Proyecto Integrador: Kingdom Barber

Integrantes:

Alejandro Urrego

Andrés Vallejo

Juan Manuel Rivera

Institución:

Institución Técnica CESDE

Fecha:

Octubre, 2025

1. Introducción y Visión General

1.1. Resumen del Proyecto

El proyecto Kingdom Barber representa la evolución hacia una plataforma digital integral, diseñada para transformar y modernizar la gestión de barberías. Su objetivo es ofrecer una solución robusta que simplifique la administración de citas, enriquezca la interacción con el cliente y proporcione herramientas avanzadas para la toma de decisiones.

La solución se compone de un ecosistema de aplicaciones cohesivo y desacoplado que funciona de manera sinérgica, donde cada componente está desplegado en una plataforma optimizada para su tecnología:

- **Una API Central (Java + Spring Boot):** El único cerebro de la operación, desplegado en **Render**. Centraliza toda la lógica de negocio y la persistencia de los datos.
- **Una Plataforma Web para Clientes (Next.js + React):** La cara visible del negocio, desplegada en **Vercel**. Garantiza una experiencia de usuario moderna y fluida para la exploración de servicios y la reserva de citas.
- **Un Panel de Gestión y Análisis (Python + Streamlit):** La herramienta de inteligencia de negocio, desplegada en **Streamlit Cloud**. Ofrece una visión profunda del rendimiento del negocio e integra un potente Asistente de Inteligencia Artificial.

1.2. Planteamiento del Problema

La gestión tradicional de barberías enfrenta desafíos en la era digital, como la dependencia de procesos manuales para agendar citas, una comunicación ineficiente con los clientes y una carencia fundamental de análisis de datos para la toma de decisiones estratégicas. Esto resulta en una menor eficiencia operativa, pérdida de oportunidades de negocio y una experiencia que no cumple con las expectativas actuales de los clientes, quienes buscan comodidad, inmediatez y personalización.

1.3. Solución Propuesta

Kingdom Barber aborda estos desafíos mediante una arquitectura de software moderna que digitaliza y optimiza las operaciones. Al centralizar el back-end en una única y robusta API de Java, se garantiza la consistencia, seguridad y escalabilidad de los datos. La plataforma ofrece a los clientes una interfaz web de última generación para agendar y gestionar sus citas de forma autónoma, mientras que el panel administrativo empodera a los dueños del negocio con herramientas de análisis de datos e Inteligencia Artificial para identificar patrones, detectar oportunidades de mercado y automatizar tareas de marketing.

2. Objetivos del Proyecto

2.1. Objetivo General

Desarrollar un ecosistema de software completo y desacoplado que modernice la gestión operativa de Kingdom Barber, mejore la experiencia del cliente a través de una plataforma

web intuitiva y proporcione herramientas de inteligencia de negocios para la toma de decisiones estratégicas basadas en datos.

2.2. Objetivos Específicos

- **Centralizar la Lógica de Negocio:** Construir una API RESTful en Java con Spring Boot que actúe como la única fuente de verdad para todos los datos del sistema.
- **Ofrecer una Experiencia de Usuario Moderna:** Desarrollar una aplicación web cliente (pi_web2) con Next.js y React que permita a los usuarios registrarse, explorar servicios, agendar citas en un calendario interactivo y gestionar su perfil.
- **Empoderar la Gestión con Datos:** Crear un panel de administración (pi_ntp) en Python con Streamlit que ofrezca dashboards interactivos, KPIs en tiempo real y la capacidad de filtrar y exportar datos históricos.
- **Integrar Inteligencia Artificial:** Implementar un Asistente de IA (Gemini) en el panel de gestión para generar análisis, responder preguntas en lenguaje natural y proponer estrategias de negocio.
- **Análisis de Mercado Externo:** Añadir una funcionalidad para analizar y visualizar datasets públicos de datos.gov.co para entender el contexto competitivo.
- **Garantizar la Escalabilidad y Mantenibilidad:** Diseñar el sistema bajo una arquitectura de servicios desacoplados, donde el back-end y los front-ends puedan evolucionar de forma independiente.

3. Arquitectura General del Sistema

El proyecto está diseñado bajo una arquitectura de servicios desacoplados. El núcleo del sistema es la **API Central de Java**, que sirve como única fuente de verdad para todos los clientes front-end, garantizando que los datos sean consistentes y seguros.

- **API Central (Back-End - pi_movil2):** Desarrollado en Java y Spring Boot, desplegado en **Render**. Gestiona todos los datos (clientes, citas, barberos, etc.) a través de una base de datos **PostgreSQL** y expone la lógica de negocio a través de endpoints RESTful. Es el único componente con acceso a la capa de persistencia de datos.
- **Front-End (Plataforma del Cliente - pi_web2):** Desarrollado con Next.js (React), desplegado en **Vercel**. Es un cliente puro que consume los datos de la API de Java para ofrecer la experiencia al usuario final (agendamiento, galería, contacto).
- **Front-End (Panel de Administración - pi_ntp):** Aplicación web independiente en Python y Streamlit, desplegada en **Streamlit Cloud**. Se conecta a la misma API Central para realizar análisis de datos, generar visualizaciones y reportes para la gerencia.

4. Módulos del Proyecto

4.1. Módulo 1: API Central (Java + Spring Boot)

- **Descripción:** Es el corazón del sistema. Construido con Spring Boot, gestiona toda la lógica, los datos y la seguridad a través de una API REST. Utiliza Spring Data JPA para la persistencia en una base de datos PostgreSQL en producción.

