



FACULTAD DE INGENIERÍA Y CIENCIAS
APLICADAS

Proyecto Integrador

Profesor:

Luis Patricio Moreno Buitrón

Autores:

Israel Eduardo Aguirre Yumiseba

Fernando David Lanas Yáñez

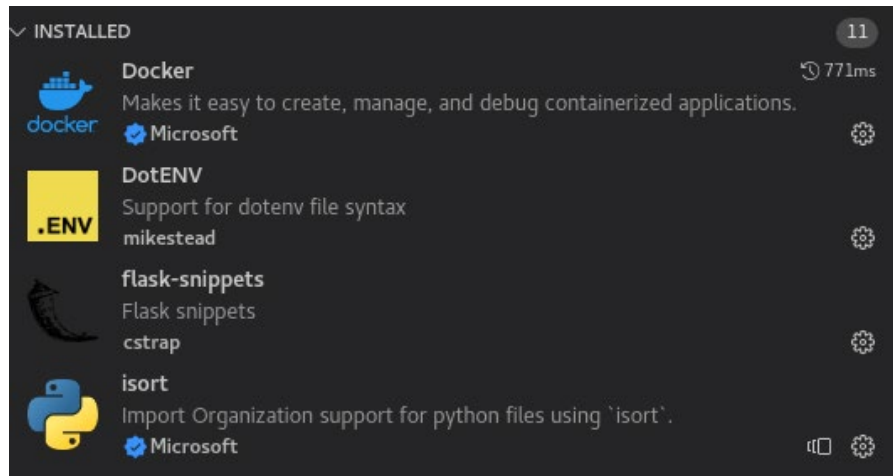
Brian Oliver Vásquez Samaniego

Fecha:

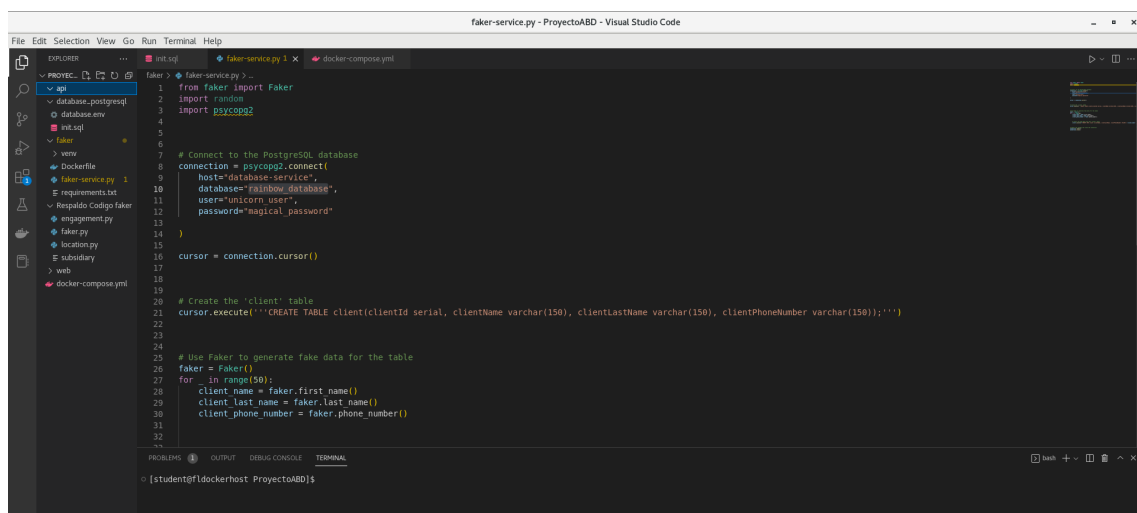
17 de enero de 2023

Se comenzó por instalar un editor de texto para facilitar la edición del texto en donde se colocarían los comandos de Python que alimentarían las bases de datos de información falsa.

