Tutorial Desarrollo de App

Lenguaje y Framework

Lenguaje: Python.
Frameworks: Flask, Tailwind.
Base De Datos: PostgreSQL

Librerías Utilizadas

Conexión a PostgreSQL: psycopg2
ORM: Flask_SQLAlchemy

Configuración y Levantamiento de la Base de Datos

En esta App usaremos PostgreSQL. Dentro del proyecto de debe tener un archivo **.env**, el contenido de este archivo debe ser similar a esto:

```
# Variables de entorno del proyecto
POSTGRES_USER=postgres
POSTGRES_PASSWORD=password
POSTGRES_DB=TUTORIAL
POSTGRES_HOST=db
POSTGRES_PORT=5432
FLASK_ENV=development
FLASK_APP=app.py
```

Este archivo es necesario para ejecutar el **Docker-Compose**

Contenido Del Archivo .yml

Para crear un contenedor de Docker en el que meteremos nuestra App (Para que pueda desplegarse desde cualquier máquina) es necesario tener unos archivos de configuración y ejecución de este contenedor, además de un archivo que indique los requerimientos para ejecutar la App. Los archivos de Docker que se usaron en esta App son los siguientes:

Proyecto > tutorial > docker-compose.yml

```yaml
version: "3.8"
services:
  db:
    image: postgres:15
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: tutorial
    volumes:
      - pgdata:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  web:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - .:/app
    environment:
      - POSTGRES_DSN=postgresql://postgres:password@db:5432/tutorial
      - POSTGRES_HOST=db
      - POSTGRES_PORT=5432
      - FLASK_ENV=development
      - FLASK_APP=app.py
    ports:
      - "5000:5000"
    depends_on:
      - db
    command: flask run --host=0.0.0.0 --port=5000

volumes:
  pgdata:

```

docker-compose.yml

```dockerfile
# Imagen base
FROM python:3.11-slim

# Esto es para evitar bytecode y bufferizado
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

WORKDIR /app

# Dependencias del sistema necesarias para psycopg2
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app", "--workers", "2", "--threads", "4"]
```

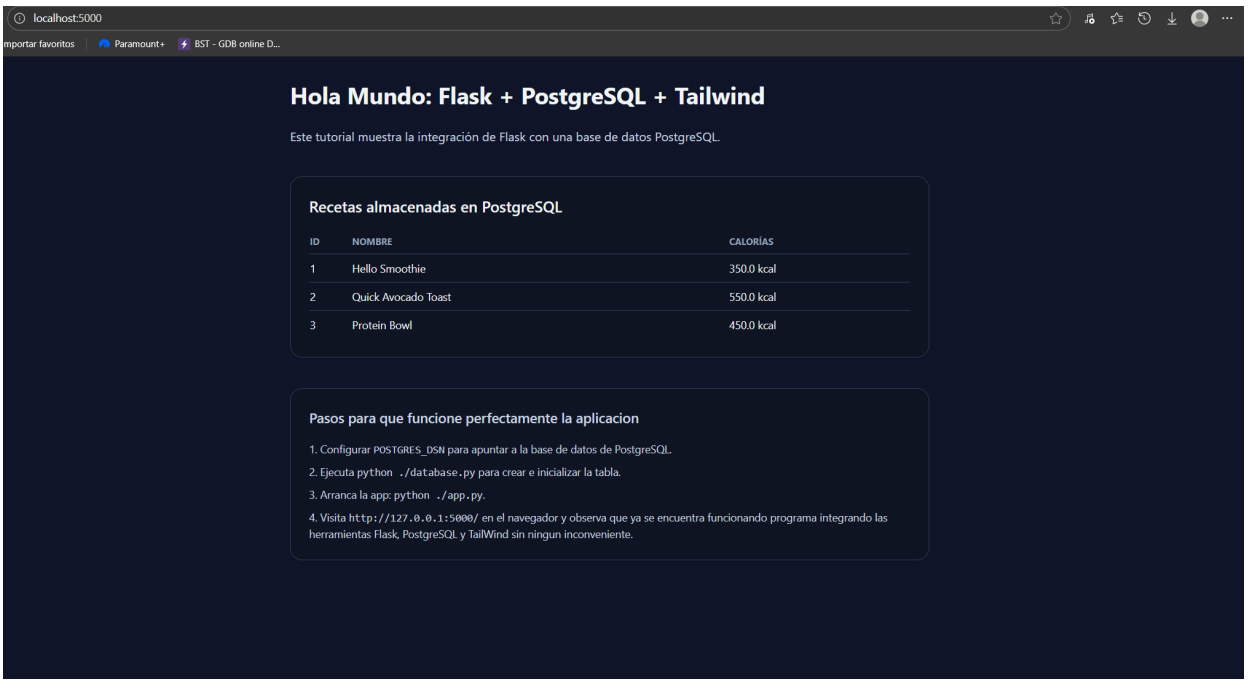
Dockerfile

```
Flask>=2.2
psycopg2-binary
gunicorn
```

Requerimientos del proyecto

Ejecución del “Hola Mundo”

Al poner “UP” al contenedor, luego de ejecutar el **docker-compose** se verá por el puerto en el que se ejecuta la App que esta ya estará corriendo.



