



Módulo de Gestión de Pacientes

Práctica Básica: Interfaz y Comportamiento

Autor: Martinez Ruiz Edgar Jair

Índice

1. Introducción	3
2. Objetivos	3
2.1. Objetivo General	3
2.2. Objetivos particulares	3
3. Glosario	3
4. Conceptos básicos	4
5. Guía de desarrollo	4
5.1. Crear proyecto Django	4
5.2. Modificar y agregar archivos	6
5.2.1. Modificación del archivo <code>settings.py</code>	6
5.2.2. Modificación del archivo <code>urls.py</code>	6
5.2.3. Crear carpetas <code>templates</code> y <code>static</code>	7
5.3. Crear un super usuario	8
5.4. Crear tablas en la base de datos	9
5.5. Backend	10
5.6. Frontend	12
5.7. Conexión entre Backend y Frontend	16
6. Pruebas y validación	16

1. Introducción

Esta práctica consiste en registrar un usuario en la plataforma Django y asignarlo al grupo "Doctor", lo que le otorga acceso a un conjunto de vistas específicas. Una vez autenticado, el usuario con rol de doctor puede interactuar con un módulo dinámico que le permite realizar las siguientes acciones:

- Registrar nuevos pacientes.
- Consultar la información detallada de cualquier paciente.
- Actualizar ciertos datos personales de los pacientes.
- Registrar nuevas consultas médicas asociadas a cada paciente.
- Realizar la eliminación lógica de un paciente (cambio de estado).

Las vistas han sido diseñadas para ofrecer una interacción clara, fluida y centrada en la experiencia del usuario con rol doctor, aprovechando componentes dinámicos en el frontend y validaciones en el backend mediante Django.

2. Objetivos

Esta sección describe los objetivos generales y particulares que se desean alcanzar en esta práctica básica.

2.1. Objetivo General

Garantizar que únicamente los usuarios con el rol "Doctor" puedan acceder e interactuar con las vistas del sistema, asegurando que cada una funcione correctamente y cumpla con su propósito específico dentro del flujo de trabajo médico.

2.2. Objetivos particulares

1. Registrar usuarios con distintos roles y verificar que solo aquellos con el rol "Doctor" tengan acceso autorizado a las vistas correspondientes.
2. Establecer relaciones entre las tablas de la base de datos para garantizar una estructura coherente, normalizada y que facilite la integridad de los datos.
3. Diseñar e implementar vistas interactivas, intuitivas y comprensibles, que faciliten al usuario la navegación y ejecución de acciones en el sistema.
4. Gestionar correctamente las vistas protegidas para que cumplan con su propósito funcional y mantengan la coherencia del flujo de trabajo.
5. Validar los formularios del sistema tanto en frontend como en backend para prevenir errores de ingreso y asegurar la calidad de los datos.
6. Incorporar retroalimentación visual (como notificaciones y mensajes de validación) para mejorar la experiencia del usuario y brindar claridad en cada acción realizada.
7. Implementar funciones de edición y eliminación lógica de pacientes que mantengan la integridad del historial médico.
8. Verificar que cada componente del sistema se comporte de forma dinámica y responda correctamente a las acciones del usuario sin errores de ejecución.

3. Glosario

Django	Framework web de alto nivel en Python que permite desarrollar aplicaciones rápidas y seguras mediante el patrón MTV (Model-Template-View).
HTML	Lenguaje de marcado utilizado para estructurar el contenido de las páginas web.
Bootstrap	Biblioteca de CSS y JavaScript que facilita el diseño de interfaces web responsivas.
CSRF Token	Token de seguridad utilizado para prevenir ataques del tipo Cross-Site Request Forgery en formularios web.
Static Files	Archivos estáticos (CSS, JS, imágenes) utilizados por Django y almacenados generalmente en la carpeta <code>static/</code> .
Template	Archivo HTML que contiene bloques dinámicos y lógica embebida usando la sintaxis de Django (<code>{% ... %}</code> y <code>{% ... %}</code>).
Formulario HTML	Componente de una página web que permite al usuario introducir datos que serán enviados al servidor.
Validación de formularios	Proceso que asegura que los datos introducidos cumplan con los requisitos antes de ser enviados al servidor.
View (Vista)	Función en Django que procesa una petición y retorna una respuesta HTTP, comúnmente una plantilla renderizada.
URLconf	Archivo <code>urls.py</code> que define las rutas disponibles en una aplicación y las vincula con las vistas correspondientes.
Grupo (Group)	Entidad en Django que permite organizar usuarios con los mismos permisos.
Permisos (Permissions)	Reglas que definen qué acciones puede realizar un usuario en el sistema.
Consulta médica	Registro creado por un doctor que documenta síntomas, medicamentos y observaciones sobre un paciente.
Eliminación lógica	Proceso mediante el cual se desactiva un registro sin eliminarlo de la base de datos.
Fetch API	Interfaz JavaScript que permite realizar peticiones HTTP de forma asíncrona desde el navegador.

4. Conceptos básicos

MTV (Modelo-Template-Vista)	Arquitectura que Django sigue, similar al patrón MVC, donde Model gestiona los datos, Template define la presentación, y View contiene la lógica.
Modelo (Model)	Clase de Python que define una tabla en la base de datos. Se escribe en el archivo <code>models.py</code> .
Ruta (URL path)	Dirección definida en <code>urls.py</code> que enlaza una URL con una vista.
Template Extends	Directiva como <code>% extends "base.html" %</code> usada para reutilizar un diseño común en varias páginas.
JavaScript DOM Manipulation	Uso de JavaScript para modificar el contenido y estilo de elementos HTML dinámicamente.
Eventos JavaScript	Acciones como <code>onclick</code> , <code>onsubmit</code> o <code>input</code> que ejecutan funciones al interactuar con el usuario.
CSRF en Django	Protección contra ataques de tipo CSRF. Django incluye tokens en los formularios mediante <code>% csrf_token %</code> .
Validación de formularios	Verificación que asegura que los datos ingresados cumplen las restricciones, tanto del lado cliente como del servidor.
Mensajes Flash	Sistema de mensajes temporales de Django que informa al usuario sobre el resultado de una acción.

5. Guía de desarrollo

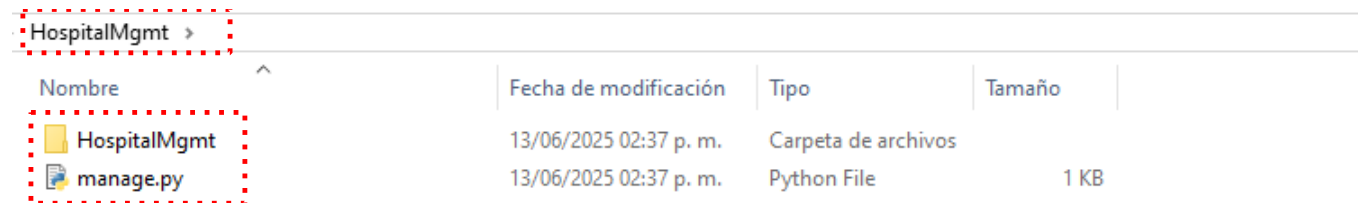
5.1. Crear proyecto Django

Para crear un nuevo proyecto en Django, se debe ejecutar el siguiente comando desde la terminal:

```
python -m django startproject <nombre_del_proyecto>
```

Este comando genera una carpeta con el nombre especificado, que contiene la estructura base de un proyecto Django.

En la **Figura 1** se muestra el contenido de la carpeta generada. Dentro de ella se encuentra el archivo **manage.py**, el cual actúa como el punto de entrada principal del proyecto. A través de este archivo se pueden ejecutar comandos administrativos, como iniciar el servidor, aplicar migraciones o crear aplicaciones internas.



Nombre	Fecha de modificación	Tipo	Tamaño
HospitalMgmt	13/06/2025 02:37 p. m.	Carpeta de archivos	
manage.py	13/06/2025 02:37 p. m.	Python File	1 KB

Figura 1: Proyecto creado para trabajar con Django.

Una vez creada la carpeta del proyecto, se abre en Visual Studio Code, el cual servirá como entorno de desarrollo para escribir el código, organizar archivos y gestionar carpetas del proyecto.

El siguiente paso consiste en abrir la terminal integrada de Visual Studio Code y, ubicándose en la raíz del proyecto (donde se encuentra el archivo **manage.py**), ejecutar el siguiente comando:

```
python manage.py startapp <nombre_de_la_app>
```

Este comando crea una nueva carpeta con la estructura necesaria para una aplicación de Django, incluyendo archivos como **views.py**, **models.py**, **admin.py**, entre otros. Estos archivos permitirán implementar la lógica del módulo correspondiente dentro del proyecto.

En la **Figura 2** se muestra la forma correcta de utilizar el comando anterior. Es importante asegurarse de ejecutarlo desde la raíz del proyecto y verificar que la carpeta de la aplicación se haya creado correctamente con todos los archivos necesarios.

