¿Qué es FastAPI?

El framework mas veloz para el desarrollo web con Python. Enfocado para realizar APIs, es el mas rápido en lo que respecta a la velocidad del servidor superando a Node.Js y a GO. Fue creado por Sebastian Ramirez, es de código abierto y se encuentra en Github, es usado por empresas como Uber, Windows, Netflix y Office.

Ubicación de FastAPI en el ecosistema de Python

FastAPI utiliza otros frameworks dentro de si para funcionar

- **Uvicorn**: es una librería de Python que funciona de servidor, es decir, permite que cualquier computadora se convierta en un servidor
- **Starlette**: es un framework de desarrollo web de bajo nivel, para desarrollar aplicaciones con este requieres un amplio conocimiento de Python, entonces FastAPI se encarga de añadirle funcionalidades por encima para que se pueda usar mas fácilmente
- **Pydantic**: Es un framework que permite trabajar con datos similar a pandas, pero este te permite usar modelos los cuales aprovechara FastAPI para crear la API

Hello World: creación del entorno de desarrollo

https://fastapi.tiangolo.com/tutorial/first-steps/ (https://fastapi.tiangolo.com/tutorial/first-steps/)

```
  #Creamos el entorno virtual
- virtualenv venv --python=3.9
  #Activamos el entorno virtual
- source venv/Scripts/activate
  #Instalamos las dependencias de fastAPI y ucicorn
- pip install fastapi uvicorn
```

```
from fastapi import FastAPI

app = FastAPI()


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

```
uvicorn main:app --reload

INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [28720]
INFO:     Started server process [28722]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Documentación interactiva de una API

FastAPI también está parado sobre los hombros de OpenAPI, el cual es un conjunto de reglas que permite definir cómo describir, crear y visualizar APIs. Es un conjunto de reglas que permiten decir que una API está bien definida.

OpenAPI necesita de un software, el cual es Swagger, que es un conjunto de softwares que permiten trabajar con APIs. FastAPI funciona sobre un programa de Swagger el cual es Swagger UI, que permite mostrar la API documentada.

Acceder a la documentación interactiva con Swagger UI:

```
{localhost}/docs
```

Acceder a la documentación interactiva con Redoc:

```
{localhost}/redoc
```

Path Operations

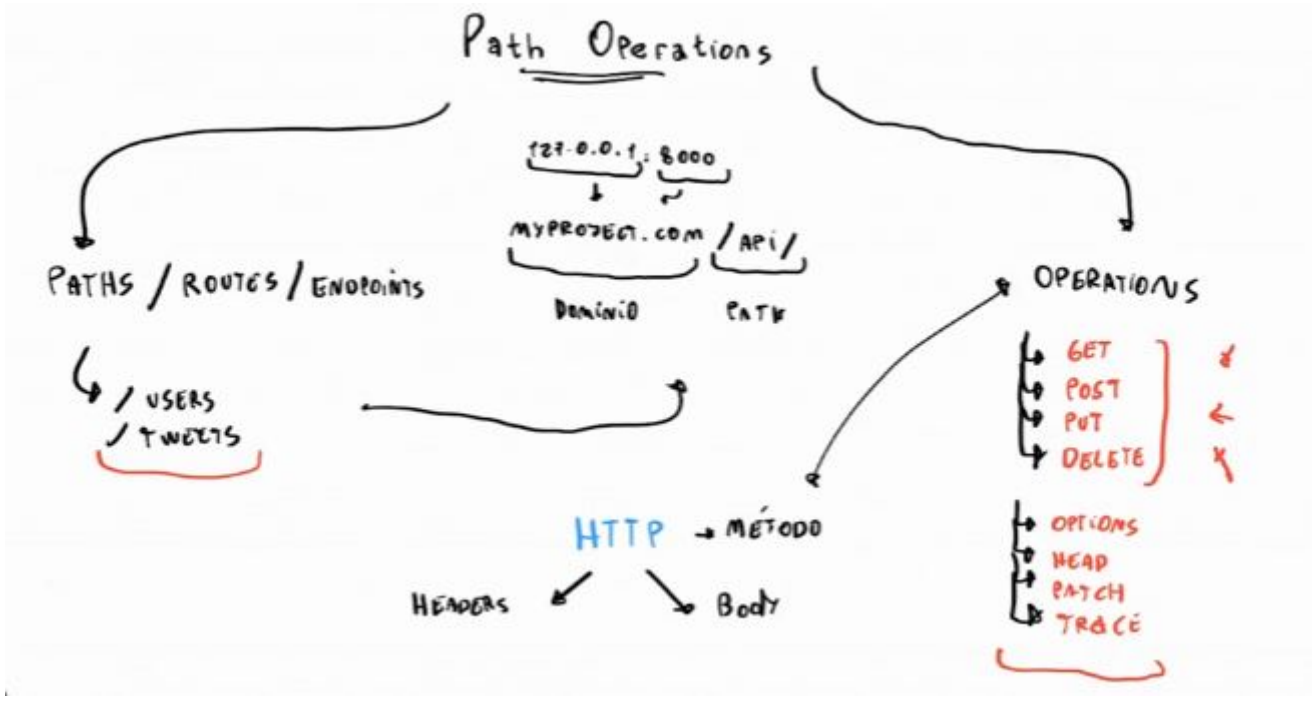
Un path es lo mismo que un route o endpoints y es todo aquello que vaya después de nuestro dominio a la derecha del mismo.

