



45GIIN SISTEMAS DE INFORMACIÓN WEB

Actividad UC2

Desarrollo del backend de una aplicación web

Moisés Sevilla Corrales
PROF. Doctor Horacio Daniel Kuna

11 de diciembre de 2024

Índice

Desarrollo del backend de una aplicación web.....	3
Consigna:	3
Desarrollo de una BD	3
Implementación servicios de backend	7
Realización de pruebas documentadas.....	9
Clientes	15
Agendar cita médica.....	15
Consultar disponibilidad de horarios.....	18
Cancelar o reprogramar citas	19
Diagrama de arquitectura	20
Conclusiones.....	22
Bibliografía	23

https://github.com/moisessevilla/45GIIN_Consultorio

Desarrollo del backend de una aplicación web.

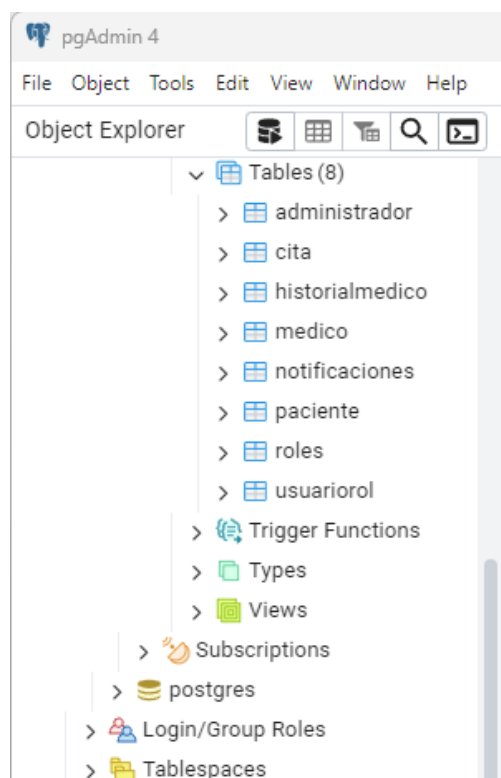
Consigna:

Sobre el mismo escenario de desarrollo de un sistema de información de la Actividad





















Práctica I se van a ejecutar las siguientes actividades:

Desarrollo de una BD

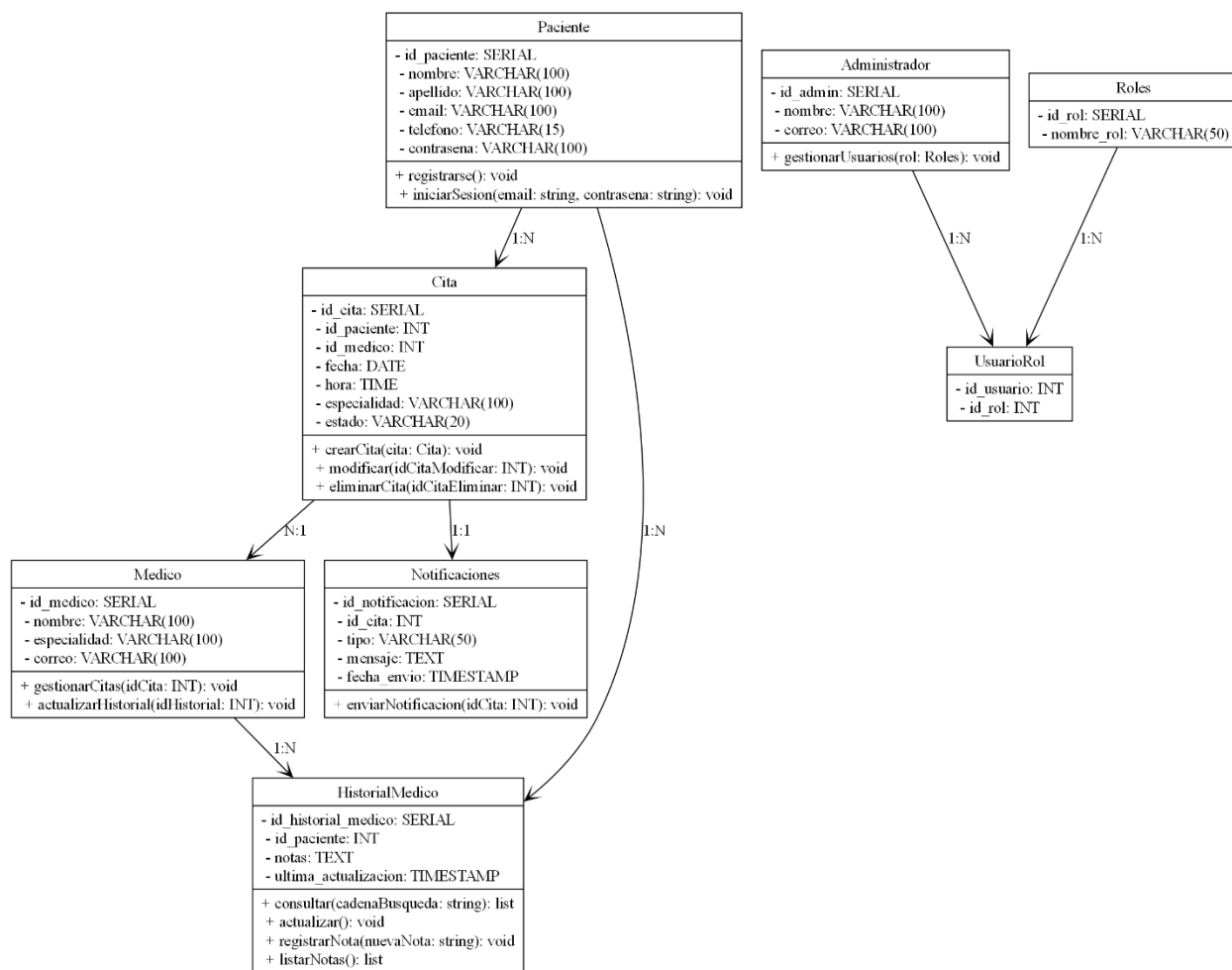
1. Desarrollar una **BD** relacionada con la necesidad de información organizacional determinada en la Actividad Práctica I. Adjuntar el documento del diagrama de clases correspondiente.
- Se crea en **PostgreSQL** una base de datos llamada “**consultorio**”, se crean también las tablas mediante un script con nombre “**consultorio.sql**”, de aquí en adelante los ficheros que se mencionen estarán disponibles en github.



