UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro

PROYECTO DE DESARROLLO DE APLICACIONES MULTIPLATAFORMA

GESTIÓN DE CONSULTAS MÉDICAS

CICLO FORMATIVO DE GRADO SUPERIOR

DESARROLLO DE APLICACIONES MULTIPLATAFORMA (IFCS02)

CURSO 2024-2025

Autor/a/es:
Maximiliano Benjamín Ramos Aguirre

Tutor/a:
Federico Banda Sierra

DEPARTAMENTO DE INFORMÁTICA Y COMUNICACIONES

IES LUIS VIVES

Resumen

Gestión de Consultas Médicas es un proyecto orientado a facilitar la gestión de citas médicas en centros de salud mediante una solución multiplataforma. Se compone de una aplicación móvil para pacientes (desarrollada en Kotlin para Android), un portal web para la gestión de citas por parte del personal sanitario (en React.js) y una API RESTful (en ASP.NET Core) que conecta ambos entornos. Este sistema permite reservar, modificar y cancelar consultas médicas, así como gestionar usuarios y agendas médicas. El objetivo es mejorar la experiencia del paciente, optimizar los recursos del centro médico y contribuir a la digitalización del sector sanitario.

Abstract

Medical Appointment Management is a cross-platform project aimed at improving appointment scheduling in healthcare centers. It includes an Android mobile app for patients (developed in Kotlin), a web portal for healthcare staff (developed in React.js), and a RESTful API (built with ASP.NET Core) that integrates both platforms. The system allows users to book, modify, and cancel appointments, and enables doctors to manage their schedules. The project seeks to enhance patient experience, optimize resources, and support the digital transformation of the healthcare industry.

Índice

1. INTRODUCCIÓN

En la actualidad, los centros de salud deben enfrentarse a innumerables problemáticas con la gestión de eficiencia de citas al médico, la organización de la agenda de los médicos y la atención oportuna de sus pacientes. Las soluciones más tradicionales, las que dependen de procesos manuales o de herramientas que están desconectadas, acaban generando un importante número de retrasos, errores administrativos y una experiencia poco favorable para los pacientes y para el personal médico.

Este proyecto, que se ha desarrollado en el contexto del Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Multiplataforma (DAM), tiene como finalidad la generación de un sistema multiplataforma para la gestión de la consulta médica que permita modernizar todos estos procesos a partir de tecnologías actuales. La solución propuesta está caracterizada por:
Una API RESTful, creada en ASP.NET Core, que es el motor del sistema y que expone funcionalidades seguras y escalables para todos los soportes.
Una aplicación móvil para los pacientes, desarrollada a partir de Kotlin, con la que los usuarios pueden visualizar fácilmente sus citas con médicos desde su terminal Android.
Un sitio web para médicos y gestores, desarrollado en React.js, el cual condiciona la organización de la agenda médica, el manejo de usuarios y la supervisión administrativa general. La finalidad de este proceso simultáneo de desarrollo de estas tres plataformas es el de aportar una solución digital total, plenamente disponible y eficiente, que ofrezca la mejor experiencia de uso de los pacientes y facilite la experiencia de uso de los profesionales de la medicina. El proyecto ha sido diseñado y desarrollado por elementos de trabajo a través de una metodología iterativa, que ha ido adaptándose a las necesidades que se han ido detectando durante el desarrollo de este.

1.1. OBJETIVO

El objetivo principal que sustenta la operación de este proyecto está cimentado en el diseño y desarrollo de una solución digital de tipo multiplataforma que gestione de forma adecuada las distintas consultas médicas de un centro de salud determinado, de forma que se facilite la gestión e interacción que puedan llevar a cabo paciente-salud, mediante la dotación de herramientas tecnológicas de última generación.
A través del desarrollo del sistema que se posteriormente se va a describir dentro del marco de este proyecto se persiguen los siguientes objetivos específicos:

Objetivos Técnicos

- El objetivo principal del proyecto es el siguiente: Crear una API RESTful, robusta y segura, con ASP.NET Core en el backend y Entity Framework Core como herramienta de acceso a datos.
- Crear una aplicación móvil Android, desarrollada en Kotlin, que permita a los pacientes consultar y gestionar sus citas médicas.
- Crear un portal web responsive en React.js y Tailwind CSS que permita a los doctores y administradores gestionar el sistema desde cualquier dispositivo con acceso a Internet.
- Crear una estructura de base de datos relacional eficiente y escalable con SQL Server.
- Asegurar la sincronización de las tres plataformas establecidas en el API común.

Objetivos de Aprendizaje

- Profundizar en el uso de frameworks modernos tanto en el backend (ASP.NET Core) como en el frontend (React.js, Kotlin).
- Consolidar conocimientos sobre desarrollo de aplicaciones móviles y su conexión con APIs externas.
- Aprender a estructurar un proyecto real aplicando principios de arquitectura limpia y buenas prácticas de desarrollo.
- Utilizar herramientas de planificación como diagramas de Gantt y metodologías de desarrollo iterativo.
- Aplicar técnicas de autenticación y autorización seguras (por ejemplo, JWT) para proteger el acceso a los datos del sistema.

Objetivos Funcionales

- Mejorar la organización interna de un centro médico a través de la digitalización de procesos.
- Reducir los tiempos de espera y la carga administrativa mediante una gestión automatizada de citas.
- Facilitar a los pacientes el acceso a sus consultas desde cualquier lugar y en cualquier momento.
- Proporcionar a los doctores una herramienta intuitiva para visualizar y gestionar su agenda.
- Ofrecer a los administradores una plataforma integral para el control y supervisión del sistema completo.

1.2.ALCANCE

El proyecto se delimitará en las siguientes áreas y funcionalidades, considerando un desarrollo que sea alcanzable en un período de 3 meses:

Funcionalidades Básicas (Obligatorias)

Backend (API RESTful en ASP.NET Core):

- Gestión de usuarios (pacientes, médicos, administradores).
- CRUD de citas médicas y gestión de horarios.
- Autenticación y autorización (mediante JWT).
- Conexión a una base de datos para el almacenamiento seguro de datos.

Aplicación Móvil (Android con Kotlin):

- Registro e inicio de sesión para pacientes.
- Visualización de disponibilidad de citas.

Portal Web de Administración:

- Panel de control para la gestión de consultas y agendas médicas.
- Módulo de administración para registrar y actualizar datos de pacientes y profesionales.
- Funcionalidades CRUD para servicios y citas.
- Creación de archivos PDF de citas.

1.3.JUSTIFICACIÓN

En el contexto actual, muchos sistemas de salud aún presentan importantes limitaciones en la gestión digital de citas médicas, lo que se traduce en largos tiempos de espera, sobrecarga administrativa y deficiencias en la comunicación entre pacientes y profesionales sanitarios. Esta situación afecta negativamente tanto a la calidad del servicio como a la experiencia del usuario.

