



**TECNOLÓGICO  
NACIONAL DE MÉXICO**



**TECNOLOGICO NACIONAL DE MÉXICO**  
**CAMPUS 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 (PostgreSQL) .....	3
Práctica 2 (Django) .....	14
Práctica 3 (Flask) .....	26
Enlace de github .....	41
Comentarios y conclusiones .....	41

## Práctica 1 (PostgreSQL)

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._onenumber = 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._onenumber} 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
```

## Equipo 1 - Python

```
@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, employee_number = None, firstname = None, lastname =
None, position = None, status = None, createat = None, updateat = None) ->
None:
        self._employeenumber = employee_number
        self._firstname = firstname
        self._lastname = lastname
        self._position = position
        self._status = status
        self._create_at = createat
        self._update_at = updateat
```

## Equipo 1 - 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
```

## Equipo 1 - Python

```
@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
```

## Equipo 1 - Python

```
@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:
```

## Equipo 1 - Python

```
        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(employeeenumber, firstname, lastname,
position, status, create_at, update_at) VALUES(%s,%s,%s,%s,%s,%s,%s,%s)"
    _UPDATE = "UPDATE employees SET employeeenumber=%s, firstname=%s,
lastname=%s, position=%s, status=%s, update_at=%s WHERE id=%s"
```



## Equipo 1 - Python

```
_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

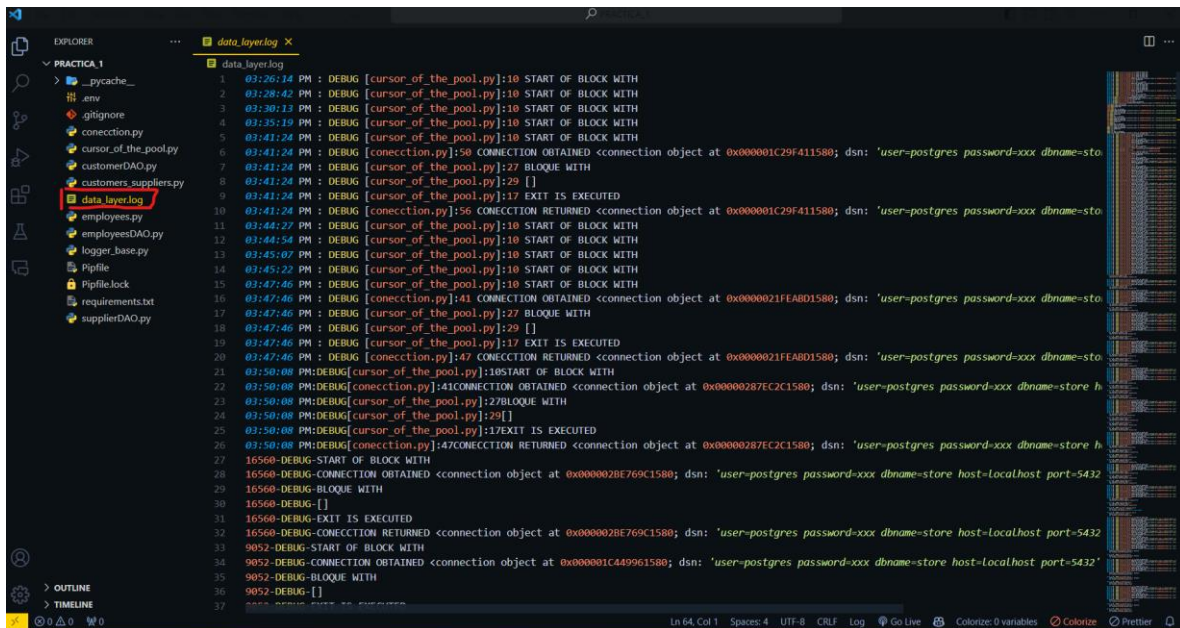
@classmethod
def post(cls, employee: Employee):
    with Cursor_of_the_pool() as cursor:
        values = (employee.employee_number, employee.firstname,
employee.lastname, employee.position, employee.status, employee.create_at,
employee.update_at)
        cursor.execute(cls._INSERT, values)
    return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

@classmethod
def put(cls, employee: Employee):
    employee.update_at = datetime.datetime.now()
    with Cursor_of_the_pool() as cursor:
        values = (employee.employee_number, 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
```

## Equipo 1 - Python

### 3. Usar archivo de logs



```
1 03:26:34 PM : DEBUG [cursor_of_the_pool.py]:10 START OF BLOCK WITH
2 03:28:42 PM : DEBUG [cursor_of_the_pool.py]:10 START OF BLOCK WITH
3 03:30:13 PM : DEBUG [cursor_of_the_pool.py]:10 START OF BLOCK WITH
4 03:35:19 PM : DEBUG [cursor_of_the_pool.py]:10 START OF BLOCK WITH
5 03:41:24 PM : DEBUG [cursor_of_the_pool.py]:10 START OF BLOCK WITH
6 03:41:24 PM : DEBUG [connection.py]:50 CONNECTION OBTAINED <connection object at 0x0000001C29F411580; dsn: 'user=postgres password=xxx dbname=sto
7 03:41:24 PM : DEBUG [cursor_of_the_pool.py]:27 BLOQUE WITH
8 03:41:24 PM : DEBUG [cursor_of_the_pool.py]:29 []
9 03:41:24 PM : DEBUG [cursor_of_the_pool.py]:17 EXIT IS EXECUTED
10 03:41:24 PM : DEBUG [connection.py]:56 CONNECTION RETURNED <connection object at 0x0000001C29F411580; dsn: 'user=postgres password=xxx dbname=sto
11 03:44:27 PM : DEBUG [cursor_of_the_pool.py]:10 START OF BLOCK WITH
12 03:44:54 PM : DEBUG [cursor_of_the_pool.py]:10 START OF BLOCK WITH
13 03:45:07 PM : DEBUG [cursor_of_the_pool.py]:10 START OF BLOCK WITH
14 03:45:22 PM : DEBUG [cursor_of_the_pool.py]:10 START OF BLOCK WITH
15 03:47:46 PM : DEBUG [cursor_of_the_pool.py]:10 START OF BLOCK WITH
16 03:47:46 PM : DEBUG [connection.py]:41 CONNECTION OBTAINED <connection object at 0x00000021FEABD1580; dsn: 'user=postgres password=xxx dbname=sto
17 03:47:46 PM : DEBUG [cursor_of_the_pool.py]:27 BLOQUE WITH
18 03:47:46 PM : DEBUG [cursor_of_the_pool.py]:29 []
19 03:47:46 PM : DEBUG [connection.py]:47 CONNECTION RETURNED <connection object at 0x00000021FEABD1580; dsn: 'user=postgres password=xxx dbname=sto
20 03:50:08 PM:DEBUG[cursor_of_the_pool.py]:10START OF BLOCK WITH
21 03:50:08 PM:DEBUG[connection.py]:41CONNECTION OBTAINED <connection object at 0x000000287EC2C1580; dsn: 'user=postgres password=xxx dbname=store h
22 03:50:08 PM:DEBUG[cursor_of_the_pool.py]:27BLOQUE WITH
23 03:50:08 PM:DEBUG[cursor_of_the_pool.py]:29[]
24 03:50:08 PM:DEBUG[cursor_of_the_pool.py]:17EXIT IS EXECUTED
25 03:50:08 PM:DEBUG[connection.py]:47CONNECTION RETURNED <connection object at 0x000000287EC2C1580; dsn: 'user=postgres password=xxx dbname=store h
26 16560-DEBUG-START OF BLOCK WITH
27 16560-DEBUG-CONNECTION OBTAINED <connection object at 0x00000028E769C1580; dsn: 'user=postgres password=xxx dbname=store host=localhost port=5432
28 16560-DEBUG-BLOQUE WITH
29 16560-DEBUG-[]
30 16560-DEBUG-EXIT IS EXECUTED
31 16560-DEBUG-CONNECTION RETURNED <connection object at 0x00000028E769C1580; dsn: 'user=postgres password=xxx dbname=store host=localhost port=5432
32 9052-DEBUG-START OF BLOCK WITH
33 9052-DEBUG-CONNECTION OBTAINED <connection object at 0x0000001C449961580; dsn: 'user=postgres password=xxx dbname=store host=localhost port=5432'
34 9052-DEBUG-BLOQUE WITH
35 9052-DEBUG-[]
36 9052-DEBUG-EXIT IS EXECUTED
37 9052-DEBUG-CONNECTION RETURNED <connection object at 0x0000001C449961580; dsn: 'user=postgres password=xxx dbname=store host=localhost port=5432'
```