Estructura de tablas	
<ul style="list-style-type: none"> administrador 	<ul style="list-style-type: none"> administrador <ul style="list-style-type: none"> Columns (3) <ul style="list-style-type: none"> id_admin nombre correo
<ul style="list-style-type: none"> cita 	<ul style="list-style-type: none"> cita <ul style="list-style-type: none"> Columns (7) <ul style="list-style-type: none"> id_cita id_paciente id_medico fecha hora especialidad estado
<ul style="list-style-type: none"> historialmedico 	<ul style="list-style-type: none"> historialmedico <ul style="list-style-type: none"> Columns (4) <ul style="list-style-type: none"> id_historial_medico id_paciente notas ultima_actualizacion
<ul style="list-style-type: none"> medico 	<ul style="list-style-type: none"> medico <ul style="list-style-type: none"> Columns (4) <ul style="list-style-type: none"> id_medico nombre especialidad correo

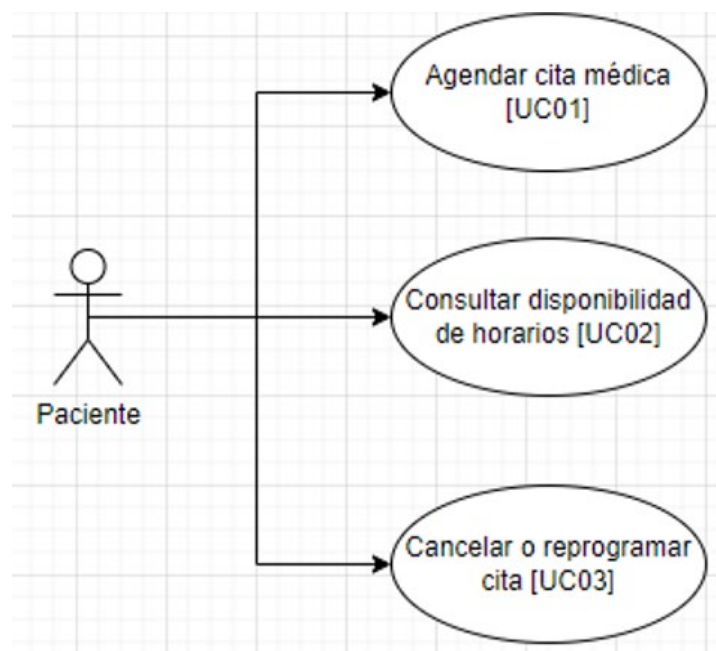
<ul style="list-style-type: none"> • notificaciones 	<ul style="list-style-type: none"> ▼  notificaciones <ul style="list-style-type: none"> ▼  Columns (5) <ul style="list-style-type: none">  id_notificacion  id_cita  tipo  mensaje  fecha_envio
<ul style="list-style-type: none"> • paciente 	<ul style="list-style-type: none"> ▼  paciente <ul style="list-style-type: none"> ▼  Columns (6) <ul style="list-style-type: none">  id_paciente  nombre  apellido  email  telefono  contraseña
<ul style="list-style-type: none"> • roles 	<ul style="list-style-type: none"> ▼  roles <ul style="list-style-type: none"> ▼  Columns (2) <ul style="list-style-type: none">  id_rol  nombre_rol
<ul style="list-style-type: none"> • usuariolor 	<ul style="list-style-type: none"> ▼  usuariolor <ul style="list-style-type: none"> ▼  Columns (2) <ul style="list-style-type: none">  id_usuario  id_rol