Este proyecto surge como respuesta a dicha problemática, proponiendo una solución ágil, moderna e intuitiva que permita:

- Mejorar la calidad y eficiencia del servicio médico.
- Reducir la carga de trabajo administrativo mediante la automatización de procesos.
- Ofrecer a pacientes y doctores una experiencia digital más fluida y accesible.
- Impulsar la transformación digital en el ámbito sanitario, especialmente en centros de salud de menor escala o sin sistemas informatizados.

2. IMPLEMENTACIÓN

2.1.ANÁLISIS DE LA APLICACIÓN

La aplicación permitirá a los pacientes gestionar sus citas de forma sencilla y a los profesionales médicos visualizar y organizar sus agendas.

2.1.1. Requisitos funcionales de la aplicación

- Registro e inicio de sesión de usuarios.
- Visualización de disponibilidad y horarios.
- Reserva, modificación y cancelación de citas.
- Gestión de usuarios, citas y agendas desde el portal web.

2.1.2. Análisis y selección de las tecnologías

Se han valorado varias tecnologías y se han elegido las siguientes:

- **Backend:** ASP.NET Core (robustez, integración con Entity Framework Core, buen soporte para APIs REST).
- **Base de datos:** PostgreSQL (compatibilidad y fiabilidad).
- **Frontend Web:** React.js (popularidad, rendimiento, componentes reutilizables).
- **App Móvil:** Kotlin (idioma oficial Android, moderno, seguro).
- **Despliegue:** Vercel (Portal Web) y Render (Api y BBDD).

2.1.3. Planificación de la realización del proyecto

Fase 1: Análisis y Diseño (Semanas 1-2)

- **Semana 1 a Semana 2:**
    - Recolección de requisitos y definición del alcance.
    - Diseño de la arquitectura del sistema (diagramas UML, modelo entidad-relación para la base de datos).
    - Especificación de la interfaz de usuario para la aplicación móvil y el portal web.

Fase 2: Desarrollo del Backend (Semanas 3-5)

- **Semana 3 a Semana 5:**
    - Implementación del API RESTful en ASP.NET Core.
    - Configuración y conexión de la base de datos.
    - Desarrollo de funcionalidades de autenticación y autorización.
    - Pruebas iniciales utilizando herramientas como Postman.

Fase 3: Desarrollo de la Aplicación Móvil (Semanas 4-7)

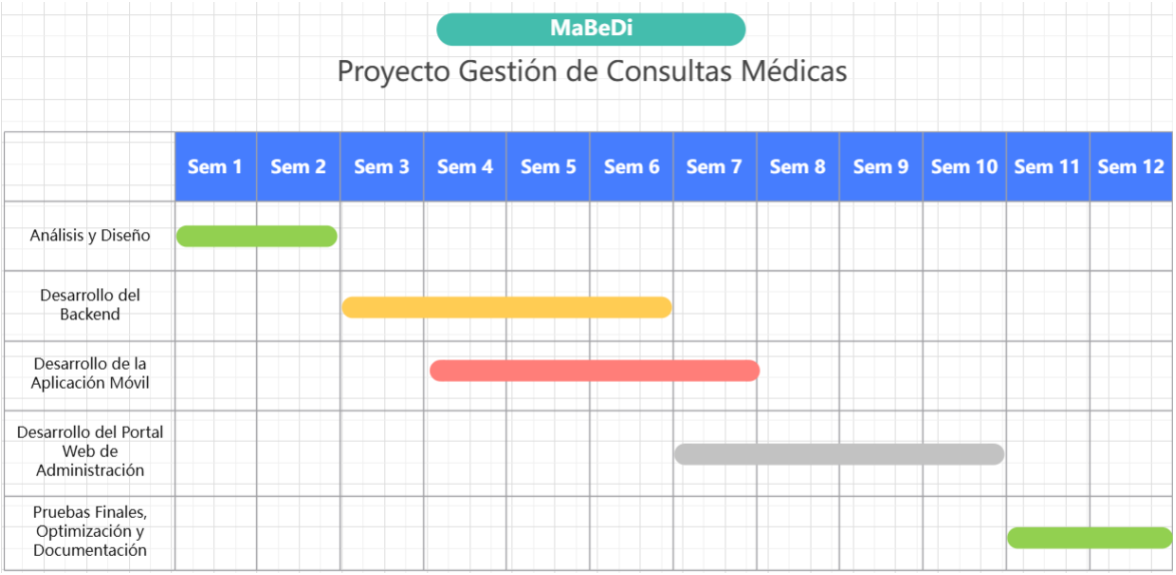
- **Semana 4 a Semana 7:**
    - Creación de la interfaz de usuario (uso de Fragments y RecyclerView).
    - Integración de la aplicación móvil con el API para la gestión de citas.
    - Implementación de funcionalidades de registro, reserva y notificaciones.

Fase 4: Desarrollo del Portal Web de Administración (Semanas 7-10)

- **Semana 7 a Semana 10:**
    - Desarrollo del panel de control para la gestión de consultas médicas.
    - Implementación de funcionalidades CRUD y generación de informes.
    - Realización de pruebas integradas entre el portal web, la aplicación móvil y el API.

Fase 5: Pruebas Finales, Optimización y Documentación (Semanas 11-12)

- **Semana 11 a Semana 12:**
    - Ejecución de pruebas de integración y usabilidad.
    - Optimización del rendimiento y corrección de errores.
    - Redacción de la documentación final y preparación del informe del proyecto.



2.1.4. Otros

Presupuesto del Proyecto

Este presupuesto estima los costes necesarios para el desarrollo completo del proyecto durante un período de 12 semanas, considerando recursos humanos, infraestructura tecnológica y otros gastos relacionados.

1. Recursos Humanos

Actividad	Horas estimadas	Tarifa por hora (€)	Coste (€)
Análisis y diseño	40	25	1,000
Desarrollo Backend	80	25	2,000
Desarrollo Frontend Web	60	25	1,500

Desarrollo App Móvil	60	25	1,500
Testing y corrección	30	20	600
Documentación y entrega	20	20	400
Total Recursos Humanos	290		7,000

2. Infraestructura y Servicios

Concepto	Coste mensual (€)	Meses	Coste total (€)
Hosting backend (Render)	10	3	30
Hosting web (Vercel)	0	3	0
Base de datos PostgreSQL	15	3	45
Servicio email SMTP	5	3	15
Licencias software	0	0	0
Total Infraestructura			90

3. Hardware y Otros Gastos

Concepto	Coste (€)
Amortización PC de desarrollo	125
Materiales y papelería	50
Total Hardware y Otros	175

4. Resumen Final del Presupuesto

Categoría	Coste (€)
Recursos Humanos	7,000
Infraestructura y Servicios	90
Hardware y Otros	175
TOTAL GENERAL	7,265

