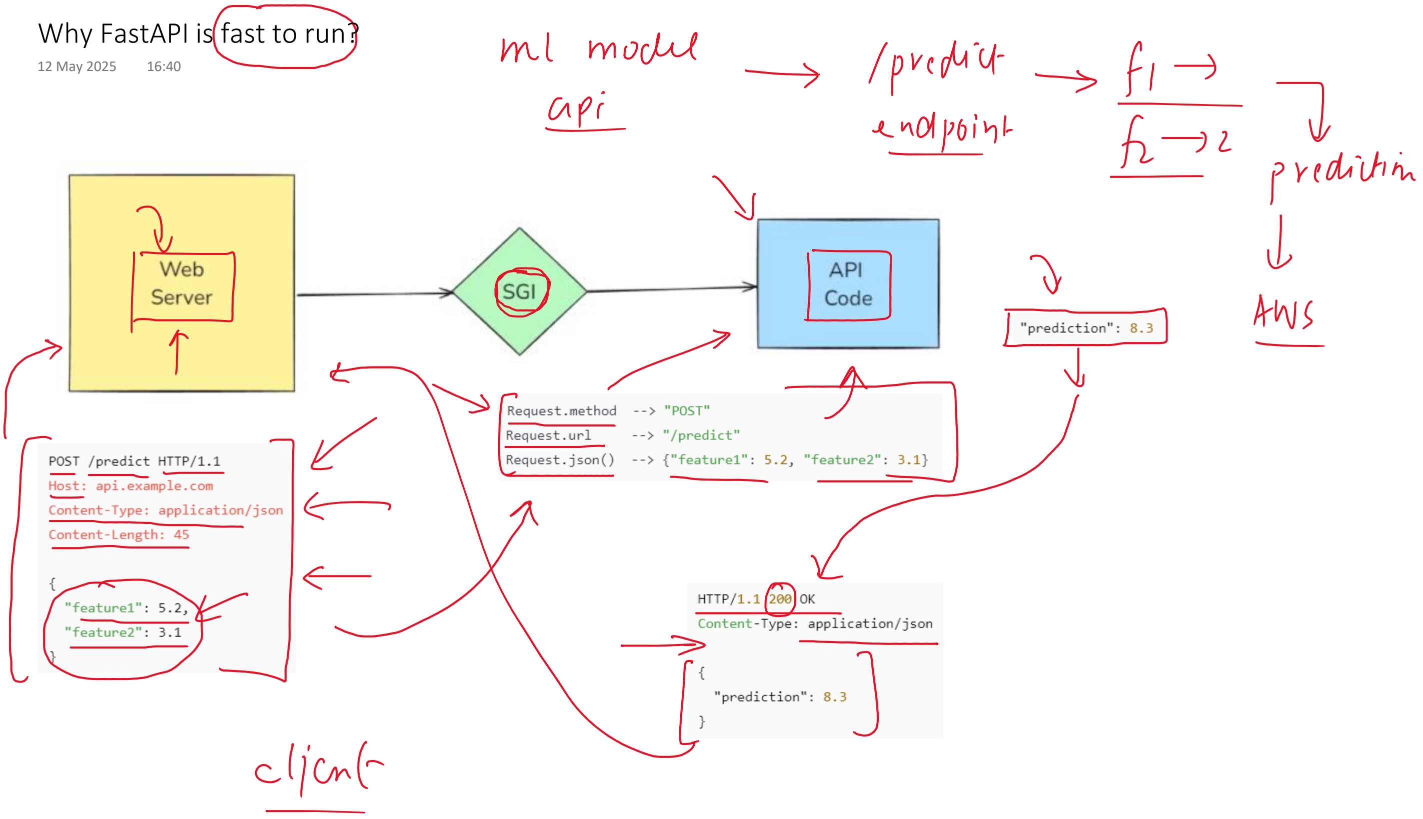
Why FastAPI is fast to run?

12 May 2025 16:40

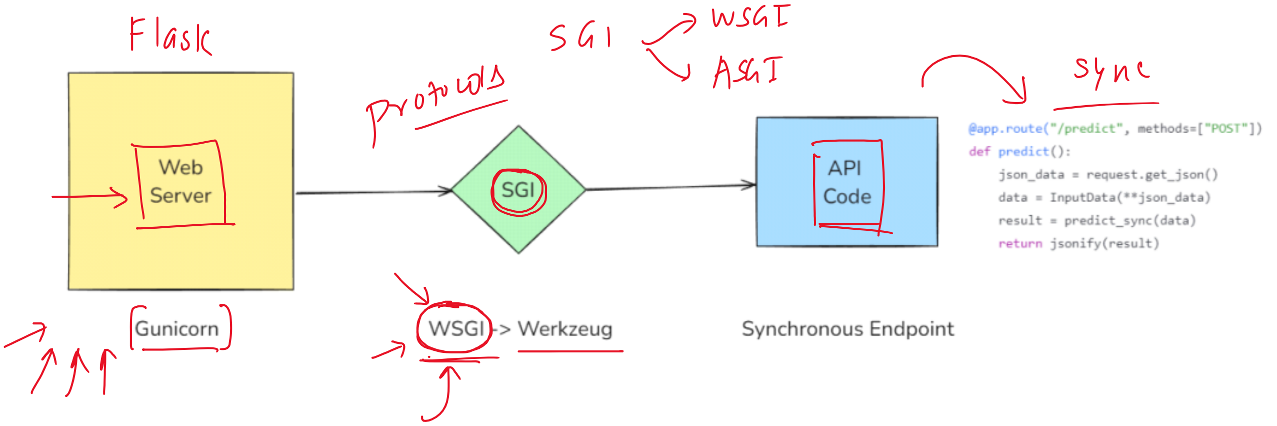


```
POST /predict HTTP/1.1
Host: api.example.com
Content-Type: application/json
Content-Length: 45

{
  "feature1": 5.2,
  "feature2": 3.1
}
```

```
Request.method  --> "POST"
Request.url     --> "/predict"
Request.json()  --> {"feature1": 5.2, "feature2": 3.1}
```

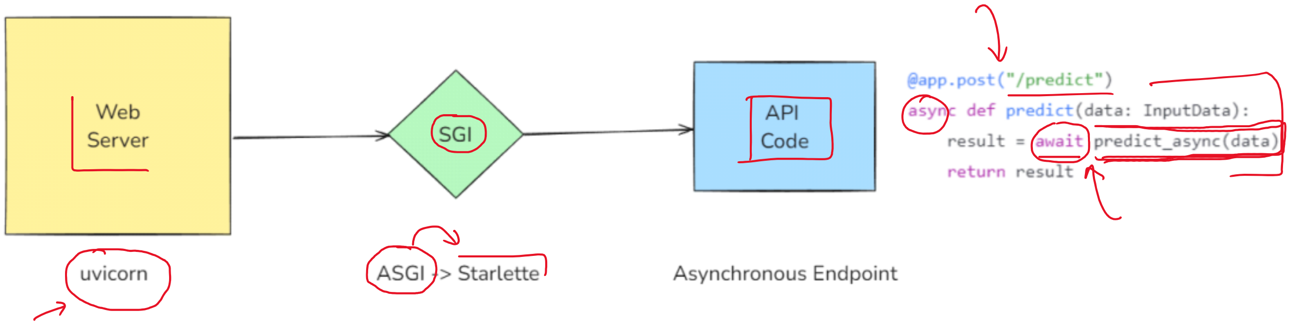
```
"prediction": 8.3
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "prediction": 8.3
}
```



```
@app.route("/predict", methods=["POST"])
def predict():
    json_data = request.get_json()
    data = InputData(**json_data)
    result = predict_sync(data)
    return jsonify(result)
```

Synchronous Endpoint



```
@app.post("/predict")
async def predict(data: InputData):
    result = await predict_async(data)
    return result
```

Asynchronous Endpoint

Why FastAPI is fast to code?