- Se adjunta el diagrama de clases correspondiente actualizado, con opciones más depuradas que en la Actividad 1, fichero “diagrama_clases.png y .py”



Implementación servicios de backend

2. Implementar los **servicios de backend** correspondientes para la gestión de la información en cuestión (similar a una operación de tipo CRUD). Al menos se deberá **implementar lo correspondiente a los tres (3) casos de uso** que fueron especificados en la actividad previa.
- En la actividad 1 vimos 6 casos de uso, pero vamos a realizar en esta ocasión los 3 casos de uso que pertenecen al paciente:



- Para el backend vamos a usar **Django** y una serie de “**endpoints**” cargados en nuestro fichero “**views.py**” como crear, leer, actualizar y eliminar pacientes, agendar citas médicas, consultar disponibilidad de horarios y cancelar o reprogramar citas.
- Esto nos darán feedback durante las pruebas CRUD con **Django REST framework**, para ello hemos diseñado un pequeño menú sencillo “**opciones.html**”, para manejarnos entre las distintas opciones y facilitarnos la tarea durante el proceso, tomando este aspecto amigable.

Paciente List Create

DELETE OPTIONS GET

GET: Lista todos los pacientes o filtra por id_paciente, nombre o telefono.
POST: Crea uno o varios pacientes.
PUT/PATCH: Actualiza un paciente existente.
DELETE: Elimina un paciente por id_paciente.

GET /citas/paciente/?id_paciente=0

HTTP 200 OK
Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
[]

Media type: application/json

Content:

POST

Media type: application/json

Content:

PUT PATCH

The screenshot shows a web browser window with the URL `127.0.0.1:8000`. The browser's address bar and tabs are visible. Below the browser window, there is a navigation bar with five links: [Gestión de Pacientes](#), [Agendar Cita](#), [Disponibilidad de Horarios](#), [Gestionar Citas](#), and [Administración](#). The main content area is titled 'Django REST framework' and 'Paciente List Create'. It features a title 'Paciente List Create' with 'OPTIONS' and 'GET' buttons to its right. Below the title, there is a description: 'GET: Lista todos los pacientes. POST: Crea un nuevo paciente si no existe por id_paciente.' The interface also shows the URL `GET /citas/paciente/` and the HTTP response details: 'HTTP 200 OK', 'Allow: GET, POST, HEAD, OPTIONS', 'Content-Type: application/json', and 'Vary: Accept'. The response body is an empty array `[]`. At the bottom, there is a form with a 'Media type' dropdown set to 'application/json' and a 'Content' text area. A 'POST' button is located at the bottom right of the form.

Realización de pruebas documentadas

3. Realizar **pruebas y documentarlas** utilizando alguna de las herramientas disponibles a tal efecto.

- Como hemos comentado en el apartado anterior para las pruebas usaremos el **Django REST framework**.

- Empezamos las pruebas creando pacientes, cargamos el JSON de ejemplo.

Media type:
application/json

Content:

```
{
  "id_paciente": 1,
  "nombre": "Juan",
  "apellido": "Pérez",
  "email": "Juan.perez@example.com",
  "telefono": "123456789",
  "contrasena": "segura123"
}
```

POST

- Nos devuelve que ha sido creado correctamente.

POST: Crea un nuevo paciente si no existe por id_paciente.

POST /citas/paciente/

HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "id_paciente": 1,
  "nombre": "Juan",
  "apellido": "Pérez",
  "email": "juan.perez@example.com",
  "telefono": "123456789",
  "contrasena": "segura123"
}
```

- Intentamos crear de nuevo el mismo paciente para ver si lo duplica y comprobar que nuestra condición de que valide, si existe el usuario nos avise.

The screenshot shows a REST client interface. On the left, a 'Media type' dropdown is set to 'application/json'. Below it, the 'Content' field contains a JSON object for a patient with id 1, name Juan, and other details. A 'POST' button is at the bottom right. On the right, the response is shown: 'POST: Crea un nuevo paciente si no existe por id_paciente.' followed by the URL '/ citas/paciente/'. Below that, the status is 'HTTP 400 Bad Request' with headers 'Allow: GET, POST, HEAD, OPTIONS', 'Content-Type: application/json', and 'Vary: Accept'. The response body is a JSON object with an error message: '{ "error": "El paciente ya existe." }'.

- Cargamos 10 pacientes y realizamos un GET para continuar con las pruebas, comprobamos que los datos han sido introducidos correctamente.