- ## Notas

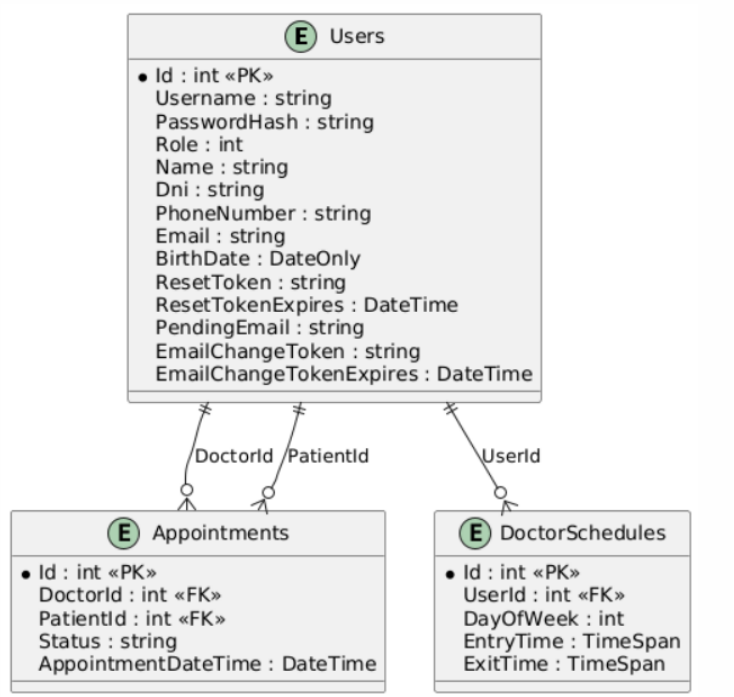
• Las tarifas por hora se han ajustado a niveles de un desarrollador junior o becario.
• Se han utilizado servicios con planes gratuitos o de bajo coste para minimizar gastos.
• El coste humano está valorado para dar una idea profesional, aunque en un TFG es trabajo personal no remunerado.
• Este presupuesto es una aproximación para un desarrollo profesional realista en el tiempo estipulado.

2.2. DISEÑO

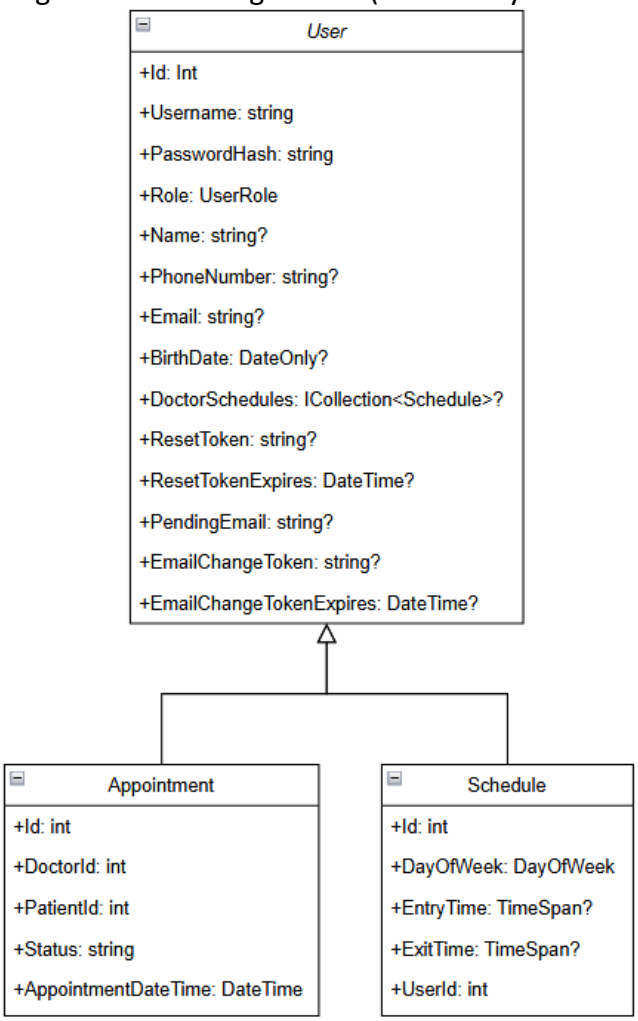
También tendrá sus propios apartados. A modo de ejemplo podremos servirnos de:

- Prototipado: Diseño de pantallas (no el resultado final), bocetos, Mockups

- Diagramas E/R o modelo de datos



- Diagramas de clase genérico (Atributos y métodos principales)



- Casos de uso (El desarrollo de cada caso de uso en anexos)

Caso de Uso: Registrar nuevo doctor o paciente

Campo	Contenido
Nombre	Registrar nuevo doctor o paciente
Actor principal	Administrador
Descripción	El administrador, desde el portal web, accede a un formulario y registra un nuevo usuario (doctor o paciente), enviando los datos a la API.
Precondiciones	- El administrador está autenticado. - Tiene rol de administrador.
Flujo principal	1. El administrador accede al formulario web. 2. Ingresa los datos del doctor o paciente. 3. El sistema envía una solicitud POST a la API. 4. La API verifica que el usuario que hace la petición tenga rol 'Administrador'. 5. Si es válido, se registra el nuevo usuario. 6. Se actualiza la base de datos y se notifica éxito.
Flujos alternativos	4a. Si no tiene rol de administrador, se devuelve un error 403. 5a. Si los datos están incompletos o inválidos, se devuelve error 400.
Postcondiciones	- El nuevo usuario queda registrado en la base de datos.

Caso de Uso: Iniciar sesión

Campo	Contenido
Actor principal	Usuario (Administrador, Doctor, Paciente)
Descripción	El usuario introduce su correo/usuario y contraseña. El sistema valida sus credenciales y permite el acceso si tiene un rol válido según el entorno.
Precondiciones	- Usuario registrado con credenciales válidas.
Flujo principal	1. El usuario accede al formulario de login. 2. Introduce correo o usuario y contraseña. 3. La API valida las credenciales. 4. Si son correctas y el rol es válido para el entorno (portal web: admin o doctor; app móvil: paciente), se inicia sesión.
Flujos alternativos	3a. Si las credenciales son incorrectas, se devuelve error. 4a. Si el rol no corresponde con el entorno, se deniega el acceso.
Postcondiciones	- Usuario autenticado y con sesión iniciada.

Caso de Uso: Cambiar contraseña o correo electrónico

Campo	Contenido
Actor principal	Doctor o Paciente (logeado)
Descripción	El usuario accede a su perfil, edita contraseña o correo mediante un formulario que solicita la contraseña actual y nueva información.
Precondiciones	- Usuario autenticado (doctor o paciente).
Flujo principal	1. El usuario accede a la pestaña Perfil. 2. Hace clic en 'Cambiar contraseña' o 'Cambiar correo'. 3. Rellena el formulario con datos válidos. 4. La API verifica los datos actuales. 5. Si son válidos, actualiza la contraseña o correo.
Flujos alternativos	4a. Si la contraseña actual es incorrecta, se devuelve error. 3a. Si las nuevas contraseñas no coinciden, se devuelve error.
Postcondiciones	- La información se actualiza correctamente.

