TECNOLOGICO NACIONAL DE MÉXICO

CAMPUES NUEVO LAREDO

INGENIERÍA EN SISTEMAS COMPUTACIONALES

PROGRAMACIÓN MULTIPARADIGMA

Docente: Ing. Luis Daniel Castillo García

Equipo No. 1

Integrantes

Andrade Domínguez Jesús Alfonso 18100149

Alemán Pérez Jordan Alejandro 18100145

Castro Salazar Cesar 18100157

03 de noviembre a 2023, Nuevo Laredo, Tamps.

INDICE

Práctica 1 (PostgresSQL)

1. Al menos 3 entidades (ejemplo clase de entidad Persona)

Customer y supplier tienen las mismas propiedades por eso se utilizó solamente una clase para las dos.

```
from logger_base import log
import datetime

class Customer_Supplier:
    def __init__(self, name = None, adress = None, phonenumber = None,
status = None, createat = None, updateat = None) -> None:
        self._name = name
        self._adress = adress
        self._phonenumber = phonenumber
        self._status = status
        self._create_at = createat
        self._update_at = updateat

    def __str__(self) -> str:
        return f"""
    Nombre:{self._name} Dirección: {self._adress}
    No. Telefono:{self._phonenumber} Estatus:{self._status}
    """

    @property
    def id(self):
        return self._id
    @id.setter
    def id(self,id_customer):
        self._id = id_customer

    @property
    def name(self):
        return self._name
    @name.setter
    def name(self,name):
        self._name = name

    @property
    def adress(self):
        return self._adress
    @adress.setter
    def lastname(self,address):
        self._adress = address
```

```
    @property
    def phonenumber(self):
        return self._phonenumber
    @phonenumber.setter
    def phonenumber(self,phone):
        self._phonenumber = phone

    @property
    def status(self):
        return self._status
    @status.setter
    def status(self,status):
        self._status = status

    @property
    def create_at(self):
        return self._create_at
    @create_at.setter
    def create_at(self,create_at):
        self._create_at = create_at

    @property
    def update_at(self):
        return self._update_at
    @update_at.setter
    def update_at(self,update_at):
        self._update_at = update_at
```

Entidad Employee

```
from logger_base import log
import datetime

class Employee:
    def __init__(self, employe_number = None, firstname = None, lastname =
None, position = None, status = None, createat = None, updateat = None) ->
None:
        self._employeenumber = employe_number
        self._firstname = firstname
        self._lastname = lastname
        self._position = position
        self._status = status
        self._create_at = createat
        self._update_at = updateat
```

```python
    def __str__(self) -> str:
        return f"""
    No. Empleado:{self._employeenumber} Nombre: {self._firstname}
    Apellido:{self._lastname} Puesto:{self._position} Estatus:{self._status}
    """

    @property
    def id(self):
        return self._id
    @id.setter
    def id(self,id_employee):
        self._id = id_employee

    @property
    def employeenumber(self):
        return self._employeenumber
    @employeenumber.setter
    def employeenumber(self,employeenumber):
        self._employeenumber = employeenumber

    @property
    def firstname(self):
        return self._firstname
    @firstname.setter
    def firstname(self,firstname):
        self._firstname = firstname

    @property
    def lastname(self):
        return self._lastname
    @lastname.setter
    def lastname(self,lastname):
        self._lastname = lastname

    @property
    def position(self):
        return self._position
    @position.setter
    def position(self,position):
        self._position = position

    @property
    def status(self):
        return self._status
```

```
    @status.setter
    def status(self,status):
        self._status = status

    @property
    def create_at(self):
        return self._create_at
    @create_at.setter
    def create_at(self,create_at):
        self._create_at = create_at

    @property
    def update_at(self):
        return self._update_at
    @update_at.setter
    def update_at(self,update_at):
        self._update_at = update_at
```

2. Realizar CRUD de las 3 entidades

CRUD Customer

```
from customers_suppliers import Customer_Supplier
from cursor_of_the_pool import Cursor_of_the_pool
from logger_base import log
import datetime

class CustomerDAO:
    _SELECT = "SELECT * From customers WHERE status=true ORDER BY id"
    _INSERT = "INSERT INTO customers(name, adress, phonenumber, status,
create_at, update_at) VALUES(%s,%s,%s,%s,%s,%s)"
    _UPDATE = "UPDATE customers SET name=%s, adress=%s, phonenumber=%s,
status=%s, update_at=%s WHERE id=%s"
    _DELETE = "UPDATE customers SET status=%s WHERE id=%s" # Eliminación
logica
    # _DELETE = "DELETE FROM employees WHERE id=%s"

    @classmethod
    def getAll(cls):
        with Cursor_of_the_pool() as cursor:
            cursor.execute(cls._SELECT)
            registros = cursor.fetchall()
            customers = []
            for r in registros:
                customers.append(Customer_Supplier(r[1], r[2], r[3], r[4]))
            return customers
```

```
    @classmethod
    def post(cls,customer:Customer_Supplier):
        with Cursor_of_the_pool() as cursor:
            values = (customer.name, customer.adress, customer.phonenumber,
customer.status, customer.create_at, customer.update_at)
            cursor.execute(cls._INSERT,values)
            return cursor.rowcount

    @classmethod
    def put(cls,customer:Customer_Supplier):
        customer.update_at = datetime.datetime.now()
        with Cursor_of_the_pool() as cursor:
            values = (customer.name, customer.adress, customer.phonenumber,
customer.status, customer.update_at, customer.id)
            cursor.execute(cls._UPDATE,values)
            return cursor.rowcount

    @classmethod
    def delete(cls,id_customer:int):
        with Cursor_of_the_pool() as cursor:
            value = (False, id_customer)
            cursor.execute(cls._DELETE,value)
            return cursor.rowcount
```

Supplier

```
from customers_suppliers import Customer_Supplier
from cursor_of_the_pool import Cursor_of_the_pool
from logger_base import log
import datetime

class SupplierDAO:
    _SELECT = "SELECT * From suppliers WHERE status=true ORDER BY id"
    _INSERT = "INSERT INTO suppliers(name, adress, phonenumber, status,
create_at, update_at) VALUES(%s,%s,%s,%s,%s,%s)"
    _UPDATE = "UPDATE suppliers SET name=%s, adress=%s, phonenumber=%s,
status=%s, update_at=%s WHERE id=%s"
    _DELETE = "UPDATE suppliers SET status=%s WHERE id=%s" # Eliminación
logica
    # _DELETE = "DELETE FROM employees WHERE id=%s"

    @classmethod
    def getAll(cls):
        with Cursor_of_the_pool() as cursor:
```

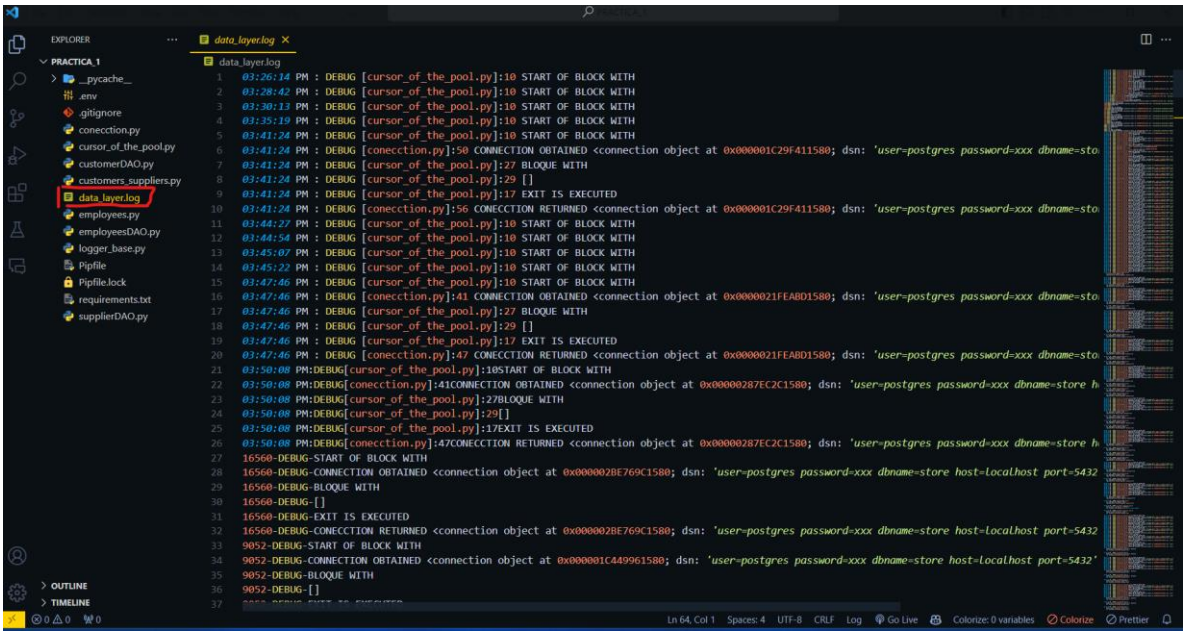
```
            cursor.execute(cls._SELECT)
            registros = cursor.fetchall()
            customers = []
            for r in registros:
                customers.append(Customer_Supplier(r[1], r[2], r[3], r[4]))
            return customers

    @classmethod
    def post(cls,supplier:Customer_Supplier):
        with Cursor_of_the_pool() as cursor:
            values = (supplier.name, supplier.adress, supplier.phonenumber,
supplier.status, supplier.create_at, supplier.update_at)
            cursor.execute(cls._INSERT,values)
            return cursor.rowcount

    @classmethod
    def put(cls,supplier:Customer_Supplier):
        supplier.update_at = datetime.datetime.now()
        with Cursor_of_the_pool() as cursor:
            values = (supplier.name, supplier.adress, supplier.phonenumber,
supplier.status, supplier.update_at, supplier.id)
            cursor.execute(cls._UPDATE,values)
            return cursor.rowcount

    @classmethod
    def delete(cls,id_supplier:int):
        with Cursor_of_the_pool() as cursor:
            value = (False, id_supplier)
            cursor.execute(cls._DELETE,value)
            return cursor.rowcount
```

Employee

```
from employees import Employee
from conecction import Conecction
from cursor_of_the_pool import Cursor_of_the_pool
from logger_base import log
import datetime

class EmployeeDAO:
    _SELECT = "SELECT * From employees WHERE status=true ORDER BY id"
    _INSERT = "INSERT INTO employees(employeenumber, firstname, lastname,
position, status, create_at, update_at) VALUES(%s,%s,%s,%s,%s,%s,%s)"
    _UPDATE = "UPDATE employees SET employeenumber=%s, firstname=%s,
lastname=%s, position=%s, status=%s, update_at=%s WHERE id=%s"
```

```
    _DELETE = "UPDATE employees SET status=%s WHERE id=%s" # Eliminación
logica
    # _DELETE = "DELETE FROM employees WHERE id=%s"

    @classmethod
    def getAll(cls):
        with Cursor_of_the_pool() as cursor:
            cursor.execute(cls._SELECT)
            registros = cursor.fetchall()
            employees = []
            for r in registros:
                employees.append(Employee(r[1], r[2], r[7], r[3], r[4]))
            return employees

    @classmethod
    def post(cls,employee:Employee):
        with Cursor_of_the_pool() as cursor:
            values = (employee.employeenumber, employee.firstname,
employee.lastname, employee.position, employee.status, employee.create_at,
employee.update_at)
            cursor.execute(cls._INSERT,values)
            return cursor.rowcount

    @classmethod
    def put(cls,employee:Employee):
        employee.update_at = datetime.datetime.now()
        with Cursor_of_the_pool() as cursor:
            values = (employee.employeenumber, employee.firstname,
employee.lastname, employee.position, employee.status, employee.update_at,
employee.id)
            cursor.execute(cls._UPDATE,values)
            return cursor.rowcount

    @classmethod
    def delete(cls,id_employee:int):
        with Cursor_of_the_pool() as cursor:
            value = (False, id_employee)
            cursor.execute(cls._DELETE,value)
            return cursor.rowcount
```