12 May 2025 16:41

1. Automatic Input Validation
2. Auto-Generated Interactive Documentation
3. Seamless Integration with Modern Ecosystem (ML/DL libraries, OAuth, JWT, SQL Alchemy, Docker, Kubernetes etc.)

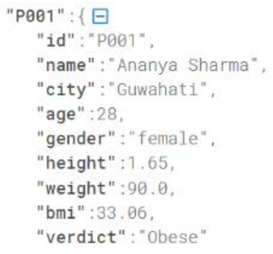
Project Overview

14 May 2025 15:06

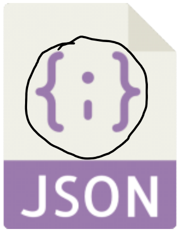
api →

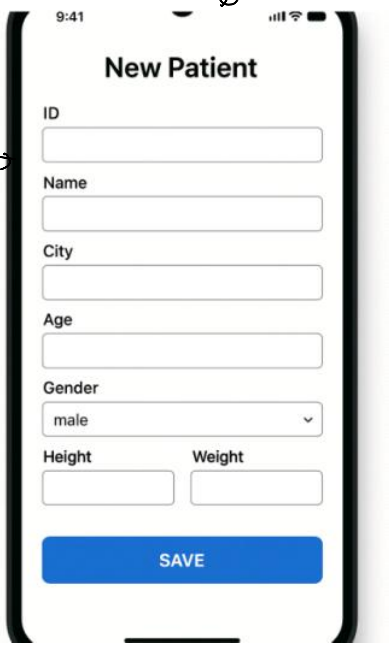
delete

app profile



```
"P001":{
   "id":"P001",
   "name":"Ananya Sharma",
   "city":"Guwahati",
   "age":28,
   "gender":"female",
   "height":1.65,
   "weight":90.0,
   "bmi":33.06,
   "verdict":"Obese"
```





9:41

New Patient

ID

Name

City

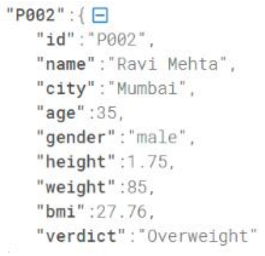
Age

Gender

male

Height Weight

SAVE



```
"P002":{
   "id":"P002",
   "name":"Ravi Mehta",
   "city":"Mumbai",
   "age":35,
   "gender":"male",
   "height":1.75,
   "weight":85,
   "bmi":27.76,
   "verdict":"Overweight"
```

