Taller Flask

1. Despliegue de una API

Para este apartado vamos a realizar un primer despliegue de un servicio web que simplemente contenga un mensaje. Para ello hay que seguir los siguientes pasos:

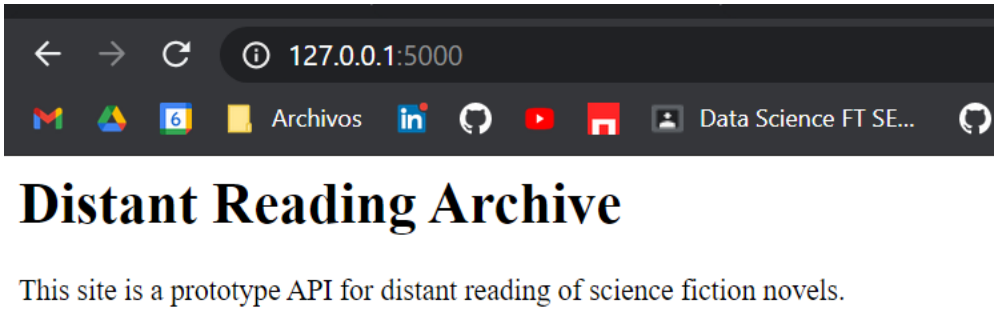
1. Busca un lugar en tu ordenador y crea una carpeta de "demo_clase", donde alojarás todos los proyectos de estos tutoriales.
2. Crea un archivo de pyton llamado "app1.py"
3. Introduce el siguiente código. Primero se crea la configuración de la API Flask. Después viene declarado el único tipo de petición, que sería un GET a la URL del servicio raíz, el cual devuelve una página web, un HTML, con un par de frases. Finalmente, se queda corriendo la app mediante "app.run()".

```
import flask

app = flask.Flask(__name__)
app.config["DEBUG"] = True


@app.route('/', methods=['GET'])
def home():
  return "<h1>Distant Reading Archive</h1><p>This site is a prototype API for distant reading of science fiction novels.</p>"

app.run()
```

4. Para ejecutar esto, ve a un “Simbolo del sistema”, “powershell” (o terminal si estás en MAC) y dentro de la carpeta “/demo_clase”, ejecuta "python app1.py". Esto lanzará el servicio en local. Accede al mismo mediante la URL sugerida por el output de la sentencia (lo normal es que sea http://127.0.0.1:5000/).

Para que el servicio deje de correr: CTRL + C

Este debería ser el output:



Fíjate en el terminal cada vez que accedes a esa URL. El servicio está recibiendo peticiones, como si un usuario estuviera accediendo a una web.

```
(base) C:\Users\Daney>python "C:\Users\Daney\Desktop\Archivos\Material\thebridge_dsmat
APIs en Flask\demo_clase\app1.py"
 * Serving Flask app "app1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 531-950-851
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [20/Dec/2020 17:23:20] "[37mGET / HTTP/1.1[0m" 200 -
127.0.0.1 - - [20/Dec/2020 17:23:21] "[37mGET / HTTP/1.1[0m" 200 -
127.0.0.1 - - [20/Dec/2020 17:23:22] "[37mGET / HTTP/1.1[0m" 200 -
```

¿Qué está haciendo Flask? Está mapeando la URL "/", con la función "home()", por lo que cada vez que se acceda a la URL "/", es decir, sin ninguna ruta extra, llamará a la función "home()" y devolverá su output, que en este caso es un texto HTML, pero podría ser un JSON con otros datos.

El proceso de mapear URLs a funciones se denomina **routing**.
@app.route('/', methods=['GET'])

Utiliza como methods, el GET, que es la acción HTTP para que el servidor devuelva datos al usuario. Se suele combinar mucho con POST, para recibir datos del usuario.

app.config["DEBUG"] = True
Se utiliza para que salten los errores en la página y podamos ver bien qué es. Si no, pondría un "Bad Gateway".

2. API con datos

Como el propósito de este taller es montar una API con la que podamos acceder a una base de datos de libros, comenzaremos creando algunos datos sintéticos, así como un enrutado a los mismos:

1. Importa:
   from flask import request, jsonify

2. Declara una lista con varios diccionarios. Éstos serán los datos que devolverá la API
   books = [
   {'id': 0,
   'title': 'A Fire Upon the Deep',
   'author': 'Vernor Vinge',
   'first_sentence': 'The coldsleep itself was dreamless.',
   'year_published': '1992'},
   {'id': 1,

```
    'title': 'The Ones Who Walk Away From Omelas',
    'author': 'Ursula K. Le Guin',
    'first_sentence': 'With a clamor of bells that set the swallows soaring, the Festival
of Summer came to the city Omelas, bright-towered by the sea.',
    'published': '1973'},
   {'id': 2,
    'title': 'Dhalgren',
    'author': 'Samuel R. Delany',
    'first_sentence': 'to wound the autumnal city.',
    'published': '1975'},
   {'id': 3,
    'title': 'The Chain',
    'author': 'Jaime G. Páramo',
    'first_sentence': 'There were tears on her eyes and fears trapped her mind but,
inside, the courage of those who have nothing to lose and all to win, flown wild and
free.',
    'published': '2025'}

]
```

3. Añade la ruta y función para acceder a estos datos

```
@app.route('/api/v1/resources/books/all', methods=['GET'])
def api_all():
    return jsonify(books)
```

4. Corre la aplicación y accede a http://127.0.0.1:5000/api/v1/resources/books/all

   Verás que la API devuelve un JSON, que es el formato de datos más común para comunicaciones web. Este json lo creamos a partir de la lista de diccionarios, gracias a la función “jsonify()”, de flask.

