Introducción a Flask

Proyecto Informático

¿Qué es?

Es un framework del lenguaje Python, utilizado para crear aplicaciones web completas.

> *Requiere el uso de otros paquetes y conocimientos adicionales, como JinJa y la generación de documentos HTML mediante templates.*

*En esta materia, haremos uso de una parte de esta herramienta, para enfocarnos en la implementación de **APIs RESTful**.*

La elección se basa en que no queremos limitar nuestras aplicaciones sólo a clientes web, sino que buscamos desarrollar un backend que pueda ser utilizado desde otro tipo de clientes, como aplicaciones móviles y de escritorio.

Antes de empezar

Como todo proyecto implementado en Python, es recomendable utilizar un entorno virtual para cada proyecto por separado.

La configuración de un nuevo entorno virtual es sencilla en una IDE como PyCharm, destinada exclusivamente a este lenguaje. En otros casos, como VSC, pueden ser necesarios algunos pasos adicionales para su correcta configuración, en particular si existe más de una versión de Python instalada en el sistema.

> *La elección de la IDE más cómoda queda a gusto y criterio de cada programador!*

Creación de un entorno virtual

1) *Creamos un directorio para el proyecto*
   > mkdir nombre_proyecto

2) *Nos movemos al directorio creado*
   > cd nombre_proyecto

3) *Creamos el entorno virtual*
   > py -3 -m venv .venv

4) *Activamos el entorno virtual*
   > .venv\Scripts\activate

En Linux:
$ mkdir myproject
$ cd myproject
$ python3 -m venv .venv
$ . .venv/bin/activate

Creación de un entorno virtual

<u>En caso de obtener un error en el paso anterior:</u>

Abrir VSC como Administrador

Modificar los permisos de usuario ejecutando el siguiente comando:

Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

Luego repetir el comando

> .venv\Scripts\activate

Instalación de paquetes

Sintaxis:

***pip install** nombre_paquete*

Instalación de flask:

pip install flask

Primer ejemplo: crear una instancia de Flask

archivo app.py

```
# Importamos la clase Flask
from flask import Flask

# Creamos una instancia
app = Flask(__name__)

# Ejecutamos el método run, indicando que estamos en modo de desarrollo y en qué puerto vamos a escuchar
app.run( debug = True, port = 5000)
```

Otra forma recomendada:

```
if __name__ == '__main__':
        app.run(debug=True, port=5000)
```

Primer ejemplo: ejecutar el script

*Desde la terminal, ejecutamos **python app.py***

En el navegador, escribimos localhost:5000

Obtendremos un mensaje

Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

*Esto no es un error. La respuesta indica que el servidor está funcionando, pero no se ha encontrado la **ruta**.*

Primer ejemplo: creación de rutas - sintaxis

Sintaxis:

```
@app.route('/ruta', methods = [<lista de métodos>] )

def funcion_para_la_ruta():

    return <respuesta de la app>
```

methods: lista de métodos HTTP permitidos para esa ruta, por defecto 'GET',

Primer ejemplo: creación de rutas - respuesta JSON

Creamos una ruta inicial:

```
@app.route('/'):
def index():
        return 'Index'
```

Como estamos diseñando una API RESTful, nuestras respuestas siempre serán en formato JSON.

```
        from flask import jsonify

@app.route('/')
def index():
   return jsonify({"message": "index"})
```

Primer ejemplo: creación de rutas - parámetros

Muchas veces necesitaremos obtener información a partir de la URL. Lo logramos mediante ***secciones variables*** *en la URL*

```
@app.route('/usuario/<nombre>')
def usuario(nombre):
    return jsonify({"Su nombre es": nombre })
```

Se indican mediante <nombre_variable> y deben ser pasadas como argumento a la función correspondiente a la ruta.

Primer ejemplo: creación de rutas - parámetros

Se puede especificar un tipo con la sintaxis <converter: nombre_variable>

```
@app.route('/cliente/<int:cliente_id>')
def cliente_int(cliente_id):
    return jsonify({"Su id de cliente es": cliente_id })


@app.route('/cliente/<string:cliente_name>')
def cliente_str(cliente_name):
    return jsonify({"Su nombre de cliente es": cliente_name })
```

Primer ejemplo: creación de rutas - parámetros

Opciones posibles de converter

string	(default) cualquier texto sin slash (/)
int	enteros positivos
float	valores positivos en punto flotante
path	como string, pero admite slash
uuid	para strings UUID

UUID: identificador único universal, compuesto por 32 dígitos hexadecimales (128 bits), separados en grupos (8-4-4-4-12)

