# Fundamentos de Redes de Datos

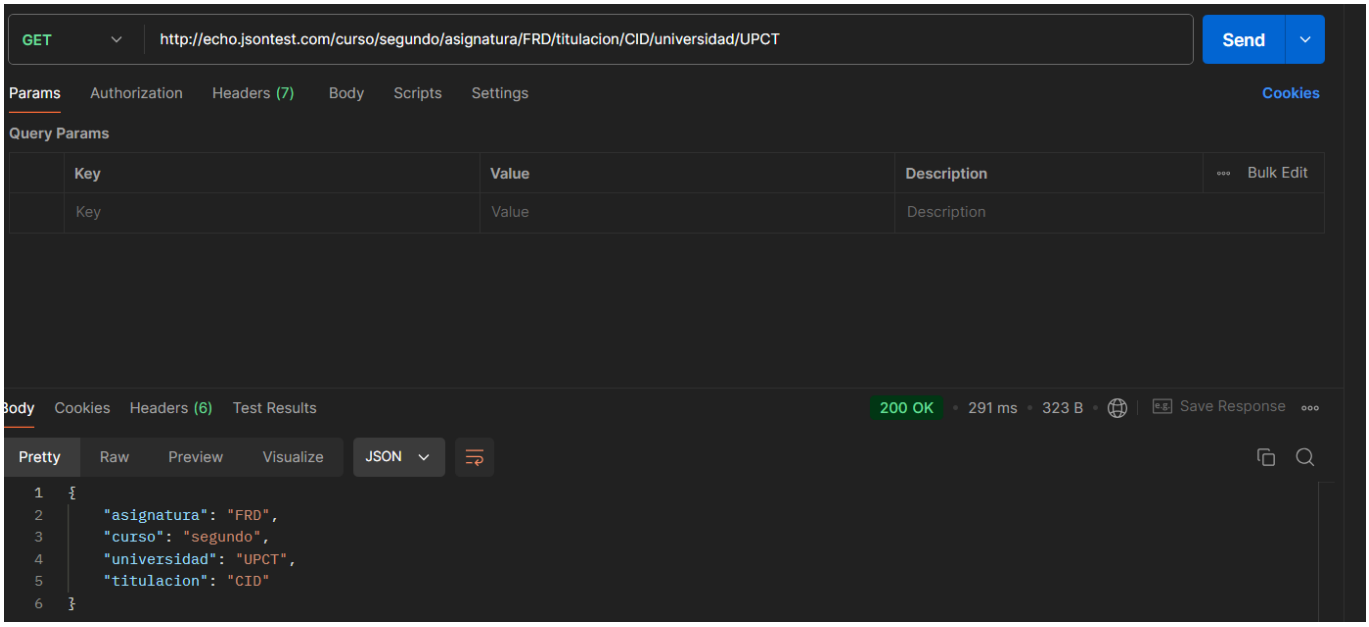
Práctica 3: Desarrollo de aplicaciones para acceso a datos

Francisco Javier Mercader Martínez

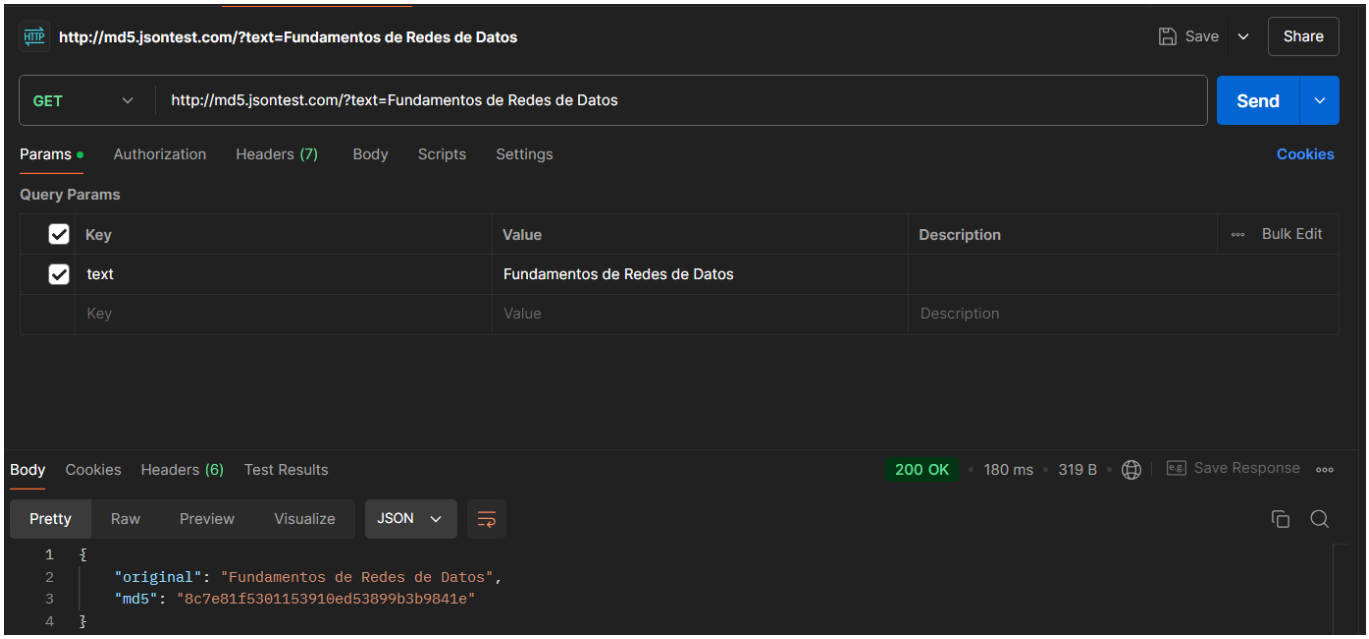
## Clientes

### Tarea 2.4

- Utiliza el servicio `echo` (visto en el ejemplo anterior) para ampliar el objeto devuelto de manera que incluya el nombre de la titulación y la universidad

