Descripción del Proyecto

El objetivo principal del proyecto es crear una aplicación web que facilite a los pacientes la programación de citas médicas. La plataforma debe ser intuitiva, segura y escalable, permitiendo la gestión eficiente de información tanto para pacientes como para doctores.

Tecnologías Utilizadas

Backend
Tecnologías Core:

- Node.js + TypeScript: Lenguaje y entorno de ejecución para desarrollar el servidor.
- Express.js: Framework para construir la API REST.
- Prisma: ORM para gestionar la interacción con la base de datos.

Arquitectura:

- Patrón Repository: Para separar la lógica de acceso a datos de la lógica de negocio.
- Controladores REST: Manejo de las solicitudes HTTP.
- Sistema de Rutas Modular: Organización de las rutas de la API de manera escalable.
- Middleware Personalizado: RateLimiter y PerfromanceMiddleware.
- Manejo de Errores Centralizado: Gestión uniforme de errores en toda la aplicación.

Frontend
Framework y Core:

- Next.js 14 con App Router: Framework React para aplicaciones web con renderizado del lado del servidor.
- React 18: Biblioteca para construir interfaces de usuario.
- TypeScript: Superset de JavaScript que añade tipado estático.
- Tailwind CSS: Framework de CSS utilitario para estilos rápidos y responsivos.

UI y Componentes:

- shadcn/ui: Biblioteca de componentes UI personalizados.
- Sistema de Componentes Modular: Componentes reutilizables y mantenibles.
- Layouts Reutilizables: Estructuras de página consistentes.
- Axios: Cliente HTTP para realizar peticiones.

Estructura de Carpetas:

- /app - Rutas y páginas.
- /components - Componentes reutilizables.
- /hooks - Custom hooks.
- /lib - Utilidades.

Bases de Datos y Docker

Contenedores:

- MongoDB Container: Base de datos NoSQL.
- PostgreSQL Container: Base de datos relacional.
- Volúmenes Persistentes: Almacenamiento de datos persistente.
- Redes Docker Aisladas: Comunicación segura entre contenedores.

MongoDB:

- Colecciones: Patients, Doctors, Appointments.
- Índices Optimizados: Mejora del rendimiento de consultas.
- Esquema Flexible: Adaptabilidad de la estructura de datos.
- Relaciones mediante Referencias: Enlaces entre diferentes colecciones.

Metodología de Desarrollo

El desarrollo del proyecto se estructuró en sprints, siguiendo una metodología ágil que permitió adaptarse a los requerimientos y mejorar continuamente la arquitectura y funcionalidades de la aplicación.

Estructura de Sprints

Cada sprint tuvo una duración de 2-3 semanas y se enfocó en objetivos específicos, garantizando entregas funcionales y mejoras incrementales en cada iteración.

Detalle de Sprints

Sprint 1: Configuración Inicial del Backend

Objetivos:

- Configurar el entorno de desarrollo.
- Establecer la estructura básica del backend.
- Configurar Docker para PostgreSQL.

Actividades Realizadas:

Configuración del Entorno:

- Instalación de Node.js y TypeScript.
- Inicialización del proyecto con npm init.
- Configuración de TypeScript (tsconfig.json).
- Configuración de Docker:
- Creación de contenedores para PostgreSQL.
- Configuración de volúmenes persistentes para la base de datos.
- Establecimiento de redes Docker aisladas para comunicación segura.

Configuración de Express.js:

- Instalación y configuración de Express.js.
- Configuración de scripts de inicio y desarrollo.

Integración de Prisma:

- Instalación de Prisma y configuración inicial.
- Definición del esquema de datos para PostgreSQL.
- Generación de migraciones iniciales.

Resultados:

- Backend configurado y listo para el desarrollo de funcionalidades.
- PostgreSQL corriendo en un contenedor Docker con acceso configurado.

Sprint 2: Implementación del CRUD con PostgreSQL

Objetivos:

- Implementar operaciones CRUD para las entidades principales (Patients, Doctors, Appointments).
- Establecer rutas y controladores básicos.

Actividades Realizadas:
Modelo de Datos:

- Definición de modelos Prisma para Patients, Doctors y Appointments.
- Migración de la base de datos.

Patrón Repository:

- Implementación de repositorios para cada entidad.
- Métodos básicos de acceso a datos (crear, leer, actualizar, eliminar).

