsx',  
    './components/**/*.ts,tsx',  
    './app/**/*.ts,tsx',  
    './src/**/*.ts,tsx',  
  ],  
  theme: {  
    extend: {  
      colors: {  
        background: 'hsl(var(--background))',  
        foreground: 'hsl(var(--foreground))',  
        primary: {  
          DEFAULT: 'hsl(var(--primary))',  
          foreground: 'hsl(var(--primary-foreground))',  
        },  
        secondary: {  
          DEFAULT: 'hsl(var(--secondary))',  
          foreground: 'hsl(var(--secondary-foreground))',  
        },  
        accent: {  
          DEFAULT: 'hsl(var(--accent))',  
          foreground: 'hsl(var(--accent-foreground))',  
        },  
        muted: {  
          DEFAULT: 'hsl(var(--muted))',  
          foreground: 'hsl(var(--muted-foreground))',  
        },  
      },  
    },  
    animation: {  
      'fade-in': 'fadeIn 0.5s ease-in-out',  
      'slide-up': 'slideUp 0.3s ease-out',  
      'pulse-slow': 'pulse 3s cubic-bezier(0.4, 0, 0.6, 1) infinite',  
    },  
    keyframes: {  
      fadeIn: {  
        '0%': { opacity: '0' },  
        '100%': { opacity: '1' },  
      },  
      slideUp: {  
        '0%': { transform: 'translateY(10px)', opacity: '0' },  
        '100%': { transform: 'translateY(0)', opacity: '1' },  
      },  
    },  
  },  
}
```

```
},  
plugins: [require('tailwindcss-animate')],  
}
```

## Next.js Configuration

```
javascript  
  
// next.config.js  
const withPWA = require('next-pwa')({  
  dest: 'public',  
  register: true,  
  skipWaiting: true,  
  disable: process.env.NODE_ENV === 'development'  
})  
  
module.exports = withPWA({  
  experimental: {  
    appDir: true,  
  },  
  images: {  
    domains: ['res.cloudinary.com'],  
  },  
  env: {  
    NEXT_PUBLIC_APP_URL: process.env.NEXT_PUBLIC_APP_URL,  
  },  
})
```



## Plan de Desarrollo

### Fase 1: Setup y Autenticación (Semana 1)

- ☒ Configuración inicial del proyecto Next.js 14
- ☒ Setup de Firebase y Firestore
- ☒ Implementación de autenticación
- ☒ Configuración de shadcn/ui y Tailwind
- ☐ Sistema de roles y protección de rutas
- ☐ Layout base y navegación

### Fase 2: Gestión de Datos Base (Semana 2)

- ☐ CRUD de tratamientos
- ☐ CRUD de profesionales
- ☐ CRUD de clientes
- ☐ Validaciones de formularios

- ☐ Subida de imágenes con Cloudinary

### Fase 3: Sistema de Citas Core (Semana 3)

- ☐ Componente de búsqueda de clientes
- ☐ Selector de horarios disponibles
- ☐ Validación de conflictos de horarios
- ☐ Formulario de creación de citas
- ☐ Vista de calendario para admin

### Fase 4: Panel del Cliente (Semana 4)

- ☐ Registro de clientes
- ☐ Formulario de datos médicos
- ☐ Dashboard del cliente
- ☐ Vista de citas del cliente
- ☐ Historial de tratamientos

### Fase 5: Optimizaciones y PWA (Semana 5)

- ☐ Optimización de performance
- ☐ Implementación PWA
- ☐ Validaciones médicas automáticas
- ☐ Sistema de notificaciones
- ☐ Testing y debugging

### Fase 6: Despliegue y Documentación (Semana 6)

- ☐ Deploy en Vercel
- ☐ Configuración de dominio
- ☐ Documentación de usuario
- ☐ Training para profesionales
- ☐ Monitoreo y analytics



## Métricas y Monitoreo

### KPIs Importantes

- **Tiempo de carga de página** < 3 segundos
- **Tasa de conversión** de registro de clientes
- **Tiempo promedio** para crear una cita (objetivo: < 2 minutos)
- **Errores de validación** de horarios
- **Adopción PWA** (instalaciones móviles)

## Herramientas de Monitoreo

- **Vercel Analytics** para performance
  - **Firebase Analytics** para eventos de usuario
  - **Sentry** para manejo de errores
  - **Google Search Console** para SEO
- 



## Consideraciones de Seguridad

### Reglas de Firestore Security

javascript