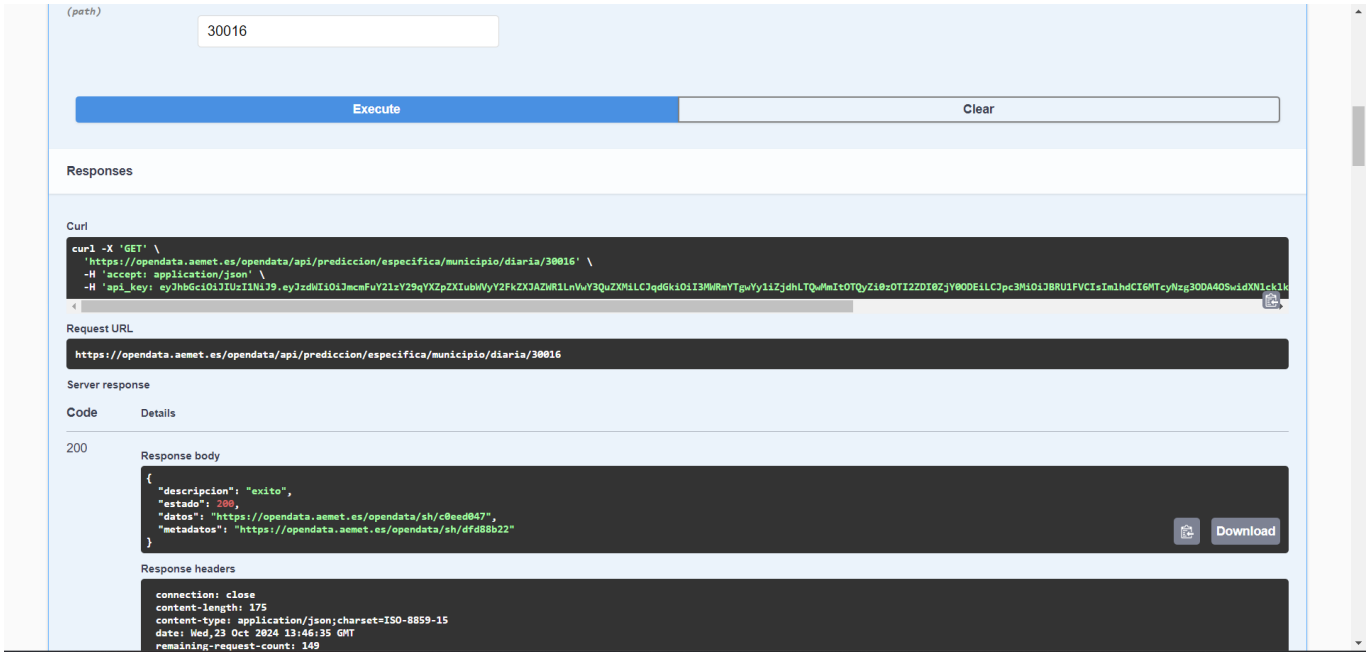



- Usa el servicio `md5` para que te devuelva el resumen digital MD5 del texto “Fundamentos de Redes de Datos”.



## Tarea 3.1

- Explora el servicio `/api/prediccion/especifica/municipio/diaria/{municipio}` para averiguar cómo funciona.
- Emplea los valores 30016 (Cartagena) o 30030 (Murcia) para consultar la predicción diaria sobre alguno de estos municipios
- Comprueba qué formato tiene la respuesta
- ¿Qué crees que deberías realizar para acceder a los datos concretos de precipitación o de estado del cielo?
- Realiza la consulta a través de Postman




La respuesta se obtiene en formato `JSON`

Utilizando *Postman* no me ha devuelto ningún dato, imagino que se deberá a que hace falta usar la `api_key` para que funcione.

## Tarea 4.1

- Introduce el programa anterior de un fichero `.py` y ejecútalo para comprobar que efectivamente se obtiene una respuesta similar a la que se ha mostrado de ejemplo.

```
import requests

url = "https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/diaria/30016"

querystring  = {"api_key": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJmcmFuY2lzY29qYXZpZXIubWVyY2FkZXJAZWR1LnVwY3QuZ

headers = {
        "cache-control": "no-cache"
    }

response = requests.request("GET", url, headers=headers, params=querystring)

print(response.text)
```

```
## {
##   "descripcion" : "exito",
##   "estado" : 200,
##   "datos" : "https://opendata.aemet.es/opendata/sh/c0eed047",
##   "metadatos" : "https://opendata.aemet.es/opendata/sh/dfd88b22"
## }
```

### Tarea 5.1

- Tomando como base el programa anterior, modifícalo para imprimir por pantalla solo los datos relativos a la predicción de precipitación y temperatura del municipio de Murcia o de Cartagena.

```
import requests

url = "https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/diaria/30016"

querystring  = {"api_key": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJmcmFuY2lzY29qYXZpZXIubWVyY2FkZXJAZWR1LnVwY3QuZ

headers = {
        "cache-control": "no-cache"
    }

response = requests.request("GET", url, headers=headers, params=querystring)

if response.status_code == 200:
    datos = response.json()
    url_datos = datos["datos"]
    response_datos = requests.request("GET", url_datos, headers=headers)
    print(response_datos.text[:1000])

else:
    print("Respuesta sin éxito")
```

```
## [ {
##   "origen" : {
##     "productor" : "Agencia Estatal de Meteorología - AEMET. Gobierno de España",
##     "web" : "https://www.aemet.es",
##     "enlace" : "https://www.aemet.es/es/eltiempo/prediccion/municipios/cartagena-id30016",
##     "language" : "es",
##     "copyright" : "© AEMET. Autorizado el uso de la información y su reproducción citando a AEMET como
##     "notaLegal" : "https://www.aemet.es/es/nota_legal"
##   },
##   "elaborado" : "2024-10-25T09:02:11",
##   "nombre" : "Cartagena",
##   "provincia" : "Murcia",
##   "prediccion" : {
##     "dia" : [ {
##       "probPrecipitacion" : [ {
##         "value" : 0,
##         "periodo" : "00-24"
##       }, {
##         "value" : 0,
##         "periodo" : "00-12"
##       }, {
##         "value" : 75,
##         "periodo" : "12-24"
##       }, {
##         "value" : 0,
##         "periodo" : "00-06"
##       }, {
##         "value" : 0,
##         "periodo" : "06-12"
##       }, {
##         "value" : 25,
##         "periodo" : "12-18"
##       }, {
##         "value" : 65,
##         "periodo" : "18
```

