Documentación: API Móviles 2 - Kingdom Barber

Fecha: Octubre, 2025
Autores: Juan Rivera, Andrés Vallejo, Alejandro Urrego

1. Resumen del Proyecto

Esta API es el **cerebro y la fuente única de verdad** para todo el ecosistema de aplicaciones de Kingdom Barber. Desarrollada con **Java y Spring Boot**, su propósito es centralizar toda la lógica de negocio y la persistencia de datos, sirviendo información de manera consistente a múltiples clientes.

Actualmente, esta API da servicio a dos aplicaciones cliente:

1. **pi_web2.0 (Front-End Web):** Una aplicación moderna en Next.js donde los clientes pueden ver información, agendar citas, ver la galería y contactar a la barbería.
2. **pi_ntp2.0 (Dashboard de Análisis):** Una aplicación en Python/Streamlit que consume grandes volúmenes de datos históricos para generar visualizaciones, reportes y análisis de negocio.

La arquitectura desacopla completamente el back-end de los front-ends, permitiendo que cada pieza del sistema evolucione de forma independiente.

2. Objetivos del Proyecto

Objetivo Principal

Centralizar toda la lógica de negocio y la persistencia de datos del ecosistema Kingdom Barber en una única API RESTful robusta, segura y escalable.

Objetivos Específicos

- **Proveer Endpoints Claros:** Exponer un conjunto de endpoints RESTful bien definidos para todas las operaciones necesarias (CRUD de citas, gestión de galería, consulta de datos maestros, etc.).
- **Desacoplar Clientes:** Eliminar la dependencia de los clientes (web y dashboard) sobre el método de almacenamiento de datos, permitiendo que la capa de datos pueda ser modificada o escalada sin afectar a los front-ends.
- **Garantizar la Consistencia:** Asegurar que todas las aplicaciones consuman y modifiquen la misma fuente de datos, evitando inconsistencias y duplicidad.
- **Establecer una Base Escalable:** Crear una fundación sólida que permita en el futuro añadir nuevos clientes, como una aplicación móvil, sin tener que reinventar la lógica del

back-end.

3. Stack Tecnológico

- **Lenguaje:** Java 17+
- **Framework Principal:** Spring Boot 3.x
- **Acceso a Datos:** Spring Data JPA / Hibernate
- **Base de Datos:** H2 Database (Base de datos en memoria para desarrollo y pruebas)
- **Servidor Web:** Apache Tomcat (embebido en Spring Boot)
- **Gestor de Dependencias:** Maven
- **Utilidades:** Lombok (para reducir código repetitivo en los modelos)

4. Arquitectura y Estructura de Carpetas

El proyecto sigue una arquitectura de API REST estándar, organizada por responsabilidades:

```
src/main/java/com/kingdombarber/api/
├── 📁 controller/   # Reciben peticiones HTTP y definen las URLs (endpoints).
│   ├── AgendamientoController.java
│   ├── ContactoController.java
│   ├── DashboardController.java
│   ├── DatosMaestrosController.java
│   └── GaleriaController.java
│
├── 📁 model/        # Clases @Entity que representan las tablas de la base de datos.
│   ├── Barbero.java
│   ├── Cita.java
│   ├── Cliente.java
│   ├── Contacto.java
│   ├── Galeria.java
│   ├── NuevaCita.java
│   ├── Sede.java
│   └── Servicio.java
│
├── 📁 repository/   # Interfaces que extienden JpaRepository para las operaciones CRUD.
│   ├── BarberoRepository.java
│   ├── CitaRepository.java
│   └── ... (y los otros 6 repositorios)
│
├── 📜 ApiApplication.java   # Punto de entrada principal de la aplicación.
├── 📜 WebConfig.java        # Configuración de CORS y para servir archivos estáticos.
└── 📜 RequestLoggingInterceptor.java # Middleware para registrar peticiones.

src/main/resources/
```

```
├── 📜 application.properties # Configuración de la base de datos, servidor, etc.
└── 📜 data.sql               # Script SQL para la carga inicial de datos.
```

5. Descripción de Endpoints Principales

La API está organizada en los siguientes controladores:

DatosMaestrosController (Datos compartidos)

- GET /sedes: Devuelve la lista de todas las sedes.
- GET /barberos: Devuelve la lista de todos los barberos.
- GET /servicios: Devuelve la lista de todos los servicios.

DashboardController (Para pi_ntp2.0)

- GET /historial/citas: Devuelve el historial completo de citas (~4000 registros).
- GET /clientes: Devuelve la lista completa de clientes.

AgendamientoController (Para pi_web2.0)

- GET /citas-activas: Devuelve las citas del calendario.
- POST /citas-activas: Crea una nueva cita.
- PUT /citas-activas/{id}: Modifica una cita existente.
- DELETE /citas-activas/{id}: Elimina una cita.

ContactoController (Para pi_web2.0)

- POST /contactanos: Recibe y guarda un nuevo mensaje del formulario de contacto.

GaleriaController (Para pi_web2.0)

- GET /galeria: Devuelve la lista de imágenes de la galería.
- POST /galeria/upload: Sube un nuevo archivo de imagen junto con su descripción y categoría.
- PUT /galeria/{id}: Modifica la información de una imagen.
- DELETE /galeria/{id}: Elimina una imagen (de la base de datos y del disco).

6. Flujo de Datos Típico: Creación de una Cita

1. **Cliente Front-End (pi_web2.0):** El usuario rellena el formulario de agendamiento y hace clic en "Confirmar Cita".
2. **JavaScript (React):** El componente captura los datos, construye un objeto JSON y utiliza la **Fetch API** para enviar una petición POST al endpoint http://localhost:8080/citas-activas.
3. **Controlador (API Java):** La clase AgendamientoController recibe la petición en el método anotado con @PostMapping("/citas-activas"). Spring Boot convierte automáticamente el cuerpo JSON en un objeto NuevaCita.
4. **Lógica de Negocio:** El controlador enriquece el objeto NuevaCita (por ejemplo,

buscando el nombre de la sede a partir del sedeId) y luego invoca al repositorio.

5. **Repositorio (JPA):** NuevaCitaRepository utiliza el método .save() heredado de JpaRepository para generar e ejecutar una sentencia INSERT INTO en la base de datos H2.
6. **Respuesta del Back-End:** Una vez guardada, el método .save() devuelve la entidad completa (con su id si es autogenerado). El Controller la devuelve al front-end como un JSON con un código de estado 200 OK.
7. **Actualización del Front-End:** El código de React recibe la respuesta, actualiza su estado interno para añadir el nuevo evento al calendario y muestra una notificación de éxito al usuario.