- **Funcionalidades Clave:**
  - Endpoints RESTful para el CRUD completo de todas las entidades del negocio.
  - Manejo de subida de archivos (imágenes de la galería), convirtiéndolas a formato Base64 para su almacenamiento directo en la base de datos.
  - Configuración de CORS para permitir la comunicación segura con los clientes front-end desde diferentes orígenes (Vercel, Streamlit Cloud).
  - Lógica de negocio para enriquecer los datos, como añadir nombres de sede y barbero a las citas antes de guardarlas.

4.2. Módulo 2: Plataforma Web para Clientes (pi_web2)

- **Descripción:** La cara visible del proyecto, construida con Next.js, React y TypeScript para una experiencia de usuario excepcional y optimizada para SEO.
- **Funcionalidades Clave:**
  - Sistema de autenticación de usuarios robusto con **Clerk**.
  - Calendario interactivo para agendar, modificar y cancelar citas en tiempo real.
  - Galería de trabajos, formulario de contacto y visualización de sedes, todo consumiendo datos de la API de Java.
  - Panel de agenda para que los barberos puedan visualizar las citas que les han sido asignadas.

4.3. Módulo 3: Panel de Gestión y Análisis (pi_ntp)

- **Descripción:** Herramienta estratégica en Python con Streamlit, diseñada para el análisis de datos y la administración del negocio.
- **Funcionalidades Clave:**
  - Dashboard interactivo con KPIs, filtros y gráficos de rendimiento en tiempo real.
  - Gestión y consulta del historial completo de citas con filtros avanzados.
  - **Asistente de Inteligencia Artificial (Gemini)** con una suite de herramientas: generador de reportes, analista interactivo, asistente de marketing y asesor de estilo.
  - **Módulo de Análisis de Datasets Reales** para realizar estudios de mercado sobre el sector de la belleza en Colombia utilizando datos de datos.gov.co.

5. Stack Tecnológico Consolidado

Capa	Tecnología	Propósito
Back-End (API Central)	Java, Spring Boot, Spring Data JPA, PostgreSQL, Docker	Lógica de negocio, API RESTful, persistencia de datos
Front-End (Cliente)	Next.js, React, TypeScript, Tailwind CSS, Clerk	Interfaz de usuario para clientes y barberos
Análisis de Datos	Python, Streamlit, Pandas,	Panel de administración, BI,

	Plotly, Gemini	Asistente de IA

6. Dificultades y Lecciones Aprendidas

El desarrollo de un ecosistema multi-tecnológico presentó varios desafíos que proporcionaron valiosas lecciones:

- **Desafíos de Integración:** La principal dificultad fue asegurar una comunicación fluida entre el back-end de Java y los front-ends de React y Python. Se presentaron obstáculos como:
  - **CORS (Cross-Origin Resource Sharing):** Fue necesario implementar y depurar una configuración de CORS robusta en la API de Java para permitir las peticiones desde los dominios de Vercel y Streamlit Cloud.
  - **Inconsistencias de Nomenclatura:** La convención camelCase de Java/JSON chocaba con la snake_case a veces usada en Python. Se solucionó implementando una "capa de traducción" en data_manager.py para normalizar los datos, resaltando la importancia de definir convenciones desde el inicio.
  - **Build Linters:** Durante el despliegue en Vercel, el linter de ESLint forzó la corrección de código, como la eliminación de importaciones no utilizadas y el ajuste de dependencias en hooks de React, mejorando la calidad general del código.
- **Retos en la Persistencia de Datos:** La decisión de almacenar imágenes como **Base64** en la base de datos, aunque más compleja inicialmente, eliminó por completo los problemas de rutas y CORS para los archivos estáticos, resultando en una solución más robusta para un entorno contenerizado.
- **Comunicación del Equipo y "Contrato de API":** Al tener tres proyectos interconectados, se hizo evidente la necesidad crítica de mantener un "contrato de API" claro y estable. Cualquier cambio en un endpoint o en la estructura de un JSON en el back-end tenía un impacto directo en los front-ends, reforzando la importancia de la comunicación constante y de las pruebas de integración.

7. Conclusiones del Proyecto

La decisión de migrar y consolidar el ecosistema bajo una única API Central en Java ha sido el paso más importante hacia la creación de una solución tecnológica robusta, escalable y profesional.

- **Evolución del Diseño:** Se transitó de un modelo fragmentado a una arquitectura de servicios limpia y desacoplada. Esto permite que cada front-end evolucione de forma independiente y se especialice en su función, consumiendo una única fuente de verdad.
- **Consistencia y Escalabilidad:** Al tener una única fuente de datos gestionada por una API robusta, se eliminan las inconsistencias y se garantiza la integridad de la información. Esta base sólida permite añadir futuros clientes (como una aplicación móvil nativa) de manera mucho más sencilla.
- **Logro Técnico:** El proyecto demuestra con éxito la integración de tres stacks

tecnológicos distintos (Java/Spring, Python/Streamlit, y JavaScript/React/Next.js) en un ecosistema cohesivo. Se aplicaron principios de software fundamentales como la separación de responsabilidades, la comunicación por API y la gestión de estado en el cliente.

El proyecto ha entregado un ecosistema tecnológico funcional, moderno y cohesivo, sentando las bases para futuras expansiones y demostrando un dominio de arquitecturas de software contemporáneas.