¿Que son las operations?

Un operations es exactamente lo mismo que un método http y tenemos las siguientes más populares:

- GET
- POST
- PUT
- DELETE

Y otros métodos como **OPTIONS, HEAD, PATCH**

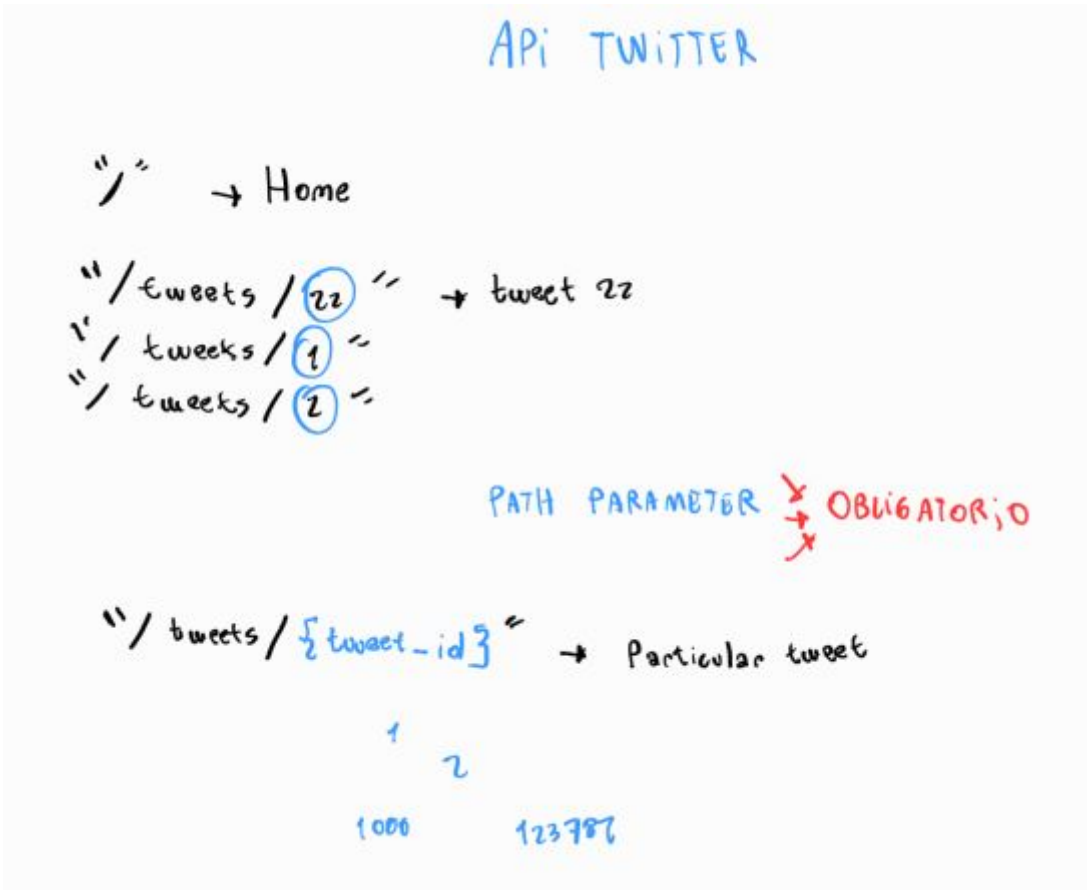


Path Parameters

https://fastapi.tiangolo.com/tutorial/path-params/ (https://fastapi.tiangolo.com/tutorial/path-params/)

Los parámetros de ruta son partes variables de una ruta URL . Por lo general, se utilizan para señalar un recurso específico dentro de una colección, como un usuario identificado por ID. Una URL puede tener varios parámetros de ruta.

