1 print(“Pruebas de Software”)

Camilo Herrera Arcila, M.Sc.
Universidad de San Buenaventura – Medellín
Semestre 2025 – 1
Semana 13

```
1 def main():
2 |  print(“Tabla de
3 |  |     Contenido”)
```

1. Introducción al testeo web
2. Preparando el entorno Django
3. Creando nuestro cliente HTTP (Rest API)
4. Testeo HTTP con Django

Introducción al testeo web

¿Qué es una REST API?

Generalidades

Una **REST API** trabaja por medio del envío de solicitudes (**request**) y recibiendo respuestas (**response**), generalmente en formato **JSON**.

REST API
REpresentational
State **T**ransfer **API**

El **Cliente** es la parte que realiza solicitudes a la API. Puede ser una aplicación web, móvil, de escritorio, o incluso un script en Python.

Métodos HTTP

GET **POST** **PUT** **DELETE**

El **Servidor** es la parte que recibe, procesa y responde a las solicitudes del cliente. Se trata del **BackEnd** donde está implementada la *REST API*

Responsabilidades de Cliente y Servidor

Cliente

- Envía solicitudes HTTP (GET, POST, DELETE, PUT)
- Forma correctamente las URLs y parámetros
- Incluye cabeceras (HEADERS) necesarias
- Procesa las respuestas del servidor (datos, códigos)

Ejemplos de clientes

- Aplicaciones web (JavaSript usando Fecth)
- Aplicaciones móviles (Android, iOS)
- Aplicaciones de escritorio (Python, C#, Java)
- Script de Python usando Requests

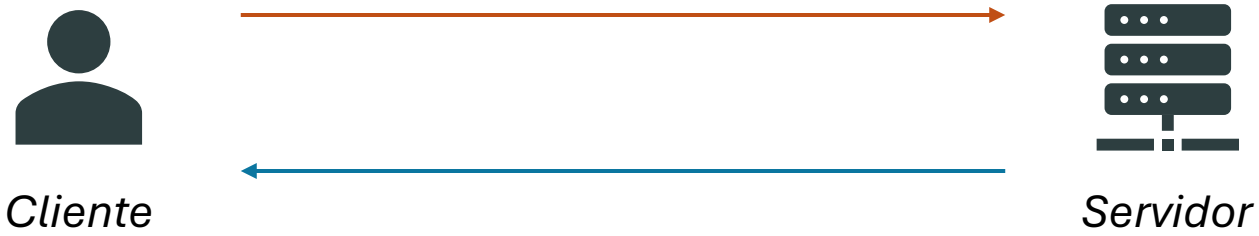
Servidor

- Escucha las solicitudes entrantes
- Procesa los datos recibidos (autenticación)
- Acceder o modificar la base de datos
- Responder con datos en formato estándar (JSON)
- Devulve códigos de estado HTTP (200, 201, 404)

Ejemplos de servidores

- Django REST Framework, Flask, FastAPI, Express.js

Ejemplo de interacción

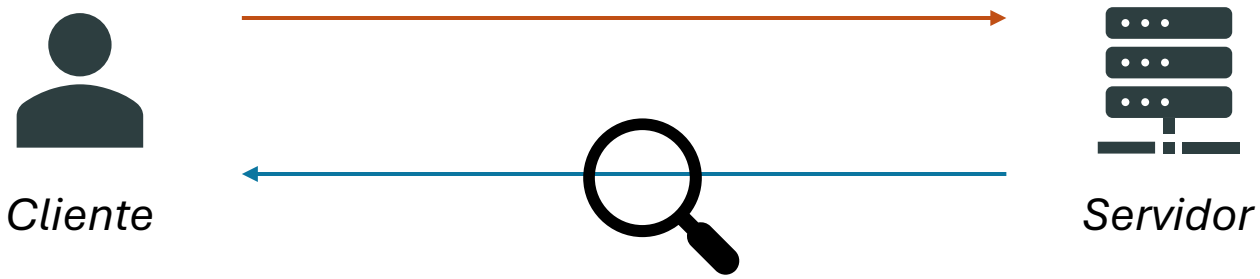


1. https://api.example.com/**usuarios/001**

2. Código HTTP: **200 OK**
   Cuerpo (JSON) **{“id”: 001, “nombre”: “Juan”}**

3. Interpretación del resultado:
   “El usuario existe”
   True

Testeo en la web



El **testeo** de una **API HTTP** consiste en verificar que los ***endpoints*** funcionen **correctamente**, devolviendo los resultados esperados según las operaciones que realizan (GET, POST, PUT, DELETE) cumpliendo con las **reglas del sistema** y la **gestión de errores**.

