Ubicación de FastAPI en el ecosistema de Python

FastAPI utiliza otros frameworks dentro de sí 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 añadir funcionalidades por encima para que se pueda usar más fácilmente

Pydantic: Es un framework que permite trabajar con datos similar a pandas, pero este te permite usar modelos los cuales aprovechará FastAPI para crear la API

Lecturas recomendadas

Uvicorn
https://www.uvicorn.org/

Starlette
https://www.starlette.io/

Pydantic
https://pydantic-docs.helpmanual.io/

Hello World: creación del entorno de desarrollo

Lecturas recomendadas

First Steps - FastAPI
https://fastapi.tiangolo.com/tutorial/first-steps/

Hello World: elaborando el código de nuestra primer API

Para correr la app desde la terminal se escribe…

uvicorn main:app - -reload

Documentación interactiva de una API

OPEN API: es una especificación que define como describir, crear y visualizar API's. Permite reconocer si una API está definida adecuadamente. Requiere de Swagger.

Swagger: software para trabajar API's.

ReDoc es una alternativa de Swagger instalada por default con FastAPI.

FastAPI funciona sobre **SwaggerUI** (User Interface) que permite mostrar gráficamente la API documentada. **SwaggerUI** obtiene especificaciones de **OPEN API** y la muestra por **Fast API**.

- Acceder a la documentación interactiva con Swagger UI:
  {localhost}/docs

- Acceder a la documentación interactiva con Redoc:
  {localhost}/redoc

Lecturas recomendadas

https://www.openapis.org/
https://www.openapis.org/

GitHub - Redocly/redoc: 📘 OpenAPI/Swagger-generated API Reference Documentation
https://github.com/Redocly/redoc