Caso de Uso: Ver pacientes, doctores y citas

Campo	Contenido
Actor principal	Administrador / Doctor
Descripción	El sistema muestra una lista de pacientes, doctores y citas dependiendo del rol del usuario.
Precondiciones	- Usuario autenticado.
Flujo principal	1. El sistema detecta el rol tras el login. 2a. Si es administrador, carga todas las listas de usuarios y citas. 2b. Si es doctor, carga sus citas propias, su perfil y su horario.
Postcondiciones	- Los datos visibles varían según el rol del usuario.

Caso de Uso: Crear cita

Campo	Contenido
Actor principal	Administrador / Doctor
Descripción	El usuario accede a un formulario de creación de citas. El admin selecciona doctor y paciente; el doctor solo selecciona paciente.
Precondiciones	- Usuario autenticado.
Flujo principal	1. Usuario hace clic en 'Crear cita'. 2. Completa el formulario. 3. La API registra la cita según el rol del usuario. 4. Se actualiza la base de datos.
Postcondiciones	- La cita queda registrada y visible según corresponda.

Caso de Uso: Cancelar cita

Campo	Contenido
Actor principal	Administrador / Doctor
Descripción	El usuario elimina una cita desde la tabla de citas. El admin puede cancelar cualquier cita; el doctor solo las suyas.
Precondiciones	- Usuario autenticado.
Flujo principal	1. Usuario accede a la tabla de citas. 2. Pulsa el botón eliminar. 3. La API valida el permiso y elimina la cita.
Postcondiciones	- La cita ya no está disponible en la base de datos.

Caso de Uso: Actualizar estado de la cita

Campo	Contenido
Actor principal	Administrador / Doctor
Descripción	El usuario modifica el estado de una cita a 'Programada', 'Completada' o 'Cancelada'.
Precondiciones	- Usuario autenticado.
Flujo principal	1. Usuario accede a la cita. 2. Cambia el campo 'Estado' desde un desplegable. 3. Pulsa 'Guardar'. 4. La API actualiza el estado en la base de datos.
Postcondiciones	- La cita actualiza su estado correctamente.

- Diagramas de flujo de datos: Pueden complementar aquellos casos de uso complejos.

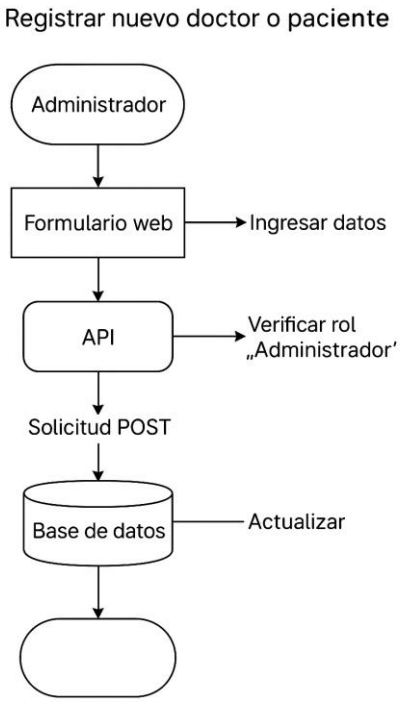
Flujo de Datos: Registrar nuevo doctor o paciente

Actores y componentes involucrados:

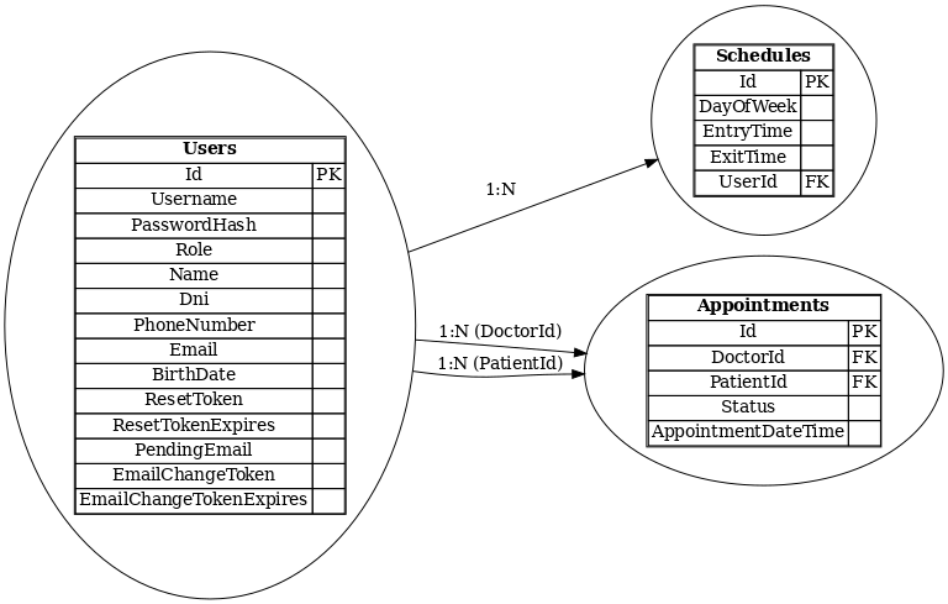
- Administrador (usuario)
- Formulario Web (frontend)
- API Backend
- Base de Datos

Flujo:

1. Administrador accede al formulario de registro.
2. Introduce los datos del nuevo usuario (doctor o paciente).
3. El formulario envía una solicitud POST con los datos a la API.
4. La API verifica el token y rol del solicitante.
5. Si es válido:
   - Los datos son validados y almacenados en la base de datos.
   - Se devuelve respuesta de éxito.
6. Si hay errores:
   - Se devuelve un error 400 (datos inválidos) o 403 (sin permisos).



- Base de datos. Estructura de tablas a utilizar, soporte lógico y físico.



- Arquitectura: Tecnologías utilizadas en cada parte del proyecto

Backend (API)

- Framework: ASP.NET 8.0
- Lenguaje: C#
- Patrón: Arquitectura RESTful
- Base de datos: PostgreSQL
- Alojamiento: Render (para API y base de datos)
- Seguridad: Autenticación basada en JWT.

