Aplicación Web de citas médicas

Estrada Arce Sergio Emilio[1], Mamani Quispe Renzo Geomar[2], Schreiber Landeo Diego Hans[3], Tijero Davila Jeic Lucciano[4],
[1]Universidad Nacional de San Agustín de Arequipa, Perú, sestradaa@unsa.edu.pe, remamaniq@unsa.edu.pe,
dschreiber@unsa.edu.pe, jtijero@unsa.edu.pe

I. Introducción y motivación

En los últimos años, el avance de las tecnologías de la información ha transformado diversos sectores, incluido el ámbito de la salud. La gestión tradicional de citas médicas, comúnmente realizada mediante llamadas telefónicas o de manera presencial, ha demostrado ser ineficiente, propensa a errores humanos y generadora de tiempos de espera innecesarios tanto para pacientes como para el personal administrativo. Esta situación resalta la necesidad de implementar soluciones tecnológicas que optimicen la administración de los recursos clínicos y mejoren la experiencia del usuario.

En este contexto, el desarrollo de plataformas web para la reserva de citas médicas se presenta como una alternativa viable y eficaz. Estas aplicaciones permiten a los pacientes gestionar sus citas de forma autónoma, consultar la disponibilidad de los profesionales de la salud en tiempo real y recibir confirmaciones automáticas, lo que se traduce en una mejora significativa de los procesos internos de las instituciones sanitarias.

El presente trabajo describe el diseño e implementación de una aplicación web para la reserva de citas médicas, desarrollada mediante el framework Django en el backend y Angular en el frontend. Django, basado en Python, proporciona una arquitectura robusta, segura y fácilmente escalable, mientras que Angular permite la creación de interfaces de usuario dinámicas y reactivas, mejorando la experiencia de interacción.

El objetivo principal del proyecto es ofrecer una herramienta funcional que automatice la programación de citas médicas, optimizando los recursos del centro de salud y facilitando el acceso de los pacientes a los servicios. A través del uso de tecnologías modernas y buenas prácticas de desarrollo, se busca una solución escalable, modular y adaptable a las necesidades de diferentes entornos clínicos.

II. Estado del arte relacionado

En los últimos años, la adopción de sistemas digitales en el sector salud ha crecido exponencialmente, motivada por la necesidad de mejorar la gestión de citas, la atención al paciente y la eficiencia operativa de las instituciones médicas. Las aplicaciones web de citas médicas permiten a los usuarios reservar, modificar o cancelar citas en línea, consultar información médica, seleccionar especialistas y recibir notificaciones, lo cual mejora significativamente la experiencia del paciente y reduce la carga administrativa.

Diversos estudios han demostrado que los sistemas de gestión de citas en línea contribuyen a disminuir las tasas de ausentismo y mejorar la puntualidad de atención médica [1]. En este contexto, plataformas como Zocdoc, Doctolib y Salud Digital han logrado posicionarse como referentes, integrando funcionalidades como recordatorios automatizados, selección de especialistas por filtros personalizados y calendarios interactivos.

En términos de tecnología, es común el uso de arquitecturas basadas en microservicios o separación cliente-servidor mediante API REST, con frameworks robustos como Django en el backend y Angular, React o Vue.js en el frontend. Estas tecnologías permiten construir interfaces interactivas, seguras y escalables. Además, la integración con sistemas de autenticación (JWT, OAuth2) y la implementación de consultas asincrónicas mediante AJAX o fetch API son prácticas habituales para lograr una experiencia de usuario fluida [2].

Frente a esta tendencia, el presente proyecto propone una solución web de citas médicas que integra los principios mencionados, con énfasis en la seguridad, la claridad de interfaz y el cumplimiento de las operaciones CRUD para administradores.

III. Metodología

A. Arquitectura del sistema

La arquitectura del sistema se diseñó siguiendo un enfoque cliente-servidor desacoplado, basado en una arquitectura de tres capas: presentación, lógica de negocio y persistencia de datos. Esta separación permite mayor escalabilidad, mantenibilidad y reutilización del código.

Primeramente tenemos a la capa de presentación (Frontend), la cual está implementada en Angular, esta capa se encarga de gestionar la interacción con el usuario. Se utilizaron componentes modulares que consumen servicios HTTP para acceder a los recursos del backend mediante peticiones asíncronas. El uso de Angular permitió la actualización dinámica de la interfaz, validaciones en tiempo real y navegación eficiente entre vistas, siguiendo el patrón SPA (Single Page Application).

Estructura:

En segundo lugar tenemos a la capa de lógica de negocio (Backend), la cual está construida con Django, esta capa contiene toda la lógica relacionada con la autenticación, control de acceso, validación de datos, procesamiento de formularios y generación de respuestas JSON. El uso del framework Django Rest Framework (DRF) facilitó la creación de una API RESTful que permite al frontend acceder a los recursos mediante endpoints seguros y bien estructurados.

Estructura:

Y por último la capa de persistencia (Base de datos), se utilizó una base de datos relacional gestionada por Django (SQLite en desarrollo, y PostgreSQL en producción). Los modelos definidos incluyen relaciones de clave foránea que representan entidades como Paciente, Médico, Especialidad y Cita. Estas relaciones permiten realizar consultas complejas de forma eficiente y consistente.

