Flask
Clase 31

Flask: Introducción

Les damos la bienvenida

Vamos a comenzar a grabar la clase

¿Qué es un API?

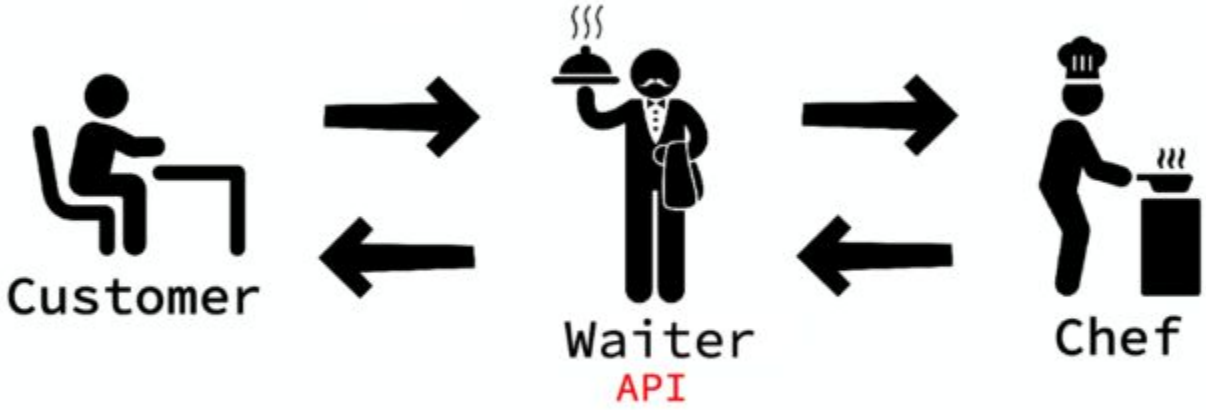
API es una abreviatura de Application Programing Interfaces, que en español significa interfaz de Programación de Aplicaciones.

Es un conjunto de definiciones y protocolos que que permite que dos programas se comuniquen entre sí. Es una forma de definir cómo se deben solicitar y enviar datos entre diferentes componentes de software. Puedes pensar en una API como un intermediario que permite que una aplicación hable con otra, solicitando ciertos servicios o datos y recibiendo respuestas en un formato específico.

ref: API - API REST

API receives a request

Similar to how a waiter takes an order from a customer to relay to the chef



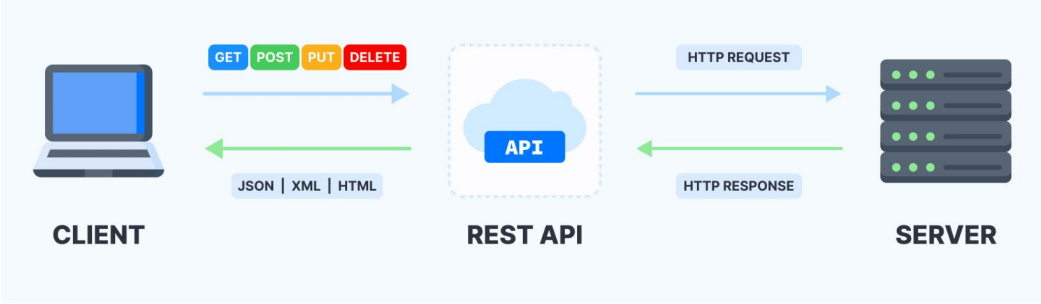
API collects and processes a response, then returns with that response

As a waiter would return the completed meal from the chef to the customer

API REST

REST significa transferencia de estado representacional. Una API REST es un tipo específico de API que sigue ciertas convenciones y principios arquitectónicos. Está diseñada para funcionar de manera eficiente en la web y utiliza los métodos estándar de HTTP (como GET, POST, PUT y DELETE) para realizar operaciones en los datos.