3. Usar archivo de logs



4. Utilizar pool de conexiones

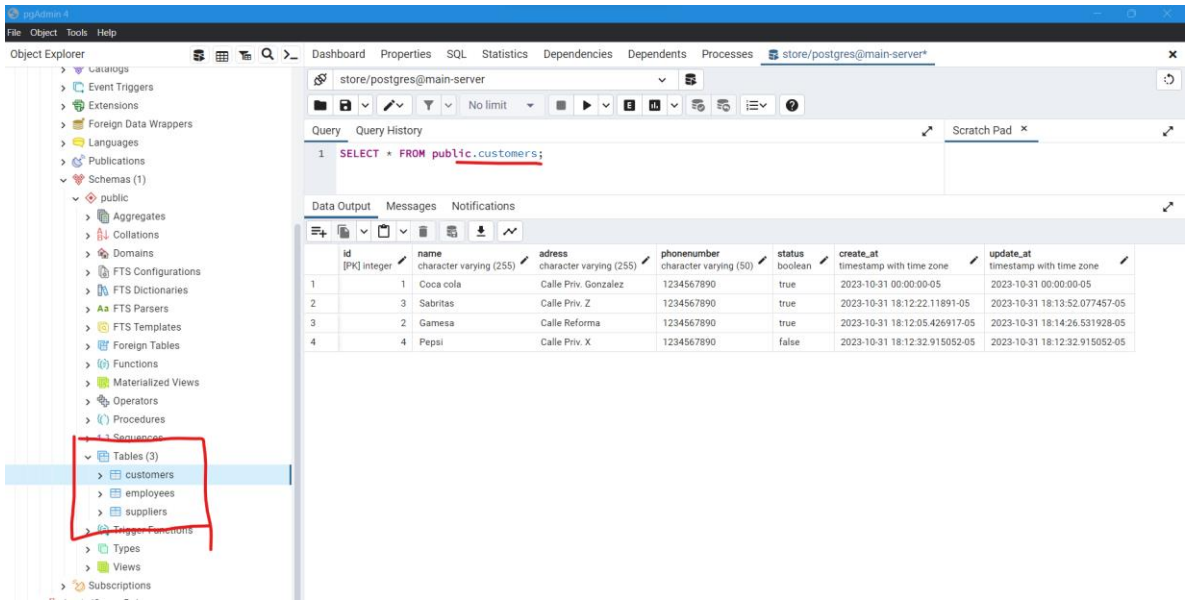
```python
from dotenv import load_dotenv
import os
from psycopg2 import pool
from logger_base import log

load_dotenv()

class Conecction:
    _DATABASE = os.getenv("DATABASE")
    _USERNAME = os.getenv("USERNAME")
    _PASSWORD = os.getenv("PASSWORD")
    _HOST = os.getenv("HOST")
    _PORT = os.getenv("PORT")
    _MIN_CON = 1
    _MAX_CON = 5
    _pool = None

    @classmethod
    def obtain_pool(cls):
        try:
            if cls._pool == None:
                cls._pool = pool.SimpleConnectionPool(
                    cls._MIN_CON,
                    cls._MAX_CON,
                    host = cls._HOST,
```

```
                    user = cls._USERNAME,
                    password = cls._PASSWORD,
                    port = cls._PORT,
                    database = cls._DATABASE
                )
                log.debug(f"CREATION OF THE POOL {pool}")
                return cls._pool
            else:
                return cls._pool
        except Exception as e :
            log.error(e)

    @classmethod
    def obtain_connection(cls):
        connection = cls.obtain_pool().getconn()
        log.debug(f"CONNECTION OBTAINED {connection}")
        return connection

    @classmethod
    def release_connection(cls, conexion):
        cls.obtain_pool().putconn(conexion)
        log.debug(f"CONECCTION RETURNED {conexion}")

    @classmethod
    def close_connections(cls):
        cls.obtain_pool().closeall()
        log.debug("CONECCTIONS CLOSED")
```

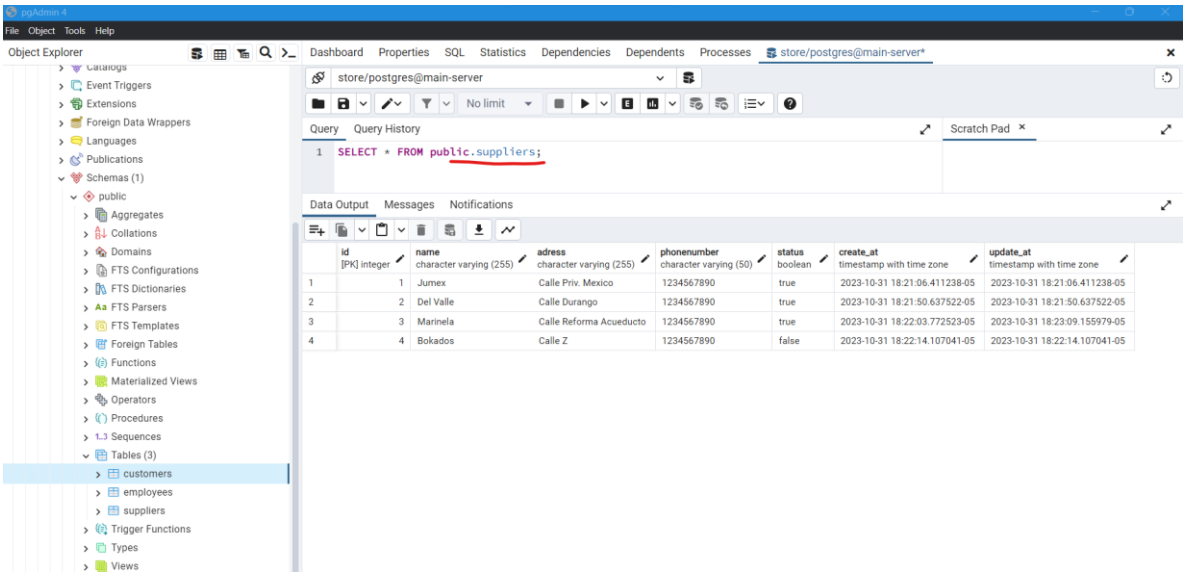
```
from logger_base import log
from conecction import Conecction

class Cursor_of_the_pool:
    def __init__(self) -> None:
        self.__conexion = None
        self.__cursor = None

    def __enter__(self):
        log.debug("START OF BLOCK WITH")
        self.__conexion = Conecction.obtain_connection()
        self.__cursor = self.__conexion.cursor()
        return self.__cursor


    def __exit__(self, tipo_excepcion, valor_excepcion, detalle_excepcion):
        log.debug("EXIT IS EXECUTED")
```

```
        if valor_excepcion:
            self.__conexion.rollback()
        else:
            self.__conexion.commit()
        self.__cursor.close()
        Conecction.release_connection(self.__conexion)

if __name__ == "__main__":
    with Cursor_of_the_pool() as cursor:
        log.debug("BLOQUE WITH")
        cursor.execute("SELECT * FROM employees")
        log.debug(cursor.fetchall())
```

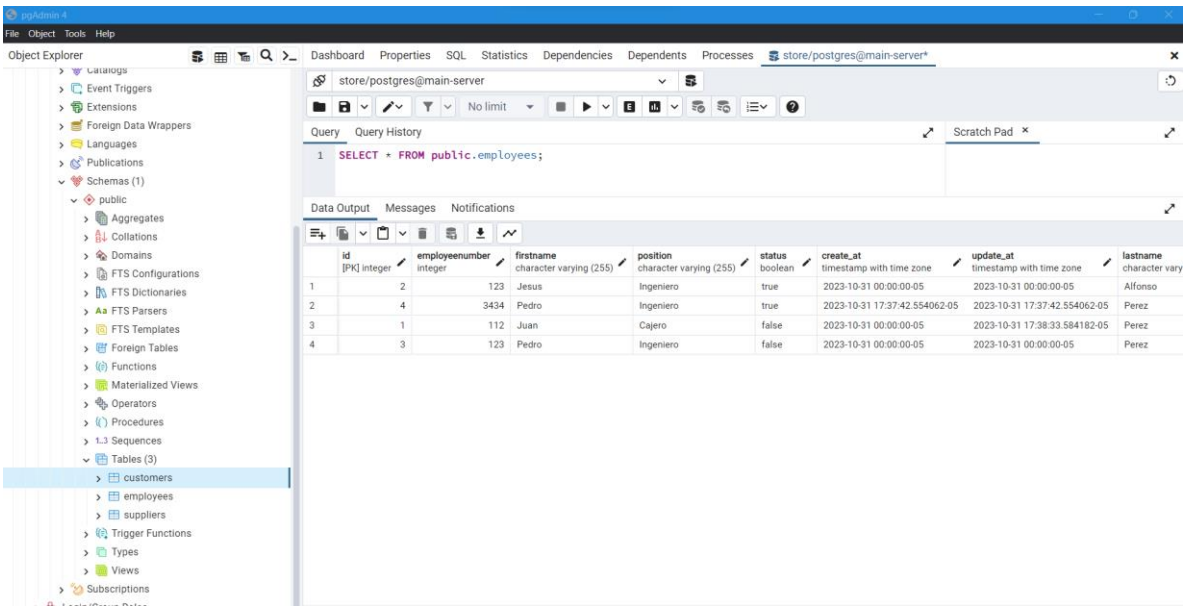
Base de datos



Equipo 1 - Python



	id [PK] integer	name character varying (255)	adress character varying (255)	phonenumber character varying (50)	status boolean	create_at timestamp with time zone	update_at timestamp with time zone
1	1	Jumex	Calle Priv. Mexico	1234567890	true	2023-10-31 18:21:06.411238-05	2023-10-31 18:21:06.411238-05
2	2	Del Valle	Calle Durango	1234567890	true	2023-10-31 18:21:50.637522-05	2023-10-31 18:21:50.637522-05
3	3	Marinela	Calle Reforma Acueducto	1234567890	true	2023-10-31 18:22:03.772523-05	2023-10-31 18:23:09.155979-05
4	4	Bokados	Calle Z	1234567890	false	2023-10-31 18:22:14.107041-05	2023-10-31 18:22:14.107041-05



	id [PK] integer	employeenumber integer	firstname character varying (255)	position character varying (255)	status boolean	create_at timestamp with time zone	update_at timestamp with time zone	lastname character vary
1	2	123	Jesus	Ingeniero	true	2023-10-31 00:00:00-05	2023-10-31 00:00:00-05	Alfonso
2	4	3434	Pedro	Ingeniero	true	2023-10-31 17:37:42.554062-05	2023-10-31 17:37:42.554062-05	Perez
3	1	112	Juan	Cajero	false	2023-10-31 00:00:00-05	2023-10-31 17:38:33.584182-05	Perez
4	3	123	Pedro	Ingeniero	false	2023-10-31 00:00:00-05	2023-10-31 00:00:00-05	Perez