El ***endpoint*** es una **URL específica** que representa un recurso o una acción que el servidor expone al cliente.

Un ***endpoint*** es una dirección web (**URL**) que el cliente puede usar para interactuar con un recurso específico de la API, mediante una operación **HTTP** (GET, POST, PUT, DELETE).

Preparando el entorno Django

Instalando las librerías necesarias

Crea una carpeta que se llame **djangotutorial/**
y dentro de la misma ejecuta los siguientes comandos:

1. `$ python -m venv venv`
2. `$ venv/scripts/activate`
3. `(venv)$ pip install django`
4. `(venv)$ django-admin startproject mysite .`

Crea el proyecto de Django en el directorio con una aplicación maestra llamada **mysite**.

Deberíamos ver la siguiente estructura de proyecto:

```
djangotutorial/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

https://docs.djangoproject.com/en/5.2/intro/tutorial01/

Elementos de un proyecto Django

`manage.py`

Script para ejecutar comandos del proyecto (migraciones, correr el servidor, crear apps, etc.

`mysite/`

Carpeta principal del proyecto (paquete Python). Contiene la configuración y puntos de entrada del sistema.

`__init__.py`

Archivo vacío que indica que `mysite/` es un paquete de Python.

`settings.py`

Archivo con toda la configuración del proyecto Django.

`urls.py`

Define las rutas (URLs) del proyecto. Conecta las URLs con las vistas de respuesta.

`asgi.py`

Punto de entrada para servidores ASGI (para apps asincrónicas o en tiempo real).

`wsgi.py`

Punto de entrada para servidores WSGI (usada en la mayoría de despliegues tradicionales de Django).

Ejecutando el proyecto Django

Para ejecutar nuestro entorno Django hacemos uso de nuestro archivo **manage.py**

1. `(venv)$ python manage.py migrate`
2. `(venv)$ python manage.py runserver`

```
Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied.
Run 'python manage.py migrate' to apply them.

April 27, 2025 - 15:50:53
Django version 5.2, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

WARNING: This is a development server. Do not use it in a production setting. Use a
production WSGI or ASGI server instead.
For more information on production servers see:
https://docs.djangoproject.com/en/5.2/howto/deployment/
```

Creando un módulo o componente

```
(venv)$ python manage.py startapp classification
```

Es importante que creemos manualmente un archivo **`classification/urls.py`** dentro del módulo. Esto nos servirá para gestionar las vistas.

Patrón de diseño

Model – Template – View

Model: Define la estructura de datos como se guardan en una DB.
Template: La parte visual de la aplicación (HTML que se muestra al usuario).
View: Contiene la lógica del negocio. Recibe la petición HTTP, accede al modelo, selecciona la plantilla y devuelve la respuesta.

```
djangotutorial/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
classification/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    urls.py
    views.py
```

Creando una vista

Nuestra primera vista

classification/views.py

```
from django.http import HttpResponse

def index(request):
    return HttpResponse(“Hello USB!")
```

classification/urls.py

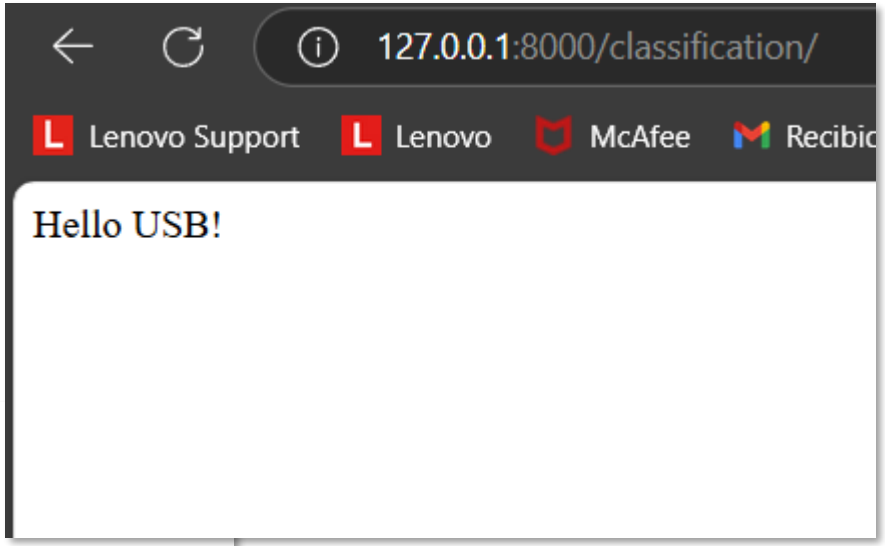
```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.index, name="index"),
]
```

Nuestro directorio debe verse así:

```
classification/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    urls.py
    views.py
```

Nuestra primera vista

El siguiente paso es configurar la configuración URL del directorio raíz del proyecto en **mysite/urls.py**




```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path(“classification/", include(“classification.urls")),
    path("admin/", admin.site.urls),
]
```

Nuevamente ejecutamos nuestro aplicativo:

```
(venv)$ python manage.py runserver
```

REST API

Creando nuestra REST API

```
from django.http import HttpResponse, JsonResponse
```

Se trata de una aplicación que clasifica a un estudiante según su calificación y retorna un ***mensaje de aliento***.

Para ello, vamos a crear la vista correspondiente y la funcionalidad de la REST API.

- Creamos la vista **`classify`** y verificamos que se trate de una solicitud GET por medio del request.

`classification/views.py`

```
def classify(request):

    if request.method != 'GET':
        return JsonResponse(
            {'error': 'Método no permitido. Solo se acepta GET.'},
            status=405
        )
```

Creando nuestra REST API

Ahora se extrae el valor de la variable ***nota*** enviada por el usuario por medio de solicitud GET y se verifica que sea una variable numérica.

classification/views.py

```
def classify(request):
    ...
nota = request.GET.get("nota")

    try:
        nota = float(nota)
    except:
        return JsonResponse(
            {'error': 'Los parámetros debe ser numérico'},
            status=422
        )
```

Creando nuestra REST API

Finalmente, si todo sale bien, debemos retornar la respuesta JSON con el mensaje de clasificación y el código de estado `200 OK`.

classification/views.py

```
def classify(request):
    ...
message = assessGrade(nota)

    return JsonResponse(
        {"result": message},
        status=200
    )
```

Creando nuestra REST API

La función **assessGrade()**, permite retornar el mensaje según la nota del estudiante.

classification/views.py

```python
def assessGrade(grade):
    if grade < 0 or grade > 5:
        return "Error: Esa nota no existe ni en otro universo."

    if 0 <= grade < 0.5:
        return "Hay que pesar... no hiciste nada. Ni nombre pusiste."
    elif 0.5 <= grade < 1:
        return "Al menos prendiste el computador… ¿no?"
    elif 1 <= grade < 2:
        return "Eso no es un trabajo, es un intento de señal de humo."
    elif 2 <= grade < 3:
        return "Ahí vas... pero aún no cruzas la línea de fuego."
    elif 3 <= grade < 4:
        return "Bien. Saliste vivo, pero podrías haber brillado más."
    elif 4 <= grade < 4.5:
        return "Muy bien, ya hueles a matrícula de honor."
    elif 4.5 <= grade <= 5:
        return "¡Excelente! Si fueras un bug, diríamos que eres de producción."
```

Conexión de la nueva vista

Debemos conectar la ruta respectiva con la vista **classify** para que sea accesible.

classification/urls.py

```
urlpatterns = [
    path("", views.index, name="index"),
    path("classify", views.classify, name="classify")
]
```

REST API Classify

Teniendo en cuenta la ruta que hemos configurado, el endpoint de nuestra REST API sería:

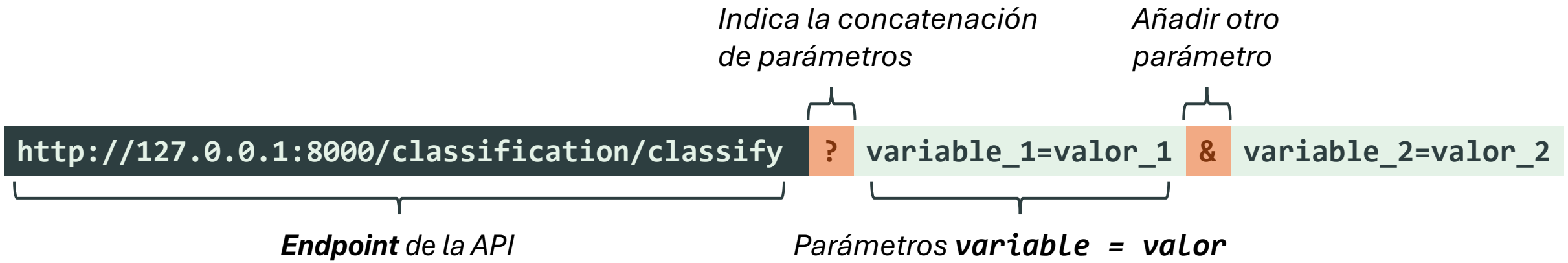
Ruta asociada a la vista