Primer ejemplo: creación de rutas -escape de cadenas

Al permitir cadenas en la url, es posible inyectar código malicioso hacia el cliente.

```
@app.route('/<path:nombre>')
def danger(nombre):
    return nombre
```

Escribir en el navegador:

http://127.0.0.1:5000/%3Cscript%3Ealert('codigo')%3C/script%3E

(<script> alert(‘codigo’) </script>)

Primer ejemplo: creación de rutas -escape de cadenas

Para proteger estas rutas, se utiliza un escape de cadenas

```
from markupsafe import escape

@app.route('/<path:nombre>')
def danger(nombre):
    return escape(nombre)
```

El paquete markupsafe (y varias otras dependencias) se instala junto a flask

Primer ejemplo: creación de rutas - métodos HTTP

Pueden especificarse en el método route, mediante el parámetro ***methods***

```
@app.route('/recurso', methods=['GET'])
def get_recurso():
    return jsonify({"recurso": "lista de todos los recursos"})

@app.route('/recurso', methods=['POST'])
def post_recurso():
    return jsonify({"recurso": "creando un nuevo recurso"})
```

Primer ejemplo: creación de rutas - métodos HTTP

O también con el método específico de la clase

```
@app.get('/producto')
def get_producto():
    return jsonify({"producto": "app.get"})

@app.post('/producto')
def post_producto():
    return jsonify({"producto": "app.post"})

@app.delete('/producto')
def delete_producto():
    return jsonify({"producto": "app.delete"})
```

Primer ejemplo: objeto request

En los casos que necesitemos recibir información a través de la petición, podemos recuperar los datos enviados por el cliente mediante el objeto request.

from flask import request

```
@app.route('/persona', methods = ['POST'])
def post_persona():
    print(request.get_json())
    return jsonify({"recibido": request.get_json()})
```

Primer ejemplo: objeto request

A través del objeto request podemos controlar si los datos recibidos son correctos:

```
@app.route('/servicio', methods = ['POST'])
def post_servicio():
    data = request.get_json()
    if not ("name" in data.keys()):
        return jsonify({"error": "falta el nombre"})
    return jsonify({"recibido": request.json})
```

Primer API

Ahora crearemos nuestra primera API RESTful.

Para simplificar y comprender inicialmente los conceptos más importantes, no utilizaremos aún una base de datos. En su lugar, cargamos datos de prueba en un objeto.
El recurso sobre el cual planteamos el ejemplo es un objeto que describe información sobre una persona:

person:{
"name": "String",
"surname" : "String",
"email": "String",
"dni": int,
"id": int
}

Primer API

Queremos implementar un CRUD para este recurso, por lo que definimos las rutas:

GET /persons -> Lista todas las personas

POST /persons -> Crea una nueva persona	*(C) Create*
GET /persons/<int:id> -> Lista una persona por su id	*(R) Read*
PUT /persons/<int:id> -> Actualiza una persona por su id	*(U) Update*
DELETE /persons/<int:id> -> Elimina una persona por su id	*(D) Delete*

Primer API

*** *Desarrollo en clase* ***

Conexión a base de datos

La conexión a una base de datos generalmente se realizará mediante una extensión conveniente, lo que simplifica notablemente la implementación.

Una de los toolkits más populares para este objetivo es SQLAlchemy

https://www.sqlalchemy.org/

Sin embargo, esta herramienta propone una manipulación de la base de datos a partir de una sintaxis "pyhtonica", la cual es apropiada en caso de no tener conocimientos específicos de SQL.

Conexión a base de datos

En este curso haremos uso de los conocimientos previos de SQL, por lo que la extensión a utilizar será flask-mysqldb