API Documentation & Design Tools for Teams | Swagger
https://swagger.io/

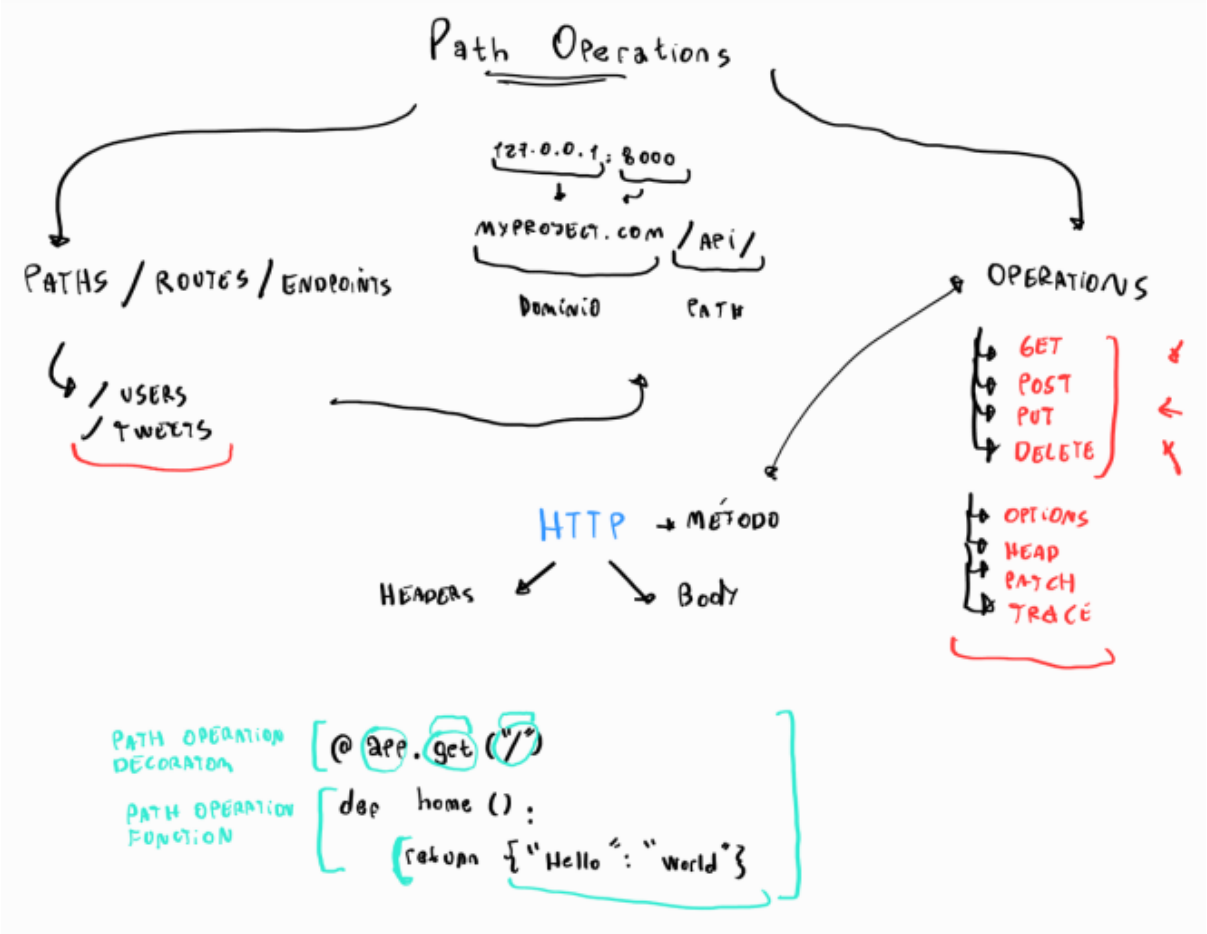
Path Operations

Path: es una ruta (route o endpoint) la cual nosotros ingresamos seguido el dominio de nuestro aplicativo
Operation: Es un método http por el cual nos comunicamos, existen los siguientes:

- **Get**: Obtener, solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
- **Post**: Crear, se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
- **Put**: Modificar/Actualizar, reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
- **Delete**: Eliminar, borrar un recurso en específico.
- **Options**: Es utilizado para describir las opciones de comunicación para el recurso de destino.
- **Head**: pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.
- **Patch**: es utilizado para aplicar modificaciones parciales a un recurso.

- **Trace**: realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.

Path Operation Decorator: permite el ingreso de una path a una operation designada para la ejecución de un código siguiente:
Path Operation Function: realiza la ejecución de un código siguiendo la especificación del path operation decorator llamado anteriormente.



Path Parameters

Un path parameter es una variable incluida en la ruta url, con la cual especificamos la espera de un cierto dato para el envió hacia dicha ruta, una url puede contener varios path parameters.

Query Parameters

Un Query Patameter es un conjunto de elementos opcionales los cuales son añadidos al finalizar la ruta, con el objetivo de definir contenido o acciones en la url,
estos elementos se añaden después de un **?**
para agregar más query parameters utilizamos **&**