endpoints

/create → json

/view

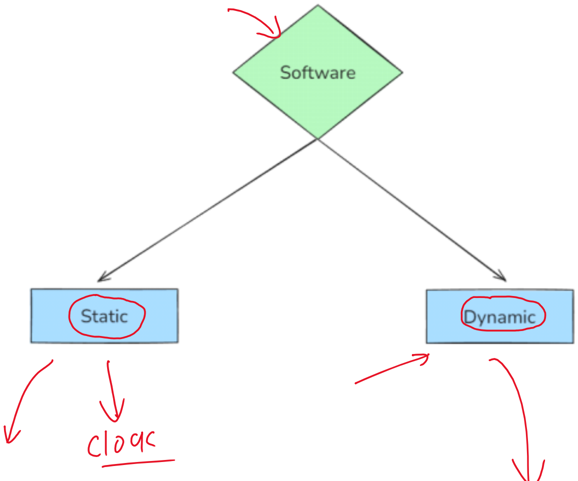
/view/ patient_id

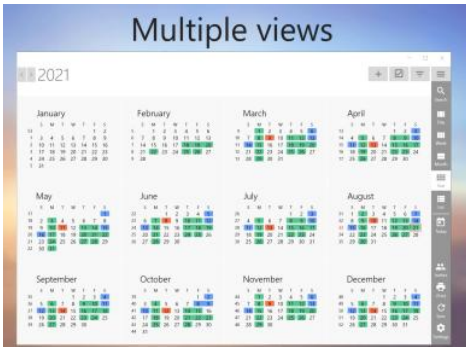
/update/ patient_id

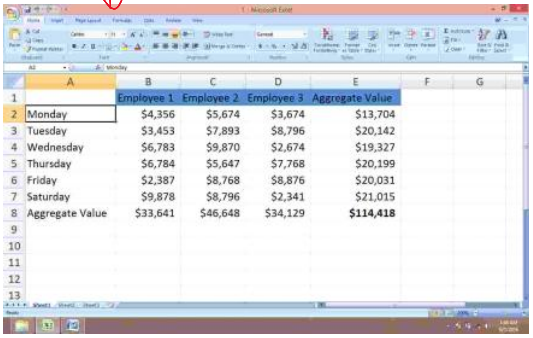
/delete / patient_id

HTTP Methods

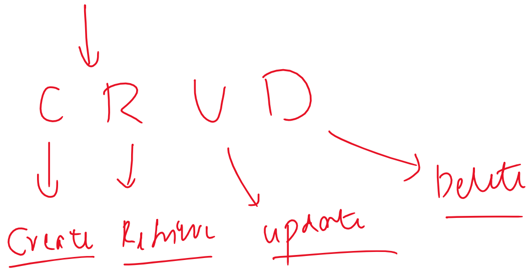
14 May 2025 15:06

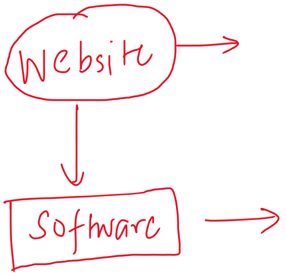


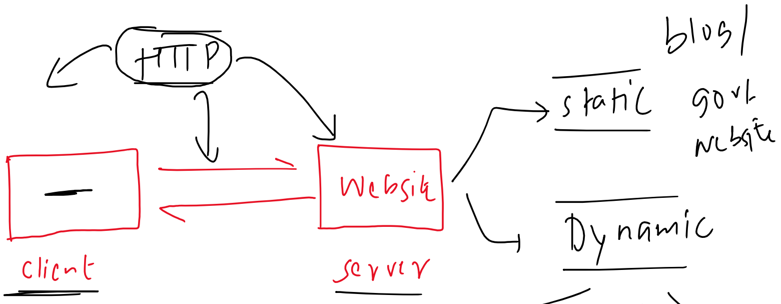


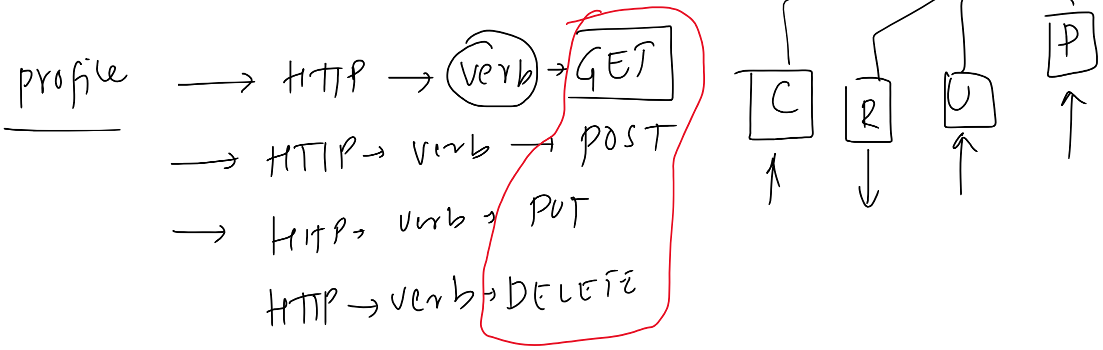


	A	B	C	D	E	F	G
1		Employee 1	Employee 2	Employee 3	Aggregate Value		
2	Monday	$4,356	$5,674	$3,674	$13,704		
3	Tuesday	$3,453	$7,893	$8,796	$20,142		
4	Wednesday	$6,783	$9,870	$2,674	$19,327		
5	Thursday	$6,784	$5,647	$7,768	$20,199		
6	Friday	$2,387	$8,768	$8,876	$20,031		
7	Saturday	$9,878	$8,796	$2,341	$21,015		
8	Aggregate Value	$33,641	$46,648	$34,129	$114,418		
9							
10							
11							
12							
13							









Path Params

15 May 2025 16:14

Path parameters are dynamic segments of a URL path used to identify a specific resource.

localhost:8000 / view → all patients

→ localhost:8000/view/3 4 → specific record health

retrieve →

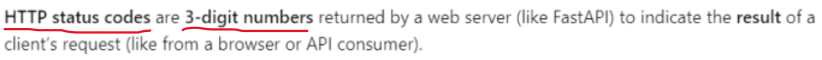
update →

delete →

endpoint →

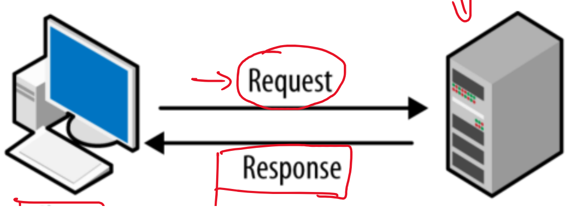
The Path() function in FastAPI is used to provide metadata, validation rules, and documentation hints for path parameters in your API endpoints.

Title
Description
Example
ge, gt, le, lt
Min_length
Max_length
regex

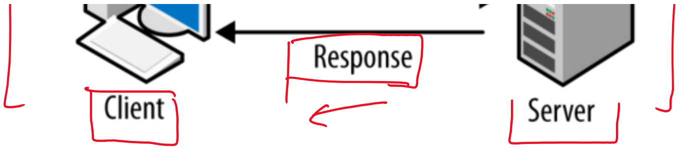


HTTP status codes are **3-digit numbers** returned by a web server (like FastAPI) to indicate the **result** of a client's request (like from a browser or API consumer).

HTTP



HTTP [2xx]



They help the **client (browser, frontend, mobile app, etc.) understand**:

- whether the request was successful,
- whether something went wrong,
- and what kind of issue occurred (if any).

2xx	✅ Success	The request was successfully received and processed
3xx	🔁 Redirection	Further action needs to be taken (e.g., redirect)
4xx	⚠️ Client Error	Something is wrong with the request from the client
5xx	❌ Server Error	Something went wrong on the server side

problem

200 OK	Standard success	A `GET` or `POST` succeeded
201 Created	Resource created	After a `POST` that creates something
204 No Content	Success, but no data returned	After a `DELETE` request
400 Bad Request	Malformed or invalid request	Missing field, wrong data type
401 Unauthorized	No/invalid authentication	Login required
403 Forbidden	Authenticated, but no permission	Logged in but not allowed
404 Not Found	Resource doesn't exist	Patient ID not in DB
500 Internal Server Error	Generic failure	Something broke on the server
502 Bad Gateway	Gateway (like Nginx) failed to reach backend	
503 Service Unavailable	Server is down or overloaded	

> `HTTPException` is a special built-in exception in FastAPI used to **return custom HTTP error responses** when something goes wrong in your API.

Instead of returning a normal JSON or crashing the server, you can **gracefully raise an error** with:

- a proper HTTP status code (like 404, 400, 403, etc.)
- a custom error message
- (optional) extra headers

Query Parameter

15 May 2025 18:17

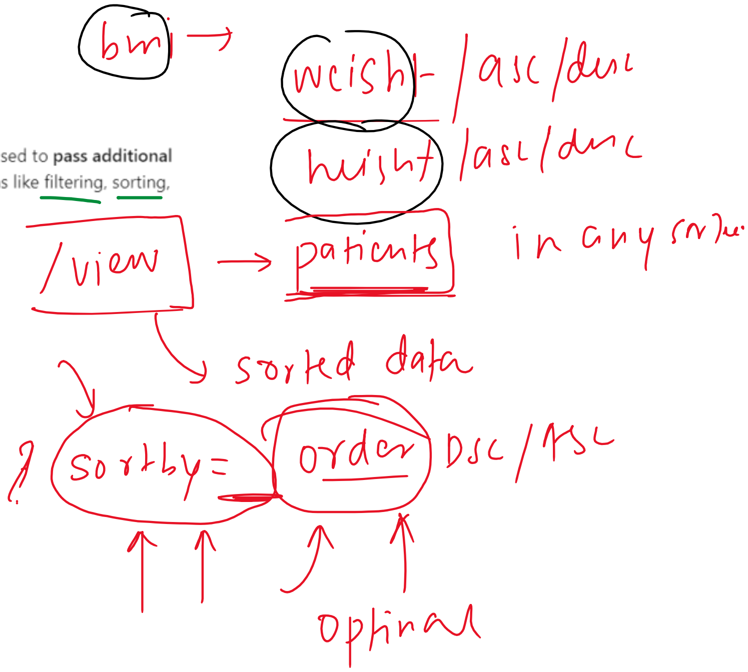
Query parameters are optional key-value pairs appended to the end of a URL, used to **pass additional data** to the server in an HTTP request. They are typically employed for operations like filtering, sorting, searching, and pagination, without altering the endpoint path itself.

```
/patients?city=Delhi&sort_by=age
```

- The `?` marks the start of query parameters.
- Each parameter is a key-value pair: `key=value`
- Multiple parameters are separated by `&`

In this case:

- `city=Delhi` is a query parameter for filtering
- `sort_by=age` is a query parameter for sorting

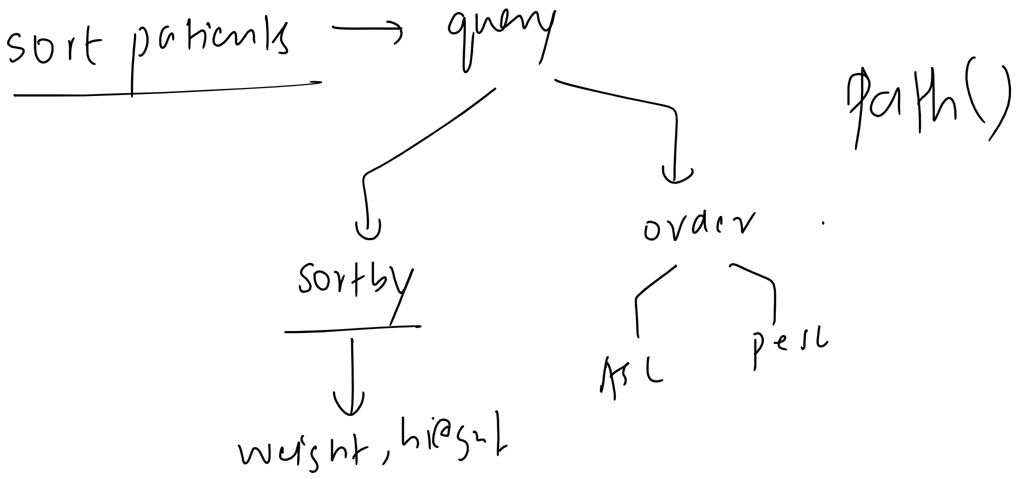


> `Query()` is a utility function provided by **FastAPI** to declare, validate, and document **query parameters** in your API endpoints.

