MANUAL TÉCNICO (BACK-END)

Citas Médicas Teruel

Versión
2.0
Tecnología base: .NET 8.0 – API RESTful

Grupo 6 gestión de citas médicas

Tabla de contenido

Tabla de contenido

1. Introducción y Objetivo del Proyecto

1.1. Nombre del Proyecto

Appointments-Project-Back

1.2. Descripción General

El proyecto Appointments-Project-Back es el componente backend, desarrollado en **.NET 8.0**, para un sistema de gestión integral de citas médicas. Expone una robusta **API RESTful** diseñada para ser el núcleo de la administración de usuarios, doctores, agendas, citas y notificaciones en tiempo real, enfocándose en un entorno de instituciones de salud.

La API está diseñada para ser consumida por clientes frontend (web, móvil u otros), garantizando alta disponibilidad, seguridad mediante **JWT**, y gestión de concurrencia a través de **Redis y SignalR**.

1.3. Objetivo Estratégico

Proveer una API RESTful robusta y escalable, desarrollada con **.NET 8.0 y EF Core 9.0**, que garantice la gestión integral del ciclo de vida de las citas médicas (programación, modificación, cancelación y notificación). El sistema deberá asegurar la integridad de los datos mediante bloqueos transaccionales verificados y lograr una implementación funcional contenerizada en Docker lista para despliegue **antes del hito de entrega de la fase actual**.

2. Arquitectura del Sistema

El sistema utiliza una **Arquitectura en Capas (Layered Architecture)** que asegura la separación de responsabilidades, la modularidad y la facilidad de mantenimiento.

2.1. Visión General

El sistema utiliza principios de desarrollo moderno como **Inyección de Dependencias (DI)**, patrones de diseño (Repository, Business Layer),

autenticación **JWT**, **SignalR** para notificaciones en tiempo real y pruebas unitarias con **xUnit y Moq**.

Flujo de Comunicación: [Frontend/Angular] HTTP/HTTPS [Web_back (API)] → [Business-Back] → [Data-Back] → [Entity-Back (DbContext)]

2.2. Arquitectura en Capas

Capa	Proyecto .NET	Responsabilidades Clave
Presentación	**Web_back**	API RESTful, Controladores ASP.NET Core, Middlewares, Configuración de CORS y JWT.
Negocio	**Business-Back**	Lógica de negocio, validaciones, orquestación de servicios (ej. AppointmentOrchestrator), mapeo DTO ↔ Entidad.
Datos	**Data-Back**	Implementación del **Repository Pattern**, encapsulación del acceso a datos, manejo de consultas específicas.
Entidades	**Entity-Back**	Modelos de dominio, DTOs, Contexto de Entity Framework Core (ApplicationDbContext), definición de relaciones y esquemas.
Utilidades	**Utilities-Back**	Clases auxiliares, servicios compartidos, configuración de Mapster y manejo de excepciones personalizadas

		(**BusinessException**).

2.3. Componentes Principales

- **Autenticación y Autorización:** Implementación de **JWT Bearer** con refresh tokens, roles granulares y permisos basados en Rol-Formulario (**RBAC**).
- **Gestión de Citas:** Sistema de reservas avanzado con gestión de concurrencia mediante **Locks en Redis** para asegurar la atomicidad en la asignación de franjas horarias.
- **Notificaciones:** Servicio híbrido que utiliza **SignalR** para la comunicación bidireccional en tiempo real (in-app) y **SMTP** para la comunicación por correo electrónico (transaccional).
- **Base de Datos: Entity Framework Core** con soporte configurable para múltiples proveedores (**SQL Server, PostgreSQL, MySQL**).

2.4. Patrones de Diseño Utilizados

Patrón	Implementación	Propósito
Repository Pattern	Interfaces en Data-Back e implementaciones concretas.	Aislamiento de la capa de persistencia (EF Core) de la lógica de negocio.
Business Layer Pattern	Lógica centralizada en Business-Back.	Orquestación de flujos de trabajo, validaciones rigurosas y aplicación de reglas de negocio.
Dependency Injection	Configuración centralizada en Program.cs y métodos de extensión.	Alta modularidad y bajo acoplamiento entre capas.
DTO Pattern	Objetos de transferencia en Entity-Back/Dto.	Separación entre el modelo de dominio interno y la estructura de

		datos expuesta por la API.
Observer Pattern	Utilización de **SignalR Hubs**.	Notificación asíncrona y en tiempo real a los clientes sobre cambios críticos (ej. bloqueo de citas).

3. Estructura del Código

3.1. Organización de Proyectos

La solución de Visual Studio (**Appointments-Project-Back.sln**) se compone de los siguientes proyectos en .NET 8.0:

1. **Web_back:** Proyecto principal de API.
2. **Business-Back:** Lógica de negocio y orquestación.
3. **Data-Back:** Repositorios y acceso a datos.
4. **Entity-Back:** Modelos de dominio, DTOs, contexto de BD.
5. **Utilities-Back:** Clases utilitarias y manejo de excepciones.
6. **Business_Back.Tests:** Suite de pruebas unitarias (**xUnit/Moq**).
7. **Diagram:** Documentación y diagramas de arquitectura.