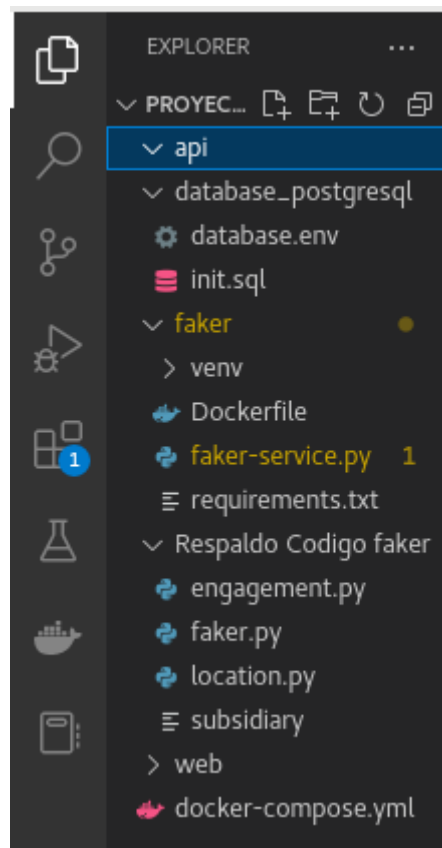
Es necesario instalar distintos complementos:



El complemento de Docker es el que nos permitirá crear archivos en donde podamos llamar al objeto Docker en el código de Python.



En el escritorio se creó una carpeta con una serie de archivos que permitiría levantar las instancias de Docker. La primera instancia de Docker sería una de postgresql y la otra con el servicio de faker en el framework de flask.



En el archivo Dockerfile contiene las siguientes líneas de código:

```
FROM python:3
COPY . /usr/src/app
COPY requirements.txt /usr/src/app/requirements.txt
WORKDIR /usr/src/app
RUN pip install -r requirements.txt
CMD ["python3", "faker-service.py"]
```

Estas líneas de código contienen una serie de instrucciones para crear el contenedor. El archivo de requirements.txt tiene los requerimientos necesarios para levantar el Docker con flask, faker y psycpg2.

```
click==8.0.4
Faker==14.2.1
Flask==2.0.3
greenlet==2.0.1
importlib-metadata==4.8.3
itsdangerous==2.0.1
Jinja2==3.0.3
MarkupSafe==2.0.1
psycpg2-binary==2.9.5
python-dateutil==2.8.2
six==1.16.0
SQLAlchemy==1.4.46
typing_extensions==4.1.1
Werkzeug==2.0.3
zipp==3.6.0
```

El archivo database.env contiene las credenciales por defecto con las que se va a entrar a la base de datos de PostgreSQL.

```
POSTGRES_USER=unicorn_user
POSTGRES_PASSWORD=magical_password
POSTGRES_DB=rainbow_database
```

El archivo docker-compose.yml es el archivo que levantara ambos containers al mismo tiempo. Este archivo los levanta utilizando los archivos mencionados anteriormente. De igual manera levanta los dos containers con su nombre respectivo y los puertos que estos dockers utilizaran

```
version: '3'

services:
  database-service:
    image: "postgres" # use latest official postgres version
    env_file:
      - ./database_postgresql/database.env # configurando postgres
    volumes:
      - ./database_postgresql/init.sql:/docker-entrypoint-initdb.d/init.sql # persistir la informacion
      - database-data:/var/lib/postgresql/data/
    ports:
      - "5432:5432"
    networks:
      - app-tier
    container_name: postgresql_service

  faker-service:
    restart: always
    build: ./faker
    volumes:
      - ./faker:/usr/src/app
    ports:
      - 5000:3000
    networks:
      - app-tier
    depends_on:
      - database-service
    container_name: servicio_datos_falsos

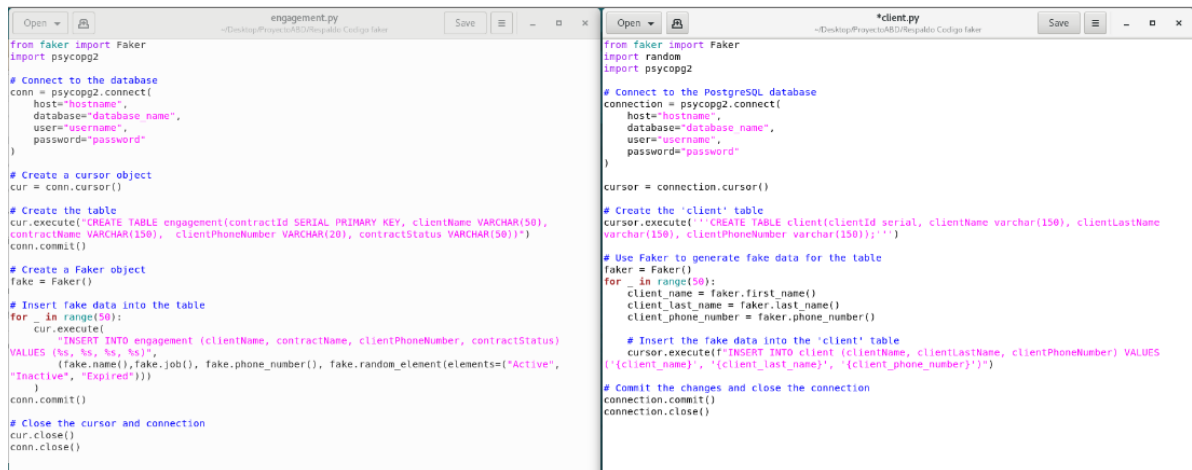
networks:
  app-tier:
    driver: bridge
volumes:
  database-data:
```

