

AI VIETNAM  
All-in-One Course  
(TA Session)

# Deploy Deep Learning models with FastAPI

## Extra Class

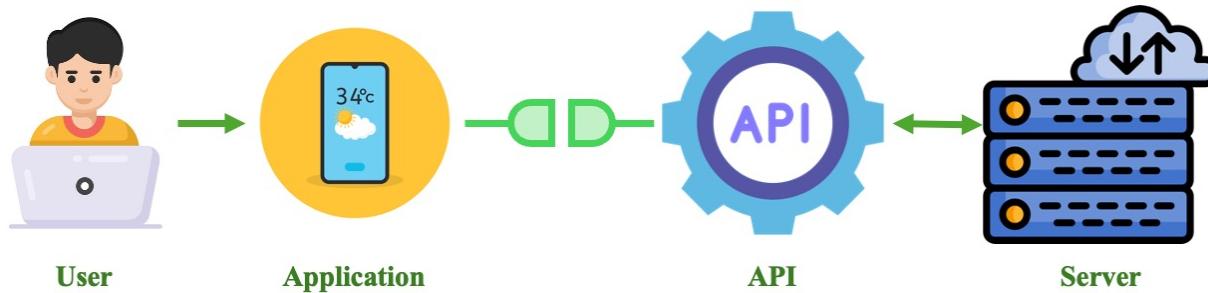


AI VIET NAM  
[@aivietnam.edu.vn](https://aivietnam.edu.vn)

Dinh-Thang Duong – TA  
Nguyen-Thuan Duong – STA

# Getting Started

## ❖ Objectives



### Our objectives:

- Discuss about the definition of API.
- Learn the basis of FastAPI.
- Learn how to deploy a Deep Learning model as an API service with FastAPI.



# Outline

- Introduction
- API
- FastAPI
- Model Deployment
- Question

# Introduction

# Introduction

## ❖ Getting Started

ChatGPT 4 ▾

You  
hi

ChatGPT  
Hello! How can I assist you today?

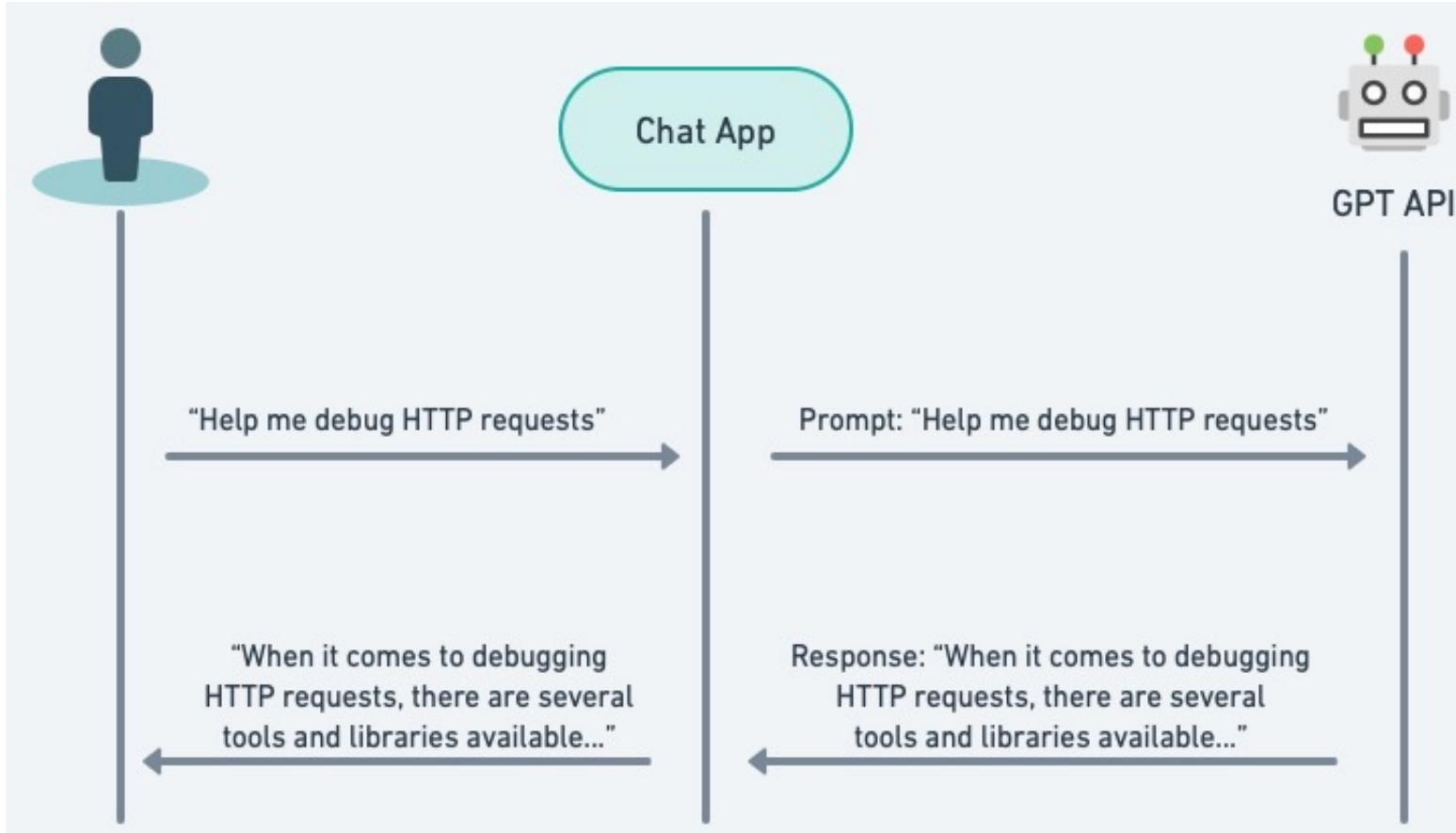
How do we actually interact with ChatGPT behind the scene?

Message ChatGPT... ↑

ChatGPT can make mistakes. Consider checking important information.

# Introduction

## ❖ Getting Started

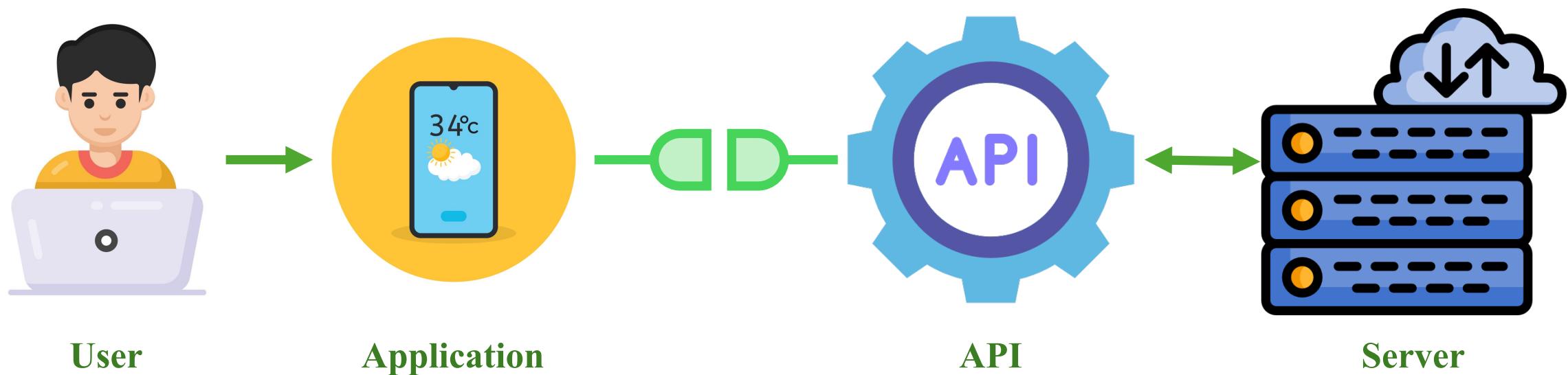


The ChatGPT website get the response from a something called **API**

# API

## ❖ Getting Started

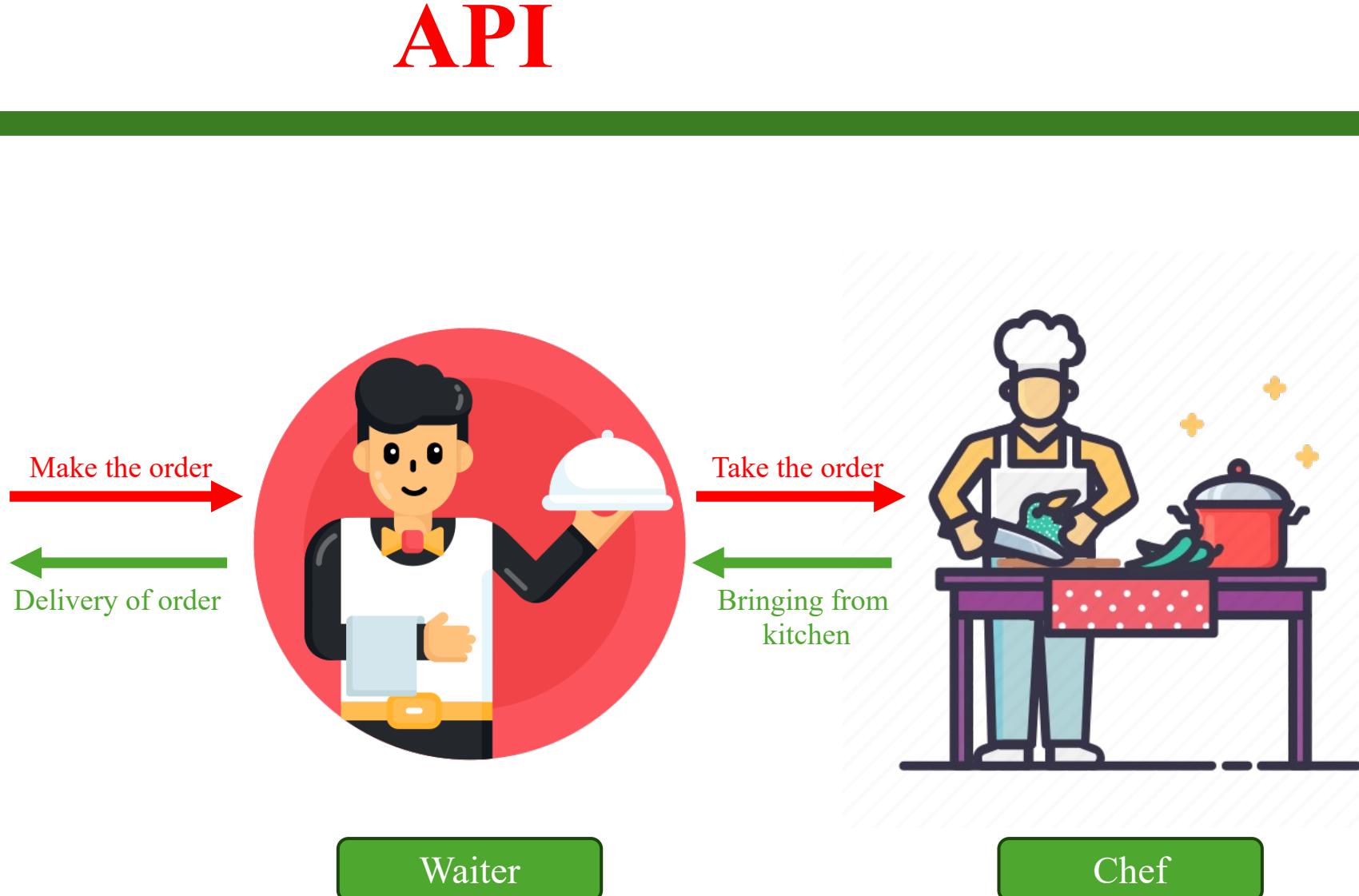
- **API (Application Programming Interface)** is a facilitator that enables apps, databases, softwares and IoT devices to communicate with each other.



## ❖ Getting Started



Customer

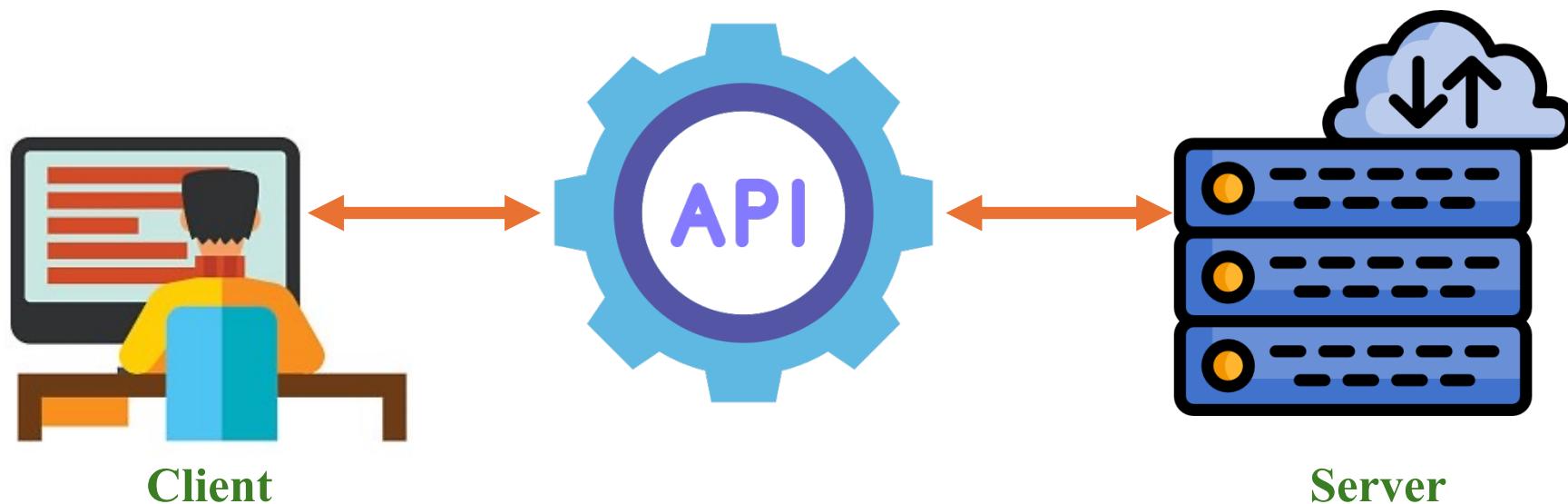


Waiter

Chef

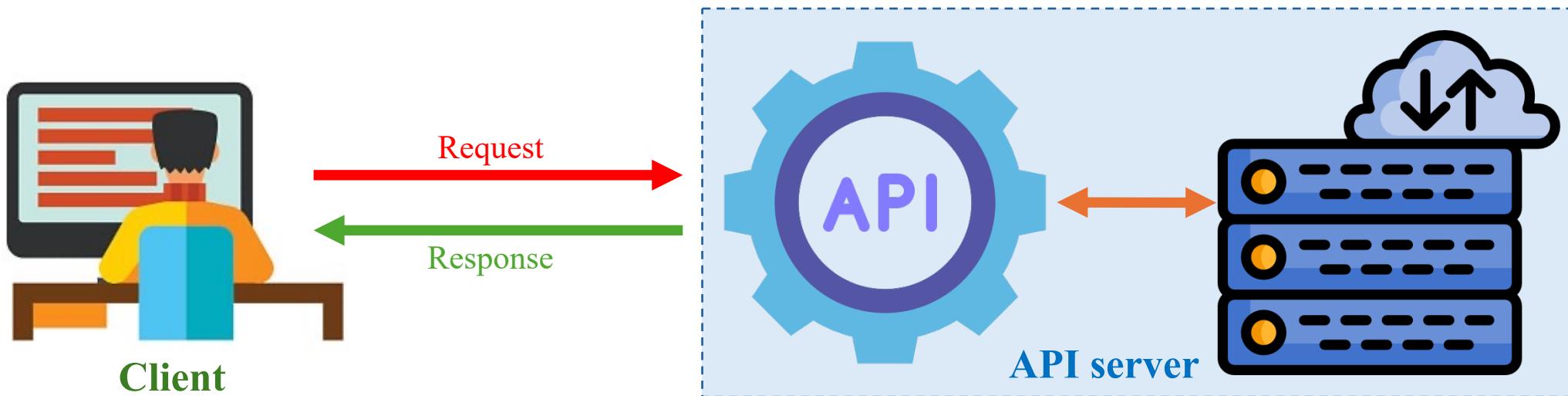
## ❖ Getting Started

- The application sending the request is called the client, and the application sending the response is called the server.