### 4. Utilizar pool de conexiones

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

load_dotenv()

class Coneccion:
    _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,
```

## Equipo 1 - Python

```
        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, connexion):
    cls.obtain_pool().putconn(connexion)
    log.debug(f"CONNECTION RETURNED {connexion}")

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

```
from logger_base import log
from connection import Connection

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

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

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

## Equipo 1 - Python

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

The screenshot shows the pgAdmin 4 interface. On the left, the 'Object Explorer' pane displays the database structure, with 'Tables (3)' highlighted under the 'public' schema. The tables listed are 'customers', 'employees', and 'suppliers'. The 'customers' table is selected. The main pane shows the 'Query' editor with the following SQL query:

```
1 SELECT * FROM public.customers;
```

Below the query editor, the 'Data Output' pane displays the results of the query in a table format:

	id	name	address	phonenumber	status	create_at	update_at
1	1	Coca cola	Calle Priv. Gonzalez	1234567890	true	2023-10-31 00:00:00-05	2023-10-31 00:00:00-05
2	3	Sabritas	Calle Priv. Z	1234567890	true	2023-10-31 18:12:22.11891-05	2023-10-31 18:13:52.077457-05
3	2	Gamesa	Calle Reforma	1234567890	true	2023-10-31 18:12:05.426917-05	2023-10-31 18:14:26.531928-05
4	4	Pepsi	Calle Priv. X	1234567890	false	2023-10-31 18:12:32.915052-05	2023-10-31 18:12:32.915052-05

# Equipo 1 - Python

The screenshot shows the pgAdmin 4 interface with the 'store/postgres@main-server' connection selected. The 'Object Explorer' on the left shows the 'public' schema with tables 'customers', 'employees', and 'suppliers'. The 'Query' tab is active, displaying the SQL query: `SELECT * FROM public.suppliers;`. The 'Data Output' tab shows the results of the query in a table with 8 columns: id, name, address, phonenumber, status, create\_at, and update\_at. The results are as follows:

id	name	address	phonenumber	status	create_at	update_at
1	Jumex	Calle Priv. Mexico	1234567890	true	2023-10-31 18:21:06.411238-05	2023-10-31 18:21:06.411238-05
2	Del Valle	Calle Durango	1234567890	true	2023-10-31 18:21:50.637522-05	2023-10-31 18:21:50.637522-05
3	Marinela	Calle Reforma Acueducto	1234567890	true	2023-10-31 18:22:03.772523-05	2023-10-31 18:23:09.155979-05
4	Bokados	Calle Z	1234567890	false	2023-10-31 18:22:14.107041-05	2023-10-31 18:22:14.107041-05

The screenshot shows the pgAdmin 4 interface with the 'store/postgres@main-server' connection selected. The 'Object Explorer' on the left shows the 'public' schema with tables 'customers', 'employees', and 'suppliers'. The 'Query' tab is active, displaying the SQL query: `SELECT * FROM public.employees;`. The 'Data Output' tab shows the results of the query in a table with 8 columns: id, employee number, first name, position, status, create\_at, update\_at, and last name. The results are as follows:

id	employee number	first name	position	status	create_at	update_at	last name
1	2	Jesus	Ingeniero	true	2023-10-31 00:00:00-05	2023-10-31 00:00:00-05	Alfonso
2	4	Pedro	Ingeniero	true	2023-10-31 17:37:42.554062-05	2023-10-31 17:37:42.554062-05	Perez
3	1	Juan	Cajero	false	2023-10-31 00:00:00-05	2023-10-31 17:38:33.584182-05	Perez
4	3	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

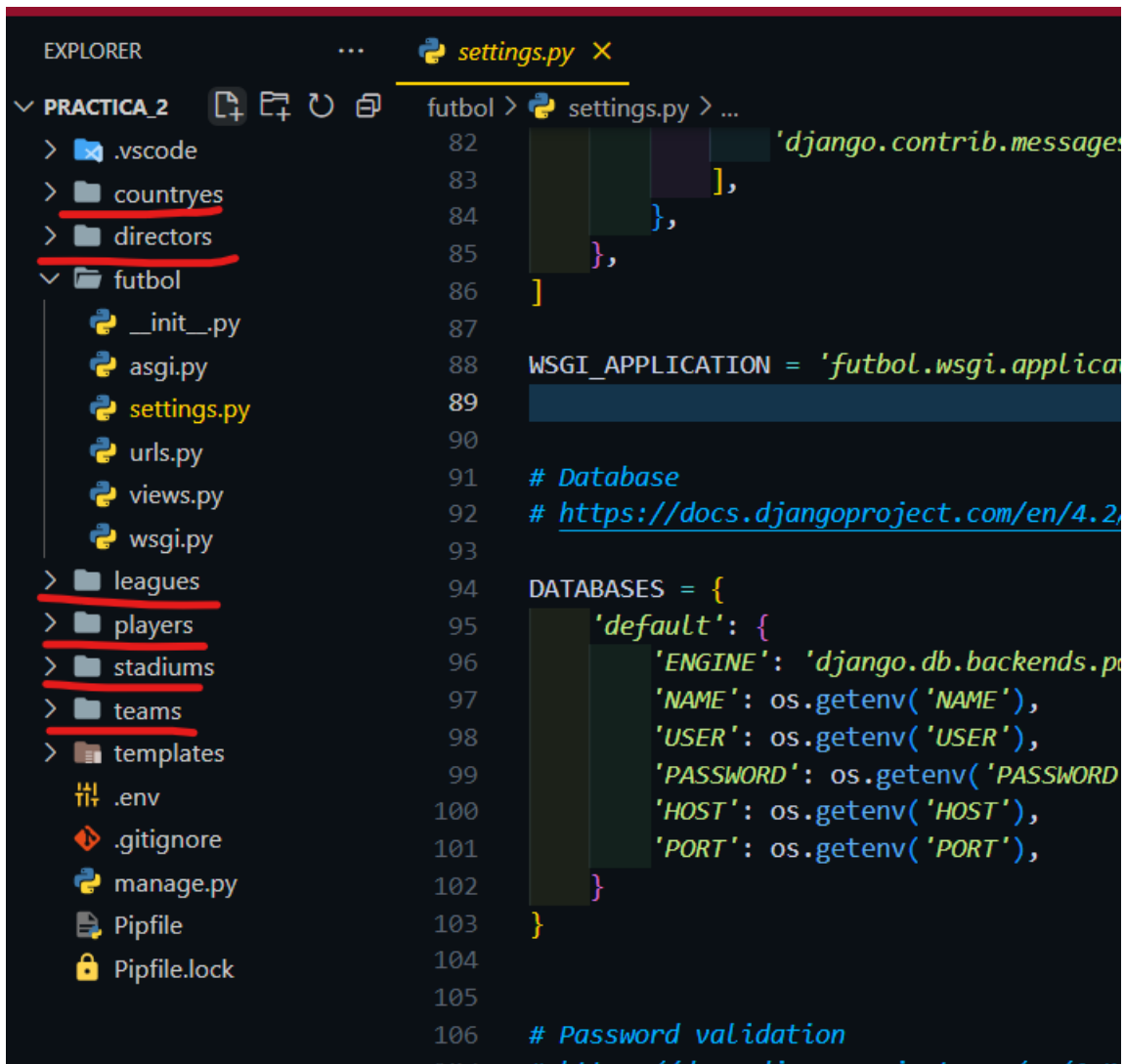
```
WSGI_APPLICATION = 'futbol.wsgi.application'

