

# 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

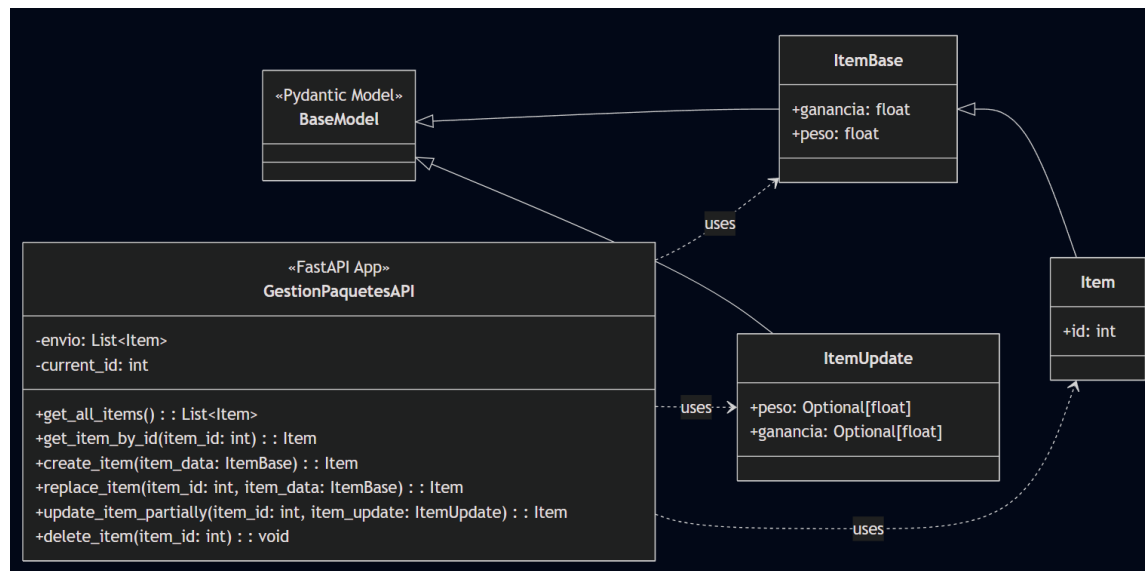
4BV1

ESCUELA SUPERIOR DE  
CÓMPUTO

**TECNOLOGÍAS PARA EL DESARROLLO DE APLICACIONES  
WEB**

25/09/2025

# 1. Diagrama UML



# 2. Implementación

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

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

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.

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

```

1  #Creamos la instancia de FastAPI
2  #Es el corazon de nyestro programa (API)
3  app = FastAPI(
4      #Nombre y descripcion de la API
5      title= "Practica 2: Metodos GET, PUT, POST, DELETE Y PATCH",
6      description= "API para almacenar, editar, actualizar y borrar la información de los paquetes (items) a ser enviados"
7  )

```

```

1  #ItemBase es la estructura base de un item, con los campos ganancia y peso
2  #Digamos que es lo que tiene cada item
3  class ItemBase(BaseModel):
4      ganancia: float
5      peso: float
6  #Item contiene el indentificador unico (id) y hereda los campos de ItemBase
7  class Item(ItemBase):
8      id: int
9  #ItemUpdate es una clase especial utilizada para cuando queremos actualizar un item parcialmente
10 #Los campos son opcionales, ya que podemos querer actualizar solo uno de ellos
11 #El valor por defecto es None
12 class ItemUpdate(BaseModel):
13     peso: Optional[float] = None
14     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.

```

1  #Creamos una lista llamada envio para almacenar los items
2  envio: List[Item] = []
3  #Variable para tener el control dek ud qye se le asigna a cada item
4  current_id = 0
5

```

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.