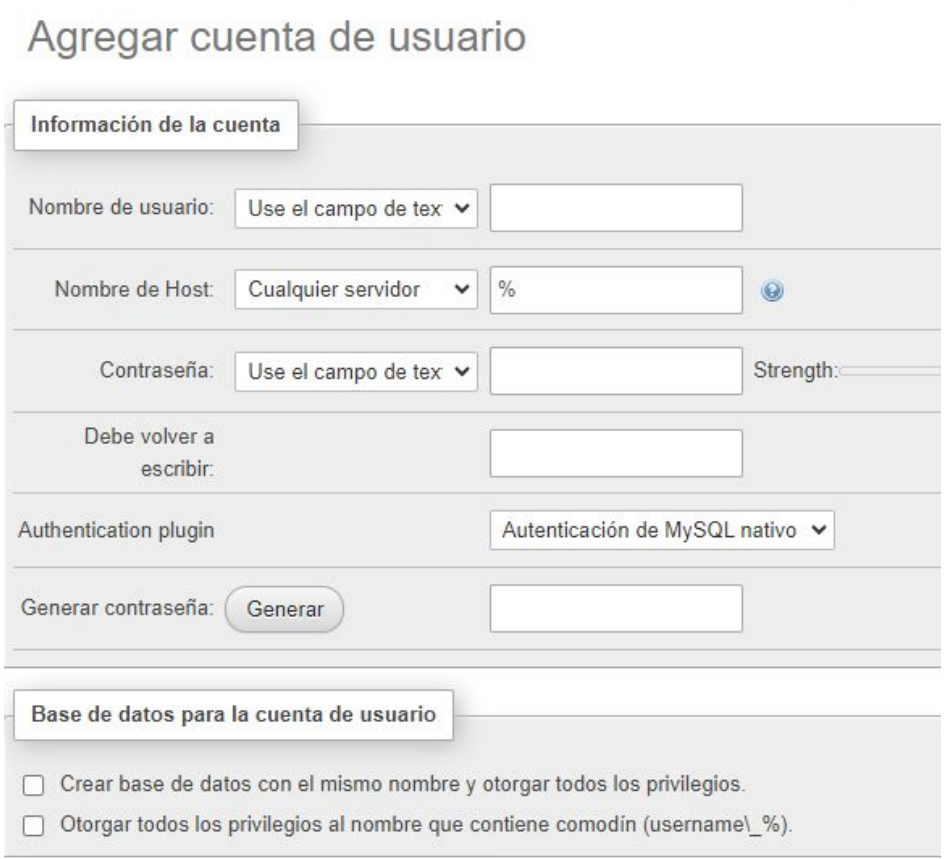
pip install flask_mysqldb

Conexión a base de datos: cuenta de usuario

http://localhost/phpmyadmin/

-> Cuentas de usuarios

-> Agregar cuenta de usuario



Conexión a base de datos: cuenta de usuario

Definimos los privilegios globales

Conexión a base de datos: base de datos

```
CREATE DATABASE IF NOT EXISTS db_flask_1;
USE db_flask_1;
CREATE TABLE IF NOT EXISTS person(
    id INT(10) NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    surname VARCHAR(255) NOT NULL,
    dni INT(8) NOT NULL,
    email VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
)
INSERT INTO person VALUES
(1, 'Juan', 'Álvarez' , 12345678, 'juan@mail.com'),
(2, 'Ana', 'Perez' , 87654321, 'ana@mail.com');
```

Primer ejemplo con BD

```
from flask import Flask, jsonify, request
from flask_mysqldb import MySQL
app = Flask(__name__)

app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'user_flask'
app.config['MYSQL_PASSWORD'] ='pass_flask'
app.config['MYSQL_DB'] = 'db_flask_1'

mysql = MySQL(app)

if __name__ == '__main__':
    app.run(debug=True, port=4500)
```

Información correspondiente a la base de datos y cuenta de usuario que usaremos en la aplicación

Primer ejemplo con BD

GET /persons

```
@app.route('/persons')
def get_persons():
    cur = mysql.connection.cursor()
    cur.execute('SELECT * FROM person')
    data = cur.fetchall()
    print(data)
    return jsonify({"persons":data})
```

Primer ejemplo con BD POST /persons

```
@app.route('/persons', methods = ['POST'])
def add_person():
    name = request.json["name"]
    surname = request.json["surname"]
    email = request.json["email"]
    dni = request.json["dni"]
    cur = mysql.connection.cursor()
    cur.execute('INSERT INTO person (name, surname, dni, email) VALUES (%s,
%s, %s, %s)',
            (name,surname,dni,email))
    mysql.connection.commit()
    return jsonify({"persons":persons})
```

Primer ejemplo con BD GET/persons/id

```
@app.route('/persons/<int:id>', methods = ['GET'])
def get_person_by_id(id):
    cur = mysql.connection.cursor()
    cur.execute('SELECT * FROM person WHERE id = {0}'.format(id))
    data = cur.fetchall()
    if len(data)>0:
        return jsonify({"person": data[0]})
    return jsonify({"message": "id not found"})
```

Primer ejemplo con BD POST /persons

```
@app.route('/persons/<int:id>', methods = [DELETE])
def delete_person(id):
    name = request.json["name"]
    surname = request.json["surname"]
    email = request.json["email"]
    dni = request.json["dni"]
    cur = mysql.connection.cursor()
    cur.execute('INSERT INTO person (name, surname, dni, email) VALUES (%s, %s, %s, %s)',
                (name,surname,dni,email))
    mysql.connection.commit()
    return jsonify({"persons":persons})
```