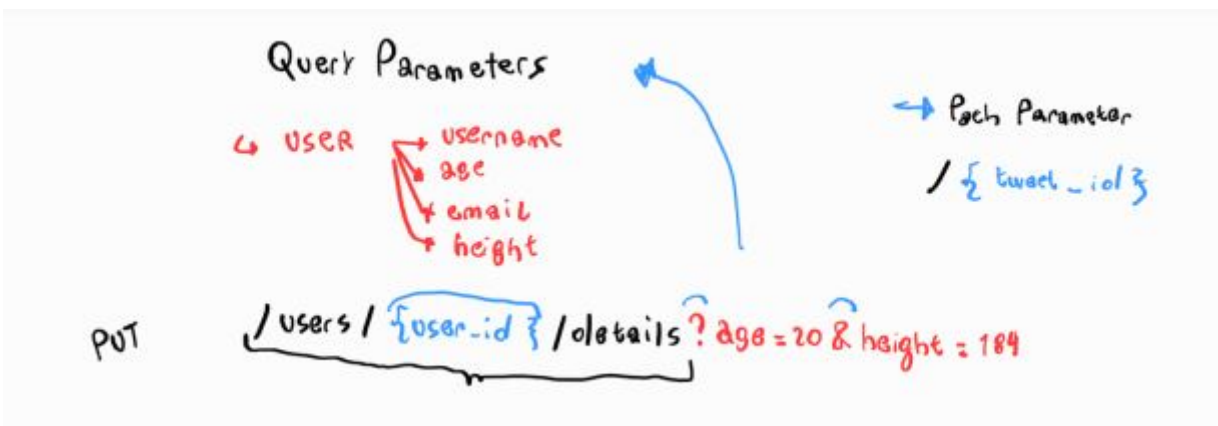
```
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_item(item_id):
    return {"item_id": item_id}
```



Query Parameters

https://fastapi.tiangolo.com/tutorial/query-params/ (https://fastapi.tiangolo.com/tutorial/query-params/)

Query parameters: son un conjunto definido de parámetros adjuntos al final de una URL . Son extensiones de la URL que se utilizan para ayudar a definir contenido o acciones específicos en función de los datos que se transmiten.