Frontend Web (Portal)

- Framework: React
- Herramienta de construcción: Vite
- Lenguaje: JavaScript con JSX
- Estilos: CSS personalizado
- Alojamiento: Vercel
- Accesibilidad: Adaptado para roles Administrador y Doctor

Aplicación móvil

- IDE: Android Studio
- Lenguaje: Kotlin
- Plataforma: Android nativo
- Estado: Funciona localmente, no alojada en producción
- Acceso: Diseñada para uso de pacientes (consultas, citas)

2.3.IMPLEMENTACIÓN

Backend – ASP.NET 8.0 (C#)

Lenguaje: C#
Framework: ASP.NET Core Web API
Estructura: Modelo-Controlador (sin vistas, tipo API REST)

Componentes principales:

- Entidades (User, Appointment, Schedule)
- Representan las tablas base. Se usan como modelos EF Core para la persistencia.
- Roles (enum):
- El tipo UserRole permite distinguir entre Administrador, Doctor y Paciente.
- Autenticación y seguridad:

Implementada mediante JWT, con lógica para:

  - Login
  - Recuperación de contraseña
  - Cambio de email (con tokens)

- Relaciones lógicas:
  - Un User con rol Doctor tiene múltiples Schedules
  - Un Appointment conecta a un Doctor y un Paciente, ambos referencias a User

Scripts importantes:

- Migraciones de EF Core (dotnet ef migrations)

Frontend Web – React + Vite (JSX)

Lenguaje: JavaScript (con JSX)
Framework: React
Herramienta de build: Vite

Componentes clave:

- Login: página que incluye un formulario de inicio de sesión que envía credenciales a la API
- DoctorForm/PatientForm: usado solo por administradores para crear usuarios
- AppointmentsTable: lista de citas con filtros según el rol del usuario
- ProfileSettings: permite cambiar contraseña y correo

Routing: react-router-dom
Estado: Manejando con useState y useContext

Aplicación móvil – Kotlin (Android Studio)

Lenguaje: Kotlin
Plataforma: Android nativo
Funcionalidad:

- Interfaz ligera para pacientes
- Consulta de citas agendadas
- Visualización de perfil
- Comunicación con la API REST (usando Retrofit)

2.4.IMPLANTACIÓN

Backend (API y Base de Datos)

- Tecnología: ASP.NET Core 8.0 (C#)
- Base de datos: PostgreSQL
- Plataforma de alojamiento: Render.com

Procedimiento de despliegue:

1. El código de la API está versionado en GitHub.
2. Se crea un servicio web en Render conectado al repositorio.
3. Se configura la base de datos PostgreSQL también en Render.
4. Variables de entorno (JWT_SECRET, cadenas de conexión, etc.) se añaden en la configuración del servicio.
5. Render ejecuta automáticamente el build y lanza la API.
6. Las migraciones de EF Core se ejecutan al iniciar para asegurar el esquema.

Portal Web (Frontend)

Tecnología: React + Vite + CSS
Plataforma de alojamiento: Vercel

Procedimiento de despliegue:

1. El proyecto React está conectado a un repositorio Git.
2. Vercel detecta el repositorio y realiza el build automáticamente.
3. Se configura el entorno (variables con las URL de la API).
4. El portal queda disponible en un subdominio de Vercel o uno personalizado.

Aplicación Móvil (Paciente)

- Tecnología: Kotlin
- Entorno de ejecución: Android Studio (local, no publicada aún)

Instalación local:

1. Clonar el proyecto desde el repositorio.
2. Abrir en Android Studio.
3. Configurar el archivo build.gradle con la URL de la API correspondiente.
4. Compilar y ejecutar en un emulador o dispositivo físico.
5. Se distribuye manualmente como .apk si es necesario.

2.5. DOCUMENTACIÓN

Se entregará:

- Documentación del proyecto que incluirá (estructura, API endpoints, esquemas de BD).

3. CONCLUSIONES

La realización de este proyecto ha supuesto un importante punto de inflexión en mi formación como desarrollador, ya que me ha permitido aplicar de forma práctica los conocimientos adquiridos durante el ciclo formativo, así como explorar y dominar nuevas tecnologías y metodologías que van más allá del currículo habitual.

Aprendizajes clave:

- Desarrollo backend profesional: La implementación de una API REST en ASP.NET Core 8.0 me permitió profundizar en conceptos como inyección de dependencias, arquitectura limpia, autenticación con JWT, y gestión avanzada de rutas y seguridad.

- Gestión de base de datos real: Configurar y desplegar PostgreSQL en un entorno real (Render) me dio una visión práctica sobre despliegue en la nube, migraciones automatizadas y modelado de datos eficiente.

- Frontend moderno: Usar React con Vite ha supuesto una mejora clara respecto a tecnologías más tradicionales, facilitando la construcción de interfaces reactivas, escalables y bien organizadas, con separación de lógica y presentación.

- Integración multiplataforma: La conexión entre el backend y tres tipos de frontend (portal web, móvil y herramientas de administración) me ayudó a comprender mejor la importancia de los contratos API y el diseño orientado a cliente.

- Despliegue real en la nube: El uso de plataformas como Render y Vercel me proporcionó una experiencia concreta sobre automatización, CI/CD y publicación de aplicaciones accesibles desde cualquier dispositivo.

Habilidades adquiridas que van más allá del ciclo:

- Desarrollo de aplicaciones completas de extremo a extremo (fullstack)

- Uso de herramientas modernas del sector profesional (Vercel, Render, Android Studio, JWT, EF Core, etc.)

- Diseño de arquitectura escalable y modular

- Gestión del ciclo de vida de software real: desarrollo, pruebas, despliegue e implantación

3.1. RESULTADOS Y DISCUSIÓN

A lo largo del desarrollo se han superado diversos retos técnicos, especialmente en la sincronización de agendas y gestión de estados entre frontend y backend. La planificación inicial no se ha cumplido en su totalidad debido a problemas inesperados, entre los que destaca una falla crítica en el disco duro que provocó la pérdida de gran parte del trabajo realizado. Este contratiempo implicó una recuperación parcial de los datos y un reajuste en los tiempos del proyecto.

Como consecuencia de esta situación, no fue posible pulir la aplicación tanto como se deseaba, tanto a nivel visual como en ciertas funcionalidades, limitando el alcance final en términos de usabilidad y acabado.

Además, durante el proceso se presentó la dificultad de aprender nuevas tecnologías, como el uso de React para conectarme a la API y el despliegue de la aplicación en Render. En este aprendizaje, la asistencia de video tutoriales de YouTube y ChatGPT fue fundamental, actuando como un “profesor” que explicaba conceptos y orientaba en la resolución de problemas, sin proporcionar respuestas directas, lo que permitió avanzar de manera autónoma.

3.2. TRABAJO FUTURO (OPCIONAL)

Hasta el momento, se ha completado gran parte de los objetivos planteados inicialmente; sin embargo, quedan aspectos por mejorar y ampliar para continuar con el desarrollo del proyecto. Entre las posibles ampliaciones destacan:

- Implementar un sistema de mensajería para la recuperación de contraseñas y confirmación de usuario, funcionalidad pendiente que mejoraría la experiencia y seguridad del usuario.
- Pulir la interfaz visual para hacerla más atractiva e intuitiva, con especial atención a la usabilidad en dispositivos móviles.
- Incorporar funcionalidades adicionales en el perfil de pacientes, como la posibilidad de añadir una foto de perfil y un selector de imágenes desde la galería, aprovechando recursos vistos en clase para enriquecer la experiencia.
- Explorar el uso de códigos QR para alguna funcionalidad útil dentro de la aplicación, aumentando así su interactividad.
- Profundizar en la seguridad de la aplicación, buscando implementar mejores prácticas y herramientas, dado que actualmente no se cuenta con todos los conocimientos necesarios en este ámbito.
- Mejorar el diseño y formato de los documentos PDF generados, para optimizar su presentación y utilidad.