It allows you to:

- Set **default values**
- Enforce **validation rules**
- Add **metadata** like description, title, examples

`default`	Set default value (e.g., `Query(0)`)
`title`	Displayed in API docs
`description`	Detailed explanation in Swagger
`example` / `examples`	Provide sample inputs
`min_length`, `max_length`	Validate string length
`ge`, `gt`, `le`, `lt`	Validate numeric bounds
`regex`	Pattern match for strings

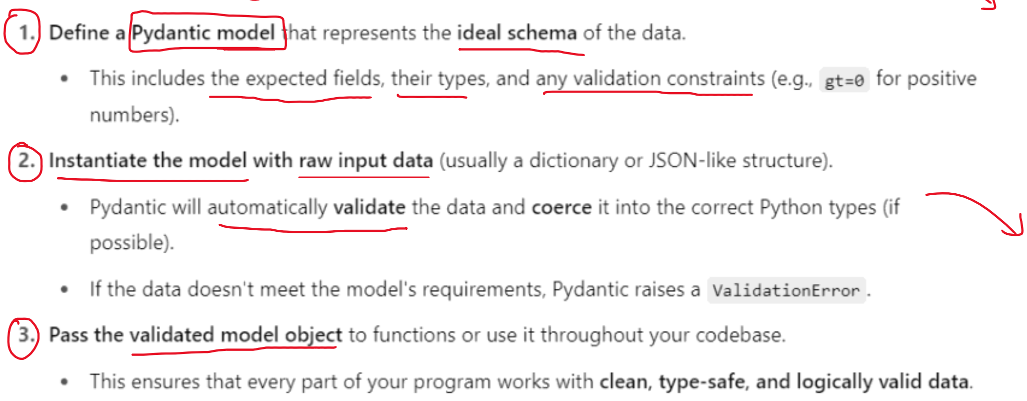


bwp

Pydantic

17 May 2025 18:37

class



1. **Define a Pydantic model** that represents the **ideal schema** of the data.
   - This includes the expected fields, their types, and any validation constraints (e.g., `gt=0` for positive numbers).
2. **Instantiate the model with raw input data** (usually a dictionary or JSON-like structure).
   - Pydantic will automatically **validate** the data and **coerce** it into the correct Python types (if possible).
   - If the data doesn't meet the model's requirements, Pydantic raises a `ValidationError`.
3. **Pass the validated model object** to functions or use it throughout your codebase.
   - This ensures that every part of your program works with **clean, type-safe, and logically valid data**.

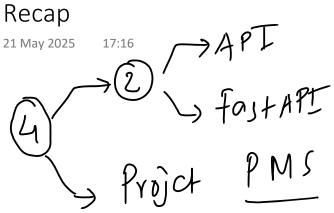
{ name → nitish
age - 30 }

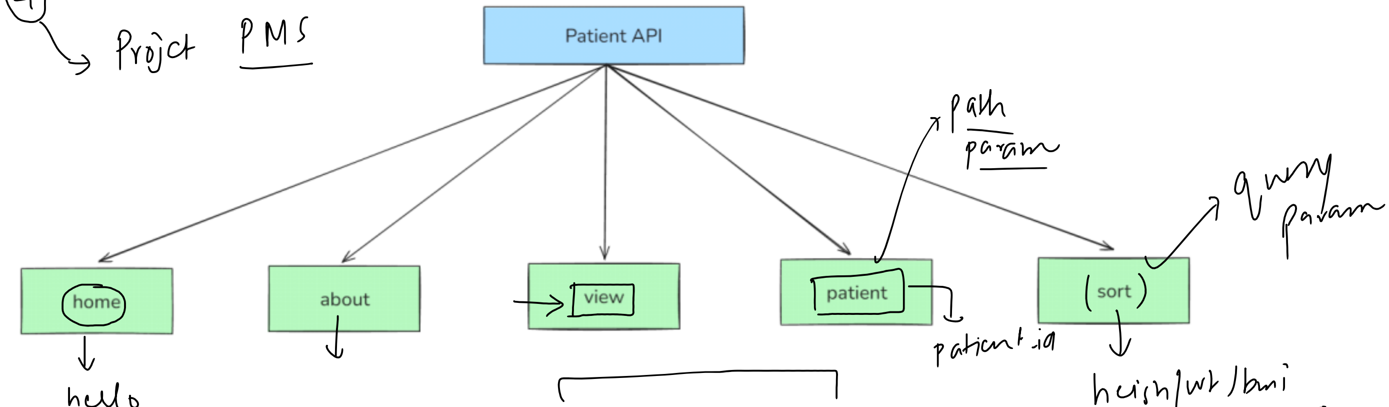
pydantic object

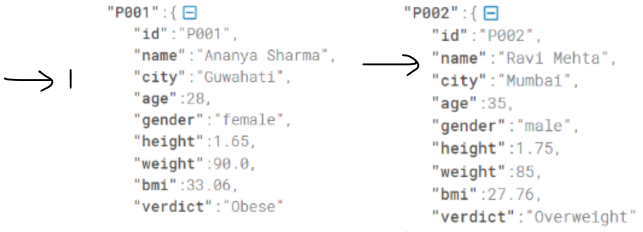
validated

Recap

21 May 2025 17:16



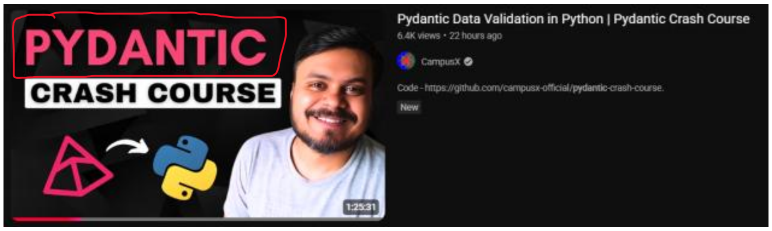




```
"P001":{ ⊟
   "id":"P001",
   "name":"Ananya Sharma",
   "city":"Guwahati",
   "age":28,
   "gender":"female",
   "height":1.65,
   "weight":90.0,
   "bmi":33.06,
   "verdict":"Obese"

"P002":{ ⊟
   "id":"P002",
   "name":"Ravi Mehta",
   "city":"Mumbai",
   "age":35,
   "gender":"male",
   "height":1.75,
   "weight":85,
   "bmi":27.76,
   "verdict":"Overweight"
```

Plan of action

21 May 2025 17:22



Pydantic Data Validation in Python | Pydantic Crash Course

6.4K views • 22 hours ago

CampusX

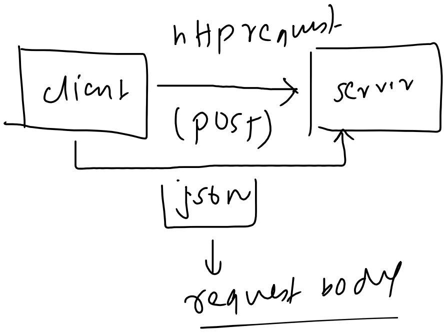
Code - https://github.com/campusx-official/pydantic-crash-course.