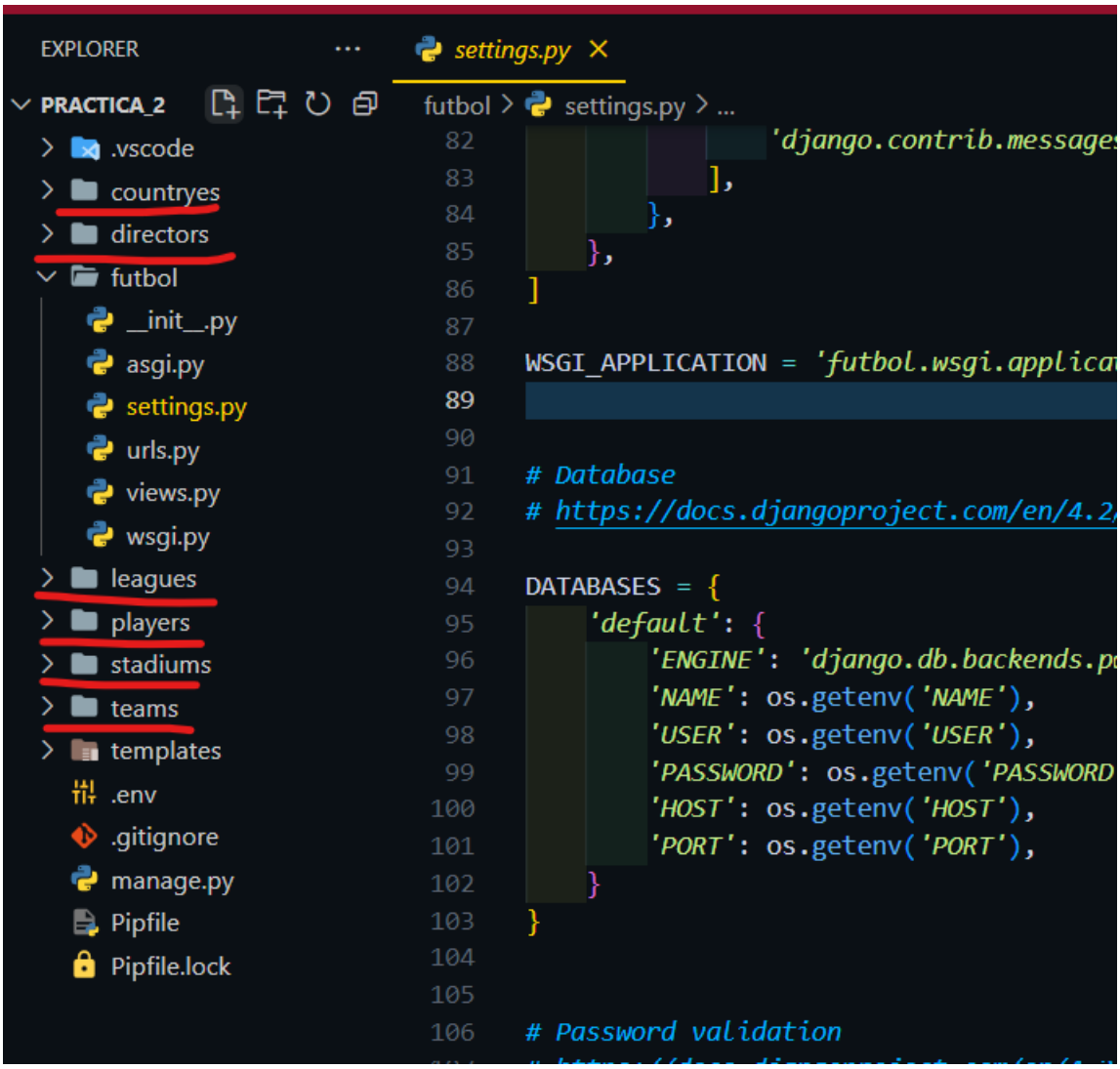
Práctica 2 (Django)

Realizar una aplicación utilizando el Framework DJANGO y que contenga lo siguiente

1. Conexión a base de datos postgresql

```
WSGI_APPLICATION = 'futbol.wsgi.application'


# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.getenv('NAME'),
        'USER': os.getenv('USER'),
        'PASSWORD': os.getenv('PASSWORD'),
        'HOST': os.getenv('HOST'),
        'PORT': os.getenv('PORT'),
    }
}
```

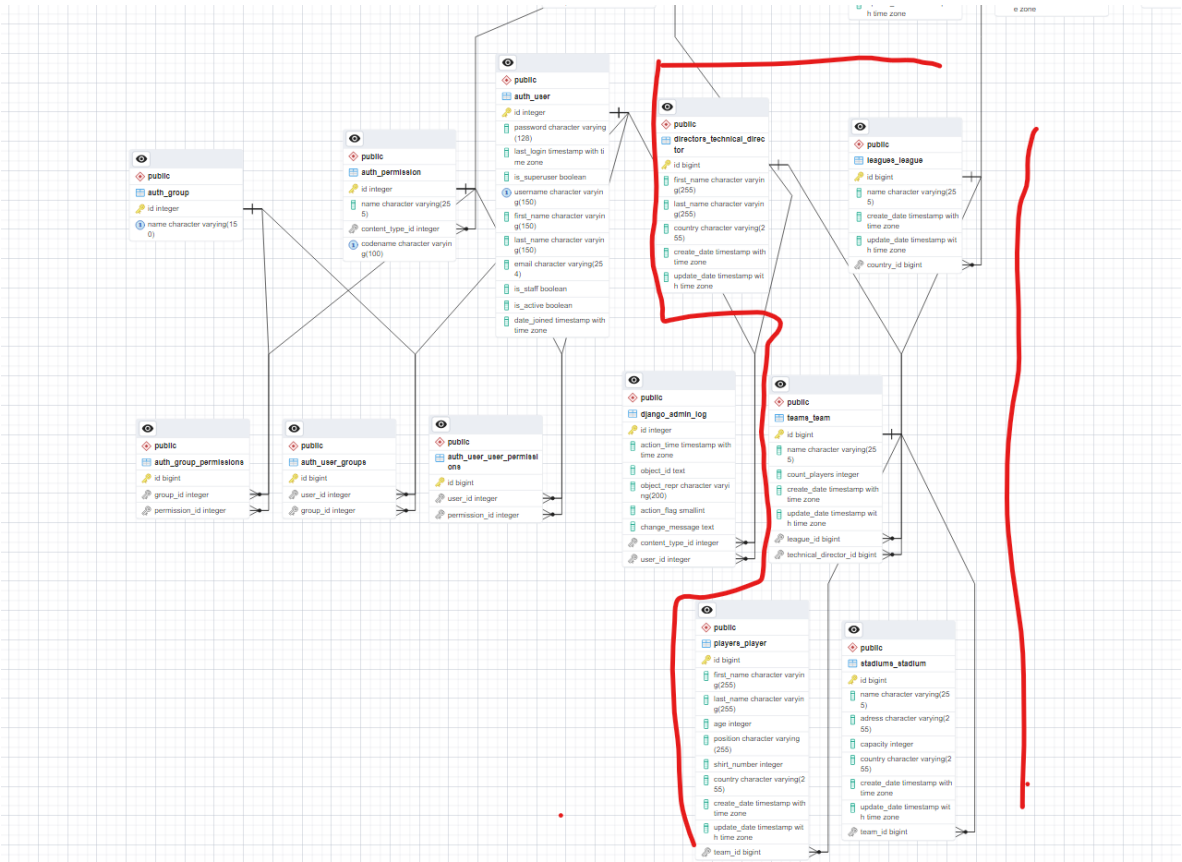
2. Utilizar al menos 6 entidades

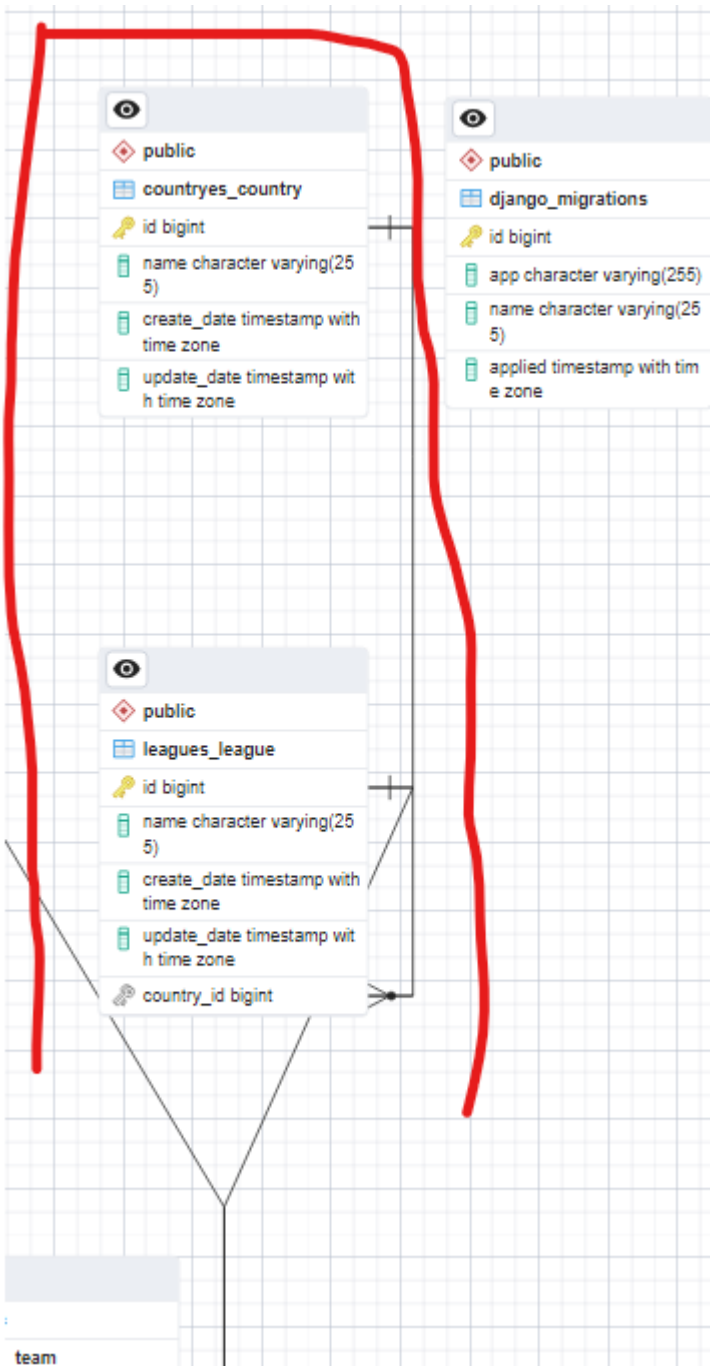


3. 2 de las entidades debe relacionarse con otra

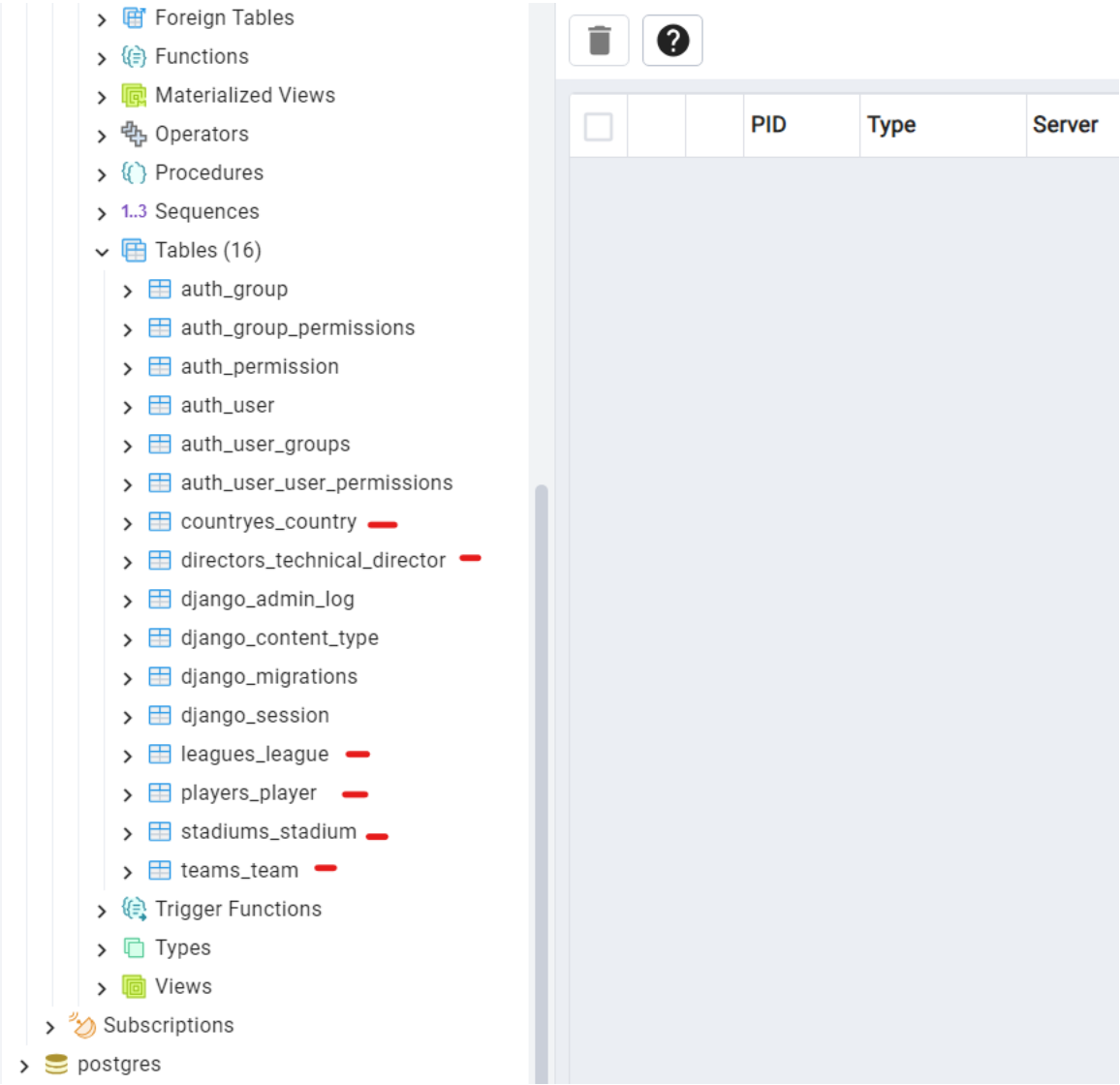
Se realizaron más de 2 relaciones entre las entidades