Controladores REST:

- Desarrollo de controladores para manejar solicitudes CRUD.
- Validación de datos de entrada.

Rutas Modulares:

- Configuración de rutas para cada entidad.
- Uso de middleware para manejo de autenticación y logging.

Frontend Inicial:

- Configuración básica de Next.js con páginas para listar y gestionar entidades.
- Implementación de formularios básicos para CRUD.

Resultados:

- Sistema completamente funcional con operaciones CRUD implementadas para PostgreSQL.
- Sin embargo, se utilizó el formato de Next.js que comparte el backend y frontend en la misma aplicación sin una diferenciación clara entre ambos, lo que complicó la escalabilidad y mantenibilidad.

Reflexión:Este sprint mostró la importancia de planificar la arquitectura desde el inicio. La falta de separación entre frontend y backend resultó en una base de código menos modular, lo que dificultó la implementación de mejoras posteriores.

Sprint 3: Integración del Frontend con Next.js
Objetivos:

- Mejorar la interacción entre el frontend y el backend.
- Optimizar la gestión de estado y las peticiones HTTP.

Actividades Realizadas:
Configuración de Axios:

- Instalación y configuración de Axios para manejar peticiones HTTP al backend.

Componentes Reutilizables:

- Desarrollo de componentes modulares con shadcn/ui.
- Creación de layouts reutilizables para consistencia en la interfaz.

Optimización de Páginas:

- Mejoras en la experiencia de usuario mediante la implementación de revalidaciones y actualizaciones en tiempo real.
- Estilización con Tailwind CSS:
- Refinamiento de estilos para una interfaz más atractiva y responsiva.

Resultados:

- Frontend más robusto y eficiente en la gestión de datos.
- Mejoras significativas en la experiencia de usuario gracias a la optimización de componentes y estilos.

Reflexión: Axios mejoró considerablemente la eficiencia de las peticiones y la gestión del estado, facilitando una interacción más fluida entre frontend y backend.

Sprint 4: Optimización de la Arquitectura y Pruebas de SQL
Objetivos:

- Reestructurar el proyecto para cumplir con la arquitectura por capas de repositorios.
- Optimizar las consultas SQL realizando pruebas con y sin índices.

Actividades Realizadas:

Reestructuración de la Arquitectura:

- Implementación del Patrón Repository para separar la lógica de acceso a datos de la lógica de negocio.
- Refactorización de controladores para interactuar con los repositorios.
- Modularización adicional del sistema de rutas para mejorar la mantenibilidad.

Optimización de Consultas SQL:
- Análisis de consultas actuales para identificar posibles cuellos de botella.
- Implementación de índices en PostgreSQL y realización de pruebas de rendimiento.
- Comparación de tiempos de respuesta con y sin índices.

Resultados:
- Arquitectura más robusta y mantenible gracias a la separación de responsabilidades.
- Mejora en el rendimiento de las consultas SQL mediante la optimización de índices, reduciendo tiempos de respuesta y aumentando la eficiencia de la base de datos.

Reflexión: La reestructuración de la arquitectura permitió una mayor modularidad y facilidad de mantenimiento. Las pruebas de optimización de consultas SQL demostraron una mejora significativa en el rendimiento, validando la importancia de una base de datos bien estructurada.

Sprint 5: Migración a MongoDB
Objetivos:
- Migrar la base de datos de PostgreSQL a MongoDB.
- Adaptar la arquitectura existente para soportar la nueva base de datos.

Actividades Realizadas:
Planificación de la Migración:
- Análisis de las diferencias entre PostgreSQL y MongoDB.
- Identificación de las modificaciones necesarias en los modelos de datos.

Implementación con Prisma:
- Configuración de Prisma para trabajar con MongoDB.
- Adaptación de los repositorios para manejar el esquema flexible de MongoDB.

Migración de Datos:
- Desarrollo de scripts para migrar datos de PostgreSQL a MongoDB.
- Verificación de la integridad y consistencia de los datos migrados.

Refactorización Mínima:
- Reutilización de la estructura de repositorios creada en el Sprint 4.
- Realización de cambios mínimos en el código gracias a la arquitectura modular.

Resultados:
- Migración exitosa a MongoDB, mejorando la flexibilidad y escalabilidad de la base de datos.

- Proceso de migración simplificado gracias a Prisma y la arquitectura previamente establecida, permitiendo la reutilización de componentes con mínimos cambios.

