



MANUAL TÉCNICO (FRONT- END)

Citas Médicas Teruel

Versión
1.0

Grupo 6 gestión de citas médicas

Tabla de contenido

1. INTRODUCCIÓN	2
1.1. PROPÓSITO DEL MANUAL	2
1.2. ALCANCE	2
1.3. AUDIENCIA OBJETIVO.....	2
2. ARQUITECTURA GENERAL.....	2
2.1. PATRÓN ARQUITECTÓNICO	2
3. STACK TECNOLÓGICO	3
3.1. NÚCLEO DEL FRAMEWORK.....	3
3.2. INTERFAZ Y EXPERIENCIA DE USUARIO (UI/UX)	3
3.3. COMUNICACIÓN E INTEGRACIÓN	3
4. ESTRUCTURA DEL PROYECTO	4
4.1. ORGANIZACIÓN DE DIRECTORIOS (SRC / APP /)	4
5. MÓDULOS Y FUNCIONALIDADES PRINCIPALES	4
5.1. MÓDULO DE AUTENTICACIÓN (AUTH).....	4
5.2. MÓDULO ADMINISTRATIVO (ADMIN).....	4
5.3. MÓDULO MÉDICO (DOCTOR)	5
5.4. MÓDULO PACIENTE (PATIENT)	5
6. INTEGRACIÓN Y SERVICIOS	5
6.1. SERVICIOS BASE (GENERICSERVICE)	5
6.2. COMUNICACIÓN EN TIEMPO REAL (SOCKETSERVICE)	5
6.3. SEGURIDAD E INTERCEPTORS	5
7. DESPLIEGUE E INFRAESTRUCTURA.....	6
7.1. ESTRATEGIA DE BUILD	6
7.2. CONTENERIZACIÓN (DOCKER).....	6
7.3. PIPELINE CI/CD (JENKINS)	6
8. GUÍAS DE DESARROLLO Y CALIDAD.....	7
8.1. ESTÁNDARES DE CÓDIGO	7
8.2. GESTIÓN DE ESTADO	7
8.3. MANEJO DE VARIABLES DE ENTORNO	7

1. Introducción

1.1. Propósito del Manual

Este manual técnico proporciona la documentación oficial del frontend de la aplicación **Appointments Project**. Está diseñado para guiar a desarrolladores, arquitectos de software y personal de DevOps en la comprensión de la arquitectura, la implementación de componentes y los procesos de despliegue de la interfaz de usuario.

1.2. Alcance

El documento abarca desde la arquitectura modular y el patrón de componentes Standalone, hasta la integración con servicios de backend (REST y WebSocket), gestión de seguridad con JWT y los pipelines de despliegue automatizado.

1.3. Audiencia Objetivo

- Desarrolladores Frontend (Angular).
- Arquitectos de Soluciones.
- Ingenieros de DevOps y QA.

2. Arquitectura General

El proyecto implementa una arquitectura de Single Page Application (SPA) basada en los estándares más recientes de Angular (v19+), priorizando la modularidad y el rendimiento.

2.1. Patrón Arquitectónico

Se utiliza una estrategia de Carga Diferida (Lazy Loading) para optimizar el tiempo de carga inicial. La aplicación se divide en módulos funcionales (Feature Modules) que se cargan bajo demanda.

- Arquitectura Basada en Standalone Components: Se reduce la dependencia de **NgModules**, simplificando la inyección de dependencias y el *bootstrapping* de la aplicación.
- Separación de Responsabilidades:
 - Presentación: Componentes visuales (HTML/SCSS).
 - Lógica de Negocio: Servicios inyectables.
 - Estado: Gestión reactiva con RxJS (**BehaviorSubject**).

3. Stack Tecnológico

La selección de tecnologías prioriza la estabilidad, el tipado fuerte y la capacidad de tiempo real.

3.1. Núcleo del Framework

Tecnología	Versión	Propósito
Angular	19.2.0	Framework SPA principal.
TypeScript	5.7.2	Lenguaje base con tipado estático.
RxJS	7.8.0	Manejo de flujos de datos asíncronos y eventos.

3.2. Interfaz y Experiencia de Usuario (UI/UX)

- Angular Material (19.2.19): Sistema de diseño base (Material Design 3).
- SweetAlert2 (11.22.3): Feedback visual para alertas y confirmaciones transaccionales.
- Chart.js + ng2-charts: Visualización de datos (KPIs y estadísticas administrativas).

3.3. Comunicación e Integración

- SignalR Client (9.0.6): Cliente WebSocket para la gestión de bloqueos de citas en tiempo real.
- JWT Decode: Procesamiento de tokens para extracción de roles y expiración.
- File Saver & XLSX: Generación y descarga de reportes en el navegador.

4. Estructura del Proyecto

El código fuente sigue una estructura semántica que facilita la escalabilidad.