### Tarea 6.1

Implemente un programa en Python que, utilizando los datos obtenidos a partir de AEMET OpenData, muestre por pantalla mensajes de este estilo.

```
$ python 3 prediccion.py
Introduce el nombre de la ciudad: Cartagena
La temperatura actual es: 24 ºC
La sensación térmica es: 23 ºC
La humedad relativa es: 56 %
```

```python
import requests

# Diccionario predefinido para mapear los nombres de las cuidades
city_codes = {
    "Cartagena": "30016",
    "Murcia": "30030",
}

def get_weather_data(city_code):
    url = f"https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/diaria/{city_code}"
    querystring = {
        "api_key": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJmcmFuY2lzY29qYXZpZXIubWVyY2FkZXJAZWR1LnVwY3QuZXMiLCJqd
    headers = {"cache-control": "no-cache"}

    response = requests.request("GET", url, headers=headers, params=querystring)
    if response.status_code == 200:
        datos = response.json()
        url_datos = datos['datos']
        response_datos = requests.request("GET", url_datos, headers=headers)
        if response_datos.status_code == 200:
            return response_datos.json()
    return None

def main():
    city_name = input("Introduce el nombre de la ciudad: ")
    city_code = city_codes.get(city_name)

    if not city_code:
        print("Ciudad no encontrada.")
        return

    weather_data = get_weather_data(city_code)
    if not weather_data:
        print("No se pudo obtener los datos del tiempo.")

    """
    Asumiendo que la estructura de los datos siempre está en formato JSON,
    se ajustan las claves según la esctructura que tenemos de momento.
    """
    current_weather = weather_data[0]['prediccion']['dia'][0]
    temperature = current_weather['temperatura']["dato"][1]["value"]
    thermal_sensation = current_weather['sensTermica']["dato"][1]["value"]
    humidity = current_weather["humedadRelativa"]["dato"][1]["value"]

    print(f"La temperatura actual es: {temperature} ºC")
    print(f"La sensación térmica es: {thermal_sensation} ºC")
    print(f"La humedad relativa es: {humidity} %")
```

```
## Introduce el nombre de la ciudad: Cartagena
## La temperatura actual es: 22 ºC
## La sensación térmica es: 22 ºC
## La humedad relativa es: 70 %
```

# Tarea Final de Bloque

Implemente un programa en Python que, utilizando lo aprendido en esta práctica, y de forma similar a como se ha realizado con AEMET OpenData, acceda a los datos meteorológicos de OpenWeather a través de su API 2.5 para obtener los datos de temperatura de los próximos 5 días de una ciudad introducida por el usuario. OpenWeather acepta acepta como nombre de ciudad valores como "Cartagena,es".

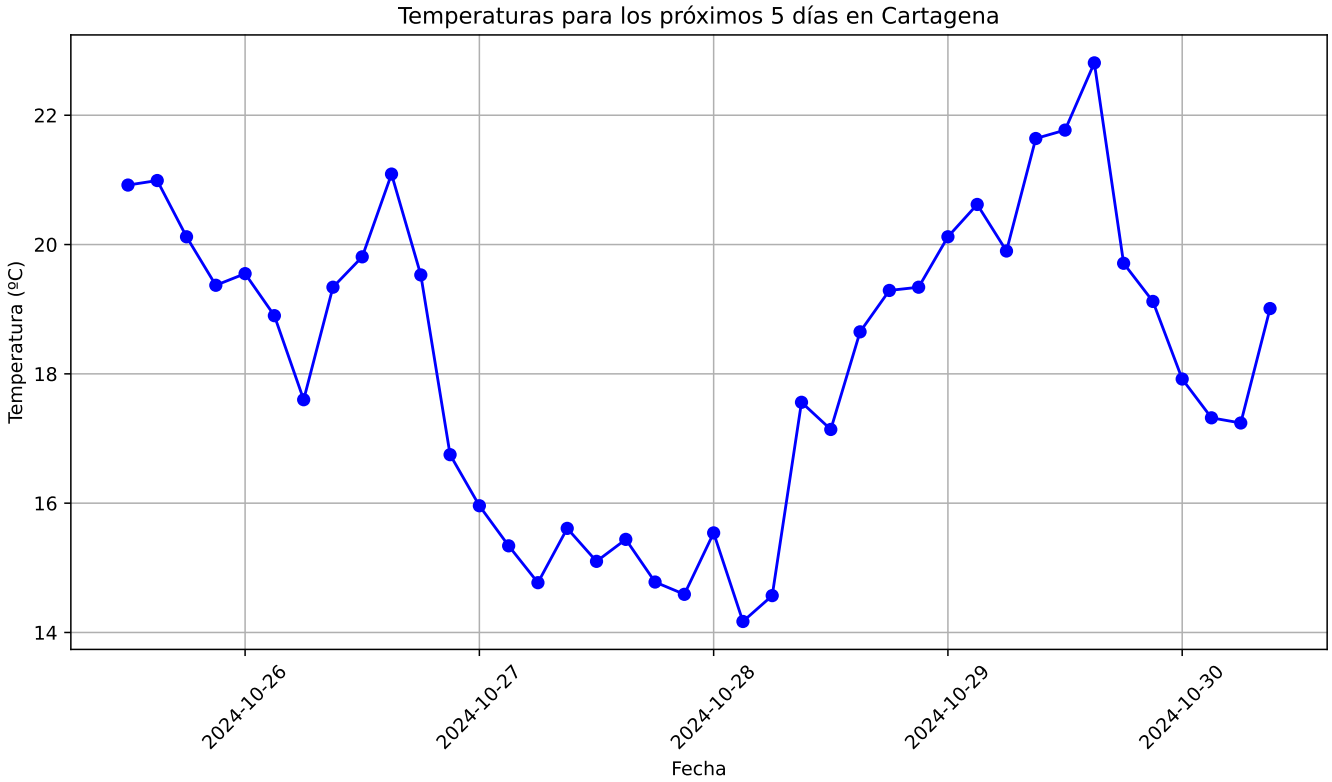
Se valorará con puntuación extra que los datos se muestren de forma gráfica.

