PRÁCTICA 2

MÉTODOS HTTP: PUT, PATCH, DELETE, GET

Equipo:
Beltrán Saucedo Axel Alejandro
Cerón Samperio Lizeth Montserrat
Higuera Pineda Angel Abraham
Lorenzo Silva Abad Rey
4BV1

ESCUELA SUPERIOR DE CÓMPUTO

TECNOLOGÍAS PARA EL DESARROLLO DE APLICACIONES WEB

25/09/2025

1. Introducción

Objetivo de la práctica:

El objetivo de la práctica es implementar los métodos HTTP: PUT, PATCH, GET, DELETE en Python con FastAPI, también realizar pruebas de cada uno de los métodos, poniendo a prueba las respuestas que brindará. La aplicación simulará la gestión de paquetes, es decir, que se puede crear un item nuevo con peso y valor, también se le puede poner un ID para ejecutar las funciones de borrar y actualizar parcialmente por el ID.

Métodos HTTP:

- **GET:** Este método se utiliza para solicitar datos de un recurso específico. En la práctica, se implementa para obtener la lista completa de paquetes o un paquete específico por su ID.
- **POST:** Este método se utiliza para enviar datos a un servidor para crear un nuevo recurso. En la práctica, se implementa para crear un nuevo paquete con atributos como peso y valor.
- **PUT:** Este método se utiliza para actualizar completamente un recurso existente. En la práctica, se implementa para actualizar todos los atributos de un paquete específico identificado por su ID.
- **PATCH:** Este método se utiliza para actualizar parcialmente un recurso existente. En la práctica, se implementa para modificar solo algunos atributos de un paquete específico identificado por su ID.
- **DELETE:** Este método se utiliza para eliminar un recurso específico. En la práctica, se implementa para eliminar un paquete identificado por su ID. [1]

2. Fundamentos teoricos

FastAPI:

FastAPI es un framework web para construir APIs en Python, lanzado en diciembre de 2018 por Sebastián Ramírez, un desarrollador colombiano. Está diseñado para ser:

- **Rápido:** comparable en rendimiento a Node.js y Go gracias a su base en ASGI (Asynchronous Server Gateway Interface).
- **Moderno:** aprovecha las anotaciones de tipo de Python 3.6
- **Automático:** genera documentación interactiva con Swagger UI y ReDoc.
- **Seguro y robusto:** ideal para APIs que requieren validación estricta y autentic

Pero, ¿qué hace realmente FastAPI?

Nos permite crear endpoints o rutas para nuestra API, que pueden ser accedidas mediante los diferentes metodos HTTP, como GET, POST, PUT, DELETE, entre otros. Además nos permite:

- Validar automaticamente los datos de entrada y salida utilizando Pydantic.
- Servir modelos de machine learning en producción.
- Integrarse facilmente con base de datos, etc.
- Generar documentación interactiva sin esfuerzo.

Pero, ¿para qué lo usamos realmente?
Usamos FastAPI para crear aplicaciones que se comunican por internet, como por ejemplo:

- Aplicaciones web que mandan y reciben datos (como una app de clima o una tienda en línea).
- Sistemas que conectan con modelos de inteligencia artificial (como chatbots o análisis de imágenes).
- Servicios que otras apps usan para pedir información (como "dame los datos del usuario" o "guárdame esta foto").

HTTP:

FastAPI utiliza HTTP para comunicarase entre la aplicación web y el cliente. Cada vez que alguien hace una petición (Por ejemplo: GET, POST, etc) a nuestra API, FastAPI recibe esa petición, la procesa y responde con los datos solicitados.

¿Qué es HTTP?
HTTP (Hypertext Transfer Protocol). Es un lenaguaje utilziado para realizar una comunicación entre un cliente (como un navegador web o una app móvil) y un servidor (donde vive la aplicación web).
Funciona con peticiones y respuestas, aglunas de ellas son:

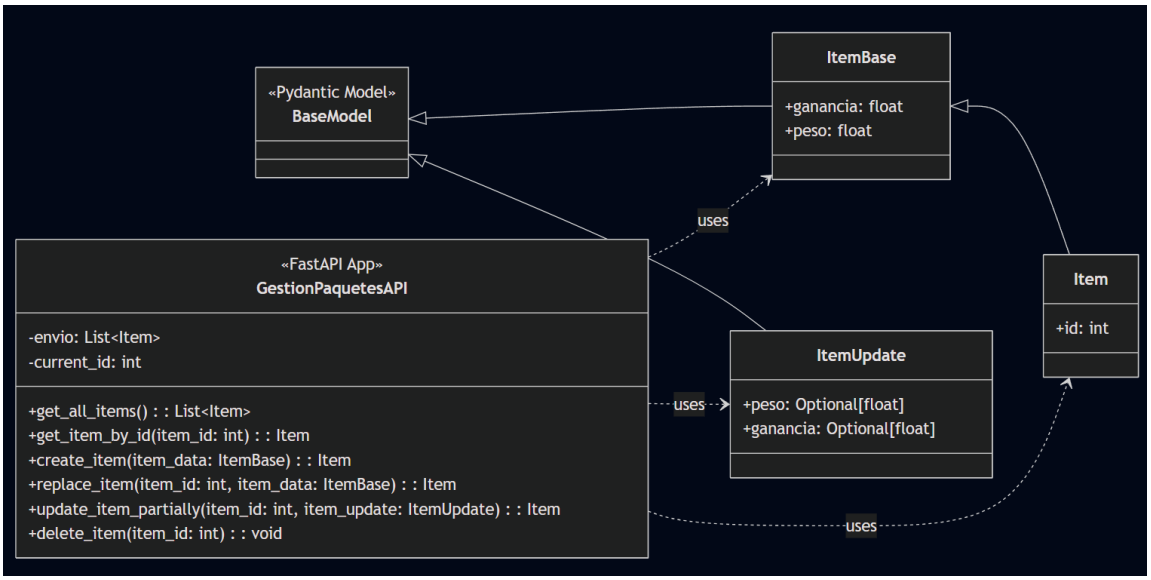
- **GET:** Solicita datos de un recurso específico.
- **POST:** Envía datos para crear un nuevo recurso.
- **PUT:** Actualiza completamente un recurso existente.

- **PATCH:** Actualiza parcialmente un recurso existente.
- **DELETE:** Elimina un recurso específico.

Tambien se cuentan con las HTTP Exceptions, que son respuestas a errores que ocurren cuando algo sale mal en la comunicación entre el cliente y el servidor.
Algunos ejemplos son:

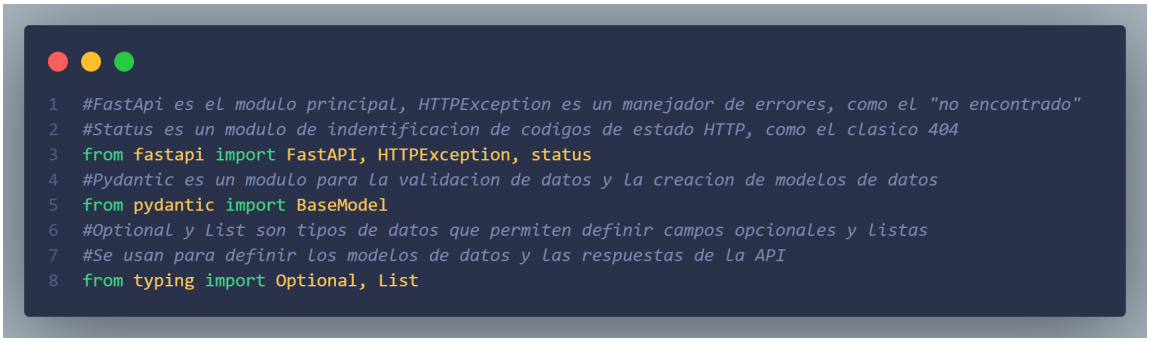
- **404:** No encontrado
- **403:** Prhohibido
- **500:** Error interno del servidor
- **400:** Solicitud incorrecta

3. Diagrama UML



4. Implementación

En este apartado se tienen los modulos a utilziar para el funcionamiento de la practica.



```python
#FastApi es el modulo principal, HTTPException es un manejador de errores, como el "no encontrado"
#Status es un modulo de indentificacion de codigos de estado HTTP, como el clasico 404
from fastapi import FastAPI, HTTPException, status
#Pydantic es un modulo para la validacion de datos y la creacion de modelos de datos
from pydantic import BaseModel
#Optional y List son tipos de datos que permiten definir campos opcionales y listas
#Se usan para definir los modelos de datos y las respuestas de la API
from typing import Optional, List
```

FastApi nos permite contruir APIS de manera rapida y sencilla, permitiendo definir rutas y manejar solicitudes HTTP de manera eficiente.
Las rutas que utilizaremos en esta practica son: GET, POST, PUT y DELETE.
Utilizamos Pydantic para validad entradas y para la creacion de un modelo de datos, que en es este caso sera para tener la id de un objeto, y su respectivo contenido.
Optional y list son tipos de datos que nos permiten definir atributos que pueden ser opcionales y listas respectivamente.

```python
#Creamos la instancia de FastAPI
#Es el corazon de nyestro programa (API)
app = FastAPI(
    #Nombre y descripcion de la API
    title= "Practica 2: Metodos GET, PUT, POST, DELETE Y PATCH",
    description= "API para almacenar, editar, actualizar y borrar la información de los paquetes (items) a ser enviados"
)
```

