

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS.
FUNDAMENTOS DE BASES DE DATOS.

PRÁCTICA 08 JDBC & PSYCOPG

EQUIPO: CHIQUESSQL

IVANA IX CHEL BONILLA NEGRETE
315131994

DYLAN ENRIQUE JUAREZ MARTINEZ
422117180

DANIEL ROJO MATA
314297967

PROFESOR:

GERARDO ÁVILES ROSAS

AYUDANTES DE TEORÍA:

GERARDO URIEL SOTO MIRANDA
VALERIA FERNANDA MANJARREZ ANGELES

AYUDANTES DE LABORATORIO:

RICARDO BADILLO MACÍAS
ROCÍO AYLIN HUERTA GONZÁLEZ

I. ACTIVIDADES

Psycopg

- Para esta práctica, deberán realizar un programa que permita realizar las operaciones CRUD sobre la base de datos que estamos realizando en el curso. Para ello, deberán utilizar Psycopg2 para la comunicación entre el programa y la base de datos. Deberán escoger al menos dos tablas de su base de datos para este propósito. Su programa deberá implementar las operaciones de inserción, eliminación y actualización para cada una de las tablas, además de poder hacer una consulta que nos permita obtener todos los valores de una tabla y una consulta para obtener un solo valor de la tabla dado el identificador.
- Django: Utilizarán Django para realizar una aplicación web. Este framework es el equivalente a Spring Boot de Java. Específicamente, deberán usar Django REST, que podrán incluir en los settings al realizar el proyecto.

Cada clase que realicen deberá estar documentada adecuadamente, siguiendo las instrucciones de documentación de DocStrings para Python. Una vez finalizado el proyecto, deberán guardarlo en la carpeta **SRC**.

- Para ambos proyectos:
 - Deberán realizar un reporte en formato PDF donde definan cuáles fueron las queries que tuvieron que utilizar para la realización de CRUD para cada una de sus tablas y explicar lo que realiza la query, además de las sentencias que usa. Este documento lo llamarán **Practica08.pdf**.
 - Utilizarán POSTMAN para probar su aplicación REST, para cada una de las consultas que realizaron. Las evidencias deberán ser incluidas en el reporte.
 - Deberán agregar el **DDL.sql** y el **DML.sql** de sus prácticas pasadas, y en caso de que modifiquen la estructura de su base de datos, agregar los modelos de Entidad-Relación y Relacional donde muestren el cambio.
- Extra: Se obtendrá un punto extra sobre la práctica si agregan interfaz gráfica para realizar las operaciones CRUD de sus tablas.

I. PSYCOPG

Se utilizó el entorno de desarrollo PyCharm para desarrollar el proyecto, siguiendo la estructura estándar proporcionada por Django.

El proyecto lleva por nombre `DjangoFBD_conectado_BD`.

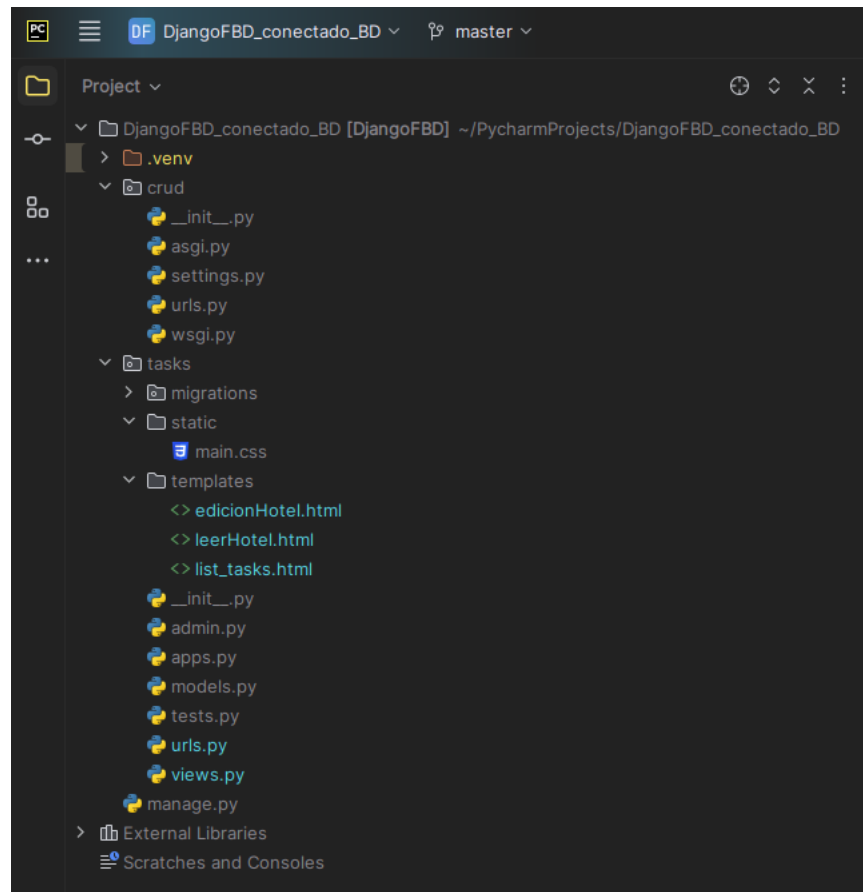


Figura 1: Estructura del proyecto

De manera general se tiene lo siguiente:

■ `DjangoFBD_conectado_BD`.

- **crud**: Contiene los archivos básicos para la configuración y operación de Django. Esto incluye archivos como `asgi.py`, `settings.py`, `urls.py`, y `wsgi.py`. La carpeta `__pycache__` contiene archivos bytecode generados por Python para optimizar el rendimiento.
- **manage.py**: Archivo utilizado para ejecutar comandos de Django desde la línea de comandos, como iniciar el servidor de desarrollo o crear migraciones.
- **tasks**: Es una aplicación Django separada. Aquí se encuentran archivos importantes como:
 - `admin.py`: Para la integración con el panel de administración de Django.
 - `apps.py`: Para configurar la aplicación dentro de Django.

- **migrations**: Contiene archivos de migración para cambios en la base de datos.
- **models.py**: Define los modelos de datos para la aplicación.
- **views.py**: Define las vistas que manejan las solicitudes HTTP.
- **urls.py**: Contiene las rutas para la aplicación.
- **static**: Directorio para archivos estáticos (como hojas de estilo).
- **templates**: Directorio para archivos de plantillas HTML.
- **tests.py**: Contiene las pruebas para la aplicación.

I. CRUD/SETTINGS.PY

El archivo `settings.py` dentro de la carpeta `crud` es fundamental para configurar la aplicación Django. Aquí es donde se especifican las aplicaciones que están instaladas y disponibles para uso en el proyecto.

```
INSTALLED_APPS = [  
    "tasks",  
    "django.contrib.admin",  
    "django.contrib.auth",  
    "django.contrib.contenttypes",  
    "django.contrib.sessions",  
    "django.contrib.messages",  
    "django.contrib.staticfiles",  
]
```

En esta sección, se incluye la aplicación `tasks` (la primera en aparecer), que es una parte esencial del proyecto. Al agregarla a `INSTALLED_APPS`, se permite que Django la reconozca y la integre en el sistema, habilitando sus modelos, vistas, urls, y demás funcionalidades.

Además de `tasks`, las demás aplicaciones predeterminadas de Django son importantes para el funcionamiento del marco.

II. CONFIGURACIÓN DE LA BASE DE DATOS

En el archivo `settings.py`, se configura la conexión con la base de datos que utilizará el proyecto Django. En este ejemplo, se está utilizando PostgreSQL.

```
DATABASES = {  
    "default": {  
        "ENGINE": "django.db.backends.postgresql",  
        "NAME": "postgres",  
        "USER": "postgres",  
        "PASSWORD": "314297967",  
    },  
}
```

```

        "HOST": "localhost",
        "PORT": "5432",
    }
}

```

Esta configuración especifica varios aspectos clave para la conexión con la base de datos:

- **ENGINE:** Define el backend de la base de datos que Django debe usar. Aquí, se utiliza `django.db.backends.postgresql`, lo que indica que se conectará a PostgreSQL.
- **NAME:** El nombre de la base de datos a la que Django se conectará. En este caso, el nombre es `postgres`.
- **USER:** El nombre de usuario utilizado para autenticarse en la base de datos. El valor aquí es `postgres`.
- **PASSWORD:** La contraseña para el usuario especificado. Aquí, el valor es `314297967`.
- **HOST:** Indica dónde se encuentra la base de datos. `localhost` significa que está en la misma máquina donde se ejecuta Django.
- **PORT:** El puerto de la base de datos. Para PostgreSQL, el puerto por defecto es `5432`.

Estos valores deben ser configurados dependiendo del entorno.

La base de datos mencionada existe de manera local, con las credenciales dadas anteriormente.

Name	Owner	Encoding	Locale Provider	Collate	Ctype	ICU Locale	ICU Rules	Access privileges
postgres	postgres	UTF8	libc	en_US.utf8	en_US.utf8			
template0	postgres	UTF8	libc	en_US.utf8	en_US.utf8			=c/postgres + postgres=CTc/postgres
template1	postgres	UTF8	libc	en_US.utf8	en_US.utf8			=c/postgres + postgres=CTc/postgres

(3 rows)

(END)

Figura 2: Bases de datos locales

III. TASKS/MODELS.PY

Dentro de la carpeta `tasks` se tiene el archivo `models.py`, el cual contiene la clase:

```
class Hotel(models.Model):
    """
    Modelo para representar un hotel en la base de datos.

    Atributos:
        idhotel (str): Identificador único del hotel. Usado como clave primaria.
        nombreestablecimiento (str): Nombre del establecimiento del hotel.
        horacheekin (time): Hora estándar de check-in del hotel.
        horacheekout (time): Hora estándar de check-out del hotel.
        petfriendly (str): Indica si el hotel admite mascotas. Ejemplo: 'SI' para sí,
        'NO' para no.
        servicio (str): Tipo de servicio que ofrece el hotel.
        numerointerior (str, opcional): Número interior del hotel. Puede ser nulo.
        numeroexterior (int): Número exterior del hotel.
        colonia (str): Nombre de la colonia donde está ubicado el hotel.
        calle (str): Calle donde está ubicado el hotel.
        estado (str): Estado donde está ubicado el hotel.

    Métodos:
        __str__(): Retorna una representación en cadena del nombre del establecimiento.

    Meta:
        db_table (str): Nombre de la tabla en la base de datos a la cual este modelo
        corresponde.
    """
```

Mediante el siguiente comando se asegura que se esté recibiendo información de y hacia la base de datos configurada en un inció, en concreto a la tabla `Hotel`.

```
class Meta:
    db_table = 'hotel'
```

IV. TASKS/VIEWS.PY

Dentro de este archivo se encuentran los siguientes métodos:

```
def list_tasks(request):
    ...

def createHotel(request):
    ...

def deleteHotel(request, hotel_id):
    ...

def editHotelRender(request, hotel_id):
    ...

def editHotel(request, hotel_id):
    ...

def readHotelRender(request, hotel_id):
    ...

def readHotel(request, hotel_id):
    ...

def obtener_detalle_hotel(request, hotel_id):
    ...
```

- **list_tasks(request):** Muestra una lista de todos los hoteles disponibles. Esta función obtiene todos los objetos del modelo 'Hotel' de la base de datos y los pasa a la plantilla `list_tasks.html` para ser renderizados. Se suele usar para mostrar una lista de hoteles en una página web.
- **createHotel(request):** Permite crear un nuevo hotel. La función verifica que la solicitud sea de tipo POST y que contenga todos los campos requeridos para crear un nuevo objeto 'Hotel'. Si falta algún campo, devuelve un error. Si todos los campos están presentes, crea el nuevo objeto y lo guarda en la base de datos. Después, redirige al usuario a la página principal.
- **deleteHotel(request, hotel_id):** Elimina un hotel basado en su identificador único (`hotel_id`). Este método usa el identificador para obtener el hotel a eliminar y, después de eliminarlo, redirige a la página principal. Se usa para eliminar un registro de la base de datos.