3.2. Modelos de Datos (Entity-Back)

Todas las entidades de dominio heredan de **BaseModel**, que define propiedades comunes como `Id` y `RegistrationDate`.

Esquema (Lógico)	Entidades Clave
ModelInfrastructure	`Departament`, `City`, `Institution`, `Branch`.
ModelSecurity	`Rol`, `Person`, `User`, `Form`, `Module`, `Permission`, `RolFormPermission`, `RolUser`.

Hospital	`DocumentType`, `Eps`, `Specialty`, `Doctor`, `ConsultingRoom`, `TypeCitation`.
MedicalShedule	`Schedule`, `ScheduleHour`, `Citation`.
dbo	`Notification`, `RefreshToken`, `RelatedPerson`.

3.3. Gestión de la Persistencia y Borrado Lógico

La gestión de la persistencia se realiza mediante **Entity Framework Core**.

3.3.1. Implementación de Borrado Lógico

La funcionalidad de borrado lógico se implementa en la **Capa de Datos (Data-Back)** dentro de los repositorios genéricos (`BaseModelData<T>`).

1. **Modelo de Entidad:** Las entidades implementan la propiedad booleana IsDeleted.
2. **Consulta:** Todas las consultas SELECT ejecutadas por los repositorios aplican globalmente un filtro (Where(e => !e.IsDeleted)) a través de **Query Filters** en el ApplicationDbContext.
3. **Operación de Borrado:** La operación DeleteAsync() en el repositorio **no elimina el registro físicamente**, sino que establece IsDeleted = true y actualiza la entidad en la base de datos.
4. **Ubicación Correcta:** La lógica de *soft-delete* es una implementación directa del acceso a datos, por lo tanto, pertenece a la capa **Data-Back**.

3.4. DTOs y Mapeos

- **DTOs:** Se utilizan objetos de transferencia de datos (AuthResultDto, CitationListDto, CitationCreateDto, etc.) para desacoplar las firmas de los endpoints de la API de los modelos de dominio.
- **Mapeo:** La librería **Mapster** se encarga del mapeo entre DTOs y entidades, con configuraciones definidas en Utilities-Back/Mapster/MapsterConfig.cs.

4. Tecnologías y Dependencias

4.1. Tecnologías Principales

Categoría	Tecnología	Versión	Uso Principal
Framework	.NET (Core)	8.0 (LTS)	Framework de desarrollo principal.
API	ASP.NET Core Web API	8.0	Exposición de endpoints RESTful.
ORM	Entity Framework Core	9.0.7	Acceso a datos relacionales (Multi-proveedor).
Bases de Datos	SQL Server	2022	BD principal (Soporte para PostgreSQL y MySQL).
Cache/Concurrencia	**Redis**	7.x	Locks distribuidos para gestión de citas.
Tiempo Real	**SignalR**	-	Notificaciones WebSockets bidireccionales.
Contenerización	**Docker/Compose**	20+	Entornos de desarrollo y despliegue.
Autenticación	**JWT Bearer**	-	Implementación de tokens de acceso y refresco.
Mapeo	**Mapster**	7.4.0	Mapeo objeto-objeto DTO ↔ Entidad.

4.2. Paquetes NuGet Críticos

- Microsoft.EntityFrameworkCore
- Microsoft.AspNetCore.Authentication.JwtBearer
- Microsoft.Extensions.Caching.StackExchangeRedis
- Pomelo.EntityFrameworkCore.MySql
- Npgsql.EntityFrameworkCore.PostgreSQL (Proveedores multi-BD)
- BCrypt.Net-Next (Hashing de contraseñas)
- Swashbuckle.AspNetCore (Documentación Swagger/OpenAPI)
- xUnit / Moq (Pruebas unitarias)

5. Configuración y Despliegue

5.1. Despliegue con Docker

El proyecto utiliza Docker para la contenerización, facilitando el despliegue en cualquier entorno.

- **Dockerfile:** Utiliza un *multi-stage build* para optimizar el tamaño de la imagen final, incluyendo solo el *runtime* de **.NET 8.0**.
- **Docker Compose:** Orquesta los servicios de api, sqlserver (instancia de BD) y redis, conectándolos a una red Docker interna (appointments_network).

5.2. Entornos y Configuración

El comportamiento del sistema es gestionado por el entorno de ASP.NET Core (`ASPNETCORE_ENVIRONMENT`).

- **Archivos de Configuración:** appsettings.json, appsettings.Development.json, appsettings.Production.json.
- **Variables de Entorno Clave:** Conexiones a BD, parámetros Redis, credenciales SMTP para correo, y secretos JWT (Key, Issuer, Audience).

5.3. Pipeline CI/CD (Jenkins)

Se utiliza **Jenkins** para la Integración Continua y el Despliegue Continuo.