# 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

## Equipo 1 - Python



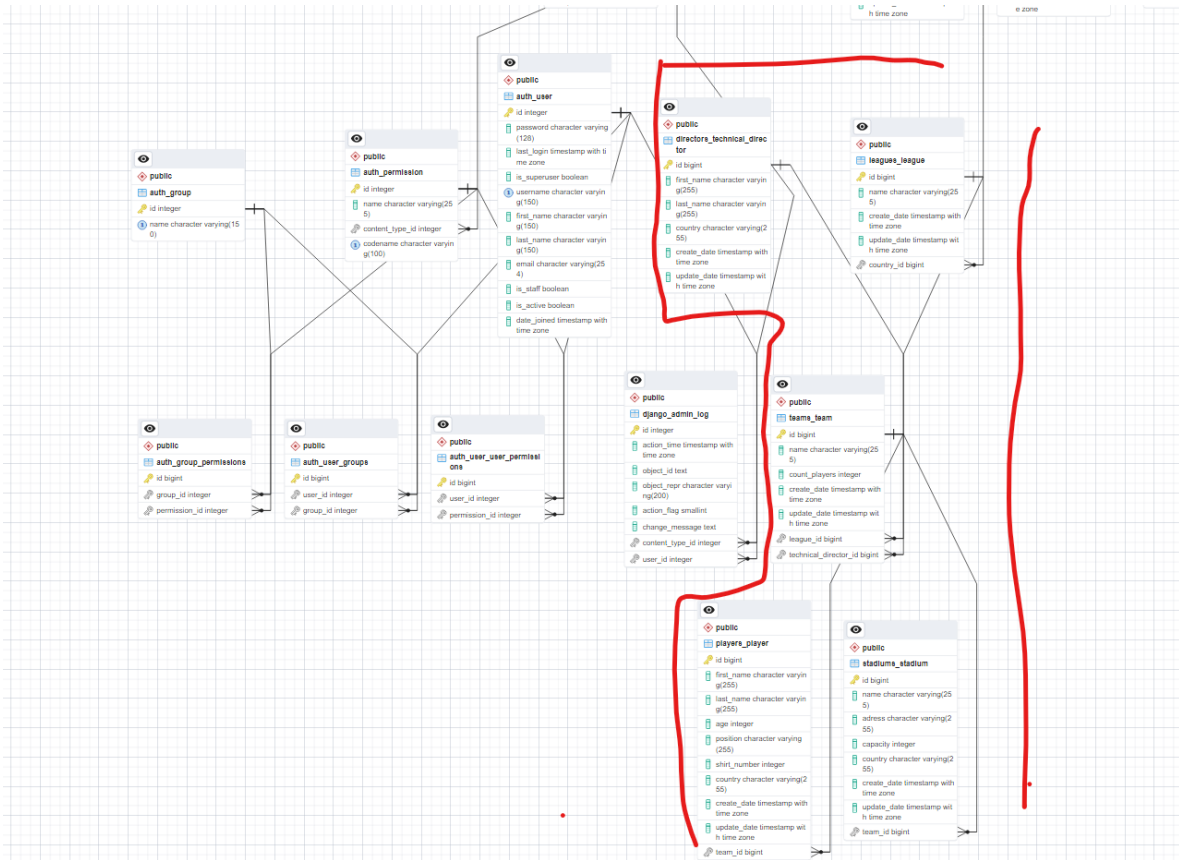
The image shows a screenshot of a Visual Studio Code editor. On the left, the 'EXPLORER' sidebar displays a file tree for a project named 'PRACTICA\_2'. The tree includes folders like '.vscode', 'countries', 'directors', 'futbol', 'leagues', 'players', 'stadiums', 'teams', and 'templates'. Files listed include '\_\_init\_\_.py', 'asgi.py', 'settings.py', 'urls.py', 'views.py', 'wsgi.py', '.env', '.gitignore', 'manage.py', 'Pipfile', and 'Pipfile.lock'. The 'futbol' folder is expanded, showing its contents. On the right, the 'settings.py' file is open, displaying Django configuration code. The code includes imports for 'django.contrib.messages', 'WSGI\_APPLICATION', database settings under 'DATABASES', and a comment for 'Password validation'. The code is syntax-highlighted with various colors.

```
82     'django.contrib.messages',
83 ],
84 },
85 },
86 ]
87
88 WSGI_APPLICATION = 'futbol.wsgi.application'
89
90
91 # Database
92 # https://docs.djangoproject.com/en/4.2/ref/settings/#databases
93
94 DATABASES = {
95     'default': {
96         'ENGINE': 'django.db.backends.postgresql',
97         'NAME': os.getenv('NAME'),
98         'USER': os.getenv('USER'),
99         'PASSWORD': os.getenv('PASSWORD'),
100        'HOST': os.getenv('HOST'),
101        'PORT': os.getenv('PORT'),
102    }
103 }
104
105
106 # Password validation
107 # https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators
```

3. 2 de las entidades debe relacionarse con otra

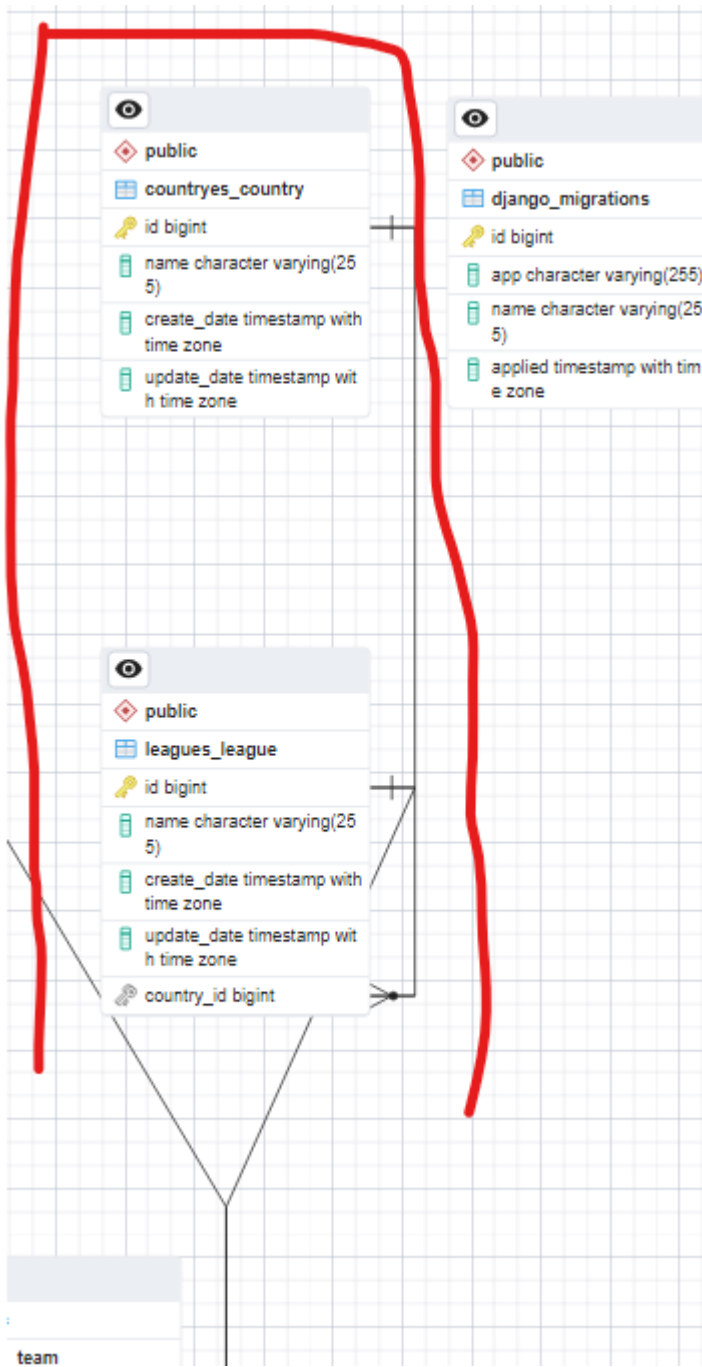
Se realizaron más de 2 relaciones entre las entidades