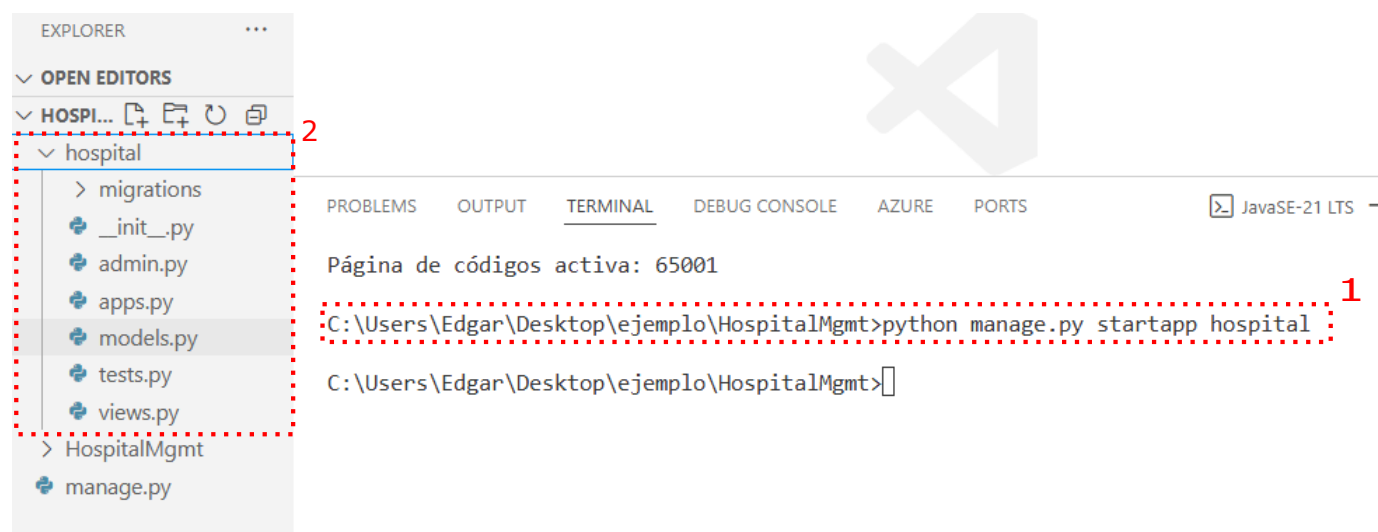


Figura 2: Carpeta creado para la aplicación.

5.2. Modificar y agregar archivos

En este paso se muestra algunas preconfiguraciones que se deben realizar para no tener problemas en el momento de correr el archivo **manage.py**.

5.2.1. Modificación del archivo `settings.py`

El archivo `settings.py` se encuentra dentro de la carpeta principal del proyecto, en este caso **HospitalMgmt**. Este archivo contiene la configuración global del proyecto Django, incluyendo aspectos como la conexión a la base de datos, las aplicaciones instaladas, el idioma, la zona horaria, rutas estáticas, entre otros.

Uno de los primeros pasos necesarios después de crear una nueva aplicación es registrarla en la configuración del proyecto. Para ello, se debe agregar el nombre de la aplicación en la lista `INSTALLED_APPS`, la cual define todas las aplicaciones activas que Django debe reconocer y cargar al iniciar el servidor.

En la **Figura 3** se muestra un ejemplo de cómo debe incluirse la aplicación dentro de esta lista. Este paso es fundamental para que Django pueda gestionar correctamente los modelos, vistas y configuraciones de dicha aplicación.

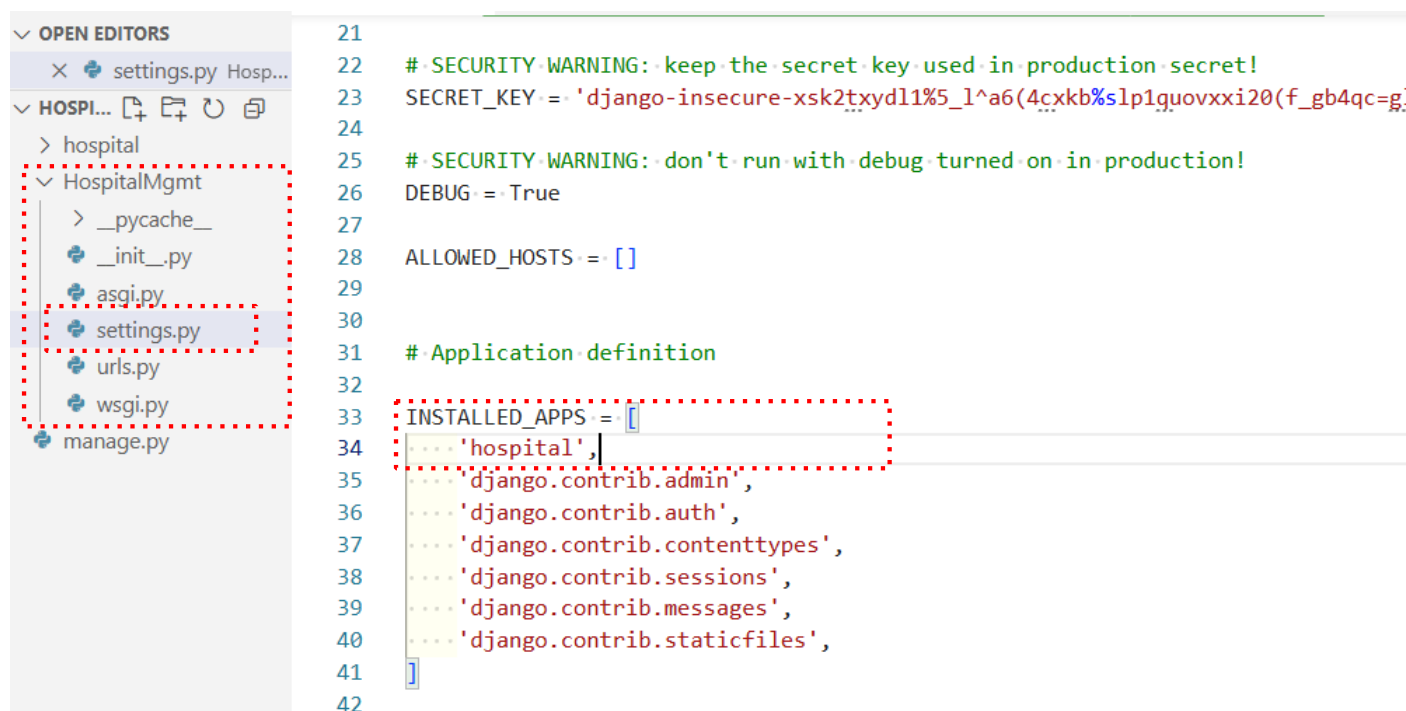


Figura 3: Agregar aplicación a `INSTALLED_APPS`.

5.2.2. Modificación del archivo `urls.py`

El siguiente paso consiste en organizar adecuadamente las rutas del proyecto. Para ello, se debe crear un archivo `urls.py` dentro de la carpeta de la aplicación —en este caso, **hospital**—, y posteriormente asociarlo al archivo `urls.py` principal ubicado en la carpeta del proyecto **HospitalMgmt**.

Esta separación permite que cada aplicación tenga su propio sistema de rutas de forma independiente, lo que favorece la modularidad y escalabilidad del proyecto cuando se manejan múltiples aplicaciones.

En la **Figura 4** se ilustran los pasos a seguir:

1. Crear el archivo `urls.py` dentro de la carpeta de la aplicación (**hospital**).
2. Abrir el archivo `urls.py` del proyecto principal (**HospitalMgmt**) e importar la función `include` desde `django.urls`. Esta función permite enlazar las URLs definidas en otras aplicaciones.
3. Agregar una entrada en la lista de `urlpatterns` para vincular las URLs de la aplicación utilizando `path` e `include`. Por ejemplo:

```
path('hospital/', include('hospital.urls')),
```

Con esto, todas las rutas definidas en el archivo `hospital/urls.py` estarán accesibles bajo el prefijo `/hospital/`.