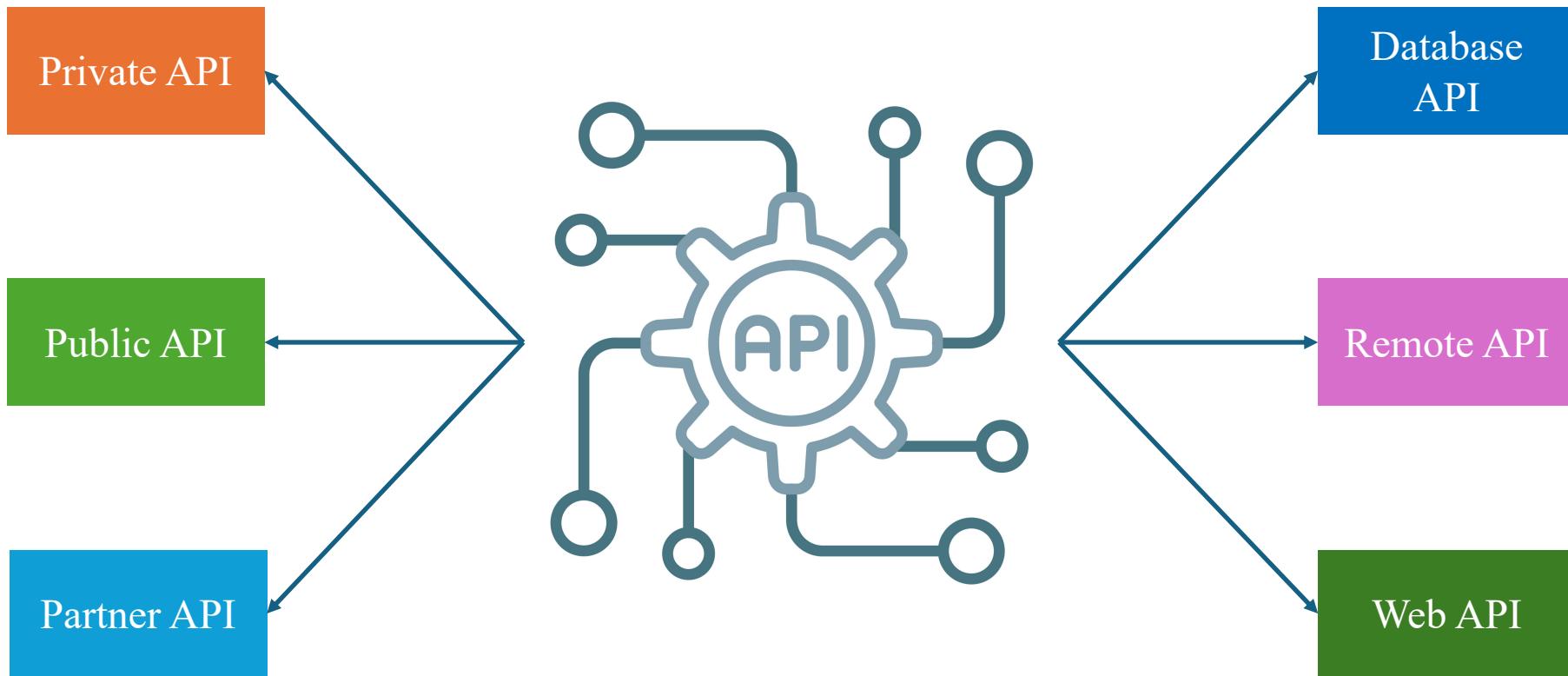
## ❖ How do APIs work ?

- A **client** sends a request to API server. The request is made using a specific protocol (such as HTTP) and includes information about the operation the client wants to perform, e.g., retrieving data or updating a resource.



- The **API server** receives the request and processes it. The API server sends a response back to the client, which may include data, an error message, or a status code indicating the result of the operation.
- The **client** receives the response and processes it.

## ❖ Types of APIs



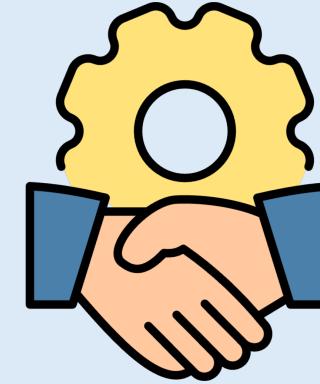
# API

## ❖ Private / Public / Partner API



APIs are open to **any developer**

Apps are more targeted towards end consumers



APIs are open to select **business partners**

Apps could be targeted at end consumers or business users

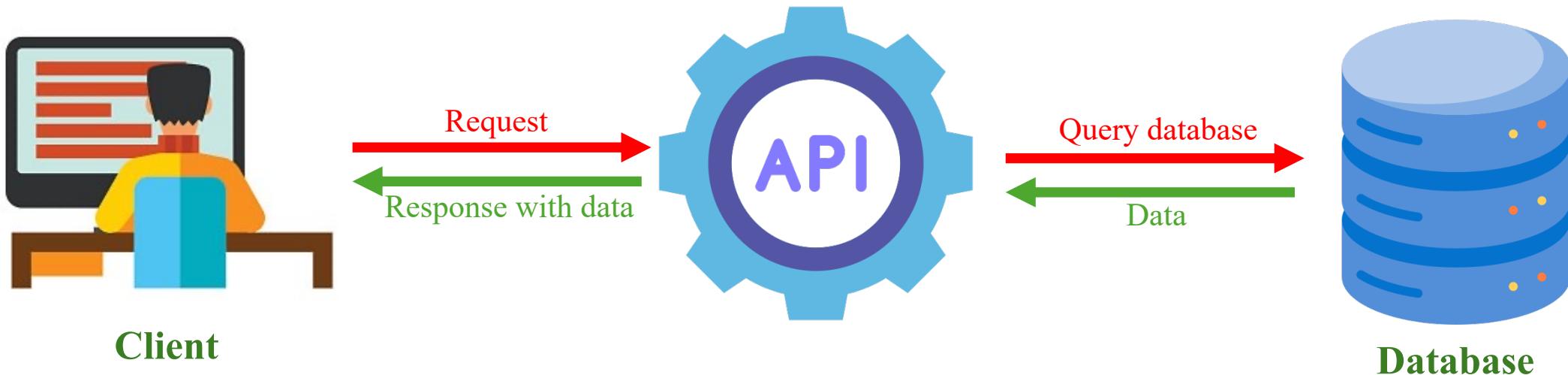


APIs are exposed only to **existing developers within the enterprise**

App are usually targeted at employees of the enterprise

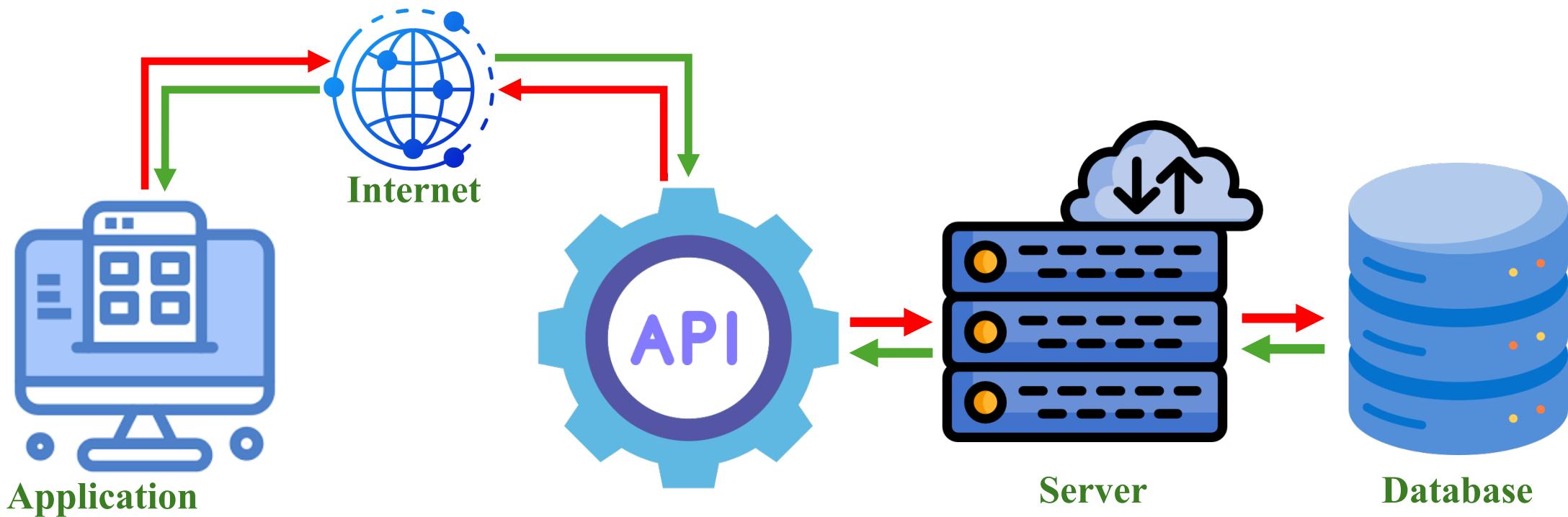
## ❖ Database API

- A database API allows applications to interact with a database to access and manipulate data.



## ❖ Remote API

- Using a remote API, two distant applications communicate across a communications network, primarily the internet.



## ❖ Web API

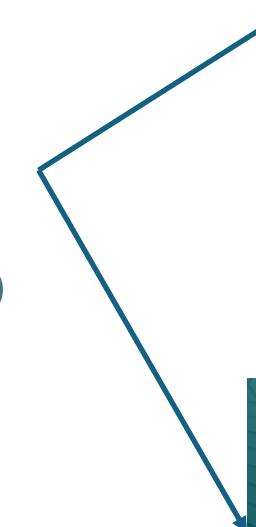
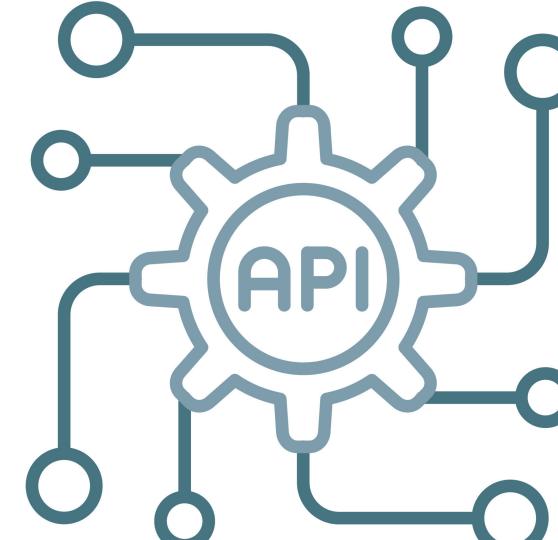
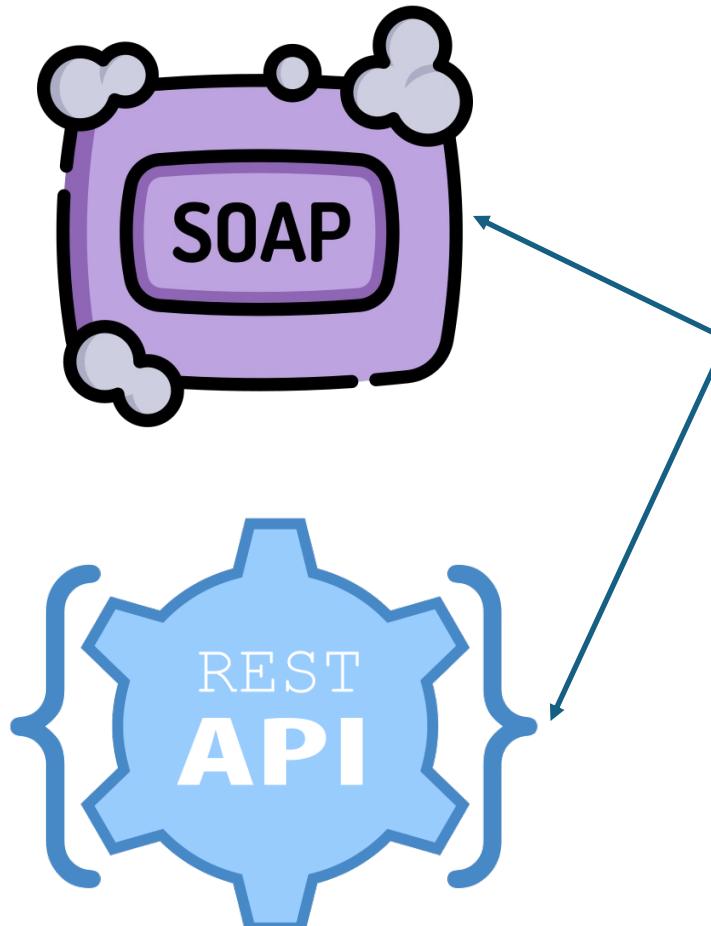
- Web API (or Web Service API) is an application interface between a web browser and a web server.



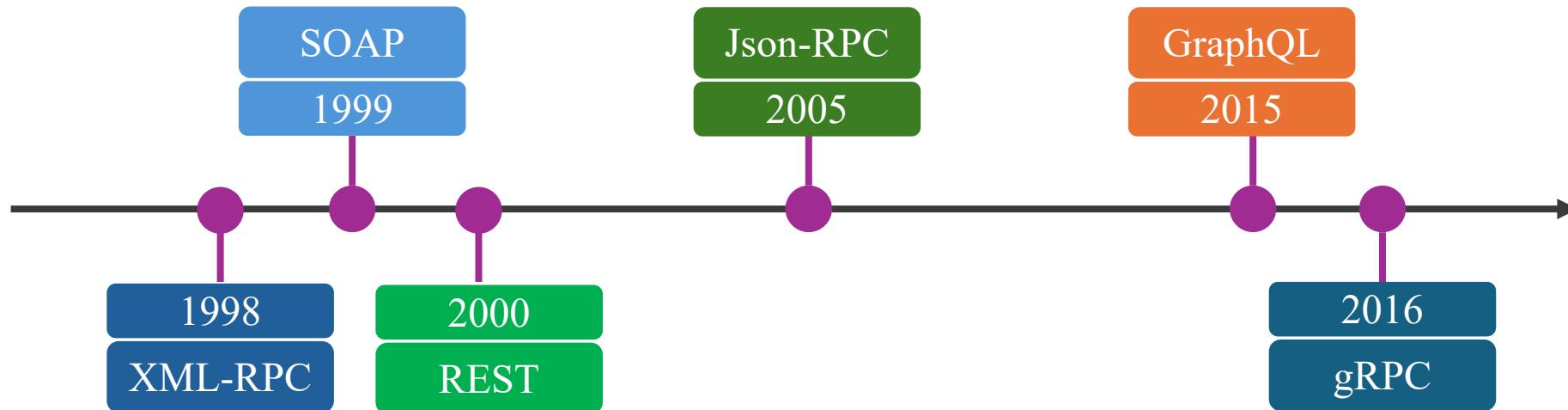
- All web services are APIs but not all APIs are web services.
- Web services require an internet connection to work, but APIs do not.

# API

## ❖ API protocol types



## ❖ Timeline



## ❖ Conclusion

	REST	GraphQL	SOAP	RPC
Structure	Follows six architectural constraints	Schema and type	Message structure	Local procedural calls
Format	Json, XML, HTML, plain text	Json	XML	Json, XML, Flatbuffers, etc
Advantages	Flexible terms of data format and structure	Solves over-fetching and under-fetching	Highly secure and extensible	Lightweight payloads make it high performing
Use cases	Resources based apps	Mobile APIs	Payment gateways	Command-focused systems

# FastAPI

# FastAPI

## ❖ Introduction

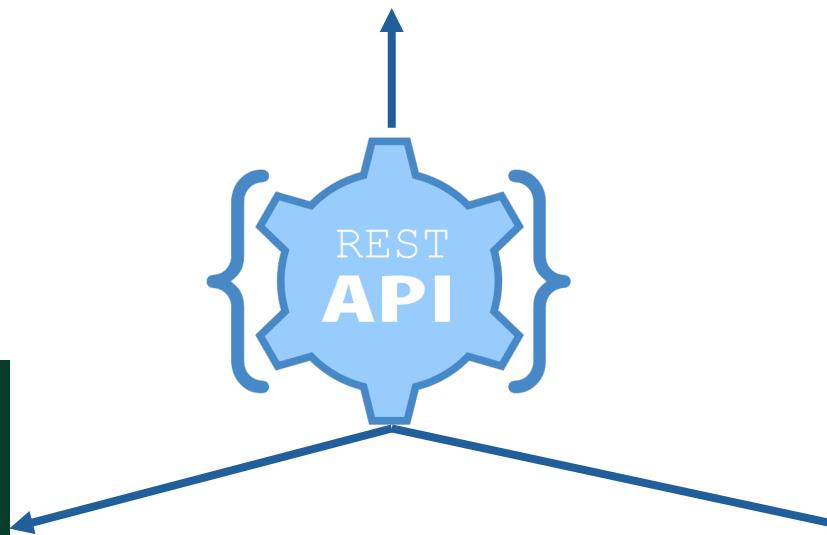


# FastAPI

**FastAPI** is a modern, fast (high-performance), web framework for building APIs with Python 3.8+ based on standard Python type hints.

# FastAPI

❖ Python Web Framework



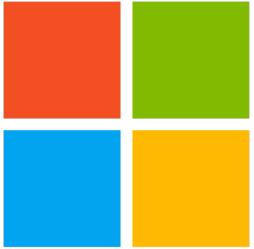
# FastAPI

## ❖ Django vs Flask vs FastAPI

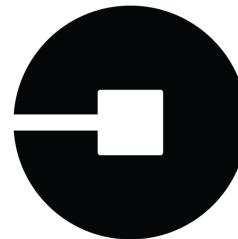
	<b>django</b>	<b>Flask</b>	<b>FastAPI</b>
<b>Performance speed</b>	Normal	Faster than Django	The fastest out there
<b>Async support</b>	<b>YES</b> with restricted latency	<b>NO</b> needs Asyncio	<b>YES</b> native async support
<b>Packages</b>	<b>Plenty</b> for robust web apps	<b>Less than Django</b> for minimalistic apps	<b>The least of all</b> for building web apps faster
<b>Popularity</b>	The most popular	The second popular	Relatively new
<b>Learning</b>	Hard to learn	Easier to learn	The easiest to learn

# FastAPI

❖ Which companies use FastAPI ?