B. Herramientas y tecnologías utilizadas

Para el desarrollo del sistema web de gestión de citas médicas, se emplearon herramientas y tecnologías modernas que permitieron una implementación eficiente, segura y de alto rendimiento. A continuación, se detallan las principales:

1) Django: Framework de desarrollo web en Python que permitió construir el backend del sistema con una arquitectura limpia y mantenible. Incluye ORM para acceso a datos, sistema de autenticación, enrutamiento y vistas genéricas. Se complementó con Django Rest Framework (DRF) para la creación de una API RESTful consumida por el frontend.
2) Angular: Framework frontend basado en TypeScript que permitió crear una SPA (Single Page Application) dinámica, modular y responsive. Incluyó servicios HTTP para consumir la API del backend, enrutamiento basado en componentes, validación de formularios y arquitectura desacoplada.
3) Bootstrap 5: Biblioteca CSS utilizada para el diseño responsivo y estético de la interfaz gráfica. Facilitó la creación de tablas, formularios, alertas y navegación con un estilo limpio y moderno.
4) SQLite / PostgreSQL: Se utilizó SQLite como base de datos local en desarrollo, y PostgreSQL en producción. Ambas se integran fácilmente con Django y proporcionan fiabilidad, soporte de integridad referencial y buenas capacidades de consulta.
5) JWT (JSON Web Tokens): Para la autenticación segura de usuarios en la API. El backend genera y valida tokens que el frontend almacena y envía en las cabeceras de cada petición autenticada.
6) Render: Plataforma utilizada para el despliegue de la aplicación. Proporcionó infraestructura gratuita para aplicaciones Django con soporte HTTPS, base de datos PostgreSQL y despliegue continuo desde GitHub.
7) Git y GitHub: Herramientas de control de versiones y colaboración empleadas durante todo el desarrollo. Se organizaron ramas por funcionalidades, se documentó el avance y se gestionaron los issues del proyecto.
8) HTML5 y TypeScript: Lenguajes de marcado y programación principales para el desarrollo del frontend, aprovechando la tipificación estricta y modularidad de TypeScript junto a las capacidades semánticas de HTML5.

C. Frontend: Angular

El desarrollo del frontend se llevó a cabo utilizando Angular, un framework moderno basado en TypeScript que permite construir aplicaciones web dinámicas con una arquitectura basada en componentes. Esta elección facilitó la creación de una interfaz de usuario responsiva, modular y mantenible, que consume datos desde una API RESTful proporcionada por el backend en Django.

1) Estructura del Frontend

La aplicación Angular se organizó en las siguientes secciones principales:

Primeramente están los componentes de Navegación y Layout, estos incluyen una barra de navegación superior y una plantilla base que estructura la visualización de los módulos dinámicos del sistema.

En segundo lugar están los módulos de funcionalidad, se desarrollaron componentes para:

- Registro y autenticación de usuarios.
- Gestión de pacientes, médicos y especialidades.
- Agendamiento y visualización de citas médicas.
- Paneles diferenciados según el tipo de usuario (paciente o administrador).

En tercer lugar están los servicios (Services), estos encapsulan la lógica de comunicación con el backend. Cada entidad (como médico, paciente, especialidad y cita) posee un servicio que utiliza HttpClient para enviar y recibir datos en formato JSON.

- Routing: Se configuró un sistema de rutas que permite la navegación entre los diferentes componentes de manera fluida, utilizando app.routes.ts y protegiendo ciertas rutas..
- Consumo de API REST: El frontend realiza llamadas HTTP (GET, POST, PUT, DELETE) a los endpoints del backend para realizar operaciones CRUD de manera asíncrona. Además, el token JWT se incluye automáticamente en las cabeceras para las rutas protegidas.

2) Experiencia del Usuario

La interfaz permite a los pacientes:

- Consultar su perfil personal y ver el historial de citas.
- Agendar una nueva cita seleccionando especialidad, médico, fecha y hora disponibles.
- Cancelar citas pendientes si es necesario.

Y al administrador:

- Gestionar registros de médicos, pacientes, especialidades y horarios.
- Visualizar y cambiar el estado de las citas (pendiente, atendida).

3) Estilo Visual

Para el diseño visual se utilizó Bootstrap 5, que permitió construir una interfaz limpia, accesible y adaptada a múltiples dispositivos (responsive). Se aplicaron estilos personalizados para mejorar la experiencia visual y la usabilidad del sistema.

D. Backend: Django y API REST

El backend del sistema fue desarrollado utilizando el framework Django, complementado con Django REST Framework (DRF) para la creación de una API RESTful robusta y escalable. Esta arquitectura permite desacoplar la lógica de negocio de la interfaz de usuario, facilitando tanto el desarrollo como el mantenimiento del sistema.

1) Organización de la Aplicación:

El backend se estructuró en una aplicación principal de Django, con las siguientes características:

1.1) Modelos de Datos: Se definieron modelos para usuarios, médicos, pacientes, especialidades, citas médicas y horarios. Se utilizaron relaciones ForeignKey para modelar las dependencias entre entidades (por ejemplo, una cita pertenece a un paciente y a un médico).

1.2) Serializadores (Serializers): Los datos fueron expuestos en formato JSON a través de serializadores personalizados, que permiten controlar qué campos se exponen y cómo se validan al crear o actualizar recursos.

1.3) Vistas Basadas en Clases y Funciones: Se utilizaron vistas APIView, ModelViewSet y decoradores como @api_view y @permission_classes para controlar el acceso a los recursos. Además, se emplearon vistas personalizadas para operaciones específicas como listar médicos por especialidad o cambiar el estado de una cita.

1.4) Autenticación y Permisos: Se implementó autenticación mediante JSON Web Tokens (JWT) utilizando SimpleJWT. Los tokens se almacenan en el cliente y se incluyen automáticamente en las cabeceras de las peticiones protegidas. Se definieron permisos personalizados para distinguir entre usuarios autenticados, pacientes y administradores.