```
http://127.0.0.1:8000/classification/classify
```

Ruta asociada al módulo

Estructura de una solicitud GET

Recuerde que en un **método GET**, los parámetros enviados por el cliente se construyen como sigue



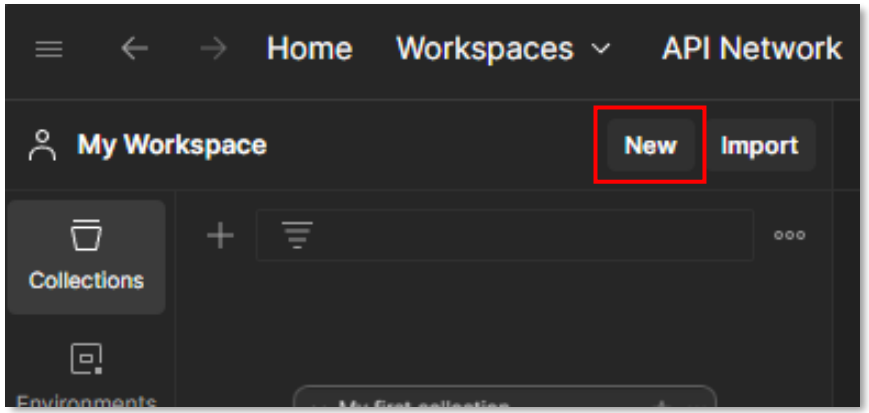
Testeo web HTTP

POSTMAN

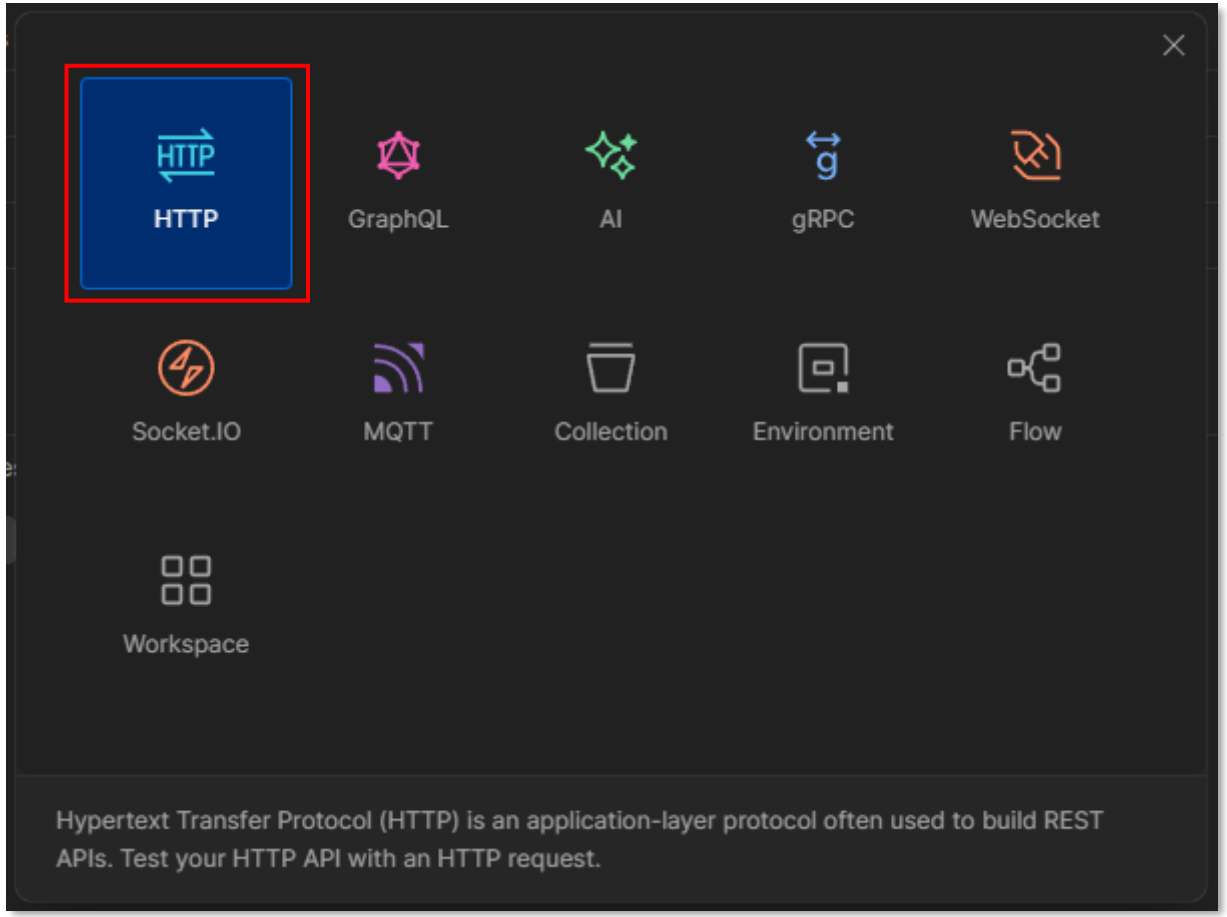
Testeo con Postman

Al abrir Postman, cree un nuevo proyecto y seleccione de tipo HTTP.

1

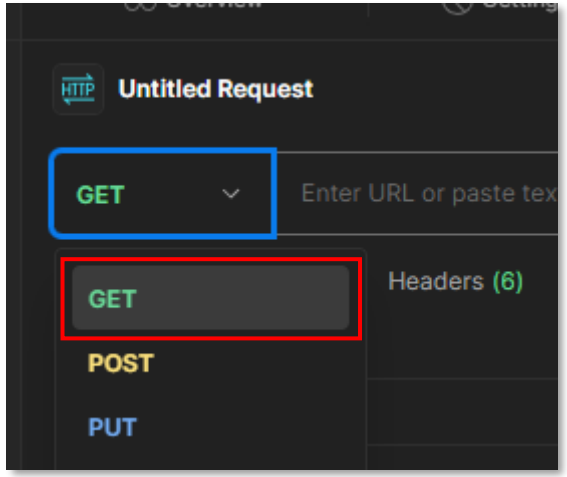


2

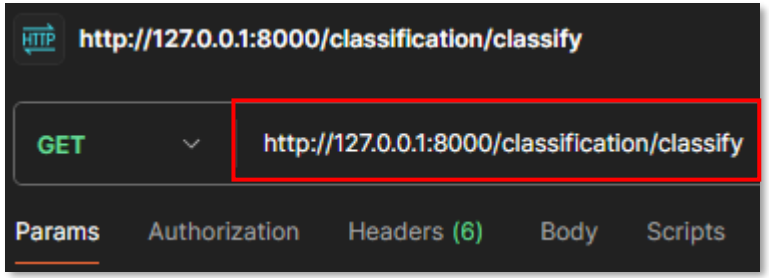


Testeo con Postman

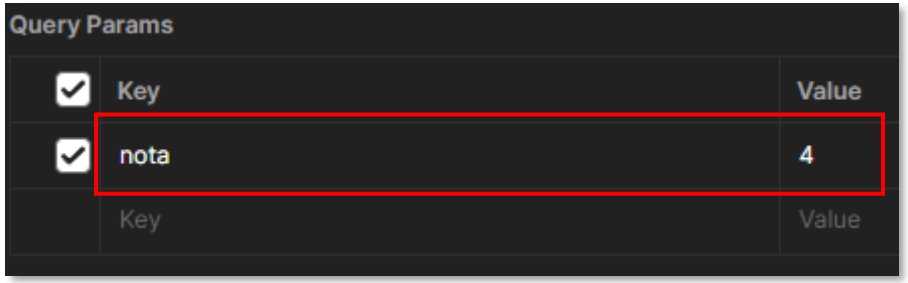
1. Seleccione el método GET



2. Agregue el endpoint de la API



3. Ahora agregue el parámetro **`nota`** con un valor numérico, por ejemplo, 4.

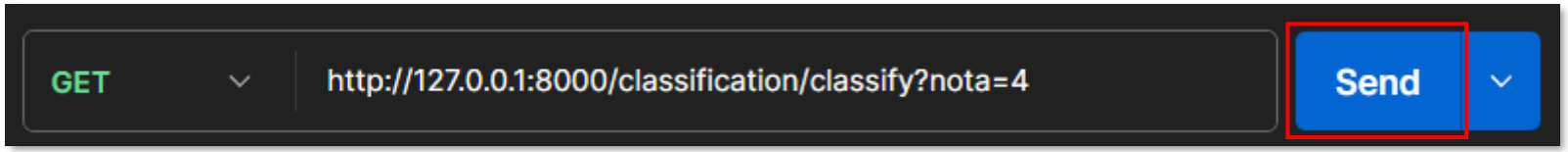