En términos aún más simples, puedes pensar en una API REST como una interfaz que te permite interactuar con los datos de una aplicación a través de la web utilizando solicitudes HTTP. Cada recurso (como un usuario o un artículo) tiene una URL única, y puedes realizar acciones sobre esos recursos utilizando los métodos de HTTP. Por ejemplo, puedes usar GET para obtener información, POST para agregar nueva información, PUT para actualizar información existente y DELETE para eliminarla.



¿Qué es Flask?

Flask es un microframework para Python que permite crear aplicaciones web de manera sencilla y rápida. Es ligero y flexible, ideal para proyectos pequeños y medianos.

Características:

- Microframework: es ligero y no incluye componentes adicionales por defecto. Esto permite a los desarrolladores elegir las bibliotecas y herramientas que necesitan. Por ej.: no incluye ORM o validación de formularios por defecto.
- Extensible: Permite agregar extensiones según las necesidades del proyecto.
- WSGI: Flask está basado en WSGI (Web Server Gateway Interface), que es un estándar para las aplicaciones web de Python. WSGI actúa como un intermediario entre el servidor web y la aplicación web.

Cómo funciona Flask

Rutas y Vistas:

- Rutas: Una ruta es una URL específica que se asocia con una función en la aplicación Flask. Estas funciones se denominan "vistas".
- Vistas: Las vistas son funciones Python que reciben solicitudes HTTP y devuelven respuestas HTTP. Las respuestas pueden ser de diferentes tipos, como HTML, JSON, texto plano, etc.

Contexto de Aplicación y Solicitud:

- Contexto de Aplicación: Contiene información sobre la configuración y el estado de la aplicación.
- Contexto de Solicitud: Contiene datos específicos de una solicitud HTTP, como datos de formularios, cookies y archivos.

ref: Python | endoflife.date

Cómo funciona Flask

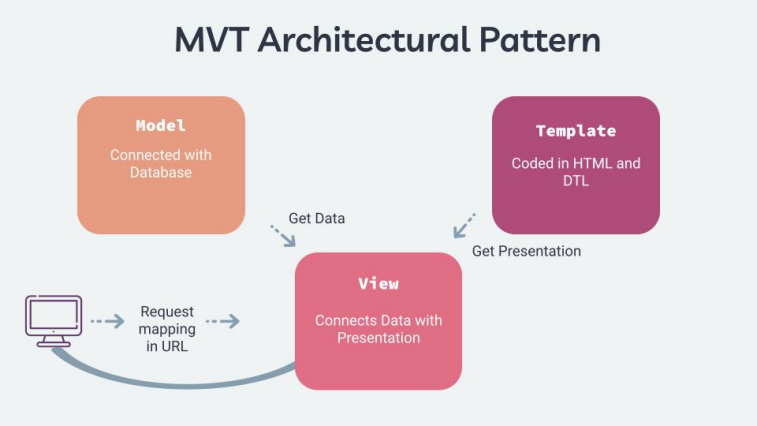
```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return '¡Hola, mundo!'

if __name__ == '__main__':
    app.run(debug=True)
```

Patrón MVT



Flask nos permitirá crear aplicaciones web rápidamente siguiendo un patrón de diseño MVT

Model: la capa de acceso a la base de datos (ORM).

View: la capa de la lógica de negocio.

Template: (Plantilla), la capa de presentación. En una API Rest no se implementa

Instalación y configuración

Creación de entorno virtual

Es un entorno aislado que permite instalar paquetes específicos para un proyecto sin afectar otros proyectos. La ventaja de utilizar un entorno virtual es que facilita la gestión de dependencias y evita conflictos entre versiones de paquetes.