New

A request body is the portion of an HTTP request that contains data sent by the client to the server. It is typically used in HTTP methods such as POST or PUT to transmit structured data (e.g., JSON, XML, form-data) for the purpose of creating or updating resources on the server. The server parses the request body to extract the necessary information and perform the intended operation.

update

1)



GET

thirty → 30

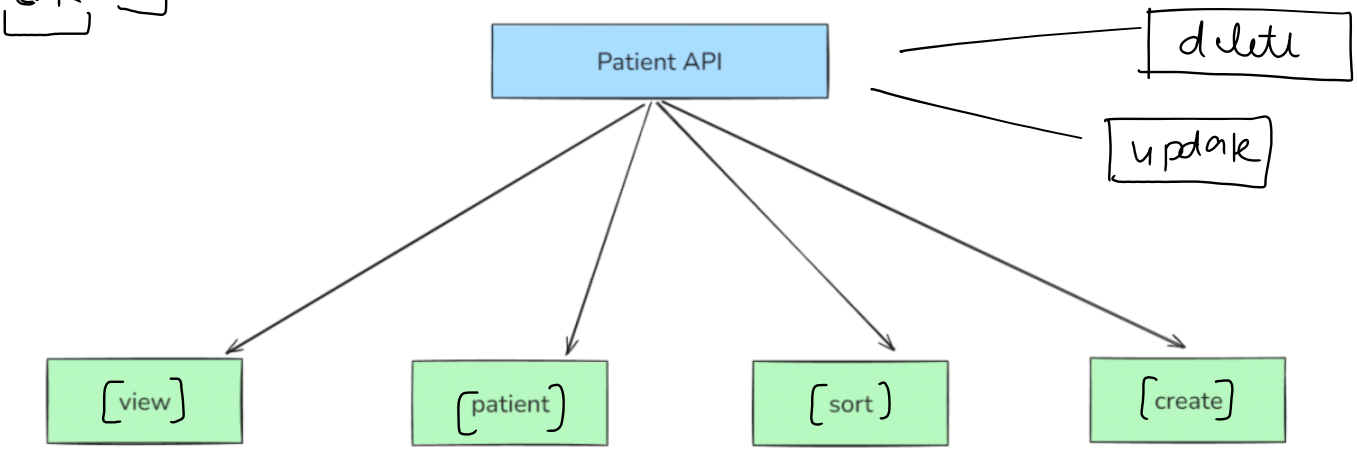
2) [validate] → pydantic model

3) json file → new record add

Project Progress

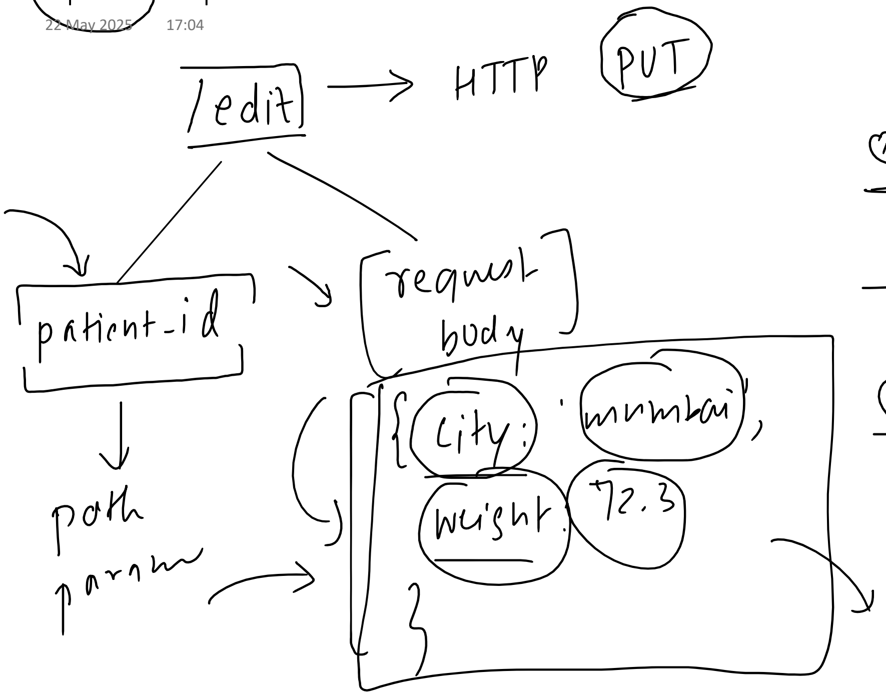
22 May 2025 15:59

CRUD



Update endpoint

22 May 2025 17:04



/edit → HTTP PUT

patient_id

path param

request body

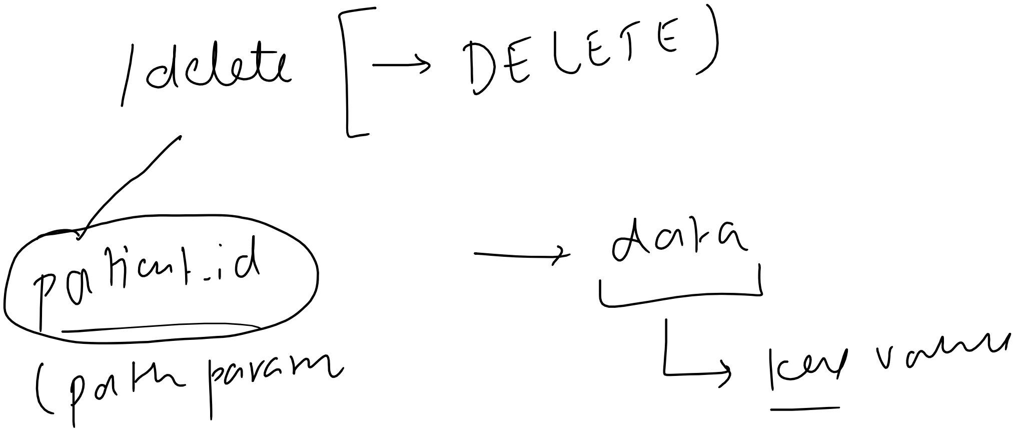
{ city: 'mumbai',
weight: 72.3
}

① → new pydantic model

→ Patient

② new data

existing → update



/delete [→ DELETE)

patient_id

(path param

→ data

→ key value

Plan of Attack

27 May 2025 16:30

6 → fastAPI funda → PMS → CRUD

PMS ↓ api

→ API

→ fastAPI.