```
{
  "id_paciente": 1,
  "nombre": "Juan",
  "apellido": "Pérez",
  "email": "juan.perez@example.com",
  "telefono": "123456789",
  "contrasena": "segura123"
},
{
  "id_paciente": 2,
  "nombre": "María",
  "apellido": "Gómez",
  "email": "maria.gomez@example.com",
  "telefono": "987654321",
  "contrasena": "password456"
},
{
  "id_paciente": 3,
  "nombre": "Carlos",
  "apellido": "Hernández",
  "email": "carlos.hernandez@example.com",
  "telefono": "555666777",
  "contrasena": "qwerty789"
},
{
  "id_paciente": 4,
  "nombre": "Ana",
  "apellido": "López",
  "email": "ana.lopez@example.com",
  "telefono": "111222333",
  "contrasena": "abc123"
},
{
  "id_paciente": 5,
  "nombre": "Luis",
  "apellido": "Martínez",
  "email": "luis.martinez@example.com",
  "telefono": "444555666",
  "contrasena": "contraseña"
},
{
  "id_paciente": 6,
  "nombre": "Sofía",
  "apellido": "Rodríguez",
  "email": "sofia.rodriguez@example.com",
  "telefono": "777888999",
  "contrasena": "segura456"
},
{
  "id_paciente": 7,
  "nombre": "Miguel",
  "apellido": "García",
  "email": "miguel.garcia@example.com",
  "telefono": "333444555",
  "contrasena": "clave123"
},
{
  "id_paciente": 8,
  "nombre": "Laura",
  "apellido": "Moreno",
  "email": "laura.moreno@example.com",
  "telefono": "888999000",
  "contrasena": "pass456"
},
{
  "id_paciente": 9,
  "nombre": "Pedro",
  "apellido": "Ruiz",
  "email": "pedro.ruiz@example.com",
  "telefono": "666777888",
  "contrasena": "123seguro"
},
{
  "id_paciente": 10,
  "nombre": "Elena",
  "apellido": "Castro",
  "email": "elena.castro@example.com",
  "telefono": "999000111",
  "contrasena": "elena123"
}
```

- Ahora hacemos pruebas para actualizar datos por **PUT** y por **PATCH**.
 - Utilizaremos **PUT** para actualizar todos los campos.
 - Utilizamos **PATCH** para actualizar campos concretos.

Probamos con **PUT** a realizar una actualización de teléfono del id_paciente=1

Media type:
application/json

Content:

```
{
  "id_paciente": 1,
  "nombre": "Juan",
  "apellido": "Pérez",
  "email": "juan.perez@example.com",
  "telefono": "111222333",
  "contrasena": "segura123"
}
```

PUT PATCH

Antes

```
{
  "id_paciente": 1,
  "nombre": "Juan",
  "apellido": "Pérez",
  "email": "juan.perez@example.com",
  "telefono": "123456789",
  "contrasena": "segura123"
},
```

Después

PUT /citas/paciente/?id_paciente=1

HTTP 200 OK
Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "id_paciente": 1,
  "nombre": "Juan",
  "apellido": "Pérez",
  "email": "juan.perez@example.com",
  "telefono": "111222333",
  "contrasena": "segura123"
}
```

- Por ejemplo con **PUT** nos pide que ingresemos todos los campos aunque no se edite el contenido, de lo contrario nos indica que es un campo requerido.

Media type:

application/json

Content:

```
{
  "id_paciente": 1,
  "nombre": "Juan",
  "apellido": "Pérez",
  "email": "Juan.perez@example.com",
  "telefono": "111222333"
}
```

PUT PATCH

PUT /citas/paciente/?id_paciente=1

HTTP 400 Bad Request

Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "contrasena": [
    "Este campo es requerido."
  ]
}
```

- Pero si utilizamos **PATCH**, podemos elegir un único campo a modificar.

Media type:

application/json

Content:

```
{
  "telefono": "444555666"
}
```

PUT PATCH

Antes

Después

PUT /citas/paciente/?id_paciente=1

```
HTTP 200 OK
Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

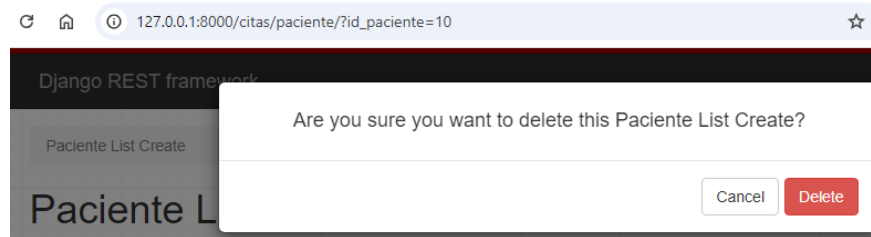
{
  "id_paciente": 1,
  "nombre": "Juan",
  "apellido": "Pérez",
  "email": "Juan.perez@example.com",
  "telefono": "111222333",
  "contrasena": "segura123"
}
```

PATCH /citas/paciente/?id_paciente=1

```
HTTP 200 OK
Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "id_paciente": 1,
  "nombre": "Juan",
  "apellido": "Pérez",
  "email": "Juan.perez@example.com",
  "telefono": "444555666",
  "contrasena": "segura123"
}
```

- Para ir finalizando el CRUD de pacientes usaremos **DELETE**.
 - Probamos eliminando el paciente 10, nos pedirá confirmación, la aceptamos.



DELETE: Elimina un paciente por id_paciente.