- **editHotelRender(request, hotel_id)**: Muestra el formulario para editar un hotel. La función obtiene el objeto 'Hotel' por su identificador único y lo pasa a la plantilla `edicionHotel.html`, donde se renderiza un formulario para editar sus detalles. Este método es útil para mostrar el formulario de edición.
- **editHotel(request, hotel_id)**: Actualiza la información de un hotel existente. Similar al anterior, pero en este caso, la función actualiza el objeto 'Hotel' con los datos de una solicitud POST. Luego, guarda los cambios en la base de datos y redirige a la página principal.
- **readHotelRender(request, hotel_id)**: Renderiza una vista de detalles para un hotel específico. Toma el identificador del hotel y lo obtiene de la base de datos. Luego, pasa este objeto al contexto para renderizar la plantilla `leerHotel.html`, mostrando la información del hotel.
- **readHotel(request, hotel_id)**: Además de leer los detalles de un hotel, esta función también puede actualizarlo si se recibe una solicitud POST. Si es POST, actualiza el hotel con la información recibida y guarda los cambios en la base de datos. Si no es POST, devuelve un error.
- **obtener_detalle_hotel(request, hotel_id)**: Devuelve los detalles de un hotel en formato JSON. Esta función obtiene el hotel por su identificador único y estructura los datos para ser devueltos como respuesta JSON. Se usa para proveer datos en formato API o para aplicaciones que requieran comunicación con JSON.

V. EQUIVALENCIA CON CONSULTAS SQL

Se muestran las equivalencias en el lenguaje SQL de los métodos anteriores.

1. LIST_TASKS

Muestra todos los hoteles disponibles en la base de datos.

```
SELECT * FROM hotel;
```

2. CREATEHOTEL

Crea un nuevo hotel con datos proporcionados en una solicitud POST.

```
INSERT INTO hotel (  
    idhotel,  
    nombreestablecimiento,  
    horacheckin,  
    horacheckout,  
    petfriendly,
```

```
servicio,  
numerointerior,  
numeroexterior,  
colonia,  
calle,  
estado  
) VALUES (  
  'ID del hotel',  
  'Nombre del establecimiento',  
  'Hora de check-in',  
  'Hora de check-out',  
  'Pet-friendly',  
  'Servicio',  
  'Número interior',  
  'Número exterior',  
  'Colonia',  
  'Calle',  
  'Estado'  
) ;
```

3. DELETEHOTEL

Elimina un hotel dado su identificador único.

```
DELETE FROM hotel WHERE idhotel = 'ID del hotel';
```

4. EDITHOTELRENDER

Obtiene un objeto 'Hotel' por su identificador único para renderizar un formulario de edición.

```
SELECT * FROM hotel WHERE idhotel = 'ID del hotel';
```

5. EDITHOTEL

Actualiza un hotel existente con datos proporcionados en una solicitud POST.

```
UPDATE hotel  
SET  
  nombreestablecimiento = 'Nuevo nombre',  
  horacheckin = 'Nueva hora de check-in',  
  horacheckout = 'Nueva hora de check-out',  
  petfriendly = 'Nuevo pet-friendly',  
  servicio = 'Nuevo servicio',  
  numerointerior = 'Nuevo número interior',  
  numeroexterior = 'Nuevo número exterior',  
  colonia = 'Nueva colonia',  
  calle = 'Nueva calle',  
  estado = 'Nuevo estado'  
WHERE idhotel = 'ID del hotel';
```

6. READHOTELRENDER

Obtiene un hotel por su identificador único para mostrar detalles.

```
SELECT * FROM hotel WHERE idhotel = 'ID del hotel';
```

7. OBTENER_DETALLES_HOTEL

Obtiene detalles de un hotel por su identificador único y devuelve información en formato JSON.

```
SELECT
    idhotel,
    nombreestablecimiento,
    horacheckin,
    horacheckout,
    petfriendly,
    servicio,
    numerointerior,
    numeroexterior,
    colonia,
    calle,
    estado
FROM hotel
WHERE idhotel = 'ID del hotel';
```

VI. TASKS/TEMPLATES

Esta carpeta contiene las plantillas HTML para la interfaz gráfica del proyecto Django.

VII. RUTAS URL PARA LA APLICACIÓN DE GESTIÓN DE HOTELES

El siguiente bloque de código define las rutas URL para tu aplicación Django, asociando cada ruta con una función de vista correspondiente. Esto permite a Django mapear las solicitudes HTTP a las funciones que deben manejar esas solicitudes.

```
# Lista de rutas para la aplicación de gestión de hoteles
urlpatterns = [
    # Muestra la página principal con todos los hoteles
```

```
path('', list_tasks, name="list_tasks"),

# Ruta para crear un nuevo hotel mediante una solicitud POST
path('new/', createHotel, name="createHotel"),

# Ruta para eliminar un hotel existente por su ID
path('delete_hotel/<str:hotel_id>', deleteHotel, name='deleteHotel'),

# Ruta para mostrar el formulario de edición para un hotel específico
path('edit_hotel_Render/<str:hotel_id>', editHotelRender, name="editHotelRender"),

# Ruta para procesar la actualización de un hotel por su ID mediante POST
path('edit_hotel/<str:hotel_id>', editHotel, name="editHotel"),

# Ruta para mostrar los detalles de un hotel específico
path('read_hotel_Render/<str:hotel_id>', readHotelRender, name="readHotelRender"),

# Ruta para editar o ver los detalles de un hotel, según el contexto
path('read_hotel/<str:hotel_id>', readHotel, name="readHotel"),

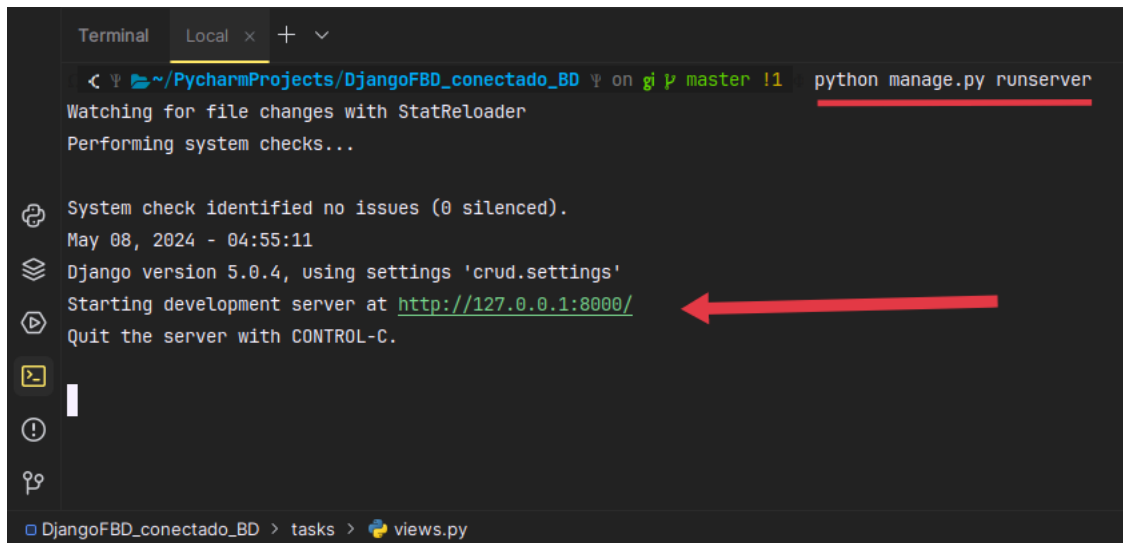
# Ruta para obtener detalles de un hotel en formato JSON
path('read_hotel/<int:hotel_id>', obtener_detalle_hotel, name='obtener_detalle_hotel'),
]
```

VIII. EJECUCIÓN

Se ejecuta desde la terminal de Pycharm el comando:

```
python manage.py runserver
```

Se hace click en el link que aparece, tal como muestra la figura 3.