→ HTTP methods

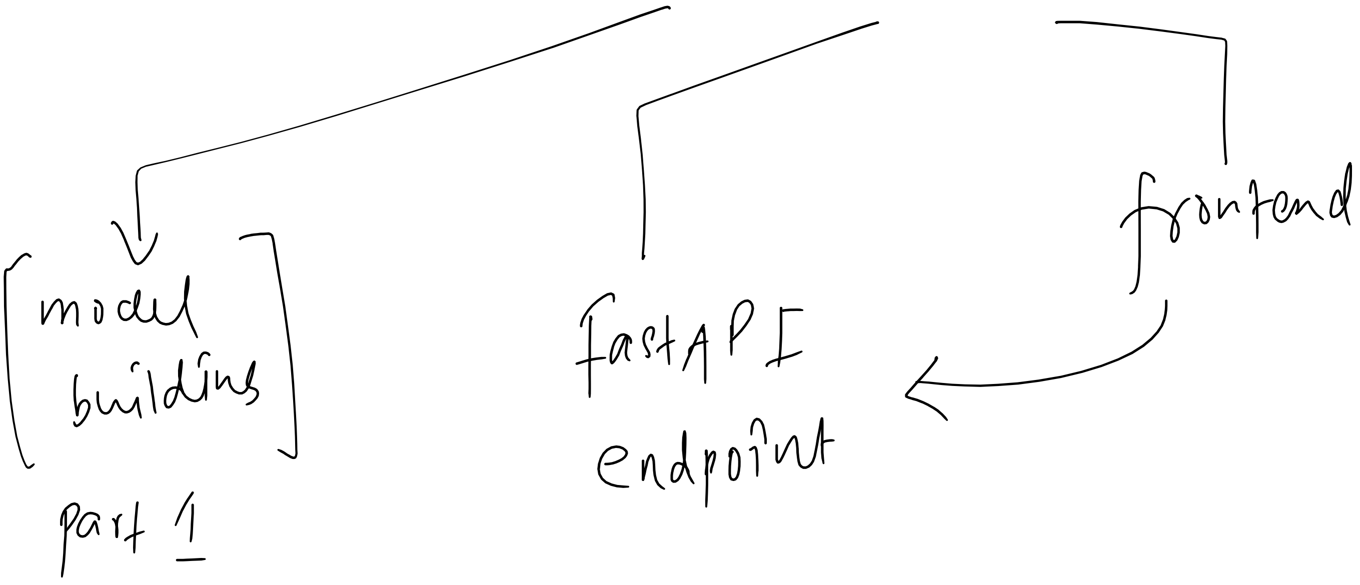
→ path | query

→ request

→ pydantic

→

ml model → fastapi → serve



model building

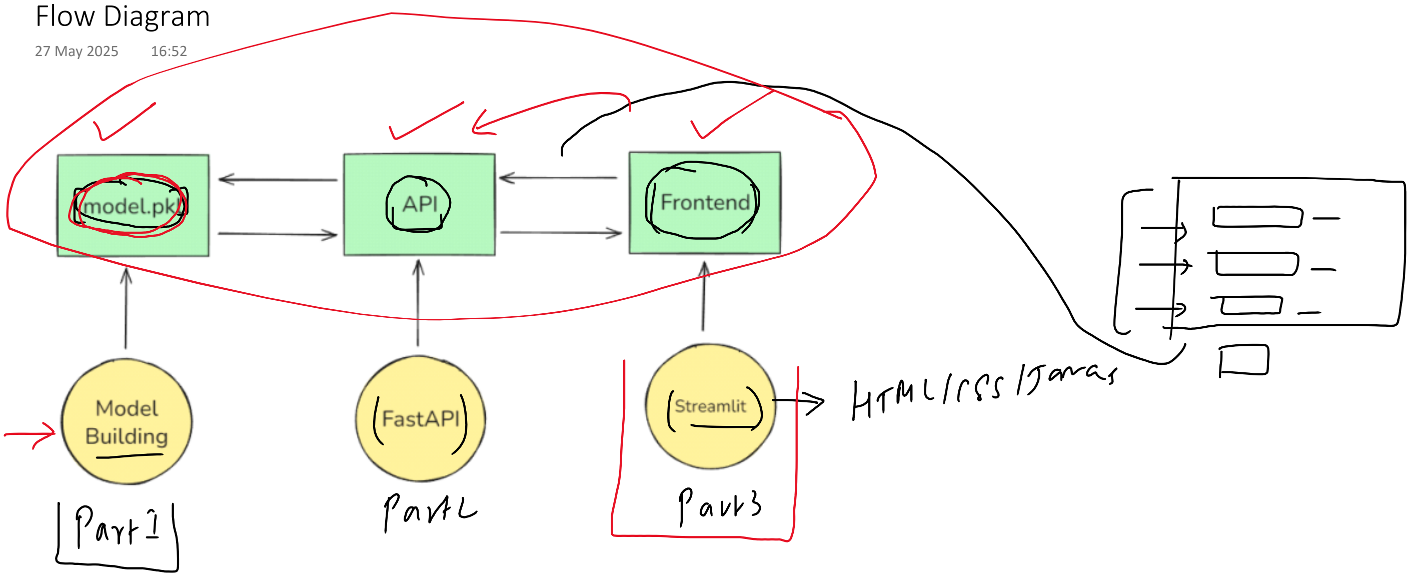
part 1

fastAPI endpoint

frontend

Flow Diagram

27 May 2025 16:52

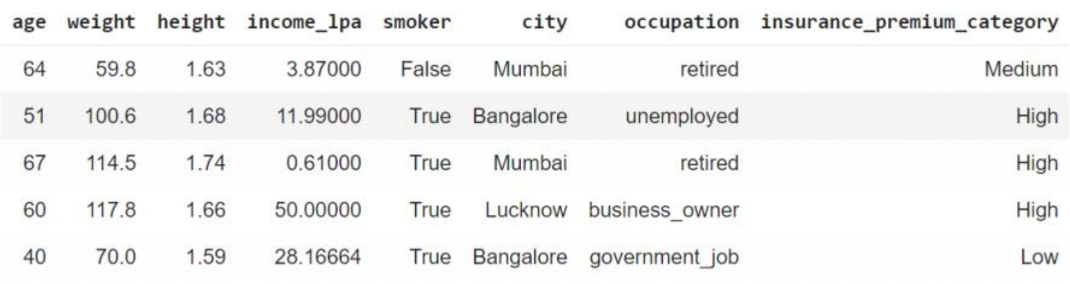


Step 1 - Building & Exporting the Model

28 May 2025 12:50

model

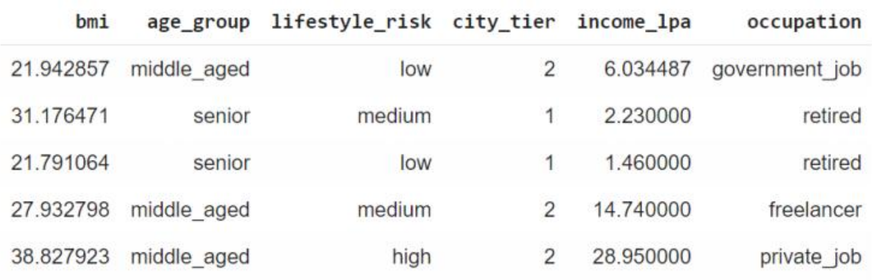
usecase → insurance premium



age	weight	height	income_lpa	smoker	city	occupation	insurance_premium_category
64	59.8	1.63	3.87000	False	Mumbai	retired	Medium
51	100.6	1.68	11.99000	True	Bangalore	unemployed	High
67	114.5	1.74	0.61000	True	Mumbai	retired	High
60	117.8	1.66	50.00000	True	Lucknow	business_owner	High
40	70.0	1.59	28.16664	True	Bangalore	government_job	Low

