Módulo 2: "Hola Mundo" en FastAPI y la Magia de Swagger

En el Módulo 1, establecimos el marco conceptual: la arquitectura REST. Ahora, vamos a implementar esa arquitectura usando FastAPI. Veremos cómo los principios de REST (verbos HTTP, URIs) se materializan en código Python y cómo FastAPI nos recompensa por definir nuestro código de forma estructurada.

2.1. Nuestro primer *Endpoint*

Un **"Endpoint" (punto final)** es la combinación de un método HTTP (verbo) y una URI (ruta) que realiza una función específica. Por ejemplo, GET /users es un *endpoint* para leer usuarios. Vamos a crear el *endpoint* más simple posible: GET /.

Creando el main.py

Siguiendo la estructura de proyecto del Módulo 0, crea un archivo llamado main.py.

```
# main.py

# 1. Importar la clase FastAPI
from fastapi import FastAPI

# 2. Crear una instancia de FastAPI
# Esta instancia 'app' será el núcleo de nuestra API.
app = FastAPI()

# 3. Definir el "decorador" de la operación de ruta.
# @app: Se refiere a nuestra instancia.
# .get: Es el método HTTP (el "verbo" REST).
# ("/"): Es la URI (la "ruta" o "path").
@app.get("/")
def read_root():
    """
    Este es el endpoint raíz.
    Devuelve un saludo simple.
    """
    # 4. La función que se ejecuta.
    # FastAPI convertirá automáticamente este diccionario de Python
    # en una respuesta JSON.
    return {"Hello": "World", "message": "Bienvenido a nuestra API"}
```

Este bloque de código es un servicio web completo. Analicémoslo:

- **Decorador (@app.get(...)):** En Python, un decorador es una función que modifica el comportamiento de *otra* función. Aquí, @app.get("/") le dice a FastAPI: "Cuando recibas una petición GET en la ruta /, ejecuta la función que está justo debajo (read_root)".
- **La Función (read_root()):** Es una función estándar de Python. Lo que devuelva (return) será el cuerpo (*body*) de la respuesta enviada al cliente.
- **Conversión a JSON:** Nota que devolvemos un dict de Python. FastAPI (gracias a Pydantic) detecta esto y lo serializa automáticamente al formato JSON {"Hello": "World", ...}, estableciendo la cabecera Content-Type: application/json como dicta el principio de "Representaciones" de REST.

Ejecutando el Servidor

Este archivo main.py es solo un script. Necesita un *servidor* que lo ejecute y escuche peticiones. Ese servidor es **Uvicorn**.
Abre tu terminal (asegúrate de que tu entorno virtual esté activado) y ejecuta:
uvicorn main:app --reload

Desglosemos este comando:

- uvicorn: El programa servidor ASGI que estamos ejecutando.
- main: El nombre de nuestro archivo Python (main.py).
- app: El nombre de la variable *dentro* de main.py que contiene la instancia de FastAPI.
- --reload: (Opcional, pero vital para desarrollo). Le dice a Uvicorn que vigile el archivo main.py. Si guardas un cambio, el servidor se reiniciará automáticamente.

Si todo va bien, verás algo como:
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [12345]
INFO: Started server process [12347]
INFO: Waiting for application startup.
INFO: Application startup complete.

¡Tu API está viva!
Abre tu navegador y ve a http://127.0.0.1:8000. Deberías ver la respuesta:
{"Hello":"World","message":"Bienvenido a nuestra API"}

2.2. Documentación Automática: Swagger UI y OpenAPI

Aquí es donde FastAPI brilla intensamente y justifica su elección.
Mientras tu servidor uvicorn sigue ejecutándose, abre en tu navegador la siguiente dirección:
http://127.0.0.1:8000/docs
Lo que verás es una página web completa, generada automáticamente, que documenta tu API. Esto es **Swagger UI**.

Esta página te permite:

- Ver todos los *endpoints* de tu API (por ahora, solo GET /).
- Ver la descripción (que FastAPI toma del *docstring* de la función).
- Hacer clic en "Try it out" (Probar) y "Execute" (Ejecutar).
- Ver la respuesta del servidor, los códigos de estado y las cabeceras.

¡Acabas de obtener documentación interactiva gratis!

¿Qué es OpenAPI?

Lo que estás viendo (Swagger UI) es solo la *interfaz gráfica*. Esta interfaz se alimenta de un archivo de especificación.
Ve a: **http://127.0.0.1:8000/openapi.json**
Verás un JSON que describe *toda* tu API de forma estructurada. Esto es la especificación **OpenAPI** (anteriormente conocida como "Swagger").