Microsoft



UBER

# FastAPI

## ❖ The key features of FastAPI

High performance

On par with NodeJS and GO

Fast to code

Increase the speed to develop features by about 200% to 300%

Easy

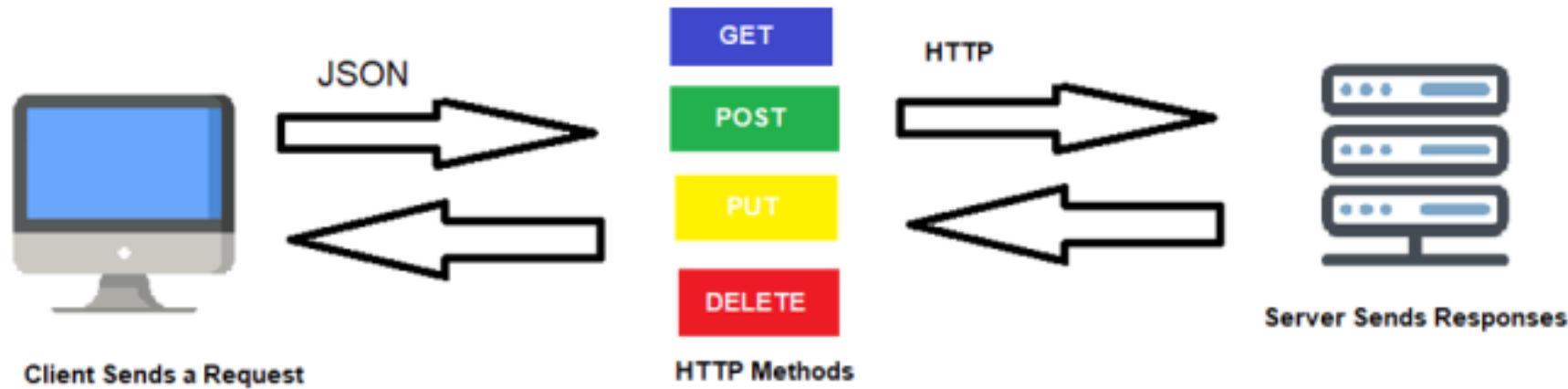
Easy to use and learn. Less time reading docs

Robust

Get production-ready code. With automatic interactive documentation

# FastAPI

## ❖ Practice



Learn how to use FastAPI to deploy an API service

# FastAPI

## ❖ Practice: Initialize first API



```
1 from fastapi import FastAPI  
2  
3 app = FastAPI()
```

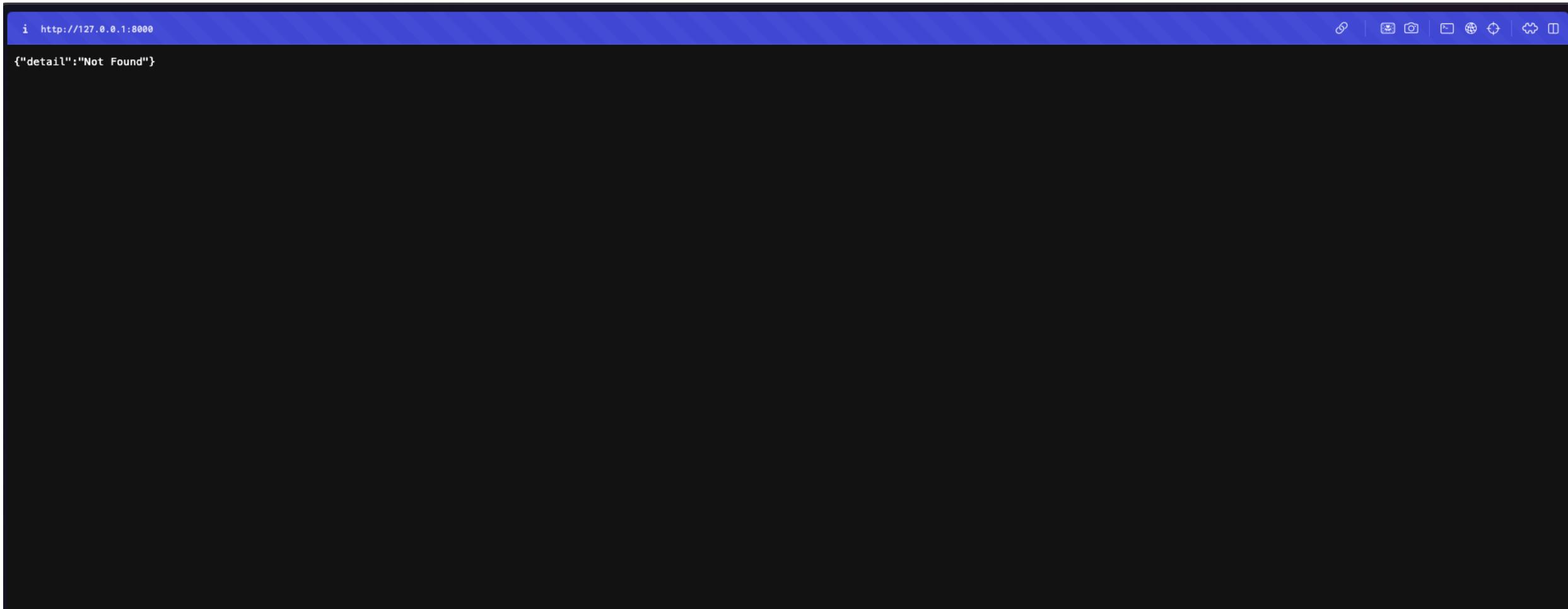
1. Create a FastAPI instance

```
○ (project_api_env) thangduong@Duongs-MacBook-Pro 1_Initialization % uvicorn main:app  
INFO:     Started server process [11216]  
INFO:     Waiting for application startup.  
INFO:     Application startup complete.  
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)  
□
```

2. Run uvicorn to deploy the API

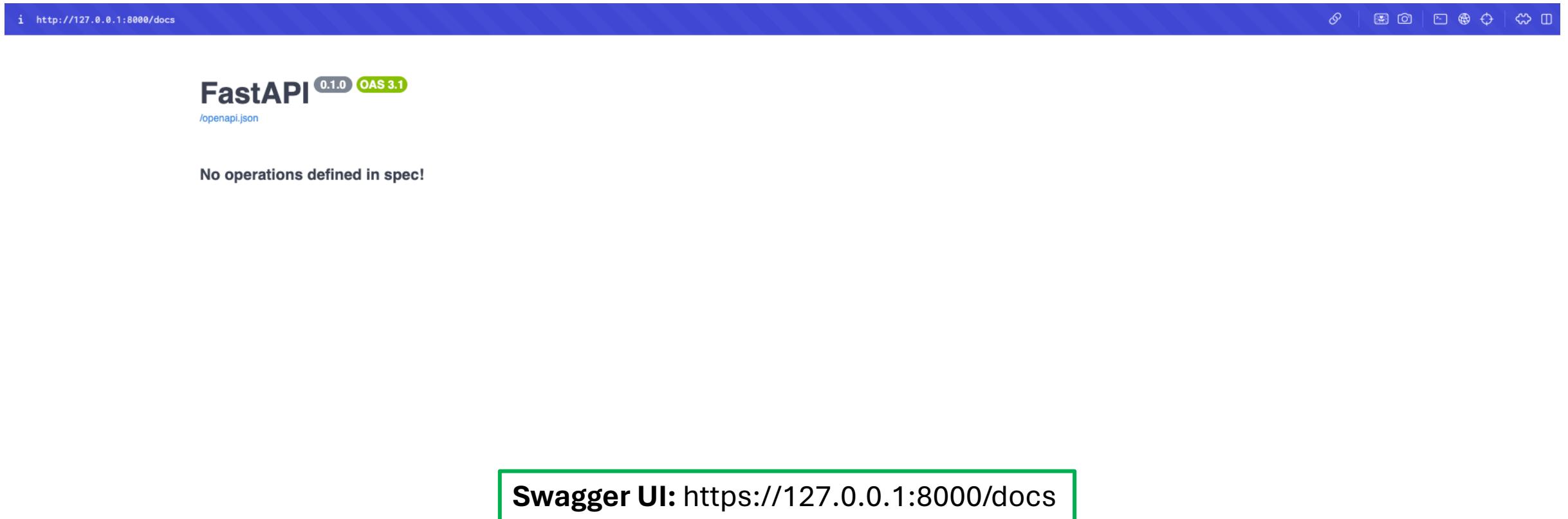
# FastAPI

## ❖ Practice: Initialize first API



# FastAPI

## ❖ Practice: Initialize first API



The screenshot shows a browser window with the URL `http://127.0.0.1:8000/docs` in the address bar. The page title is "FastAPI 0.1.0 OAS 3.1". Below the title, there is a link to `/openapi.json`. The main content area displays the message "No operations defined in spec!".

Swagger UI: <https://127.0.0.1:8000/docs>

# FastAPI

## ❖ Practice: Initialize first API

Schemes HTTP Authorize

**pet** Everything about your Pets

POST /pet Add a new pet to the store

PUT /pet Update an existing pet

GET /pet/findByStatus Finds Pets by status

GET /pet/findByTags Finds Pets by tags

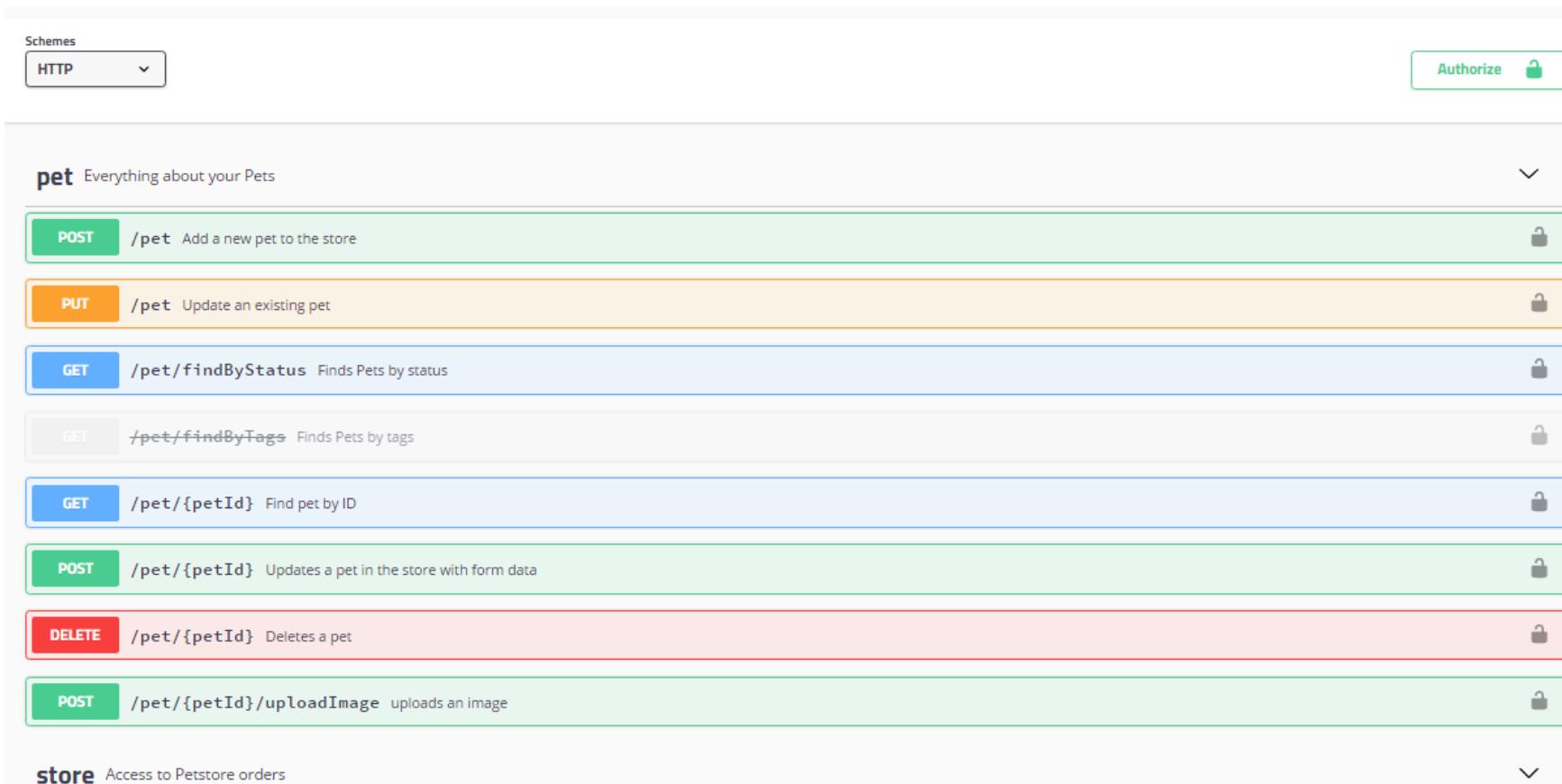
GET /pet/{petId} Find pet by ID

POST /pet/{petId} Updates a pet in the store with form data

DELETE /pet/{petId} Deletes a pet

POST /pet/{petId}/uploadImage uploads an image

**store** Access to Petstore orders



**Swagger UI:** An interface to visualize and interact with API. Can be accessed through suffix /docs.

# FastAPI

## ❖ Practice: Initialize first API

Schemes HTTP Authorize

**pet** Everything about your Pets

POST /pet Add a new pet to the store

PUT /pet Update an existing pet

GET /pet/findByStatus Finds Pets by status

GET /pet/findByTags Finds Pets by tags

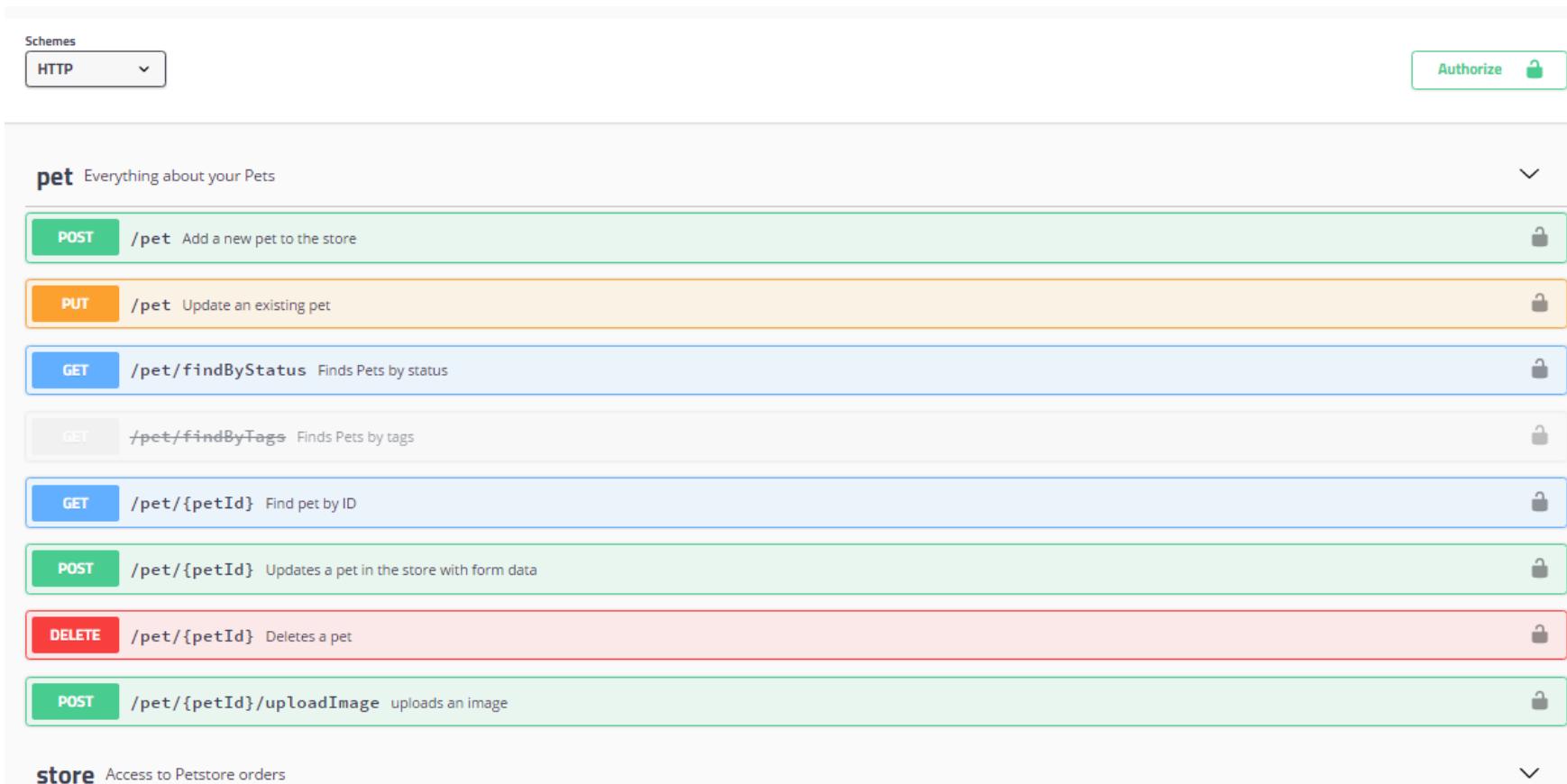
GET /pet/{petId} Find pet by ID

POST /pet/{petId} Updates a pet in the store with form data

DELETE /pet/{petId} Deletes a pet

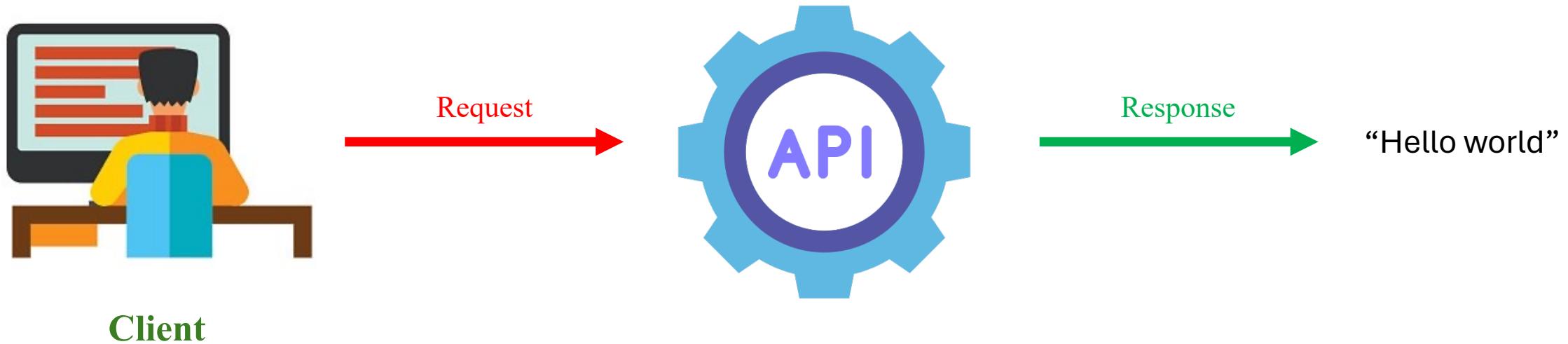
POST /pet/{petId}/uploadImage uploads an image