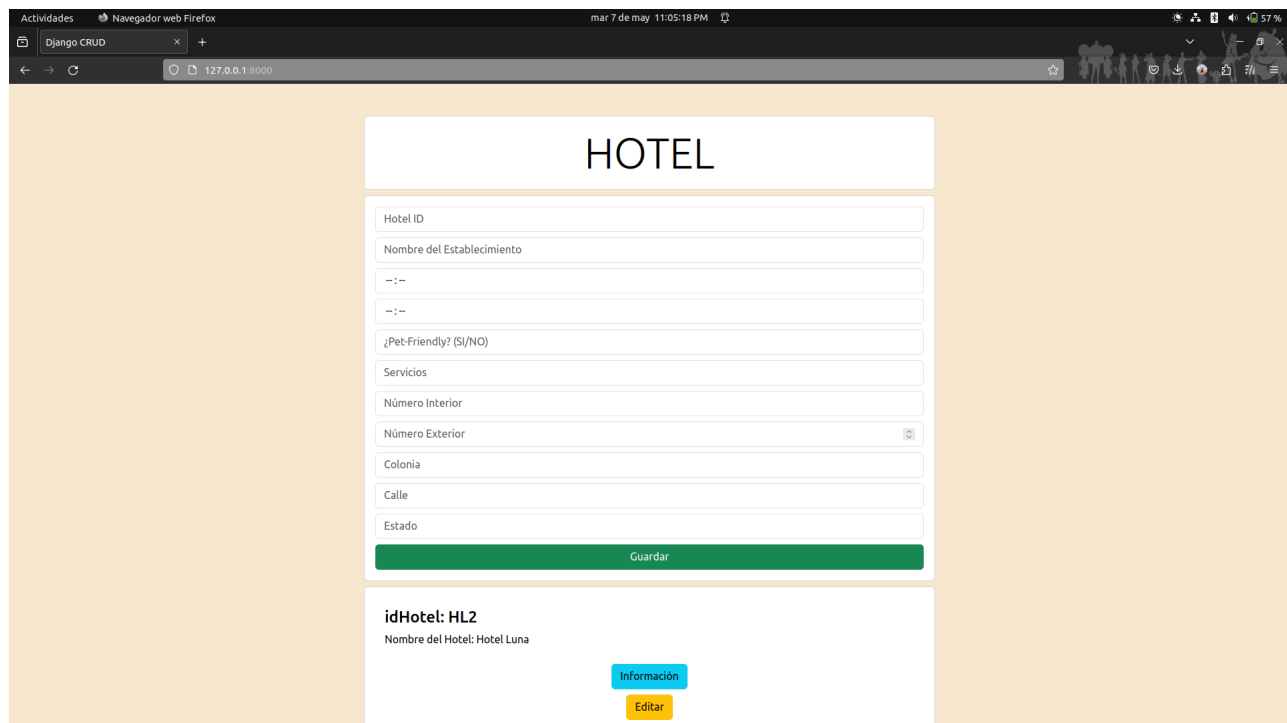


```
Terminal Local x + v
~/PycharmProjects/DjangoFBD_conectado_BD on gi p master !1 python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 08, 2024 - 04:55:11
Django version 5.0.4, using settings 'crud.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Figura 3: Ejecución del proyecto

Si todo se ha ejecutado de manera adecuada, se mostrará la siguiente pantalla:



Actividades Navegador web Firefox mar 7 de may 11:05:18 PM 57%

Django CRUD x +

127.0.0.1:8000

HOTEL

Hotel ID

Nombre del Establecimiento

--:--

--:--

¿Pet-Friendly? (Si/NO)

Servicios

Número Interior

Número Exterior

Colonia

Calle

Estado

Guardar

idHotel: HL2

Nombre del Hotel: Hotel Luna

Información

Editar

Figura 4: Interfaz inicial del proyecto.

Si se quiere agregar un hotel es necesario llenar los campos que están presentes y posteriormente oprimir el botón **Guardar**, el cuarto y quinto campo hacen referencia a las horas **CheckIn** y **CheckOut** respectivamente.

Por otra lado, se muestran los hoteles que están dentro de la base de datos, junto a tres botones para realizar las operaciones CRUD; el botón de **Información** sirve para poder ver la información del hotel solicitado.

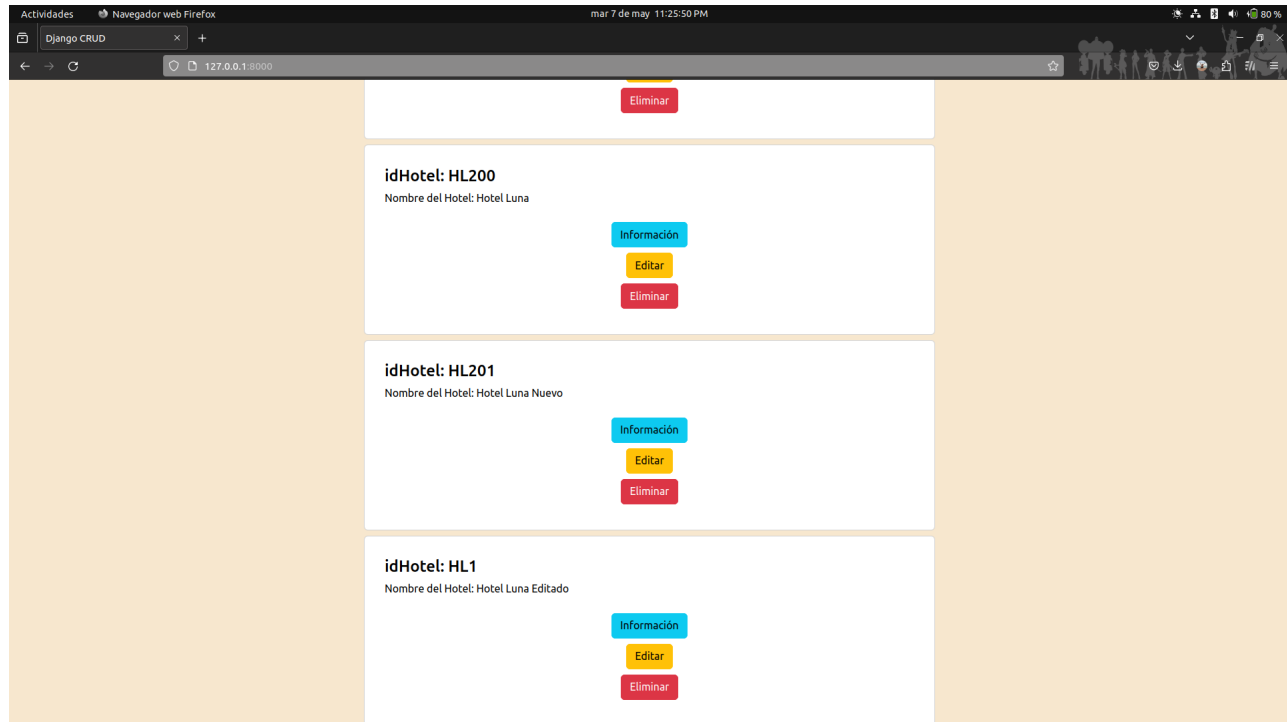


Figura 5: En este caso particular, se muestra el hotel con `idHotel = 201` pues éste se añadió para probar la operación de guardar hotel. Se muestra el hotel con `idHotel = 1` en este apartado final pues se hizo una modificación para probar la operación editar, sin embargo, por defecto, se agrega al final el hotel que ha sido modificado, en éste se editó el nombre.

Los cambios que se realizan se ven reflejados en la base de datos. Lo propio ocurre si se elimina algún elemento.

Actividades Terminal

mar 7 de may 11:34:56 PM

sudo docker exec -it 84bb69d19615 psql -U postgres

HL181	Hotel Luna	15:00:00	13:00:00	SI	Salones de eventos, Gimnasio, salón de belleza, piscina	327	Hybrid Oak
HL182	Sage	15:00:00	13:00:00	SI	Bar, Salones de eventos, Tienda de regalos, Gimnasio	2	Munz's Bedstraw
HL183	Division	15:00:00	13:00:00	NO	Bar, Salones de eventos, Tienda de regalos, Gimnasio	15	Plagiobryum Moss
HL184	Hotel Luna	15:00:00	13:00:00	NO	Bar, Salones de eventos, Tienda de regalos, Gimnasio	4871	Grama
HL185	Del Sol	15:00:00	13:00:00	NO	Bar, Salones de eventos, Tienda de regalos, Gimnasio	59	Latin Thorn-apple
HL186	Rieder	15:00:00	13:00:00	SI	Lavandería, Bar, Salones de eventos, Gimnasio	6	False Boneset
HL187	Monument	15:00:00	13:00:00	SI	Salones de eventos, Gimnasio, salón de belleza, piscina	8367	Prickly Phlox
HL188	Hallows	15:00:00	13:00:00	SI	Salones de eventos, Gimnasio, salón de belleza, piscina	8	Mountain Misery
HL189	Erie	15:00:00	13:00:00	SI	Lavandería, Bar, Salones de eventos, Gimnasio	68780	Twogrooved Milkvetch
HL190	Shopko	15:00:00	13:00:00	NO	Lavandería, Bar, Salones de eventos, Gimnasio	14	Lackschewitz's Milkv
HL191	South	15:00:00	13:00:00	NO	Lavandería, Bar, Salones de eventos, Gimnasio	740	3