Iniciamos con el corazon de nuestro programa, que es la instancia de FastAPI, en donde definimos el nombre y la descripcion de la API.

```python
#ItemBase es la estructura base de un item, con los campos ganancia y peso
#Digamos que es lo que tiene cada item
class ItemBase(BaseModel):
    ganancia: float
    peso: float
#Item contiene el indentificador unico (id) y hereda los campos de ItemBase
class Item(ItemBase):
    id: int
#ItemUpdate es una clase especial utilizada para cuando queremos actualizar un item parcialmente
#Los campos son opcionales, ya que podemos querer actualizar solo uno de ellos
#El valor por defecto es None
class ItemUpdate(BaseModel):
    peso: Optional[float] = None
    ganancia: Optional[float] = None
```

Gracias a Pydantic, podemos crear una estructura de datos para nuestros objetos a utilizar. Se tiene un id de tipo entero, este servira para identificar cada objeto. Cda objeto tiene un contenido de datos, en este caso cuenta con ganancia y perso, ambos de tipo flotante. Además se tiene una estrcutura para actualizar los datos parcialmente, los campos son opcionales, esto es debido a que no es obligatorio actualizar ambos.

```python
#Creamos una lista llamada envio para almacenar los items
envio: List[Item] = []
#Variable para tener el control dek ud qye se le asigna a cada item
current_id = 0

```

Utilizamos variables globales para almacenar los objetos y llevar un control del id. El id se inicializa en 0, y se incrementa cada vez que se crea un nuevo objeto. Los objetos se almacenan en una lista, que inicialmente esta vacia.

Para poder iniciar nuestra API, se debe utilizar un comando en la terminal: python -m uvicorn Codigo.Practica2:app –reload. Existen diferentes maneras de iniciar la API, pero esta es la que se utilizo en este caso. Una cosa interesante es que debemos especificar

en el comando que trabajaremos con python, ya que sin el detecta como desconocido a uvicorn.

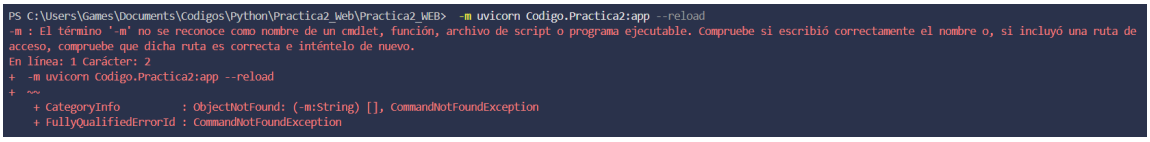


Figura 1: Error si no se especifica que se trabajara con python

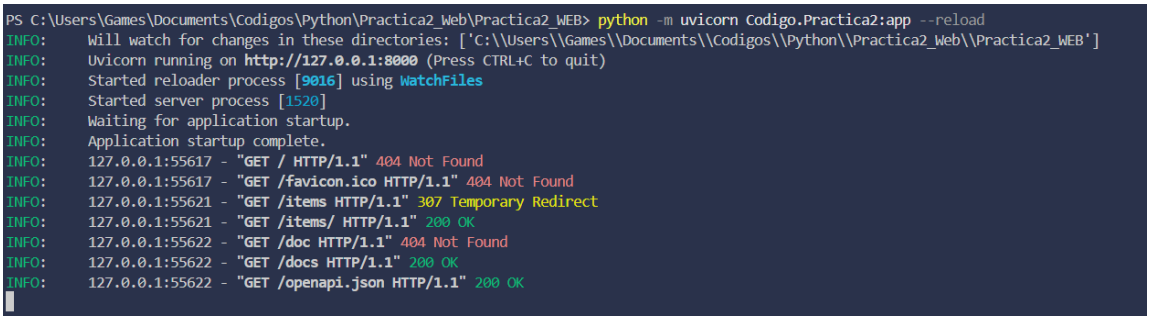


Figura 2: Resultado correcto al iniciar la API

Cuando se inicia la api, en la consola podemos visualizar la direccion en la que se esta ejecutando, en este caso es de manera local, por lo cual se ustiliza: http://127.0.0.1:8000 Además podemos ver las peticiones que se le realizan a la pagina y si hubo un error o se realizo correctamente.

Referencias

[1] Métodos de petición HTTP. (s/f). *MDN Web Docs*. Recuperado el 23 de septiembre de 2025, de https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Methods

[2] Ramírez Montaño, S. (s.f.). "Historia, diseño y futuro de FastAPI". *FastAPI*. Recuperado el 23 de septiembre de 2025, de https://fastapi.tiangolo.com/es/history-design-future/

[3] Wikipedia. (2025, julio 11). "FastAPI". *Wikipedia, la enciclopedia libre*. Recuperado el 23 de septiembre de 2025, de https://es.wikipedia.org/wiki/FastAPI

[4] Lubanovic, B. (2019). *Introducing Python: Modern Computing in Simple Packages* (2ª ed.). O'Reilly Media. ISBN: 9781492051367

[5] Linode Guides & Tutorials. (2021, agosto 6). "Document a FastAPI App with OpenAPI". *Linode*. Recuperado el 23 de septiembre de 2025, de https://www.linode.com/docs/guides/document-a-fastapi-app-with-openapi/