**store** Access to Petstore orders



**Swagger UI:** An interface to visualize and interact with API. Can be accessed through suffix /docs.

## ❖ Practice: Path Operations



Make API to return “Hello world” when we access to it

## ❖ Practice: Path Operations GET Method



```
1 from fastapi import FastAPI
2
3 app = FastAPI()
4
5 @app.get('/')
6 async def root():
7     return 'Hello world'
```

Defining a route

Define a function with decorator “`@app.get('/')`”:

- The `root()` function is **the handler** for requests to the root URL path ‘/’ using HTTP GET method.

# FastAPI

## ❖ Practice: Path Operations GET Method

```
i  http://127.0.0.1:8000
"Hello world"
```

FastAPI 0.1.0 OAS 3.1 /openapi.json

default

GET / Root

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/
```

Server response

Code	Details
200	Response body "Hello world"

Download

Response headers

```
content-length: 13
content-type: application/json
date: Sat, 30 Mar 2024 09:15:10 GMT
server: uvicorn
```

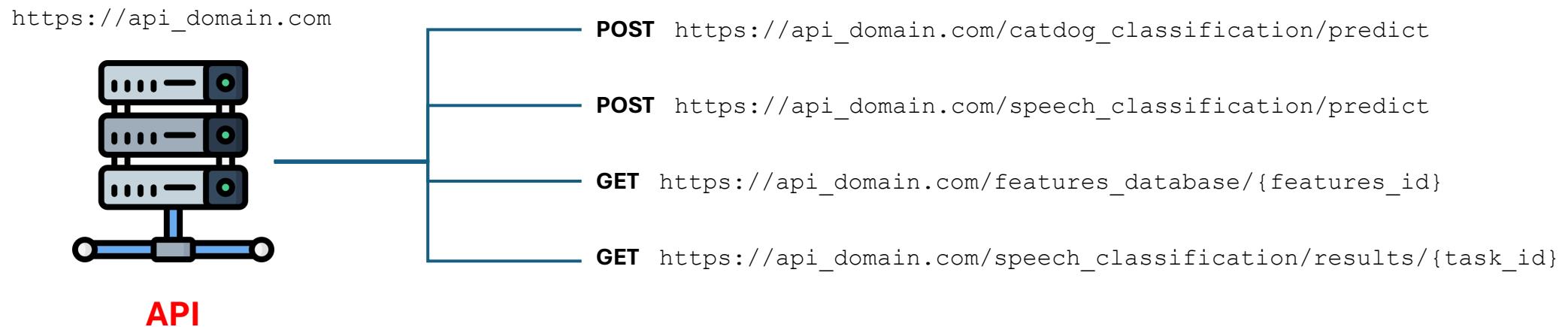
Responses

## ❖ Practice: Path Operations

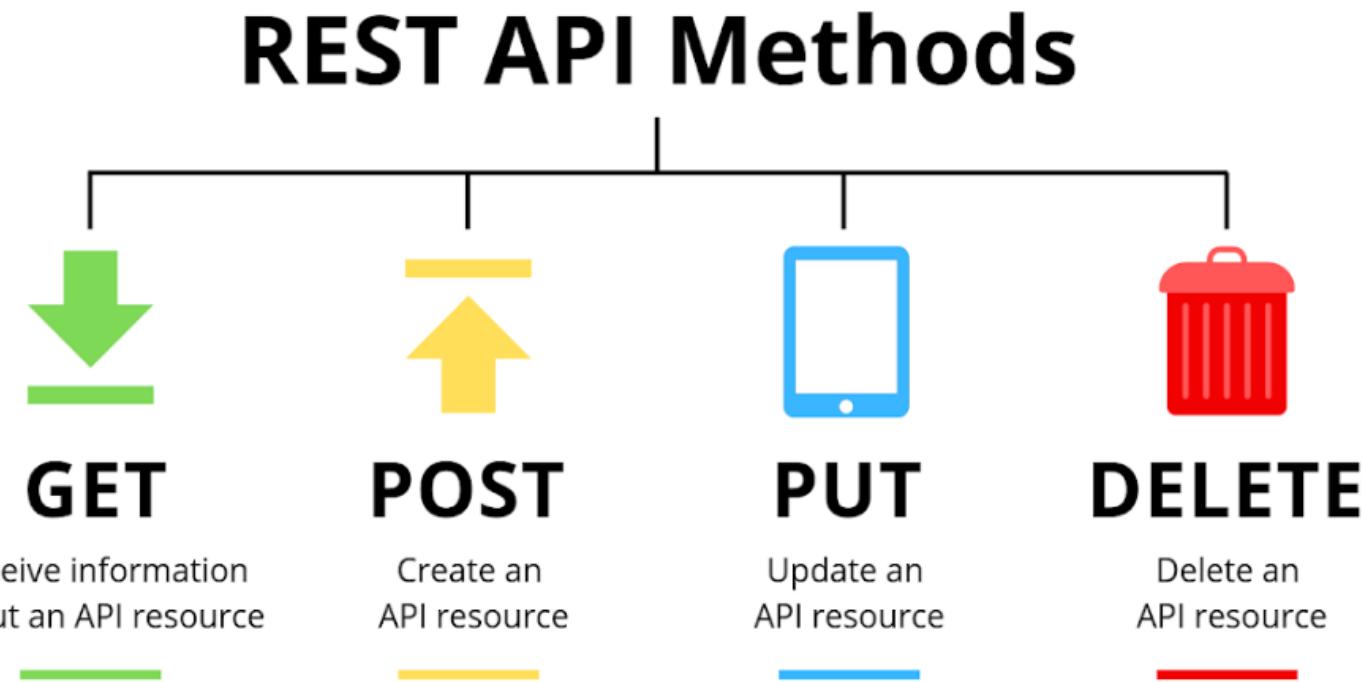
users		
GET	/api/user/	List All Users
POST	/api/user/	Create User
GET	/api/user/{name}	User Info By Name
PATCH	/api/user/{name}	Update User
DELETE	/api/user/{user_id}	Delete User
items		
GET	/api/items/	List All Items
POST	/api/items/	Create Item
GET	/api/items/{user_id}	List All Items By User Id
GET	/api/items/filter	Filter Items By Name And Category
GET	/api/items/{name}	Item Info By Name
PATCH	/api/items/{name}	Update Item
DELETE	/api/items/{item_id}	Delete Item
security		
POST	/api/login/	Login For Access Token

An API service may have several operations with difference purposes. To separate those, we define them as API Endpoints (routes, paths).

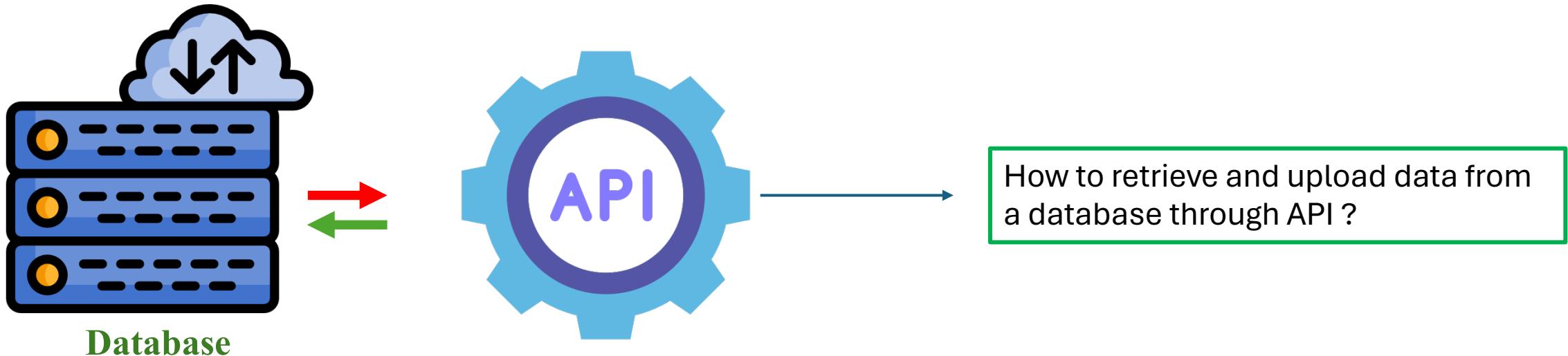
## ❖ Practice: Path Operations



## ❖ Practice: Path Operations



## ❖ Practice: Path Operations



## ❖ Practice: Path Operations GET Method



```
1 from fastapi import FastAPI
2
3 db = [
4     {
5         'student_id': '20240001',
6         'student_name': 'A'
7     },
8     {
9         'student_id': '20240002',
10        'student_name': 'B'
11    }
12 ]
13
14 app = FastAPI()
```



```
1 # Declare a GET method
2 @app.get('/')
3 def root():
4     return {
5         'message': 'Hello AIVN'
6     }
7
8 @app.get('/predict')
9 def predict():
10    return {
11        'predict_id': 0
12    }
13
14 @app.get('/student_db/{student_id}')
15 def get_student(student_id: str):
16     for record in db:
17         if record['student_id'] == student_id:
18             return record
```

## ❖ Practice: Path Operations (Path Parameters)

```
1 # Declare a GET method
2 @app.get('/')
3 def root():
4     return {
5         'message': 'Hello AIVN'
6     }
7
8 @app.get('/predict')
9 def predict():
10    return {
11        'predict_id': 0
12    }
13
14 @app.get('/student_db/{student_id}')
15 def get_student(student_id: str):
16    for record in db:
17        if record['student_id'] == student_id:
18            return record
```

```
1 from fastapi import FastAPI
2
3 app = FastAPI()
4
5 @app.get("/greet/{name}")
6 async def greet(name: str):
7     return {"message": f"Hello, {name}!"}
```

**Path Parameter:** A part of URL path that is expected to be a variable and is used to capture a value directly from the path.

## ❖ Practice: Path Operations POST Method



```
1 from fastapi import FastAPI
2
3 db = [
4     {
5         'student_id': '20240001',
6         'student_name': 'A'
7     },
8     {
9         'student_id': '20240002',
10        'student_name': 'B'
11    }
12 ]
13
14 app = FastAPI()
```



```
1 @app.get('/student_db/{student_id}')
2 def get_student(student_id: str):
3     for record in db:
4         if record['student_id'] == student_id:
5             return record
6
7
8 @app.post('/student_db')
9 def create_student(student_id: str, student_name: str):
10    record = {
11        'student_id': student_id,
12        'student_name': student_name
13    }
14
15    db.append(record)
```

## ❖ Practice: Path Operations PUT Method



```
1 from fastapi import FastAPI
2
3 db = [
4     {
5         'student_id': '20240001',
6         'student_name': 'A'
7     },
8     {
9         'student_id': '20240002',
10        'student_name': 'B'
11    }
12 ]
13
14 app = FastAPI()
```



```
1 @app.get('/student_db/{student_id}')
2 def get_student(student_id: str):
3     for record in db:
4         if record['student_id'] == student_id:
5             return record
6
7 @app.put('/student_db/{student_id}')
8 def update_student(student_id: str, student_name: str):
9     for record in db:
10        if record['student_id'] == student_id:
11            record['student_name'] = student_name
```

## ❖ Practice: Path Operations DELETE Method



```
1 from fastapi import FastAPI
2
3 db = [
4     {
5         'student_id': '20240001',
6         'student_name': 'A'
7     },
8     {
9         'student_id': '20240002',
10        'student_name': 'B'
11    }
12 ]
13
14 app = FastAPI()
```



```
1 @app.get('/student_db/{student_id}')
2 def get_student(student_id: str):
3     for record in db:
4         if record['student_id'] == student_id:
5             return record
6
7 @app.delete('/student_db/{student_id}')
8 def delete_student(student_id: str):
9     for record in db:
10        if record['student_id'] == student_id:
11            db.remove(record)
```

## ❖ Practice: Path Operations

FastAPI 0.1.0 OAS 3.1

[/openapi.json](#)

### default ^

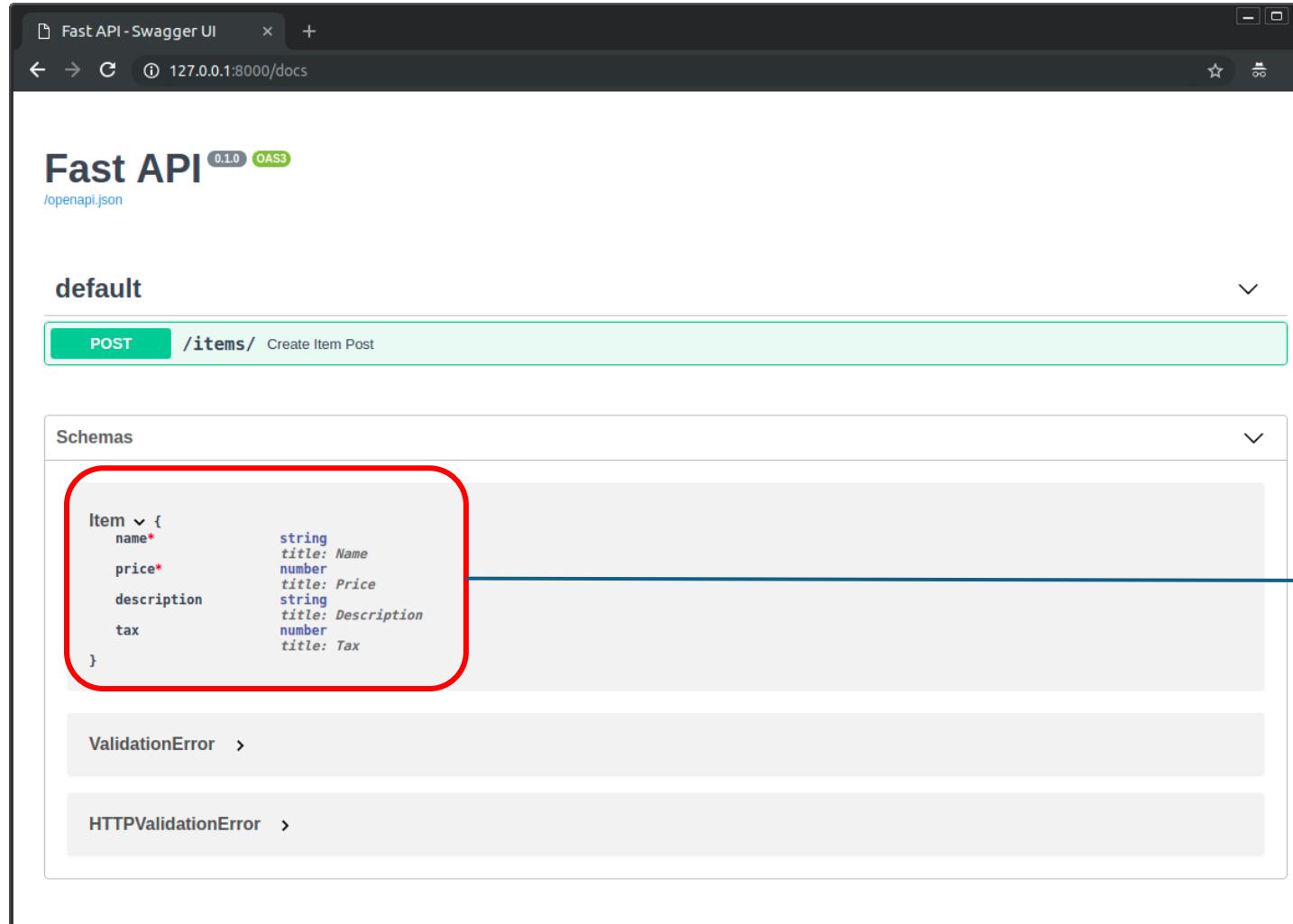
**GET** /student\_db/{student\_id} Get Student ▾

**PUT** /student\_db/{student\_id} Update Student ▾

**DELETE** /student\_db/{student\_id} Delete Student ▾

**POST** /student\_db Create Student ▾

## ❖ Practice: Pydantic Model



Request body: Data that is sent by Client.

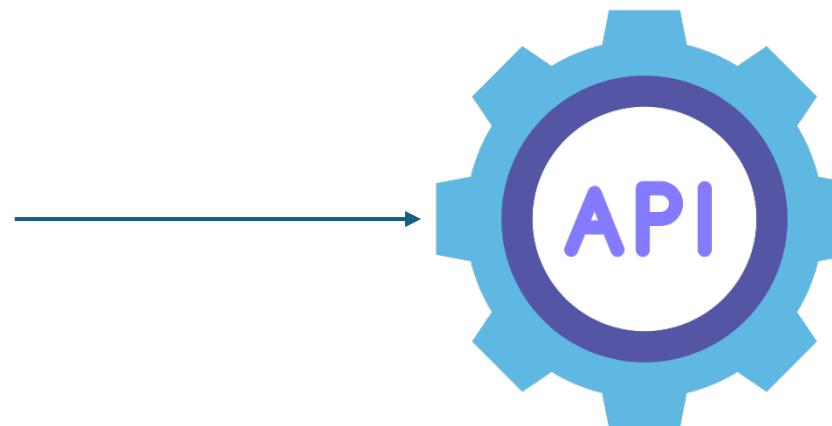
Response body: Data that is sent by API.

Item: {  
 "name": string,  
 "price": number,  
 "description": string,  
 "tax": number  
}

An example of request body

## ❖ Practice: Pydantic Model