```
DELETE /citas/paciente/?id_paciente=10
```

```
HTTP 200 OK
Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "mensaje": "Paciente eliminado exitosamente."
}
```

- Si intentamos eliminar de nuevo el paciente, nos dirá que no lo encuentra, lo mismo pasaría si intenta buscar un id_paciente que no esté registrado.

DELETE: Elimina un paciente por id_paciente.

```
DELETE /citas/paciente/?id_paciente=10
```

```
HTTP 404 Not Found
Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "error": "Paciente no encontrado."
}
```

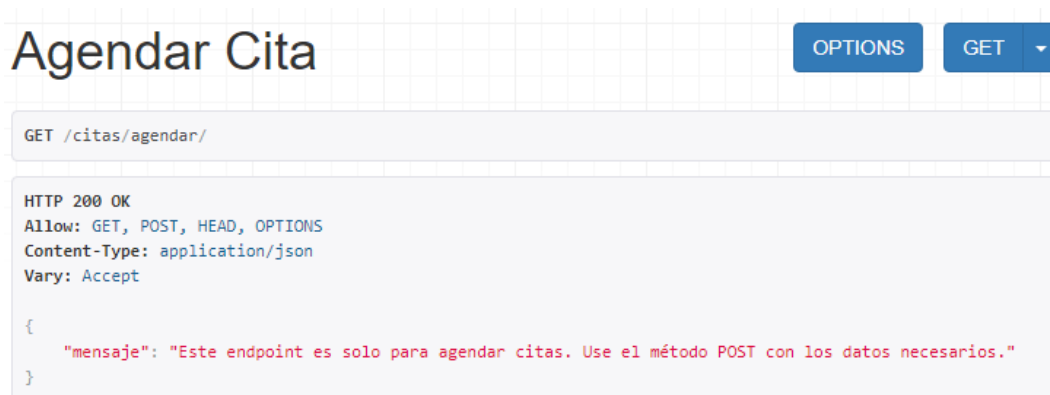
- Aplicamos la misma metodología para los siguientes 3 casos de uso, repitiendo las mismas pruebas, pero con los casos específicos para cada uno de ellos, ampliando el código de los endpoints en “**views.py**”.
- Desglose de verificaciones usadas en cada caso:

Cientes

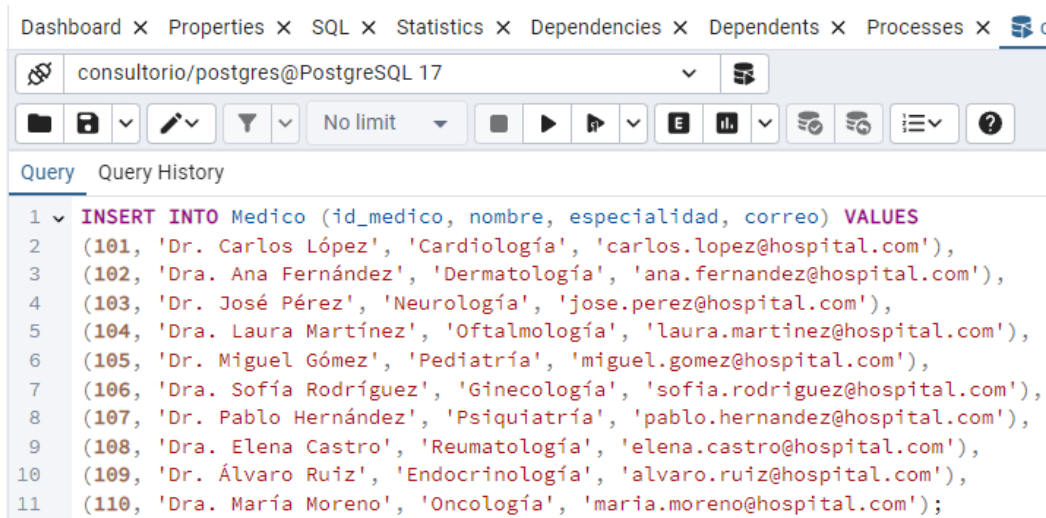
- GET – Muestra todo o filtra la búsqueda por id_paciente, nombre o telefono.
- POST – Comprueba si el paciente ya existe, si no lo crea.
- PUT – Actualiza todos los datos del id_paciente, verifica si falta alguna columna, primero verifica que el id_paciente exista.
- PATCH – Actualiza parcialmente columnas de una tabla buscada por id_paciente, primero verifica que el id_paciente exista.
- DELETE – Verifica primero que el id_paciente exista, si lo encuentra pide consentimiento de eliminación, si aprobamos se elimina.

Agendar cita médica

- En este caso limitamos solo el endpoint para agendar citas, mostramos un mensaje si intentamos leer datos con GET.



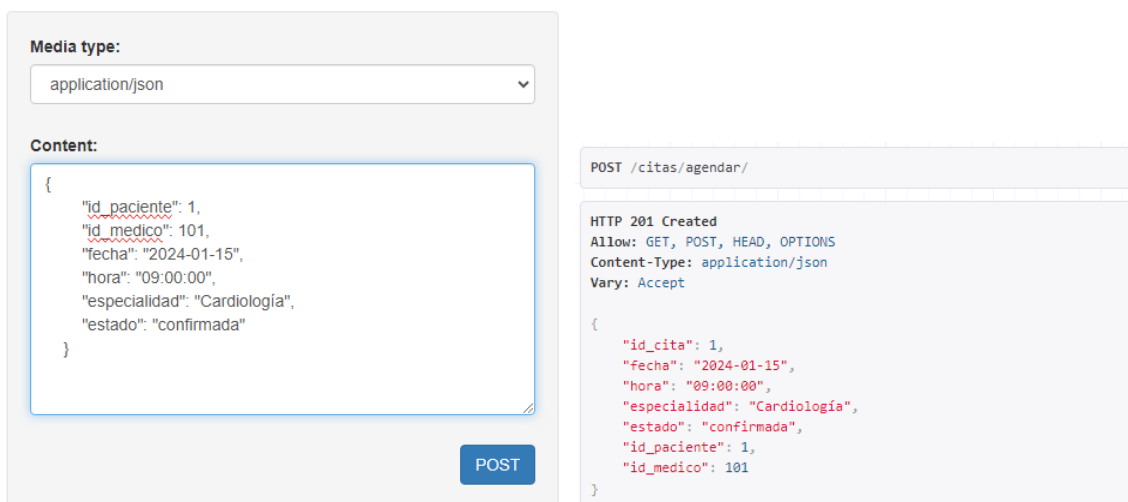
- Como no tenemos médicos dados de alta, de forma rápida le cargo un script SQL al “PgAdmin 4”, para seguir con las pruebas.



The screenshot shows the PgAdmin 4 interface with the 'Query' tab selected. The query is an INSERT statement into the 'Medico' table, adding 10 records with IDs 101 through 110. Each record includes a name, specialty, and email address.