De momento nuestro programa tiene un punto de acceso y devuelve todos los libros de nuestra “base de datos”.

3. API para búsqueda de datos

Hasta el momento hemos creado un endpoint y una ruta de acceso a todos los datos. Para este apartado implementaremos en la API una búsqueda de objetos por ID:

1. Añade la siguiente función para buscar por ID. Comprueba en los argumentos de la petición si existe ID. De ser así, buscamos en la base de datos con ese ID, y en caso contrario, devolvemos un mensaje de error.

```
@app.route('/api/v1/resources/book', methods=['GET'])
def api_id():

    if 'id' in request.args:
        id = int(request.args['id'])
    else:
        return "Error: No id field provided. Please specify an id."

    results = []


    for book in books:
        if book['id'] == id:
            results.append(book)

    return jsonify(results)
```

Accede a las siguientes URLs para comprobar su funcionamiento:
127.0.0.1:5000/api/v1/resources/books?id=0
127.0.0.1:5000/api/v1/resources/books?id=1
127.0.0.1:5000/api/v1/resources/books?id=2
127.0.0.1:5000/api/v1/resources/books?id=3

Todo lo que va después del "?", se denominan query parameters, empleados para filtrar un tipo de datos concreto.

En este punto tenemos creado un nuevo enrutado: "`/api/v1/resources/books`", que llamará a la función "`api_id()`", cada vez que se acceda a esa ruta.

Otra manera de acceder a los datos es con el argumento en la propia URL. Como en el siguiente ejemplo donde vamos a buscar por título:

```
@app.route('/api/v1/resources/book/<string:title>', methods=['GET'])
def get_by_title(title):
    for book in books:
        if book['title'] == title:
            return jsonify(book)
    return jsonify({'message': "Book not found"})
```

O también con los argumentos en el propio cuerpo de la petición HTTP. Fíjate que se ha cambiado la version del enrutado, para no confundir con los anteriores.

```
@app.route('/api/v2/resources/book', methods=['GET'])
```

```
def get_by_id():
    id = int(request.get_json()['id'])
    for book in books:
        if book['id'] == id:
            return jsonify(book)
    return jsonify({'message': "Book not found"})
```

Si quisiésemos subir a la BD algún libro, realizaremos un POST:

```
@app.route('/api/v1/resources/book', methods=['POST'])
def post_book():
    data = request.get_json()
    books.append(data)
    return data
```

4. API con BD SQL

Finalmente crearemos una API que sea capaz de manejar errores de consulta a la base de datos, descargarse todos los libros y filtrarlos por fecha de publicación. La BD se puede descargar desde este enlace. Utilizaremos el siguiente código:

```
from flask import Flask, request
import sqlite3

app = Flask(__name__)
app.config["DEBUG"] = True



@app.route('/api/v1/resources/books/all', methods=['GET'])
def get_all():
    connection = sqlite3.connect('books.db')
    cursor = connection.cursor()

    select_books = "SELECT * FROM books"

    result = cursor.execute(select_books).fetchall()

    connection.close()

    return {'books': result}
```

```
@app.route('/api/v1/resources/book/<string:author>', methods=['GET'])
def get_by_author(author):
    connection = sqlite3.connect('books.db')
    cursor = connection.cursor()

    select_books = "SELECT * FROM books WHERE author=?"

    result = cursor.execute(select_books, (author,)).fetchall()

    connection.close()

    return {'books': result}



@app.route('/api/v1/resources/book/filter', methods=['GET'])
def filter_table():
    query_parameters = request.get_json()

    id = query_parameters.get('id')
    published = query_parameters.get('published')
    author = query_parameters.get('author')

    connection = sqlite3.connect('books.db')
    cursor = connection.cursor()

    query = "SELECT * FROM books WHERE"
    to_filter = []

    if id:
        query += ' id=? AND'
        to_filter.append(id)
    if published:
        query += ' published=? AND'
        to_filter.append(published)
    if author:
        query += ' author=? AND'
        to_filter.append(author)
    if not (id or published or author):
        return "page not found 404"

    query = query[:-4] + ';'

    result = cursor.execute(query, to_filter).fetchall()

    connection.close()
```

```
    return {'books': result}


app.run()
```

Prueba las siguientes URLs:
http://127.0.0.1:5000/api/v1/resources/books/all
http://127.0.0.1:5000/api/v1/resources/book/Connie%20Willis

La BD tiene 67 entradas, con libros ganadores del premio Hugo entre 1953 y 2014. Incluye nombres de las novelas, autor, id, año de publicación y primera frase.

La función “dict_factory()” permite devolver los objetos de la BD como diccionarios, y no como listas.

5. Material extra

Si quieres seguir formándote con Flask, estos recursos te pueden ayudar:

Tutorial completo y nivel avanzado
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Curso con videos de Oreilly
https://learning.oreilly.com/videos/rest-apis-with/9781788621526/

Flask Web Development - Oreilly
https://learning.oreilly.com/library/view/flask-web-development/9781491991725/