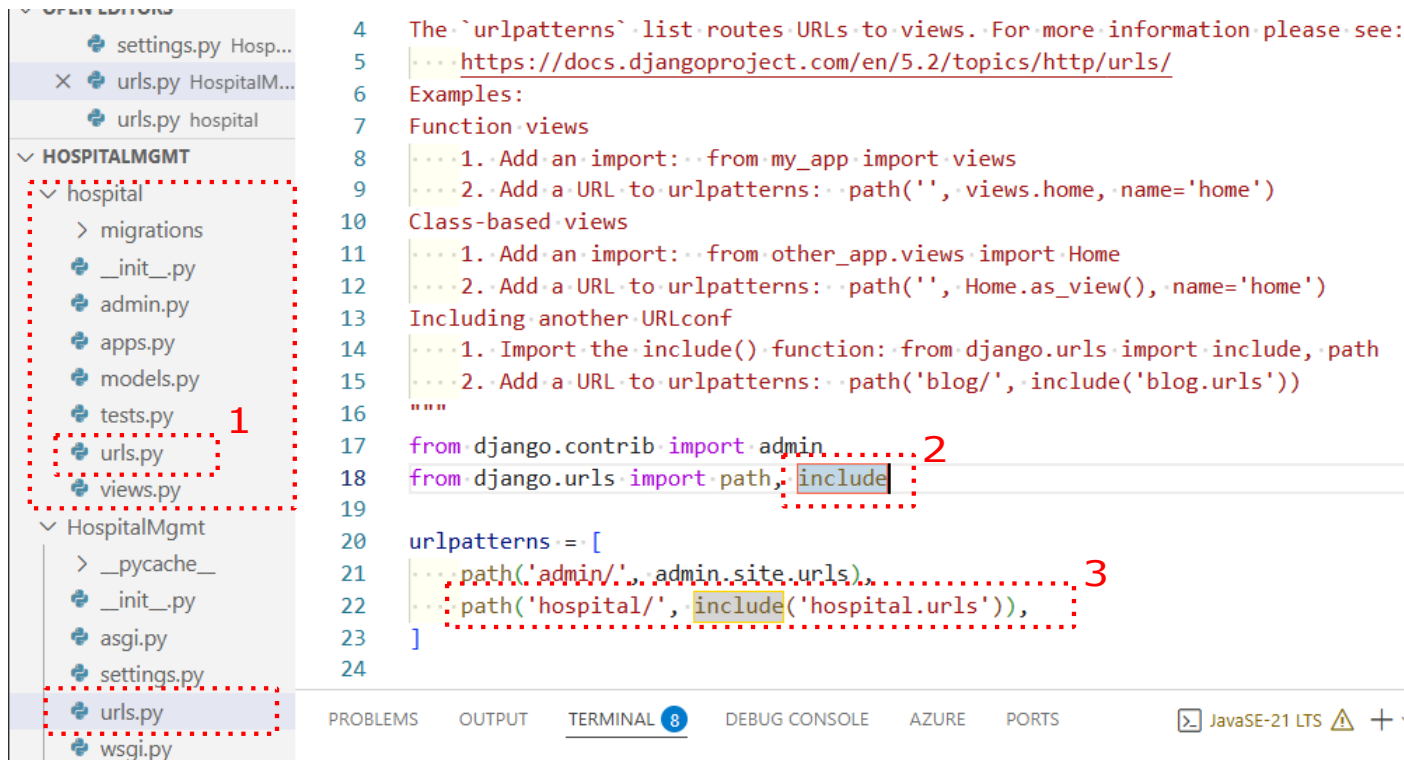


Figura 4: Relacionar urls de la aplicacion a la raiz.

5.2.3. Crear carpetas templates y static

El último paso previo a implementar la lógica del proyecto y su conexión con la base de datos consiste en crear dos carpetas fundamentales dentro de la aplicación —en este caso, **hospital**—: **templates** y **static**.

- La carpeta **templates** se utiliza para almacenar los archivos `.html` que conforman la interfaz visual del usuario. Estos archivos representan las vistas del frontend y se integran con las funcionalidades definidas en Django mediante el motor de plantillas.
- La carpeta **static** contiene los archivos estáticos del proyecto, como hojas de estilo `.css`, imágenes y scripts `.js`, que mejoran la presentación y experiencia de usuario en el frontend.

En la **Figura 5** se muestra la ubicación correcta de estas carpetas dentro de la estructura del proyecto.

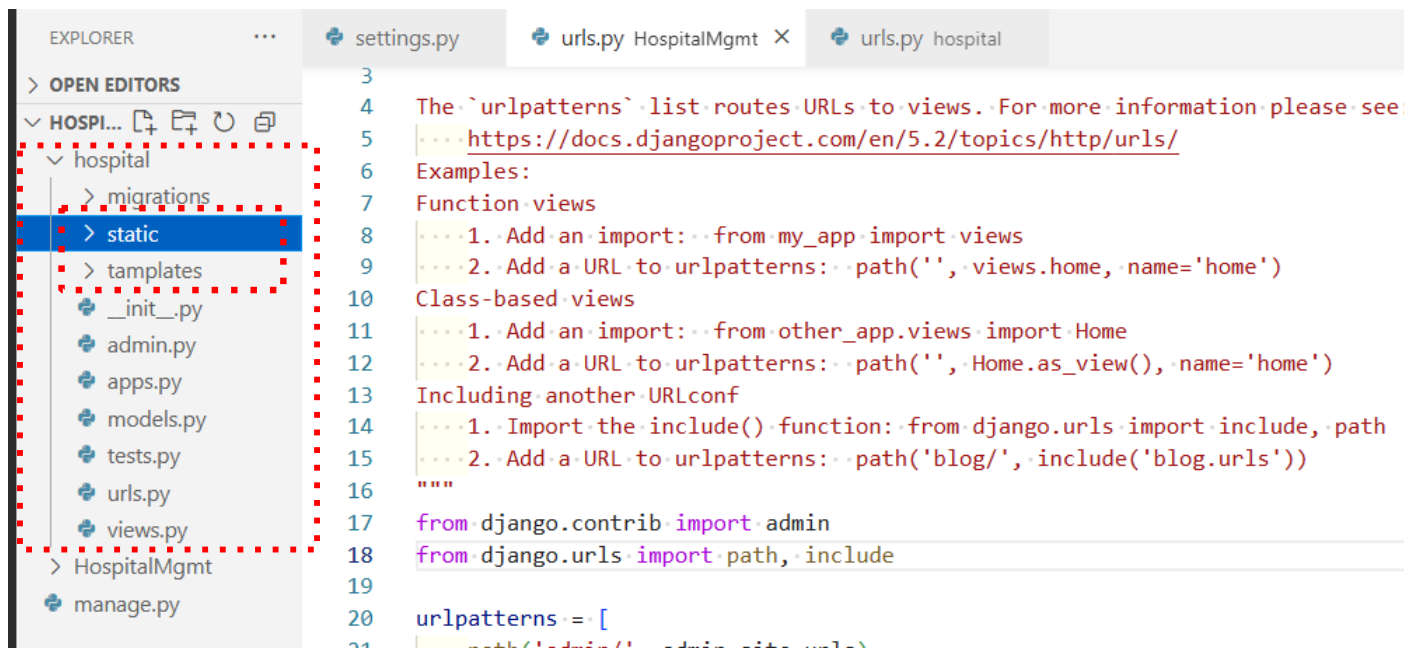


Figura 5: Crear carpetas templates y static.

5.3. Crear un super usuario

Para acceder a la interfaz administrativa de Django y gestionar usuarios, es necesario crear un super-usuario. Este usuario tendrá permisos totales sobre el sistema y permitirá, entre otras acciones, registrar nuevos usuarios y asignarles el rol de doctor, lo cual será esencial para probar el acceso restringido a las vistas que se desarrollarán posteriormente.

1. Agregar el siguiente código al archivo **urls.py** de la aplicación creada, para evitar errores con los **url-patterns**

```
from django.urls import path
from .views import *

urlpatterns = [
]
```

2. Ejecutar migraciones iniciales Antes de crear cualquier usuario, se deben aplicar las migraciones pre-determinadas de Django. Estas migraciones crean las tablas básicas del sistema, como las relacionadas con usuarios, grupos y permisos:

```
python manage.py migrate
```

3. Crear el superusuario Una vez aplicadas las migraciones, se puede crear el superusuario con el siguiente comando:

```
python manage.py createsuperuser
```

Al ejecutarlo, el sistema solicitará los siguientes datos:

- Nombre de usuario
- Correo electrónico (opcional)

■ Contraseña

Para esta práctica, la contraseña no necesita cumplir con requisitos de complejidad estricta.

En la **Figura 6** se muestran visualmente los pasos anteriores,

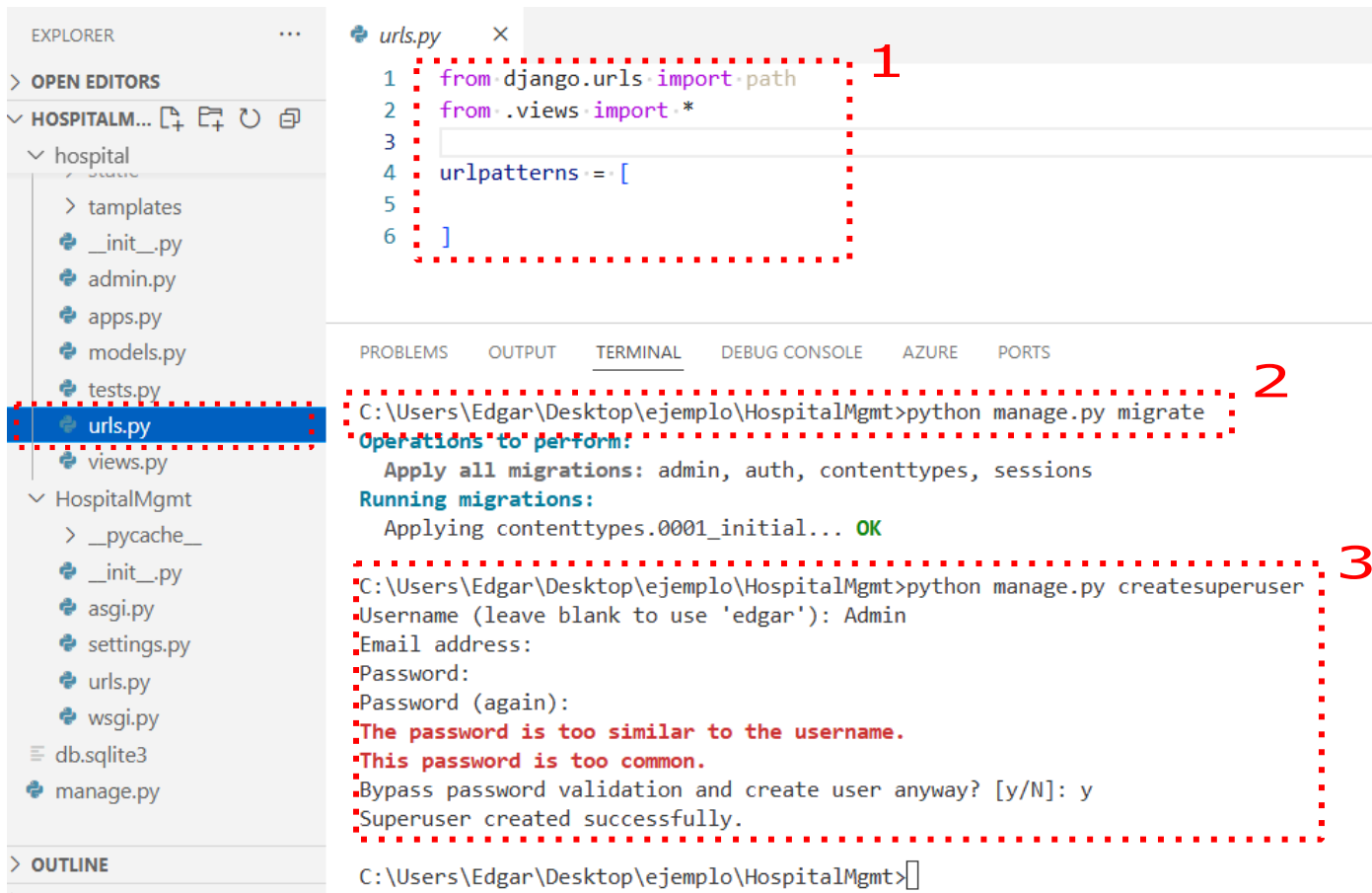


Figura 6: Crear usuario desde la terminal.