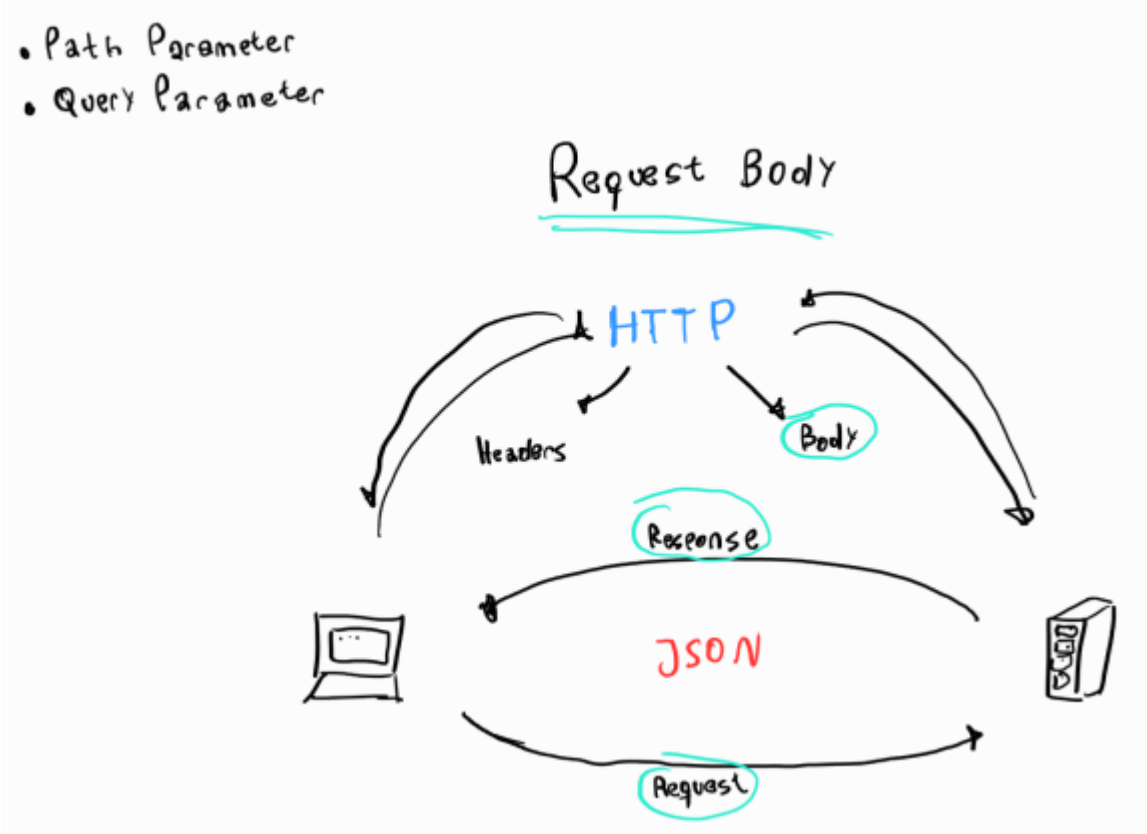
Request Body y Response Body

Debes saber que bajo el protocolo **HTTP** existe una comunicación entre el usuario y el servidor. Esta comunicación está compuesta por cabeceras (**headers**) y un cuerpo (**body**). Por lo mismo, se tienen dos direcciones en la comunicación entre el cliente y el servidor y definen de la siguiente manera:

- **Request** : Cuando el cliente solicita/pide datos al servidor.
- **Response** : Cuando el servidor responde al cliente.

Request Body
Con lo anterior mencionado, **Request Body** viene a ser el cuerpo (**body**) de una **solicitud** del cliente al servidor.

Response Body
Con lo anterior mencionado, **Response Body** viene a ser el cuerpo (**body**) de una **respuesta** del servidor al cliente.