```

// firestore.rules
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Clients can only read/write their own data
    match /clients/{clientId} {
      allow read, write: if request.auth != null && request.auth.uid == clientId;
    }

    // Only admin/professionals can manage appointments
    match /appointments/{appointmentId} {
      allow read, write: if request.auth != null &&
        exists(/databases/{database}/documents/users/{request.auth.uid}) &&
        get(/databases/{database}/documents/users/{request.auth.uid}).data.role in ['admin', 'professional'];
      allow read: if request.auth != null &&
        resource.data.clientId == request.auth.uid;
    }

    // Treatments are read-only for clients, admin can modify
    match /treatments/{treatmentId} {
      allow read: if true; // Public reading
      allow write: if request.auth != null &&
        get(/databases/{database}/documents/users/{request.auth.uid}).data.role == 'admin';
    }

    // Professionals - admin only
    match /professionals/{professionalId} {
      allow read, write: if request.auth != null &&
        get(/databases/{database}/documents/users/{request.auth.uid}).data.role == 'admin';
    }

    // Users - only admin can modify
    match /users/{userId} {
      allow read: if request.auth != null && request.auth.uid == userId;
      allow write: if request.auth != null &&
        get(/databases/{database}/documents/users/{request.auth.uid}).data.role == 'admin';
    }
  }
}

```

## Validaciones de Seguridad

- **Input Sanitization** en todos los formularios
- **Rate Limiting** para prevenir spam
- **HTTPS Only** en producción

- **Sanitización de imágenes** antes de subir a Cloudinary
  - **Validación de tokens** en middleware
- 

## **Testing Strategy**

### **Testing Stack**

- **Jest** para unit testing
- **React Testing Library** para component testing
- **Cypress** para E2E testing
- **Firebase Emulator** para testing de Firestore

### **Test Coverage Goals**

- **Components:** 90% coverage
- **Utils/Hooks:** 95% coverage
- **Critical User Flows:** 100% E2E coverage

### **Critical Test Cases**

typescript

```
describe('Appointment Creation Flow', () => {
  it('should prevent overlapping appointments', async () => {
    // Test conflict validation
  })

  it('should validate medical restrictions', async () => {
    // Test medical warnings
  })

  it('should calculate end time correctly', async () => {
    // Test time calculations
  })
})

describe('Client Registration', () => {
  it('should create client profile after auth', async () => {
    // Test registration flow
  })

  it('should validate medical info form', async () => {
    // Test medical form validation
  })
})
```

## Referencias y Recursos

### Documentación Técnica

- [Next.js 14 Documentation](#)
- [Firebase Documentation](#)
- [shadcn/ui Components](#)
- [Tailwind CSS](#)
- [Zustand State Management](#)

### Recursos de Diseño

- [Figma Design System](#)
- [Brand Guidelines](#)
- [Icon Library - Lucide](#)

### Herramientas de Desarrollo

- **VS Code Extensions:**



- ES7+ React/Redux/React-Native snippets
  - Tailwind CSS IntelliSense
  - Firebase Explorer
  - TypeScript Importer
- 

## Próximos Pasos

### Inmediatos (Esta Semana)

1. **Setup del proyecto** con todas las dependencias
2. **Configuración de Firebase** y colecciones base
3. **Implementación de autenticación** básica
4. **Layout principal** con navegación

### Corto Plazo (2-3 Semanas)

1. **Funcionalidad CRUD** completa para todas las entidades
2. **Sistema de citas** con validación de horarios
3. **Panel del cliente** funcional
4. **Validaciones médicas** automáticas

### Mediano Plazo (1-2 Meses)

1. **PWA** completamente funcional
  2. **Sistema de notificaciones** push
  3. **Analytics** y métricas implementadas
  4. **Testing** comprehensivo y documentación de usuario
- 

## Equipo y Roles

### Desarrollador Principal

- **Responsabilidades:** Implementación completa del frontend y backend
- **Skills requeridos:** Next.js, TypeScript, Firebase, React

### Diseñador UX/UI (Opcional)

- **Responsabilidades:** Refinamiento de interfaces y experiencia de usuario
- **Skills requeridos:** Figma, principios de UX móvil

### Tester/QA (Opcional)

- **Responsabilidades:** Testing manual y automatizado
  - **Skills requeridos:** Cypress, testing de aplicaciones móviles
- 

*Última actualización: Agosto 2025 Versión: 1.0 Estado: En desarrollo*