5.4. Crear tablas en la base de datos

Para definir las tablas que utilizará la aplicación **hospital**, se hace uso del archivo `models.py`. Este archivo permite declarar, mediante código Python, los modelos que Django interpretará como tablas en la base de datos. Para este proyecto se ocupa la base de datos **SQLite** que Django tiene por defecto.

Las tablas que se crean corresponden a los elementos clave con los que interactuarán los usuarios del sistema, en particular aquellos con el rol de doctor. Cada tabla se representa mediante una clase modelo en Django, lo cual facilita su manipulación desde el ORM (Object-Relational Mapping).

El código base utilizado para definir los modelos en este proyecto puede consultarse en el archivo **models.py** dentro de la carpeta **hospital**.

En esta práctica se definen cinco modelos principales:

1. **Doctor:** Representa a los usuarios con rol médico, quienes tienen acceso a las vistas protegidas del sistema. Cada instancia corresponde a un doctor que puede registrar pacientes y generar consultas médicas.
2. **Síntoma:** Almacena los síntomas que pueden presentar los pacientes. Su existencia permite realizar análisis estadísticos, como identificar síntomas recurrentes en ciertas zonas geográficas o periodos específicos.

3. **Medicamento:** Contiene los medicamentos disponibles o registrados en el sistema, los cuales pueden ser recetados durante las consultas médicas. Esta tabla facilita su reutilización y asociación a múltiples consultas.
4. **Paciente:** Guarda la información personal de los pacientes atendidos por los doctores. Cada paciente puede estar vinculado a uno o varios doctores y puede tener múltiples consultas médicas registradas.
5. **Consulta:** Registra cada atención médica realizada por un doctor a un paciente. Incluye campos como la fecha de la consulta, observaciones, peso del paciente, síntomas identificados y medicamentos prescritos. Este modelo tiene relaciones de muchos a muchos con los modelos **Síntoma** y **Medicamento**.

Estas tablas están interrelacionadas para asegurar la integridad referencial y la coherencia de los datos. Esta estructura facilita una gestión eficiente de la información médica y mejora la escalabilidad del sistema en futuras extensiones.

5.5. Backend

En este apartado se trabaja con el archivo **views.py**, el cual es responsable de manejar la lógica del lado del servidor en la arquitectura de Django. Su función principal es recibir las solicitudes realizadas por el usuario (conocidas como *request*), procesarlas según la lógica de negocio definida y devolver una respuesta adecuada (*response*), ya sea en forma de una página HTML, redirección o datos estructurados (por ejemplo, JSON).

Dentro de este archivo se definen las vistas, es decir, funciones o clases que responden a las rutas configuradas en el archivo `urls.py`. Cada vista tiene un propósito específico en el sistema, como mostrar información, procesar formularios, actualizar registros o eliminar datos de forma lógica.

A continuación se describe brevemente el propósito de cada vista implementada en este proyecto, y en el archivo **views.py** dentro de la carpeta **hospital** se puede consultar el código fuente completo correspondiente.

1.
 - **Nombre de la función:** `es_doctor(user)`.
 - **Tipo de vista:** Función auxiliar (no retorna una vista, sino que sirve como filtro de acceso).
 - **Acceso:** Interno (llamada desde una vista, no expuesta directamente).
 - **Descripción General:** La función `es_doctor` es una vista auxiliar utilizada como condición de seguridad para controlar el acceso a ciertas partes del sistema. En particular, permite restringir vistas (como dashboards o acciones administrativas) a usuarios que pertenezcan explícitamente al grupo de nombre 'Doctores'.
2.
 - **Nombre de la vista:** `login_doctor(request)`
 - **Tipo de vista:** Función basada en vista (FBV).
 - **Acceso:** Pública (disponible para usuarios no autenticados).
 - **Descripción General:** La vista `login_doctor` es responsable de autenticar usuarios que tienen el rol de doctor, validando tanto sus credenciales como su pertenencia al grupo 'Doctores'. Si el usuario ya está autenticado, se redirige directamente a la página principal. En caso contrario, se muestra el formulario de inicio de sesión. Solo los usuarios que pertenezcan al grupo 'Doctores' podrán acceder correctamente.
3.
 - **Nombre de la vista:** `index(request)`
 - **Tipo de vista:** Función basada en vista (FBV)
 - **Acceso:** Protegida (accesible para usuarios autenticados que pertenezcan al grupo 'Doctores')
 - **Descripción General:** La vista `index` representa la página de inicio para los usuarios con rol de doctor. Está protegida mediante el decorador `@user_passes_test`, que valida que el usuario autenticado forme parte del grupo 'Doctores'. Una vez verificada su pertenencia, la vista recupera el objeto Doctor asociado al `user` actual y lo utiliza para personalizar la plantilla `index.html` con el nombre completo del doctor.
4.
 - **Nombre de la función:** `validar_datos_paciente(post_data)`

- **Tipo de función:** Función auxilia (Validación de formulario).
 - **Acceso:** Interno (llamada desde una vista, no expuesta directamente)
 - **Descripción General:** La función `validar_datos_paciente` tiene como propósito validar los datos del formulario enviados al registrar un nuevo paciente. Esta validación incluye:
 - Verificar que todos los campos obligatorios estén presentes.
 - Validar que la edad esté dentro de un rango lógico y sea numérica.
 - Se utiliza una vista que gestiona el registro o creación de un nuevo paciente.
5. ▪ **Nombre de la función:** `crear_paciente(data, doctor)`
- **Tipo de función:** Auxiliar (Creación de modelo)
 - **Acceso:** Interno (llamada desde una vista, no expuesta directamente).
 - **Descripción General:** La función `crear_paciente` es responsable de crear una nueva instancia del modelo `Paciente` utilizando los datos enviados desde un formulario. Una vez creado el paciente, se asocia con el doctor que ha iniciado sesión. Esta asociación se basa en una relación **Many-ToMany** entre `Paciente` y `Doctor`. Esta función encapsula la lógica de creación para mantener la vista limpia y reutilizable.
6. ▪ **Nombre de la función:** `crear_consulta(data, paciente, doctor)`.
- **Tipo de función:** Función auxilia (Lógica de negocio).
 - **Acceso:** Interno (llamada desde una vista).
 - **Descripción General:** La función `crear_consulta` se encarga de registrar una nueva consulta médica en el sistema. Esta consulta se asocia tanto al paciente como al doctor que la realiza, y también permite asociar síntomas y medicamentos utilizando relaciones **ManyToMany**. Los síntomas y medicamentos se reciben como cadenas separadas por comas y se vinculan a la consulta si ya existen en la base de datos.
7. ▪ **Nombre de la vista:** `registro_paciente(request)`
- **Tipo de vista:** Función basada en vista (FBV)
 - **Acceso:** Protegida (`@login_required`)
 - **Descripción General:** La vista `registro_paciente` permite al usuario autenticado (médico) registrar un nuevo paciente junto con su primera consulta médica. La lógica incluye validación de los datos del formulario, creación de un nuevo objeto `Paciente`, asociación con el `Doctor` autenticado, y registro de una Consulta médica con síntomas y medicamentos asociados.
8. ▪ **Nombre de la vista:** `listar_pacientes(request)`
- **Tipo de vista:** Función basada en vista (FBV)
 - **Acceso:** Protegida (`@login_required`)
 - **Descripción General:** La vista `listar_pacientes` permite a los usuarios autenticados visualizar el listado de pacientes cuyo estado es "activo". Además de pasar los objetos `Paciente` al contexto para ser mostrados en el HTML, también los serializa en formato JSON. Esto permite que puedan ser utilizados dinámicamente desde el frontend mediante JavaScript (por ejemplo, para búsquedas, filtros, o carga dinámica).
9. ▪ **Nombre de la vista:** `actualizar_paciente(request)`
- **Tipo de vista:** Función basada en vista (FBV)
 - **Acceso:** Requiere que el usuario esté autenticado.
 - **Descripción General:** La vista `actualizar_paciente` permite modificar dinámicamente los datos básicos de un paciente a partir de una solicitud HTTP POST con contenido JSON. Está diseñada para integrarse con interfaces interactivas que no requieren recargar la página a través de `fetch()`. Solo actualiza los campos definidos como permitidos (nombre, apellido y edad) para evitar modificaciones no deseadas.
10. ▪ **Nombre de la vista:** `eliminacion_logica_paciente(request)`
- **Tipo de vista:** Función basada en vista (FBV)