companies

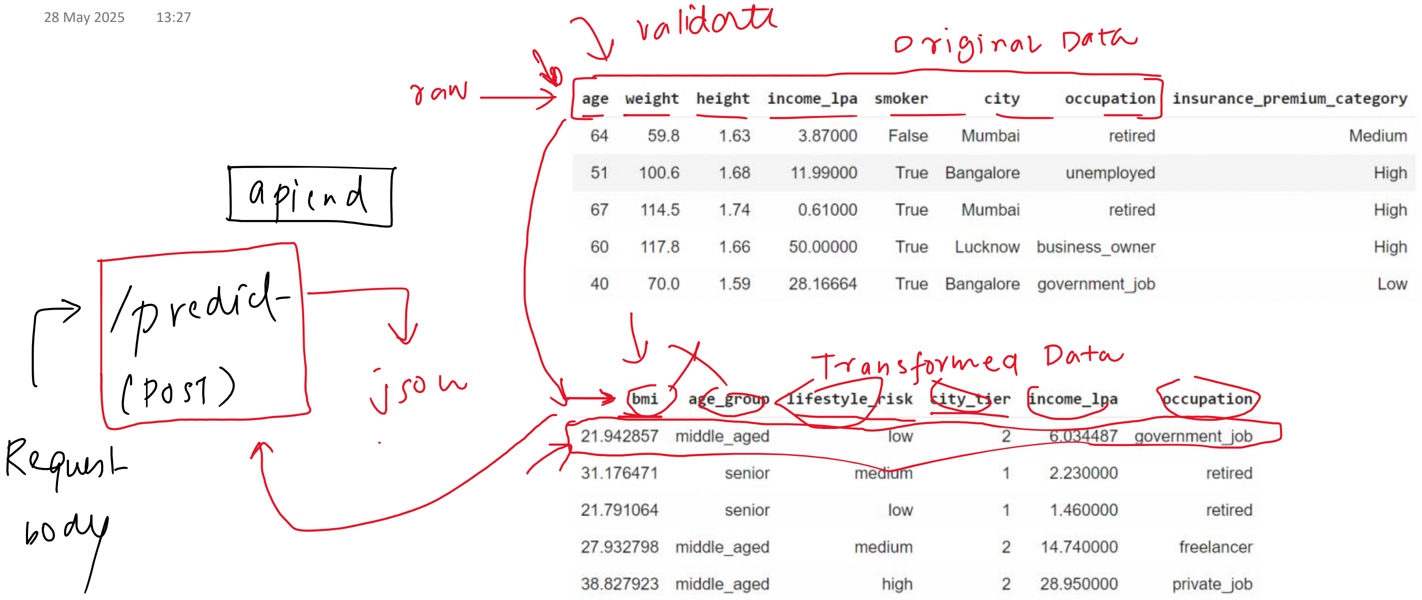
users



bmi	age_group	lifestyle_risk	city_tier	income_lpa	occupation
21.942857	middle_aged	low	2	6.034487	government_job
31.176471	senior	medium	1	2.230000	retired
21.791064	senior	low	1	1.460000	retired
27.932798	middle_aged	medium	2	14.740000	freelancer
38.827923	middle_aged	high	2	28.950000	private_job

Step 2 - Building the API Endpoint

28 May 2025 13:27

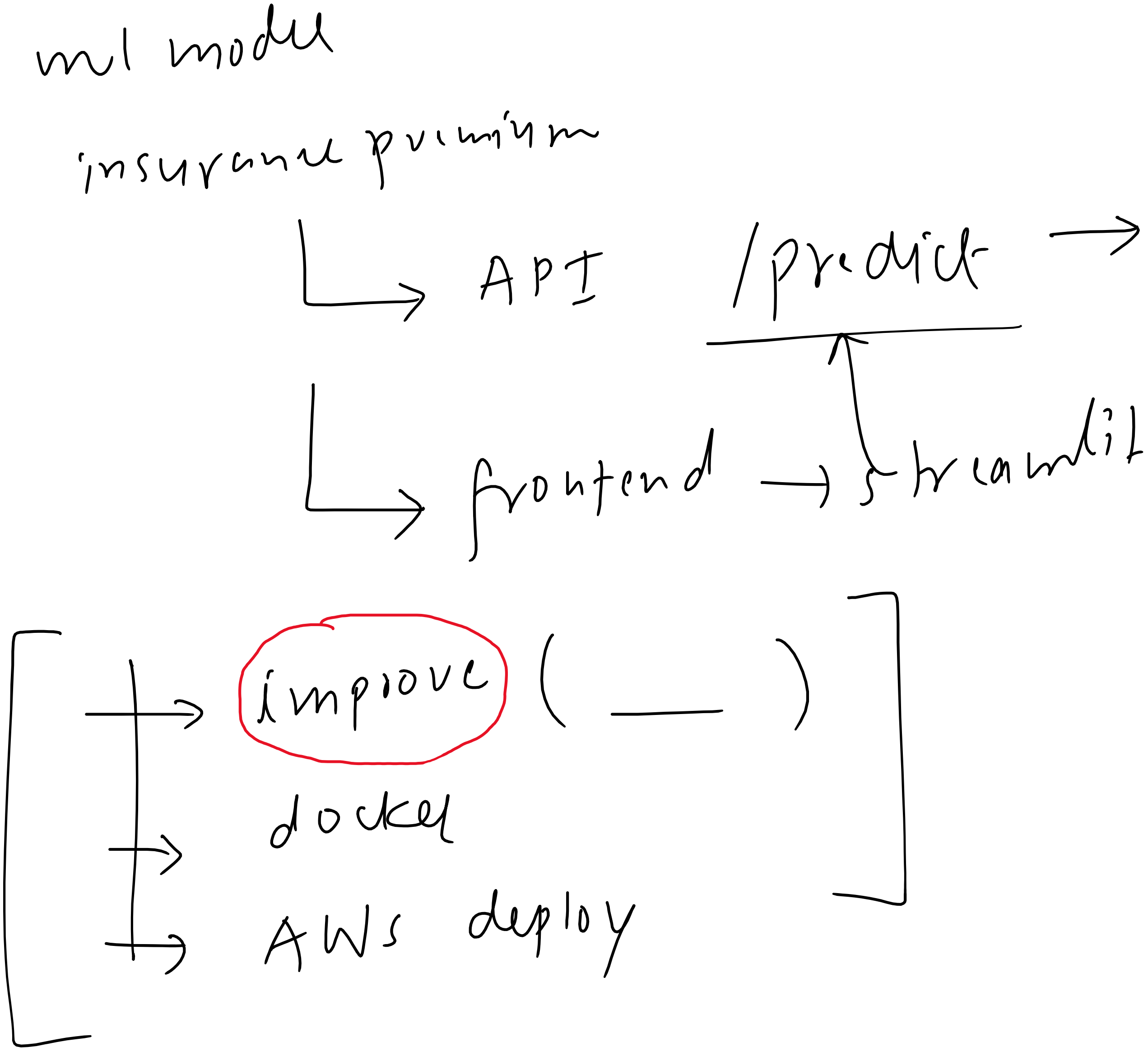


age	weight	height	income_lpa	smoker	city	occupation	insurance_premium_category
64	59.8	1.63	3.87000	False	Mumbai	retired	Medium
51	100.6	1.68	11.99000	True	Bangalore	unemployed	High
67	114.5	1.74	0.61000	True	Mumbai	retired	High
60	117.8	1.66	50.00000	True	Lucknow	business_owner	High
40	70.0	1.59	28.16664	True	Bangalore	government_job	Low

bmi	age_group	lifestyle_risk	city_tier	income_lpa	occupation
21.942857	middle_aged	low	2	6.034487	government_job
31.176471	senior	medium	1	2.230000	retired
21.791064	senior	low	1	1.460000	retired
27.932798	middle_aged	medium	2	14.740000	freelancer
38.827923	middle_aged	high	2	28.950000	private_job