## Equipo 1 - Python



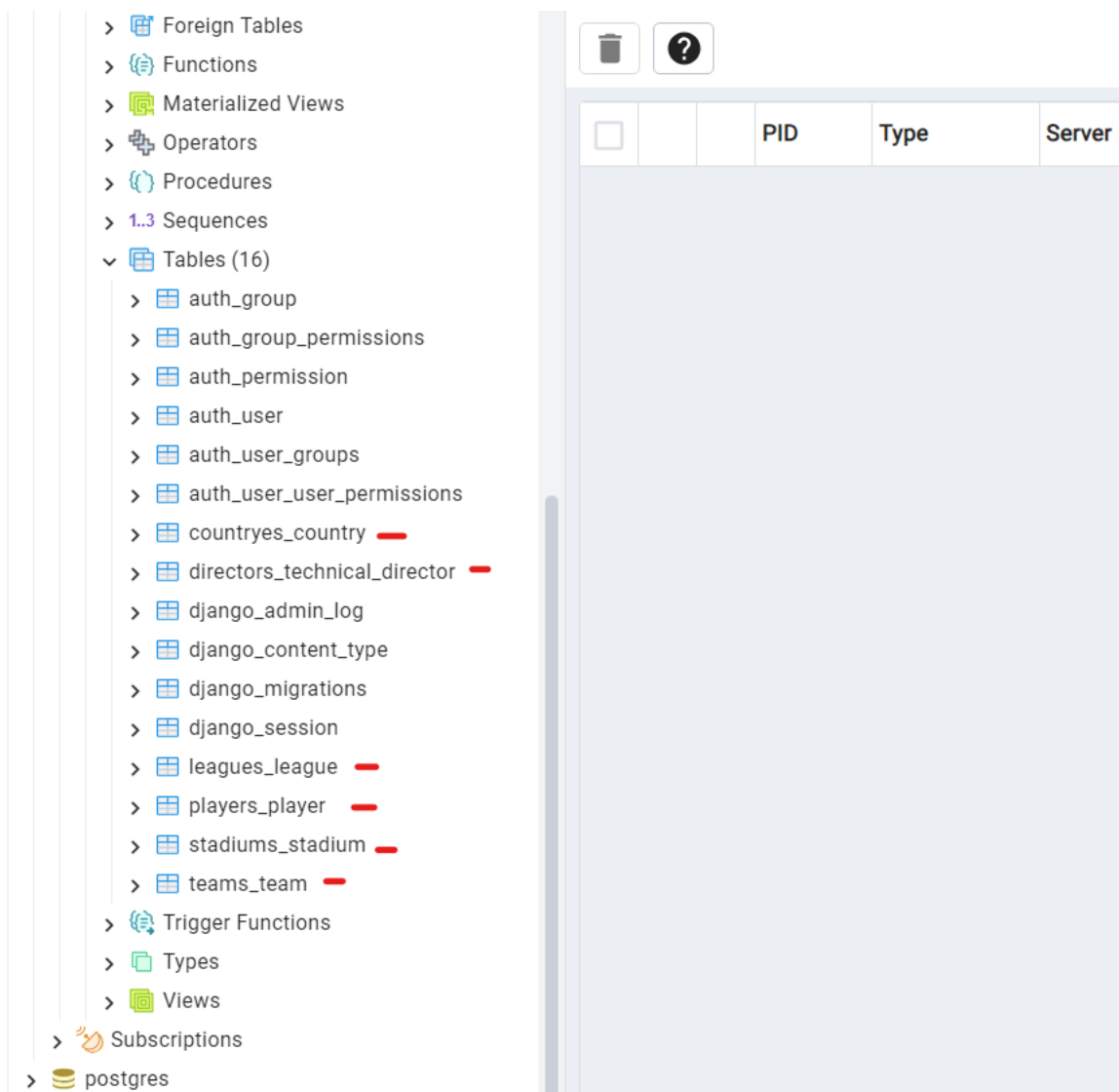


## Equipo 1 - Python



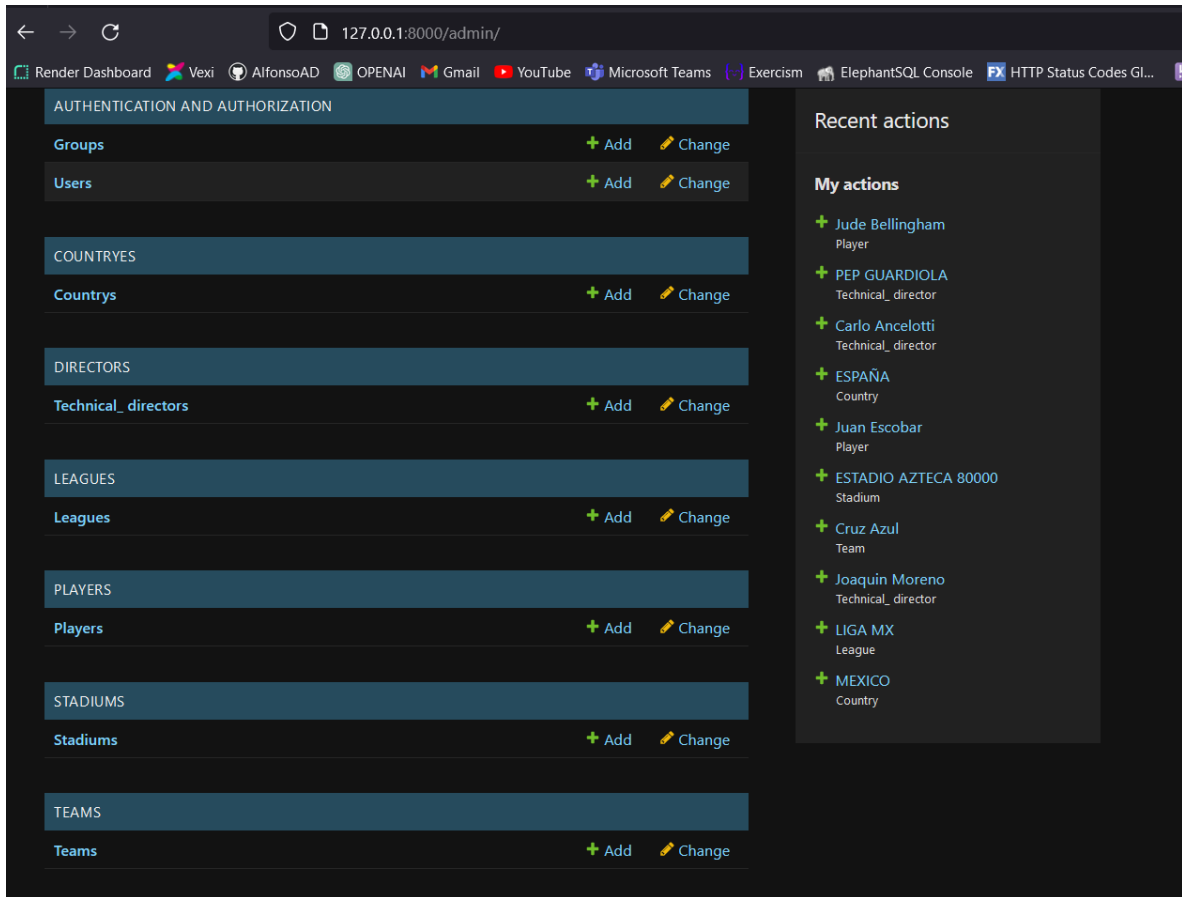
### 4. Realizar migraciones

## Equipo 1 - Python



5. Agregar las 6 entidades a la página de administración

## Equipo 1 - Python



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

The screenshot shows a database management interface with a sidebar on the left listing various database objects. The 'Country' table is selected. The main panel displays the table's structure and data.