4. Su solicitud HTTP GET, debería verse como sigue:

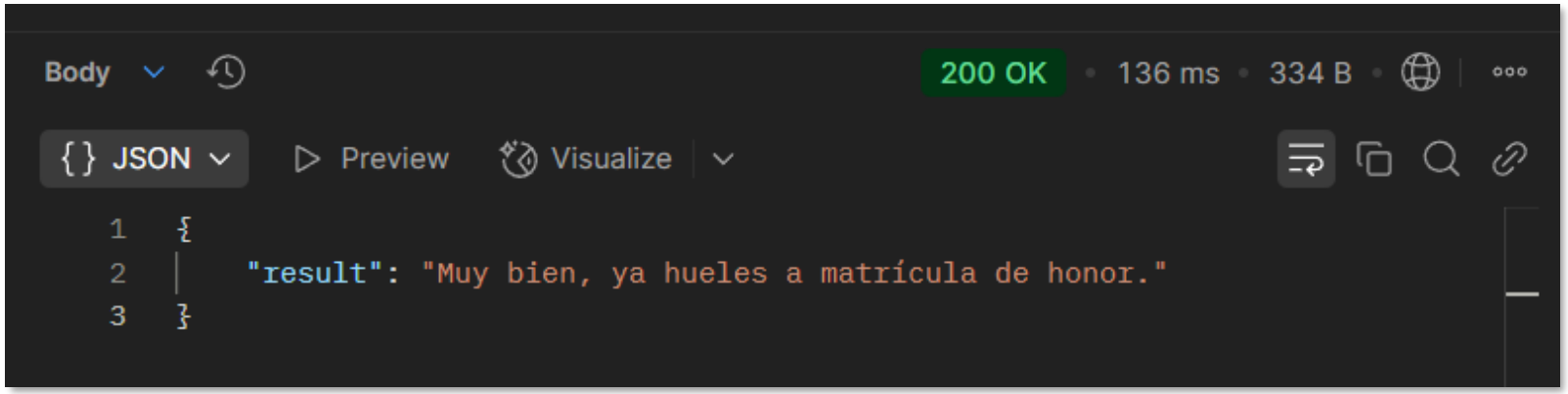
```
http://127.0.0.1:8000/classification/classify?nota=4
```

Probar la conexión y respuesta de la API

1. De clic en enviar (Send) la solicitud



2. Si todo sale bien, deberíamos ver la respuesta del servidor **JSON** con el código de estado `200 OK`.



Testeo web HTTP

Testeo http con UnitTest

En la carpeta del módulo classification, debería ver un archivo **tests.py.** Por defecto, Django importa las herramientas de UnitTest. Entonces, vamos a crear un test sencillo utilizando el esquema de unittest.

```
from django.test import TestCase
import requests

# Create your tests here.

class test_api(TestCase):

    def test_peticion(self):
        params = {"nota":4}
        response =
requests.get("http://127.0.0.1:8000/classification/classify",params=params)
        assert response.status_code == 200
```

https://docs.djangoproject.com/en/5.1/topics/testing/overview/

Testeo http con UnitTest

Para ejecutar el test, utilice el siguiente comando:

```
(venv)$ python manage.py test
```

Recuerde que en una terminal auxiliar, el servidor debe estar ejecutando el aplicativo.

Terminal

```
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.017s

OK
Destroying test database for alias 'default'...
```

https://docs.djangoproject.com/en/5.1/topics/testing/overview/

Gracias

Pruebas de Software
Universidad de San Buenaventura Medellín
Camilo Herrera Arcila, M.Sc.
camilo.herreraa@tau.usbmed.edu.co