```
1 INSERT INTO Medico (id_medico, nombre, especialidad, correo) VALUES
2 (101, 'Dr. Carlos López', 'Cardiología', 'carlos.lopez@hospital.com'),
3 (102, 'Dra. Ana Fernández', 'Dermatología', 'ana.fernandez@hospital.com'),
4 (103, 'Dr. José Pérez', 'Neurología', 'jose.perez@hospital.com'),
5 (104, 'Dra. Laura Martínez', 'Oftalmología', 'laura.martinez@hospital.com'),
6 (105, 'Dr. Miguel Gómez', 'Pediatría', 'miguel.gomez@hospital.com'),
7 (106, 'Dra. Sofía Rodríguez', 'Ginecología', 'sofia.rodriguez@hospital.com'),
8 (107, 'Dr. Pablo Hernández', 'Psiquiatría', 'pablo.hernandez@hospital.com'),
9 (108, 'Dra. Elena Castro', 'Reumatología', 'elena.castro@hospital.com'),
10 (109, 'Dr. Álvaro Ruiz', 'Endocrinología', 'alvaro.ruiz@hospital.com'),
11 (110, 'Dra. María Moreno', 'Oncología', 'maria.moreno@hospital.com');
```

- POST – Verificamos si existe el id_paciente, el id_medico, fecha y hora, si existe no se asigna una cita, si no se registra una nueva.



The screenshot shows a REST client interface. On the left, the 'Media type' is set to 'application/json' and the 'Content' field contains a JSON object representing a booking request. A 'POST' button is visible. On the right, the response is shown, indicating a successful '201 Created' status with a JSON object containing the assigned booking ID and details.

Media type: application/json

Content:

```
{
  "id_paciente": 1,
  "id_medico": 101,
  "fecha": "2024-01-15",
  "hora": "09:00:00",
  "especialidad": "Cardiología",
  "estado": "confirmada"
}
```

POST

POST /citas/agendar/

HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "id_cita": 1,
  "fecha": "2024-01-15",
  "hora": "09:00:00",
  "especialidad": "Cardiología",
  "estado": "confirmada",
  "id_paciente": 1,
  "id_medico": 101
}
```


- Verificamos para no duplicar citas y generamos 10 citas por JSON o SQL

Media type:
application/json

Content:

```
{
  "id_paciente": 1,
  "id_medico": 101,
  "fecha": "2024-01-15",
  "hora": "09:00:00",
  "especialidad": "Cardiología",
  "estado": "confirmada"
}
```

POST

POST /citas/agendar/

HTTP 400 Bad Request
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "error": "Ya existe una cita en este horario."
}
```

Query Query History