Para obtener la **API key** hay que acceder a la dirección

```
import requests
import matplotlib.pyplot as plt
import pandas as pd

# Diccionario predefinido para mapear los nombres de las ciudades
city_codes = {
    "Cartagena" : "Cartagena,es",
    "Murcia": "Murcia, es"
}

def get_weather(city_name):
    api_key = '83de65eb45b2381624501b8d4a320409'
    url = f"http://api.openweathermap.org/data/2.5/forecast?q={city_name}&appid={api_key}&units=metric"
    headers = {"cache-control": "no-cache"}

    response = requests.request("GET", url, headers=headers)
    if response.status_code == 200:
        return response.json()
    return None

def main():
    city_name = input("Ingrese la ciudad que desea buscar: ")
    city_code = city_codes.get(city_name)

    if not city_code:
        print("Ciudad no encontrada")
        return

    weather_data = get_weather(city_code)
    if not weather_data:
        print("No se pudieron obtener los datos del tiempo.")
        return

    # Procesar los datos para la gráfica
    dates = []
    temperatures = []
    for forecast in weather_data['list']:
        dates.append(forecast['dt_txt'])
        temperatures.append(forecast['main']['temp'])

    # Imprime los datos en la consola
    print(f"Temperaturas para los próximos 5 días en {city_name}:")
    for date, temp in zip(dates, temperatures):
        print(f"{date}: {temp} ºC")

    # Convertir los datos en un DataFrame
    df = pd.DataFrame({'Fecha': pd.to_datetime(dates), 'Temperatura (ºC)': temperatures})

    # Crear la gráfica
    plt.figure(figsize=(10, 6))
    plt.plot(df['Fecha'], df['Temperatura (ºC)'], 'bo-')

    # Etiquetas y título
```

```
    plt.title(f'Temperaturas para los próximos 5 días en {city_name}')
    plt.xlabel('Fecha')
    plt.ylabel('Temperatura (ºC)')
    plt.xticks(rotation=45)
    plt.grid(True)

    # Mostrar la gráfica
    plt.tight_layout()
    plt.show()

if __name__ == '__main__':
    main()
```

```
## Introduce el nombre de la ciudad: Cartagena
## Temperaturas para los próximos 5 días en Cartagena:
## 2024-10-25 12:00:00: 20.92 ºC
## 2024-10-25 15:00:00: 20.99 ºC
## 2024-10-25 18:00:00: 20.12 ºC
## 2024-10-25 21:00:00: 19.37 ºC
## 2024-10-26 00:00:00: 19.55 ºC
## 2024-10-26 03:00:00: 18.9 ºC
## 2024-10-26 06:00:00: 17.6 ºC
## 2024-10-26 09:00:00: 19.34 ºC
## 2024-10-26 12:00:00: 19.81 ºC
## 2024-10-26 15:00:00: 21.09 ºC
## 2024-10-26 18:00:00: 19.53 ºC
## 2024-10-26 21:00:00: 16.75 ºC
## 2024-10-27 00:00:00: 15.96 ºC
## 2024-10-27 03:00:00: 15.34 ºC
## 2024-10-27 06:00:00: 14.77 ºC
## 2024-10-27 09:00:00: 15.61 ºC
## 2024-10-27 12:00:00: 15.1 ºC
## 2024-10-27 15:00:00: 15.44 ºC
## 2024-10-27 18:00:00: 14.78 ºC
## 2024-10-27 21:00:00: 14.59 ºC
## 2024-10-28 00:00:00: 15.54 ºC
## 2024-10-28 03:00:00: 14.17 ºC
## 2024-10-28 06:00:00: 14.57 ºC
## 2024-10-28 09:00:00: 17.56 ºC
## 2024-10-28 12:00:00: 17.14 ºC
## 2024-10-28 15:00:00: 18.65 ºC
## 2024-10-28 18:00:00: 19.29 ºC
## 2024-10-28 21:00:00: 19.34 ºC
## 2024-10-29 00:00:00: 20.12 ºC
## 2024-10-29 03:00:00: 20.62 ºC
## 2024-10-29 06:00:00: 19.9 ºC
## 2024-10-29 09:00:00: 21.64 ºC
## 2024-10-29 12:00:00: 21.77 ºC
## 2024-10-29 15:00:00: 22.81 ºC
## 2024-10-29 18:00:00: 19.71 ºC
## 2024-10-29 21:00:00: 19.12 ºC
## 2024-10-30 00:00:00: 17.92 ºC
## 2024-10-30 03:00:00: 17.32 ºC
## 2024-10-30 06:00:00: 17.24 ºC
## 2024-10-30 09:00:00: 19.01 ºC
```




# Servidores

## Tarea 2.1

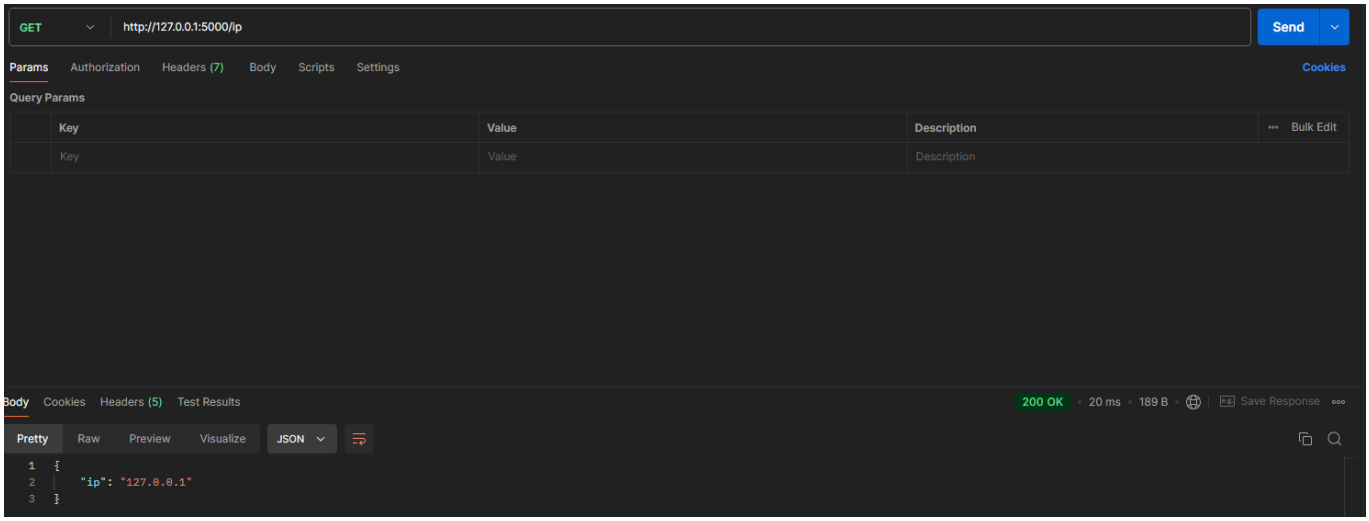
- Introduce el código anterior en un fichero `server1.py` para ponderlo en ejecución, tal y como se mostró anteriormente.
- Accede al servicio mediante las tres formas siguientes:
  - Usando el comando curl: `curl -i http://127.0.0.1:5000/ip`
  - Usando Postman
  - Usando un navegador web