```
Item: {  
    "name": string,  
    "price": number,  
    "description": string,  
    "tax": number  
}
```



An example of request body

To ensure API to receive the exact structure, we can use Pydantic Model.

## ❖ Practice: Pydantic Model



```
1 from pydantic import BaseModel
2
3 class Item(BaseModel):
4     name: str
5     price: float
6     description: str
7     tax: int
```

**Pydantic Model** is a way of defining data structures with type annotations, ensuring that the data adheres to a specified format and type (Data Validator). We can call a pydantic model as a model or schema.

## ❖ Practice: Pydantic Model

### E.g: Define student information record:

- ID: string
- Name: string
- Age: number
- Gender: string

```
1 from pydantic import BaseModel
2
3 class Student(BaseModel):
4     id: str
5     name: str
6     gender: str
7     age: int
8
9 a_student = Student(
10     id="20240001",
11     name="A",
12     gender="Male",
13     age=20
14 )
15
16 print(a_student)
17 # id='20240001' name='A' gender='Male' age=20
```

## ❖ Practice: Pydantic Model



```
1 from fastapi import FastAPI, HTTPException
2 from pydantic import BaseModel
3
4 class Student(BaseModel):
5     id: str
6     name: str
7     gender: str
8     age: int
9
10 db = [
11     Student(
12         id="20240001",
13         name="A",
14         gender="Male",
15         age=20
16     )
17 ]
18
19 app = FastAPI()
```



```
1 @app.post('/student_db')
2 def get_student(student: Student):
3     return student
4
5 @app.get('/student_db/{student_id}')
6 def get_student(student_id: str):
7     for s in db:
8         if s.id == student_id:
9             return s
10
11     raise HTTPException(status_code=404, detail='Student not found')
```

## ❖ Practice: Pydantic Model

default

POST /student\_db Get Student

Parameters Cancel

No parameters

Request body required application/json

```
{  
    "id": "string",  
    "name": "string",  
    "gender": "string",  
    "age": 0  
}
```

Execute

## ❖ Practice: Pydantic Model

```
1 from fastapi import FastAPI
2 from pydantic import BaseModel
3
4 app = FastAPI()
5
6 class Image(BaseModel):
7     url: str
8     name: str
9
10 class Item(BaseModel):
11     name: str
12     description: str | None = None
13     price: float
14     tax: float | None = None
15     tags: set[str] = set()
16     image: Image | None = None
```

```
1 @app.put("/items/{item_id}")
2 async def update_item(item_id: int, item: Item):
3     results = {"item_id": item_id, "item": item}
4     return results
```

A pydantic model can be nested

## ❖ Practice: Pydantic Model

FastAPI 0.1.0 OAS 3.1 /openapi.json

default ^

PUT /items/{item\_id} Update Item

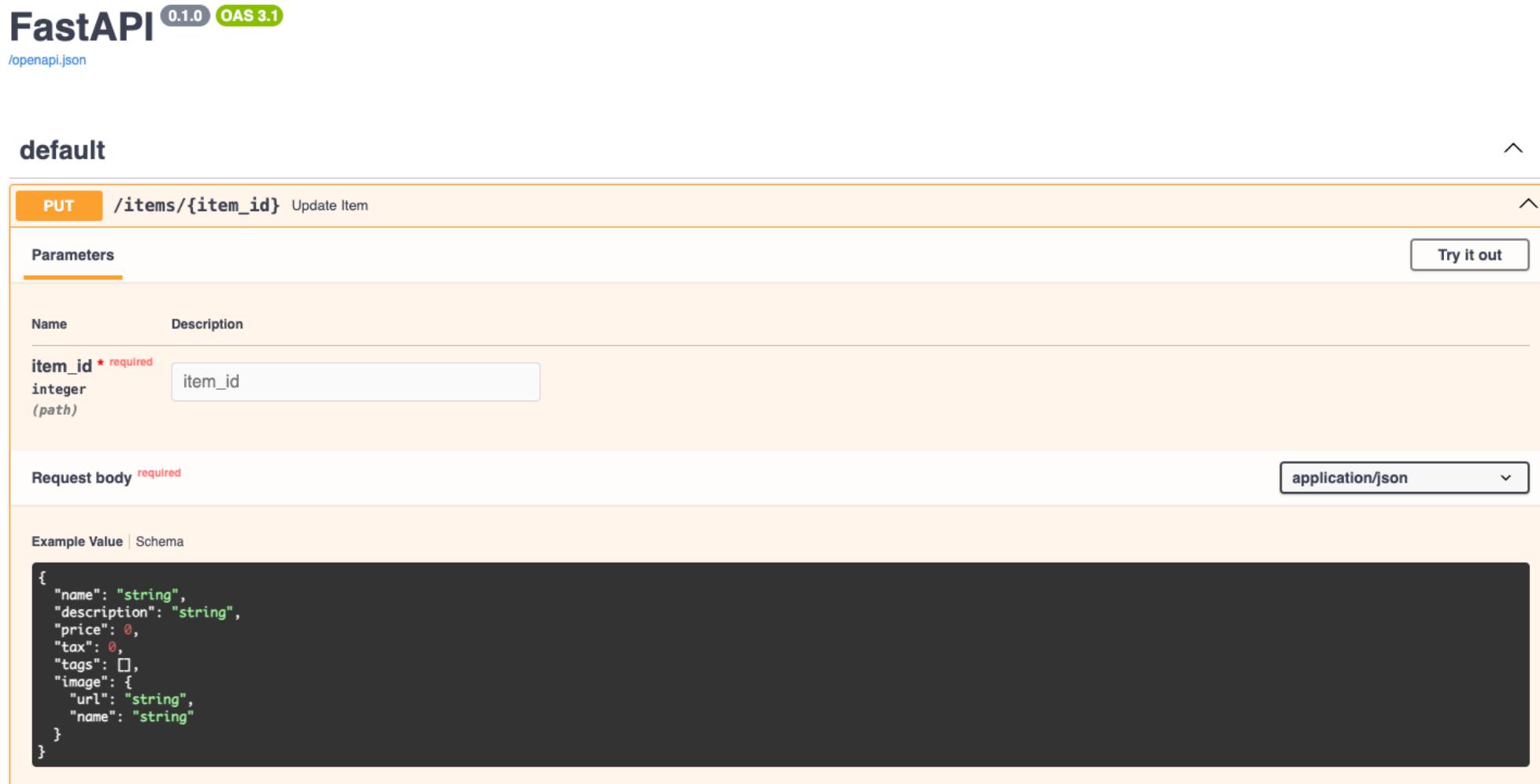
Parameters Try it out

Name	Description
item_id * required	integer (path)

Request body required application/json

Example Value | Schema

```
{  "name": "string",  "description": "string",  "price": 0,  "tax": 0,  "tags": [],  "image": {    "url": "string",    "name": "string"  }}
```



## ❖ Practice: Parameters Annotation

```
1 from typing import Optional, Union
2
3 def add(x: int, y: int, op: Optional[str]) -> int:
4     return x + y
5 add(3, 5, None)
6
7 def add2(x: int, y: int, op: Union[None, str]) -> int:
8     return x + y
9 add2(3, 5, None)
```

FastAPI allows us to provide additional information for parameters (path and query) using typing Annotated.

## ❖ Practice: Parameters Annotation

```
1 from typing import Annotated
2
3 from fastapi import FastAPI, Path, Query
4
5 app = FastAPI()
6
7 @app.get("/items/{item_id}")
8 async def read_items(
9     item_id: Annotated[int, Path(title="The ID of the item to get")],
10    q: Annotated[str | None, Query(alias="item-query")] = None,
11 ):
12    results = {"item_id": item_id}
13    if q:
14        results.update({"q": q})
15    return results
```

Use Annotated to give additional information for read\_items() function.

## ❖ Practice: Response Model

```
● ● ●  
1 from typing import Any  
2  
3 from fastapi import FastAPI  
4 from pydantic import BaseModel  
5  
6 app = FastAPI()  
7  
8 class Item(BaseModel):  
9     name: str  
10    description: str | None = None  
11    price: float  
12    tax: float | None = None  
13    tags: list[str] = []
```

```
● ● ●  
1 @app.post("/items/", response_model=Item)  
2 async def create_item(item: Item) -> Any:  
3     return item  
4  
5 @app.get("/items/", response_model=list[Item])  
6 async def read_items() -> Any:  
7     return [  
8         {"name": "Portal Gun", "price": 42.0},  
9         {"name": "Plumbus", "price": 32.0},  
10    ]
```

We can specify the response type of the route

## ❖ Practice: Status Code

### HTTP STATUS CODES

#### 2xx Success

200

Success / OK

#### 3xx Redirection

301

Permanent Redirect

302

Temporary Redirect

304

Not Modified

#### 4xx Client Error

401

Unauthorized Error

403

Forbidden

404

Not Found

405

Method Not Allowed

#### 5xx Server Error

501

Not Implemented

502

Bad Gateway

503

Service Unavailable

504

Gateway Timeout

**HTTP status codes** are three-digit response codes or messages sent by a server to user at the other end of the server. These codes or responses enable servers to communicate with users on the internet.

## ❖ Practice: Status Code

### HTTP STATUS CODES

#### 2xx Success

**200** Success / OK

#### 3xx Redirection

- 301** Permanent Redirect
- 302** Temporary Redirect
- 304** Not Modified

#### 4xx Client Error

- 401** Unauthorized Error
- 403** Forbidden
- 404** Not Found
- 405** Method Not Allowed

#### 5xx Server Error

- 501** Not Implemented
- 502** Bad Gateway
- 503** Service Unavailable
- 504** Gateway Timeout

```
(project_api_env) thangduong@Duongs-MacBook-Pro 6_Respone_Model % uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['/Users/thangduong/Desktop/aio2023/m
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [14316] using StatReload
INFO:     Started server process [14318]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:53399 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:53399 - "GET /openapi.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:53400 - "GET /items/ HTTP/1.1" 200 OK
```

When a request is completed successfully, we got 200 OK

## ❖ Practice: Status Code