Estas tareas representan una continuación natural del proyecto, orientada a consolidar y expandir sus capacidades.

4. BIBLIOGRAFÍA

MDN Web Docs. (n.d.). MDN Web Docs. https://developer.mozilla.org/es/

ChatGPT. (n.d.). https://chatgpt.com/?model=

Newest questions. (n.d.). Stack Overflow. https://stackoverflow.com/questions

GitHub.com Documentación de ayuda. (n.d.). GitHub Docs. https://docs.github.com/es

Docs + Quickstarts | Render. (n.d.). https://render.com/docs

Wadepickett. (n.d.). *documentación de ASP.NET*. Microsoft Learn. https://learn.microsoft.com/es-es/aspnet/core/?view=aspnetcore-9.0

Kotlin y Android. (n.d.). Android Developers. https://developer.android.com/kotlin?hl=es-419

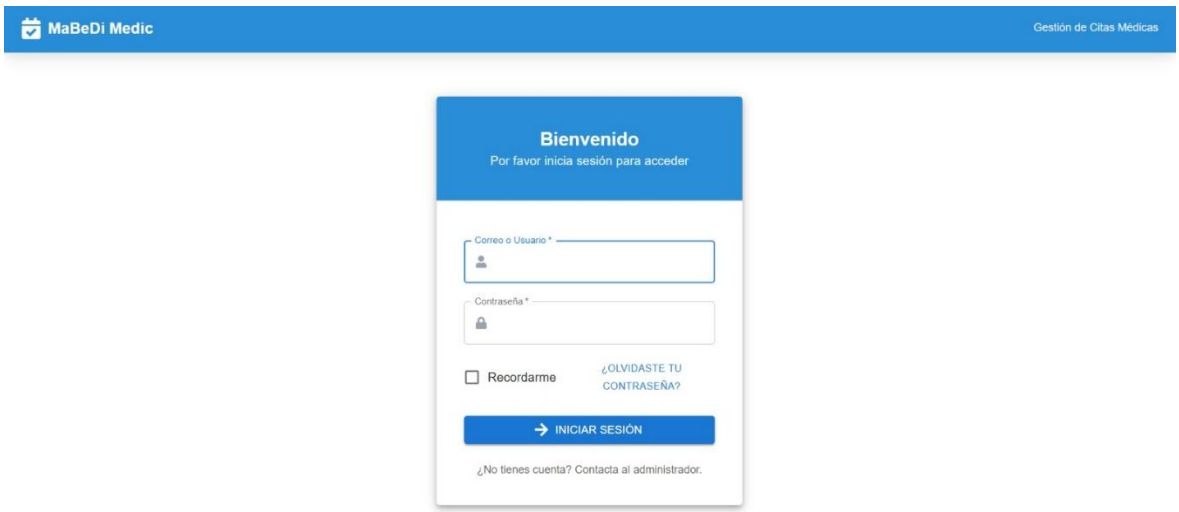
Figma. (n.d.). Figma. https://www.figma.com/files/team/1418851146717904054/recents-and-sharing?fuid=1418851144324329124

PostgreSQL: documentation. (n.d.). The PostgreSQL Global Development Group. https://www.postgresql.org/docs/