id	name	create_date	update_date
1	MEXICO	2023-11-01 16:13:03-05	[null]
2	ESPAÑA	2023-11-01 17:16:01-05	[null]
3	ENGLAND	2023-11-03 22:23:17-05	[null]

## Technical directors

The screenshot shows a database management interface with a sidebar on the left listing various database objects. The 'Technical directors' table is selected. The main panel displays the table's structure and data.

id	first_name	last_name	country	create_date	update_date
1	3	PEP	GUARDIOLA	ESPAÑA	2023-11-01 18:10:45-05
2	4	THOMAS	TUCHEL	ALEMANIA	2023-11-01 19:20:20.587162-05
3	5	MARCELO	GALLARDO	ARGENTINA	2023-11-01 19:21:05.703967-05
4	2	CARLO	ANCELOTTI	ITALIA	2023-11-01 18:10:12-05
5	1	JOAQUIN	MORENO	MEXICO	2023-11-01 16:14:20-05

## Leagues

The screenshot shows a database management interface with a sidebar on the left listing various database objects. The 'Leagues' table is selected. The main panel displays the table's structure and data.

id	name	create_date	update_date	country_id
1	LIGA MX	2023-11-01 16:13:38-05	[null]	1
2	LIGA EXPANSION MX	2023-11-01 17:11:09.802774-05	[null]	1
3	LA LIGA SANTANDER	2023-11-01 17:16:28.721478-05	[null]	2

# Equipo 1 - Python

## Players

The screenshot shows a database management interface with a sidebar on the left listing various database objects. The main panel displays a SQL query and its results.

**Query:**

```
2 SELECT * FROM public.directors_technical_director;  
3 SELECT * FROM public.players_player;  
4
```

**Data Output:**

id	first_name	last_name	age	position	shirt_number	country	create_date
1	Juan	Escobar	25	DFC	24	PARAGUAY	2023-11-01 16:15:34-05
2	Jude	Bellingham	20	MC	5	ENGLAND	2023-11-01 18:43:04-05
3	Federico	Valverde	25	MC	15	URUGUAY	2023-11-01 18:53:04.890034-05
4	ROBERT	LEWANDOSKI	33	DC	9	POLONIA	2023-11-01 18:53:43.235938-05

## Stadiums

The screenshot shows a database management interface with a sidebar on the left listing various database objects. The main panel displays a SQL query and its results.

**Query:**

```
4 SELECT * FROM public.stadiums_stadium;  
5  
6
```

**Data Output:**

id	name	address	capacity	country	create_date	update_date	team_id
1	ESTADIO AZTECA	Reforma	80000	MEXICO	2023-11-01 16:14:50-05	[null]	1
2	SANTIAGO BERNABEU	PRIV X	80000	ESPAÑA	2023-11-03 22:27:09-05	[null]	2
3	CAMP NOU	PRIV Z	100000	ESPAÑA	2023-11-03 22:27:41-05	[null]	4

## Teams

## Equipo 1 - Python

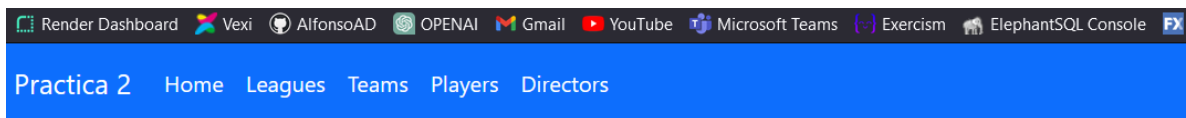
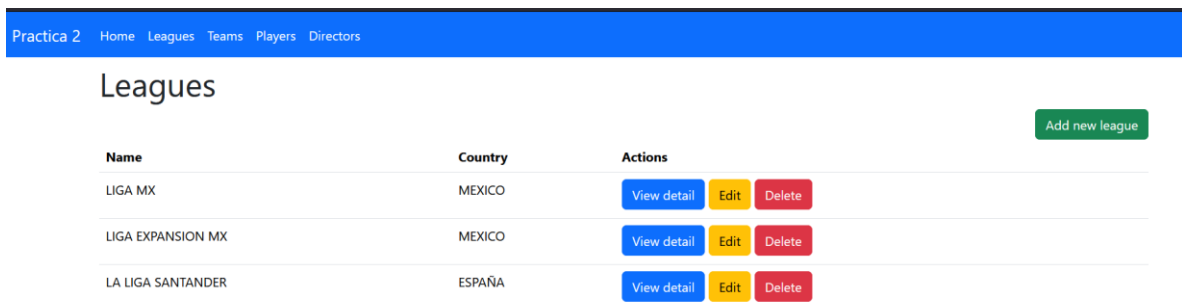
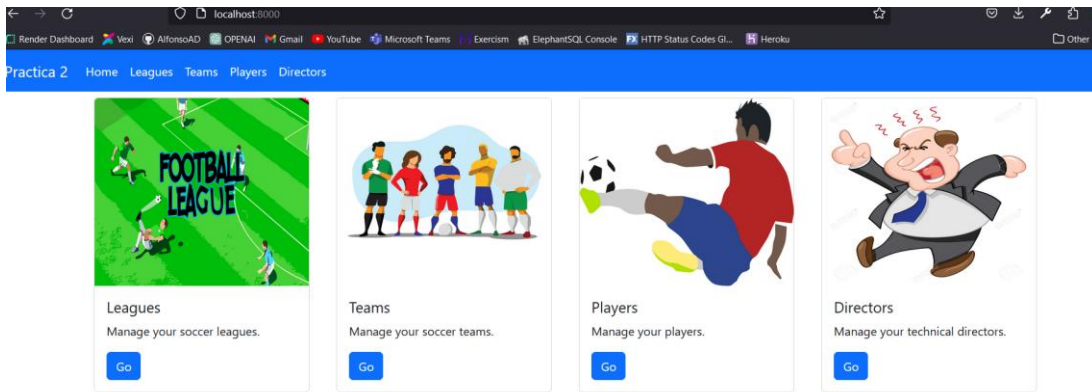
The screenshot shows a PostgreSQL database interface. On the left is the 'Explorer' pane with a tree view of database objects. The 'Tables (16)' folder is expanded, and 'countries\_country' is selected. The main pane shows a SQL query with three lines: `SELECT * FROM public.directors_technical_director;`, `SELECT * FROM public.players_player;`, and `SELECT * FROM public.teams_team;`. Below the query is the 'Data Output' tab, which displays a table with 8 columns: `id` (PK, bigint), `name` (character varying (255)), `count_players` (integer), `create_date` (timestamp with time zone), `update_date` (timestamp with time zone), `league_id` (bigint), and `technical_director_id` (bigint). The table contains three rows of data.

id	name	count_players	create_date	update_date	league_id	technical_director_id
1	REAL MADRID FC	34	2023-11-01 18:15:11.199323-05	[null]	3	2
2	BARCELONA	32	2023-11-01 18:51:10.228401-05	[null]	3	3
3	CRUZ AZUL	27	2023-11-01 16:13:59-05	[null]	1	1

7. Realizar el listado de al menos 3 entidades

Para acceder a los listados (4/6)

## Equipo 1 - Python



## League

Name

LIGA MX

Country

MEXICO

Create date

Nov. 1, 2023, 9:13 p.m.