Equipo 1 - Python

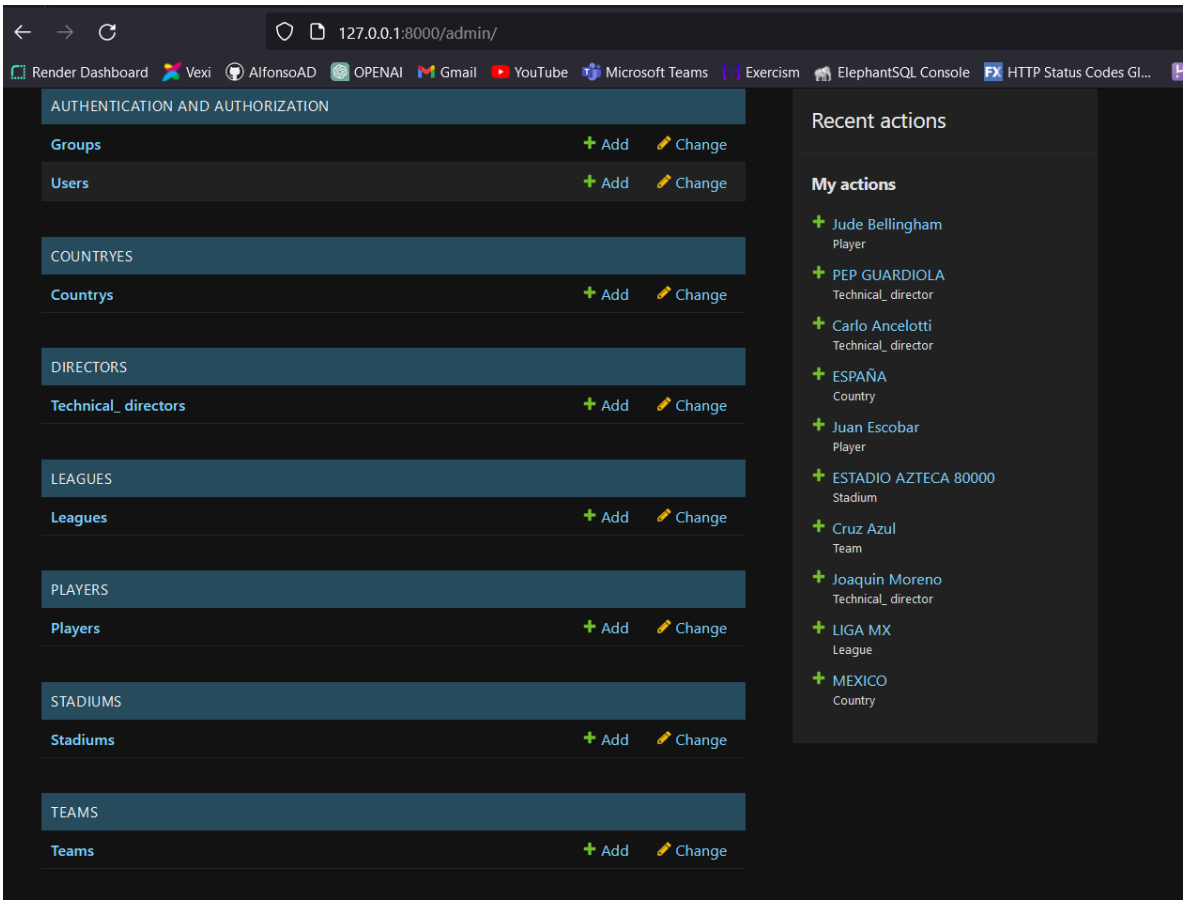




4. Realizar migraciones



5. Agregar las 6 entidades a la página de administración



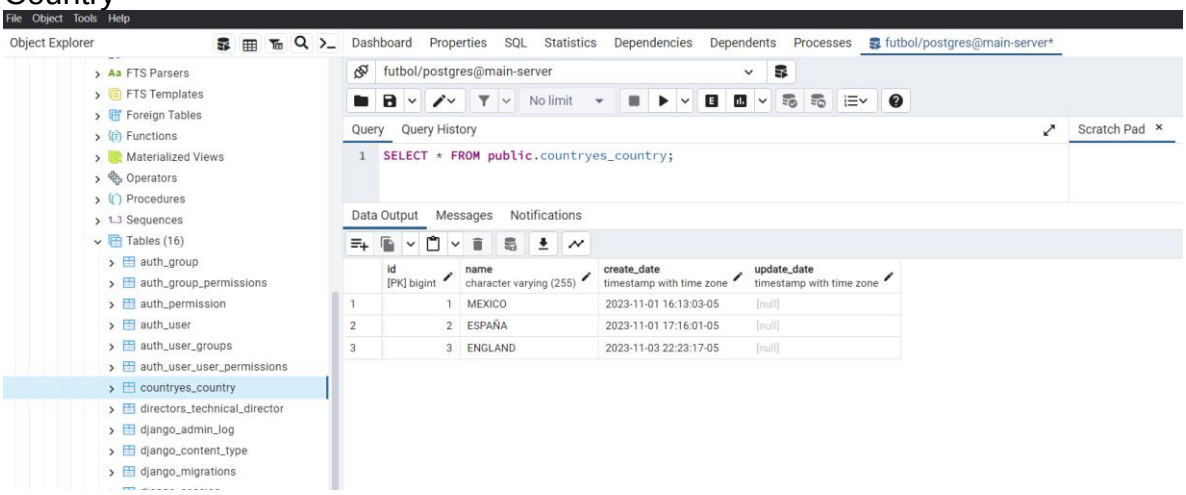
Ejemplo de código de como se agregaron a la página de administración

```
from django.contrib import admin
from .models import Player


class PlayersAdmin(admin.ModelAdmin):
    list_display = ('id', 'first_name', 'last_name', 'age',
                    'position', 'shirt_number', 'country')


admin.site.register(Player, PlayersAdmin)
```

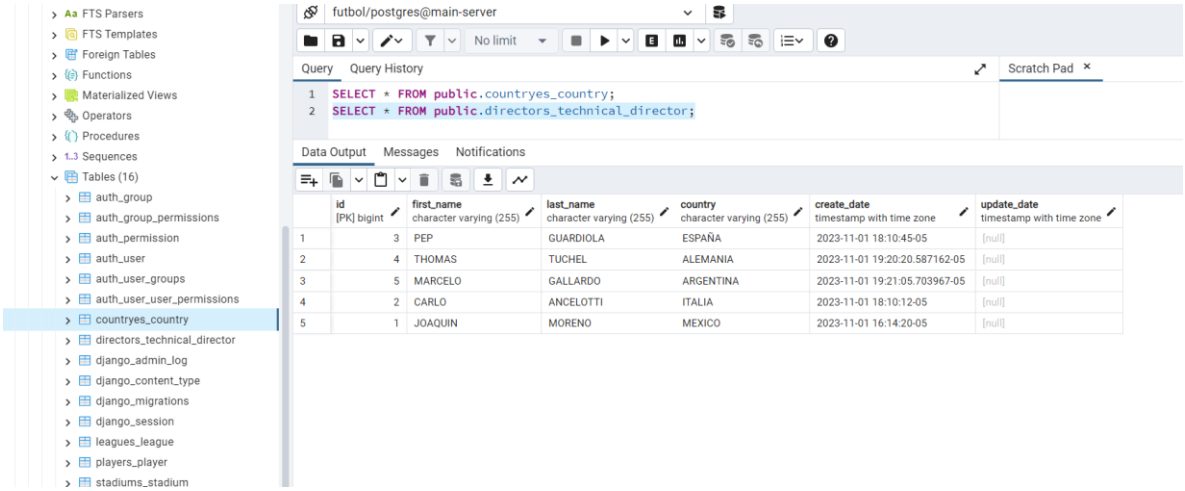
6. Crear al menos 3 registros por entidad