- **Acceso:** Protegida (requiere usuario autenticado)
 - **Descripción General:** La vista `eliminacion_logica_paciente` permite realizar una eliminación lógica de un paciente, lo cual consiste en marcar su estado como "inactivo" sin borrar físicamente el registro de la base de datos. Esto permite conservar el historial clínico y cumplir con principios de trazabilidad y auditoría. Es útil en interfaces donde el usuario puede "eliminar" visualmente un paciente, pero los datos siguen almacenados para futuros análisis o recuperación.
11. ■ **Nombre de la vista:** `logout_doctor(request)`
- **Tipo de vista:** Función basada en vista (FBV)
 - **Acceso:** Protegida (requiere usuario autenticado)
 - **Descripción General:** La vista `logout_doctor` gestiona el cierre de sesión para usuarios del sistema administrativo, específicamente para doctores o usuarios autenticados. Primero verifica si el usuario está activo. Si no lo está (por ejemplo, usuario no autenticado o deshabilitado), se redirige inmediatamente a la página de inicio de sesión. Si el usuario está activo, se ejecuta el cierre de sesión y luego se redirige a la misma página de login.
12. ■ **Nombre de la vista:** `detalle_paciente(request, paciente_id)`
- **Tipo de vista:** Función basada en vista (FBV)
 - **Acceso:** Protegida (requiere usuario autenticado)
 - **Descripción General:** La vista `detalle_paciente` muestra el historial completo de consultas médicas de un paciente específico identificado por `paciente_id`. Recupera el paciente de la base de datos y obtiene todas sus consultas ordenadas por fecha descendente para mostrar la información más reciente primero. Finalmente, renderiza la plantilla con estos datos para su visualización.
13. ■ **Nombre de la vista:** `nueva_consulta(request, paciente_id)`
- **Tipo de vista:** Función basada en vista (FBV)
 - **Acceso:** Protegida (requiere usuario autenticado)
 - **Descripción General:** La vista `nueva_consulta` permite registrar una nueva consulta médica para un paciente específico. En solicitudes POST, crea una nueva consulta vinculada al paciente identificado por `paciente_id` y al doctor autenticado, agregando los síntomas y medicamentos indicados en el formulario. En solicitudes diferentes a POST, renderiza el formulario para crear una nueva consulta.

5.6. Fronted

En esta sección se trabaja con la carpeta `templates`, la cual tiene como propósito almacenar los archivos HTML que conforman la interfaz gráfica del sistema. Esta carpeta contiene las plantillas que serán renderizadas por Django y presentadas al usuario final.

Cada archivo HTML define la estructura visual y los elementos interactivos que permiten al usuario comunicarse con el sistema. Las plantillas se organizan de forma modular y reutilizable, permitiendo separar la lógica de presentación de la lógica de negocio.

Las vistas de Django (`views.py`) se encargan de enviar datos a estas plantillas para que puedan ser mostrados de forma dinámica, facilitando la visualización de información médica, formularios, listados y navegación general del sistema.

Los códigos bases para esta sección se encuentran en la carpeta **templates** dentro de la carpeta **hospital**.

1. ■ **Archivo HTML:** `login.html`
 - **Propósito:** Este archivo representa la interfaz de inicio de sesión para los usuarios del sistema. Su objetivo es autenticar al usuario (en este caso, doctores) permitiéndoles ingresar sus credenciales.
 - **Componentes principales:**
 - Formulario de acceso con campos para usuario y contraseña, y validación HTML5.

- Mensajes de error (renderizados desde Django con `% if messages %`) que informan si el acceso fue denegado o incorrecto.
 - Diseño responsivo basado en Bootstrap 5.
 - Seguridad: se incluye `% csrf_token %` para proteger el formulario contra ataques CSRF.
- **Funcionalidad JavaScript:**
 - Se utiliza una IIFE (Immediately Invoked Function Expression) para aplicar validación personalizada al formulario.
 - Se muestra una notificación temporal de error cuando Django envía mensajes desde el backend.
 - Si los campos del formulario están vacíos o inválidos, se evita el envío del formulario.
 - **Enlaces relacionados:**
 - Vista relacionada: `login_doctor(request)`
 - Ruta en `urls.py`: `path('login/', login_doctor, name='login')`
2. ■ **Archivo HTML:** `navegador.html`
- **Propósito:** Este archivo funciona como la plantilla base para todas las vistas del sistema. Define la estructura general de la interfaz gráfica, incluyendo el diseño del encabezado (`<head>`), la navegación principal y bloques reutilizables para contenido (`% block %`), facilitando la herencia y reutilización de código HTML en otras páginas.
 - **Componentes principales:**
 - `% load static %` permite usar rutas relativas a recursos estáticos como CSS o JS.
 - Se incluyen Bootstrap 5 y una hoja de estilos personalizada (`style.css`) para diseño y estilos consistentes.
 - Se genera una barra de navegación condicionalmente, solo si el usuario está autenticado (`request.user.is_active`).
 - Cada elemento `` representa una sección accesible para el usuario de tipo "doctor".
 - Muestra mensajes tipo alerta enviados desde las vistas mediante el framework de mensajes de Django (`django.contrib.messages`).
 - Se implementa un temporizador que oculta automáticamente la alerta después de 2 segundos.
 - `% block title %`: permite a cada página definir su propio título.
 - `% block content %`: espacio principal donde cada página hija inserta su contenido.
 - `% block scripts %`: permite insertar scripts adicionales según las necesidades de cada vista.
3. ■ **Archivo HTML:** `index.html`
- **Propósito:** Este archivo representa la página de inicio del sistema para los usuarios autenticados con rol de doctor. Se encarga de dar una bienvenida personalizada al profesional de salud que ha iniciado sesión.
 - **Componentes principales:**
 - **Herencia de plantilla:** `% extends 'navegador.html' %`: el archivo hereda la estructura y navegación definidas en `navegador.html`, reutilizando encabezado, estilos y barra de navegación.
 - **Bloques heredados:**
 - `% block title %`: define el título de la pestaña del navegador como Inicio.
 - `% block content %`: contiene el mensaje de bienvenida centrado horizontalmente mediante `position: relative`.
 - **Enlaces relacionados:**
 - `index(request)` en `views.py`
 - Ruta en `urls.py`: `path('/', index, name='index')`
4. ■ **Archivo HTML:** `registroPaciente.html`
- **Propósito:** Este archivo define la interfaz gráfica para registrar a un nuevo paciente junto con su primera consulta médica. Está diseñado para ser utilizado exclusivamente por usuarios con rol de doctor, proporcionando un formulario organizado, validado y responsivo.

- **Componentes principales:**
 - **Bloque de contenido principal (% block content %):** Contiene el formulario principal distribuido en dos tarjetas (card):
 - Datos Personales
 - Consulta Médica Inicial
 - **Formulario:**
 - Utiliza `class="needs-validation novalidate` para aplicar validaciones personalizadas mediante Bootstrap y JavaScript.
 - Incluye el token de seguridad `% csrf_token %` necesario para proteger la petición POST.
 - **Funcionalidad de JavaScript:**
 - **Validación personalizada del formulario (con Bootstrap):**
 - **Ubicación:** dentro de una IIFE (Immediately Invoked Function Expression) para encapsular el código y evitar conflictos con otras variables o scripts globales.
 - **Lógica:**
 - ◊ Se seleccionan todos los formularios que tengan la clase `needs-validation`.
 - ◊ A cada uno se le añade un listener que intercepta el evento `submit`.
 - ◊ Si el formulario no es válido (`checkValidity()`), se previene el envío y se detiene la propagación del evento.
 - ◊ En todos los casos, se añade la clase `was-validated` al formulario, lo cual activa los estilos de validación visual de Bootstrap.
 - ◊ Antes de enviar, se inserta automáticamente la fecha local actual en formato ISO (sin zona horaria) en el campo oculto con `id="fecha"`.
 - **Generación de fecha local (obtenerFechaLocal()):**
 - **Propósito:** capturar la fecha y hora local del navegador del usuario y convertirla en un string ISO compatible con formatos de base de datos (sin el sufijo "Z" que indica UTC).
 - **Proceso:**
 - ◊ Obtiene la fecha y hora actual con `new Date()`.
 - ◊ Aplica una compensación (`getTimezoneOffset`) para eliminar la diferencia de zona horaria.
 - ◊ Devuelve el resultado con `toISOString().slice(0, 19)` para obtener solo la parte `YYYY-MM-DDTHH:MM:SS`.
 - **Enlaces relacionados:**
 - `registro_paciente(request)` en `views.py`
 - Ruta en `urls.py`: `path('registro/', registro_paciente, name='registro-paciente')`
5. ■ **Nombre del archivo:** `listarPacientes.html`
- **Propósito:** Esta plantilla está diseñada para permitir a los doctores visualizar, gestionar y operar sobre los pacientes activos registrados en el sistema. Facilita la interacción con los datos de los pacientes mediante una interfaz clara y dinámica.
 - **Componentes principales:**
 - **Tabla de pacientes (<table>):** Muestra el listado de pacientes con sus nombres, apellidos y un conjunto de botones de acción:
 - Información
 - Actualizar
 - Nueva Consulta
 - Eliminar (lógica)
 - **Contenedores dinámicos ocultos (div.opciones_pacientes):** Secciones que se muestran al hacer clic en alguno de los botones:
 - **#info_paciente:** Información completa del paciente y su historial.
 - **#actualizaPaciente:** Formulario para editar los datos del paciente.
 - **#consulta_paciente:** Formulario para crear una nueva consulta médica
 - **#ver_consulta:** Detalle completo de una consulta médica anterior.