[Return](#)

## Equipo 1 - Python

Practica 2 Home Leagues Teams Players Directors

### Edit League

Name:

Country:

[Cancel](#) [Save](#)

Practica 2 Home Leagues Teams Players Directors

### Create League

Name:

Country:

[Cancel](#) [Save](#)

Practica 2 Home Leagues Teams Players Directors

### Confirm

Want to delete the league LIGA MX?

[Cancel](#) [Yes i want](#)

Todos los formularios son similares con sus respectivos inputs.

Practica 2 Home Leagues Teams Players Directors

### Teams

[Add new team](#)

Name	Count players	Technical director	League	Actions
REAL MADRID FC	34	CARLO ANCELOTTI	LA LIGA SANTANDER	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>
BARCELONA	32	PEP GUARDIOLA	LA LIGA SANTANDER	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>
CRUZ AZUL	27	JOAQUIN MORENO	LIGA MX	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>



# Equipo 1 - Python

Practica 2 Home Leagues Teams Players Directors

## Players

Add new player

First name	Last name	Age	Team	Position	Shirt number	Country	Actions
Juan	Escobar	25		DFC	24	PARAGUAY	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>
Jude	Bellingham	20		MC	5	ENGLAND	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>
Federico	Valverde	25		MC	15	URUGUAY	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>
ROBERT	LEWANDOSKI	33		DC	9	POLONIA	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>

Practica 2 Home Leagues Teams Players Directors

## Technical directors

Add new director

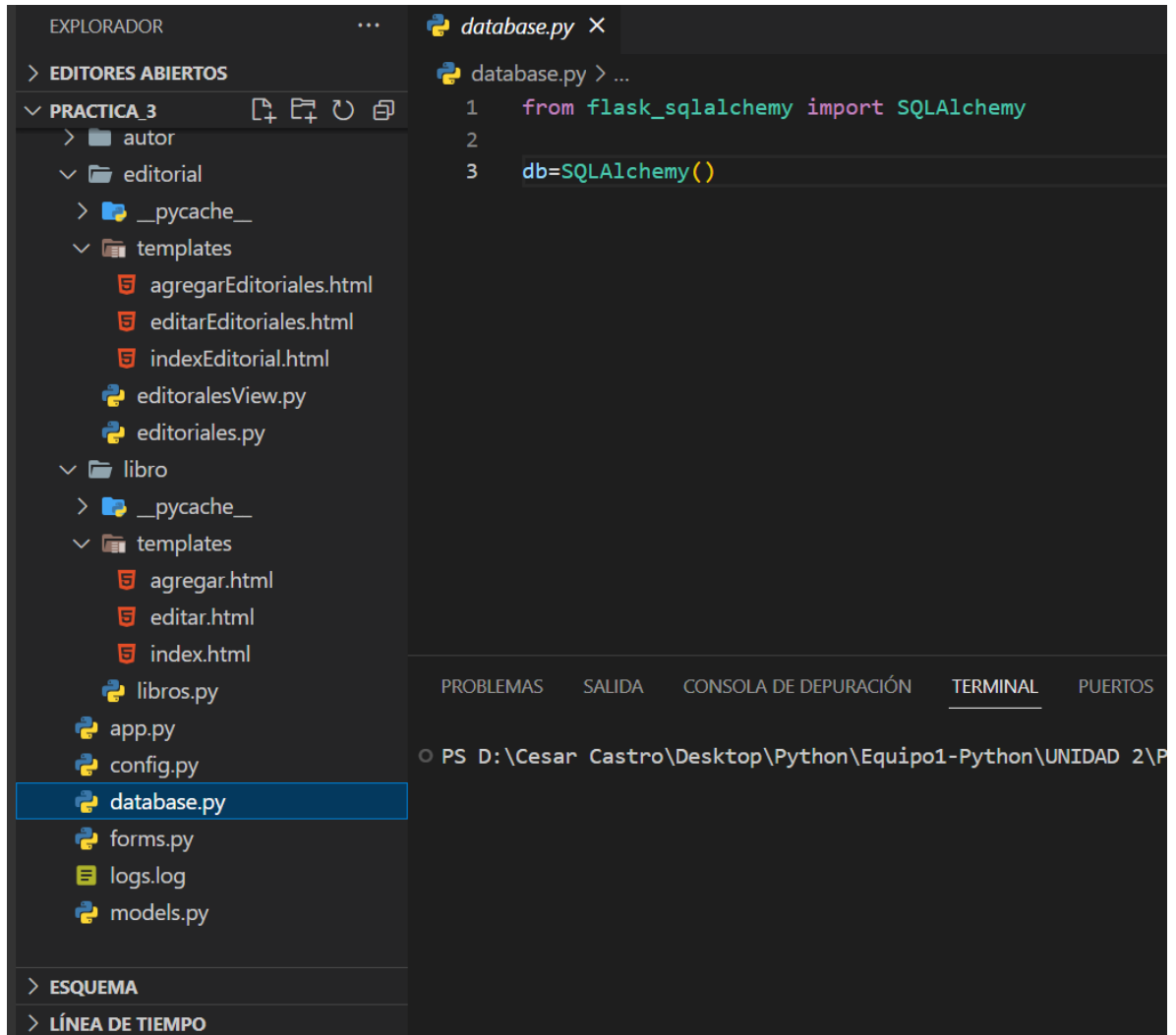
First name	Last name	Country	Actions
PEP	GUARDIOLA	ESPAÑA	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>
THOMAS	TUCHEL	ALEMANIA	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>
MARCELO	GALLARDO	ARGENTINA	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>
CARLO	ANCELOTTI	ITALIA	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>
JOAQUIN	MORENO	MEXICO	<a href="#">View detail</a> <a href="#">Edit</a> <a href="#">Delete</a>