- **OpenAPI:** Es la *especificación*, el "plano" o el "contrato" de la API. Es un estándar abierto (definido por la OpenAPI Initiative) para describir APIs REST de forma agnóstica al lenguaje. Es un archivo JSON o YAML.
- **Swagger UI:** Es una *herramienta* (entre muchas) que sabe leer un archivo openapi.json y renderizarlo como una página web interactiva.

FastAPI genera el openapi.json automáticamente basándose en tu código (tus rutas, tus funciones, y como veremos, tus modelos Pydantic). Luego, expone ese JSON a través de la herramienta Swagger UI en /docs.

Alternativa: ReDoc

FastAPI también incluye *otra* interfaz de documentación, llamada **ReDoc**.
Ve a: **http://127.0.0.1:8000/redoc**
ReDoc es otra herramienta que lee el mismo openapi.json, pero presenta la información de una manera diferente: en una sola página, más limpia y menos "interactiva", ideal para una documentación estática.

2.3. Parámetros de Ruta (*Path Parameters*)

Nuestro *endpoint* GET / es estático. ¿Qué pasa si queremos un *endpoint* dinámico, como el GET /users/123 que discutimos en el Módulo 1?
Ese 123 es un **Parámetro de Ruta (Path Parameter)**. Se define usando llaves {} en la URI.
Añadamos un nuevo *endpoint* a nuestro main.py:

```
# main.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}
```

```
# --- NUEVO ENDPOINT ---
# Observa la sintaxis {item_id} en la ruta.
@app.get("/items/{item_id}")
def read_item(item_id):
    # FastAPI pasa el valor de la URL como argumento a la función.
    # El nombre del argumento DEBE coincidir con el nombre en la ruta.
    return {"item_id_recibido": item_id}
```

(Tu servidor uvicorn debería haberse recargado automáticamente. Si no, reinícialo).
Ahora, si vas a http://127.0.0.1:8000/docs, verás tu nuevo *endpoint* GET /items/{item_id}.
Si lo pruebas en el navegador:

- http://127.0.0.1:8000/items/5 -> {"item_id_recibido":"5"}
- http://127.0.0.1:8000/items/hola -> {"item_id_recibido":"hola"}

Tipado Estricto (Data Validation)

Lo anterior funciona, pero item_id siempre se recibe como un *string*. ¿Qué pasa si *sabemos* que el ID de un ítem debe ser un *entero*?
Aquí es donde FastAPI se integra con los **type hints** (pistas de tipo) de Python.
Modifiquemos la función read_item:

```
# main.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int):  # <--- ¡HEMOS AÑADIDO ': int'!
    # Ahora, FastAPI no solo pasa el valor, sino que
    # 1. VALIDA que sea un entero.
    # 2. CONVIERTE el string "5" al entero 5.
    return {"item_id_recibido": item_id, "es_entero": isinstance(item_id, int)}
```

Prueba esto (después de que el servidor recargue):

1. Ve a http://127.0.0.1:8000/items/5
    - **Respuesta:** {"item_id_recibido":5, "es_entero":true}
    - FastAPI validó que "5" podía ser int y lo convirtió.
2. Ve a http://127.0.0.1:8000/items/hola
    - **Respuesta:**
      {

```
 "detail": [
  {
   "type": "int_parsing",
   "loc": ["path", "item_id"],
   "msg": "Input should be a valid integer, unable to parse string as an integer",
   "input": "hola"
  }
 ]
}
```


- ¡Esto es increíble! Nuestra función read_item **ni siquiera se ejecutó**.
- FastAPI interceptó la petición, intentó validar item_id como int, falló, y automáticamente devolvió un error **HTTP 422 (Unprocessable Entity)**, explicando exactamente qué campo (item_id) y por qué (unable to parse string as an integer).

Este es el poder de la validación de datos nativa que discutimos en el Módulo 0. Simplemente añadiendo : int, hemos hecho nuestra API más robusta.

Código Completo del Módulo 2

Tu main.py al final de este módulo debería verse así:

```
# main.py
from fastapi import FastAPI

# Crear la instancia de la aplicación
app = FastAPI(
    title="API del Curso",
    description="Esta es una API de prueba para el curso de FastAPI.",
    version="0.1.0",
)

@app.get("/")
def read_root():
    """
    Endpoint raíz de la API.
    """
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int):
    """
    Endpoint para leer un ítem por su ID (debe ser un entero).

    - **item_id**: El ID del ítem a recuperar (obligatorio).
    """
```

```
    return {"item_id": item_id, "tipo_dato": str(type(item_id))}

@app.get("/users/{user_name}")
def read_user(user_name: str):
    """
    Endpoint para leer un usuario por su nombre.

    - **user_name**: El nombre del usuario (obligatorio).
    """
    return {"user_name": user_name, "tipo_dato": str(type(user_name))}
```