## What is uvicorn?

### In Simple Words:

Think of **Uvicorn** as the **engine that runs your FastAPI app** and makes it accessible through your browser or Postman via <http://localhost:8000.>

### 🧩 Summary

**Uvicorn** is the tool that **runs your FastAPI web server**.

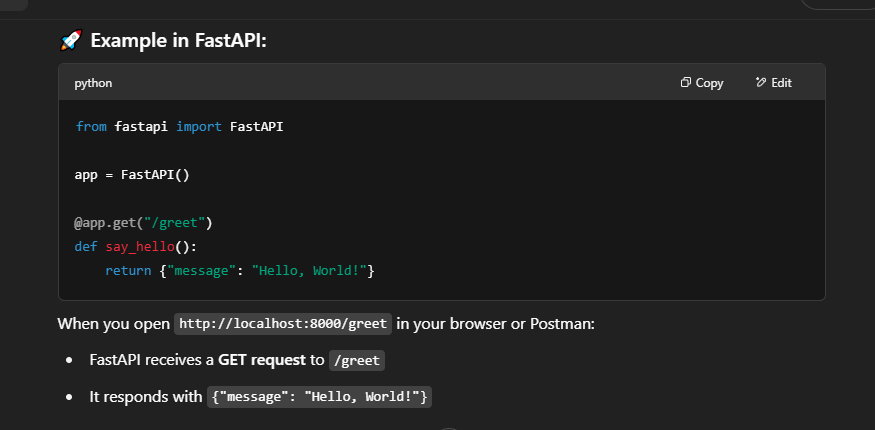
It’s **fast**, **asynchronous**, and **perfect for modern Python web apps**.

### What is a ****GET request****?

A **GET request** is a type of **HTTP method** used to **ask a server for some data** — like saying:

"Hey server, give me this info!"

**Get request:**



**JSON** is the standard format for APIs because it’s lightweight and readable across all platforms (Python, JavaScript, etc.).

Like when we return response in dictionary or list fastapi automatically convert it inot json format beacause of it competabiltiy.

### ****What is JSON?****

**JSON** stands for **JavaScript Object Notation**, and it's a **lightweight** data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate.

It is commonly used for **transferring data between a server and a client** (like between a web server and a browser, or between APIs).

Example of json format

{

"name": "Alice",

"age": 25,

"is\_student": false,

"courses": ["Math", "Physics", "Computer Science"]

}

### 🛣️ ****What Are Path Parameters?****

Path parameters are variables **embedded directly in the URL** of the request. They're typically used when you want to capture part of the URL and pass it to the server as a **parameter**. You define these parameters in your FastAPI route by **adding curly braces** {} around them in the URL.

### What is a POST Request?

A **POST request** is one of the HTTP methods used by the client (usually a web browser or any HTTP client) to send data to the server. Unlike **GET** requests, which are used to retrieve data from the server, **POST** requests are used to **send data** to the server for creating or updating resources.

In the context of **web development** and **APIs**, POST requests are commonly used to:

Submit form data

Upload files

Create a new resource in a database

Trigger actions that affect the server’s state

For example, when a user submits a form to register an account, a POST request might be used to send the registration data to the server.

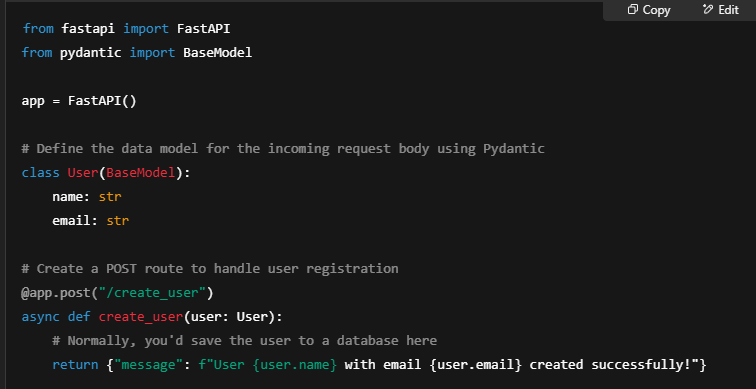
### POST Request in FastAPI:

FastAPI allows you to handle POST requests by creating a route with the @app.post() decorator. You can define the data structure expected in the POST body using **Pydantic models** for validation.

#### Example of Handling a POST Request in FastAPI:

python

CopyEdit



What Are Query Parameters in FastAPI?

**Query parameters** are values passed in the **URL after the** ? **symbol**, used to filter or control the data your API returns.

See example in vscode.

### What does limit mean?

In APIs, limit is a **query parameter** that controls **how many items** you want the API to return.

### 📦 For example:

If your backend has 100 products, but you only want the **first 5**, you can call:

bash

CopyEdit

/products?limit=5

Then the response will include only 5 products instead of all 100.

Difference between path and query paramerts

### ****Path Parameters****

Are part of the **URL path**.

Used to **identify a specific resource**.

Declared **inside the route** like /blog/{id}.

### ****Query Parameters****

Come **after the** ? in the URL.

Used to **filter or modify** the result.

Declared as **function parameters**, not in the path.

### What is a Pydantic Model in FastAPI?

A **Pydantic model** is a Python class that:

**Defines the structure of data** (like JSON requests or responses)

**Automatically validates** the data types

Is used to create **request bodies** and **response formats**

It comes from the pydantic library, which FastAPI is built on.

### How It Works:

The API expects a **JSON object** in the POST request.

FastAPI **automatically validates** that the JSON matches the Student model.

If any field is missing or has the wrong type, FastAPI returns a **400 Bad Request** error with de

**Pydantic automatically validates data types** behind the scenes by checking the JSON request body against the data types you define in your model.