HL192	Hintze	15:00:00	13:00:00	NO	Bar, Salones de eventos, Tienda de regalos, Gimnasio	8187	8
HL193	Little Fleur	15:00:00	13:00:00	SI	Bar, Salones de eventos, Tienda de regalos, Gimnasio	2294	Grand Canyon Blazing
HL194	Kropf	15:00:00	13:00:00	SI	Lavandería, Bar, Salones de eventos, Gimnasio	31	38723
HL195	Sundown	15:00:00	13:00:00	SI	Bar, Salones de eventos, Tienda de regalos, Gimnasio	975	Oregon White Oak
HL196	Eastlawn	15:00:00	13:00:00	SI	Salones de eventos, Gimnasio, salón de belleza, piscina	0676	81
HL197	Merry	15:00:00	13:00:00	NO	Salones de eventos, Gimnasio, salón de belleza, piscina	44	216
HL198	Truax	15:00:00	13:00:00	NO	Bar, Salones de eventos, Tienda de regalos, Gimnasio	6	Lopsided Rush
HL199	Porter	15:00:00	13:00:00	SI	Bar, Salones de eventos, Tienda de regalos, Gimnasio	33369	Tropical Royalblue W
HL200	Scofield	15:00:00	13:00:00	SI	Lavandería, Bar, Salones de eventos, Gimnasio	1741	1354
HL201	Independence	13:00:00	14:00:00	SI	Alberca	15	16
HL1	Hotel Luna Editado	15:00:00	13:00:00	NO	Bar, Salones de eventos, Tienda de regalos, Gimnasio	None	8

(201 rows)

(END)

Figura 6: Los cambios o agregados que se hacen, se ven reflejados en la base de datos.

II. POSTMAN

Después de descargar Postman se hace la primera solicitud tal como lo muestra la siguiente imagen:

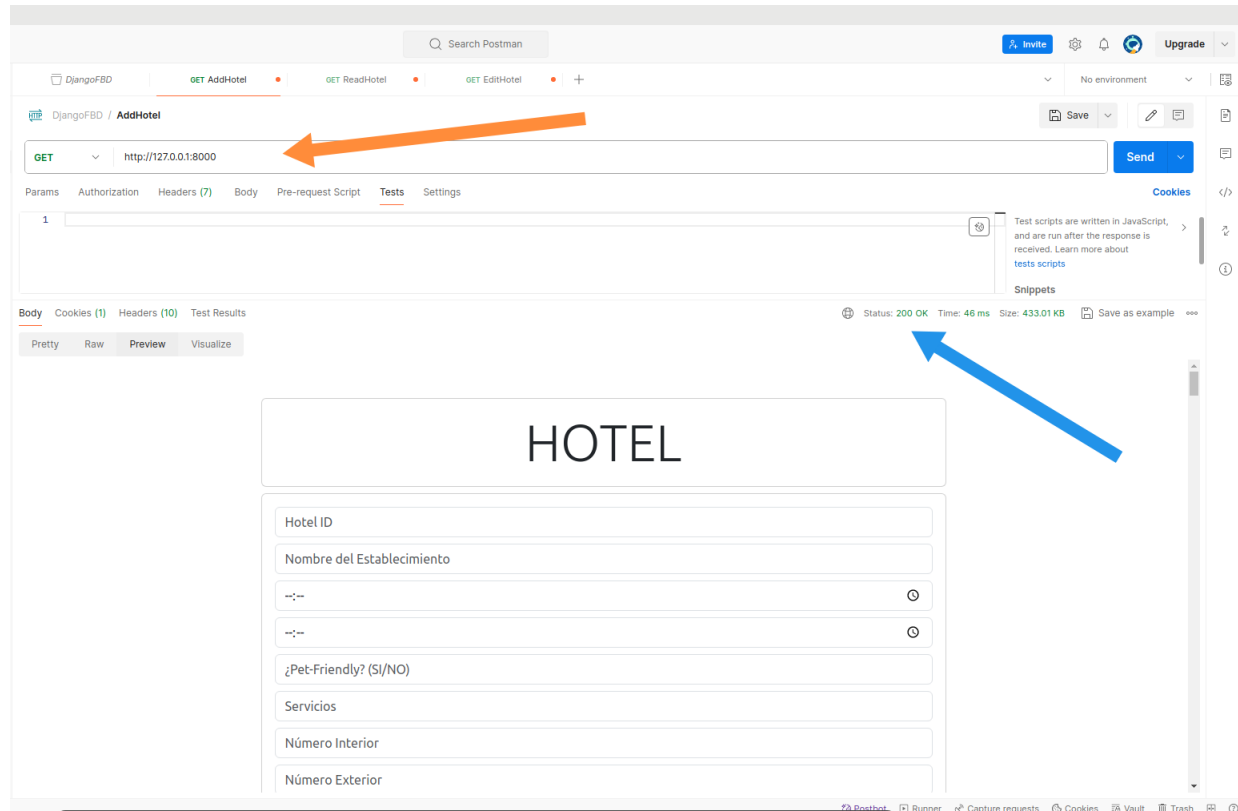


Figura 7: Solicitud a Postman

En la flecha de color naranja se agregó el URL de la petición, mientras que la flecha de color azul indica que la respuesta del servidor tuvo un resultado exitoso.

El código de estado HTTP 200 es un estándar para indicar que la solicitud realizada al servidor fue recibida, comprendida y procesada correctamente.

Se hace la prueba de un test para comprobar que la conexión fue exitosa.

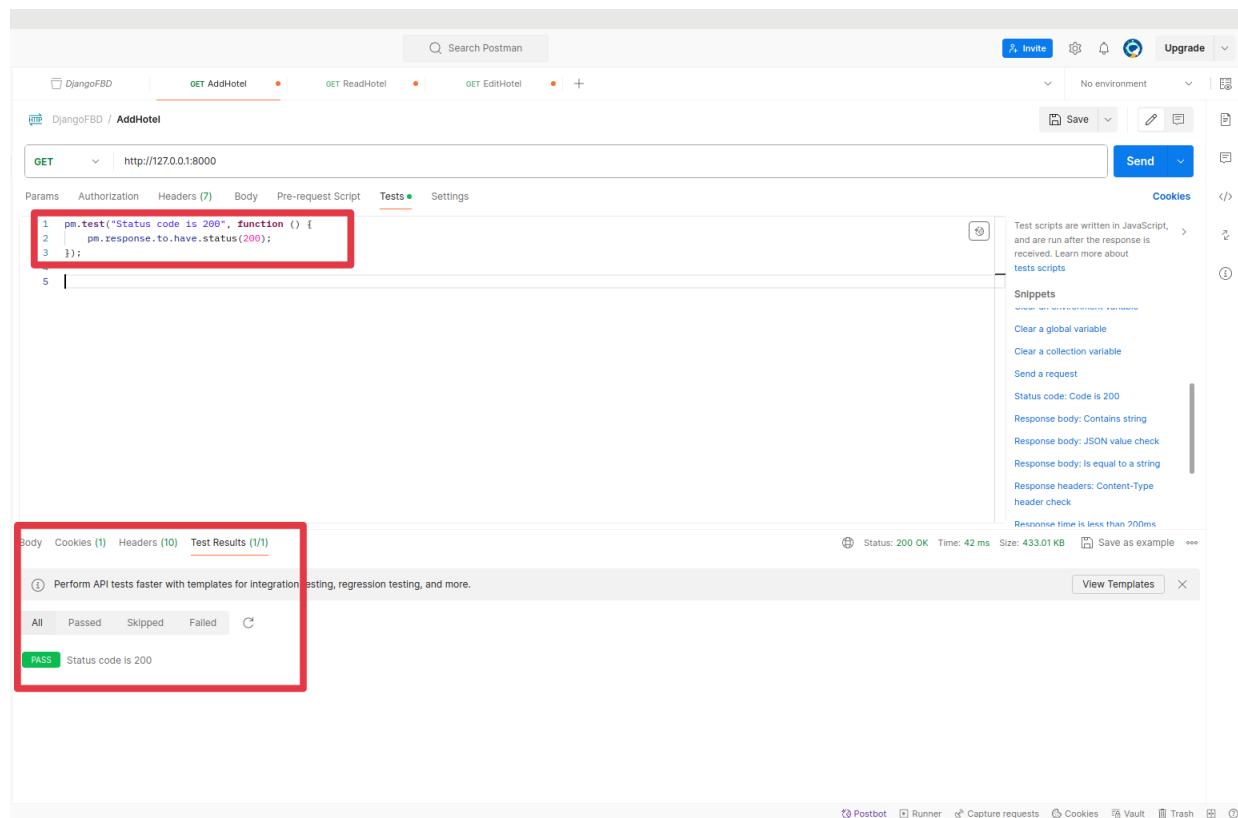


Figura 8: Solicitud a Postman; operación añadir

El proceso anterior se repitió pero para los casos de leer y editar, dando el mismo resultado de conectividad.

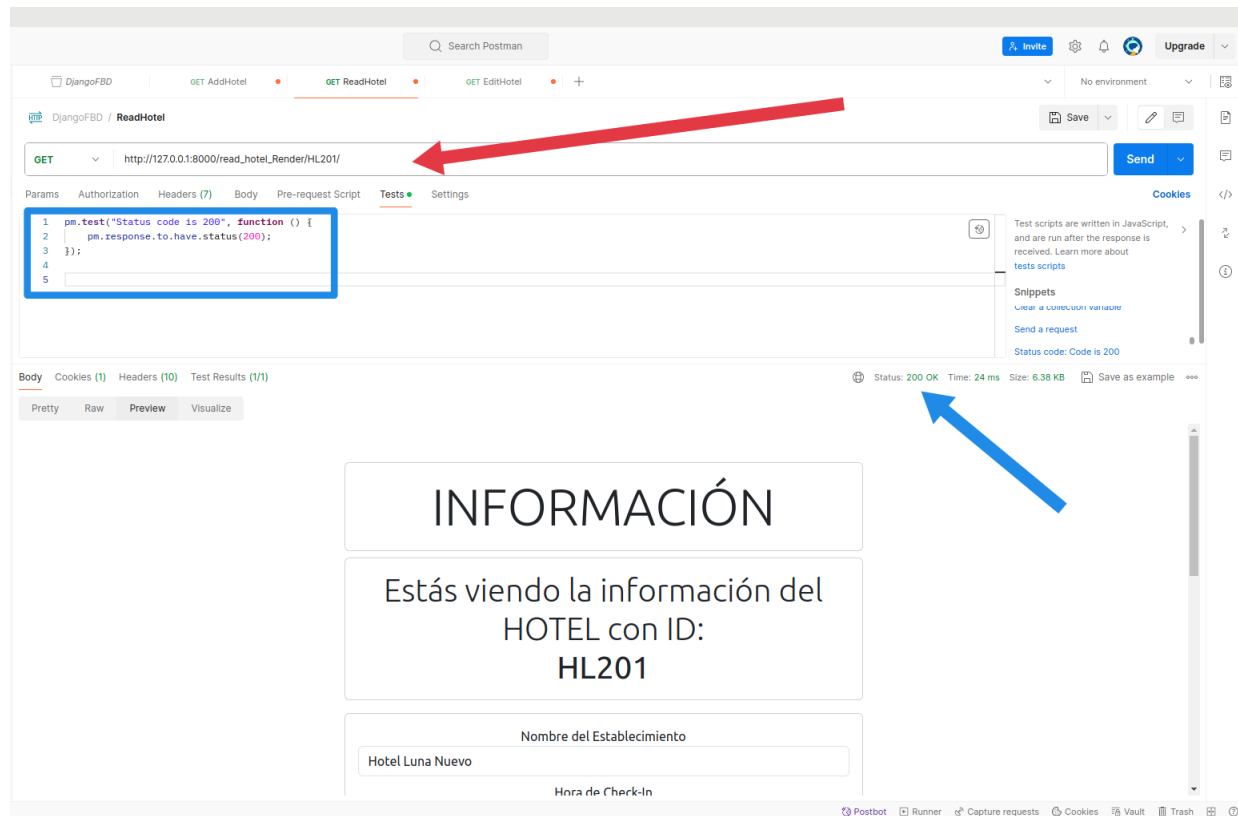


Figura 9: Solicitud a Postman; operación leer.

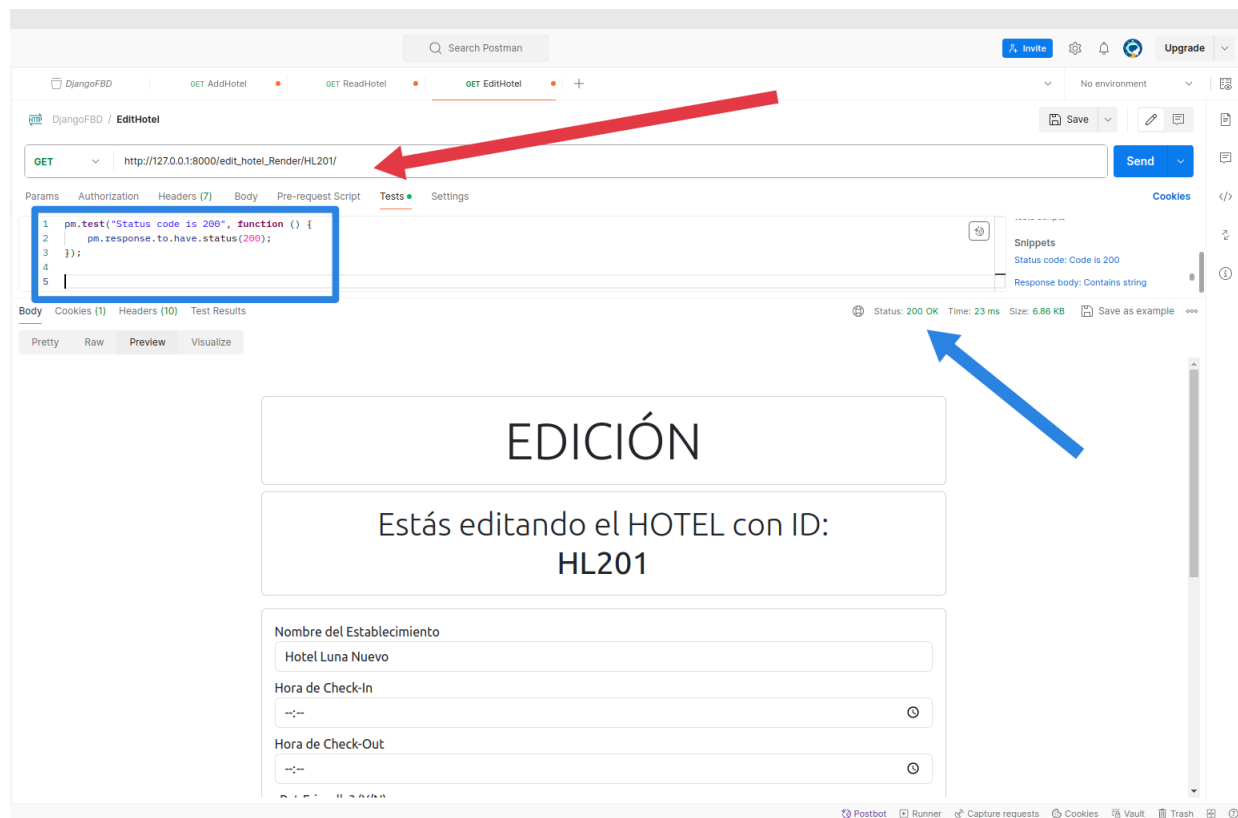


Figura 10: Solicitud a Postman; operación editar.

JDBC

- Para esta Practica deberan realizar un programa que permita realizar las operaciones CRUD sobre la base de datos que estamos realizando en el curso, para ello deberan utilizar JDBC para la comunicacion entre el programa y la base de datos. Deberan escoger al menos 2 tablas de su base de datos, para esto. Su programa debera implementar el insertar, eliminar y actualizar para cada una de las tablas, ademas de poder hacer una consulta que nos permita obtener todos los valores de una tabla y una consulta para obtener un solo valor de la tabla dado el identificador.

Actividades:

1. Deberan realizar un reporte en formato pdf, donde definan cuales fueron las queries que tuvieron que utilizar para la realizacion de CRUD para cada una de sus tablas y explicar lo que realiza la query, ademas de las sentencias que usa, este documento lo llamen Practica08.pdf.

Metodos utilizados para las operaciones CRUD