python –m venv mientornovirtual

Activación entorno virtual

source path\mientornovirutual\Scripts\activate

Instalación de flask

pip install flask

Instalación y configuración

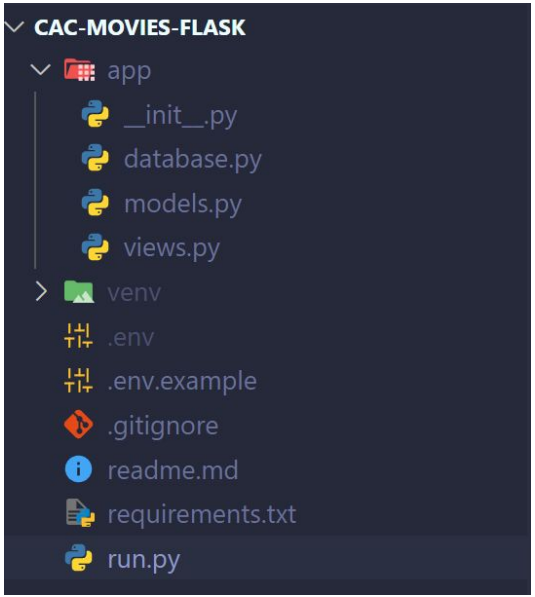
Creación de archivo requirements.txt

pip freeze > requirements.txt

Este comando nos generará un archivo requirements.txt que contendrá el listado de todas las librerías y sus versiones instaladas en nuestro virtual env para el proyecto. Esto es útil para que posteriormente podamos instalarlas directamente por medio de este archivo en cualquier otro entorno virtual donde llevemos al proyecto.

pip install -r requirements.txt

Estructura de directorios Flask



app/: Carpeta principal de la aplicación.

__init__.py: Un archivo vacío que le indica a Python que este directorio debería ser considerado como un paquete Python.

views.py: define las rutas y la logica de negocio de la aplicación.

models.py: define los modelos de datos que se implementaran en la aplicación.

database.py: Contiene la configuración de la base de datos.

venv/: entorno virtual

run.py: Script para ejecutar la aplicación.

requirements.txt: Archivo que contiene la lista de dependencias del proyecto.

.env: archivo que contiene la definición de las variables de entornos del proyecto.

Nuestra prima vista

Para completar el ejemplo anterior, en el archivo controllers.py de la aplicación deberemos crear una nueva función index, que retorna una respuesta del tipo JSON. Posteriormente en el archivo run.py, creamos nuestra aplicación Flask y asociamos la función del controlador con una ruta y un método HTTP.

app/views.py

```
from flask import jsonify

def index():
    return jsonify({'message':
'Hello World API TODO Lists'})
```

run.py

```
from flask import Flask
from app.views import *

app = Flask(__name__)

# Rutas de la API-REST
app.route('/', methods=['GET'])(index)

if __name__ == '__main__':
    app.run(debug=True)
```

Nuestra primera vista

Luego procedemos a correr el servidor de desarrollo de flask para probar nuestra API-Rest. Desde consola, posicionados en el directorio del proyecto, ejecutamos el siguiente comando

python run.py

En consola podremos ver que el servidor está a la espera de recibir peticiones, por medio de la ruta http://127.0.0.1:5000. Si se accede desde un navegador podremos ver el mensaje de bienvenida.

```
$ python run.py
 * Serving Flask app 'run'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a pr
oduction WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
```

Instalación y configuración – Variables de entorno

Las variables de entorno se utilizan para almacenar los secretos de la aplicación y los datos de configuración, que son recuperados por tu aplicación en ejecución cuando se necesitan. El valor de estas variables puede proceder de diversas fuentes: archivos de texto, gestores secretos de terceros, scripts de llamada, etc.