Equipo 1 - Python

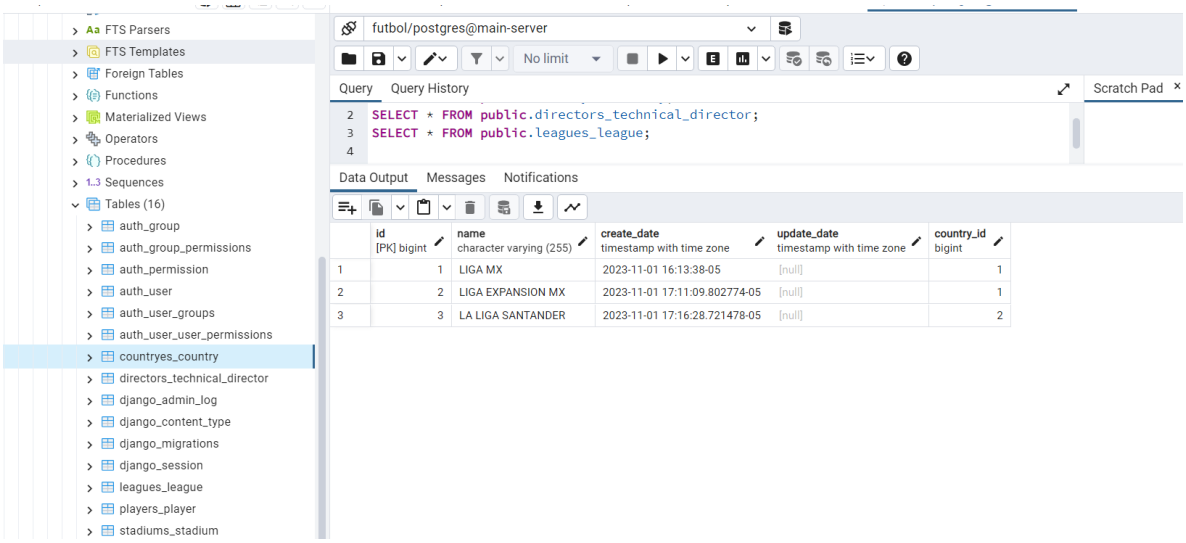
Country



Technical directors

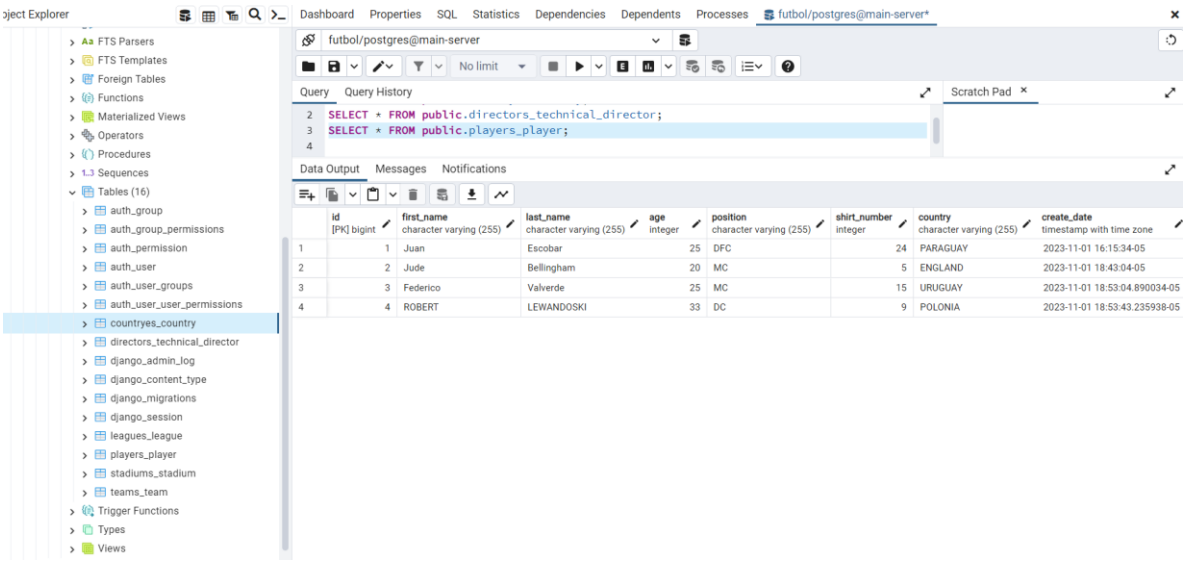


Leagues

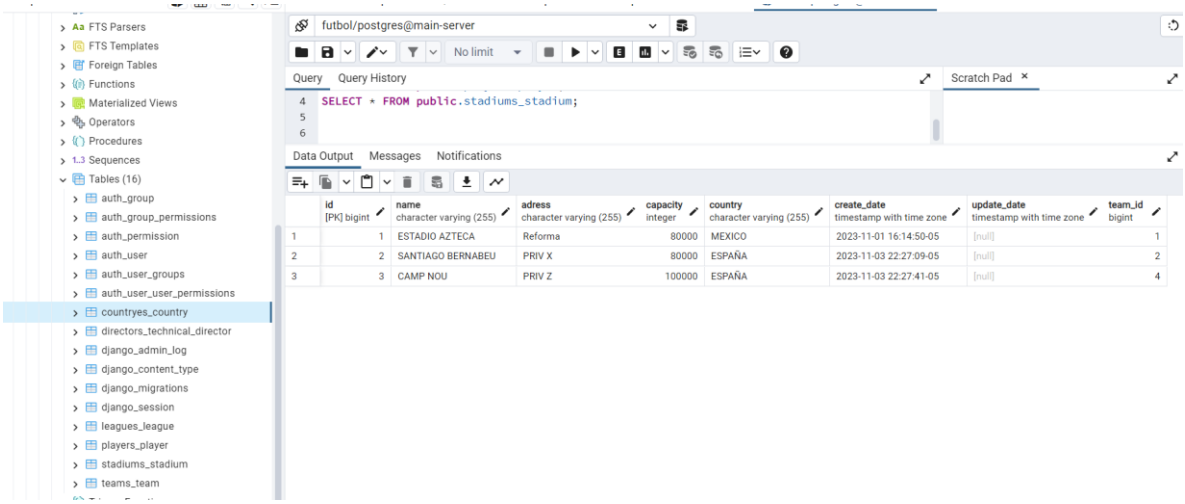


Equipo 1 - Python

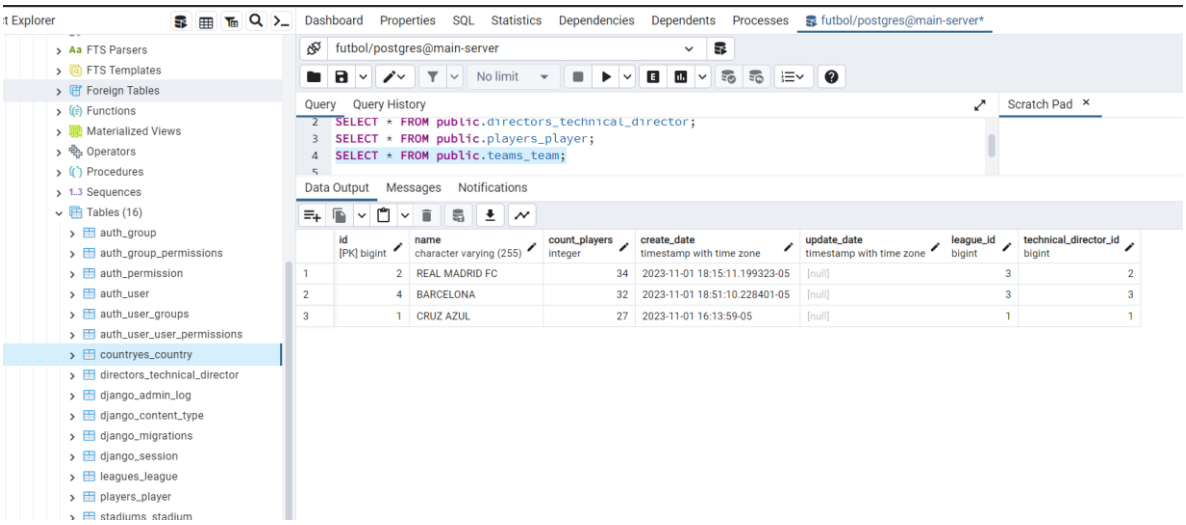
Players



Stadiums



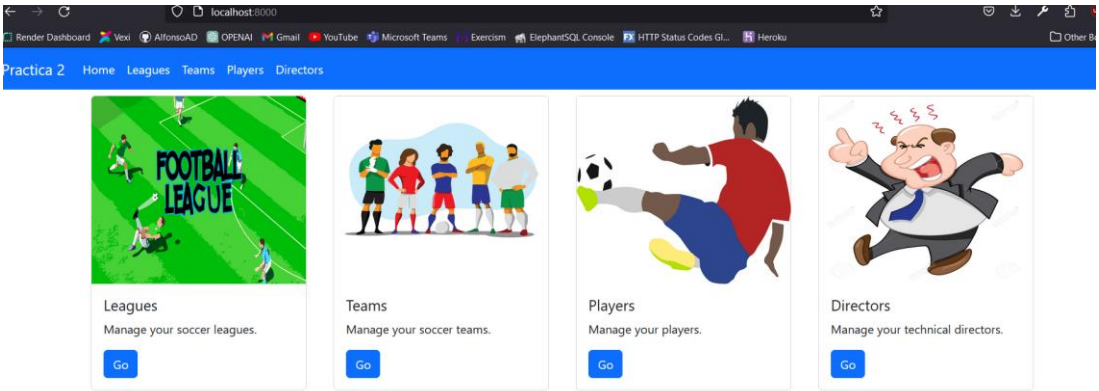
Teams

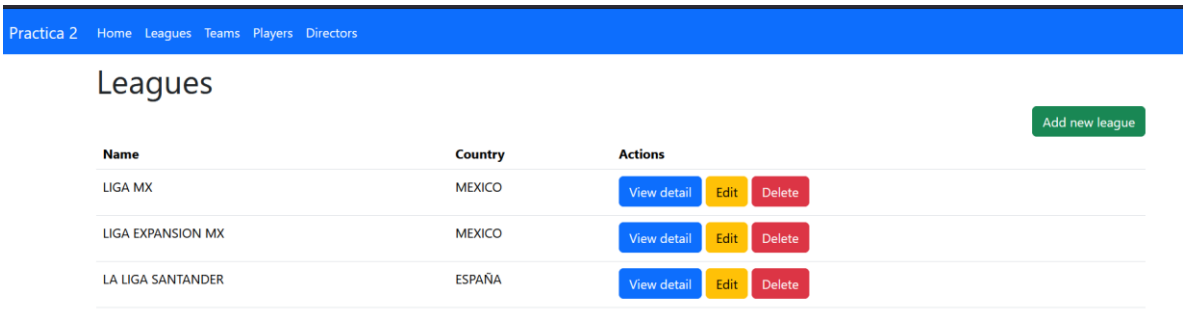


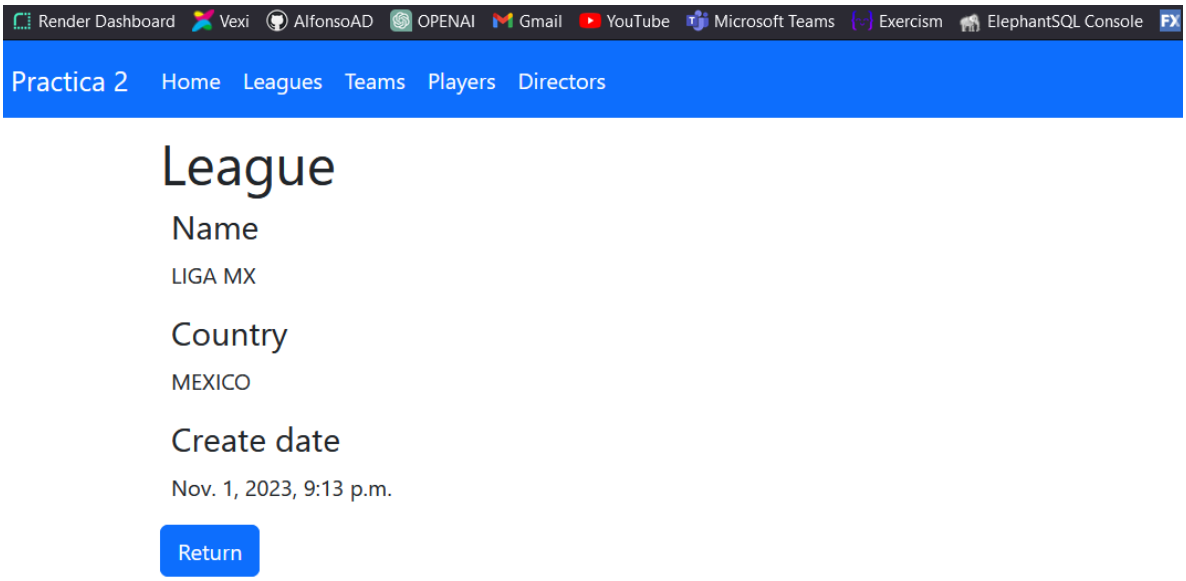
7. Realizar el listado de al menos 3 entidades

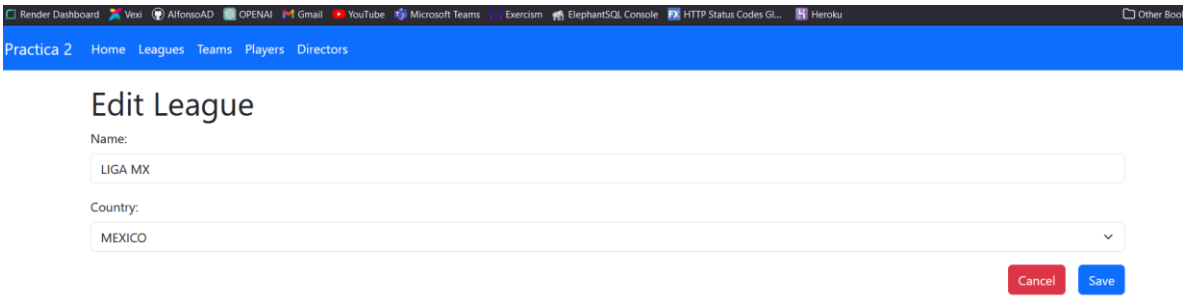
Para acceder a los listados (4/6)