```
public interface Repositorio<T, K>
```

```
//Metodo para insertar un elemento.
public void insertar(T h);
//Metodo para modificar un elemento
public void modificar(T h);
//Metodo para eliminar un elemento.
public void eliminar(T h);
//Metodo para obtenerTodos los elementos.
public List<T>obtenerTodos();
//Metodo para obtener un elemento.
public T obtener(K id);
//Metodo para ejecutar la modificacion.
public void EjecutarModificacion(T h);
```

Vamos implementar esta interfaz para las tablas Huesped y Hotel de nuestra base de datos.

Implementación para la tabla Hotel:

- **public List<Hotel>obtenerTodos() :**
Para este metodo utilizamos la consulta: SELECT * FROM hotel;
Esta consulta obtiene todas las tuplas de la tabla hotel.
Por lo que el metodo devuelve todas las tuplas de la tabla hotel en una lista.
- **public Hotel obtener(String idHotel):**
Para este metodo utilizamos la consulta: SELECT * FROM hotel WHERE (idHotel=:idHotel);
Esta consulta obtiene una tupla de la tabla hotel apartir de su id.
Por lo que el metodo devuelve un elemento de la clase Hotel con toda la informacion de la tupla obtenida.
- **public void eliminar(Hotel h):**
Para este metodo utilizamos la consulta:DELETE FROM hotel WHERE (idHotel=:idHotel);
Esta consulta elimina un elemento de la tabla hotel apartir de su id.
Por lo que el metodo elimina un elemento de la tabla hotel apartir de su id obtenido del objeto hotel que le pasamos como argumento.

- **public void modificar(Hotel h):**

Para este metodo utilizamos la consulta:

```
UPDATE hotel SET idHotel=:idHotel,nombreEstablecimiento=:nombreEstablecimiento,
horaCheckIn=:horaCheckIn,horaCheckOut=:horaCheckout,petFriendly=:petFriendly,
servicio=:servicio,numeroInterior=:numeroInterior,numeroExterior=:numeroExterior,
colonia=:colonia,calle=:calle,estado=:estado WHERE (idHotel=:idHotel);
```

Esta consulta modifica los atributos de una tupla existente en la tabla hotel.

Por lo que el método modifica los atributos de una tupla obtenida partir de su id, ese id es obtenido por el objeto hotel que le pasamos por argumento.

- **public void insertar(Hotel h):**

Para este metodo utilizamos la consulta: INSERT INTO hotel (idHotel,nombreEstablecimiento, horaCheckIn, horaCheckOut, petFriendly, servicio, numeroInterior, numeroExterior, colonia, calle, estado) VALUES (:idHotel,:nombreEstablecimiento, :horaCheckIn, :horaCheckOut, :petFriendly, :servicio, :numeroInterior, :numeroExterior, :colonia, :calle, :estado);

Es consulta inserta una tupla en la tabla hotel.

Por lo que el metodo inserta en la tabla hotel una tupla con la información del objeto Hotel que le pasamos por argumento.

Implementación para la tabla Huesped:

- **public List<Huesped>obtenerTodos() :**
Para este metodo utilizamos la consulta: `SELECT * FROM huesped;`
Esta consulta obtiene todas las tuplas de la tabla huesped.
Por lo que el metodo devuelve todas las tuplas de la tabla huesped en una lista.
- **public Huesped obtener(String idP):**
Para este metodo utilizamos la consulta: `SELECT * FROM huesped WHERE (idP=:idP);`
Esta consulta obtiene una tupla de la tabla huesped apartir de su id.
Por lo que el metodo devuelve un elemento de la clase Huesped con toda la informacion de la tupla obtenida.
- **public void eliminar(Huesped h):**
Para este metodo utilizamos la consulta: `DELETE FROM huesped WHERE (idP=:idP);`
Esta consulta elimina un elemento de la tabla huesped apartir de su id.