1.5) Restricciones de Seguridad: Los formularios y endpoints del backend aplican validaciones adicionales para evitar inconsistencias, duplicaciones o accesos indebidos. Por ejemplo, los pacientes solo pueden ver sus propias citas, y solo los administradores pueden acceder a la gestión de médicos y especialidades.

1.6) Consultas JSON: Además de las operaciones CRUD, el backend ofrece endpoints de consulta, como:

- Obtener médicos por especialidad.
- Obtener las citas de un paciente autenticado.
- Listar especialidades con sus respectivos médicos.

2) Envío de Correos

Como funcionalidad adicional opcional, se incluye:

2.1) Notificaciones por Correo: Se configuró el backend para el envío de correos electrónicos (al momento de registrarse), utilizando el backend SMTP de Django.

Despliegue en Producción

El backend fue desplegado en la plataforma Render.com, utilizando configuraciones para producción como:

- Almacenamiento de archivos estáticos y de medios.

- Configuración de dominio personalizado con HTTPS.

IV. Modelo de Datos

El modelo de datos de la aplicación fue diseñado para reflejar las relaciones y flujos propios de un sistema de agendamiento de citas médicas, considerando los roles de usuarios (pacientes y administradores), los recursos médicos (especialidades, doctores, horarios) y las citas como núcleo de la funcionalidad.

A. Estructura General

El diseño sigue una estructura relacional con integridad referencial, aprovechando el ORM de Django. Los modelos principales y sus relaciones son:

1) Usuario (User): Se utiliza el modelo de usuarios de Django extendido mediante campos personalizados como teléfono y rol. Permite distinguir entre pacientes y administradores del sistema.

2) Paciente: Asociado uno a uno con el modelo User, almacena información adicional del paciente.

3) Médico: Asociado también al modelo User, contiene campos como especialidad (clave foránea), horarios disponibles y estado (activo/inactivo).

4) Especialidad: Define las distintas ramas médicas (pediatría, odontología, etc.) que se ofrecen. Tiene una relación uno-a-muchos con médicos.

5) Cita: Es el modelo central del sistema. Cada cita está relacionada con un paciente y un médico, incluye la fecha, hora, estado (pendiente, atendida, cancelada), y se genera mediante validaciones para evitar duplicados o conflictos.

6) HorarioMedico: Modelo auxiliar para establecer los días y horarios en los que un médico está disponible. Facilita la validación y propuesta automática de citas.

B. Relaciones clave

1) Paciente y Médico se asocian a través de OneToOneField con el modelo de User.
2) Médico tiene una ForeignKey hacia Especialidad.
3) Cita tiene dos claves foráneas: una hacía Paciente y otra hacia Médico.

Estas relaciones permiten consultas eficientes, como obtener todas las citas de un paciente, los médicos de una especialidad, o verificar si un médico tiene disponibilidad para una fecha específica.

C. Consideraciones de Integridad

Se aplicaron restricciones adicionales mediante validaciones del modelo y vistas para evitar:

1) Agendamiento duplicado en la misma fecha y hora.
2) Eliminación de especialidades con médicos activos asociados.
3) Acceso de usuarios a datos que no les corresponden.

V. Funcionalidades del Sistema

La aplicación de agendamiento de citas médicas fue diseñada para ofrecer una experiencia fluida, segura y accesible tanto para pacientes como para administradores. Las funcionalidades se organizaron en módulos, cada uno cumpliendo con los requisitos establecidos y permitiendo operaciones completas sobre los datos médicos.

A. Registro e Inicio de Sesión

Los usuarios pueden registrarse como pacientes mediante un formulario con validaciones. Una vez autenticados, acceden a funcionalidades según su rol:

1) Pacientes: Visualización de perfil, historial de citas, agendamiento.
2) Administradores: Gestión de usuarios, médicos, especialidades y citas.

La autenticación se maneja con tokens JWT, asegurando acceso restringido a las vistas protegidas.

B. Gestión de Pacientes

El panel de administración permite a los administradores:

1) Visualizar todos los pacientes registrados.
2) Editar y eliminar pacientes desde una tabla interactiva.
3) Buscar pacientes por nombre o correo electrónico.

Los formularios utilizan validaciones adicionales (longitud, patrones, campos obligatorios) y están protegidos contra CSRF.

C. Gestión de Médicos

Desde el panel de administración se puede:

1) Ver una tabla con todos los médicos, sus especialidades y estado.
2) Editar la información personal y médica de un doctor.
3) Asignar o cambiar la especialidad.
4) Agregar y gestionar los horarios disponibles del médico.

Además, en el módulo de especialidades se listan los médicos relacionados, permitiendo un control más contextualizado.

D. Gestión de Especialidades

Permite:

1) Crear nuevas especialidades médicas.
2) Editar nombres de especialidades.
3) Eliminar especialidades sin médicos asignados.
4) Visualizar, dentro de cada especialidad, los médicos que la ejercen.

Esto facilita la administración estructurada de los servicios médicos ofrecidos.

E. Gestión de Citas Médicas

Una de las funcionalidades centrales del sistema:

1) Para pacientes:

   1.1) Ver historial de citas (fecha, hora, médico, especialidad, estado).

   1.2) Agendar nuevas citas según disponibilidad médica y especialidad.

   1.3) Visualizar el estado de sus citas: pendiente, atendida o cancelada.

2) Para administradores:

   2.1) Visualizar todas las citas registradas.

   2.2) Editar el estado de la cita (por ejemplo, de pendiente a atendida).