## Equipo 1 - Python

### Práctica 3 (Flask)

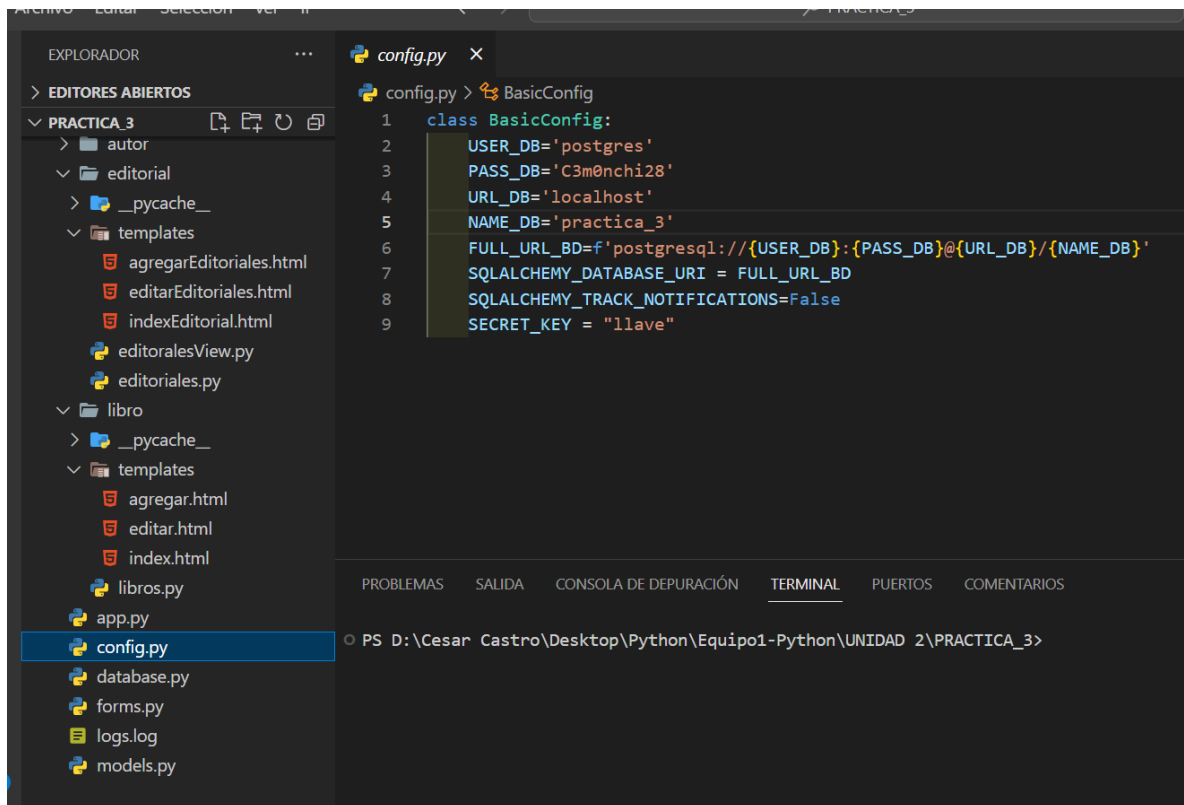
1. Conexión a base de datos postgresql con SQLAlchemy

Importación de SQLAlchemy para el manejo de la base de datos

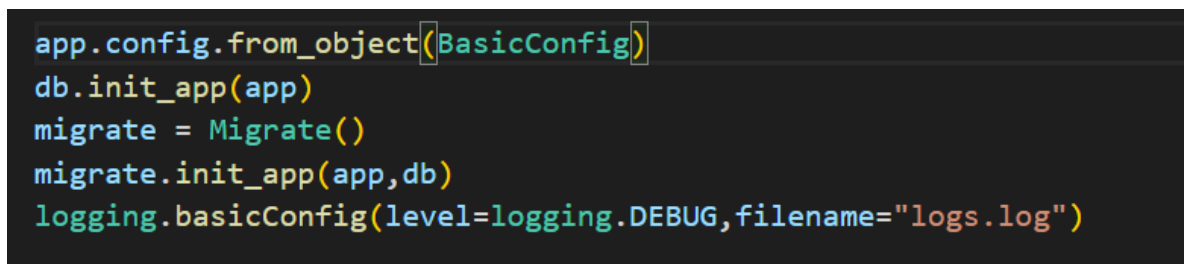


## Equipo 1 - Python

Estableciendo la liga de conexión de la base de datos



Realizamos la configuración en nuestra app

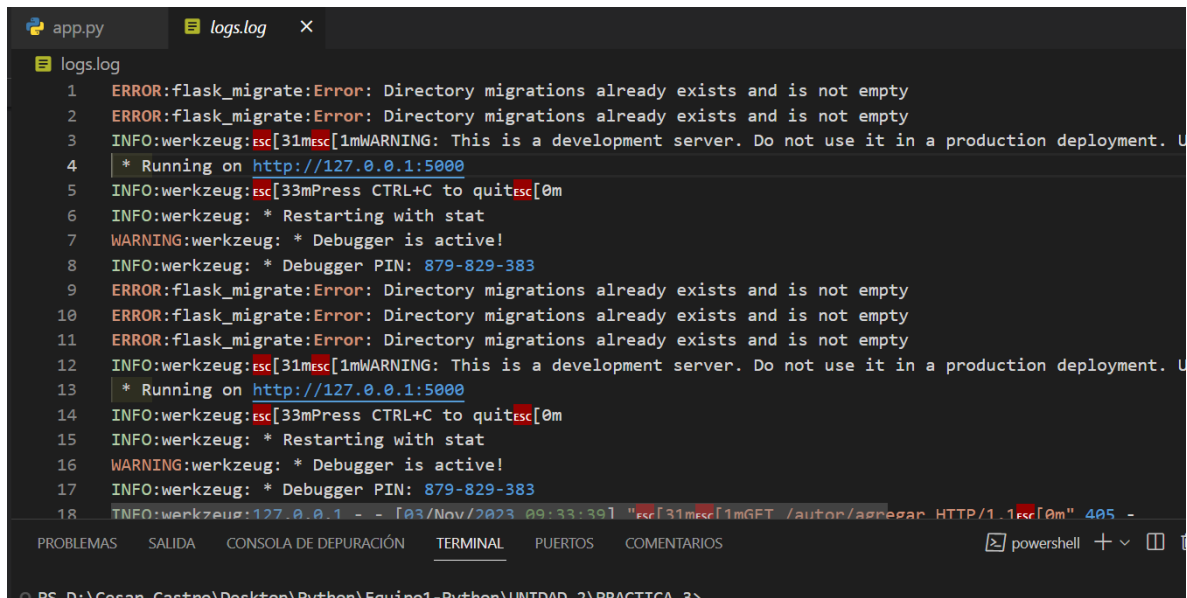


1. Utilizar app logging

En la imagen anterior se aprecia que usamos app login

## Equipo 1 - Python

App login corriendo



```
app.py logs.log x
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 -
```

2. Utilizar al menos 3 entidades

Creamos 3 modelos simulando una biblioteca

## Equipo 1 - Python

```
models.py > Libro
1  from app import db
2
3  class Autor(db.Model):
4      id = db.Column(db.Integer, primary_key=True)
5      nombre = db.Column(db.String(100), nullable=False)
6      nacionalidad = db.Column(db.String(50))
7
8  class Libro(db.Model):
9      id = db.Column(db.Integer, primary_key=True)
10     titulo = db.Column(db.String(200), nullable=False)
11     autor = db.Column(db.String(100), nullable=False)
12     editorial = db.Column(db.String(100), nullable=False)
13     year = db.Column(db.Integer)
14     genero = db.Column(db.String(50))
15
16     class Editorial(db.Model):
17         id = db.Column(db.Integer, primary_key=True)
18         nombre = db.Column(db.String(100), nullable=False)
19         pais = db.Column(db.String(50))
20         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
32     return str(get_engine().url).replace('%', '%%')
33
34
35     # add your model's MetaData object here
36     from models import Libro, Autor, Editorial
37     # for 'autogenerate' support
38     # from myapp import mymodel
39     # target_metadata = mymodel.Base.metadata
40     config.set_main_option('sqlalchemy.url', get_engine_url())
41     target_db = current_app.extensions['migrate'].db
42
```

## **Equipo 1 - Python**

CRUD HTTP con el modelo Autor

## Equipo 1 - Python

```
1  from flask import Blueprint,jsonify,request
2  from models import Autor
3  from app import db
4
5  appautor = Blueprint("appautor",__name__)
6
7  @appautor.route('/autor/agregar',methods=['POST'])
8  def agregarAutor():
9      try:
10         json = request.get_json()
11         autor=Autor()
12         autor.nombre=json['nombre']
13         autor.nacionalidad=json['nacionalidad']
14         db.session.add(autor)
15         db.session.commit()
16         return jsonify({"status":200,"mensaje":"Autor"})
17     except Exception as ex:
18         return jsonify({"status":400,"mensaje":ex})
19
20 @appautor.route('/autor/editar',methods=['POST'])
21 def editarAutor():
22     try:
23         json = request.get_json()
24         autor=Autor.query.get_or_404(json["id"])
```

## Equipo 1 - 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})

@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



## Equipo 1 - Python

The screenshot displays a REST client interface with a POST request to `http://127.0.0.1:5000/autor/agregar`. The request body is a JSON object with `"nombre": "Juan"` and `"nacionalidad": "Argentino"`. The response is a 200 OK status with a JSON body containing `"mensaje": "Autor"` and `"status": 200`.