Por lo que el metodo elimina un elemento de la tabla huesped apartir de su id obtenido del objeto huesped que le pasamos como argumento.
- **public void modificar(Huesped h):**
Para este metodo utilizamos la consulta:
`UPDATE huesped SET idHotel=:idHotel, nombre=:nombre, paterno=:paterno, materno=:materno, fechaNacimiento=:fechaNacimiento, genero=:genero,nacionalidad=:nacionalidad, idFormaEfectivo=:idFormaEfectivo, idFormaTarjeta=:idFormaTarjeta WHERE (idP=:idP);`
Esta consulta modifica los atributos de una tupla existente en la tabla huesped.
Por lo que el método modifica los atributos de una tupla obtenida partir de su id, ese id es obtenido por el objeto Huesped que le pasamos por argumento.
- **public void insertar(Huesped h):**
Para este metodo utilizamos la consulta: `INSERT INTO huesped (idPersona,idHotel, nombre, paterno, materno, fechaNacimiento, genero, nacionalidad, idFormaEfectivo, idFormaTarjeta) VALUES (:idPersona, :idHotel, :nombre, :paterno, :materno, :fechaNacimiento, :genero, :nacionalidad, :idFormaEfectivo, :idFormaTarjeta);`
Es consulta inserta una tupla en la tabla huesped.
Por lo que el metodo inserta en la tabla huesped una tupla con la información del objeto huesped que le pasamos por argumento.

- Utilizaran POSTMAN para probar su aplicación REST, para cada una de las consultas que realizaron. Las evidencias las deberán incluir en el reporte.

POSTMAN de nuestra aplicación para tabla hotel:

■ obtener todas las tuplas de la tabla hotel:

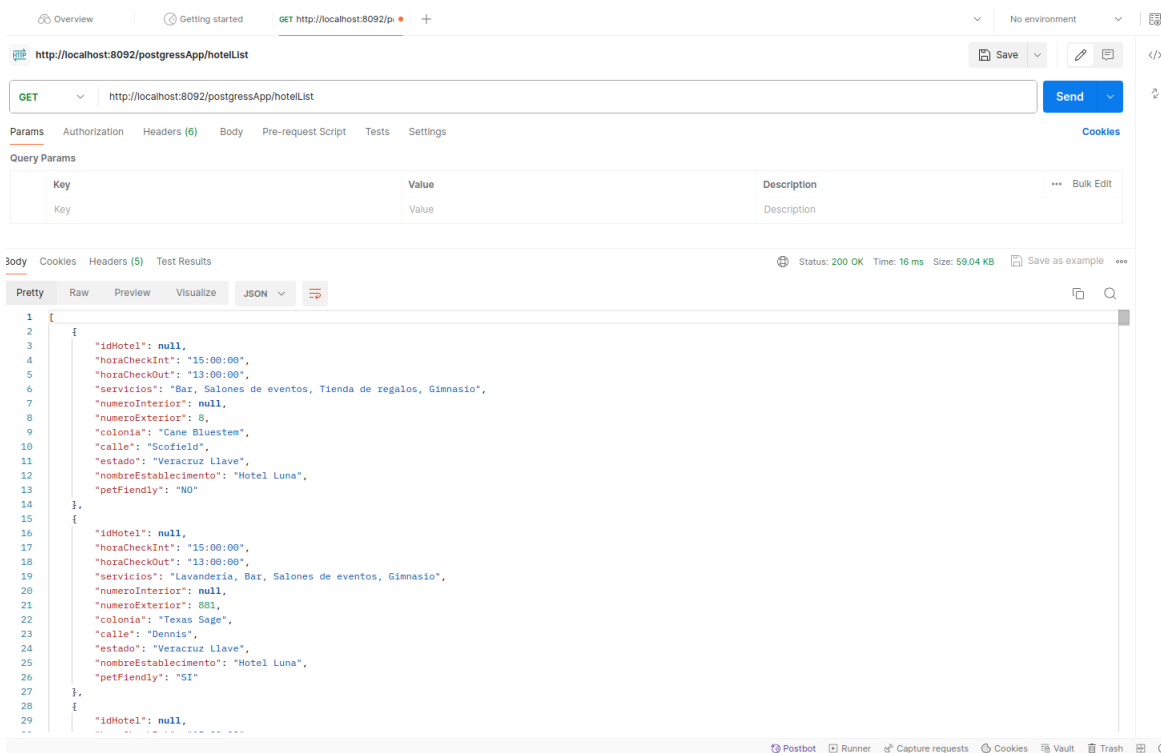


Figura 11: Solicitud a Postman; operación consultar.

■ obtener una tupla de la tabla hotel

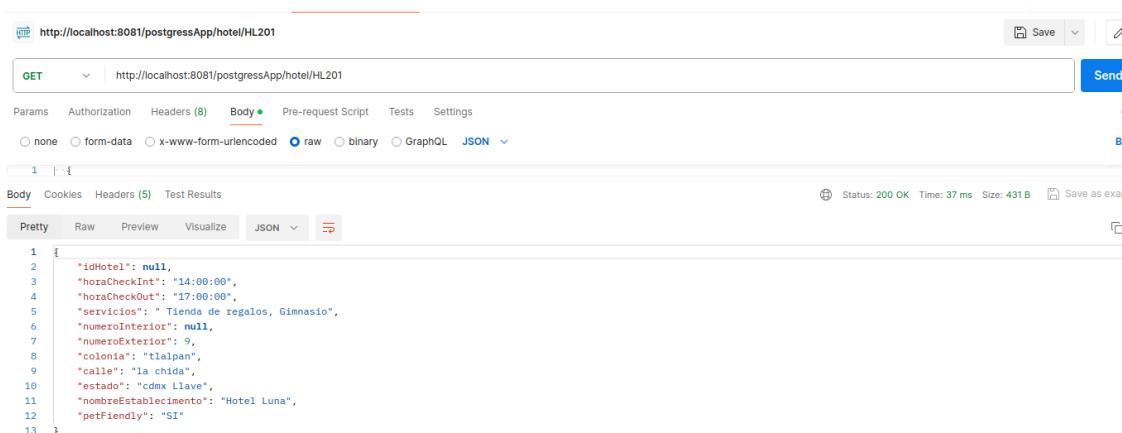


Figura 12: Solicitud a Postman; operación consultar.

■ eliminar una tupla de la tabla hotel:

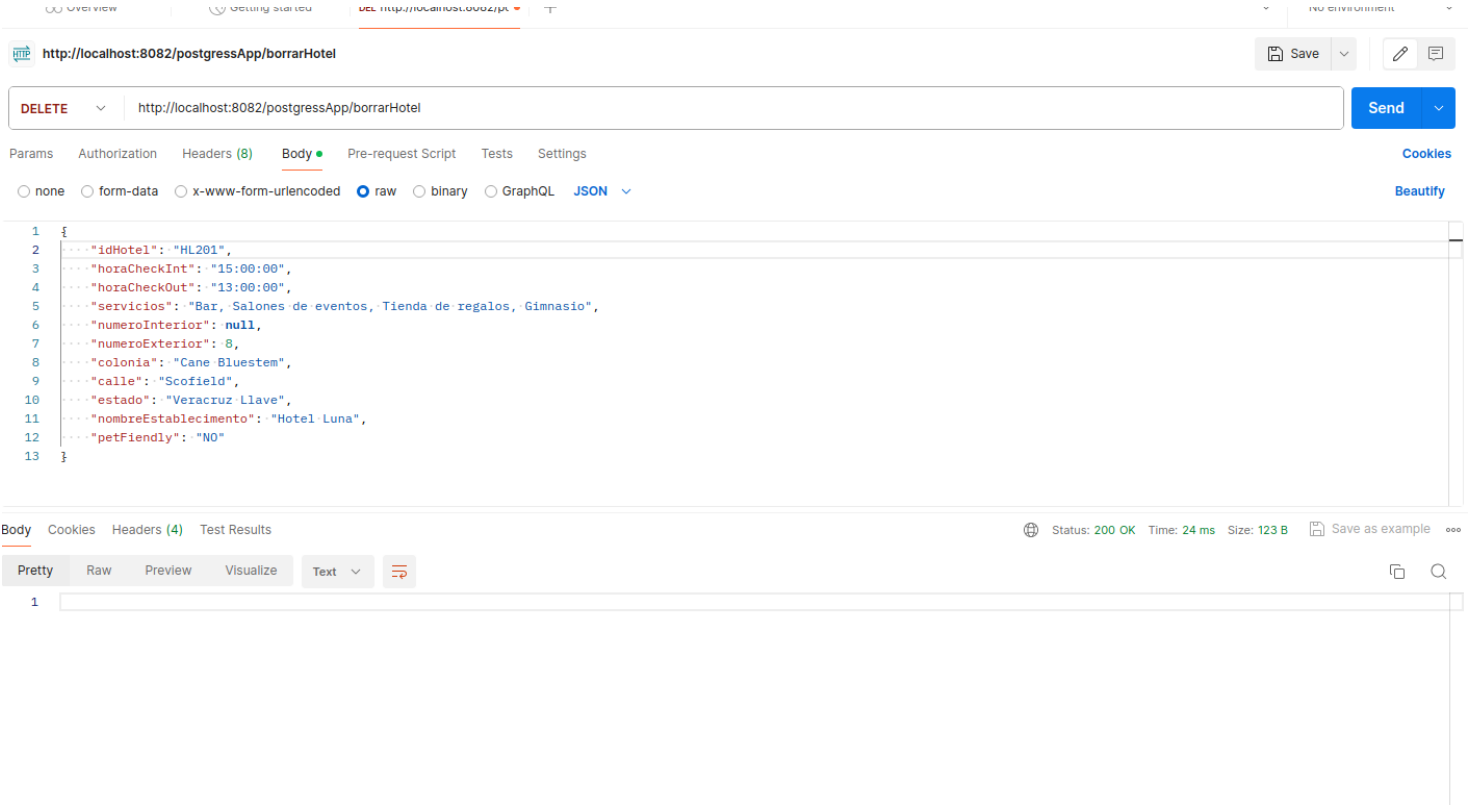