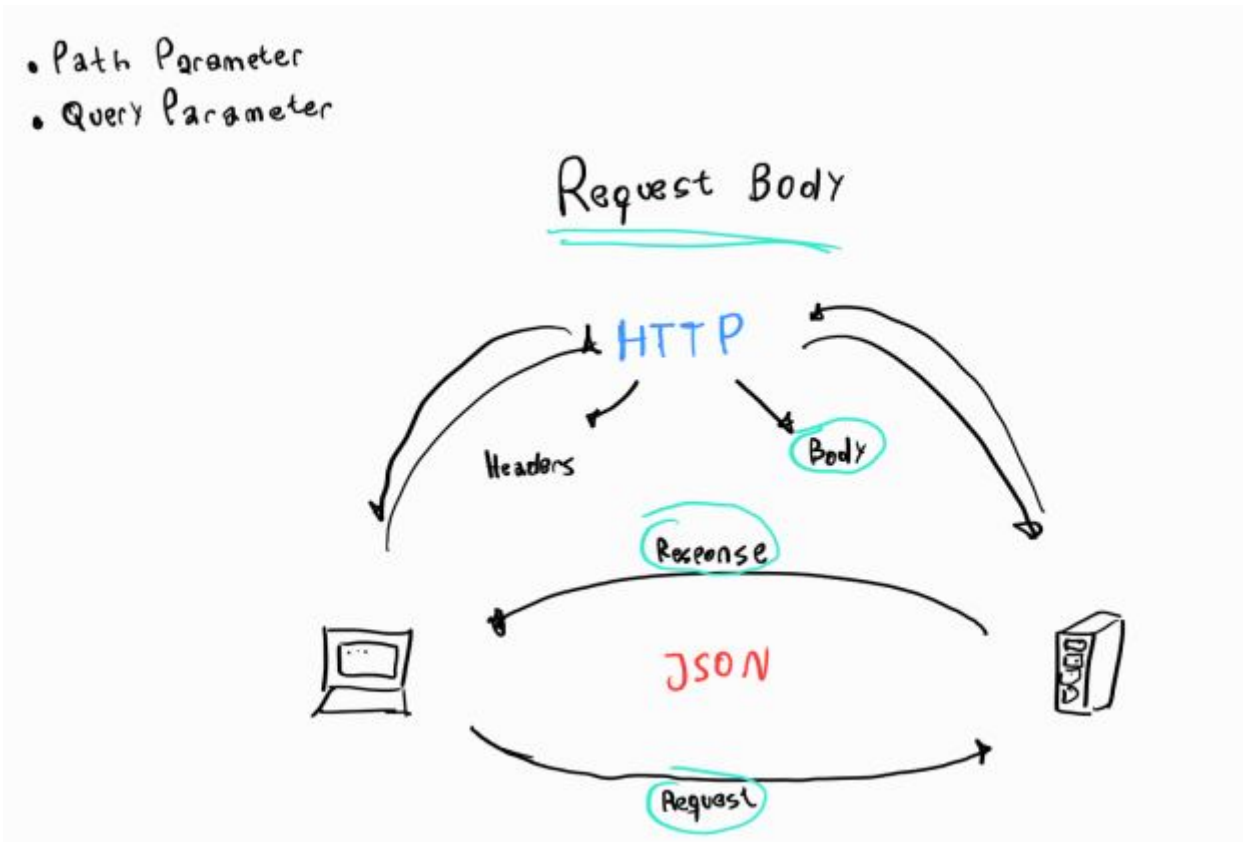


Request Body y Response Body

Documentacion: https://fastapi.tiangolo.com/tutorial/body/ (https://fastapi.tiangolo.com/tutorial/body/)

Un **Request Body** son datos enviados por el cliente a su API.

Un **Response Body** son los datos que su API envía al cliente.



Models

Documentacion Oficial: https://fastapi.tiangolo.com/tutorial/sql-databases/
(https://fastapi.tiangolo.com/tutorial/sql-databases/)

Un modelo es la representacion de una entidad en codigo, al menos de una manera descriptiva.

¿Como luce un modelo dentro de FastAPI?