```
● ● ●  
1 from fastapi import FastAPI  
2  
3 app = FastAPI()  
4  
5 @app.post("/items/", status_code=201)  
6 async def create_item(name: str):  
7     return {"name": name}
```

We can define the status code return in decorator

default

POST /items/ Create Item

Parameters

Name	Description
name * required string (query)	name

Responses

Code	Description
201	Successful Response  Media type application/json Controls Accept header.  Example Value   Schema "string"

## ❖ Practice: Upload File

Drag and drop your files anywhere or

Plate19\_D28\_E7-2.mp4

Ready to upload



Plate19\_D28\_E9-2.mp4  
1413433B



Plate19\_D28\_E9.mp4  
1421942B



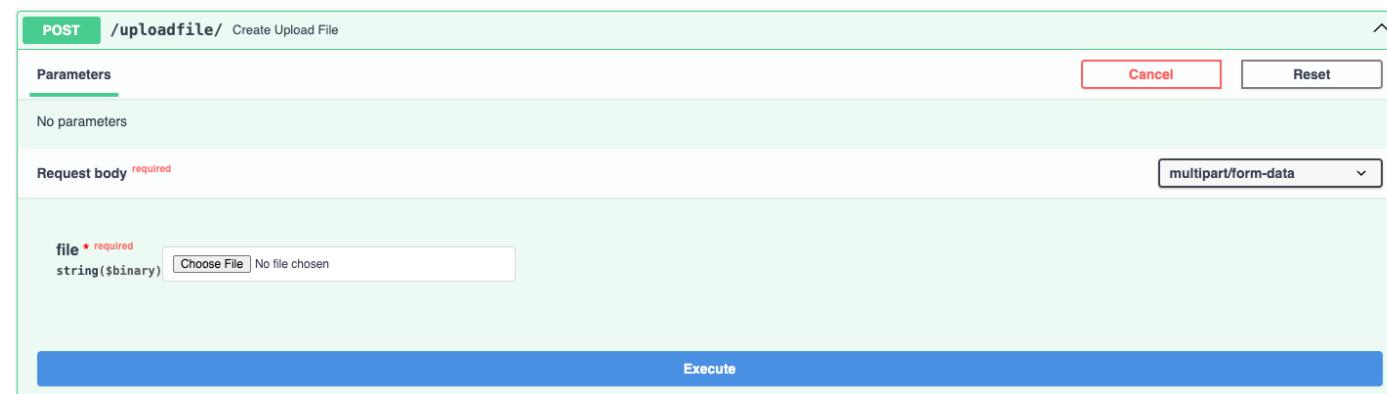
Plate19\_D28\_E7-2.mp4  
1403401B

Beside Pydantic model, we can also have request body as a uploaded file

## ❖ Practice: Upload File



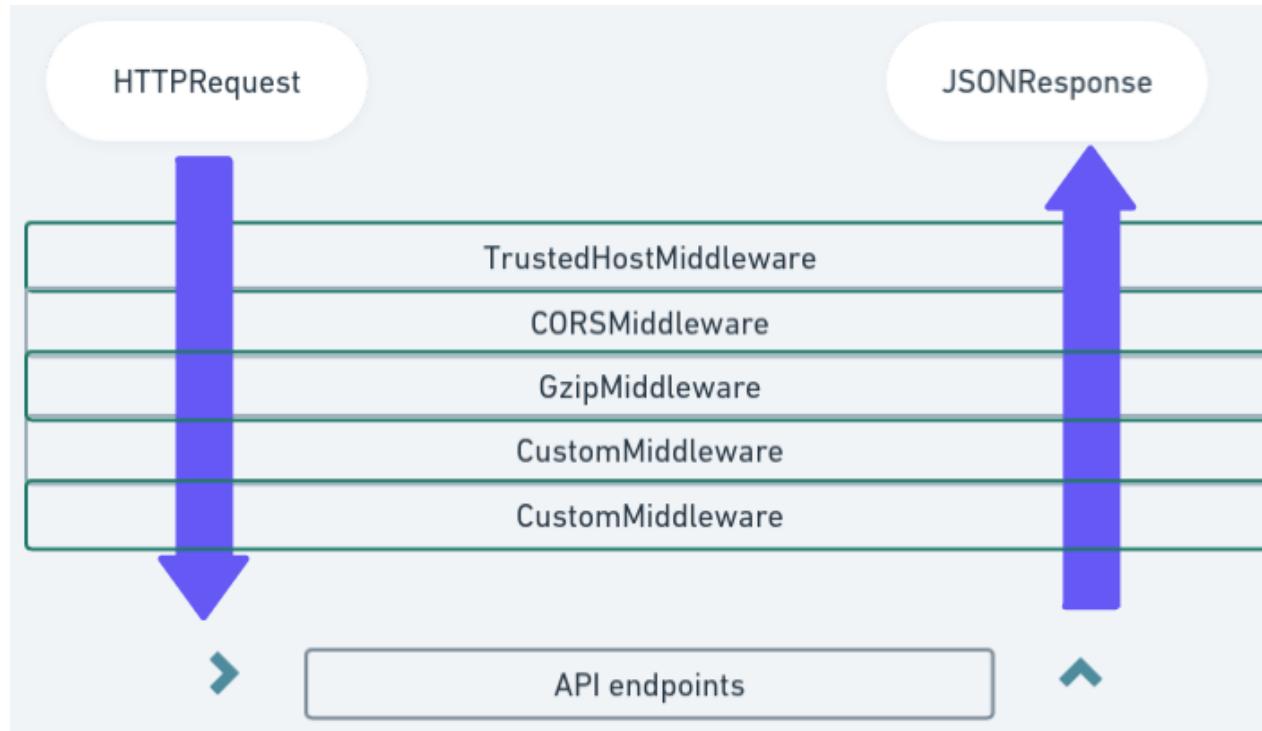
```
1 from fastapi import FastAPI, File, UploadFile
2 from typing_extensions import Annotated
3
4 app = FastAPI()
5
6 @app.post("/files/")
7 async def create_file(file: Annotated[bytes, File()]):
8     return {"file_size": len(file)}
9
10 @app.post("/uploadfile/")
11 async def create_upload_file(file: UploadFile):
12     return {"filename": file.filename}
```



Use File or UploadFile module in FastAPI

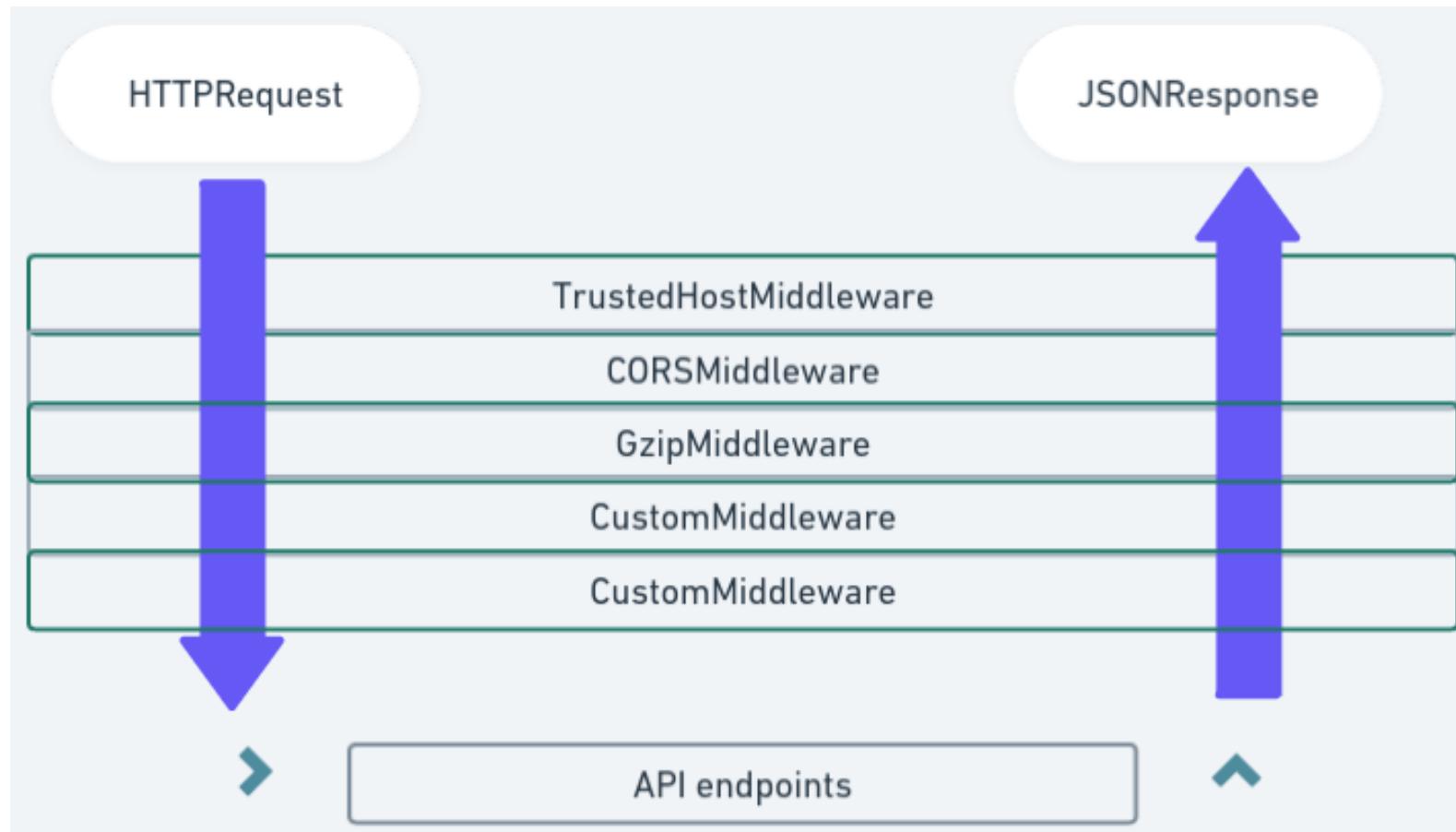
# FastAPI

## ❖ Practice: Middleware



A "**middleware**" is a function that works with every request before it is processed by any specific path operation. And also with every response before returning it.

## ❖ Practice: Middleware



## ❖ Practice: Middleware

```
1 import time
2
3 from fastapi import FastAPI, Request
4
5 app = FastAPI()
6
7 @app.get('/')
8 async def root():
9     return {
10         'msg': 'hello world'
11     }
```

```
1 @app.middleware("http")
2 async def add_process_time_header(request: Request,
3                                   call_next):
4     start_time = time.time()
5     response = await call_next(request)
6     process_time = time.time() - start_time
7     response.headers["X-Process-Time"] = str(process_time)
8
9     return response
```

Add a response time (process time) to response headers.

# FastAPI

## ❖ Practice: Middleware

default

GET / Root

Parameters

No parameters

Cancel

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/' \
-H 'accept: application/json'
```

Request URL

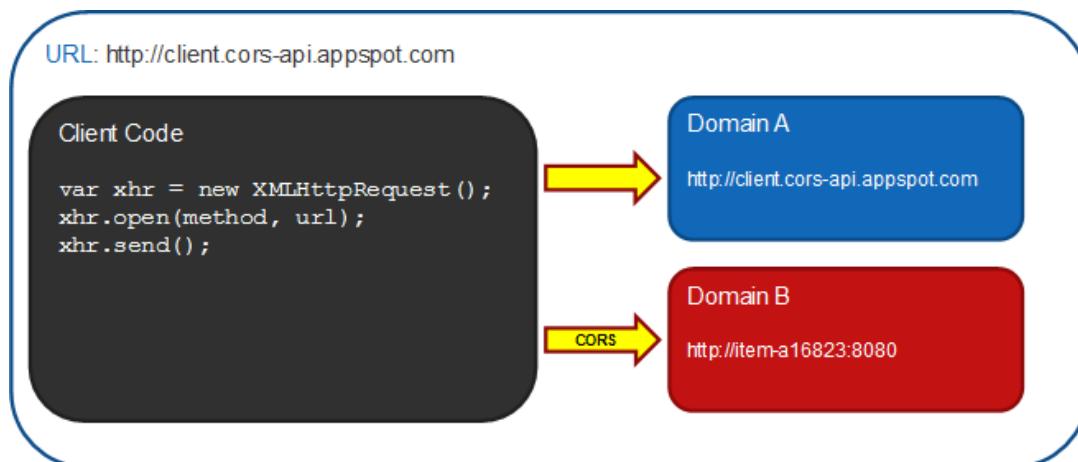
```
http://127.0.0.1:8000/
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "msg": "hello world" }</pre> <p>Download</p> <p>Response headers</p> <pre>content-length: 21 content-type: application/json date: Sat, 30 Mar 2024 11:09:40 GMT server: uvicorn x-process-time: 0.0004971027374267578</pre>

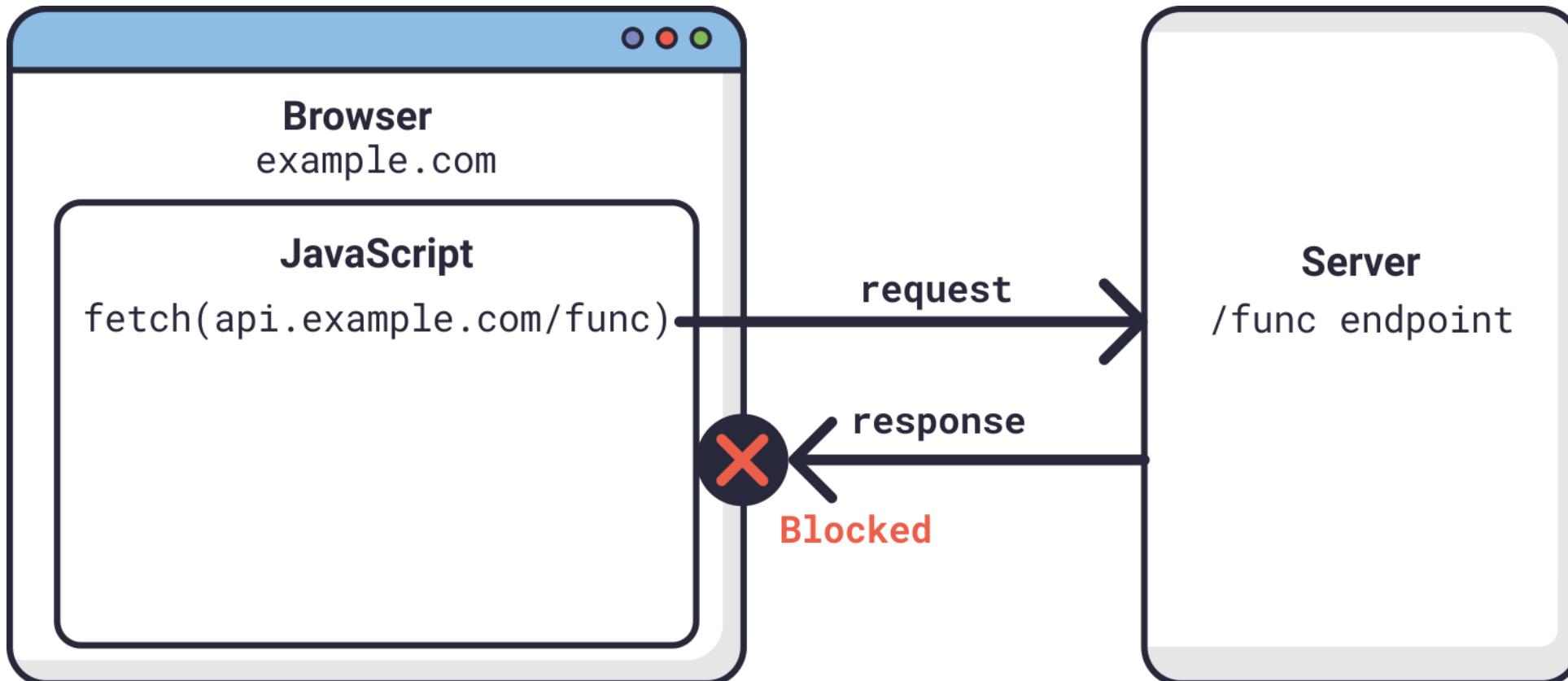
Responses

## ❖ Practice: CORS Middleware



**CORS (Cross-Origin Resource Sharing)** refers to the situations when a frontend running in a browser has JavaScript code that communicates with a backend, and the backend is in a different "origin" than the frontend.

## ❖ Practice: CORS Middleware



## ❖ Practice: CORS Middleware