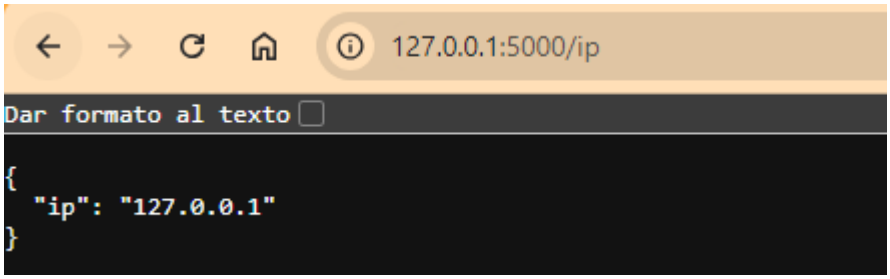
```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/ip')
def obtener_ip():
    direccion_ip = request.remote_addr
    respuesta = {"ip": direccion_ip}
    return jsonify(respuesta)

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

```
 * Serving Flask app 'server1'
 * Debug mode: on
WARNING: This is a development server.
Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 309-414-841
```

```
curl -i http://127.0.0.1:5000/ip
```

```
StatusCode        : 200
StatusDescription : OK
Content           : {
                      "ip": "127.0.0.1"
                    }

RawContent        : HTTP/1.1 200 OK
                    Connection: close
                    Content-Length: 24
                    Content-Type: application/json
                    Date: Wed, 23 Oct 2024 20:51:11 GMT
                    Server: Werkzeug/3.0.4 Python/3.9.12

                    {
                      "ip": "127.0.0.1"
                    }

Forms             : {}
Headers           : {[Connection, close], [Content-Length, 24],
  [Content-Type, application/json], [Date, Wed, 23 Oct 2024 20:51:11 GMT]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 24
```




## Tarea 3.1

- Integra el código del servicio MD5 con el del servicio IP en un fichero `server2.py` de forma que el mismo servidor Flask pueda atender a ambos tipos de soluciones. Comprueba con *Postman* que ambos servicios funcionan correctamente.
- Amplía el servidor con el servicio de ECHO ("/echo") que se encargaba de devolver un objeto JSON utilizando como nombres de los campos y como valores los datos proporcionados en la URL.

- Como mejora, modifica el servicio ECHO para que pueda aceptar cualquier número de pares de (campo, valor)

```
from flask import Flask, request, jsonify
import hashlib

app = Flask(__name__)

@app.route('/ip')
def obtener_ip():
    direccion_ip = request.remote_addr
    respuesta = {"ip": direccion_ip}
    return jsonify(respuesta)

@app.route('/md5/<string:texto>')
def obtener_md5(texto):
    md5_resumen = hashlib.md5(texto.encode('utf-8')).hexdigest()
    respuesta = {"md5": md5_resumen}
    return jsonify(respuesta)

# Función creada para permitir dos pares (campo, valor)
@app.route('/echo/<string:campo1>/<string:valor1>/<string:campo2>/<string:valor2>')
def echo(campo1, valor1, campo2, valor2):
    respuesta = {
        campo1: valor1,
        campo2: valor2
    }
    return jsonify(respuesta)

# Función que permite añadir cualquier par (campo, valor)
@app.route('/echo', methods=['POST'])
def echo():
    # Obtén los datos del cuerpo de la solicitud en formato JSON
    data = request.get_json()
    if not data:
        return jsonify({'error': 'No JSON data provided'}), 400

    # Validar si los datos recibidos son un diccionario
    if not isinstance(data, dict):
        return jsonify({'error': 'Invalid format, expected a JSON object'}), 400

    # Responder con los mismos pares campo-valor
    response = {campo: valor for campo, valor in data.items()}
    return jsonify(response)

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

## Tarea 4.1

- Desarrolla un programa Python llamado `test_server.py` que realice una llamada al servicio `/ip` de tu servidor Flask y que imprima por pantalla la dirección IP obtenida.
- Amplía el programa para que, además, solicite al usuario una cadena de texto y dicha cadena se envíe al servicio `/md5` para que nos devuelva el resultado y se imprima por pantalla.

```
import requests

url = "http://localhost:5000"

ip_service = "/ip"
md5_service = "/md5/"

headers = {'cache-control': "no-cache"}
```

```
ip_response = requests.get(url + ip_service, headers=headers)

# Comprueba si la respuesta de la IP ha sido exitosa
if ip_response.status_code == 200:
    datos = ip_response.json()
    print(f"Mi dirección IP es: {datos['ip']}")
else:
    print(f"Error al obtener la IP: {ip_response.status_code}")
```

```
## Mi dirección IP es: 127.0.0.1
```

```
text_to_hash = "Fundamentos de Redes de Datos"

md5_response = requests.get(url + md5_service + text_to_hash, headers=headers)

if md5_response.status_code == 200:
    hash_data = md5_response.json()
    print(f"El hash MD5 de '{text_to_hash}' es: {hash_data['md5']}")
else:
    print(f"Error al obtener el hash MD5: {md5_response.status_code}")
```

```
## El hash MD5 de 'Fundamentos de Redes de Datos' es: 8c7e81f5301153910ed53899b3b9841e
```

### Tarea 5.1

Implemente el siguiente código en un archivo `books.py`:

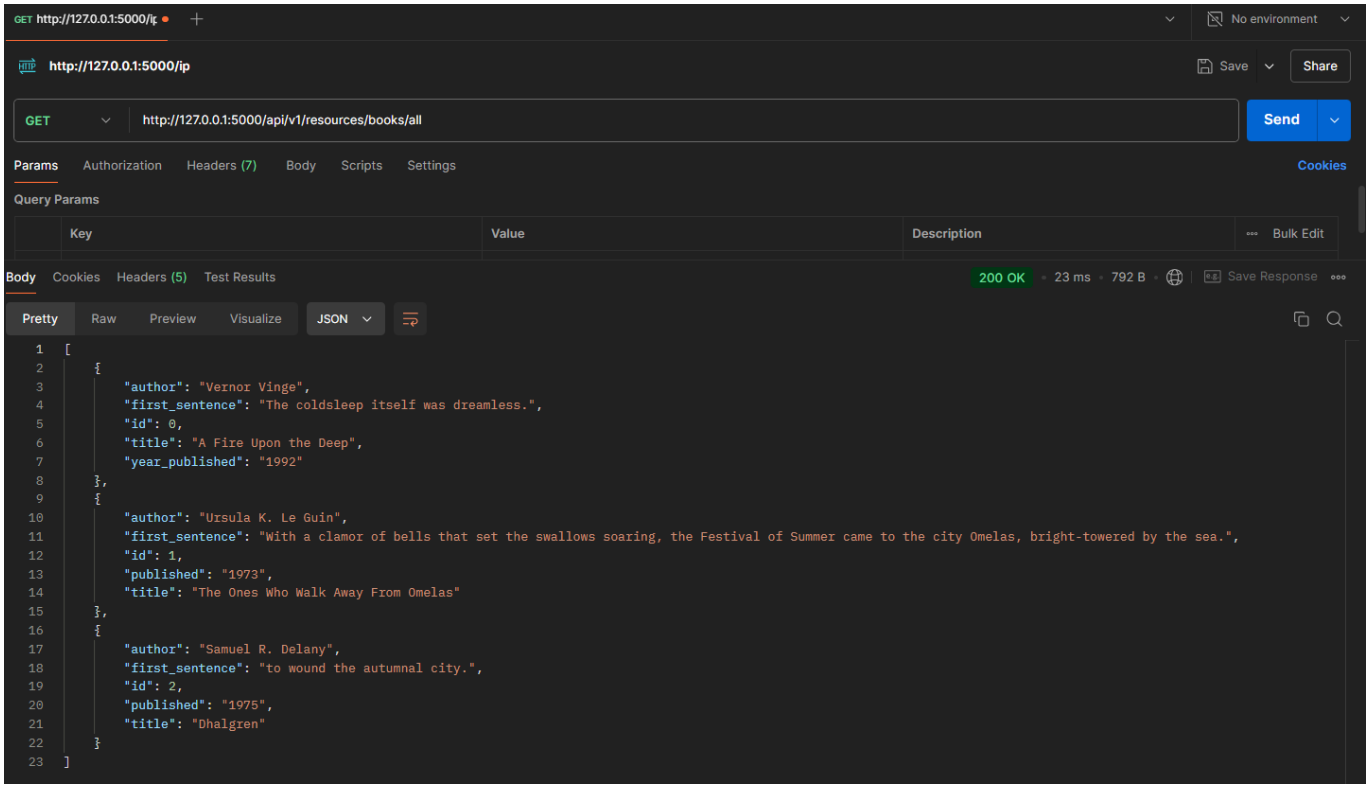
Como se puede ver, la página de inicio simplemente mostrará un mensaje de bienvenida (función `home()`). Por otra parte, el catálogo de libros se define en el código como una lista de diccionarios (`books = [...]`). La ruta '`/api/v1/resouces/book/all`' devolverá todas las entradas disponibles en nuestro catálogo de libros.

Guarda y ejecuta el programa. Navega a http://127.0.0.1:5000/api/v/resources/books/all para ver los datos de prueba devueltos por el endpoint `api/v1/resources/books/all`.

Deberías ver los tres libros de nuestros datos de prueba devueltos con formato JSON.

```
from flask import Flask, request, jsonify

app = Flask(__name__)

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
     'published': '1973'},
    {'id': 2,
     'title': 'Dhalgren',
     'author': 'Samuel R. Delany',
     'first_sentence': 'to wound the autumnal city.',
     'published': '1975'}
]

@app.route('/')
def home():
    return "Distant Reading Archive: A prototype API for distant reading of science fiction novels."
```

```python
@app.route('/api/v1/resources/books/all')
def api_all():
    return jsonify(books)
```




## Tarea 5.2

En el apartado anterior la lista completa se entrega a un endpoint. Sin embargo, hay muchas circunstancias en las que nos interesa los resultados a diferentes endpoints de API.

En este apartado, vamos a crear un nueva función llamada `api_id` y la vamos a asginar al endpoint de la API `api/v1/resources/books`. En esta función, vamos a agregar como parámetro de consulta de los datos el '`id`' de los libros, que se ingresará después de la URL con un `?`.