Instalación librería

pip install python-dotenv

Siempre que se instala un nueva librería, recordar de actualizar el archivo requirements.txt

pip freeze > requirements.txt

ref: Variables de entorno

Instalación y configuración – PostgreSQL

Instalación librería para conectar el proyecto con PostgreSQL
Se debe tener en cuenta que para instalar librerías se debe tener el entorno virtual activado, de otro modo la librería se instalará en el entorno global de Python.

pip install psycopg2

Creación de Base de datos
En una primera instancia vamos a tener que crear una base de datos desde el cliente de base de datos SQL que se esté usando (pgAdmin en nuestro caso). Creamos por ejemplo una base de datos llamada: **todo_list_24172** y la tabla **tareas** con el siguiente SQL.

```
CREATE TABLE IF NOT EXISTS Tareas (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(300) NOT NULL,
    activo BOOLEAN NOT NULL
);
```

Instalación y configuración – PostgreSQL

Creación de archivo .env
En la raíz del proyecto creamos el archivo .env que contendrá los datos de accesos a la base de datos de PostgreSQL con la que trabajaremos. Tiene que tener la siguiente estructura.

```
DB_USERNAME=postgres
DB_PASSWORD=tupassword
DB_HOST=127.0.0.1
DB_PORT=5432
DB_NAME=todo_list_24172
```

Creación del archivo app/database.py
En el directorio app, creamos el archivo database.py que contendrá las funciones para establecer la conexión con la base de datos de PostgreSQL.

```python
import os
import psycopg2
from flask import g
from dotenv import load_dotenv

# Cargar variables de entorno desde el archivo .env
load_dotenv()

# Configuración de la base de datos usando variables de entorno
DATABASE_CONFIG = {
    'user': os.getenv('DB_USERNAME'),
    'password': os.getenv('DB_PASSWORD'),
    'host': os.getenv('DB_HOST'),
    'database': os.getenv('DB_NAME'),
    'port': os.getenv('DB_PORT', 5432)
}
```

Instalación y configuración – PostgreSQL

database.py
Función de prueba para corroborar que podemos conectarnos correctamente a nuestra BBDD.

```python
def create_table_tareas():
    conn = psycopg2.connect(**DATABASE_CONFIG)
    cur = conn.cursor()
    cur.execute(
        """
        CREATE TABLE IF NOT EXISTS Tareas (
            id SERIAL PRIMARY KEY,
            nombre VARCHAR(50) NOT NULL,
            descripcion VARCHAR(300) NOT NULL,
            activo BOOLEAN NOT NULL
        );
        """
    )
    conn.commit()

    cur.close()
    conn.close()
```

```python
def test_connection():
    conn = psycopg2.connect(**DATABASE_CONFIG)
    cur = conn.cursor()
    conn.commit()
    cur.close()
    conn.close()
```

Agregamos un script para la creación de la tabla de tareas.

Instalación y configuración – PostgreSQL

Creación del archivo app/database.py

```
# Función para obtener la conexión a la base de datos
def get_db():
    # Si 'db' no está en el contexto global de Flask 'g'
    if 'db' not in g:
        # Crear una nueva conexión a la base de datos y guardarla en 'g'
        g.db = psycopg2.connect(**DATABASE_CONFIG)
    # Retornar la conexión a la base de datos
    return g.db

# Función para cerrar la conexión a la base de datos
def close_db(e=None):
    # Extraer la conexión a la base de datos de 'g' y eliminarla
    db = g.pop('db', None)
    # Si la conexión existe, cerrarla
    if db is not None:
        db.close()

# Función para inicializar la aplicación con el manejo de la base de datos
def init_app(app):
    # Registrar 'close_db' para que se ejecute al final del contexto de la
aplicación
    app.teardown_appcontext(close_db)
```

Instalación y configuración – PostgreSQL

Inicialización de la conexión de la base de datos con el proyecto
En el archivo run.py, importamos la función init_app del archivo app/database.py, de tal manera que podamos inicializar la conexión de la base de datos.

```
from flask import Flask
from app.database import init_app
from app.views import *

app = Flask(__name__)

# Inicializar la base de datos con la aplicación Flask
init_app(app)
```

No te olvides de completar la asistencia y consultar dudas

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.

TODO EN EL AULA VIRTUAL