- **Formularios con validación (needs-validation):** Aplicados en las vistas de actualización y nueva consulta, permiten la validación del lado del cliente mediante Bootstrap.
- **Funcionalidad de JavaScript:**
 - **Variables globales y configuración inicial:** Declara e inicializa las variables necesarias para gestionar la lógica de actualización de pacientes, selección de filas y control de estado del formulario.
 - **Inicialización de Validación de Formularios:** Inicializa la validación de formularios con Bootstrap al cargar la página. Si el formulario no es válido, se previene su envío. Además, antes de enviarlo, se inserta la fecha local en el campo oculto `#fecha`.
 - **Detección de cambios en campos editables:** Escucha los cambios en los campos editables del formulario de actualización de paciente. Una vez que se detecta algún cambio, el botón de enviar se habilita.
 - **Función obtenerIdFila(boton):** Identifica el índice y el ID del paciente correspondiente a la fila de la tabla desde la cual se ha hecho clic en un botón (por ejemplo, "Información", "Actualizar", "Eliminar", etc.).
 - **Función eliminarInfo(boton):** Eliminar lógicamente un paciente del sistema al presionar el botón correspondiente en la tabla. La acción se comunica con el backend mediante una solicitud POST.
 - **Función enviarActualizacion():** Enviar al servidor los datos modificados de un paciente para actualizar su información en la base de datos de forma dinámica, sin recargar la página.
 - **Función obtenerFechaLocal():** Obtener la fecha y hora local del sistema en formato estándar (YYYY-MM-DDTHH:MM:SS), sin el sufijo "Z" (que indica zona horaria UTC), para registrarla de forma coherente en los formularios del sistema.
 - **Función ocultarFondo(nombre):** Resaltar visualmente una sección específica de la interfaz (como información, edición o consulta de un paciente) desenfocando y desactivando temporalmente la interacción con el resto del contenido (lista de pacientes y barra de navegación).
 - **Función cerrarInfo(nombre):** Cerrar u ocultar una sección secundaria de la interfaz (como un panel de información, formulario de edición o consulta médica) y restaurar la visualización y la interacción completa del contenido principal (lista de pacientes y barra de navegación).
 - **Función muestraNotificacion(type, message):** Mostrar notificaciones temporales en pantalla para informar al usuario sobre el resultado de una acción (éxito, error u otro estado), sin interrumpir el flujo del sistema ni requerir una recarga de página.
 - **Función obtenerCookie(nombre):** Recuperar el valor de una cookie específica almacenada en el navegador del usuario. Es especialmente útil en aplicaciones Django para obtener el token CSRF necesario en solicitudes POST realizadas con JavaScript.
 - **Función actualizarInfo(boton):** Preparar e iniciar el proceso de edición de un paciente. Muestra el formulario de actualización con los datos actuales del paciente seleccionado, listos para ser modificados por el usuario.
 - **Función muestraInfo(boton):** Mostrar de forma detallada la información personal y el historial médico del paciente seleccionado. Esta función habilita la vista lateral de información sin necesidad de recargar la página.
 - **Función nuevaConsulta(boton):** Activar el formulario de "Nueva Consulta" para el paciente seleccionado, ajustando dinámicamente el destino (action) del formulario según el paciente correspondiente.
 - **Función verConsulta(boton):** Mostrar los detalles completos de una consulta médica previa de un paciente en una sección dedicada, ocultando la información general del paciente.
 - **Función cerrarInfoConsulta(nombre):** Cerrar el panel de visualización de una consulta médica específica y restaurar la vista general de información del paciente.
- **Enlaces relacionados:**
 - `views.listar_pacientes:` Renderiza esta plantilla
 - `views.actualizar_paciente:` Se invoca vía `fetch()` al hacer clic en "Actualizar".
 - `views.eliminacion_logica_paciente:` Se llama desde el botón "Eliminar".
 - `views.nueva_consulta:` Procesa el formulario de consulta.
 - `views.detalle_paciente:` Provee los datos médicos del paciente.

5.7. Conexión entre Backend y Frontend

Para establecer la comunicación entre el usuario (frontend) y el servidor (backend), se utiliza el archivo `urls.py`. Este archivo tiene como función principal capturar las solicitudes realizadas por el navegador mediante URLs y redirigirlas a la vista correspondiente definida en el archivo `views.py`.

En esta sección se trabaja específicamente con el archivo `urls.py` de la aplicación **hospital**. En él, se definen las rutas (denominadas `path`) que permiten conectar una URL específica con una función del backend.

Cada entrada en `urls.py` tiene la siguiente estructura:

```
path('ruta/', vista_asociada, name='nombre_interno')
```

A continuación se explican los parámetros:

- **'ruta/':** Es la cadena que define el fragmento de la URL que debe coincidir. Por ejemplo, 'login/' corresponde a la URL `http://localhost:8000/login/`.
- **vista_asociada:** Es la función definida en el archivo `views.py` que se ejecutará cuando se acceda a la ruta especificada.
- **name='nombre_interno':** Es un identificador simbólico que permite referirse a esa ruta desde otras partes del sistema, como plantillas HTML o funciones de redirección. Esto hace que las URLs sean más fáciles de mantener.

En la Figura 7 se muestra el listado de rutas que se utiliza en esta práctica, junto con sus respectivas funciones en el backend.



Figura 7: path utilizados para los views.

6. Pruebas y validación

En esta sección se describe el procedimiento necesario para ejecutar la aplicación web, verificar su correcto funcionamiento y confirmar que cumple con los objetivos establecidos en etapas previas del desarrollo.

Una vez creados los archivos y configuraciones descritos en las secciones anteriores, es necesario aplicar las migraciones para reflejar los modelos definidos en la base de datos. Para ello, se deben ejecutar los siguientes comandos en la terminal:

```
python manage.py makemigrations python manage.py migrate
```

El primer comando genera los archivos de migración que traducen los modelos definidos en `models.py` a instrucciones comprensibles por la base de datos. El segundo comando aplica esas migraciones, creando físicamente las tablas en la base de datos.

Posteriormente, se inicia el servidor de desarrollo de Django con el siguiente comando:

```
python manage.py runserver
```

Este comando activa el servidor local y proporciona una dirección URL –por defecto `http://127.0.0.1:8000/`– que permite acceder al sistema desde cualquier navegador web. Esta URL da acceso tanto a la interfaz de usuario desarrollada como al panel administrativo de Django.

En la Figura 8, se muestra la salida de la terminal al ejecutar el comando `runserver`, así como la URL que debe ingresarse en el navegador para interactuar con el sistema.

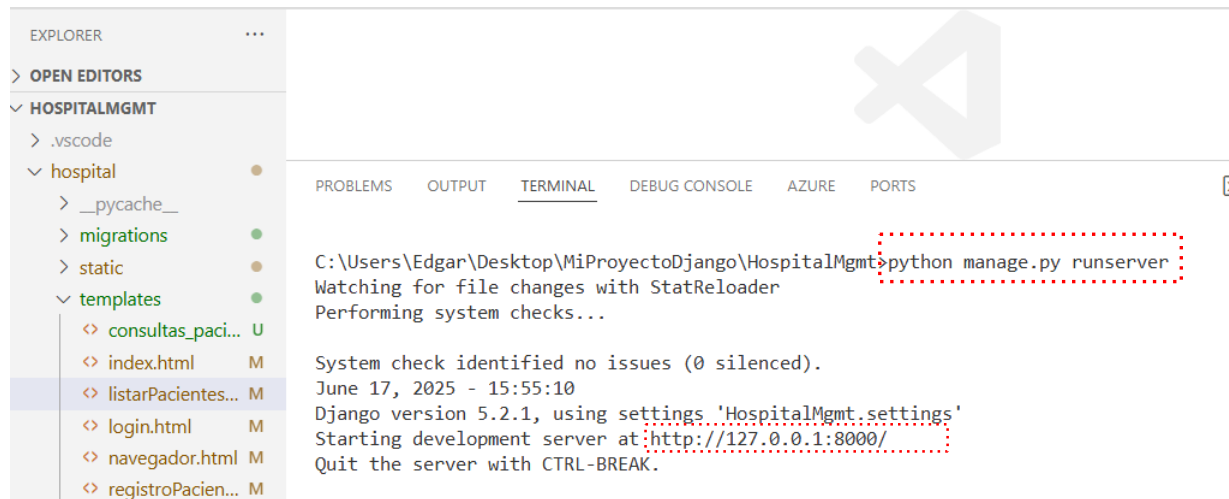


Figura 8: Ejecución del servidor local con runserver

Una vez iniciada la aplicación, se puede acceder a la interfaz administrativa de Django mediante la URL mostrada (por lo general, `http://127.0.0.1:8000/admin`). En esta interfaz se debe iniciar sesión con el super-usuario creado anteriormente.

La Figura 9 ilustra el panel de administración de Django, donde se pueden visualizar las tablas que fueron definidas en el archivo `models.py` y gestionarlas directamente desde esta interfaz.