Para la población de datos, se debe crear un archivo inicial.init. Este archivo permitirá generar las tablas con faker en orden que los coloquemos.

```
create table client(clientId serial,
clientName varchar(150),
clientLastName varchar(150),
clientPhoneNumber varchar(150));
```

En este ejemplo, el archivo .init está creando la tabla de client. Este archivo debe contener los atributos de la tabla con su formato, su tipo de dato y la cantidad de caracteres que puede almacenar.

En los códigos de Python utilizados, es necesario colocar estos mismos atributos en el orden que se colocó en el archivo .init. De igual manera se debe importar las librerías de faker y pycpg2.



```
engagement.py
from faker import Faker
import pycpg2

# Connect to the database
conn = pycpg2.connect(
    host="hostname",
    database="database name",
    user="username",
    password="password"
)

# Create a cursor object
cur = conn.cursor()

# Create the table
cur.execute('CREATE TABLE engagement(contractId SERIAL PRIMARY KEY, clientName VARCHAR(50),
contractName VARCHAR(150), clientPhoneNumber VARCHAR(20), contractStatus VARCHAR(50))')
conn.commit()

# Create a Faker object
fake = Faker()

# Insert fake data into the table
for _ in range(50):
    cur.execute(
        "INSERT INTO engagement (clientName, contractName, clientPhoneNumber, contractStatus)
VALUES (%s, %s, %s, %s)",
        (fake.name(), fake.job(), fake.phone_number(), fake.random_element(elements=("Active",
"Inactive", "Expired")))
    )
conn.commit()

# Close the cursor and connection
cur.close()
conn.close()

client.py
from faker import Faker
import random
import pycpg2

# Connect to the PostgreSQL database
connection = pycpg2.connect(
    host="hostname",
    database="database name",
    user="username",
    password="password"
)

cursor = connection.cursor()

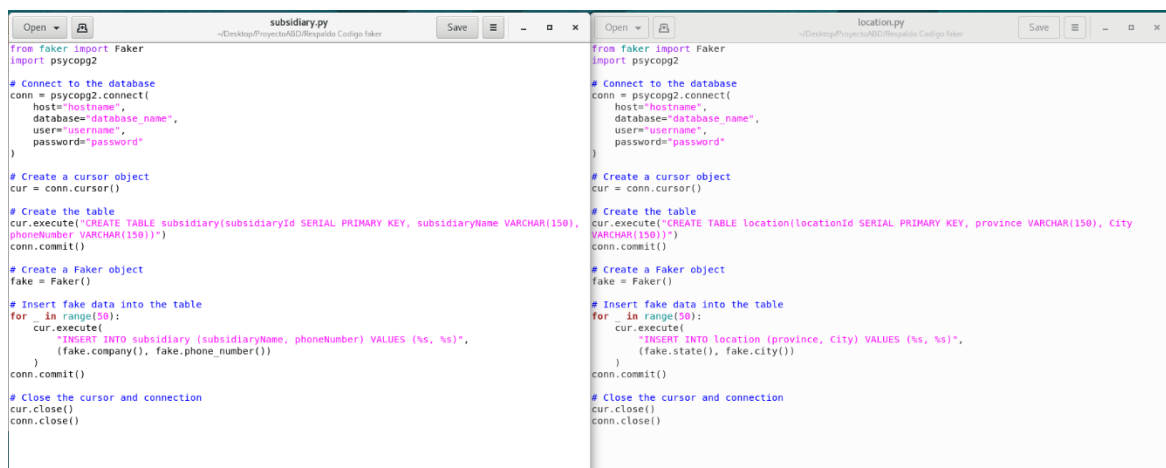
# Create the 'client' table
cursor.execute('CREATE TABLE client(clientId serial, clientName varchar(150), clientLastName
varchar(150), clientPhoneNumber varchar(150));')

# Use Faker to generate fake data for the table
fake = Faker()
for _ in range(50):
    client_name = fake.first_name()
    client_last_name = fake.last_name()
    client_phone_number = fake.phone_number()

    # Insert the fake data into the 'client' table
    cursor.execute("INSERT INTO client (clientName, clientLastName, clientPhoneNumber) VALUES
('client_name', 'client_last_name', 'client_phone_number')")

# Commit the changes and close the connection
connection.commit()
connection.close()
```