Figura 13: Solicitud a Postman; para borrar un hotel con el id HL201.

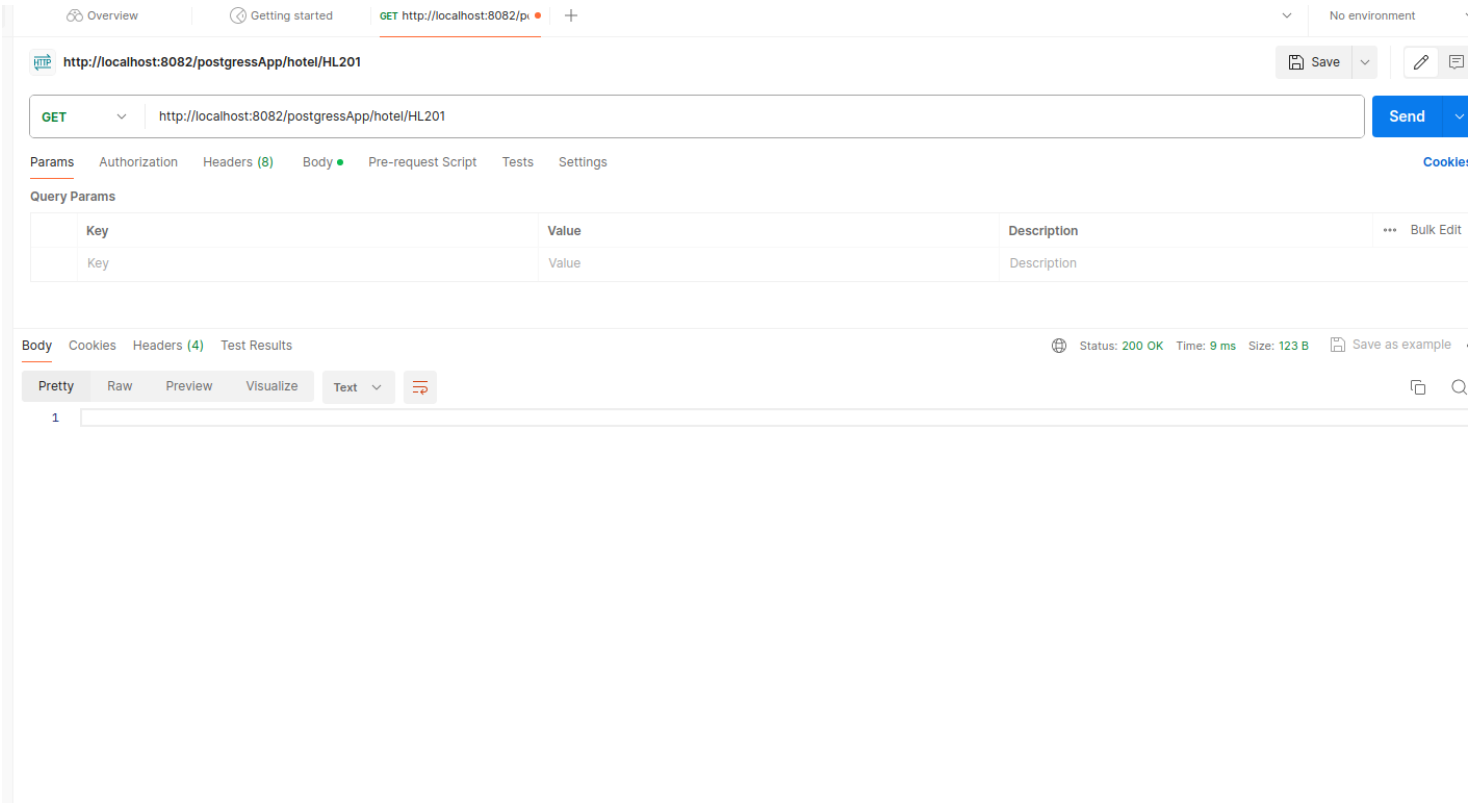


Figura 14: buscamos el elemento, pero ya no esta disponible

■ modificar una tupla de la tabla hotel:

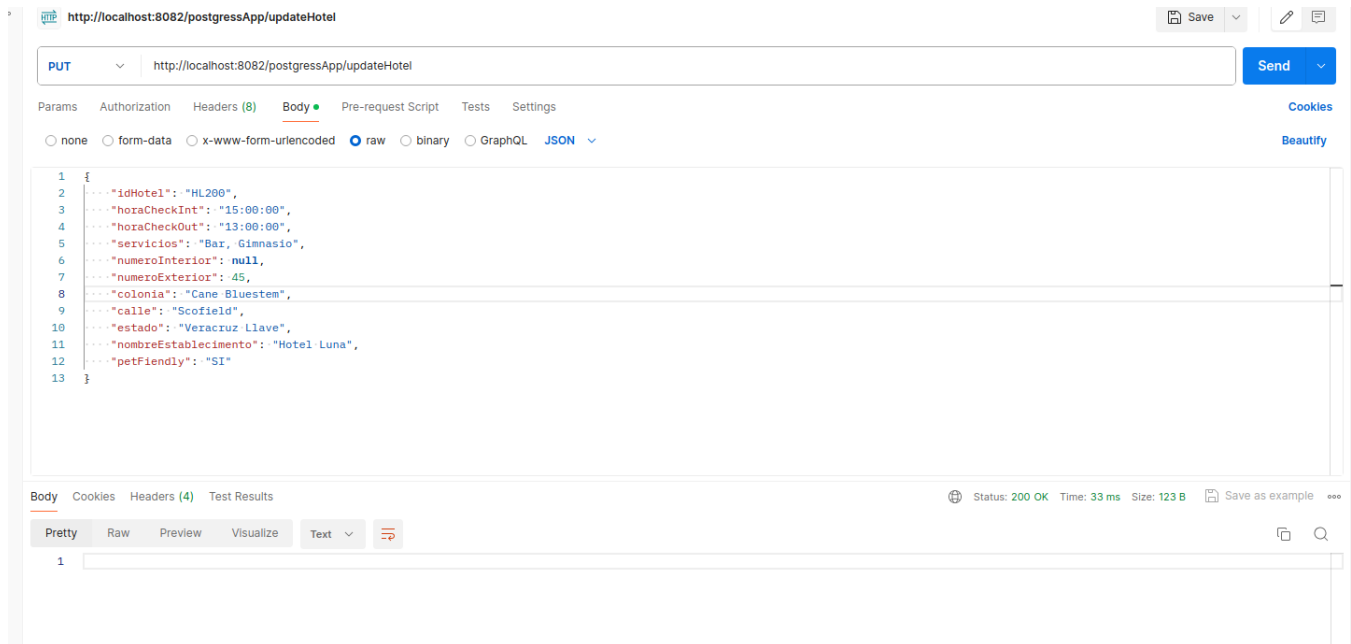


Figura 15: construimos un hotel con los atributos que deseamos modificar

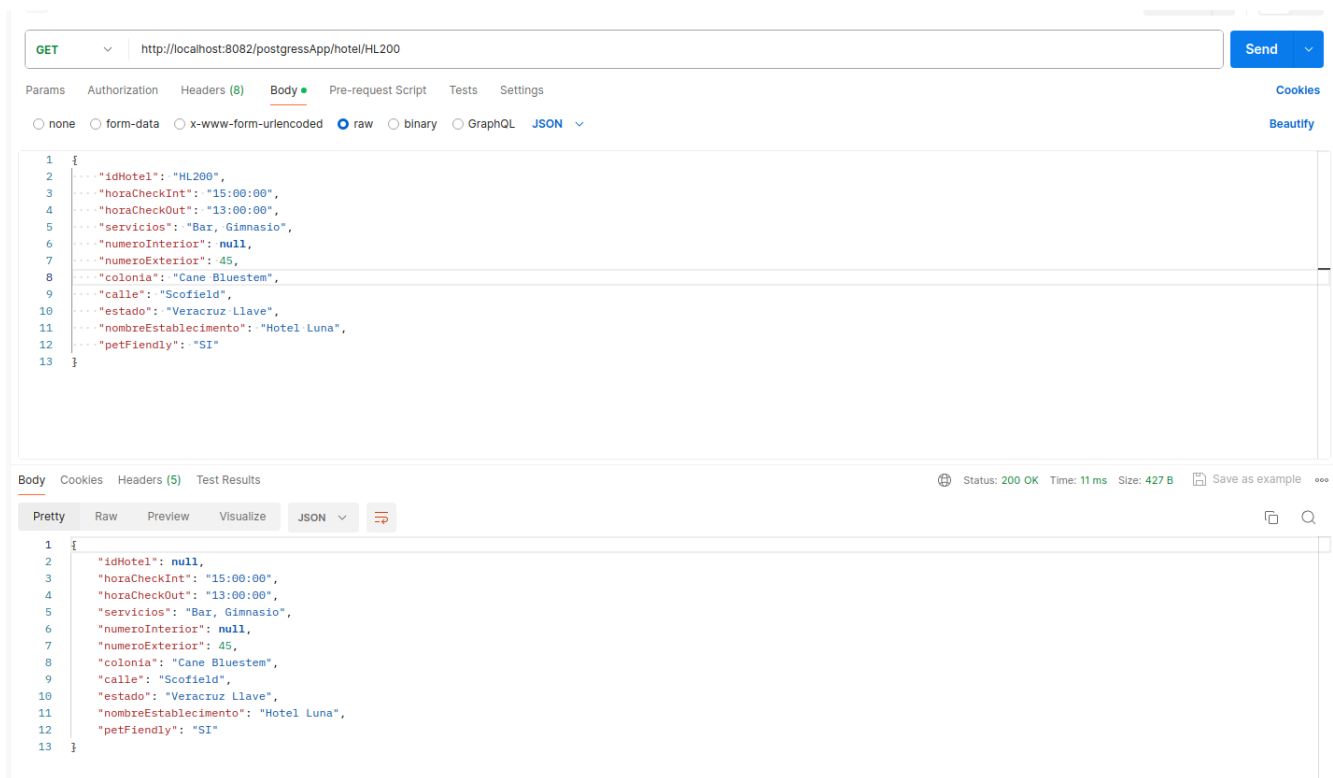


Figura 16: le hemos modificado los atributos petfriendly,servicios y numero exterior al hotel con el id HL200.

- insertar una tupla en la tabla hotel:

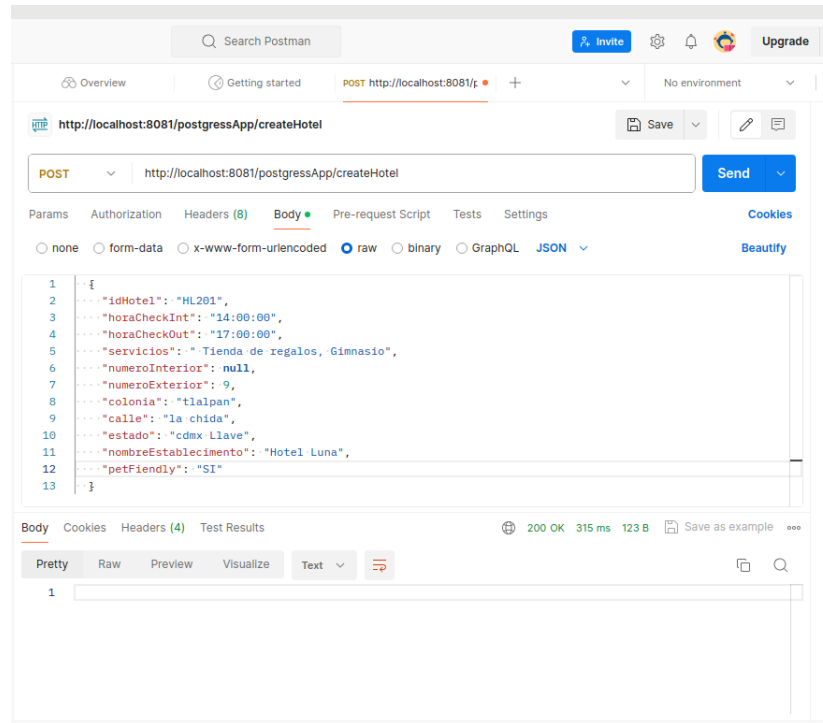


Figura 17: Insertamos un hotel con el id HL201

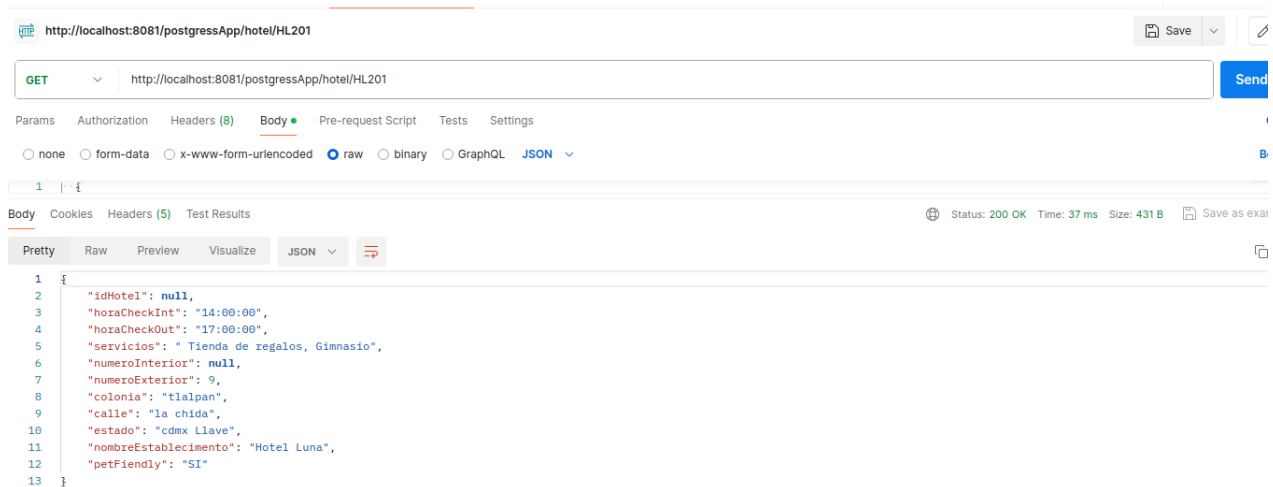


Figura 18: si buscamos por id HL201 veremos que obtenemos el hotel insertado

POSTMAN de nuestra aplicación para tabla huesped:

■ obtener todas las tuplas de la tabla huesped:

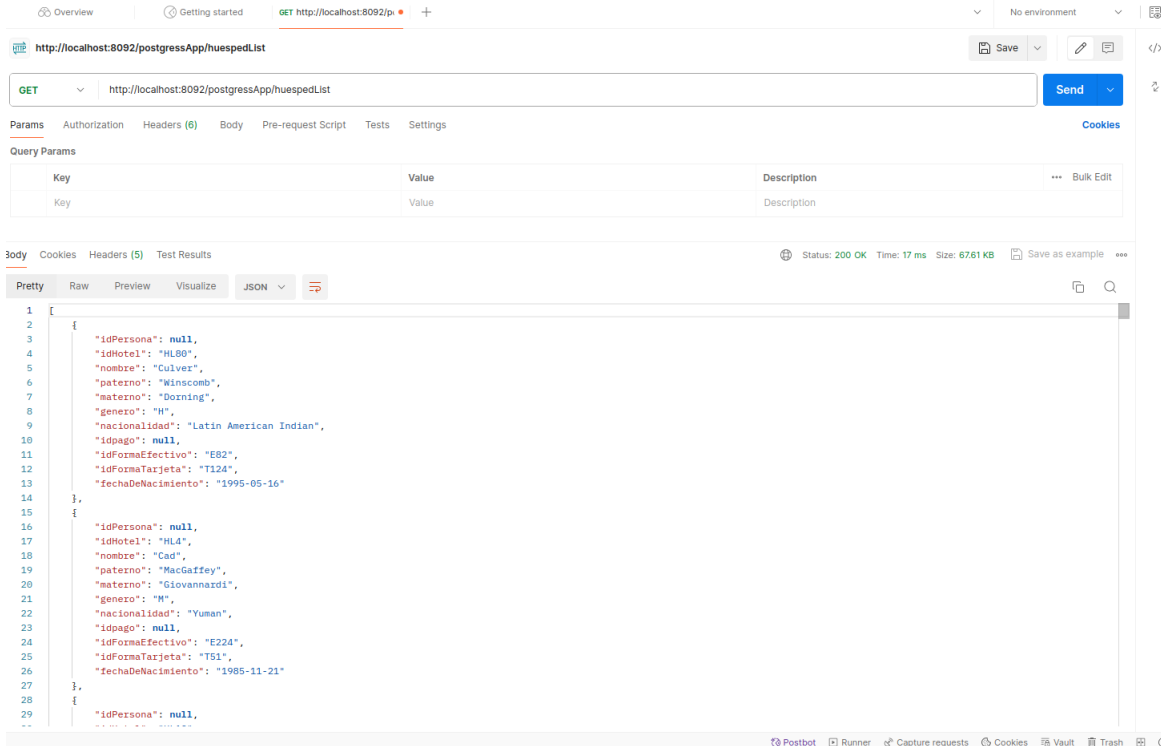


Figura 19: Solicitud a Postman; operación consultar.

■ obtener una tupla de la tabla huesped

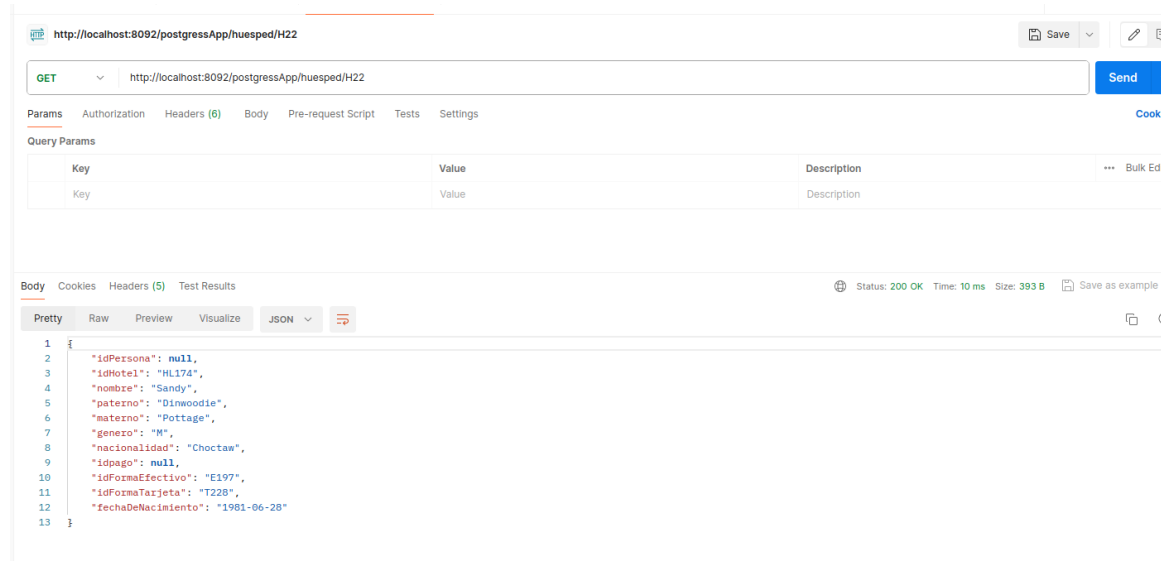


Figura 20: Solicitud a Postman; operación consultar.

■ eliminar una tupla de la tabla huesped:

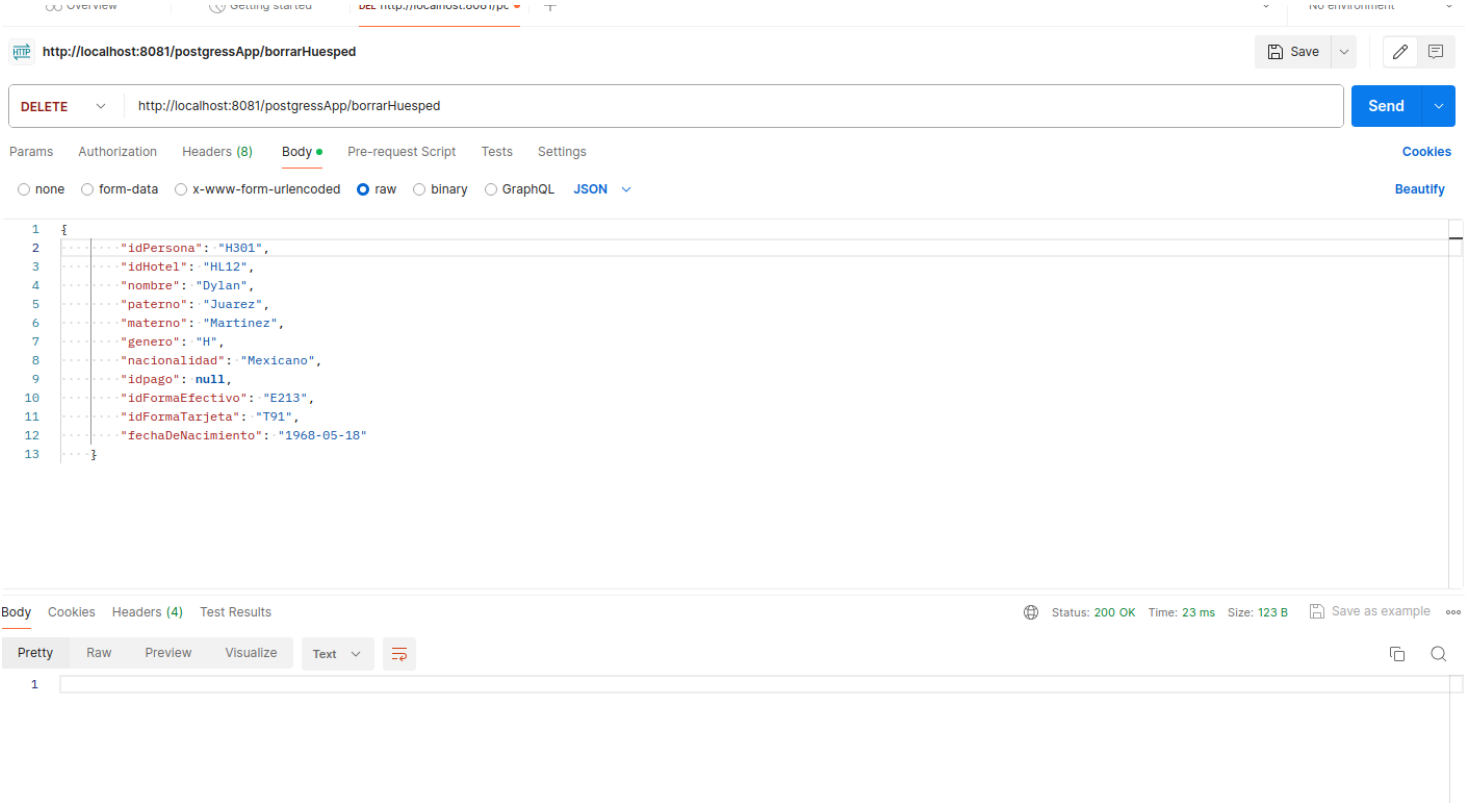


Figura 21: Solicitud a Postman; para borrar un huesped con el id H301.

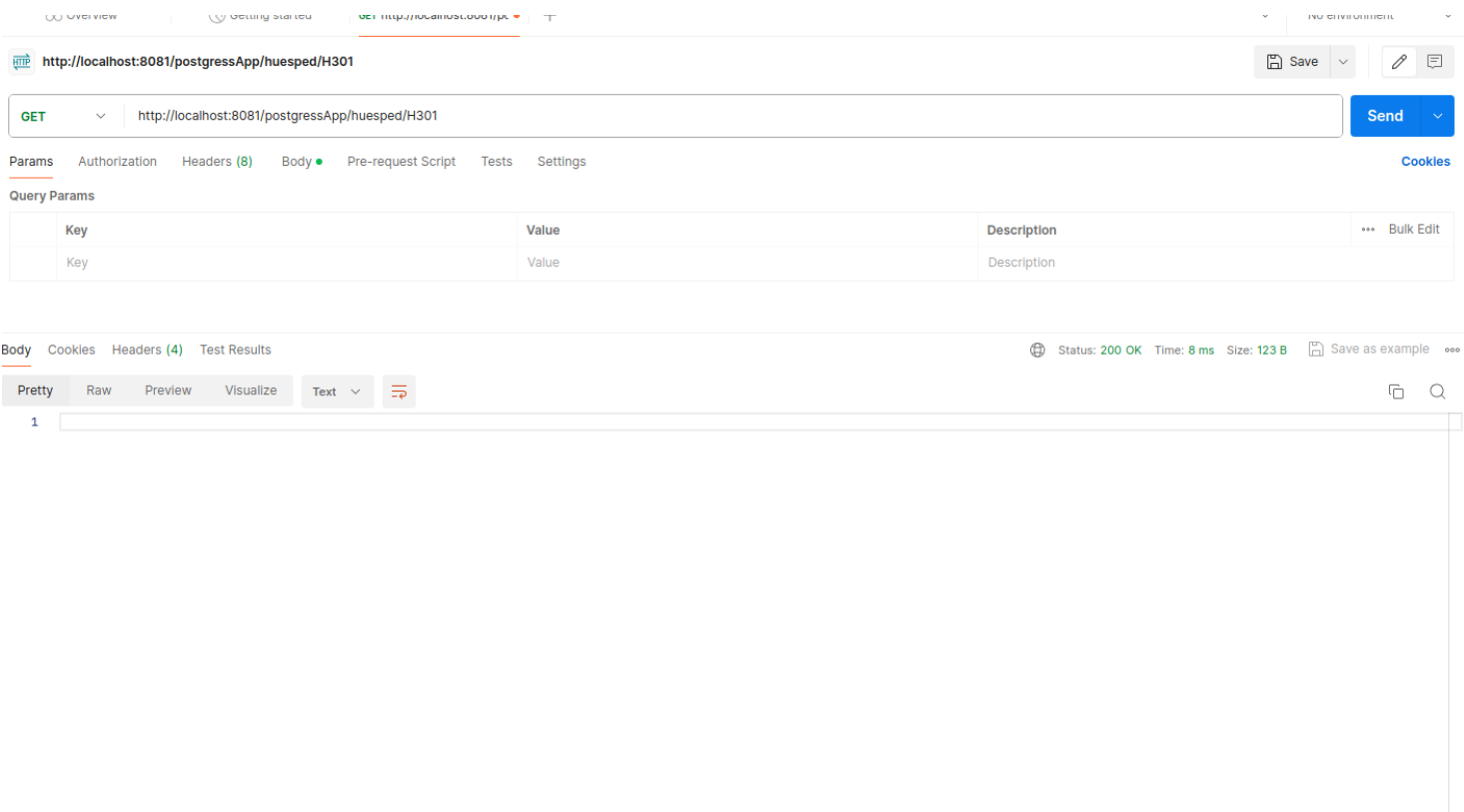


Figura 22: buscamos el elemento, pero ya no esta disponible

■ modificar una tupla de la tabla huesped:

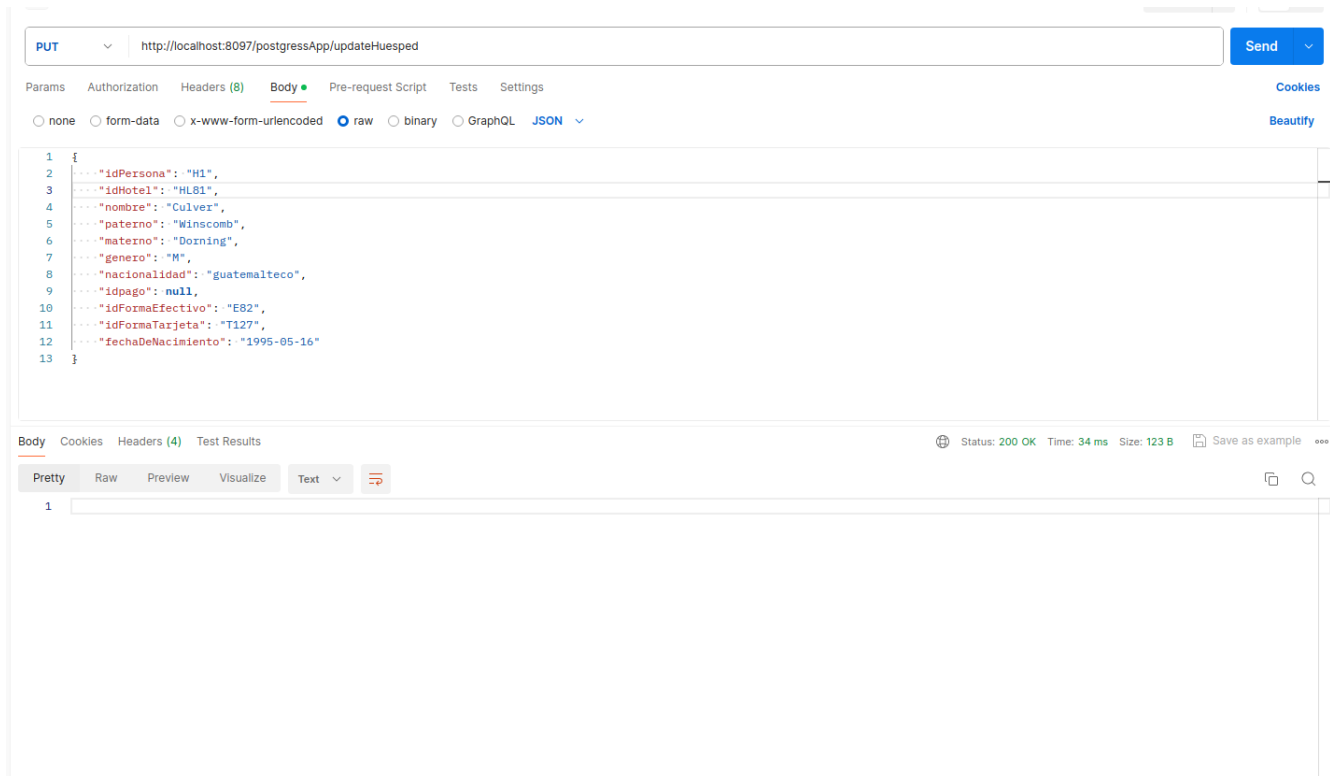


Figura 23: construimos un huesped con los atributos que deseamos modificar

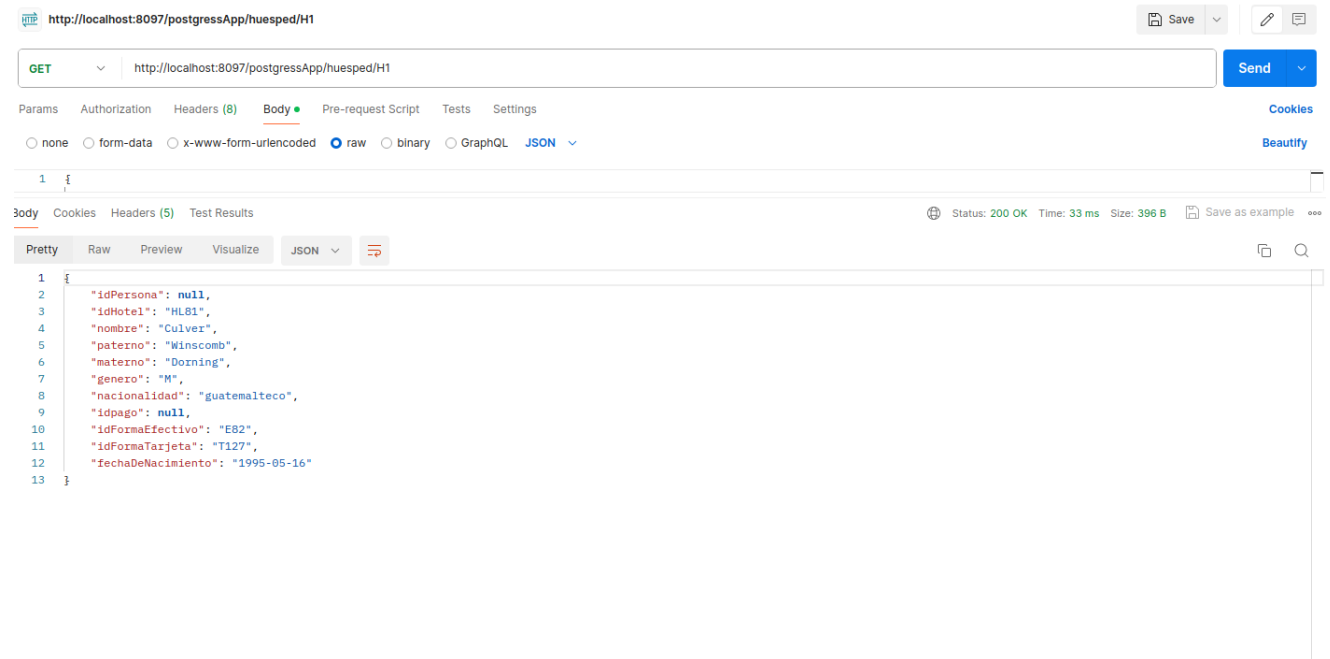


Figura 24: le hemos modificado los atributos genero y nacionalidad al huesped con el id H1.

- insertar una tupla en la tabla huesped:

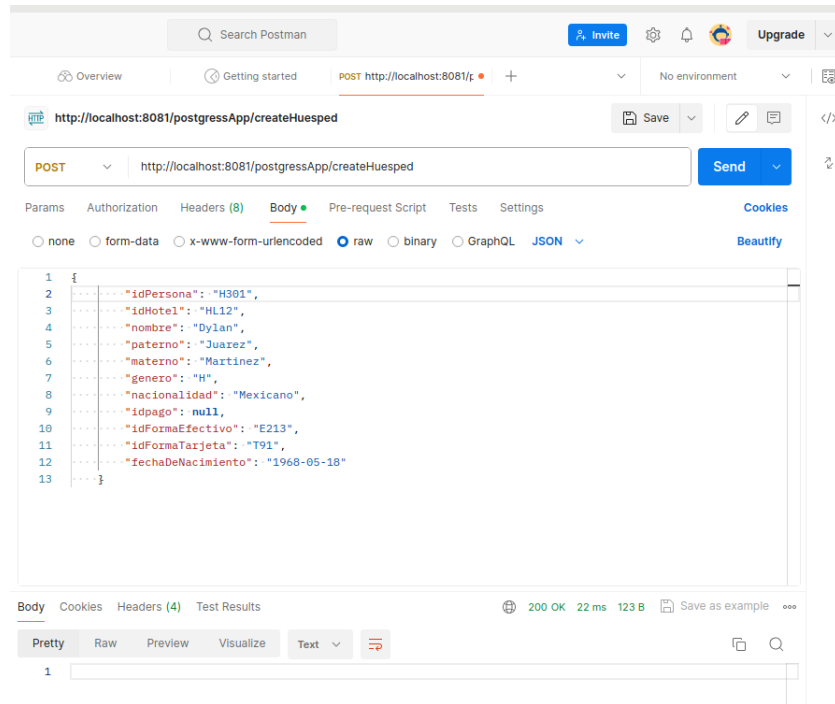


Figura 25: creamos un huesped con los atributos necesarios y lo insertamos

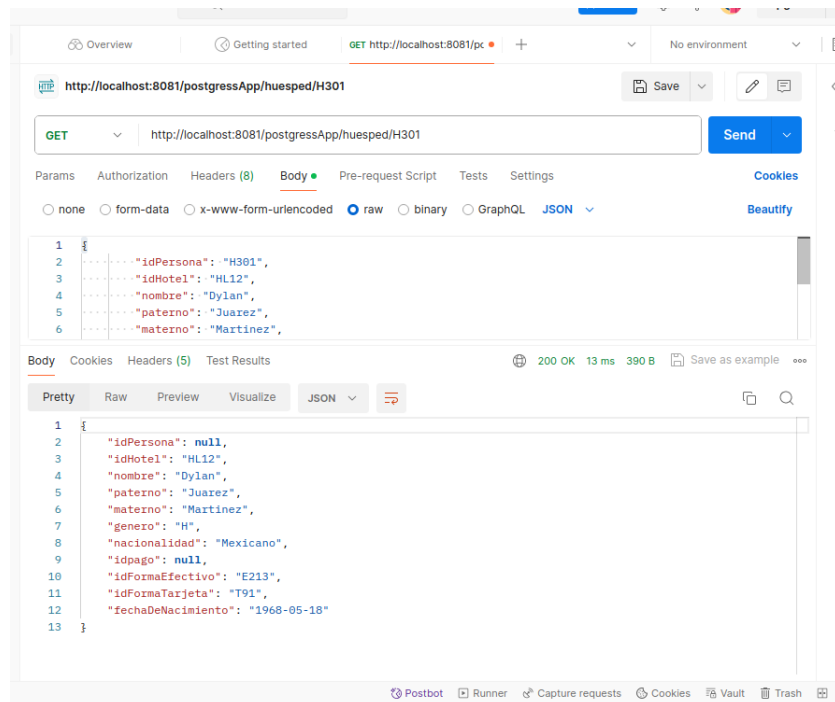


Figura 26: buscamos al huesped con el id del huesped insertado