Documentación Técnica: KINGDOM BARBER (PI_MOVIL2)

1. Resumen del Proyecto

Kingdom Barber (PI_MOVIL2) es una aplicación web robusta desarrollada con **Spring Boot (Java)** para el backend y **Vanilla JavaScript** para el frontend. La plataforma está diseñada para simplificar y modernizar la gestión de citas en una barbería, ofreciendo portales dedicados y funcionales para clientes y barberos.

El proyecto demuestra una sólida implementación de una API RESTful y una arquitectura backend bien definida, conectada a una interfaz de usuario dinámica construida con tecnologías web fundamentales.

2. Arquitectura y Estructura de Carpetas

La aplicación sigue una arquitectura cliente-servidor clásica. El backend, construido con Spring Boot, sigue el patrón **Modelo-Vista-Controlador (MVC)** adaptado para una API REST.

Backend (Spring Boot)

```
src/
└── main/
    ├── java/com/kingdombarber/api/
    │   ├── 📁 controller/      # Endpoints REST. Reciben peticiones HTTP y devuelven JSON.
    │   │   ├── BarberoController.java
    │   │   └── CitaController.java
    │   │
    │   ├── 📁 model/           # Entidades JPA. Representan las tablas de la base de datos.
    │   │   ├── Barbero.java
    │   │   ├── Cita.java
    │   │   └── Cliente.java
    │   │
    │   └── 📁 repository/      # Interfaces de Spring Data JPA para operaciones CRUD.
    │       ├── BarberoRepository.java
    │       └── CitaRepository.java
    │
    └── resources/
        ├── 📁 static/          # Archivos del Frontend (HTML, CSS, JS)
        ├── data.sql            # Script para la carga inicial de datos en la BD.
        └── application.properties # Configuración de la BD, servidor, etc.
```

Frontend (Vanilla JS, HTML, CSS)

```
src/
└── main/
    └── resources/
        └── static/
            ├── 📁 barbero/
            │   ├── agenda.html
            │   └── barbero.js
            │
            ├── 📁 cliente/
            │   ├── citas.html
            │   └── cliente.js
            │
            ├── 📁 css/
            │   └── style.css
            │
            ├── index.html      # Página principal (selector de rol)
            └── login.js        # Lógica para la selección de rol
```

3. Funcionalidades Principales

👤 Portal del Cliente

- **Reservar Cita:** Un formulario intuitivo permite a los clientes seleccionar la sede, el barbero, el servicio y la fecha/hora para agendar una nueva cita.
- **Visualizar Barberos:** Los clientes pueden explorar la lista de barberos disponibles en cada sede.
- **Gestión de Citas:** Permite a los usuarios modificar los detalles de una reserva existente o cancelarla si es necesario.

✂️ Portal del Barbero

- **Inicio de Sesión por Sede:** Los barberos acceden a su panel personal seleccionando primero su sede y luego su nombre de una lista.
- **Visualización de Agenda:** Muestra una lista clara y ordenada de todas las citas agendadas, con la información esencial (fecha, hora, cliente, servicio).
- **Detalles de Cita:** Al hacer clic en una cita, se accede a una vista detallada con toda la información relevante del servicio y del cliente.
- **Cancelación de Citas:** Los barberos tienen la capacidad de cancelar citas directamente desde su agenda.

4. Stack Tecnológico

- **Backend:**
  - **Framework:** Spring Boot
  - **Lenguaje:** Java
  - **Persistencia de Datos:** Spring Data JPA, Hibernate
  - **Base de Datos:** H2 (en memoria para desarrollo) o cualquier base de datos SQL compatible.
- **Frontend:**
  - **Lenguajes:** JavaScript (ES6+), HTML5, CSS3
  - **Librerías/Frameworks:** Ninguno (desarrollo con Vanilla JS).
  - **Comunicación:** Fetch API para peticiones asíncronas al backend.

5. Conceptos Clave

- **Spring Boot:** Framework del ecosistema Spring que facilita la creación de aplicaciones Java autocontenidas y listas para producción con una configuración mínima.
- **Arquitectura MVC (para API REST):**
  - **Modelo:** Las clases de entidad (Ej: Cita.java) que definen la estructura de los datos.
  - **Vista:** La representación de los datos, que en este caso es JSON.
  - **Controlador:** Las clases que manejan las peticiones HTTP (@RestController), procesan la lógica de negocio y devuelven la respuesta JSON.
- **Spring Data JPA:** Proyecto de Spring que simplifica enormemente la capa de persistencia de datos, permitiendo crear repositorios CRUD sin necesidad de escribir implementaciones SQL.
- **Vanilla JS:** Término que se refiere al uso de JavaScript puro, sin librerías o frameworks como React, Angular o Vue.
- **Fetch API:** Interfaz moderna de JavaScript para realizar peticiones de red (HTTP) de forma asíncrona, reemplazando al antiguo XMLHttpRequest.

6. Flujo de Datos: Creación de una Cita

1. **Cliente (Frontend):** El usuario rellena el formulario de reserva en citas.html y hace clic en "Agendar".
2. **JavaScript (cliente.js):** El script captura los datos del formulario, crea un objeto JSON y utiliza la Fetch API para enviar una petición POST al endpoint /citas del backend.
3. **Controlador (Backend):** CitaController.java recibe la petición en su método anotado con @PostMapping. El cuerpo JSON de la petición se convierte automáticamente en un objeto Cita.
4. **Lógica y Servicio:** El controlador invoca al repositorio (CitaRepository).
5. **Repositorio (JPA):** CitaRepository utiliza el método .save() de Spring Data JPA para insertar una nueva fila en la tabla de citas de la base de datos.
6. **Respuesta del Backend:** Una vez guardada, la base de datos devuelve el objeto creado (con su nuevo ID). El controlador lo encapsula en una ResponseEntity y lo envía de vuelta

al frontend como JSON con un código de estado 201 CREATED.

7. **Actualización de UI (Frontend):** El .then() de la promesa fetch en cliente.js recibe la respuesta. El script manipula el DOM para mostrar un mensaje de confirmación al usuario.