Ilustración 1: Código .py de la tabla engagement y de la tabla client



```
subsidiary.py
from faker import Faker
import pycpg2

# Connect to the database
conn = pycpg2.connect(
    host="hostname",
    database="database name",
    user="username",
    password="password"
)

# Create a cursor object
cur = conn.cursor()

# Create the table
cur.execute('CREATE TABLE subsidiary(subsidiaryId SERIAL PRIMARY KEY, subsidiaryName VARCHAR(150),
phoneNumber VARCHAR(150))')
conn.commit()

# Create a Faker object
fake = Faker()

# Insert fake data into the table
for _ in range(50):
    cur.execute(
        "INSERT INTO subsidiary (subsidiaryName, phoneNumber) VALUES (%s, %s)",
        (fake.company(), fake.phone_number())
    )
conn.commit()

# Close the cursor and connection
cur.close()
conn.close()

location.py
from faker import Faker
import pycpg2

# Connect to the database
conn = pycpg2.connect(
    host="hostname",
    database="database name",
    user="username",
    password="password"
)

# Create a cursor object
cur = conn.cursor()

# Create the table
cur.execute('CREATE TABLE location(locationId SERIAL PRIMARY KEY, province VARCHAR(150), City
VARCHAR(150))')
conn.commit()

# Create a Faker object
fake = Faker()

# Insert fake data into the table
for _ in range(50):
    cur.execute(
        "INSERT INTO location (province, City) VALUES (%s, %s)",
        (fake.state(), fake.city())
    )
conn.commit()

# Close the cursor and connection
cur.close()
conn.close()
```

Ilustración 2: Código .py de la tabla subsidiary y la tabla location

Para correr ambos dockers a la vez se utilizó el comando docker-compose up, que realmente llama al archivo docker-compose para levantar ambos dockers.


```
[student@fldockerhost ~]$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                    NAMES
5d5db16e45ed   postgres  "docker-entrypoint.s..." 45 hours ago   Up 22 minutes   0.0.0.0:5432->5432/tcp   postgresql_service
8a3e5fc25be9   proyectoabd_faker-service  "python3 faker-servi..." 45 hours ago   Restarting (1) 34 seconds ago
[student@fldockerhost ~]$ docker inspect 5d5db16e45ed | grep IPAddress
"SecondaryIPAddresses": null,
"IPAddress": "",
"IPAddress": "172.18.0.2",
[student@fldockerhost ~]$ ^C
[student@fldockerhost ~]$
```

La creación de un usuario con permisos tales como select, insert, drop, etc. Se los da en el usuario de super administrador, el modelo del Código es el siguiente

```
CREATE USER bvasconez WITH ENCRYPTED PASSWORD 'osamaniego';
```

```
GRANT SELECT, INSERT ON TABLE client TO bvasconez;
```

```
GRANT SELECT, INSERT ON TABLE engagement TO bvasconez;
```

```
GRANT SELECT, INSERT ON TABLE location TO bvasconez;
```

Para conectar el Usuario creado se lo hace en pgadmin, la siguiente imagen muestra las configuraciones realizadas