```
1 1 INSERT INTO cita (id_cita, id_paciente, id_medico, fecha, hora, especialidad, estado)
2 (2, 2, 102, '2024-01-15', '10:00:00', 'Dermatología', 'confirmada'),
3 (3, 3, 103, '2024-01-15', '11:00:00', 'Neurología', 'confirmada'),
4 (4, 4, 104, '2024-01-15', '13:00:00', 'Oftalmología', 'confirmada'),
5 (5, 5, 105, '2024-01-15', '14:00:00', 'Pediatría', 'confirmada'),
6 (6, 6, 106, '2024-01-15', '15:00:00', 'Ginecología', 'confirmada'),
7 (7, 7, 107, '2024-01-16', '09:00:00', 'Psiquiatría', 'confirmada'),
8 (8, 8, 108, '2024-01-16', '10:00:00', 'Reumatología', 'confirmada'),
9 (9, 9, 109, '2024-01-16', '11:00:00', 'Endocrinología', 'confirmada'),
10 (10, 10, 110, '2024-01-16', '13:00:00', 'Oncología', 'confirmada');
```

Consultar disponibilidad de horarios

- Aquí hacemos solo una lectura de datos a modo de consulta con GET, filtramos por especialidad y fecha para ver los horarios disponibles.

Verificamos que existen datos

```
GET /citas/gestionarcita/

HTTP 200 OK
Allow: GET, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
  {
    "id_cita": 1,
    "fecha": "2024-01-15",
    "hora": "09:00:00",
    "especialidad": "Cardiología",
    "estado": "confirmada",
    "id_paciente": 1,
    "id_medico": 101
  },
]
```

Filtramos búsqueda

Disponibilidad Horarios

OPTIONS GET

```
GET /citas/disponibilidad/?especialidad=Cardiolog%C3%ADa&fecha=2024-01-15

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "disponibles": [
    "10:00:00",
    "11:00:00",
    "13:00:00",
    "14:00:00",
    "15:00:00"
  ]
}
```

- Si no tenemos citas disponibles nos marca todos los horarios disponibles para esa especialidad:

```
GET /citas/disponibilidad/?especialidad=Cardiolog%C3%ADa&fecha=2024-01-14

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "disponibles": [
    "09:00:00",
    "10:00:00",
    "11:00:00",
    "13:00:00",
    "14:00:00",
    "15:00:00"
  ]
}
```

Cancelar o reprogramar citas

- Aquí no se permite el método POST, solo GET, PATCH y DELETE, para la lectura de citas, reprogramarla o cancelarlas, no introducir nuevas. Aplicamos un filtro para la búsqueda de citas por id_cita, id_paciente o id_medico. Para realizar el patch es imprescindible indicar el id_cita, luego modificamos fecha y hora.

Filtramos por id_medico	Reagendamos la cita	Comprobamos el estado
<pre>GET /citas/gestionarcita/?id_medico=110 HTTP 200 OK Allow: GET, PATCH, DELETE, HEAD, OPTIONS Content-Type: application/json Vary: Accept [{ "id_cita": 10, "fecha": "2024-01-16", "hora": "13:00:00", "especialidad": "Oncología", "estado": "confirmada", "id_paciente": 10, "id_medico": 110 }]</pre>	<pre>PATCH /citas/gestionarcita/?id_medico=110 HTTP 200 OK Allow: GET, PATCH, DELETE, HEAD, OPTIONS Content-Type: application/json Vary: Accept { "mensaje": "Cita reprogramada exitosamente." }</pre> <div>Content:</div> <pre>{ "id_cita": 10, "fecha": "2024-01-16", "hora": "13:00:00", }</pre>	<pre>GET /citas/gestionarcita/?id_medico=110 HTTP 200 OK Allow: GET, PATCH, DELETE, HEAD, OPTIONS Content-Type: application/json Vary: Accept [{ "id_cita": 10, "fecha": "2024-01-20", "hora": "10:00:00", "especialidad": "Oncología", "estado": "confirmada", "id_paciente": 10, "id_medico": 110 }]</pre>

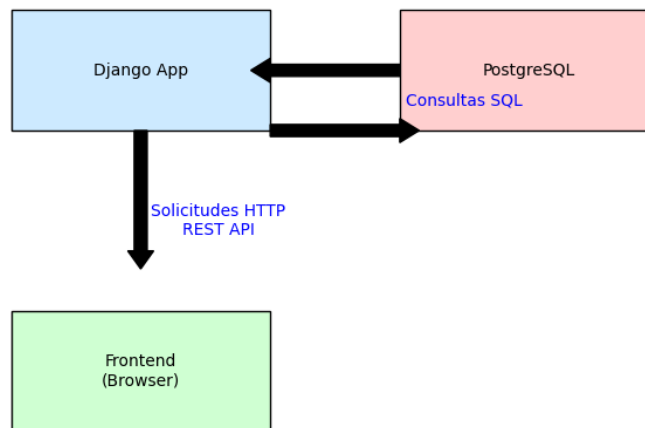
- Usamos DELETE para eliminarla, se solicita el id_cita para su búsqueda.

Cita agendada	Correcta cancelación	Ya no hay cita programada
<pre>GET /citas/gestionarcita/?id_cita=10 HTTP 200 OK Allow: GET, PATCH, DELETE, HEAD, OPTIONS Content-Type: application/json Vary: Accept [{ "id_cita": 10, "fecha": "2024-01-20", "hora": "10:00:00", "especialidad": "Oncología", "estado": "confirmada", "id_paciente": 10, "id_medico": 110 }]</pre>	<pre>DELETE /citas/gestionarcita/?id_cita=10 HTTP 200 OK Allow: GET, PATCH, DELETE, HEAD, OPTIONS Content-Type: application/json Vary: Accept { "mensaje": "Cita cancelada exitosamente." }</pre>	<pre>GET /citas/gestionarcita/?id_cita=10 HTTP 200 OK Allow: GET, PATCH, DELETE, HEAD, OPTIONS Content-Type: application/json Vary: Accept []</pre>

Diagrama de arquitectura

4. Elaborar un **diagrama de la arquitectura** básica que se implementa en la aplicación, se adjunta fichero “**diagrama_arquitectura.png y .py**”.

Diagrama de Arquitectura Básica



- **Cliente web (Browser):**
 - Representa al usuario que interactúa con la aplicación mediante el navegador.
- **Servidor de aplicaciones (Django App):**
 - Procesa las solicitudes REST API enviadas por el cliente y maneja la lógica del negocio.
- **Base de datos (PostgreSQL):**
 - Almacena los datos estructurados de la aplicación, como citas, pacientes y médicos.
- **Interacciones (Django REST framework):**
 - Solicitudes HTTP entre el cliente web y el servidor.
 - Consultas SQL entre el servidor y la base de datos.

- Los ficheros que se adjuntan en GitHub son:
 - Del entorno Django los más significativos.
 - **models.py** (Estructuras de datos y las tablas)
 - **opciones.html** (Menú básico para tener accesos rápidos)
 - **serializers.py** (Convierte modelos django a JSON y viceversa)
 - **settings.py** (Configuración principal, incluye configuración de la BD)
 - **urls.py** (Mapea las rutas URL)
 - **views.py** (Define la lógica de negocio y proceso de rutas HTTP)
 - Ficheros utilizados durante el desarrollo de esta actividad
 - **consultorio.sql** (Con opción a poner los “id_*” en manual)
 - **consultoriomedico.sql** (Opción de los “id_*” con autoincremento)
 - **pacientes.sql** (Opción para volcado desde pgAdmin 4 o similar)
 - **pacientes.json** (Para volcado desde django REST framework o similar)
 - **medico.sql** (Opción para volcado desde pgAdmin 4 o similar)
 - **medico.json** (Para volcado desde django REST framework o similar)
 - **cita.sql** (Opción para volcado desde pgAdmin 4 o similar)
 - **cita.json** (Para volcado desde django REST framework o similar)
 - **diagrama_clases.png** (Representa el diagrama de clases en PNG)
 - **diagrama_clases.py** (Diagrama de clases editable desde Python)
 - **diagrama_arquitectura.png** (Diagrama de arquitectura en PNG)
 - **diagrama_arquitectura.py** (Diagrama de arquitectura editable desde Python sin depender de programas de terceros.)

Conclusiones

Modelado de la base de datos:

- Definimos tablas estructuradas y relaciones clave entre pacientes, médicos y citas, asegurando integridad referencial y optimización para las consultas.

Desarrollo de la API REST:

- Implementamos endpoints funcionales para la gestión de pacientes y citas, permitiendo operaciones CRUD (crear, leer, actualizar y eliminar) con validaciones robustas para evitar inconsistencias en los datos, como conflictos de horarios en las reprogramaciones.

Interacción cliente-servidor:

- Diseñamos endpoints accesibles para clientes web, siguiendo estándares HTTP, y aseguramos una interacción fluida mediante validaciones de parámetros y respuestas significativas.

Lógica de negocio:

- Implementamos reglas claras, como la validación de horarios ocupados y la protección contra actualizaciones redundantes, garantizando un sistema funcional y confiable.

Visualización de la arquitectura:

- Representamos gráficamente la arquitectura del sistema, destacando los principales componentes (cliente, servidor de aplicaciones y base de datos) y sus interacciones.
- El sistema resultante es escalable, flexible y preparado para integrar futuras funcionalidades, como autenticación de usuarios, notificaciones y más especializaciones médicas. Este trabajo refuerza la importancia de planificar, estructurar y validar cada componente de un sistema para garantizar su éxito en entornos reales.

Bibliografía

El código utilizado para la realización de esta actividad se encuentra en GitHub.

URL: https://github.com/moisessevilla/45GIIN_Consultorio

Universidad Internacional de Valencia (VIU).

Libro Sistemas de Información Web

Wilfredo J. Torres Moya, José Javier Barrios Alvarado.

Diagrams.net.

Herramienta en línea para la creación de diagramas UML.

URL: <https://app.diagrams.net/>

Graphviz.org.

Herramienta de visualización de gráficos y diagramas.

URL: <https://graphviz.org/>