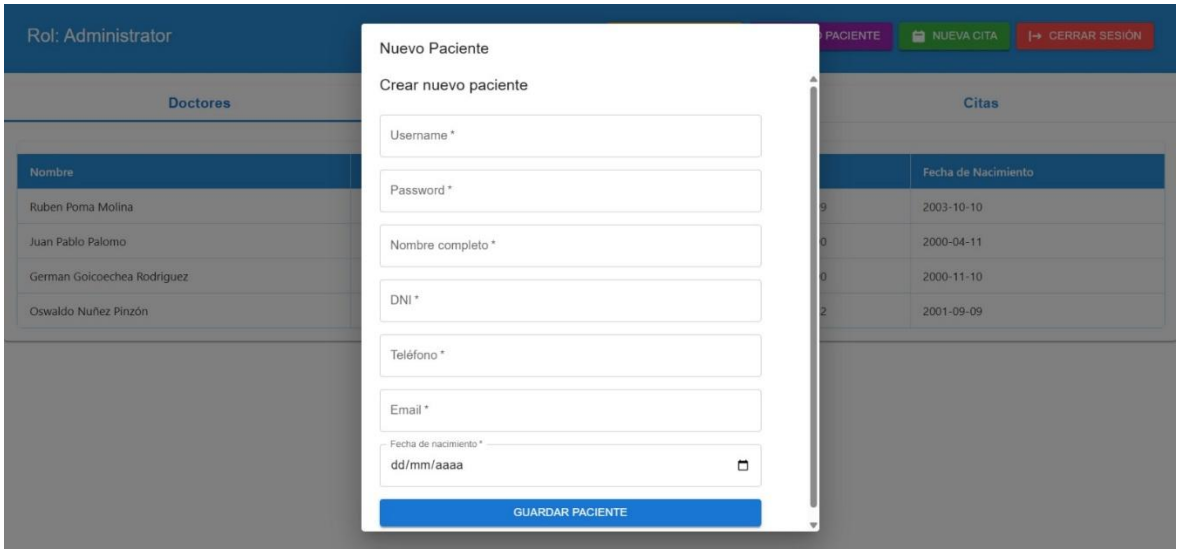
React. (n.d.). https://react.dev/

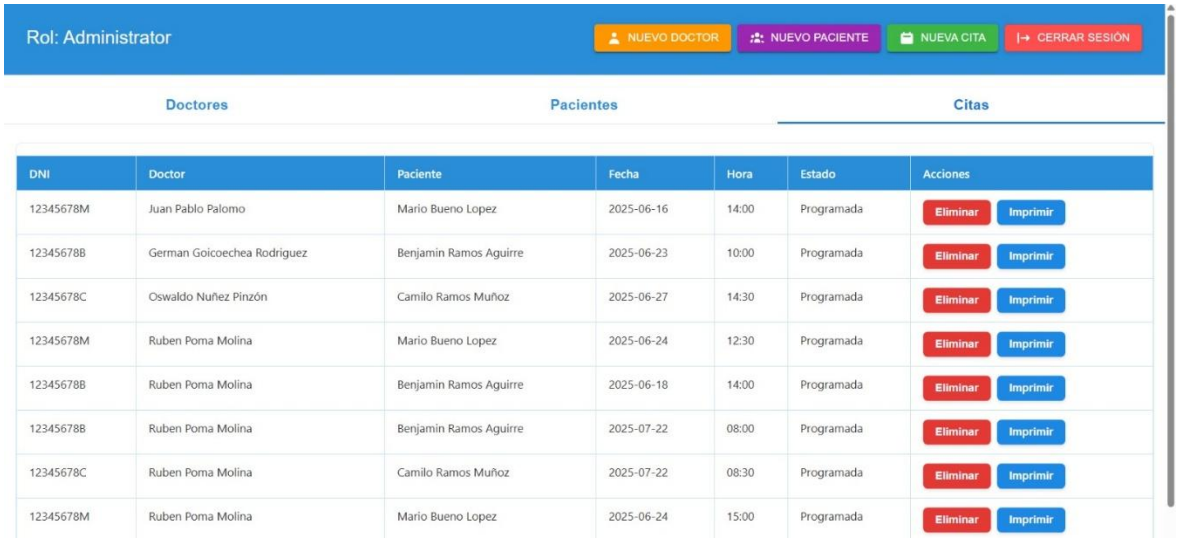
ANEXOS

Detalle del código y otra documentación que consideréis oportuna.
I. Portal Web Corriendo En Vercel

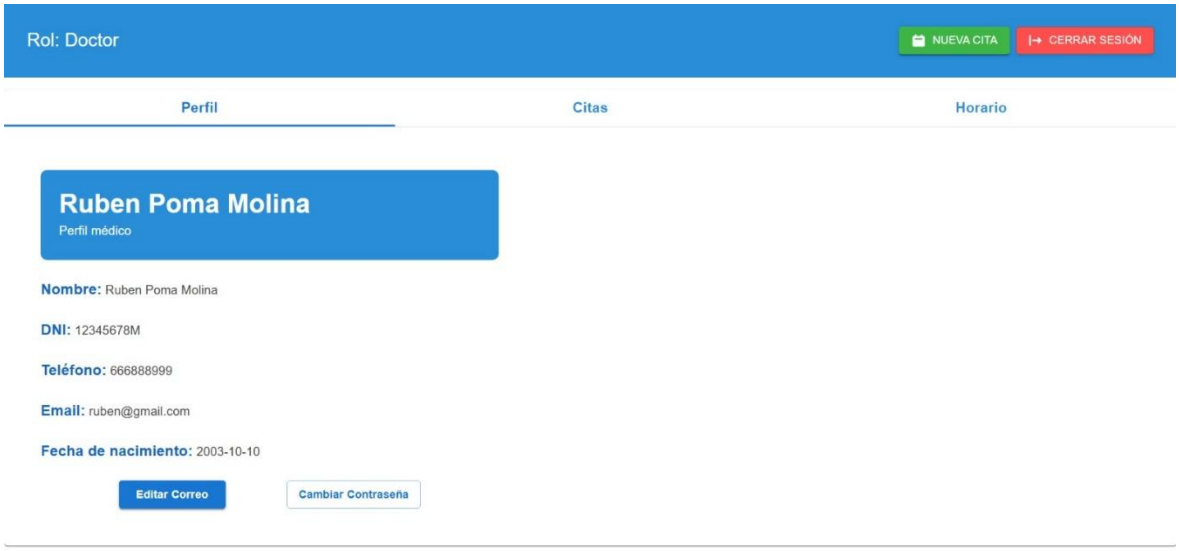


FORMULARIO CREAR PACIENTE:

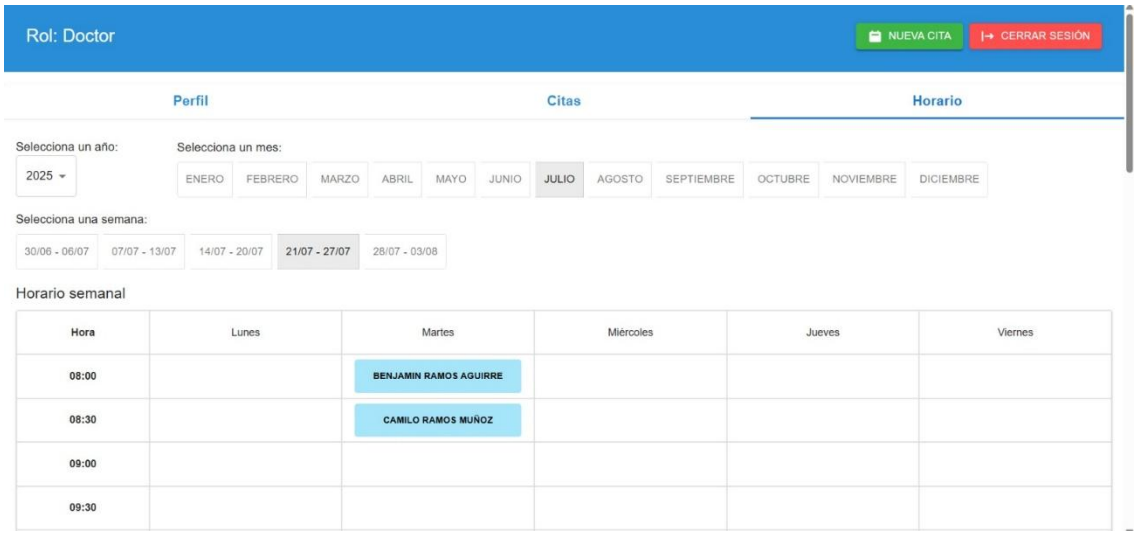




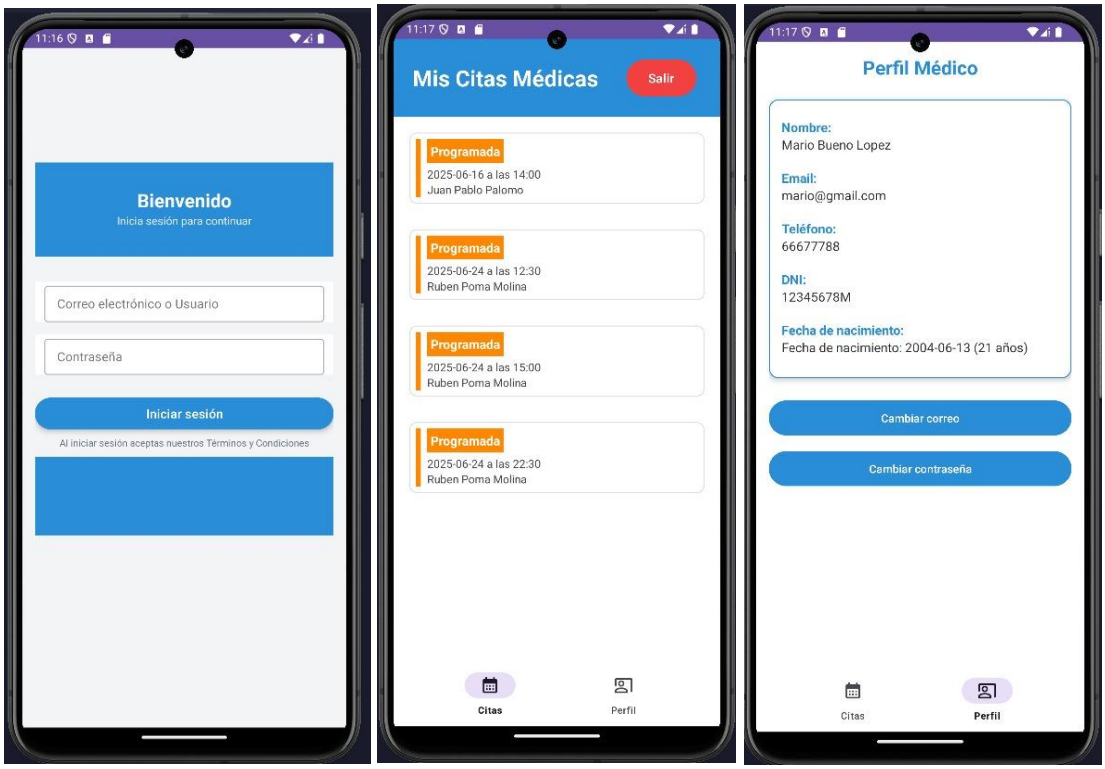
VISUALIZAR PERFIL DOCTOR:



VISUALIZADOR DE CITAS PARA EL DOCTOR:



II. Aplicación Móvil Corriendo En Local



III. Código fuente relevante

Implementado que si le das al botón de volver una vez cerrada la sesión se borre los fragmentos del back stack.

HomeFragment.kt × LoginFragment.kt

```
class HomeFragment : Fragment() {
    }

    private fun cerrarSesion() {
        val sharedPref = requireContext().getSharedPreferences( name: "prefs", AppCompatActivity.MODE_PRIVATE)
        sharedPref.edit().remove( key: "token").apply()
        findNavController().navigate(
            R.id.loginFragment,
             args: null,
            androidx.navigation.NavOptions.Builder()
                .setPopUpTo(R.id.homeFragment,  inclusive: true)
                .build()
        )
    }

```

Método de controlador para registrar un Doctor:

```
[Authorize(Roles = "Administrator")]
[HttpPost("register/doctor")]
0 referencias
public async Task<IActionResult> RegisterDoctor([FromBody] RegisterDoctorRequest request)
{
    if (_context.Users.Any(u => u.Username == request.Username))
        return BadRequest("Username already exists");

    var user = new User
    {
        Username = request.Username,
        PasswordHash = _passwordHasher.HashPassword(null!, request.Password),
        Role = UserRole.Doctor,
        Name = request.Name,
        Dni = request.Dni,
        PhoneNumber = request.PhoneNumber,
        Email = request.Email,
        BirthDate = request.BirthDate,
        DoctorSchedules = request.Schedules?.Select(s => new Schedule
        {
            DayOfWeek = s.DayOfWeek,
            EntryTime = s.EntryTime,
            ExitTime = s.ExitTime
        }).ToList()

    };

    _context.Users.Add(user);
    await _context.SaveChangesAsync();

    return Ok("Doctor registered");
}
```

Login para el Portal Web únicamente para roles de Admin y Doctor

```
[HttpPost("login")]

public IActionResult Login([FromBody] LoginRequest request)
{
    if (request == null || string.IsNullOrEmpty(request.Username) || string.IsNullOrEmpty(request.Password))
    {
        return BadRequest("Username and password are required.");
    }

    var identifier = request.Username;
    var user = _context.Users
        .FirstOrDefault(u => u.Username == identifier || u.Email == identifier);

    if (user == null)
    {
        return Unauthorized("Invalid credentials.");
    }

    var result = _passwordHasher.VerifyHashedPassword(user, user.PasswordHash, request.Password);

    if (result == PasswordVerificationResult.Failed)
    {
        return Unauthorized("Invalid credentials.");
    }

    if (user.Role == UserRole.Patient)
    {
        return Unauthorized("Patients can´t log in.");
    }

    var token = _jwtTokenGenerator.GenerateJwtToken(user);

    Response.Headers.Append("X-Role", user.Role.ToString());

    return Ok(new
    {
        Token = token,
        Role = user.Role.ToString()
    });
}
```

Generador del Token de 8 horas de duración.

```
public class JwtTokenGenerator
{
    private readonly IConfiguration _configuration;


    public JwtTokenGenerator(IConfiguration configuration)
    {
        _configuration = configuration;
    }


    public string GenerateJwtToken(User user)
    {
        var claims = new[]
        {
            new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
            new Claim(ClaimTypes.Name, user.Username),
            new Claim(ClaimTypes.Role, user.Role.ToString())
        };

        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]!));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            issuer: _configuration["Jwt:Issuer"],
            audience: _configuration["Jwt:Audience"],
            claims: claims,
            expires: DateTime.UtcNow.AddHours(8),
            signingCredentials: creds
        );

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

Método para crear una cita con validaciones:

```
[Authorize(Roles = "Administrator, Doctor")]
[HttpPost("appointments/register")]
0 referencias
public async Task<IActionResult> RegisterAppointment(RegisterAppointmentRequest request)
{
    var doctor = await _context.Users.Include(u => u.DoctorSchedules)
                    .FirstOrDefaultAsync(u => u.Id == request.DoctorId && u.Role == UserRole.Doctor);
    if (doctor == null)
        return NotFound("Doctor not found");

    var patient = await _context.Users.FirstOrDefaultAsync(u => u.Id == request.PatientId && u.Role == UserRole.Patient);
    if (patient == null)
        return NotFound("Patient not found");

    var time = request.AppointmentDateTime.TimeOfDay;
    if (time.Minutes != 0 && time.Minutes != 30)
        return BadRequest("Las citas solo pueden programarse a las hh:00 o hh:30.");

    var appointmentDay = request.AppointmentDateTime.DayOfWeek;
    var schedule = doctor.DoctorSchedules?
        .FirstOrDefault(s => s.DayOfWeek == appointmentDay &&
                             s.EntryTime.HasValue && s.ExitTime.HasValue &&
                             time >= s.EntryTime.Value &&
                             time < s.ExitTime.Value);

    if (schedule == null)
        return BadRequest("La cita no está dentro del horario laboral del doctor.");

    var overlapping = await _context.Appointments.AnyAsync(a =>
        a.DoctorId == doctor.Id &&
        a.AppointmentDateTime == request.AppointmentDateTime);

    if (overlapping)
        return Conflict("Ya existe una cita para este doctor en esa franja horaria.");

    var appointment = new Appointment
    {
        DoctorId = doctor.Id,
        PatientId = patient.Id,
        Status = request.Status,
        AppointmentDateTime = request.AppointmentDateTime
    };

    _context.Appointments.Add(appointment);
    await _context.SaveChangesAsync();
    return Ok(appointment);
}
```