Documentación: API Móviles 2 - Kingdom Barber

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

1. Resumen del Proyecto

Esta API constituye el núcleo central y la fuente única de verdad para todo el ecosistema de aplicaciones de Kingdom Barber. Desarrollada sobre una base robusta de Java y Spring Boot, su propósito es centralizar la lógica de negocio, la seguridad y la persistencia de datos, sirviendo información de manera consistente a múltiples clientes a través de una arquitectura RESTful.

La arquitectura actual desacopla completamente el back-end de los front-ends, permitiendo que cada componente del sistema evolucione de forma independiente y garantizando la mantenibilidad a largo plazo. La API está desplegada en **Render** y da servicio a dos aplicaciones cliente:

1. **Aplicación Web (pi_web2):** Una aplicación moderna en Next.js desplegada en **Vercel**, que provee la interfaz para que los clientes agenden citas y exploren la oferta de la barbería.
2. **Dashboard de Análisis (pi_ntp):** Una aplicación en Python/Streamlit desplegada en **Streamlit Cloud**, que consume datos para generar visualizaciones, reportes y análisis avanzados con Inteligencia Artificial.

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 que sirva como la fundación para todas las operaciones presentes y futuras.
- **Objetivos Específicos:**
    - **Proveer Endpoints Claros:** Exponer un conjunto de endpoints RESTful bien definidos y documentados para todas las operaciones CRUD y de negocio.
    - **Desacoplar Clientes:** Abstraer por completo la capa de persistencia, permitiendo que la base de datos pueda ser modificada o escalada sin impactar a las aplicaciones cliente.
    - **Garantizar la Consistencia de Datos:** Asegurar que todas las aplicaciones consuman y modifiquen una única fuente de verdad, eliminando inconsistencias y duplicidad de información.
    - **Establecer una Base Escalable:** Crear una arquitectura sólida que permita en el futuro integrar nuevos clientes (ej. una aplicación móvil nativa) sin necesidad de reescribir la lógica del servidor.

3. Stack Tecnológico

- **Lenguaje:** Java 17+
- **Framework Principal:** Spring Boot 3.x
- **Acceso a Datos:** Spring Data JPA / Hibernate
- **Base de Datos (Desarrollo):** H2 Database (en memoria).
- **Base de Datos (Producción):** PostgreSQL (gestionada por Render).
- **Servidor Web:** Apache Tomcat (embebido en Spring Boot).
- **Gestor de Dependencias:** Maven
- **Contenerización:** Docker
- **Utilidades:** Lombok
- **Plataforma de Despliegue:** Render

4. Arquitectura y Estructura de Carpetas

El proyecto sigue una arquitectura de API REST estándar, organizada por capas de responsabilidad para facilitar la mantenibilidad y el testeo.

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
│ └── ... (6 modelos más)
│
├── 📁 repository/ # Interfaces que extienden JpaRepository para las operaciones CRUD.
│ ├── BarberoRepository.java
│ ├── CitaRepository.java
│ └── ... (6 repositorios más)
│
├── 📜 ApiApplication.java # Punto de entrada de la aplicación Spring Boot (@SpringBootApplication).
├── 📜 WebConfig.java # Configuración centralizada de CORS para permitir peticiones cross-origin.
└── 📜 RequestLoggingInterceptor.java # Middleware para interceptar y registrar peticiones HTTP entrantes.

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 URL base para todos los endpoints en producción es: **https://pi-movil2-0.onrender.com**

- **DatosMaestrosController:** Sirve datos fundamentales que cambian con poca frecuencia.
  - GET /sedes, GET /barberos, GET /servicios
- **DashboardController:** Provee endpoints optimizados para el consumo masivo de datos por parte del panel de análisis.
  - GET /historial/citas, GET /clientes
- **AgendamientoController:** Contiene la lógica transaccional para la gestión de citas por parte de los clientes.
  - GET, POST, PUT, DELETE /citas-activas
- **ContactoController:** Endpoint para la funcionalidad de contacto.
  - POST /contactanos
- **GaleriaController:** Gestiona el portafolio de trabajos de los barberos.
  - GET, POST, PUT, DELETE /galeria. La subida de imágenes se realiza convirtiendo el archivo a Base64 y almacenándolo como texto en la base de datos. Esta estrategia simplifica la gestión de archivos en un entorno de despliegue contenerizado.

6. Flujo de Datos Típico: Creación de una Cita

1. **Cliente Front-End (pi_web2):** El usuario interactúa con la interfaz de React para seleccionar un servicio, barbero y horario.
2. **Petición HTTP:** Al confirmar, se envía una petición POST con un cuerpo JSON al endpoint de producción: https://pi-movil2-0.onrender.com/citas-activas.
3. **Controlador (API Java):** AgendamientoController recibe la petición. Spring Boot deserializa automáticamente el JSON en un objeto DTO (Data Transfer Object) de Java.
4. **Lógica de Negocio:** El controlador valida los datos, enriquece el objeto (ej. asignando la entidad Sede completa a partir de su ID) y llama al repositorio correspondiente.
5. **Repositorio (JPA):** El método save() de JpaRepository es invocado. Hibernate traduce esta llamada a una sentencia INSERT INTO optimizada para la base de datos **PostgreSQL** de producción.
6. **Respuesta del Back-End:** La base de datos confirma la transacción. El método save() devuelve la entidad completa con su ID generado, la cual es serializada a JSON por el Controller y devuelta al cliente con un código de estado 200 OK.
7. **Actualización del Front-End:** La aplicación React recibe la respuesta, actualiza su estado local y la interfaz se re-renderiza para reflejar la cita recién creada.

7. Despliegue y Puesta en Producción

- **Contenerización:** El proyecto incluye un Dockerfile que empaqueta la aplicación Java y

todas sus dependencias en una imagen de contenedor ligera y portable. Esto asegura que la aplicación se ejecute de la misma manera en cualquier entorno.

- **Plataforma:** La API está desplegada en **Render**. El servicio está configurado para construir y desplegar directamente desde el Dockerfile del repositorio.
- **Despliegue Continuo (CI/CD):** Render está vinculado al repositorio de GitHub del proyecto. Un push a la rama principal (main) dispara automáticamente un nuevo "build" de la imagen Docker y un despliegue de la nueva versión, minimizando el tiempo de inactividad.
- **Variables de Entorno:** Las credenciales de la base de datos PostgreSQL y otras configuraciones sensibles se gestionan de forma segura a través de las variables de entorno de Render, garantizando que no se expongan en el código fuente.
- **URL Pública:** La API es accesible globalmente a través de la URL: https://pi-movil2-0.onrender.com