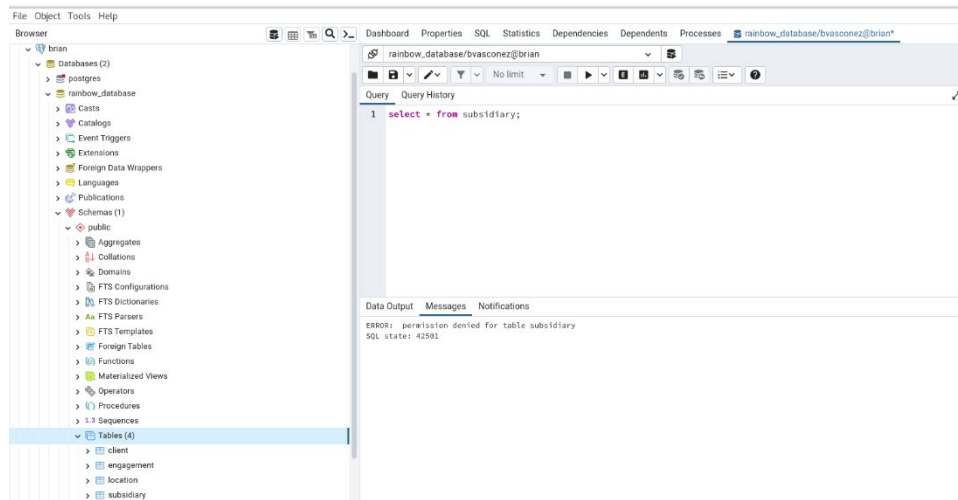
The screenshot shows the 'Register - Server' dialog box in pgAdmin, specifically the 'Connection' tab. The fields are filled as follows:

- Host name/addresses: 172.18.0.2
- Port: 5432
- Maintenance database: rainbow_database
- Username: bvasconez
- Password: [masked with dots]
- Kerberos authentication?: ☐ (unchecked)
- Save password?: ☐ (unchecked)
- Role: [empty field]
- Service: [empty field]

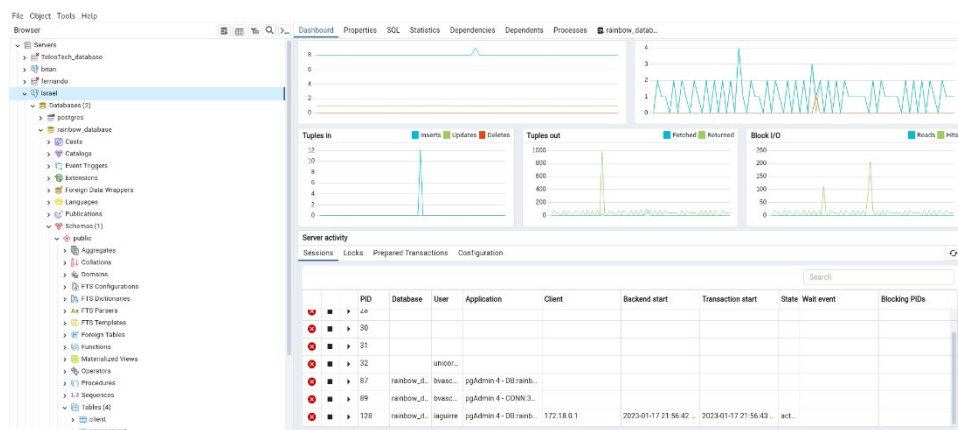
At the bottom of the dialog, there are three buttons: 'Close', 'Reset', and 'Save'.

Se necesita colocar el host del docker para la conexión, además del nombre de la base de datos, el Usuario que se creara y su contraseña

Para corroborar que se cuenta con los permisos especificados se va a intentar ingresar a la tabla “subsidiary” la cual no tiene permisos de acceso para la cuenta de “bvasconez”



Como se puede validar salta un error de permisos denegados para realizar un select a la tabla mencionada anteriormente



En pgadmin también cuenta con un dashboard que permite visualizar el número de perfiles que están activos, transacciones realiza, además se puede validar las acciones que fueron denegadas a cada usuario que no tiene permisos o para alterar las tablas o para ver su contenido