**What is FastAPI?**

FastAPI is a modern, high-performance web framework for building APIs with Python 3.6+ based on standard Python type hints.

* Fast
* Easy to learn
* Automatic docs (Swagger & ReDoc)
* Built-in validation
* Perfect for APIs

# pip install fastapi uvicorn pydantic

# uvicorn main:app --reload

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

from typing import List

# -----------------------------------

# from fastapi import FastAPI, Request

# from fastapi.responses import HTMLResponse

# from fastapi.templating import Jinja2Templates

# app = FastAPI()

# templates = Jinja2Templates(directory="templates")

# @app.get("/", response\_class=HTMLResponse)

# async def read\_html(request: Request):

#     return templates.TemplateResponse("index.html", {"request": request, "name": "Kiran"})

# -----------------------------------------------------

# Simulated in-memory database

students\_db = []

# Schema (Model) for student

class Student(BaseModel):

    id: int

    name: str

    age: int

    course: str

@app.get("/")

def read\_root():

    return {"message": "Welcome to FastAPI CRUD Example!"}

# Get all students

@app.get("/students", response\_model=List[Student])

def get\_students():

    return students\_db

# Get single student

@app.get("/students/{student\_id}", response\_model=Student)

def get\_student(student\_id: int):

    for student in students\_db:

        if student.id == student\_id:

            return student

    raise HTTPException(status\_code=404, detail="Student not found")

# Create student

@app.post("/students", response\_model=Student)

def create\_student(student: Student):

    for s in students\_db:

        if s.id == student.id:

            raise HTTPException(

                status\_code=400, detail="Student ID already exists")

    students\_db.append(student)

    return student

# Update student

@app.put("/students/{student\_id}", response\_model=Student)

def update\_student(student\_id: int, updated\_student: Student):

    for index, s in enumerate(students\_db):

        if s.id == student\_id:

            students\_db[index] = updated\_student

            return updated\_student

    raise HTTPException(status\_code=404, detail="Student not found")

# Delete student

@app.delete("/students/{student\_id}")

def delete\_student(student\_id: int):

    for index, student in enumerate(students\_db):

        if student.id == student\_id:

            del students\_db[index]

            return {"message": "Student deleted successfully"}

    raise HTTPException(status\_code=404, detail="Student not found")

## Postman API Testing

### 1. ****GET all students****

* **URL:** GET http://127.0.0.1:8000/students

### 2. ****POST create student****

* **URL:** POST http://127.0.0.1:8000/students
* **Body (JSON):**

{

  "id": 1,

  "name": "John",

  "age": 21,

  "course": "CS"

}

### 3. ****GET single student****

* **URL:** GET http://127.0.0.1:8000/students/1

### 4. ****PUT update student****

* **URL:** PUT http://127.0.0.1:8000/students/1
* **Body:**

{

  "id": 1,

  "name": "John Wick",

  "age": 22,

  "course": "AI"

}

### 5. ****DELETE student****

* **URL:** DELETE http://127.0.0.1:8000/students/1

curl Commands

Create Student

curl -X POST "http://127.0.0.1:8000/students" -H "Content-Type: application/json" -d "{\"id\":1,\"name\":\"Alice\",\"age\":20,\"course\":\"Python\"}"

Get ALL Students

curl -X GET "http://127.0.0.1:8000/students"

Get one Student

curl -X GET "http://127.0.0.1:8000/students/1"

Update Student

curl -X PUT "http://127.0.0.1:8000/students/1" -H "Content-Type: application/json" -d "{\"id\":1,\"name\":\"Alice Updated\",\"age\":21,\"course\":\"ML\"}"

Delete Student

curl -X DELETE "http://127.0.0.1:8000/students/1"