**Request:**

```
POST http://127.0.0.1:5000/autor/agregar
```

**Body (JSON):**

```
{  "nombre": "Juan",  "nacionalidad": "Argentino"}
```

**Response:**

```
200 OK 118 ms 207 B
```

**Body (JSON):**

```
{  "mensaje": "Autor",  "status": 200}
```

## Equipo 1 - Python

The screenshot displays a REST client interface with the following details:

- URL:** `http://127.0.0.1:5000/autor/editar`
- Method:** `POST`
- Body (Request):** A JSON object with the following structure:

```
{  "id": "4",  "nombre": "Jose",  "nacionalidad": "Argentino"}
```
- Response:** A JSON object with the following structure:

```
{  "mensaje": "Autor modificado",  "status": 200}
```
- Status:** `200 OK`, `98 ms`, `218 B`
- Buttons:** `Save`, `Send`, `Beautify`, `Save as example`

## Equipo 1 - Python

The screenshot displays a REST client interface with a POST request to `http://127.0.0.1:5000/autor/eliminar`. The request body is a JSON object with the following structure:

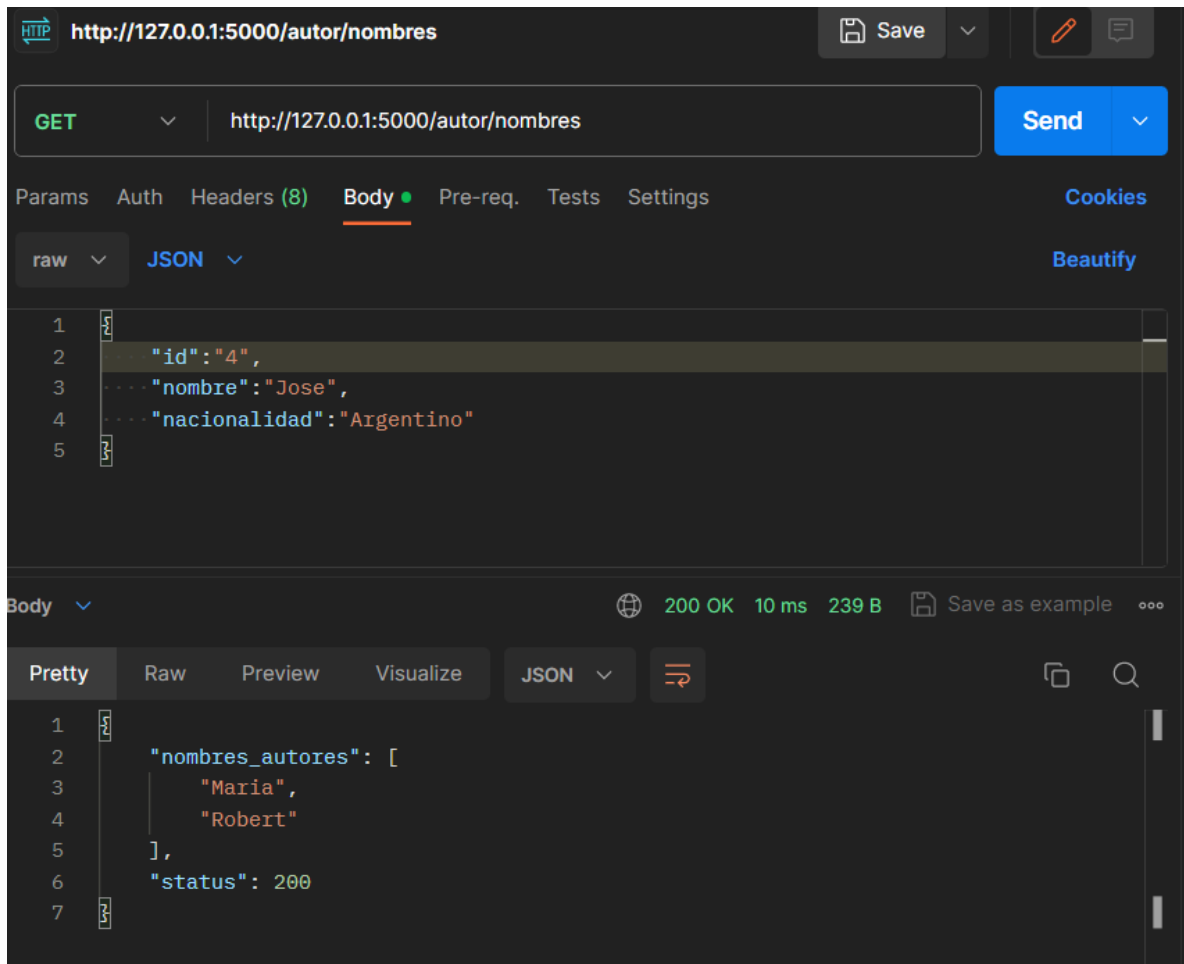
```
1 {
2   "id": "4",
3   "nombre": "Jose",
4   "nacionalidad": "Argentino"
5 }
```

The response status is `200 OK` with a response time of `28 ms` and a body size of `217 B`. The response body is a JSON object:

```
1 {
2   "mensaje": "Autor eliminado",
3   "status": 200
4 }
```

The interface includes tabs for Params, Auth, Headers (8), Body (selected), Pre-req., Tests, and Settings. The Body tab is further divided into raw and JSON (selected) views. A Beautify button is visible next to the JSON view. The response section includes a Pretty view (selected), Raw, Preview, and Visualize tabs, along with a JSON view selector and a copy icon.

## Equipo 1 - Python



Tambien se realizaron para el modelo de Editoriales

## Equipo 1 - 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

```
@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})

@appeditorial.route('/editorial/nombres', methods=['GET'])
def obtenerNombresEditoriales():
    try:
        editoriales = Editorial.query.all()
        nombres_editoriales = [editorial.nombre for editorial in editoriales]
        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

```
1 {  
2   "nombre": "Milenio",  
3   "pais": "Salvador",  
4   "telefono": "789456"  
5 }
```

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

```
1 {  
2   "id": "4",  
3   "nombre": "Milenio",  
4   "pais": "Chile",  
5   "telefono": "789456"  
6 }
```

Body ▼ 200 OK 25 ms 222 B Save as example ⋮

Pretty Raw **Preview** Visualize

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

## Equipo 1 - Python

**POST** ▼ `http://127.0.0.1:5000/editorial/eliminar` **Send** ▼

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

raw ▼ **JSON** ▼ Beautify

```
1 {
2   "id": "2",
3   "nombre": "Milenio",
4   "pais": "Salvador",
5   "telefono": "7894561230"
6 }
```

Body ▼ 200 OK 29 ms 221 B Save as example

Pretty Raw Preview Visualize **JSON** ▼

```
1 {
2   "mensaje": "Editorial eliminado",
3   "status": 200
4 }
```

**GET** ▼ `http://127.0.0.1:5000/editorial/nombres` **Send** ▼

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

raw ▼ **JSON** ▼ Beautify

```
1 {
2   "nombre": "Milenio",
3   "pais": "Salvador",
4   "telefono": "7894561230"
5 }
```

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.