Modelo pydantic para validar datos:

```
from typing import List, Optional

from pydantic import BaseModel


class ItemBase(BaseModel):
    title: str
    description: Optional[str] = None


class ItemCreate(ItemBase):
    pass


class Item(ItemBase):
    id: int
    owner_id: int

    class Config:
        orm_mode = True


class UserBase(BaseModel):
    email: str


class UserCreate(UserBase):
    password: str


class User(UserBase):
    id: int
    is_active: bool
    items: List[Item] = []

    class Config:
        orm_mode = True
```

Un modelo para mapear los datos a la base de datos (ORM, SQLalchemy)

```
from sqlalchemy import Boolean, Column, ForeignKey, Integer, String
from sqlalchemy.orm import relationship

from .database import Base


class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True)
    hashed_password = Column(String)
    is_active = Column(Boolean, default=True)

    items = relationship("Item", back_populates="owner")


class Item(Base):
    __tablename__ = "items"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, index=True)
    description = Column(String, index=True)
    owner_id = Column(Integer, ForeignKey("users.id"))
    owner = relationship("User", back_populates="items")
```

Validaciones: Query Parameters

```
#Python
from typing import Optional

#FastAPI
from fastapi import FastAPI
from fastapi import Body, Query

app = FastAPI()

@app.get("/person/detail")
def show_person(
    name: Optional[str] = Query(None, min_length=1, max_length=50),
    age: str = Query(...)
):
    return {name: age}
```

Validaciones: Path Parameters

```
#Pydantic
from pydantic import BaseModel

#FastAPI
from fastapi import FastAPI
from fastapi import Path

app = FastAPI()

# Validaciones: Path Parameters

@app.get("/person/detail/{person_id}")
def show_person(
    person_id: int = Path(..., gt=0)
):
    return {person_id: "It exists!"}
```

Validaciones: Request Body

```
#Pydantic
from pydantic import BaseModel

#FastAPI
from fastapi import FastAPI
from fastapi import Path

app = FastAPI()


# Validaciones: Request Body

@app.put("/person/{person_id}")
def update_person(
    person_id: int = Path(
        ...,
        title="Person ID",
        description="This is the person ID",
        gt=0
    ),
    person: Person = Body(...),
    location: Location = Body(...)
):
    results = person.dict()
    results.update(location.dict())
    return results
```

Validaciones: Models

```python
#Python
from typing import Optional
from enum import Enum

#Pydantic
from pydantic import BaseModel
from pydantic import Field

#FastAPI
from fastapi import FastAPI
from fastapi import Body, Query, Path

app = FastAPI()

# Models

class HairColor(Enum):
    white = "white"
    brown = "brown"
    black = "black"
    blonde = "blonde"
    red = "red"

class Location(BaseModel):
    city: str
    state: str
    country: str

class Person(BaseModel):
    first_name: str = Field(
        ...,
        min_length=1,
        max_length=50,
        example="Miguel"
        )
    last_name: str = Field(
        ...,
        min_length=1,
        max_length=50,
        example="Torres"
        )
    age: int = Field(
        ...,
        gt=0,
        le=115,
        example=25
    )
    hair_color: Optional[HairColor] = Field(default=None, example=HairColor.black)
    is_married: Optional[bool] = Field(default=None, example=False)

    # class Config:
    #     schema_extra = {
    #         "example": {
    #             "first_name": "Facundo",
    #             "last_name": "García Martoni",
    #             "age": 21,
    #             "hair_color": "blonde",
    #             "is_married": False
    #         }
    #     }

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

Tipos de datos especiales

https://pydantic-docs.helpmanual.io/usage/types/#pydantic-types (https://pydantic-docs.helpmanual.io/usage/types/#pydantic-types)

Tipos de Datos Especiales

Pydantic → Field

Clásicos

- str
- int
- Float
- bool

Exóticos

- Enum
- HttpUrl
  - ✓ → http://myproject.com
  - ✓ → www.platzi.com
  - ✗ → hola
- FilePath
  - ✓ C:/windows/system32/432.dll
- DirectoryPath
  - ✓ //mnt/c/Some folder
- EmailStr
  - ✓ hola@hola.com
  - ✗ Facundo.com
- PaymentCardNumber
- IPvAnyAddress
- NegativeFloat
- PositiveFloat
- NegativeInt
- PositiveInt