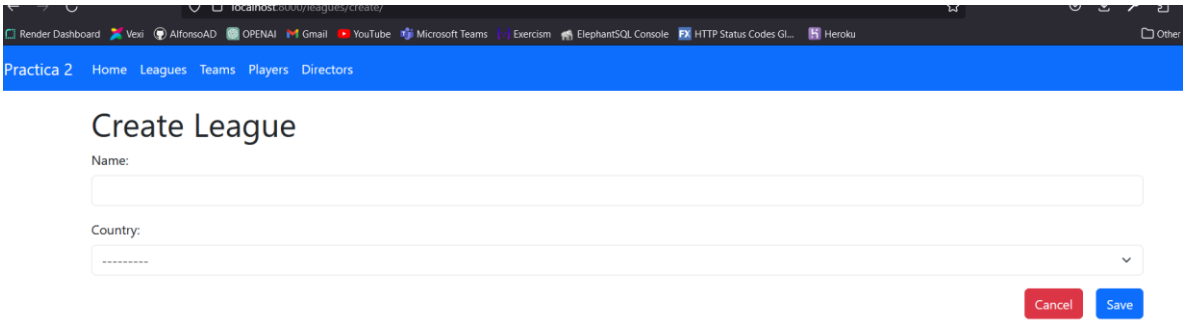
Equipo 1 - Python

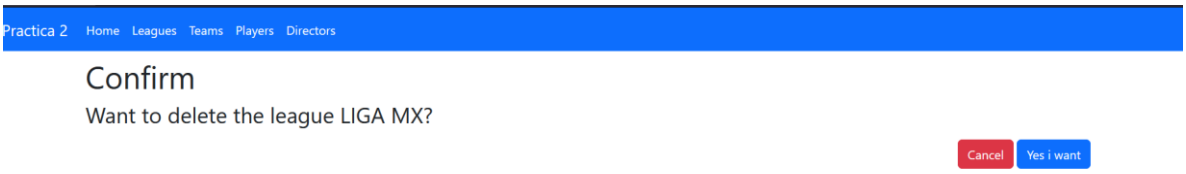




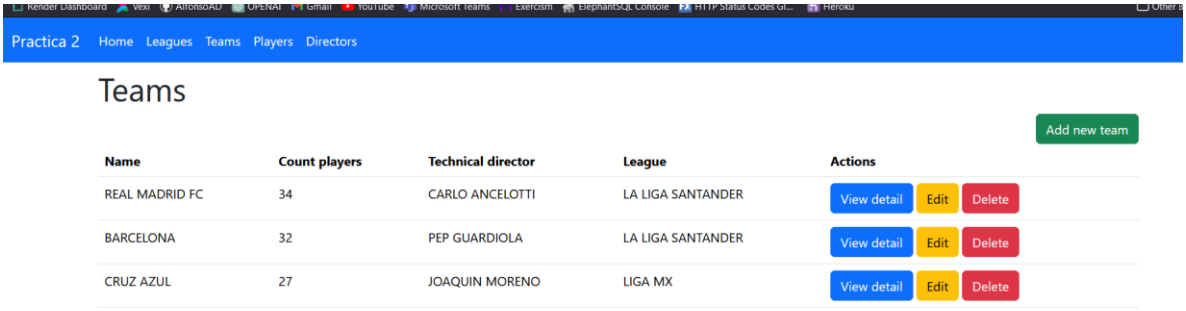




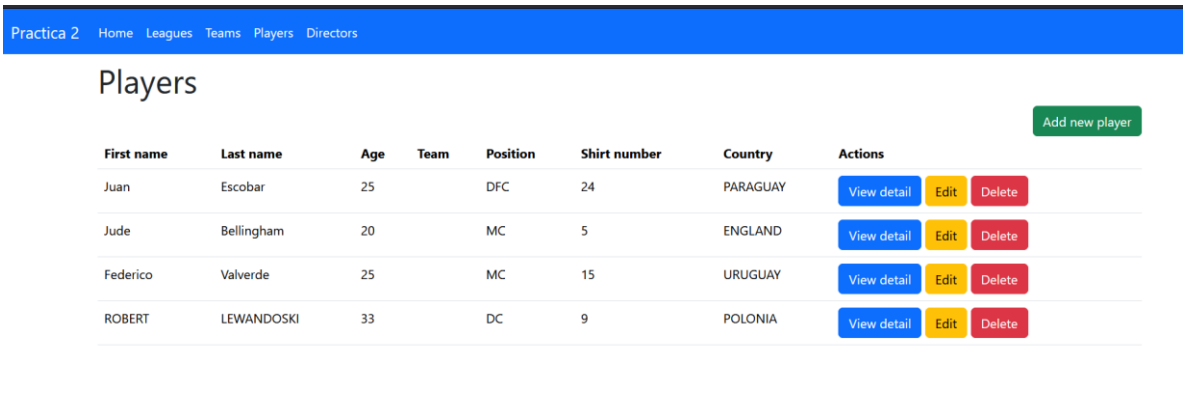


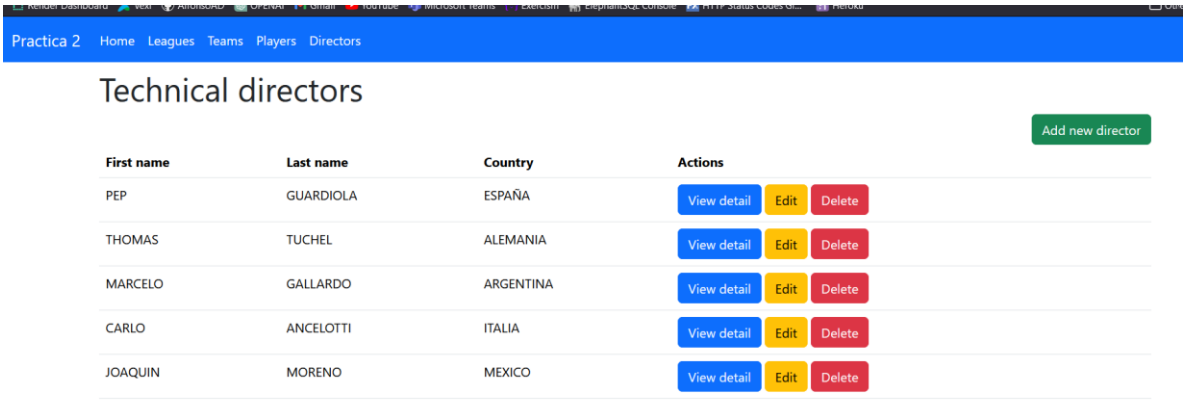


Todos los formularios son similares con sus respectivos inputs.



Equipo 1 - Python

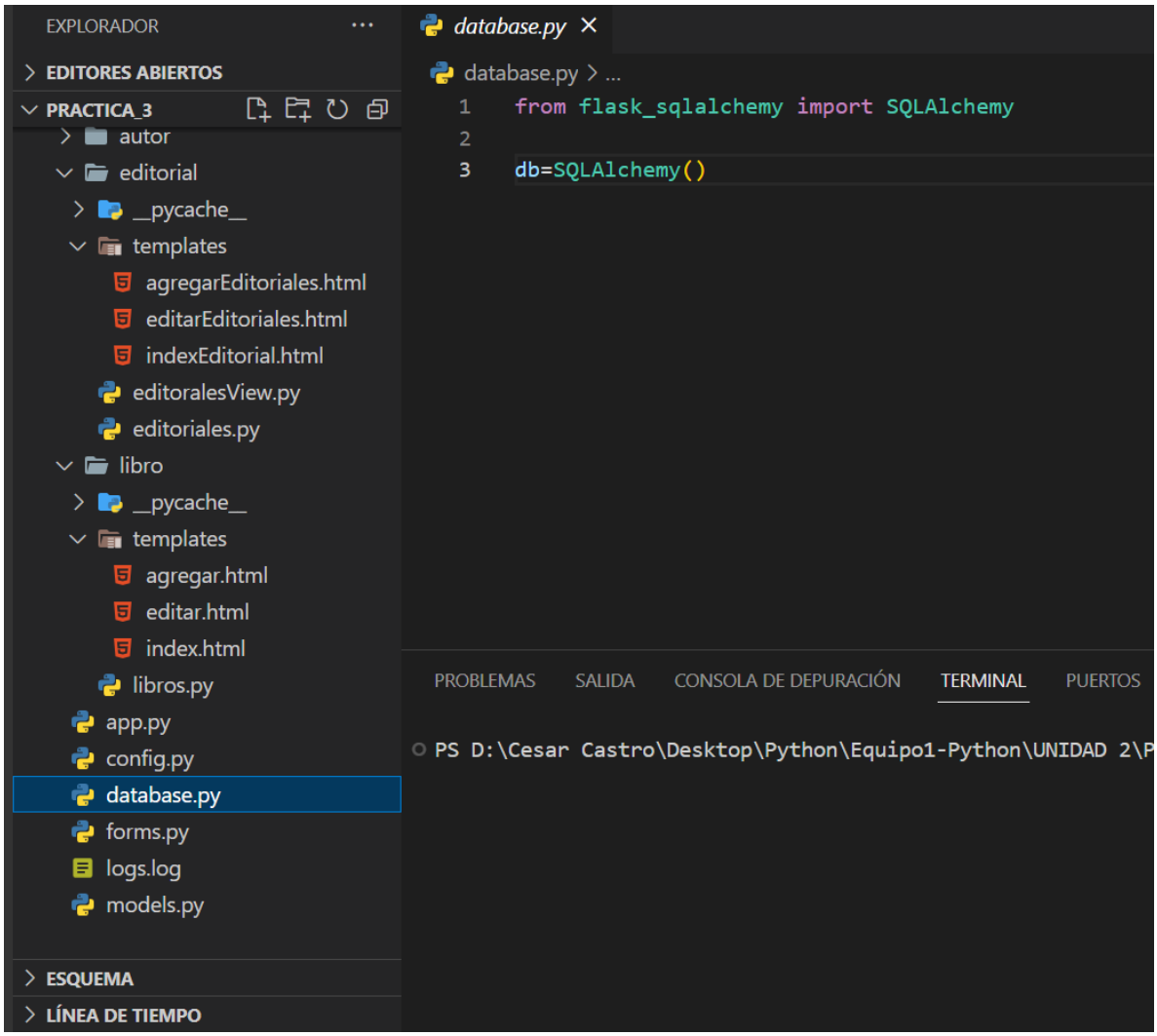




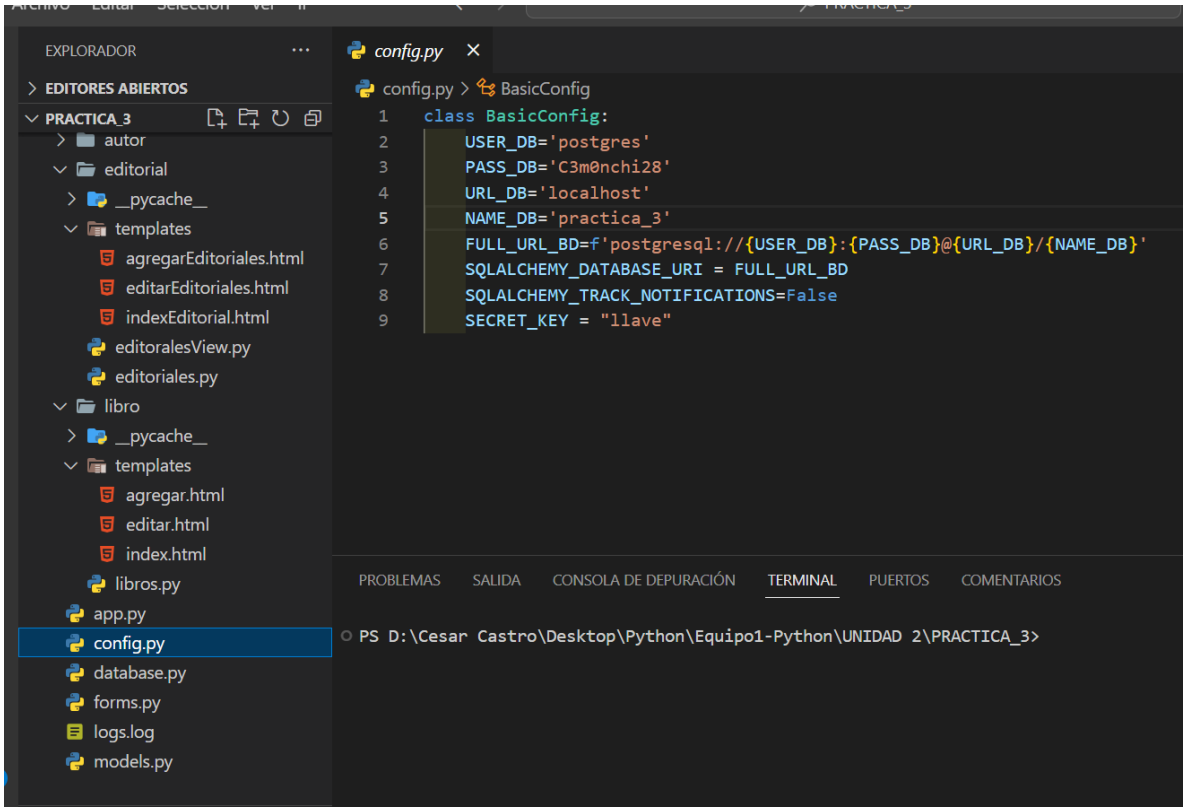
Práctica 3 (Flask)

1. Conexión a base de datos postgresql con SQLAlchemy

Importación de SQLAlchemy para el manejo de la base de datos



Estableciendo la liga de conexión de la base de datos



Realizamos la configuración en nuestra app

```
app.config.from_object(BasicConfig)
db.init_app(app)
migrate = Migrate()
migrate.init_app(app,db)
logging.basicConfig(level=logging.DEBUG,filename="logs.log")
```