- **Jenkinsfile:** Define los pasos automatizados: *checkout*, *build* de la solución .NET, construcción de la imagen Docker de la API, *push* al *registry*, y ejecución del *docker-compose* en el servidor de destino.
- **Configuración de Producción:** El despliegue de producción utiliza **Nginx**

como *proxy* reverso para manejar la terminación SSL (puerto 443) y redirigir el tráfico al contenedor de la API (puerto 5001).

6. Seguridad y Concurrencia

6.1. Autenticación (JWT)

- **Mecanismo:** Login mediante email/password. Se genera un par de *tokens*: un **Token de Acceso** (corta duración, ej., 60 minutos) y un **Refresh Token** (larga duración, ej., 7 días) para la rotación segura de sesiones.
- **Hashing:** Las contraseñas se almacenan con **BCrypt.Net-Next**, garantizando un *hash* seguro con *salt* aleatorio.

6.2. Autorización (RBAC)

Se implementa un modelo de **Control de Acceso Basado en Roles (RBAC)** con granulosidad.

- **Roles:** SuperAdmin, Admin, Doctor, Usuario (Paciente).
- **Permisos Granulares:** La seguridad se valida mediante las entidades Rol, Form y Permission, permitiendo controlar el acceso a funciones específicas (ej., solo un Doctor puede acceder a /api/doctor-agenda).

6.3. Concurrencia y Locks (Redis)

Para garantizar que dos usuarios no reserven la misma cita simultáneamente, se utiliza un mecanismo de bloqueo distribuido.

- **Proceso:** Al iniciar una reserva, se adquiere un **lock distribuido en Redis** para el *slot* de la cita.
- **Duración:** El lock tiene un **TTL (Time-To-Live) configurable (ej., 120 segundos)**.
- **Atomicidad:** La operación de confirmación de cita (ConfirmAsync) es transaccional (Validación → Registro en BD → Notificación → Liberación de Lock).

7. Servicios y Funcionalidades Clave

7.1. Módulos Funcionales

Módulo	Servicios Clave	Descripción
Autenticación	AuthService	Login, Registro, Refresh Token, Recuperación de Contraseña.
Citas	CitationBusiness, AppointmentOrchestrator	Reserva con concurrencia, gestión de estados (Programada, Cancelada, etc.), validación de agenda.
Notificaciones	NotificationService	Envío de correos SMTP y transmisión de eventos en tiempo real vía SignalR.
Maestras	DoctorBusiness, SpecialtyBusiness	CRUD para entidades de apoyo (Doctores, Consultorios, Especialidades).
Reportes	DashboardService	Cálculo de KPIs y estadísticas de citas para la gestión administrativa.

7.2. Notificaciones y Comunicación

- **In-App (SignalR):** Utilizado para el *broadcast* instantáneo de eventos críticos, como el bloqueo o la liberación de un *slot* de cita.
- **Email (SMTP):** Utilizado para notificaciones transaccionales (confirmación de cita, recordatorios, recuperación de contraseña), con plantillas HTML para un formato profesional.

8. Pruebas y Calidad

8.1. Suite de Pruebas Unitarias

- **Framework: xUnit**.
- **Mocking: Moq**.
- **Objetivo:** El proyecto Business_Back.Tests se enfoca en validar la lógica de negocio contenida en la capa **Business-Back**, incluyendo reglas de disponibilidad, autenticación y flujos de orquestación, asegurando la integridad de los datos.

9. Términos y Condiciones

A continuación, se presenta una interpretación técnica y funcional de las áreas de un proyecto de gestión de citas que se relacionan con los Términos y Condiciones (T&C).

Área de T&C	Implicación Técnica/Funcional en el Backend
1. Privacidad de Datos (HIPAA/GDPR)	Hashing de contraseñas (**BCrypt**), uso de **JWT** para sesiones seguras, control de Autorización (**RBAC**) para limitar el acceso a datos sensibles (ej., historial médico solo visible por el Doctor/Paciente/Admin autorizado).
2. Responsabilidad de la Información	Validaciones de Negocio rigurosas en la capa **Business-Back** para garantizar que solo se persistan datos coherentes (ej., una cita no puede ser programada en el pasado o con un doctor inhabilitado).
3. Disponibilidad y Servicio (SLA)	Despliegue en Docker para rápida recuperación, uso de **Redis Locks** para evitar sobre-reserva (*overbooking*), y la implementación de un Monitoreo (recomendado en la sección 10) para medir y garantizar el *uptime*.

4. Modificaciones del Servicio	Diseño modular del código (Arquitectura en Capas) que permite la actualización y extensión de funcionalidades (ej., añadir un nuevo proveedor de BD o un nuevo método de pago) con impacto mínimo.
5. Terminación de la Cuenta	Implementación del **Borrado Lógico (Soft-Delete)** para inhabilitar al usuario (IsDeleted=true) en lugar de eliminar el registro, permitiendo la restauración de datos o la auditoría posterior, según el T&C.

10. Actualizaciones

Nuevas versiones del backend:

- Compilación y generación de nueva imagen Docker.
- Despliegue gradual (staging → producción).
- Migraciones de base de datos controladas.