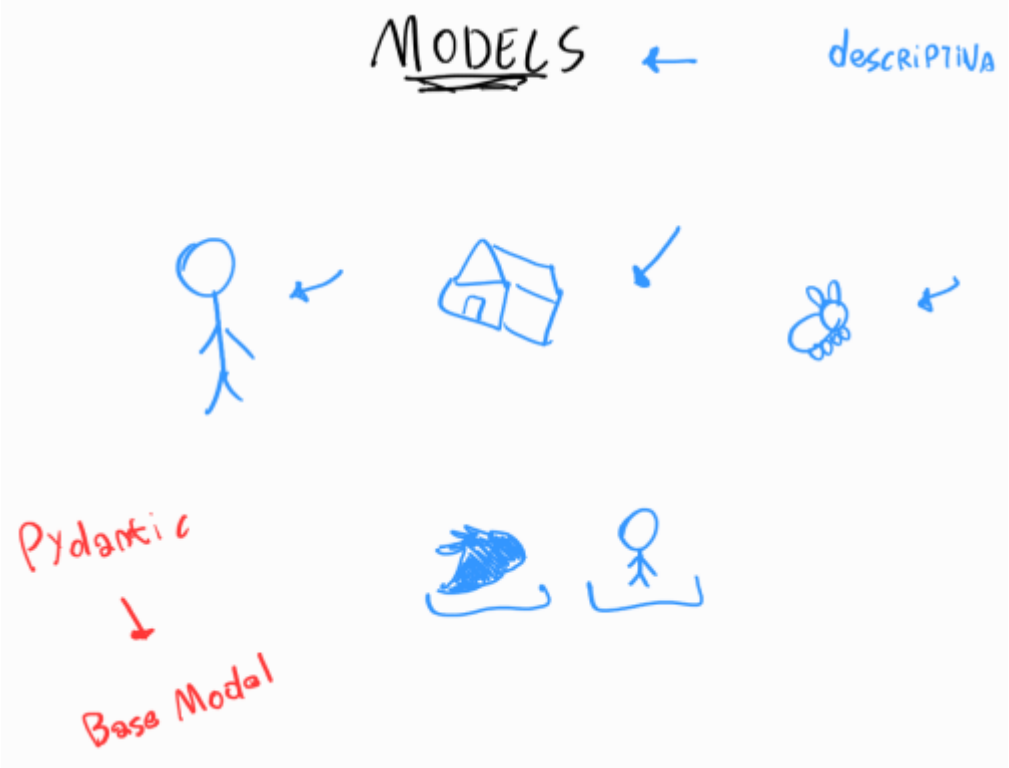
Models

Los modelos son la representación de una entidad en código, una entidad es un objeto de la vida real, que tiene ciertos atributos. Por ejemplo:

- Carro: color, motor, año, marca

- Persona: edad, nombres, apellidos, altura

Para poder crear modelos en el código se utiliza la librería Pydantic, importando la clase BaseModel:
from pydantic import BaseModel

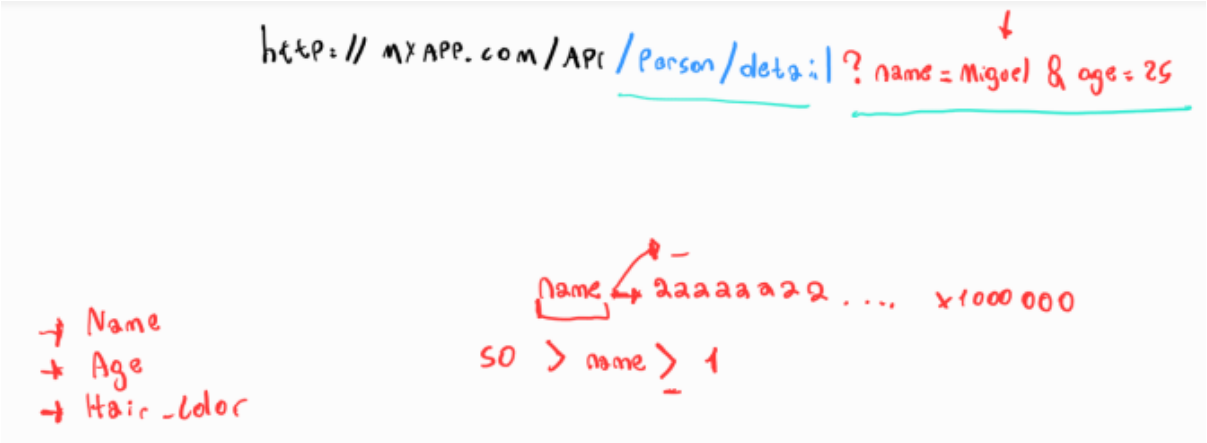


Validaciones: Query Parameters

Validaciones
Las validaciones tal como se definen, nos sirven para comprobar si son correctos los parámetros entregados en cada una de las peticiones. Estas validaciones funcionan restringiendo o indicando el formato de entrega en cada una de las peticiones.

Query parameters
Entonces, se definen las validaciones para las Query Parameters para definir un estándar de consulta y especificar cómo se deben entregar los datos.