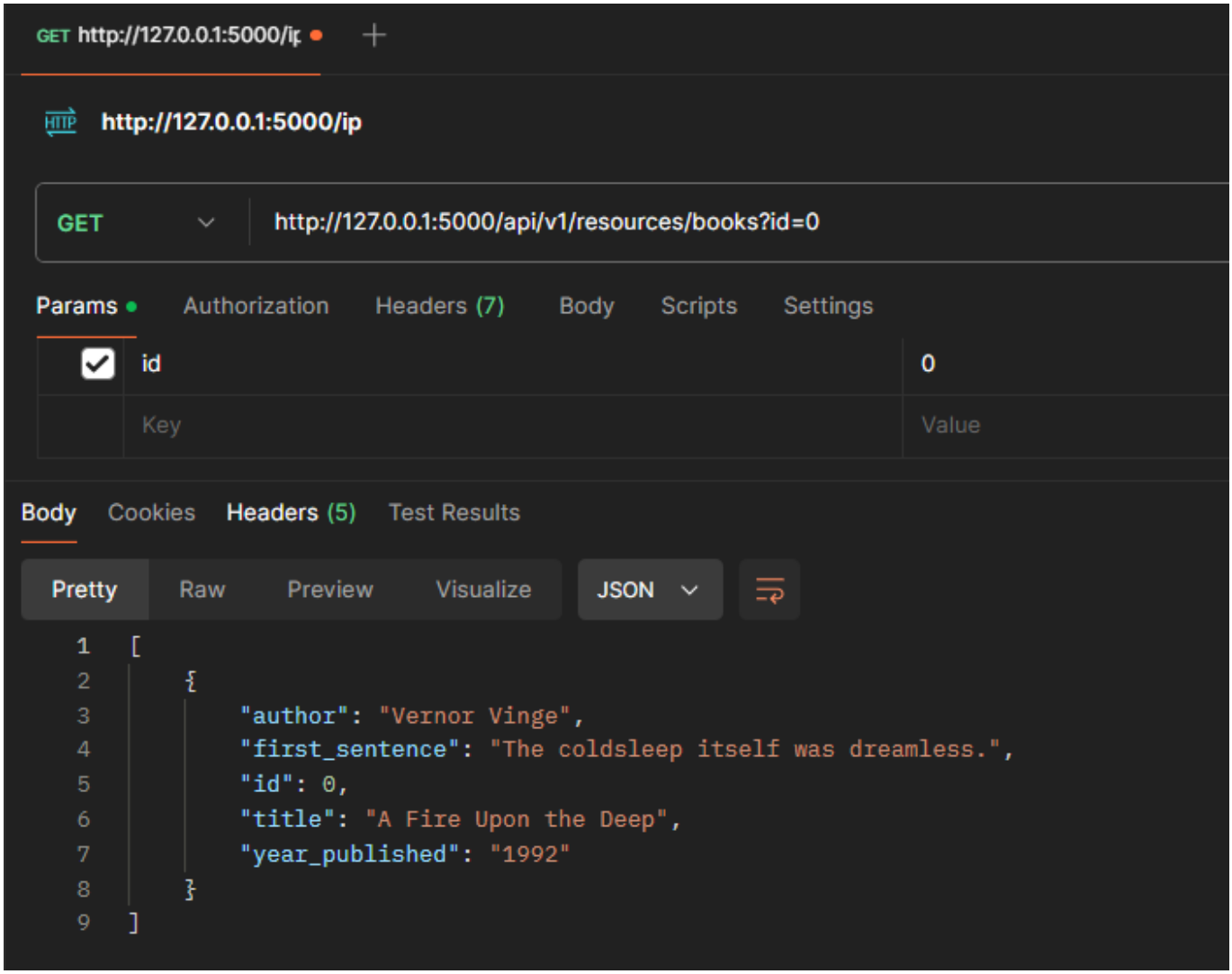
La función deberá buscar este '`id`' particular en nuestros datos. En caso de coincidencia agregará los datos del libro a la lista que se devolverá al usuario en formato JSON.

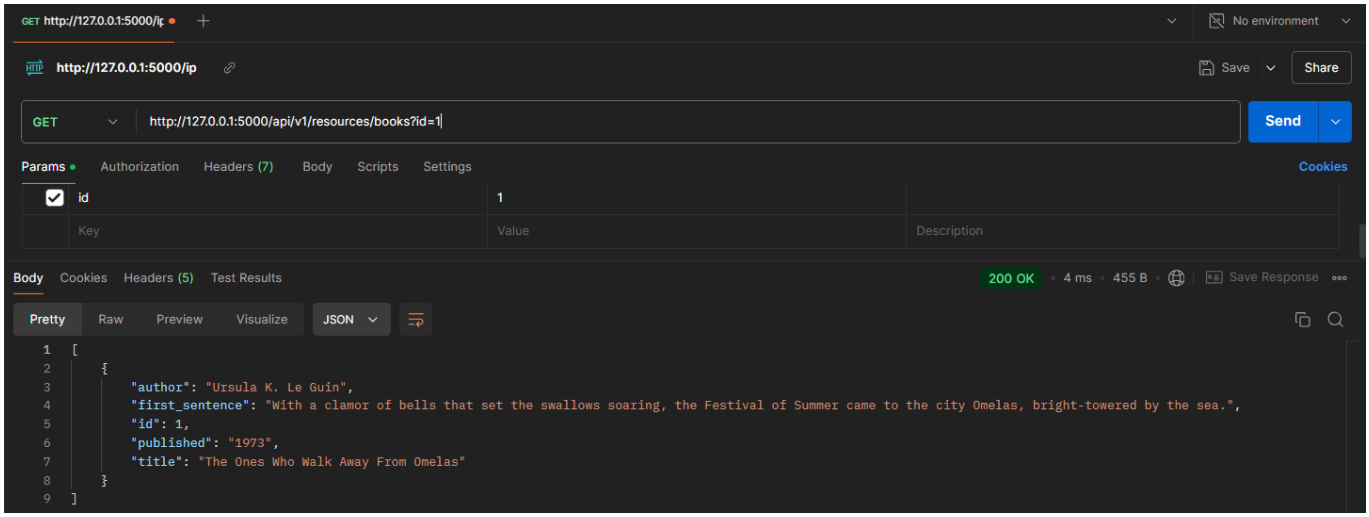
Una vez que el servidor esté funcionando, los siguientes enlaces deberán funcionar:

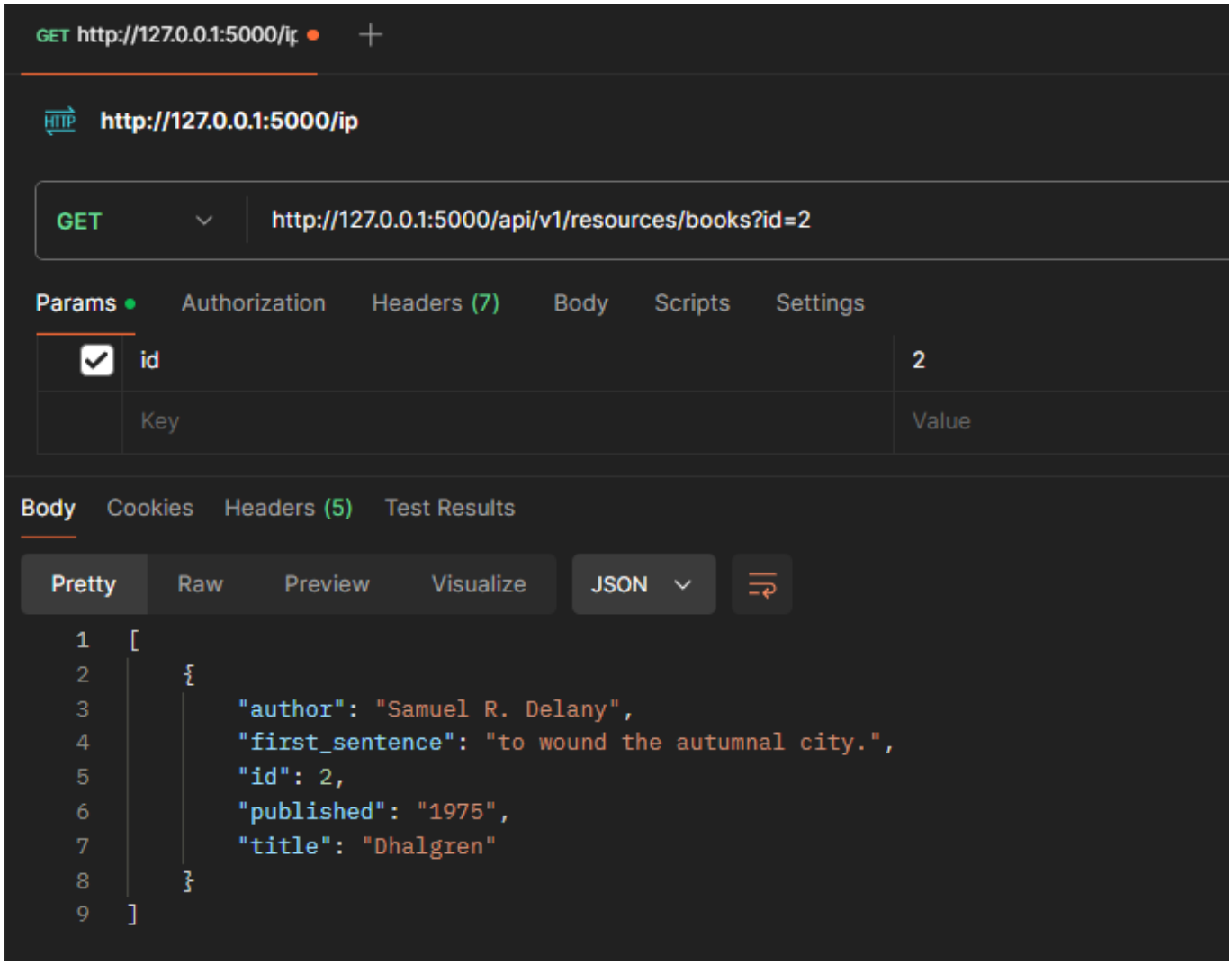
- http://127.0.0.1:5000/api/v/resources/books?id=0
- http://127.0.0.1:5000/api/v/resources/books?id=1
- http://127.0.0.1:5000/api/v/resources/books?id=2

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

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
```

```
     'published': '1973'},
    {'id': 2,
     'title': 'Dhalgren',
     'author': 'Samuel R. Delany',
     'first_sentence': 'to wound the autumnal city.',
     'published': '1975'}
]

@app.route('/')
def home():
    return "Distant Reading Archive: A prototype API for distant reading of science fiction novels."

@app.route('/api/v1/resources/books', methods=['GET'])
def api_id():
    book_id = request.args.get('id')

    if book_id:
        book_id = int(book_id)

        result = [book for book in books if book['id'] == book_id]

        if result:
            return jsonify(result)
        else:
            return jsonify({"error": "No book found with the given ID"}), 404
    else:
        return jsonify({"error": "ID parameter is required"}), 400

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


GET http://127.0.0.1:5000/ip
http://127.0.0.1:5000/ip
GET
http://127.0.0.1:5000/api/v1/resources/books?id=0
Params
Authorization
Headers (7)
Body
Scripts
Settings
id
0
Key
Value
Body
Cookies
Headers (5)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"author": "Vernor Vinge",
"first_sentence": "The coldsleep itself was dreamless.",
"id": 0,
"title": "A Fire Upon the Deep",
"year_published": "1992"
}
]


GET http://127.0.0.1:5000/ip
No environment
http://127.0.0.1:5000/ip
Save
Share
GET
http://127.0.0.1:5000/api/v1/resources/books?id=1
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
id
1
Key
Value
Description
Body
Cookies
Headers (5)
Test Results
200 OK
4 ms
455 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"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.",
"id": 1,
"published": "1973",
"title": "The Ones Who Walk Away From Omelas"
}
]




## Tarea Final de Bloque

Implemente una aplicación Flask que, tras la solicitud http://127.0.0.1:5000/ciudad?q=cartagena,es, devuelva un mensaje de la forma *"La temperatura actual de Cartagena, ES es de 22.8 ºC"*.

```
import requests
from flask import Flask, request

app = Flask(__name__)

@app.route("/ciudad")
def get_weather():
    # Obtener el parámetro de la ciudad
    ciudad = request.args.get("q")

    # Asegurarse de que el parámetro de ciudad esté presente
    if not ciudad:
        return "Error: La ciudad no ha sido proporcionada.", 400

    # Clave API de OpenWeather
    api_key = '83de65eb45b2381624501b8d4a320409'
    url = f"http://api.openweathermap.org/data/2.5/weather?q={ciudad}&appid={api_key}&units=metric"

    # Hacer solicitud a la API de OpenWeather
    response = requests.get(url)

    # Verificar si la respuesta fue exitosa
    if response.status_code != 200:
```

```
        return f"Error: No se pudo obtener el clima para {ciudad}", response.status_code

    # Convertir la respuesta en JSON
    data = response.json()

    # Verificar si la respuesta contiene la información de temperatura
    if 'main' in data and 'temp' in data['main']:
        temperatura = data["main"]["temp"]
    else:
        return "Error: No se han encontrado datos de temperatura.", 500

    # Formatear la ciudad para que la primera letra sea mayúscula
    ciudad_formato = ciudad.split(',')[0].capitalize()

    # Construir el mensaje con el formato solicitado
    return f"La temperatura actual de {ciudad_formato} es {temperatura} ºC"

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

```
curl -i http://127.0.0.1:5000/ciudad?q=cartagena,es
```

```
## La temperatura actual de Cartagena es 15.88 ºC
```