```
1 from fastapi import FastAPI
2 from fastapi.middleware.cors import CORSMiddleware
3
4 app = FastAPI()
5
6 origins = [
7     "http://localhost.tiangolo.com",
8     "https://localhost.tiangolo.com",
9     "http://localhost",
10    "http://localhost:8080",
11 ]
```

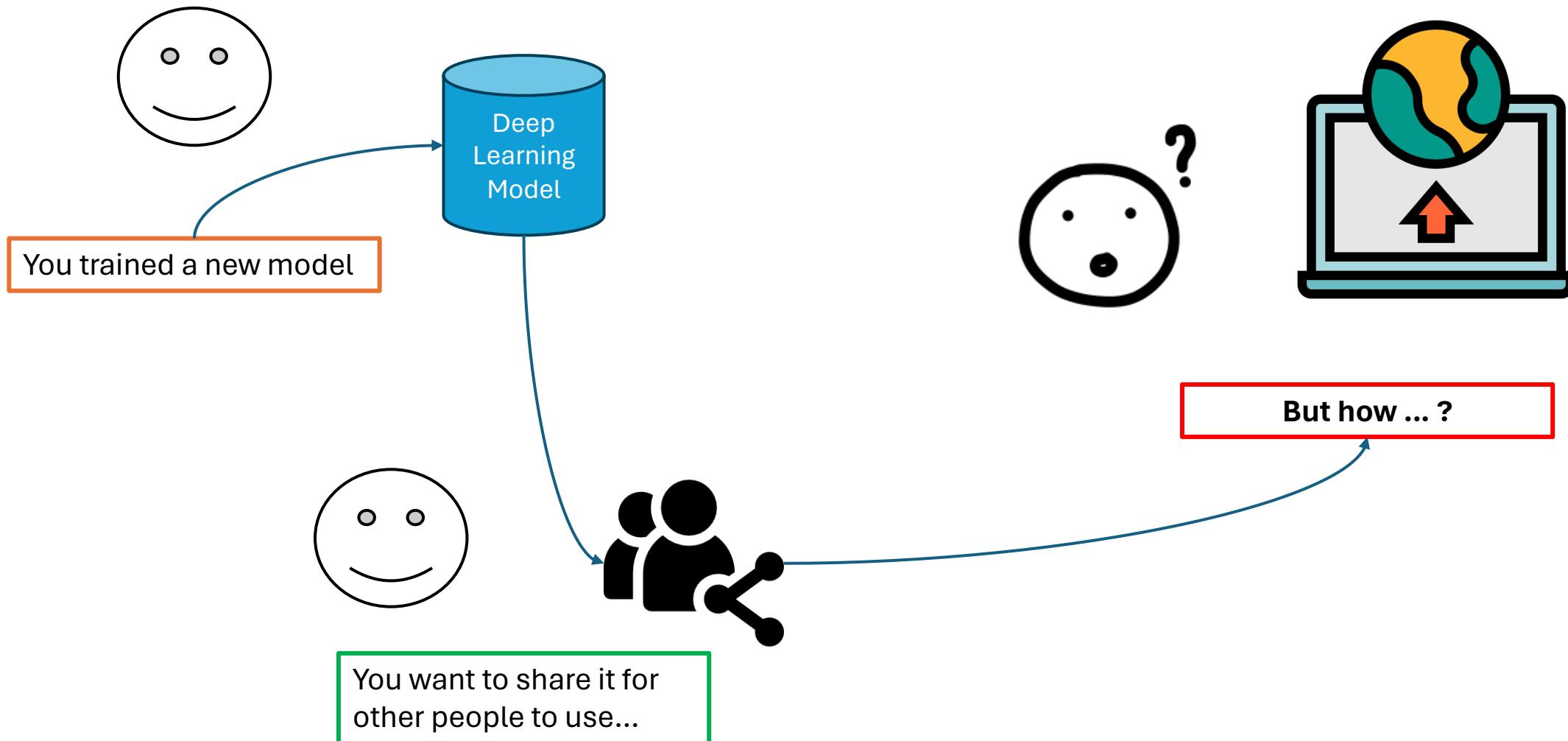
```
1 app.add_middleware(
2     CORSMiddleware,
3     allow_origins=origins,
4     allow_credentials=True,
5     allow_methods=["*"],
6     allow_headers=["*"],
7 )
8
9
10 @app.get("/")
11 async def main():
12     return {"message": "Hello World"}
```

# Quiz

# Model Deployment

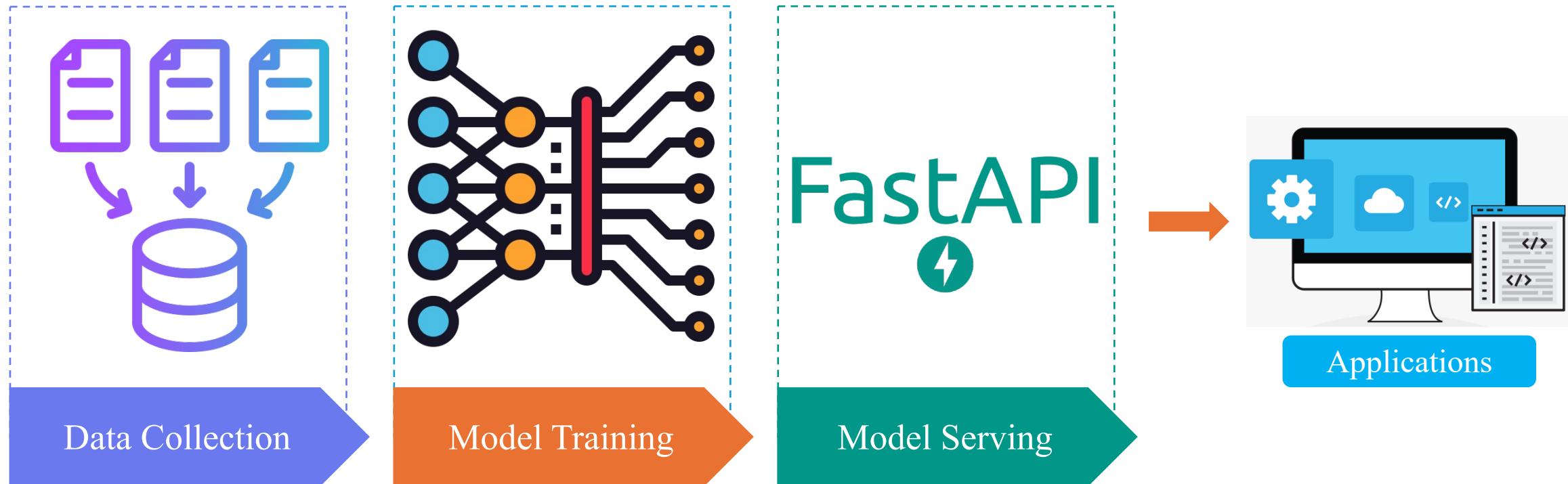
# Model Deployment

## ❖ Introduction



# Model Deployment

## ❖ Pipeline



# Model Deployment

## ❖ Practice

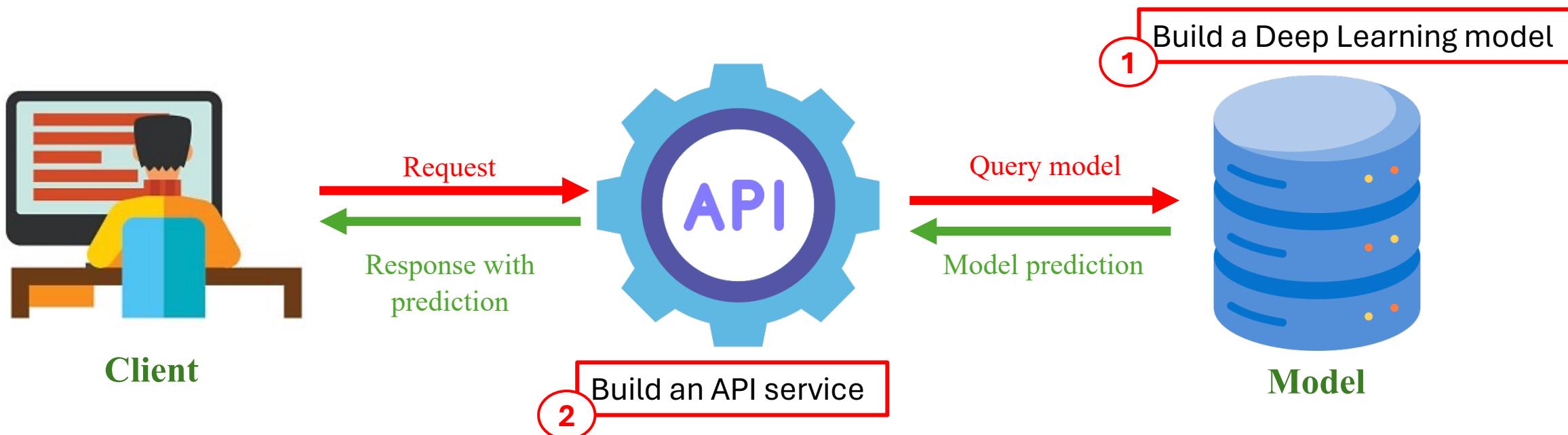
**Problem statement:** Deploy a Cat Dog Image Classification model as an API service using FastAPI.



# Model Deployment

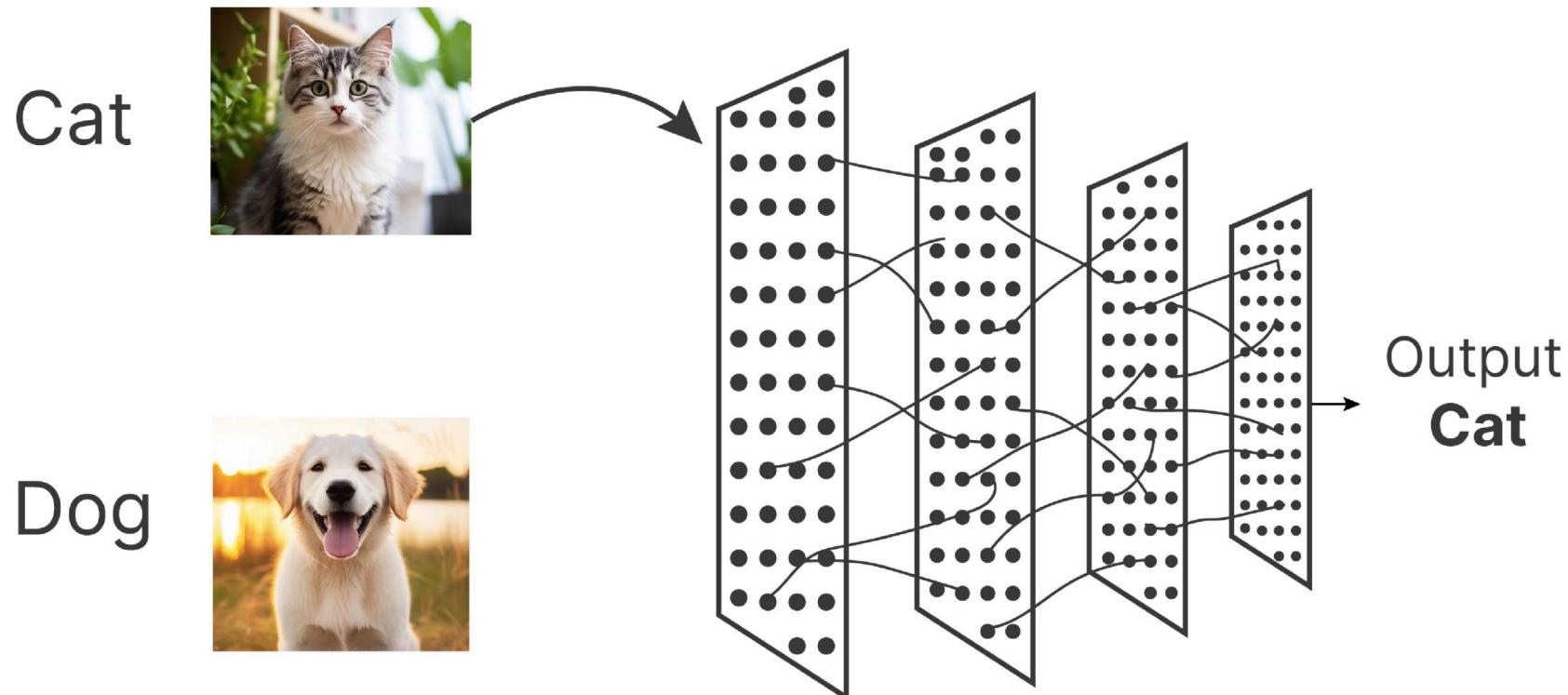
## ❖ Practice

**Problem statement:** Deploy a Cat Dog Image Classification model as an API service using FastAPI.



# Model Deployment

## ❖ Build model



# Model Deployment

## ❖ Build model: Dataset

Dataset Preview ⓘ

Split (1)  
train

The full dataset viewer is not available (click to read why). Only showing a preview of the rows.

image	labels
image	class label
	0 cat
	0 cat
	0 cat
	0 cat
	0 cat

[Cat Dog dataset](#) on HuggingFace.

# Model Deployment

## ❖ Build model step 1: Install and import libraries

```
1 !pip install datasets==2.18.0 -q

Collecting datasets
  Downloading datasets-2.18.0-py3-none-any.whl (510 kB)
    ━━━━━━━━━━━━━━━━━━━━ 510.5/510.5 kB 3.2 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets) (3.13.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.25.2)
Requirement already satisfied: pyarrow>=12.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (14.0.2)
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.10/dist-packages (from datasets) (0.6)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl (116 kB)
    ━━━━━━━━━━━━━━━━━━ 116.0/116.0 kB 1.0 MB/s eta 0:00:00
```

```
1 import torch
2 import torch.nn as nn
3
4 from PIL import Image
5 from datasets import load_dataset
6 from torch.utils.data import Dataset, DataLoader
7 from torchvision.models import resnet18
8 from torchvision import transforms
```

# Model Deployment

## ❖ Build model step 2: Load dataset and train val test split

```
1 DATASET_NAME = 'cats_vs_dogs'  
2 datasets = load_dataset(DATASET_NAME)  
3 datasets  
  
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarning:  
The secret `HF_TOKEN` does not exist in your Colab secrets.  
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://hub.huggingface.co).  
You will be able to reuse this secret in all of your notebooks.  
Please note that authentication is recommended but still optional to access public models  
    warnings.warn(  
/usr/local/lib/python3.10/dist-packages/datasets/load.py:1461: FutureWarning: The repository  
You can avoid this message in future by passing the argument `trust_remote_code=True`.  
Passing `trust_remote_code=True` will be mandatory to load this dataset from the next major  
    warnings.warn(  
DatasetDict({  
    train: Dataset({  
        features: ['image', 'labels'],  
        num_rows: 23410  
    })  
})  
  
1 TEST_SIZE = 0.2  
2 datasets['train'].train_test_split(test_size=TEST_SIZE)
```

# Model Deployment

## ❖ Build model step 3: Create DataLoader

```
1 IMG_SIZE = 64
2 img_transforms = transforms.Compose([
3     transforms.Resize((IMG_SIZE, IMG_SIZE)),
4     transforms.Grayscale(num_output_channels=3),
5     transforms.ToTensor(),
6     transforms.Normalize(
7         [0.485, 0.456, 0.406],
8         [0.229, 0.224, 0.225]
9     )
10])
```

```
1 class CatDogDataset(Dataset):
2     def __init__(self, data, transform=None):
3         self.data = data
4         self.transform = transform
5
6     def __len__(self):
7         return len(self.data)
8
9     def __getitem__(self, idx):
10        images = self.data[idx]['image']
11        labels = self.data[idx]['labels']
12
13        if self.transform:
14            images = self.transform(images)
15
16        labels = torch.tensor(labels, dtype=torch.long)
17
18        return images, labels
```

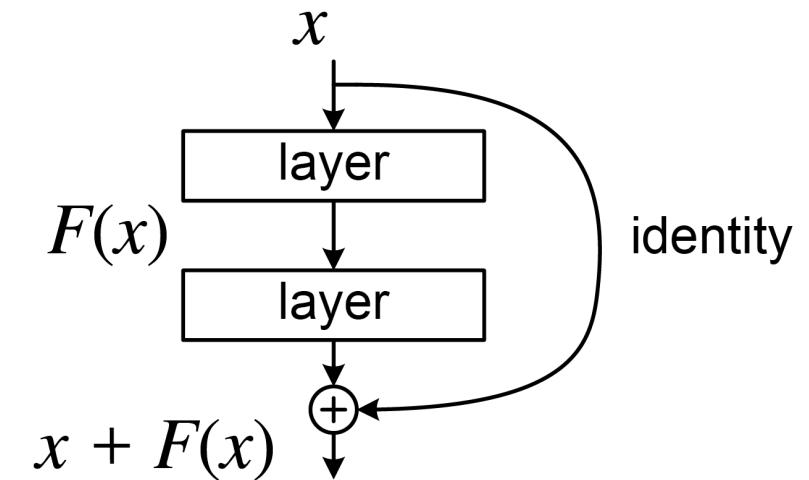
```
1 TRAIN_BATCH_SIZE = 512
2 VAL_BATCH_SIZE = 256
3
4 train_dataset = CatDogDataset(
5     datasets['train'], transform=img_transforms
6 )
7 test_dataset = CatDogDataset(
8     datasets['test'], transform=img_transforms
9 )
10
11 train_loader = DataLoader(
12     train_dataset,
13     batch_size=TRAIN_BATCH_SIZE,
14     shuffle=True
15 )
16 test_loader = DataLoader(
17     test_dataset,
18     batch_size=VAL_BATCH_SIZE,
19     shuffle=False
20 )
```

# Model Deployment

## ❖ Build model step 4: Build model

```
1 class CatDogModel(nn.Module):
2     def __init__(self, n_classes):
3         super(CatDogModel, self).__init__()
4
5         resnet_model = resnet18(weights='IMAGENET1K_V1')
6         self.backbone = nn.Sequential(*list(resnet_model.children())[:-1])
7         for param in self.backbone.parameters():
8             param.requires_grad = False
9
10        in_features = resnet_model.fc.in_features
11        self.fc = nn.Linear(in_features, n_classes)
12
13    def forward(self, x):
14        x = self.backbone(x)
15        x = torch.flatten(x, 1)
16        x = self.fc(x)
17
18        return x
19
20 device = 'cuda' if torch.cuda.is_available() else 'cpu'
21 N_CLASSES = 2
22 model = CatDogModel(N_CLASSES).to(device)
23 test_input = torch.rand(1, 3, 224, 224).to(device)
24 with torch.no_grad():
25     output = model(test_input)
26     print(output.shape)

torch.Size([1, 2])
```



# Model Deployment

## ❖ Build model step 5: Training and saving model weights

```
1 EPOCHS = 10
2 LR = 1e-3
3 WEIGHT_DECAY = 1e-5
4
5 optimizer = torch.optim.Adam(model.parameters(), lr=LR, weight_decay=WEIGHT_DECAY)
6 criterion = torch.nn.CrossEntropyLoss()

7 for epoch in range(EPOCHS):
8     train_losses = []
9     model.train()
10    for images, labels in train_loader:
11        images = images.to(device)
12        labels = labels.to(device)
13
14        outputs = model(images)
15
16        optimizer.zero_grad()
17        loss = criterion(outputs, labels)
18        loss.backward()
19        optimizer.step()
20
21        train_losses.append(loss.item())
22
23    train_loss = sum(train_losses) / len(train_losses)

24    val_losses = []
25    model.eval()
26    with torch.no_grad():
27        for images, labels in test_loader:
28            images = images.to(device)
29            labels = labels.to(device)
30
31            outputs = model(images)
32            loss = criterion(outputs, labels)
33
34            val_losses.append(loss.item())
35
36    val_loss = sum(val_losses) / len(val_losses)
37
38    print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.3f}\tVal loss: {val_loss:.3f}'')
```

EPOCH 1:	Train loss: 0.658	Val loss: 0.612
EPOCH 2:	Train loss: 0.544	Val loss: 0.542
EPOCH 3:	Train loss: 0.526	Val loss: 0.529
EPOCH 4:	Train loss: 0.516	Val loss: 0.525
EPOCH 5:	Train loss: 0.513	Val loss: 0.530
EPOCH 6:	Train loss: 0.508	Val loss: 0.535
EPOCH 7:	Train loss: 0.508	Val loss: 0.521
EPOCH 8:	Train loss: 0.504	Val loss: 0.517
EPOCH 9:	Train loss: 0.505	Val loss: 0.518
EPOCH 10:	Train loss: 0.507	Val loss: 0.516

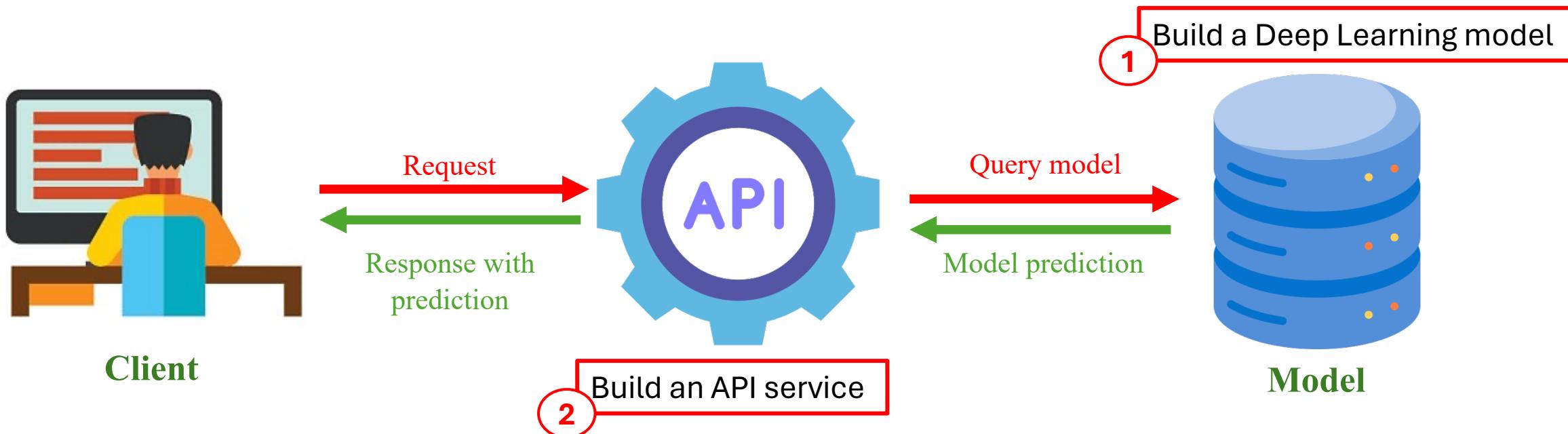
## 7. Save model