EJ:
Se define la siguiente consulta (QUERY):
https://domain.com/user/detail?nombre=Manolo&edad=20



Validaciones: explorando más parameters

Más sobre validaciones
Para especificar las validaciones, debemos pasarle como parámetros a la función Query lo que necesitemos validar.

Para tipos de datos str:

- **max_length** : Para especificar el tamaño máximo de la cadena.
- **min_length** : Para especificar el tamaño mínimo de la cadena.
- **regex** : Para especificar expresiones regulares.

Para tipos de datos int:

- **ge** : (greater or equal than ≥) Para especificar que el valor debe ser mayor o igual.
- **le** : (less or equal than ≤) Para especificar que el valor debe ser menor o igual.
- **gt** : (greater than >) Para especificar que el valor debe ser mayor.
- **lt** : (less than <) Para especificar que el valor debe ser menor.

Ejemplo de uso:

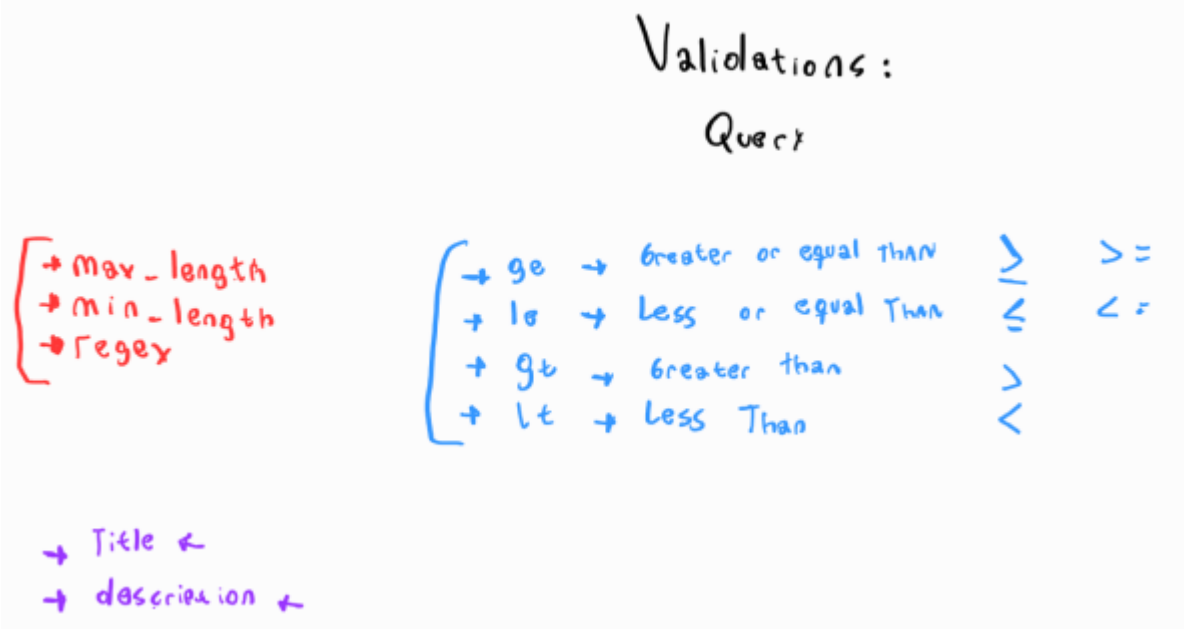
 # Validaciones de un nombre de usuario.
Query(None, min_length=1, max_length=50)):

Es posible dotar de mayor contexto a nuestra documentación. Se deben usar los parámetros title y description.