1. Utilizar app logging

En la imagen anterior se aprecia que usamos app loggin

App loggin corriendo

```
app.py    logs.log
logs.log
1   ERROR:flask_migrate:Error: Directory migrations already exists and is not empty
2   ERROR:flask_migrate:Error: Directory migrations already exists and is not empty
3   INFO:werkzeug:esc[31mesc[1mWARNING: This is a development server. Do not use it in a production deployment. U
4    * Running on http://127.0.0.1:5000
5   INFO:werkzeug:esc[33mPress CTRL+C to quitesc[0m
6   INFO:werkzeug: * Restarting with stat
7   WARNING:werkzeug: * Debugger is active!
8   INFO:werkzeug: * Debugger PIN: 879-829-383
9   ERROR:flask_migrate:Error: Directory migrations already exists and is not empty
10  ERROR:flask_migrate:Error: Directory migrations already exists and is not empty
11  ERROR:flask_migrate:Error: Directory migrations already exists and is not empty
12  INFO:werkzeug:esc[31mesc[1mWARNING: This is a development server. Do not use it in a production deployment. U
13   * Running on http://127.0.0.1:5000
14  INFO:werkzeug:esc[33mPress CTRL+C to quitesc[0m
15  INFO:werkzeug: * Restarting with stat
16  WARNING:werkzeug: * Debugger is active!
17  INFO:werkzeug: * Debugger PIN: 879-829-383
18  INFO:werkzeug:127.0.0.1 - - [03/Nov/2023 09:33:39] "esc[31mesc[1mGET /autor/agregar HTTP/1.1esc[0m" 405 -
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS   COMENTARIOS   powershell
PS D:\Cesar Castro\Desktop\Python\Equipo1-Python\UNIDAD 2\PRACTICA 3>
```

2. Utilizar al menos 3 entidades

Creamos 3 modelos simulando una biblioteca

Equipo 1 - Python

models.py > Libro

```python
from app import db

class Autor(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(100), nullable=False)
    nacionalidad = db.Column(db.String(50))

class Libro(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    titulo = db.Column(db.String(200), nullable=False)
    autor = db.Column(db.String(100), nullable=False)
    editorial = db.Column(db.String(100), nullable=False)
    year = db.Column(db.Integer)
    genero = db.Column(db.String(50))

class Editorial(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(100), nullable=False)
    pais = db.Column(db.String(50))
    telefono = db.Column(db.Integer)
```

Lograron migrarse correctamente

- Tables (4)
  - alembic_version
  - autor
  - editorial
  - libro

Migraciones.

- migrations
  - __pycache__
  - versions
    - __pycache__
    - b0e47ed85c23_.py
  - alembic.ini
  - env.py
  - README
  - script.py.mako

```python
        return str(get_engine().url).replace('%', '%%')


# add your model's MetaData object here
from models import Libro, Autor, Editorial
# for 'autogenerate' support
# from myapp import mymodel
# target_metadata = mymodel.Base.metadata
config.set_main_option('sqlalchemy.url', get_engine_url())
target_db = current_app.extensions['migrate'].db

```

Equipo 1 - Python

CRUD HTTP con el modelo Autor

Equipo 1 - Python

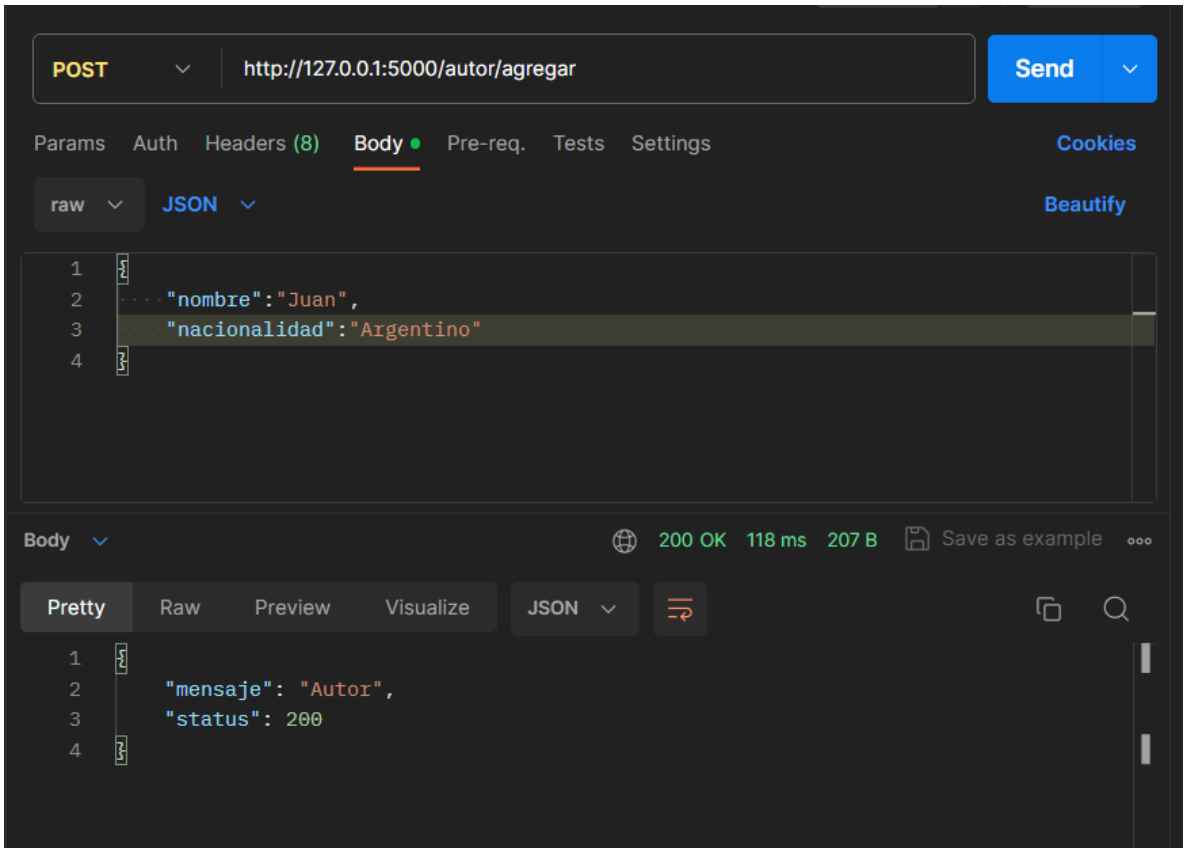
```python
from flask import Blueprint,jsonify,request
from models import Autor
from app import db

appautor = Blueprint("appautor",__name__)

@appautor.route('/autor/agregar',methods=['POST'])
def agregarAutor():
    try:
        json = request.get_json()
        autor=Autor()
        autor.nombre=json['nombre']
        autor.nacionalidad=json['nacionalidad']
        db.session.add(autor)
        db.session.commit()
        return jsonify({"status":200,"mensaje":"Autor"})
    except Exception as ex:
        return jsonify({"status":400,"mensaje":ex})

@appautor.route('/autor/editar',methods=['POST'])
def editarAutor():
    try:
        json = request.get_json()
        autor=Autor.query.get_or_404(json["id"])
```