   2.3) Eliminar citas innecesarias.

Estas operaciones se realizan desde una interfaz dinámica que reemplaza la tabla de citas por formularios según la acción seleccionada.

F. AJAX y Framework JS

Se usó Angular como framework de frontend moderno, permitiendo:

1) Consumo de API RESTful mediante HttpClient.
2) Actualizaciones asíncronas del contenido (sin recargar la página).
3) Uso de componentes, servicios y rutas modulares para separar funcionalidades.
4) Formularios reactivos y controlados, con estados dinámicos y validaciones.

Además, se utilizó Bootstrap para el diseño responsive y elegante.

VI. Resultados y despliegue Web

En está parte se detalla los resultados alcanzados en el desarrollo de la aplicación de citas médicas, así como el proceso de despliegue final en la web con un dominio accesible públicamente.

A. Cumplimiento de requisitos

Durante el desarrollo de la aplicación de citas médicas, se lograron satisfacer los requerimientos técnicos y funcionales definidos en el proyecto. Se implementó una aplicación independiente, con su propio conjunto de URLs utilizando la función reverse de Django, lo que facilita la mantenibilidad y navegación entre vistas.

Las plantillas fueron diseñadas específicamente para esta aplicación, aplicando principios de diseño limpio y el uso de widgets personalizados para mejorar la experiencia del usuario en los formularios. Se implementaron todas las funcionalidades esenciales de un sistema CRUD, incluyendo vistas para listar, crear, consultar, actualizar y eliminar registros de pacientes, médicos, citas y especialidades.

En cuanto a seguridad, se incorporaron restricciones adicionales en los formularios, como la protección contra ataques CSRF, validación de entradas del usuario y control de acceso basado en roles. Además, se desarrolló una vista que entrega datos en formato JSON, permitiendo su consumo asíncrono desde el frontend.

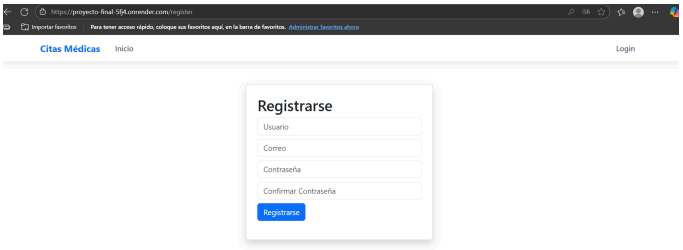
El cliente web fue desarrollado con Angular como framework de JavaScript, integrando peticiones AJAX para operaciones como carga, edición y eliminación de datos sin recargar la página. Se definieron al menos dos modelos principales en la base de datos, como Cita y Paciente, estableciendo relaciones mediante claves foráneas que permiten organizar adecuadamente la información sobre especialidades y médicos.

Para el diseño visual se emplearon CSS y Bootstrap, logrando una interfaz moderna y adaptable. Finalmente, la aplicación fue desplegada exitosamente en la plataforma Render, incluyendo un dominio con HTTPS. También se implementó la exportación de citas médicas a formato PDF, cumpliendo con requisitos adicionales que enriquecen la experiencia del usuario.

A. Capturas de pantalla

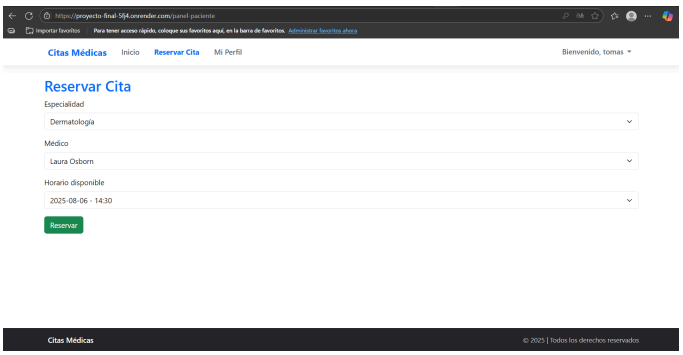
Pantalla de Inicio de Sesión:

Pantalla de Registrarse:

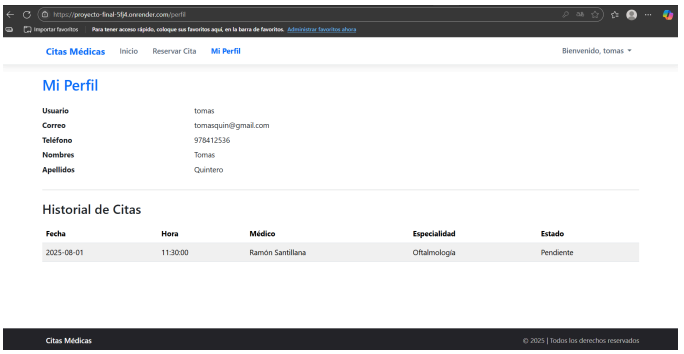


Panel de inicio:

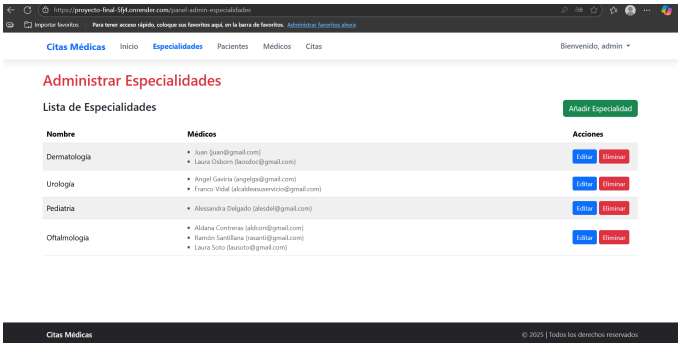
Panel de reservar cita:



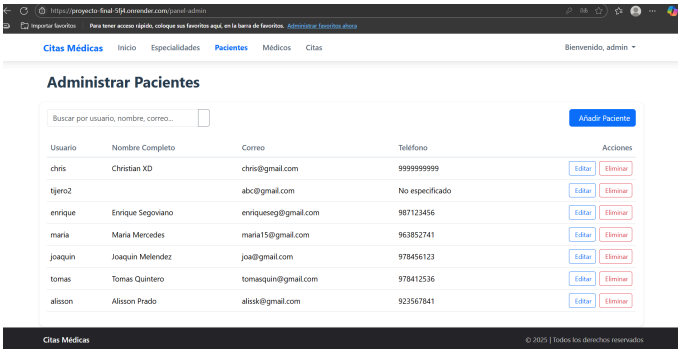
Perfil del Paciente:



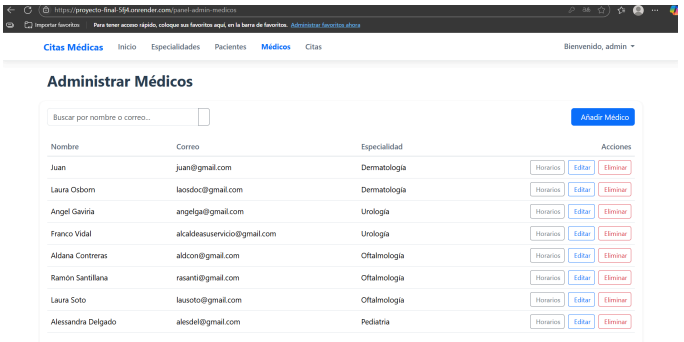
Panel de Administración (Especialidades):



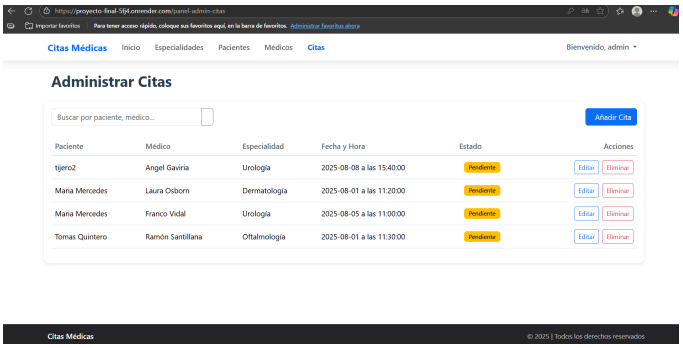
Panel de Administración (Pacientes):



Panel de Administración (Médicos):



Panel de Administración (Citas):



B. Despliegue en Render

El sistema fue desplegado exitosamente en la plataforma Render, utilizando:

1) Backend Django.
2) Frontend Angular servido como archivos estáticos dentro de Django.
3) Base de datos SQLite para pruebas, con posibilidad de migrar a PostgreSQL para producción.
4) Certificado HTTPS activo y dominio personalizado habilitado.

URL pública:
https://proyecto-final-5fj4.onrender.com/

Credenciales de prueba:

Para aclarar, con las credenciales que se están proporcionando se accede a la vista del administrador, para ver la vista de un usuario normal, debe registrarse.

1) Usuario: admin
2) Contraseña: 1234

VII. Conclusiones y recomendaciones

A. Conclusiones

El desarrollo de la aplicación web de gestión de citas médicas permitió poner en práctica diversas competencias adquiridas a lo largo del curso, tales como el diseño de modelos de datos relacionales, la construcción de interfaces dinámicas y el consumo eficiente de APIs RESTful. A través de la integración entre Django y Angular, se logró construir una solución robusta, escalable y orientada a mejorar la experiencia de los usuarios en la gestión de sus citas médicas.

El cumplimiento de los requisitos obligatorios y opcionales demuestra una sólida implementación del proyecto, destacando funcionalidades avanzadas como el consumo de datos en formato JSON, el uso de AJAX para operaciones asincrónicas, y la generación de documentos PDF a partir de datos del sistema. Además, el despliegue exitoso en la nube y la adopción de buenas prácticas de seguridad refuerzan la viabilidad de esta solución en un entorno real.

Asimismo, la organización modular del sistema y la separación clara entre frontend y backend permiten una fácil mantenibilidad y futuras mejoras sin comprometer la estructura base del proyecto.

B. Recomendaciones

1) Ampliar la gestión de usuarios: Se sugiere incorporar más funcionalidades en la administración de roles, como la creación de perfiles personalizados (por ejemplo, recepcionistas o asistentes médicos).
2) Notificaciones por correo: Implementar un sistema de notificación automática vía email ante la creación, modificación o cancelación de citas puede mejorar la comunicación entre pacientes y médicos.
3) Internacionalización: Adaptar la aplicación a múltiples idiomas para aumentar su accesibilidad a una audiencia más amplia.
4) Pruebas automatizadas: Incluir pruebas unitarias y de integración para asegurar la estabilidad del sistema ante futuras actualizaciones.
5) Mejorar la accesibilidad: Implementar buenas prácticas de accesibilidad (WCAG) para garantizar el uso del sistema por parte de usuarios con discapacidades.

VIII. Evidencia de requerimientos cumplidos

En esta sección se muestran capturas de pantalla que evidencian el cumplimiento de los requisitos funcionales y técnicos solicitados para el desarrollo de la aplicación web de citas médicas. Las imágenes presentadas corresponden a funcionalidades implementadas tanto en el frontend como en el backend.