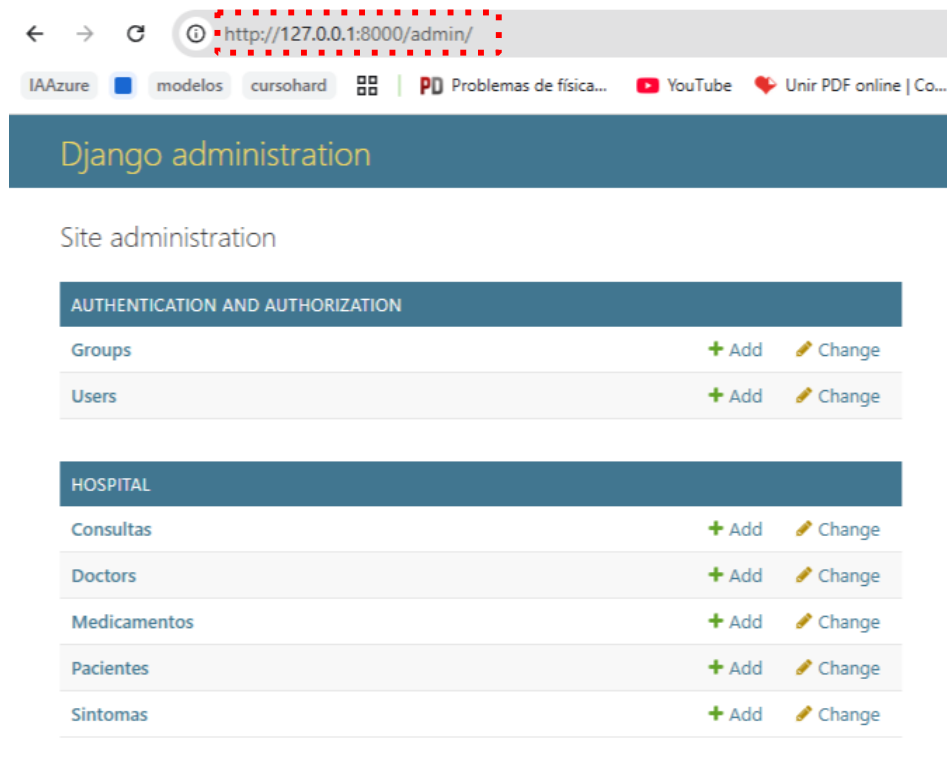


Figura 9: Interfaz administrativa de Django

Dentro del panel de administración de Django, es posible crear usuarios personalizados, grupos y objetos asociados a modelos definidos en la aplicación. En este caso, se crea un nuevo usuario, un grupo denominado "doctor", y posteriormente se asocia un objeto del modelo Doctor. Este procedimiento permite asignar un rol específico al usuario y establecer la relación correspondiente con la tabla Doctor.

Para crear un nuevo usuario, siga los siguientes pasos:

1. En el panel de administración, haga clic en la opción Users.
2. Luego, seleccione Add user.
3. Complete los campos requeridos: Username, Password y Password confirmation.
4. Finalmente, haga clic en el botón Save para registrar el nuevo usuario.

En la Figura 10 se muestra visualmente este proceso.

Add user

After you've created a user, you'll be able to edit more user options.

Username:

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password-based authentication: ☒ Enabled ☐ Disabled
Whether the user will be able to authenticate using a password or not. If disabled, they may still be able to authenticate using other backends, such as Single Sign-On.

Password:

Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.

Password confirmation:

Enter the same password as before, for verification.

SAVE **Save and add another** **Save and continue editing**

Figura 10: Creación de usuario en el panel administrativo de Django

Para crear un grupo en el panel de administración de Django, siga los pasos que se detallan a continuación:

1. En el panel de administración, haga clic en la opción Groups.
2. Luego, seleccione la opción Add group.
3. Complete los campos requeridos:
 - Name: nombre que identificará al grupo (por ejemplo, doctor).
 - Permissions: conjunto de permisos que determinarán a qué acciones o vistas podrán acceder los usuarios que pertenezcan a este grupo.

En el campo de permisos, es posible asignar selectivamente los accesos específicos para cada grupo. Sin embargo, para efectos de esta práctica y simplificar el proceso, se recomienda hacer clic en Choose all permissions para otorgar todos los permisos disponibles al grupo. En la Figura 11 se muestra el formulario correspondiente a esta configuración.

The screenshot shows the Django Admin interface for creating a group. The 'Name' field is set to 'doctores'. The 'Permissions' section is divided into two panes: 'Available permissions' and 'Chosen permissions'. The 'Available permissions' pane has a search filter and a list of permissions. The 'Chosen permissions' pane also has a search filter and a list of permissions. A red arrow points to the 'Choose all permissions' button. At the bottom, a red arrow points to the 'SAVE' button.

Figura 11: Creación de grupo con permisos en Django

El siguiente paso consiste en registrar un nuevo doctor en el sistema, asociándolo tanto al grupo doctor como al usuario previamente creado. Esta relación es fundamental para que dicho usuario tenga acceso autorizado a las vistas correspondientes dentro del sistema.

Para crear un registro en la tabla Doctor y vincularlo correctamente, siga los pasos que se describen a continuación:

1. En el panel de administración de Django, haga clic en la opción Doctors.
2. Luego, seleccione Add Doctor.
3. En el campo User, elija el usuario creado previamente (por ejemplo, **DoctorPrueba**). Este será el usuario con el que el doctor podrá iniciar sesión en el sistema.
4. Complete los campos requeridos: Nombre, Apellido, Edad y Sexo.

Cabe destacar que estos campos pueden adaptarse a un enfoque más profesional si se desea, por ejemplo, incluyendo campos como la cédula profesional, especialidad médica, entre otros. No obstante, para efectos de simplificación en esta práctica, solo se consideran los datos básicos.

En la Figura 12 se muestran los campos a completar y el proceso de creación de un nuevo doctor en la base de datos.

Django administration

Home > Hospital > Doctors > Add doctor

Add doctor

User: DoctorPrueba

Nombre: daniel

Apellido: perez gomez

Edad: 45

Sexo: hombre

SAVE Save and add another Save and continue editing

Figura 12: Formulario para registrar un nuevo doctor

El último paso consiste en asociar el usuario **DoctorPrueba** al grupo **doctores**. Esta asignación es necesaria para que el sistema reconozca que dicho usuario tiene el rol correspondiente y, por lo tanto, acceso autorizado a las vistas protegidas.

Para realizar esta acción, siga los siguientes pasos:

1. Ingrese al panel de administración de Django.
2. Haga clic en la opción **Users**.
3. Seleccione el usuario previamente creado, en este caso **DoctorPrueba**.
4. En el formulario de edición del usuario, ubique la sección **Groups** y seleccione el grupo **doctores**.
5. Finalmente, haga clic en **Save** para guardar los cambios.

En la Figura 13 se muestra visualmente cómo se realiza la asignación del usuario al grupo correspondiente.

☐ Superuser status
Designates that this user has all permissions without explicitly assigning them.

Groups:

Available groups
Choose groups by selecting them and then select the "Choose" arrow button.

Filter

Chosen groups
Remove groups by selecting them and then select the "Remove" arrow button.

Filter

doctores

Choose all groups

Remove all groups

The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down "Control", or "Command" on a Mac, to select more than one.

Figura 13: Asignación del usuario DoctorPrueba al grupo doctores

Para finalizar el uso del panel de administración de Django, se recomienda agregar datos de ejemplo en las tablas **Medicamentos** y **Síntomas**. Esto permitirá simular un entorno real de trabajo y evitar posibles errores al momento de registrar una consulta médica desde la interfaz del sistema.

Para realizar esta tarea, siga los siguientes pasos:

1. Ingrese al panel de administración.
2. Acceda a la tabla **Medicamentos** y agregue al menos cinco medicamentos distintos.
3. Luego, acceda a la tabla **Síntomas** y registre cinco síntomas comunes.

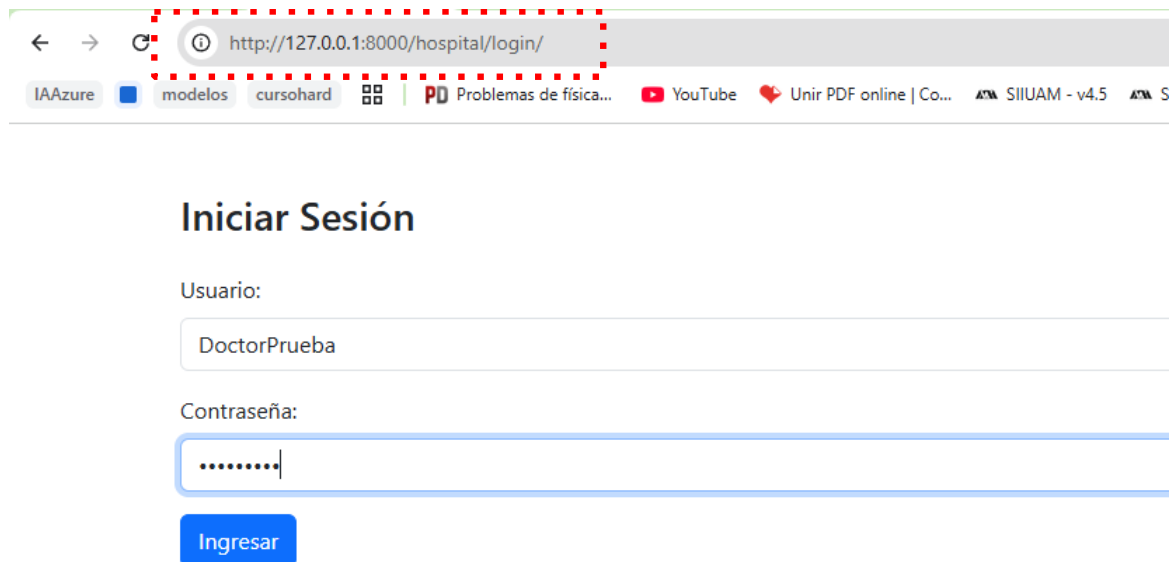
Estos registros iniciales facilitarán el correcto funcionamiento del sistema durante las pruebas y el uso regular de la aplicación.