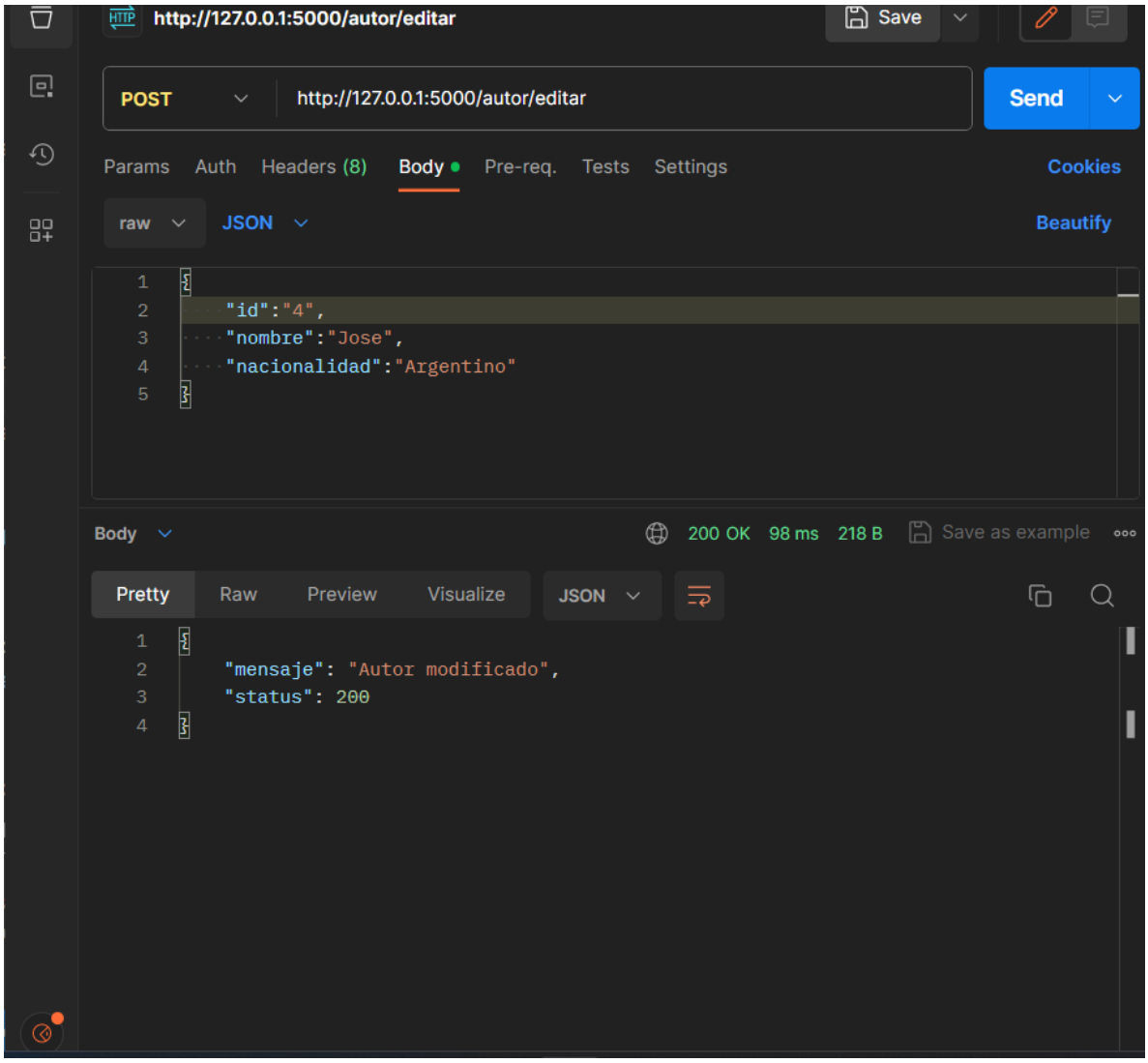
Equipo 1 - Python

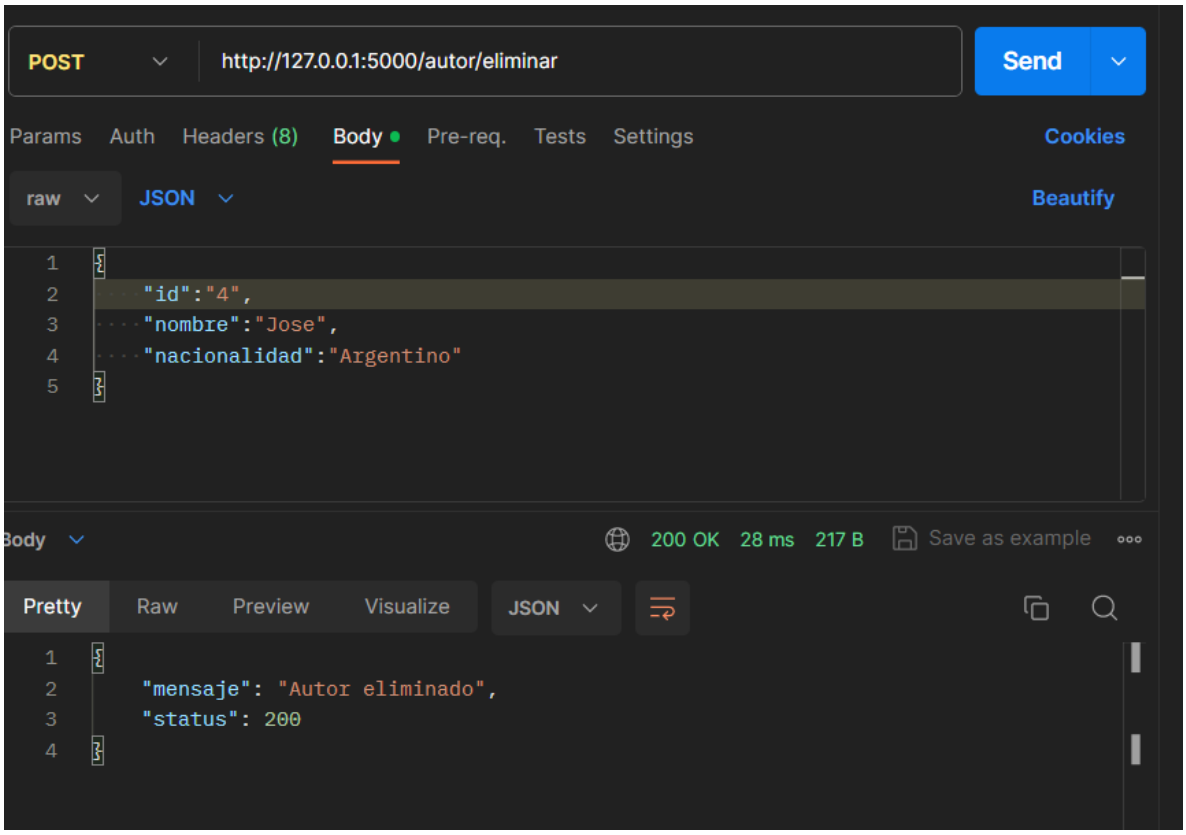
```python
@appautor.route('/autor/editar',methods=['POST'])
def editarAutor():
    try:
        json = request.get_json()
        autor=Autor.query.get_or_404(json["id"])
        autor.nombre=json['nombre']
        autor.nacionalidad =json['nacionalidad']
        db.session.commit()
        return jsonify({"status":200,"mensaje":"Autor modificado"})
    except Exception as ex:
        return jsonify({"status":400,"mensaje":ex})

@appautor.route('/autor/eliminar',methods=['POST'])
def eliminarAutor():
    try:
        json = request.get_json()
        autor=Autor.query.get_or_404(json["id"])
        db.session.delete(autor)
        db.session.commit()
        return jsonify({"status":200,"mensaje":"Autor eliminado"})
    except Exception as ex:
        return jsonify({"status":400,"mensaje":ex})
```

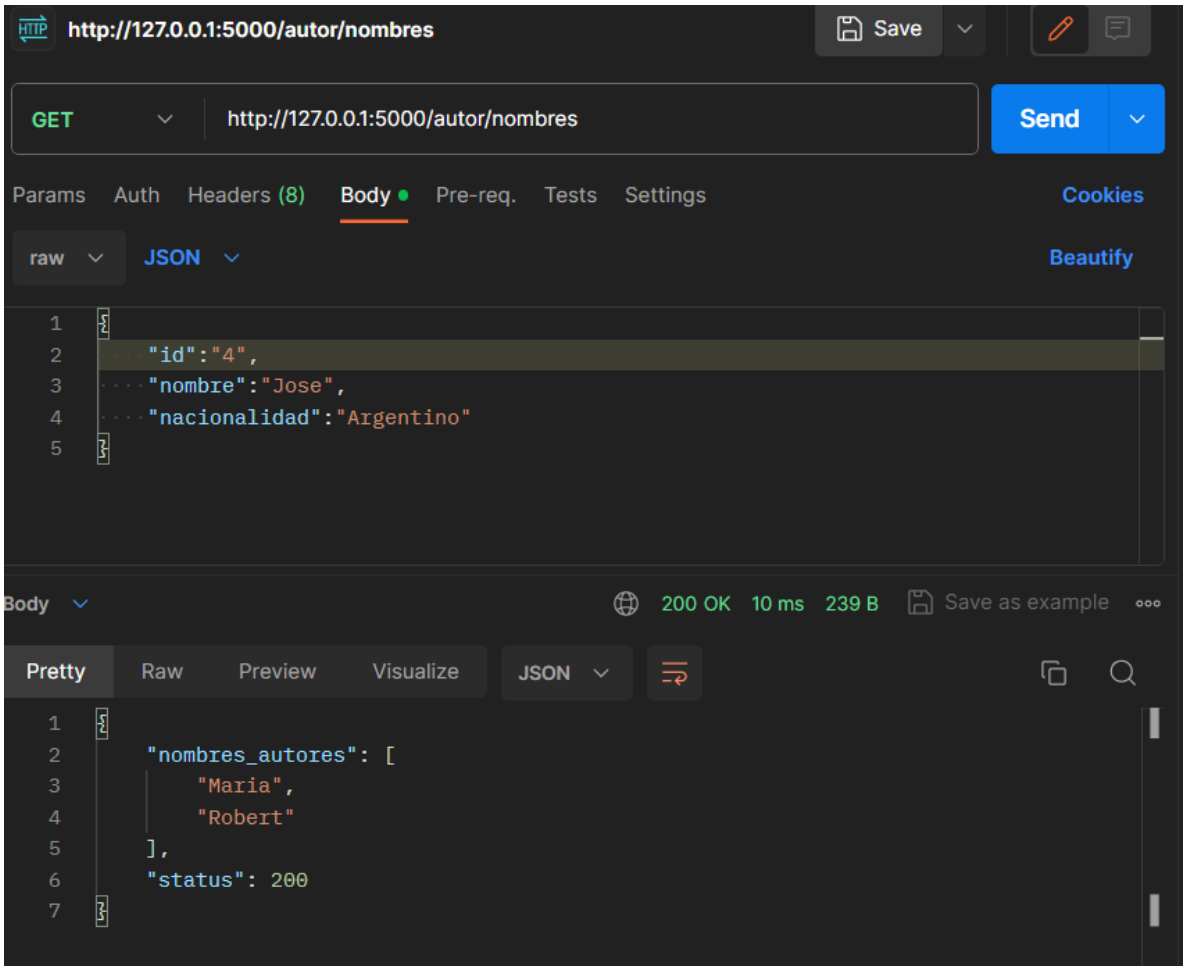
```python
@appautor.route('/autor/nombres', methods=['GET'])
def obtenerNombresAutores():
    try:
        autores = Autor.query.all()
        nombres_autores = [autor.nombre for autor in autores]
        return jsonify({"status": 200, "nombres_autores": nombres_autores})
    except Exception as ex:
        return jsonify({"status": 400, "mensaje":ex})
```

Prueba


POST
http://127.0.0.1:5000/autor/agregar
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"nombre":"Juan",
"nacionalidad":"Argentino"
}
Body
200 OK
118 ms
207 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"mensaje": "Autor",
"status": 200
}


http://127.0.0.1:5000/autor/editar
Save
POST
http://127.0.0.1:5000/autor/editar
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"id":"4",
"nombre":"Jose",
"nacionalidad":"Argentino"
}
Body
200 OK
98 ms
218 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"mensaje": "Autor modificado",
"status": 200
}


POST
http://127.0.0.1:5000/autor/eliminar
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
1
2 "id":"4",
3 "nombre":"Jose",
4 "nacionalidad":"Argentino"
5
Body
200 OK
28 ms
217 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
1
2 "mensaje": "Autor eliminado",
3 "status": 200
4



Tambien se realizaron para el modelo de Editoriales

Equipo 1 - Python

```python
from flask import Blueprint,jsonify,request
from models import Editorial
from app import db

appeditorial = Blueprint("appeditorial",__name__)

@appeditorial.route('/editorial/agregar',methods=['POST'])
def agregarEditorial():
    try:
        json = request.get_json()
        editorial=Editorial()
        editorial.nombre=json['nombre']
        editorial.pais=json['pais']
        editorial.telefono=json['telefono']
        db.session.add(editorial)
        db.session.commit()
        return jsonify({"status":200,"mensaje":"Editorial"})
    except Exception as ex:
        return jsonify({"status":400,"mensaje":ex})
```

Equipo 1 - Python

```python
@appeditorial.route('/editorial/editar',methods=['POST'])
def editarEditorial():
    try:
        json = request.get_json()
        editorial=Editorial.query.get_or_404(json["id"])
        editorial.nombre=json['nombre']
        editorial.pais=json['pais']
        editorial.telefono=json['telefono']
        db.session.commit()
        return jsonify({"status":200,"mensaje":"Editorial modificado"})
    except Exception as ex:
        return jsonify({"status":400,"mensaje":ex})

@appeditorial.route('/editorial/eliminar',methods=['POST'])
def eliminarEditorial():
    try:
        json = request.get_json()
        editorial=Editorial.query.get_or_404(json["id"])
        db.session.delete(editorial)
        db.session.commit()
        return jsonify({"status":200,"mensaje":"Editorial eliminado"})
    except Exception as ex:
        return jsonify({"status":400,"mensaje":ex})
```

```python
@appeditorial.route('/editorial/nombres', methods=['GET'])
def obtenerNombresEditoriales():
    try:
        editorales = Editorial.query.all()
        nombres_editoriales = [editorial.nombre for editorial in editorales]
        return jsonify({"status": 200, "nombres_editoriales": nombres_editoriales})
    except Exception as ex:
        return jsonify({"status": 400, "mensaje":ex})
```

Pruebas

Equipo 1 - Python

POST http://127.0.0.1:5000/editorial/agregar Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

```
{
    "nombre":"Milenio",
    "pais":"Salvador",
    "telefono":"789456"
}
```

Body 200 OK 22 ms 211 B Save as example

Pretty Raw Preview Visualize

{ "mensaje": "Editorial", "status": 200 }

POST http://127.0.0.1:5000/editorial/editar Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

```
{
    "id":"4",
    "nombre":"Milenio",
    "pais":"Chile",
    "telefono":"789456"
}
```

Body 200 OK 25 ms 222 B Save as example

Pretty Raw Preview Visualize

{ "mensaje": "Editorial modificado", "status": 200 }

POST http://127.0.0.1:5000/editorial/eliminar Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

```
{
    "id":"2",
    "nombre":"Milenio",
    "pais":"Salvador",
    "telefono":"7894561230"
}
```

Body 200 OK 29 ms 221 B Save as example

Pretty Raw Preview Visualize JSON

```
{
    "mensaje": "Editorial eliminado",
    "status": 200
}
```

GET http://127.0.0.1:5000/editorial/nombres Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

```
{
    "nombre":"Milenio",
    "pais":"Salvador",
    "telefono":"7894561230"
}
```

Body 200 OK 16 ms 245 B Save as example

Pretty Raw Preview Visualize

{ "nombres_editoriales": ["Panini", "Planeta"], "status": 200 }

Repositorio de github

https://github.com/AlfonsoAD/Equipo1-Python.git

Comentarios y conclusiones

Como equipo sentimos que trabajar con PostgreSQL y Psycopg2 para realizar operaciones CRUD es una combinación poderosa que ofrece un manejo eficiente de bases de datos en Python. La robustez de PostgreSQL junto con la conectividad proporcionada por Psycopg2 brinda un enfoque confiable para la gestión de datos.

En cuanto a Django, consideramos que es un marco de desarrollo web sólido y completo. Su enfoque basado en convenciones y la amplia gama de herramientas integradas hacen que sea una opción excelente para proyectos de gran escala. El ORM de Django facilita la interacción con la base de datos, lo que simplifica considerablemente el desarrollo.

Por otro lado, Flask nos parece una opción más flexible y liviana. Su enfoque minimalista es excelente para proyectos más pequeños o cuando se prefiere una estructura más modular. Aunque ofrece menos en términos de funcionalidades integradas en comparación con Django, su flexibilidad es una ventaja para ciertos escenarios de desarrollo.

En resumen, cada una de estas herramientas tiene sus propias fortalezas y áreas de aplicación. La elección depende del contexto del proyecto ya que cada una ofrece un enfoque único para el desarrollo en Python.