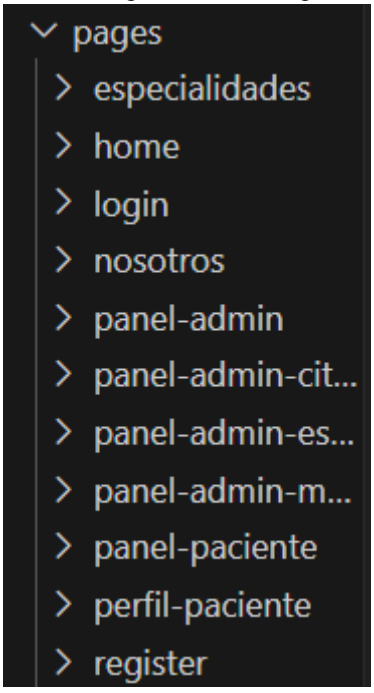
1) Aplicación Independiente con URLs Propias y Uso de reverse:

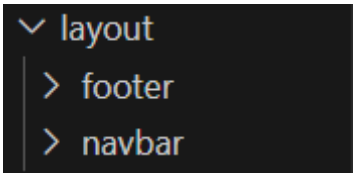
```python
urlpatterns = [
    path('especialidades/', listar_especialidades),          # GET
    path('especialidades/crear/', crear_especialidad),       # POST
    path('especialidades/<int:pk>/editar/', editar_especialidad),  # PUT/PATCH
    path('especialidades/<int:pk>/eliminar/', eliminar_especialidad),  # DELETE
    path('medicos/', medicos_por_especialidad),
    path('horarios/', horarios_por_medico),
    path('register/', register_user, name='api_register'),
    path('login/', TokenObtainPairView.as_view(), name='api_login'),
    path('citas/reservar/', reservar_cita),
    path('paciente/perfil/', perfil_paciente, name='api_perfil_paciente'),
    path('paciente/citas/', citas_paciente, name='api_citas_paciente'),
    path('admin/pacientes/', admin_pacientes_list_create, name='api_admin_pacientes'),
    path('admin/pacientes/<int:pk>/', admin_paciente_detail, name='api_admin_paciente_detalle'),
    path('rol/', obtener_rol, name='api_rol'),
    path('admin/medicos/', admin_medicos, name='admin_medicos'),
    path('admin/medicos/<int:pk>/', admin_medico_detalle, name='admin_medico_detalle'),
    # Admin horarios por médico
    path('admin/medicos/<int:medico_id>/horarios/', admin_medico_horarios, name='admin_medico_horarios'),
    # Admin horario individual
    path('admin/horarios/<int:pk>/', admin_horario_detalle, name='admin_horario_detalle'),
    path('especialidades-con-medicos/', especialidades_con_medicos, name='especialidades_con_medicos'),
    # --- ADMIN CITAS ---
    path('admin/citas/', admin_citas_list_create, name='admin_citas'),
    path('admin/citas/<int:pk>/', admin_cita_detail, name='admin_cita_detalle'),
]
```

views.py usando reverse:

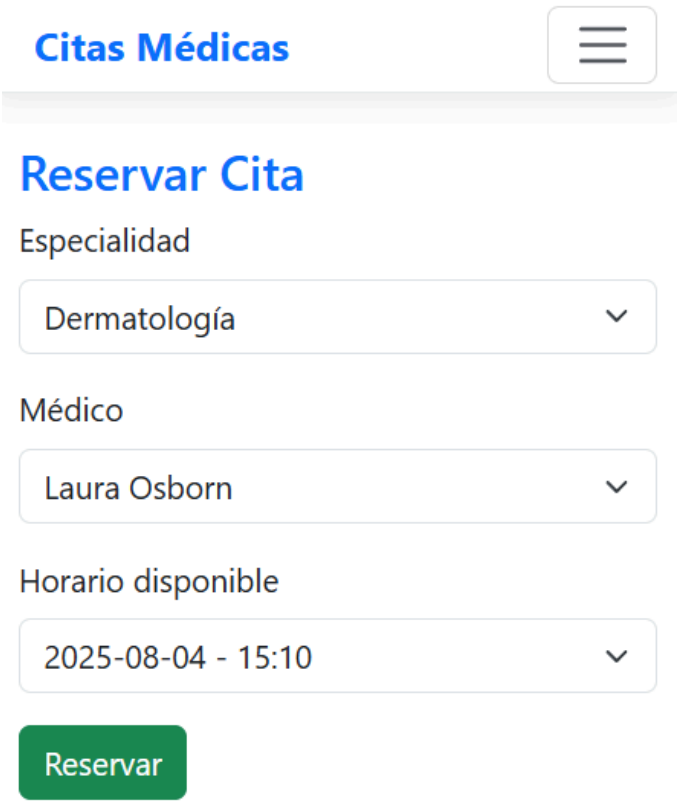
```python
    #  Generar URL completa a la vista de login
    login_url = request.build_absolute_uri(reverse('api_login'))
```

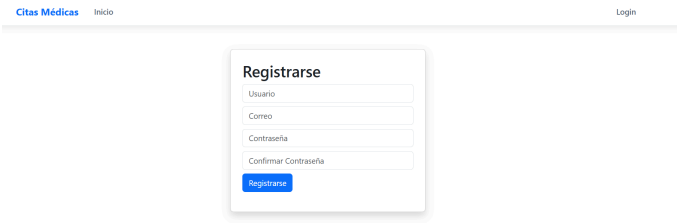
2) Plantillas propias de la aplicación:
Son son las plantillas de la aplicación:



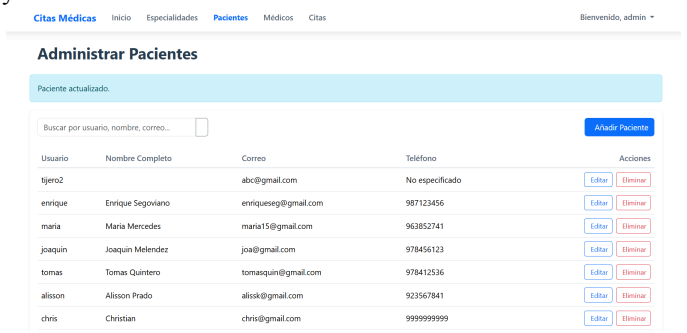


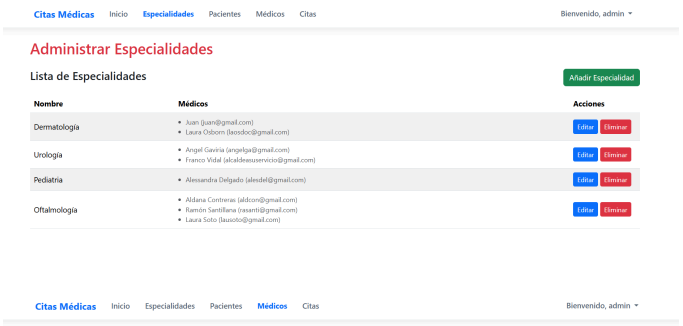
3) Que usen widgets de manera elegante:
Se muestran a continuación algunos de los formularios estilizados.

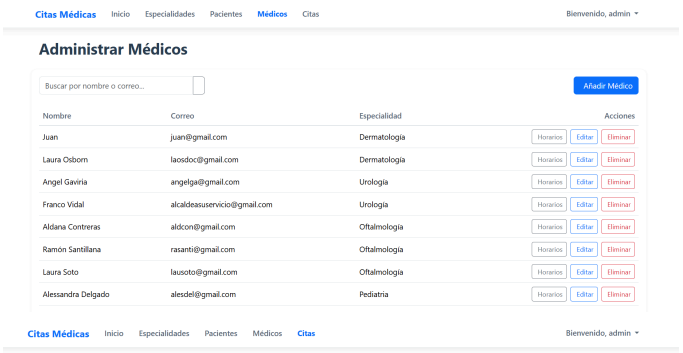




4) Vistas de Listado, Detalle, Crear, Actualizar y Borrar:
Se muestra a continuación la vista del administrador que puede listar, añadir, editar y borrar pacientes, especialidades, médicos y citas.







5) Formulario con restricciones de seguridad adicionales:
A continuación se muestran algunos ejemplos.
Formulario de register.

```
    if not username or not password or not email:
        return Response({'error': 'Todos los campos son obligatorios'}, status=status.HTTP_400_BAD_REQUEST)

    if User.objects.filter(username=username).exists():
        return Response({'error': 'El usuario ya existe'}, status=status.HTTP_400_BAD_REQUEST)
```

Reservar cita.

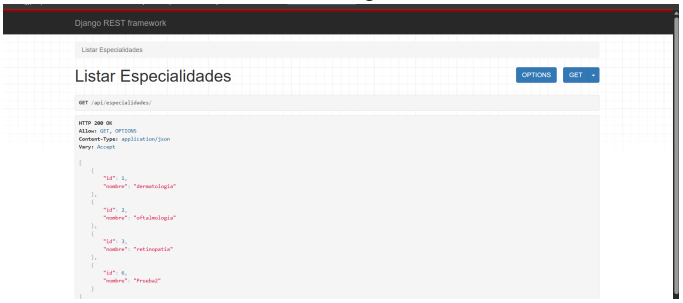
```
    except Paciente.DoesNotExist:
        return Response({'error': 'Solo los pacientes pueden reservar citas.'},
                        status=status.HTTP_403_FORBIDDEN)
```

```
    if not (medico_id and fecha and hora):
        return Response({'error': 'Campos incompletos.'},
                        status=status.HTTP_400_BAD_REQUEST)
```

```
    except Medico.DoesNotExist:
        return Response({'error': 'Médico no encontrado.'},
                        status=status.HTTP_404_NOT_FOUND)
```

```
    if not horario:
        return Response({'error': 'Horario no disponible.'},
                        status=status.HTTP_400_BAD_REQUEST)
```

6) Vista de consultas que devuelva Json:
A continuación se muestra una respuesta en JSON.



7) Programa cliente para hacer y consumir las consultas:

a) Con AJAX (Algunos ejemplos que actualizan el contenido sin recargar la página):
En register.

```
    return this.http.post(`${this.apiUrl}/register/`, data, {
```