Reflexión: La planificación cuidadosa y la arquitectura modular implementada en sprints anteriores facilitaron enormemente la migración a MongoDB. Prisma demostró ser una herramienta esencial para manejar diferentes tipos de bases de datos sin incurrir en grandes refactorizaciones.

Sprint 6: Pruebas de Rendimiento y Profiling
Objetivos:
- Realizar pruebas de perfilado y carga para asegurar la estabilidad y el rendimiento de la aplicación.
- Implementar mejoras basadas en los resultados de las pruebas.

Actividades Realizadas:
Implementación de Herramientas de Profiling:
- Configuración de Profiling Clinic para identificar cuellos de botella en el código.
- Integración de Autocannon para realizar pruebas de carga y evaluar la capacidad de la aplicación bajo diferentes niveles de tráfico.

Ejecución de Pruebas de Carga:
- Realización de pruebas de carga con distintos escenarios de tráfico.

Monitoreo de respuestas del sistema y recopilación de métricas de rendimiento.
- Análisis y Optimización:
- Análisis de los resultados obtenidos de las pruebas de profiling y carga.
- Implementación de optimizaciones en el código y la configuración del servidor para mejorar el rendimiento.

Documentación de Resultados:
- Generación de informes detallados sobre los resultados de las pruebas.
- Registro de las optimizaciones realizadas y sus impactos en el rendimiento.

Resultados:
- Obtención de buenos resultados en las pruebas de carga y profiling, asegurando que la aplicación puede manejar un alto volumen de solicitudes sin degradar su funcionamiento.
- Identificación y resolución de cuellos de botella, mejorando la eficiencia general de la aplicación.
- La migración a MongoDB se ejecutó de manera sencilla gracias a Prisma y la estructura modular, permitiendo reutilizar componentes con mínimos cambios.

Reflexión: Las pruebas de rendimiento y profiling fueron cruciales para garantizar la estabilidad y eficiencia de la aplicación en entornos de producción. Los buenos resultados obtenidos

validan la efectividad de las optimizaciones implementadas y la robustez de la arquitectura del proyecto.

Reflexiones y Retos

Durante el desarrollo del proyecto, enfrentamos varios desafíos que fueron superados gracias a una planificación cuidadosa y una metodología ágil:

Separación de Frontend y Backend:

- Reto: Inicialmente, utilizamos el formato de Next.js que compartía el backend y frontend en la misma aplicación, lo que complicó la escalabilidad.
- Solución: En sprints posteriores, reestructuramos la arquitectura para separar claramente las responsabilidades del frontend y backend, adoptando una arquitectura por capas de repositorios que facilitó futuras migraciones y mejoras.

Migración de Base de Datos:

- Reto: Migrar de PostgreSQL a MongoDB podría haber sido complejo debido a las diferencias entre bases de datos relacionales y NoSQL.
- Solución: Gracias a Prisma y a la organización modular implementada en el Sprint 4, la migración fue sencilla, permitiendo reutilizar gran parte del código con mínimos cambios.

Optimización de Rendimiento:

- Reto: Garantizar que la aplicación soportara un alto volumen de tráfico sin degradar su rendimiento.
- Solución: Implementamos herramientas de profiling y pruebas de carga que nos permitieron identificar y solucionar cuellos de botella, obteniendo buenos resultados en las pruebas de rendimiento.

Gestión de Estado y Peticiones HTTP:

- Reto: Manejar eficientemente el estado de la aplicación y las peticiones al backend.
- Solución: Axios mejoró significativamente la gestión de datos y el rendimiento de las peticiones HTTP.

Conclusiones

El desarrollo de la web app para agendar citas con doctores se realizó de manera estructurada y eficiente mediante la utilización de tecnologías modernas y una metodología ágil basada en sprints. La adopción del Patrón Repository y el uso de Prisma como ORM permitieron una arquitectura limpia y escalable, facilitando futuras expansiones y mantenimientos.

La migración de PostgreSQL a MongoDB se ejecutó sin contratiempos gracias a la planificación y la arquitectura previamente establecida, demostrando la importancia de una buena estructura de código desde las etapas iniciales del proyecto. Además, la implementación de pruebas de rendimiento aseguró que la aplicación cumple con los estándares necesarios para operar en un entorno de producción robusto.

En resumen, el proyecto alcanzó sus objetivos iniciales, proporcionando una solución efectiva para la gestión de citas médicas, y estableció una base sólida para futuras mejoras y escalabilidad.