Para acceder al sistema como un usuario con el rol de doctor, primero es necesario cerrar sesión en el panel de administración de Django. Para ello, haga clic en la opción **Logout** ubicada en la esquina superior derecha.

Una vez cerrada la sesión, abra el navegador y diríjase a la siguiente URL:

`http://127.0.0.1:8000/hospital/login/`

Esta dirección redirige a la interfaz de inicio de sesión del sistema. En ese formulario, deberá ingresar el nombre de usuario y la contraseña del usuario previamente creado, en este caso **DoctorPrueba**. La Figura 14 muestra el aspecto de esta pantalla de acceso.



The screenshot shows a web browser window with the address bar displaying `http://127.0.0.1:8000/hospital/login/`, which is highlighted by a red dashed rectangle. Below the browser window, the login page is visible. It has a title "Iniciar Sesión". There are two input fields: "Usuario:" containing the text "DoctorPrueba" and "Contraseña:" containing masked characters ".....". A blue button labeled "Ingresar" is positioned below the password field.

Figura 14: iniciar sesión con usuario doctor.

En la Figura 15 se muestra la pantalla que aparece tras un inicio de sesión exitoso. En ella se presenta un mensaje de bienvenida personalizado que incluye el nombre del doctor, recuperado desde la tabla **Doctor**, confirmando que el usuario ha sido autenticado correctamente en el sistema.

[Agregar Paciente](#) [Listar Pacientes](#) [Modificar Usuario](#)

Bienvenidos doctor daniel perez gomez

Figura 15: sesion existosa.

En la Figura 16 se observa el comportamiento del formulario cuando se intenta enviar con campos vacíos. El sistema valida que todos los campos obligatorios estén completos y, en caso contrario, muestra mensajes de advertencia específicos en cada campo, evitando así el envío de información incompleta.

[Agregar Paciente](#)

[Listar Pacientes](#)

[Modificar Usuario](#)

Registro de Paciente

Datos Personales

Nombre:

Nombre del paciente

Por favor ingrese un nombre válido (solo letras)

Apellidos:

Apellidos del paciente

Por favor ingrese apellidos válidos (solo letras)

Edad:

Edad

La edad debe estar entre 0 y 120 años

Sexo:

☒ Hombre

☐ Mujer

☐ Otro/Prefiero no decir

Consulta Medica

Peso:

En la Figura 17 se muestra el proceso de creación de un nuevo paciente una vez que todos los campos han sido completados correctamente. Al ingresar la información de forma válida, el sistema registra exitosamente al paciente en la base de datos y lo asocia con el doctor que inició sesión.

Figura 17: crear un paciente.

Para visualizar los pacientes que han sido registrados, se debe hacer clic en la opción Listar Pacientes ubicada en la barra de navegación. Esta opción despliega una tabla con todos los pacientes activos registrados en el sistema.

En la Figura 18 se muestra dicha lista, junto con los botones de Información, Actualizar, Nueva Consulta y Eliminar (eliminación lógica). Estas funcionalidades permiten gestionar de forma individual los datos de cada paciente, facilitando un control personalizado y eficiente dentro del sistema.

Nombres	Apellidos	Opciones
bruno	perez	Informacion Actualizar Nueva Consulta Eliminar
laura	angeles	Informacion Actualizar Nueva Consulta Eliminar

Figura 18: Listado de pacientes activo y registrados.

Para comprobar el funcionamiento de los botones disponibles, se muestra la información de un paciente,

se realiza una actualización de sus datos, se registra una nueva consulta y se verifica que la información correspondiente se despliegue correctamente y se refleje en la interfaz. Finalmente, se aplica una eliminación lógica al paciente, lo que implica un cambio en su estado de activo a inactivo dentro del sistema.

En la Figura 19 se observa a la izquierda, el resultado de hacer clic en el botón Información, donde se despliega un panel con los datos personales del paciente. A la derecha, se muestra la vista de una consulta médica específica al seleccionar la opción Ver dentro del historial médico. Este comportamiento implica ocultar el bloque actual para dar paso visual al siguiente, permitiendo una navegación secuencial y ordenada entre secciones.

The figure displays two side-by-side screenshots of a web application interface. The left screenshot shows a modal titled 'Información Paciente' with a 'Cerrar Información' button in the top right. It contains two sections: 'Datos Personales' with fields for Nombre (bruno), Apellidos (perez), Edad (20), and Género (hombre); and 'Historial Médico' with a sub-header 'Detalles de bruno perez' and a list of medical history entries. A red arrow points to a 'Ver' button within the history list. The right screenshot shows a modal titled 'Consulta' with a 'Cerrar Información' button in the top right. It contains fields for Fecha (2025-06-17 23:23), Peso (60), Síntomas (Fiebre), and Medicamentos (Paracetamol).

Figura 19: Información del paciente.

En la Figura 20 se observa el resultado de seleccionar el botón Actualizar. En este ejemplo, se modifica el contenido de los campos Nombre y Apellido del segundo paciente. El botón de actualizar solo se activará cuando un campo sea modificado.

The figure shows a modal titled 'Actualizar Paciente' with a 'Cerrar Información' button in the top right. It contains a 'Datos Personales' section with input fields for Nombre (laura itxel), Apellidos (angeles gomez), and Edad (65). Red dashed boxes highlight the Nombre and Apellidos fields. A red arrow points to an 'Actualizar' button at the bottom center of the modal.

Figura 20: panel para actualizar datos personales.

Cuando un paciente se actualiza correctamente, se resalta con una línea azul la fila correspondiente como indicativo visual de que la operación fue exitosa. Además, los cambios realizados se reflejan de inmediato en el listado de pacientes, sin necesidad de recargar la página. Véase la Figura 23.

Agregar Paciente		Listar Pacientes	Modificar Usuario
Lista de Pacientes Activos			
Nombres	Apellidos	Opciones	
bruno	perez	Informacion	Actualizar Nueva Consulta Eliminar
laura itxel	angeles gomez	Informacion	Actualizar Nueva Consulta Eliminar

Figura 21: panel para actualizar datos personales.

En la Figura 22 se muestra el panel correspondiente a la creación de una nueva consulta, el cual se despliega al hacer clic en el botón "Nueva Consulta". En este caso, se registra una nueva consulta para el paciente previamente actualizado. Posteriormente, en la figura 24 se muestra un mensaje de éxito al crear una nueva consulta.

Medicamentos:

Naproxend

Observaciones o Instrucciones:

El paciente refiere dolor abdominal de tipo cólico, de leve a moderada intensidad, sin signos de irritación peritoneal ni alteraciones gastrointestinales graves. Se indica tratamiento sintomático con naproxeno 250 mg cada 12 horas, acompañado de alimentos, durante 2 a 3 días. Se recomienda vigilancia de la evolución del dolor, evitar automedicación adicional y acudir a reevaluación si el dolor se intensifica o se acompaña de vómito persistente, fiebre o sangrado.

Limpiar

Crear Consulta

Figura 22: Crear nueva consulta.

se creo con exito la consulta

Lista de Pacientes Activos

Nombres	Apellidos	Opciones
bruno	perez	Informacion Actualizar Nueva Consulta Eliminar
laura itxel	angeles gomez	Informacion Actualizar Nueva Consulta Eliminar

Figura 23: Mensaje de exito al crear una consulta.

Al acceder nuevamente a la opción de información, se muestra que los valores fueron modificados correctamente, lo que confirma que los datos han sido almacenados exitosamente en la base de datos, ver figura 24.

Nombre:laura itxel
Edad:65

Apellidos:angeles gomez
Género:hombre

Historial Médico

Detalles de laura itxel angeles gomez

- 2025-06-18 00:23 - El paciente refiere dolor abdominal de tipo cólico, de leve a moderada intensidad, sin signos de irritación peritoneal ni alteraciones gastrointestinales graves. Se indica tratamiento sintomático con naproxeno 250 mg cada 12 horas, acompañado de alimentos, durante 2 a 3 días. Se recomienda vigilancia de la evolución del dolor, evitar automedicación adicional y acudir a reevaluación si el dolor se intensifica o se acompaña de vómito persistente, fiebre o sangrado. [Ver](#)
- 2025-06-17 23:31 - El paciente presenta cefalea de tipo tensional, sin signos de alarma neurológica. Se indica tratamiento con naproxeno como analgésico y antiinflamatorio, a una dosis de 250 mg cada 12 horas por un periodo de 3 días, acompañado de alimentos. Se recomienda reposo, adecuada hidratación y reevaluación si el dolor persiste o se intensifica. [Ver](#)

Figura 24: Visualizar información del paciente con los cambios hechos.

A continuación, se utiliza el botón Eliminar, el cual remueve visualmente al paciente de la lista de pacientes activos. Sin embargo, este procedimiento no elimina el registro de la base de datos, sino que realiza una eliminación lógica, cambiando su estado a inactivo. Esta decisión permite conservar el historial clínico del paciente para futuras consultas o análisis. En la Figura 25, se observa que, tras ejecutar esta acción, el paciente ya no aparece en la lista.

[Agregar Paciente](#)[Listar Pacientes](#)[Modificar Usuario](#)

Lista de Pacientes Activos

Nombres	Apellidos	Opciones
laura itxel	angeles gomez	Informacion Actualizar Nueva Consulta Eliminar

Figura 25: eliminacion logica de paciente.

Finalmente, para cerrar sesión en el sistema, se debe hacer clic en la opción Salir, lo cual redirige al usuario a la pantalla de inicio de sesión.