En el panel de administrar paciente:

```
  listar(q: string = '') {
    const url = q ? `${this.baseUrl}?q=${encodeURIComponent(q)}` : this.baseUrl;
    return this.http.get<PacienteAdmin[]>(url, { headers: this.authHeaders() });
  }

  crear(data: { username: string; email?: string; first_name?: string; last_name?: string; telefono?: string; password?: string; }) {
    return this.http.post<PacienteAdmin>(this.baseUrl, data, { headers: this.authHeaders() });
  }

  obtener(id: number) {
    return this.http.get<PacienteAdmin>(`${this.baseUrl}${id}/`, { headers: this.authHeaders() });
  }

  actualizar(id: number, data: Partial<PacienteAdmin>) {
    return this.http.patch<PacienteAdmin>(`${this.baseUrl}${id}/`, data, { headers: this.authHeaders() });
  }

  eliminar(id: number) {
    return this.http.delete(`${this.baseUrl}${id}/`, { headers: this.authHeaders() });
  }
}
```

b) Con Framework de JavaScript:
A continuación se muestran las rutas.

```
export const routes: Routes = [
  { path: '', component: Home },
  { path: 'nosotros', component: Nosotros },
  { path: 'login', component: Login },
  { path: 'register', component: Register },
  { path: 'especialidades', component: Especialidades },
  // agregar rutas para paneles
  { path: 'panel-paciente', loadComponent: () => import('./pages/panel-paciente/panel-paciente').then(m => m.PanelPaciente) },
  { path: 'perfil', loadComponent: () => import('./pages/perfil-paciente/perfil-paciente').then(m => m.PerfilPaciente) },
  { path: 'panel-admin', loadComponent: () => import('./pages/panel-admin/panel-admin').then(m => m.PanelAdmin) },
  { path: 'panel-admin-medicos', loadComponent: () => import('./pages/panel-admin-medicos/panel-admin-medicos').then(m => m.PanelAdminMedic
  {
    path: 'panel-admin-especialidades',
    loadComponent: () => import('./pages/panel-admin-especialidades/panel-admin-especialidades')
      .then(m => m.PanelAdminEspecialidades)
  },
  { path: 'panel-admin-citas',
    loadComponent: () => import('./pages/panel-admin-citas/panel-admin-citas')
      .then(m => m.PanelAdminCitas)
  },
];
```

8) Al menos dos modelos:
A continuación se muestran todos los modelos implementados.

```
class Especialidad(models.Model):
    nombre = models.CharField(max_length=100)

    def __str__(self):
        return self.nombre

class Medico(models.Model):
    nombres = models.CharField(max_length=100)
    correo = models.EmailField()
    especialidad = models.ForeignKey(Especialidad, on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.nombres} - {self.especialidad.nombre}"

class Paciente(models.Model):
    usuario = models.OneToOneField(User, on_delete=models.CASCADE)
    telefono = models.CharField(max_length=15, blank=True, null=True)
    def __str__(self):
        return self.usuario.username

class Cita(models.Model):
    paciente = models.ForeignKey(Paciente, on_delete=models.CASCADE)
    medico = models.ForeignKey(Medico, on_delete=models.CASCADE)
    fecha = models.DateField()
    hora = models.TimeField()
    estado = models.CharField(max_length=20, default='Pendiente')
    creada_en = models.DateTimeField(auto_now_add=True)  # Solo con auto_now_add=True

    def __str__(self):
        return f"Cita {self.paciente} / {self.medico} / {self.fecha} {self.hora}"
```

```
class Horario(models.Model):
    medico = models.ForeignKey('Medico', on_delete=models.CASCADE)
    fecha = models.DateField()
    hora = models.TimeField()
    disponible = models.BooleanField(default=True)

    def __str__(self):
        return f"{self.medico.nombres} - {self.fecha} {self.hora}"

class Administrador(models.Model):
    usuario = models.OneToOneField(User, on_delete=models.CASCADE)
    telefono = models.CharField(max_length=15, blank=True, null=True)
    nombres = models.CharField(max_length=100, blank=True)
    apellidos = models.CharField(max_length=100, blank=True)

    def __str__(self):
        return f"{self.nombres} {self.apellidos}"
```

9) Modelo con clave externa: foreign key:
A continuación se muestra el Fragmento del modelo Cita con ForeignKey a Paciente y Medico.

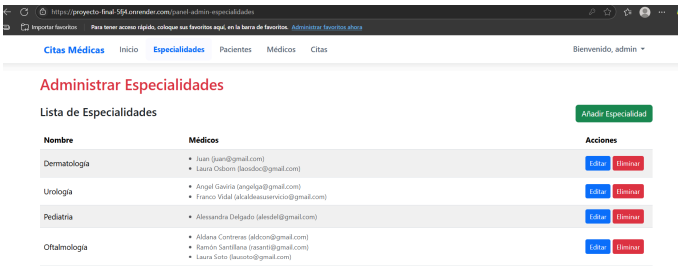
```python
class Cita(models.Model):
    paciente = models.ForeignKey(Paciente, on_delete=models.CASCADE)
    medico = models.ForeignKey(Medico, on_delete=models.CASCADE)
```

10) CSS o Bootstrap:
A continuación se muestra la parte donde se está colocando Bootstrap y como la página tiene un diseño responsive.
En index.html:

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
```

Diseño responsive:

11) Publicó su aplicación en el web:
https://proyecto-final-5fj4.onrender.com/



12) Enviar correo:
Configuración para enviar un correo de prueba.

```python
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'sandbox.smtp.mailtrap.io'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = '22eb3b11fb949c'
EMAIL_HOST_PASSWORD = '032eab2fce1893'
DEFAULT_FROM_EMAIL = 'notificaciones@citasmedicas.com'
```

Link del repositorio:
https://github.com/SrEstrada/Proyecto_Final_Angular.git

References

[1] C. S. Kruse, K. Kothman, K. Anerobi, and L. Abanaka, "Adoption factors of the electronic health record: A systematic review," *JMIR Medical Informatics*, vol. 4, no. 2, p. e19, 2016.

[2] D. Ghimire, "Estudio comparativo de frameworks web Python: Flask y Django," Revista de Ingeniería de Software y Sistemas, vol. 12, no. 3, pp. 45-60, 2020.