Funcionamiento de la App. *Se mira la conexión con los datos que se almacenaron en* ***Postgresql.***

Instancia De Entidades

Actualmente es casi obligatorio usar programación orientada a objetos, por eso siempre es necesario instanciar entidades con sus respectivos atributos. Para esta App (tutorial) usaremos una sola entidad (clase), y con esta guardaremos y extraemos datos en la base de datos, ya que la manera más facil de comunicarse con una base de datos es por medio de entidades (tablas en las bases de datos).

```
30  @dataclass
31  class Recipe:
32      recipe_id: int
33      name: str
34      calories: float
35      fat: float
36      saturated_fat: float
37      cholesterol: float
38      sodium: float
39      carbohydrates: float
40      fiber: float
41      sugar: float
42      protein: float
43
```

Instanciación mínima de entidades. *La única entidad necesaria para esta prueba de la App fue la clase "recibe" (receta).*

```
CREATE_TABLE_SQL = """
CREATE TABLE tutorial_recipes (
    recipe_id SERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    calories DOUBLE PRECISION NOT NULL,
    fat DOUBLE PRECISION NOT NULL,
    saturated_fat DOUBLE PRECISION NOT NULL,
    cholesterol DOUBLE PRECISION NOT NULL,
    sodium DOUBLE PRECISION NOT NULL,
    carbohydrates DOUBLE PRECISION NOT NULL,
    fiber DOUBLE PRECISION NOT NULL,
    sugar DOUBLE PRECISION NOT NULL,
    protein DOUBLE PRECISION NOT NULL
);
"""
```

Entidad en la base de datos. *La tabla "tutorial_recipes" en la base de datos.*

```
def initialize_table() -> None:
    conn = get_connection()
    with conn.cursor() as cur:
        _ensure_schema(cur)


def insert_sample_data(samples: Iterable[tuple[str, float, float, float, float, float, float, float, float, float]]) -> None:
    conn = get_connection()
    with conn.cursor() as cur:
        for sample in samples:
            cur.execute(INSERT_SQL, sample)

```

Funciones para inicializar e insertar datos en la tabla.

Componentes VIsuales En La Interfaz De Usuario

En esta App se muestran los datos almacenados en la base de datos.

Encabezado y datos de la base de datos mostrados en la App

El HTML usado para hacer esto es el siguiente:

```
<section class="space-y-6">
    <h1 class="text-3xl font-bold">Hola Mundo: Flask + PostgreSQL + Tailwind</h1>
    <p class="text-slate-300">Este tutorial muestra la integración de Flask con una base de datos PostgreSQL.</p>
</section>
```

Texto (Encabezado) en la página principal de la App

```
<section class="bg-surface/90 border border-slate-700 rounded-2xl p-6 space-y-4">
    <h2 class="text-xl font-semibold">Recetas almacenadas en PostgreSQL</h2>
    <p class="text-sm text-slate-400"></p>
    <table class="w-full text-left text-sm">
        <thead class="text-slate-400 uppercase text-xs">
            <tr>
                <th class="py-2">ID</th>
                <th class="py-2">Nombre</th>
                <th class="py-2">Calorías</th>
            </tr>
        </thead>
        <tbody>
            {% for recipe in recipes %}
            <tr class="border-t border-slate-700">
                <td class="py-2">{{ recipe.recipe_id }}</td>
                <td class="py-2">{{ recipe.name }}</td>
                <td class="py-2">{{ recipe.calories }} kcal</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</section>
```

Datos extraídos de la base de datos

Ejecución

En Local: Ejecutar en consola, en el directorio -**/SmartBite/Proyecto/tutorial**:

pip install -r requirements.txt
python app.py

Luego se debe acceder a http://localhost:5000/

Con Docker Compose: Solo se debe ejecutar en el directorio -**/SmartBite/Proyecto/tutorial** el comando:

docker-compose up –build

Luego se debe acceder a http://localhost:5000/