4.1. Organización de Directorios (`src/app/`)

```
src/app/
├── modules/ # Módulos de Funcionalidad (Lazy Loaded)
│   ├── auth/      # Login, Recuperación de contraseña
│   ├── admin/     # Gestión de Usuarios, Roles, Maestra
│   ├── doctor/    # Agenda médica y atención
│   ├── paciente/  # Reserva de citas y perfil
│   └── parameter/ # Configuración (Ciudades, EPS, etc.)
├── shared/ # Recursos Transversales
│   ├── components/ # Tablas, Botones, Layouts
│   ├── services/   # HTTP, Auth, Sockets
│   └── models/     # Interfaces TypeScript
├── core/ # (Opcional) Singleton services, Guards, Interceptors
├── app.routes.ts # Definición de rutas principales
└── app.config.ts # Configuración de proveedores globales
```

5. Módulos y Funcionalidades Principales

5.1. Módulo de Autenticación (Auth)

Gestiona el ingreso y seguridad inicial.

- Flujos: Login, Registro (Usuario + Persona), Reset Password.
- Lógica: Redirección inteligente basada en el rol decodificado del JWT (ej. un Doctor es redirigido a `/doctor/dashboard` tras el login).

5.2. Módulo Administrativo (Admin)

Panel de control centralizado.

- **Gestión de Usuarios:** CRUD completo con asignación de roles.

- **Configuración Médica:** Administración de especialidades, consultorios y horarios.
- **Dashboards:** Visualización de métricas de rendimiento utilizando Chart.js.

5.3. Módulo Médico (Doctor)

Herramienta de trabajo diaria para el especialista.

- **Agenda Interactiva:** Visualización de citas programadas.
- **Atención:** Registro de notas clínicas y cambio de estados de cita (Asistida/Cancelada/No Asistida).

5.4. Módulo Paciente (Patient)

Portal de autogestión.

- **Reserva de Citas:** Interfaz visual para selección de especialidad y horario.
- **Integración SignalR:** Al seleccionar un horario, este se bloquea visualmente para otros usuarios en tiempo real.

6. Integración y Servicios

6.1. Servicios Base (GenericService)

Se implementa un patrón de **Servicio Genérico** para estandarizar las peticiones HTTP. Esto reduce la duplicidad de código en operaciones CRUD estándar (Get, Post, Put, Delete).

6.2. Comunicación en Tiempo Real (SocketService)

El frontend mantiene una conexión persistente con el Hub de SignalR del backend.

- **Eventos Escuchados:** `SlotLocked`, `SlotReleased`, `AppointmentBooked`.
- **Acción:** Actualización reactiva de la interfaz (calendario) sin necesidad de recargar la página (F5).

6.3. Seguridad e Interceptors

- **AuthInterceptor:** Intercepta todas las peticiones HTTP salientes y adjunta el encabezado `Authorization: Bearer <token>`.

- **ErrorInterceptor:** Captura respuestas 401/403 para cerrar sesión automáticamente o redirigir al login si el token expira.

7. Despliegue e Infraestructura

7.1. Estrategia de Build

El proceso de construcción genera archivos estáticos optimizados.

- **Comando:** `npm run build -- --configuration production`
- **Optimizaciones:** *Tree-shaking* (eliminación de código muerto), minificación de JS/CSS y *hashing* de archivos para control de caché.

7.2. Contenerización (Docker)

El frontend se sirve a través de un servidor ligero **Nginx** dentro de un contenedor.

- **Dockerfile (Multi-stage):**
- **Stage Build:** Imagen Node.js para compilar el proyecto Angular.
- **Stage Run:** Imagen Nginx Alpine. Copia los artefactos de la carpeta `dist/` al directorio `/usr/share/nginx/html`.
- **Configuración Nginx:** Se incluye un archivo `nginx.conf` personalizado para manejar el enrutamiento SPA (redirigir todas las peticiones a `index.html` para evitar errores 404 al recargar).

7.3. Pipeline CI/CD (Jenkins)

El ciclo de vida de despliegue está automatizado:

1. **Checkout:** Obtención del código.
2. **Build & Test:** Compilación y ejecución de pruebas unitarias (`ng test`).
3. **Docker Build:** Creación de la imagen.
4. **Deploy:** Actualización del servicio en el entorno correspondiente (Dev/QA/Prod) mediante Docker Compose.

8. Guías de Desarrollo y Calidad

8.1. Estándares de Código

- Linting: Uso estricto de ESLint para asegurar consistencia.
- Nomenclatura: `camelCase` para variables/métodos, `PascalCase` para clases/interfaces.

8.2. Gestión de Estado

Se evita la complejidad de librerías externas (como NgRx) en favor de servicios con `BehaviorSubject` para estados simples (ej. usuario actual, tema, *loading spinner*), manteniendo la arquitectura ligera (**Principio KISS**).

8.3. Manejo de Variables de Entorno

No se queman credenciales ni URLs en el código. Se utilizan archivos `environment.ts` para desarrollo y sustitución de variables durante el *build* de producción o mediante inyección de configuración en tiempo de ejecución (`assets/env.js`) para despliegues Docker agnósticos.