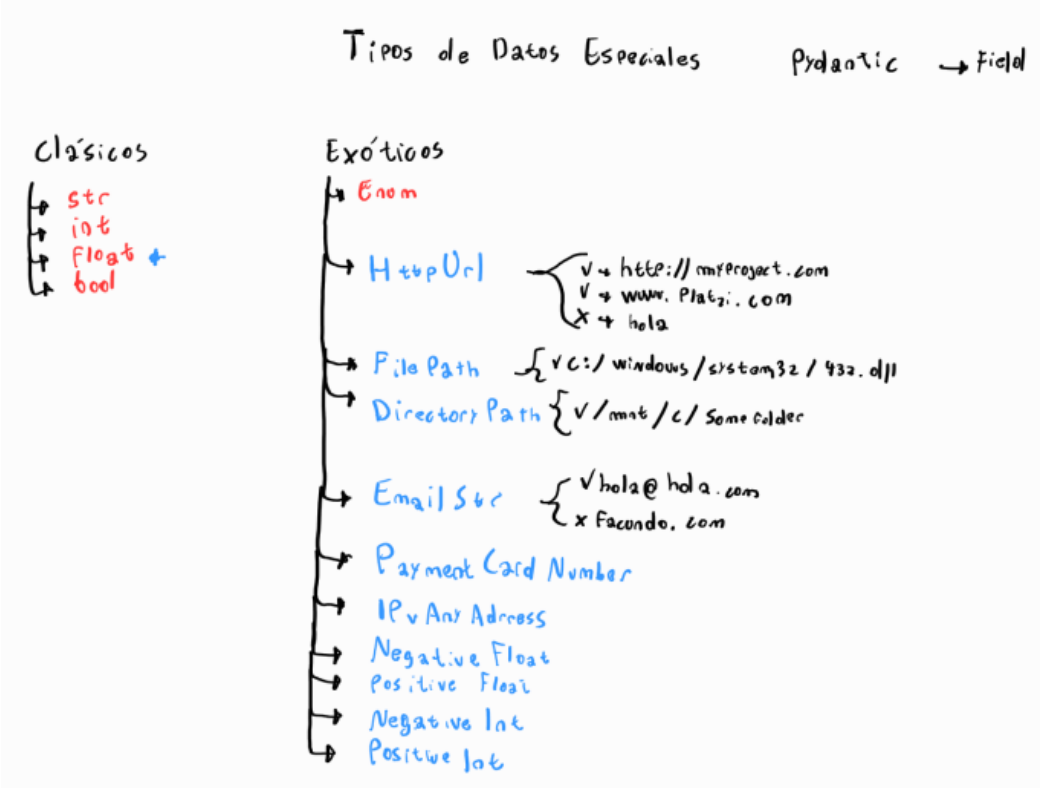
- **title** : Para definir un título al parámetro.
- **description** : Para especificar una descripción al parámetro.

Ejemplo de uso:

 # Validaciones para un Identificador.
Query(
None,
title="ID del usuario",
description="El ID se consigue entrando a las configuraciones del perfil");



Tipos de datos especiales



Todos estos tipos de datos corresponden a Pydantic, se pueden importar al igual que Field.

Tipos de datos clásicos:

- **str** → Cadena de texto
- **int** → Número entero
- **float** → Número flotante (decimal)
- **bool** → Booleano

Tipos de datos exóticos:

Enum → Enumerar caracteres
HttpUrl → Revisa si una URL es válida (***https://myapp.com***, *www.google.com*)
FilePath → Válida si la ruta que envía el cliente es un archivo (*c:/windows/system32/432.dll*)
DirectoryPath → Válida si la ruta que envía el cliente es un directorio (***/mnt/c/someFolder***)
EmailStr → Válida si el cliente ingresa un email (*hola@email.com*)
PaymentCardNumber → Válida si el cliente ingresa un número de tarjeta
IPvAnyAdress → Válida si el cliente ingresa una dirección IP
NegativeFloat → Válida si el cliente ingresa un número negativo de tipo flotante
PositiveFloat → Válida si el cliente ingresa un número positivo de tipo flotante
NegativeInt → Valida si el cliente ingresa un número entero negativo
PositiveInt → Valida si el cliente ingresa un número entero positivo

Lecturas recomendadas

Field Types - pydantic
https://pydantic-docs.helpmanual.io/usage/types/#pydantic-types