Recap

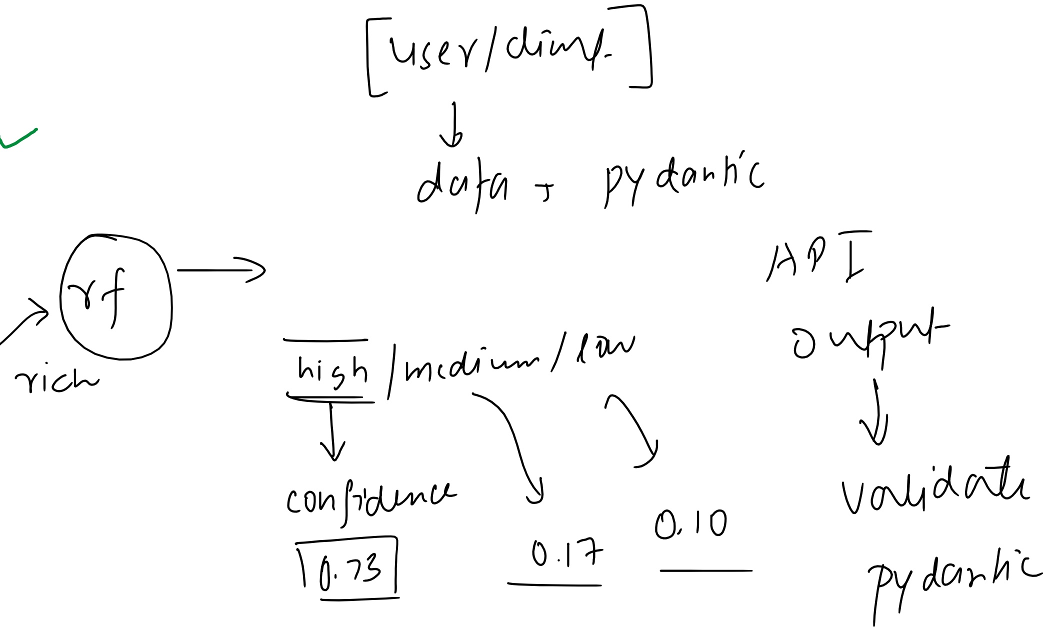
02 June 2025 17:24

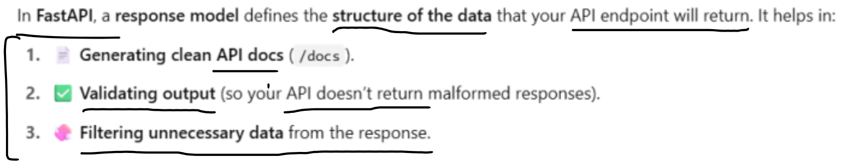


Improvements

02 June 2025 17:26

1. Create a new folder ✓
2. Field validator for city feature ✓
3. Add routes ✓
   a. Home
   b. Health check
4. Add model version ✓
5. Separation of logic
   a. Pydantic model ✓
   b. City tier ✓
   c. ML logic ✓
6. Try catch ✓
7. Add confidence score ✓
8. Response Model ✓



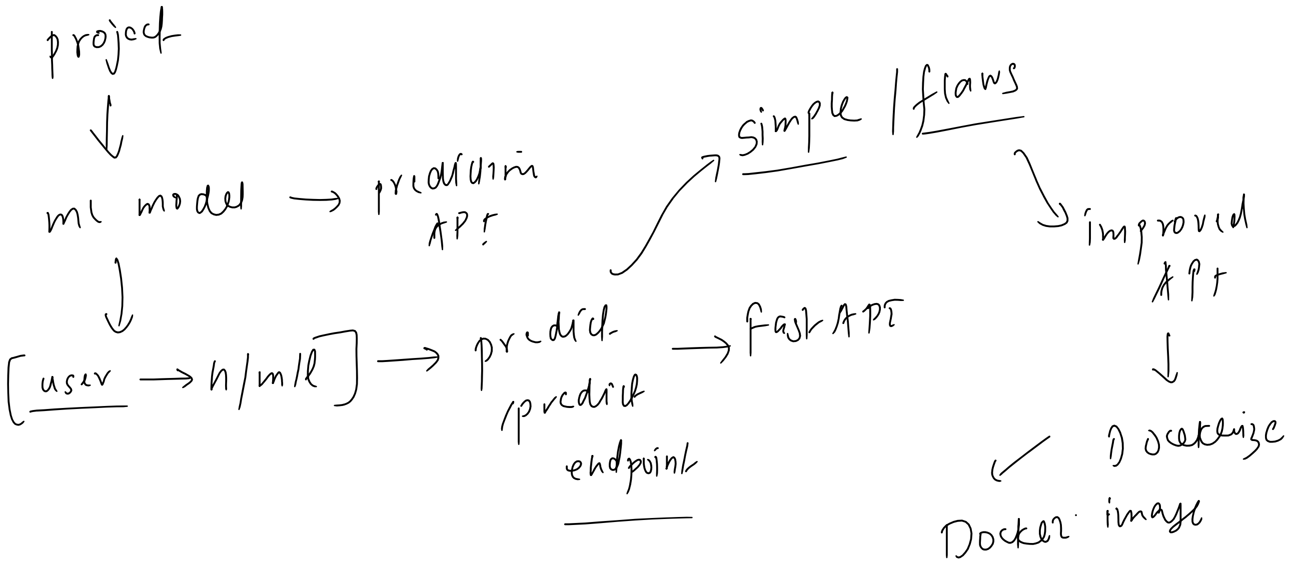


In **FastAPI**, a **response model** defines the **structure of the data** that your API endpoint will return. It helps in:

1. 📄 **Generating clean API docs** (`/docs`).
2. ✅ **Validating output** (so your API doesn't return malformed responses).
3. 🧽 **Filtering unnecessary data** from the response.

Recap

05 June 2025 18:12



project

ml model

prediction API

user → html

predict

/predict

endpoint

FastAPI

simple / flaws

improved API

Dockerize

Docker image

Steps to create a Docker Image

05 June 2025 18:52

Setup

1. Install Docker
2. Create account on Docker Hub

Step 1 - Create a Dockerfile

Step 2 - Build the docker image [docker build -t tweakster24/insurance-premium-api .]

Step 3 - Login to Docker Hub [docker login]

Step 4 - Push the image to Docker Hub [docker push tweakster24/insurance-premium-api]

Step 5 - Pull the docker image

Step 6 - Run the docker image locally [docker run -p 8000:8000 tweakster24/insurance-premium-api]

Recap

06 June 2025 16:52

ml model → Insurance premium pred

↓

API → /predict → FastAPI

→ Improve →

→ Docker → DockerHub

→ AWS → account

Steps for Deployment

06 June 2025 16:52

1. create an EC2 instance ✓
2. Connect to the EC2 instance ✓
3. Run the following commands ✓
    a. sudo apt-get update

    b. sudo apt-get install -y docker.io

    c. sudo systemctl start docker

    d. sudo systemctl enable docker

    e. sudo usermod -aG docker $USER

    f. exit
4. Restart a new connection to EC2 instance
5. Run the following commands ✓
    a. docker pull tweakster24/insurance-premium-api:latest

    b. docker run -p 8000:8000 tweakster24/insurance-premium-api
6. change security group settings ✓
7. Check the API ✓
8. Change the frontend code