```
1 SAVE_PATH = 'catdog_weights.pt'
2 torch.save(model.state_dict(), SAVE_PATH)
```

# Model Deployment

## ❖ Build API

**Problem statement:** Deploy a Cat Dog Image Classification model as an API service using FastAPI.



# Model Deployment

## ❖ Build API Step 1: Organize source code

The screenshot shows a file explorer window with the following directory structure:

- Basic\_Structure
  - \_\_pycache\_\_
  - config
    - \_\_pycache\_\_
    - catdog\_cfg.py
    - logging\_cfg.py
  - logs
    - http.log
    - predictor.log
  - middleware
    - \_\_pycache\_\_
    - \_\_init\_\_.py
    - cors.py
    - http.py
  - models
    - \_\_pycache\_\_
    - weights
      - catdog\_model.py
      - catdog\_predictor.py
  - routes
    - \_\_pycache\_\_
    - base.py
    - catdog\_route.py
  - schemas
    - \_\_pycache\_\_
    - catdog\_schema.py
  - utils
    - \_\_pycache\_\_
    - logger.py
    - app.py
  - requirements.txt
  - server.py

We build this API in local computer with the following source code structure

# Model Deployment

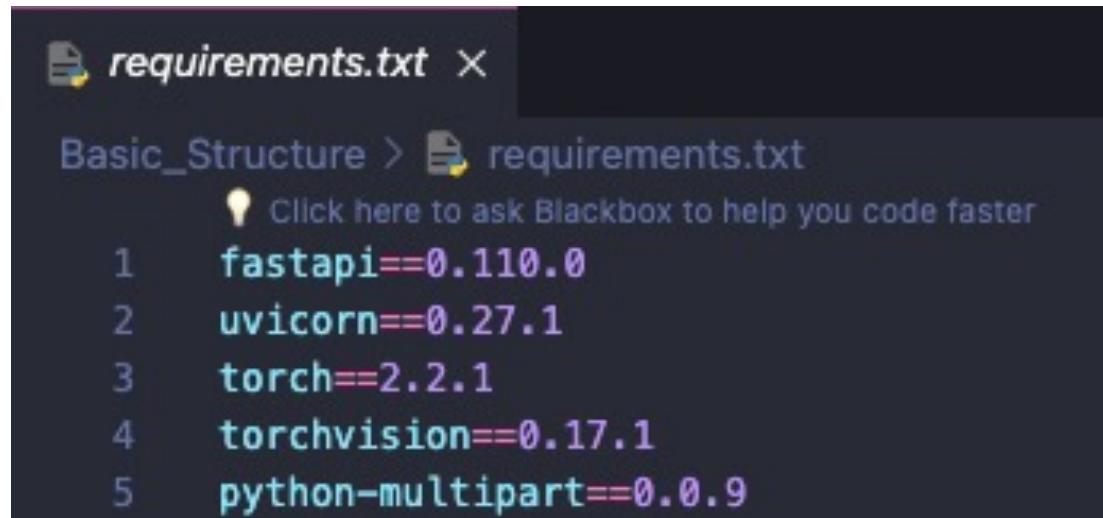
## ❖ Build API Step 1: Organize source code

```
root/
  - config/
    - catdog_cfg.py
    - logging_cfg.py
  - logs/
  - middleware/
    - __init__.py
    - cors.py
    - http.py
  - models/
    - weights/
      - catdog_weights.pt
    - catdog_model.py
    - catdog_predictor.py
  - routes/
    - base.py
    - catdog_route.py
  - schemas/
    - catdog_schema.py
  - utils/
    - logger.py
  - app.py
  - requirements.txt
  - server.py
```

- **config/**: Folder containing configuration for some modules.
- **logs/**: Folder containing logging information when running API.
- **middleware/**: Folder containing code for middleware.
- **models/**: Folder containing Deep Learning weights.
- **routes/**: Folder containing API Endpoints declaration.
- **schemas/**: Folder containing Pydantic model declaration.
- **utils/**: Folder containing codes for general purpose (varying between projects).
- **app.py**: Python file containing codes for FastAPI app initialization.
- **requirements.txt**: File containing packages version information to run the source code.
- **server.py**: Python file containing codes to host the API service.

# Model Deployment

## ❖ Build API Step 2: List packages requirements



A screenshot of a code editor showing a file named "requirements.txt". The file contains the following content:

```
fastapi==0.110.0
uvicorn==0.27.1
torch==2.2.1
torchvision==0.17.1
python-multipart==0.0.9
```

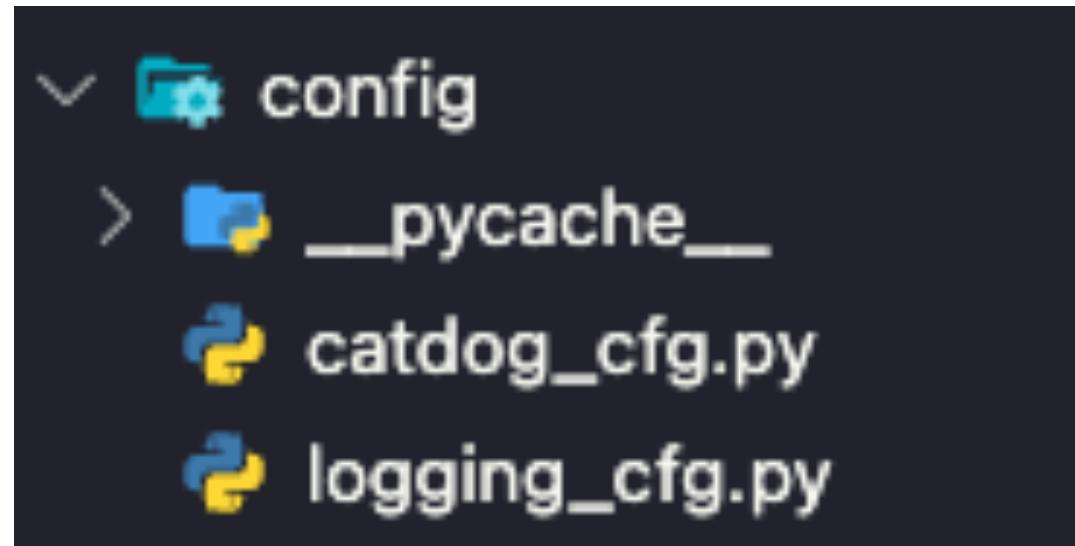


A screenshot of a terminal window with a dark theme. It shows a command to install packages from a requirements file:

```
$ pip install -r requirements.txt
```

# Model Deployment

## ❖ Build API Step 3: Define configuration



We define configuration for Deep Learning model  
and logging module

# Model Deployment

## ❖ Build API Step 3: Define configuration

catdog\_cfg.py ×

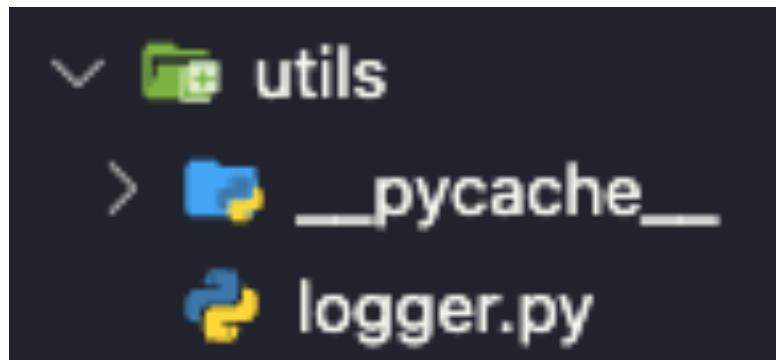
```
Basic_Structure > config > catdog_cfg.py > ...
    ⚡ Click here to ask Blackbox to help you code faster
1 import sys
2
3 from pathlib import Path
4 sys.path.append(str(Path(__file__).parent))
5
6 class CatDogDataConfig:
7     N_CLASSES = 2
8     IMG_SIZE = 64
9     ID2DLABEL = {0: 'Cat', 1: 'Dog'}
10    LABEL2ID = {'Cat': 0, 'Dog': 1}
11    NORMALIZE_MEAN = [0.485, 0.456, 0.406]
12    NORMALIZE_STD = [0.229, 0.224, 0.225]
13
14 class ModelConfig:
15     ROOT_DIR = Path(__file__).parent.parent
16     MODEL_NAME = 'resnet18'
17     MODEL_WEIGHT = ROOT_DIR / 'models' / 'weights' / 'catdog_weights.pt'
18     DEVICE = 'cpu'
```

logging\_cfg.py ×

```
Basic_Structure > config > logging_cfg.py > ...
    ⚡ Click here to ask Blackbox to help you code faster
1 from pathlib import Path
2
3 class LoggingConfig:
4     ROOT_DIR = Path(__file__).parent.parent
5
6     LOG_DIR = ROOT_DIR / "logs"
7
8     LoggingConfig.LOG_DIR.mkdir(parents=True, exist_ok=True)
```

# Model Deployment

## ❖ Build API Step 4: Build logging function



```
 1 import sys
 2 import logging
 3
 4 from pathlib import Path
 5 sys.path.append(str(Path(__file__).parent))
 6
 7 from logging.handlers import RotatingFileHandler
 8 from config.logging_cfg import LoggingConfig
 9
10 class Logger:
11     def __init__(self, name="", log_level=logging.INFO, log_file=None) -> None:
12         self.log = logging.getLogger(name)
13         self.get_logger(log_level, log_file)
14
15     def get_logger(self, log_level, log_file):
16         self.log.setLevel(log_level)
17         self._init_formatter()
18         if log_file is not None:
19             self._add_file_hander(LoggingConfig.LOG_DIR / log_file)
20         else:
21             self._add_stream_hander()
```

# Model Deployment

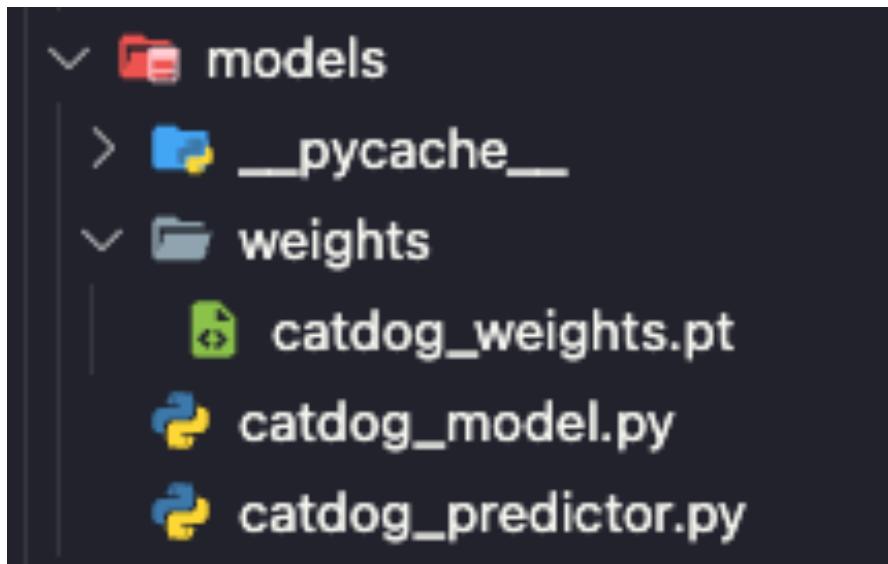
## ❖ Build API Step 4: Build logging function

```
1 # Continue previous code
2
3 def _init_formatter(self):
4     self.formatter = logging.Formatter(
5         "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
6     )
7 def _add_stream_hander(self):
8     stream_handler = logging.StreamHandler(sys.stdout)
9     stream_handler.setFormatter(self.formatter)
10    self.log.addHandler(stream_handler)
11
12 def _add_file_hander(self, log_file):
13    file_handler = RotatingFileHandler(log_file, maxBytes=10000, backupCount=10)
14    file_handler.setFormatter(self.formatter)
15    self.log.addHandler(file_handler)
16
17 def log_model(self, predictor_name):
18    self.log.info(f"Predictor name: {predictor_name}")
19
20 def log_response(self, pred_prob, pred_id, pred_class):
21    self.log.info(f"Predicted Prob: {pred_prob} - Predicted ID: {pred_id} -
Predicted Class: {pred_class}")
```

```
INFO - 127.0.0.1 - "GET /docs 1.1" 200 0.00s
INFO - 127.0.0.1 - "GET /openapi.json 1.1" 200 0.00s
INFO - 127.0.0.1 - "POST /catdog_classification/predict 1.1" 200 0.13s
INFO - 127.0.0.1 - "POST /catdog_classification/predict 1.1" 200 0.04s
```

# Model Deployment

## ❖ Build API Step 5: Define models folder



```
•••  
1 import torch  
2 import torch.nn as nn  
3  
4 from torchvision.models import resnet18  
5  
6 class CatDogModel(nn.Module):  
7     def __init__(self, n_classes):  
8         super(CatDogModel, self).__init__()  
9  
10        resnet_model = resnet18(weights='IMAGENET1K_V1')  
11        self.backbone = nn.Sequential(*list(resnet_model.children())[:-1])  
12        for param in self.backbone.parameters():  
13            param.requires_grad = False  
14  
15        in_features = resnet_model.fc.in_features  
16        self.fc = nn.Linear(in_features, n_classes)  
17  
18    def forward(self, x):  
19        x = self.backbone(x)  
20        x = torch.flatten(x, 1)  
21        x = self.fc(x)  
22  
23    return x
```

catdog\_model.py

# Model Deployment

## ❖ Build API Step 5: Define models folder



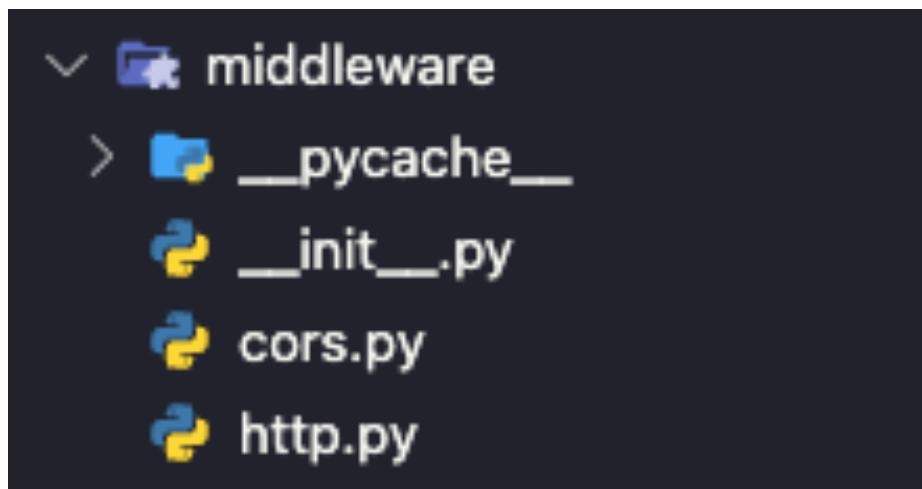
```
1 import sys
2 import torch
3 import torchvision
4
5 from pathlib import Path
6 sys.path.append(str(Path(__file__).parent.parent))
7
8 from PIL import Image
9 from torch.nn import functional as F
10 from utils.logger import Logger
11 from config.catdog_cfg import CatDogDataConfig
12 from .catdog_model import CatDogModel
13
14 LOGGER = Logger(__file__, log_file="predictor.log")
15 LOGGER.info("Starting Model Serving")
```

catdog\_predictor.py

```
•••
1 class Predictor:
2     def __init__(self, model_name: str, model_weight: str, device: str = "cpu"):
3         self.model_name = model_name
4         self.model_weight = model_weight
5         self.device = device
6         self.load_model()
7         self.create_transform()
8
9     async def predict(self, image):
10        pil_img = Image.open(image)
11
12        if pil_img.mode == 'RGBA':
13            pil_img = pil_img.convert('RGB')
14
15        transformed_image = self.transforms_(pil_img).unsqueeze(0)
16        output = await self.model_inference(transformed_image)
17        probs, best_prob, predicted_id, predicted_class = self.output2pred(output)
18
19        LOGGER.log_model(self.model_name)
20        LOGGER.log_response(best_prob, predicted_id, predicted_class)
21
22        torch.cuda.empty_cache()
23
24        resp_dict = {
25            "probs":probs,
26            "best_prob": best_prob,
27            "predicted_id": predicted_id,
28            "predicted_class": predicted_class,
29            "predictor_name": self.model_name
30        }
31
32        return resp_dict
```

# Model Deployment

## ❖ Build API Step 6: Define middleware folder



We declare each Middleware in separated files

A screenshot of a code editor showing the 'cors.py' file. The code defines a function 'setup\_cors' that adds CORS middleware to an FastAPI application. It specifies allowing all origins, credentials, methods, and headers.

```
1 from fastapi.middleware.cors import CORSMiddleware
2
3 def setup_cors(app):
4     app.add_middleware(
5         CORSMiddleware,
6         allow_origins=[ "*" ],
7         allow_credentials=True,
8         allow_methods=[ "*" ],
9         allow_headers=[ "*" ],
10    )
```

cors.py

# Model Deployment

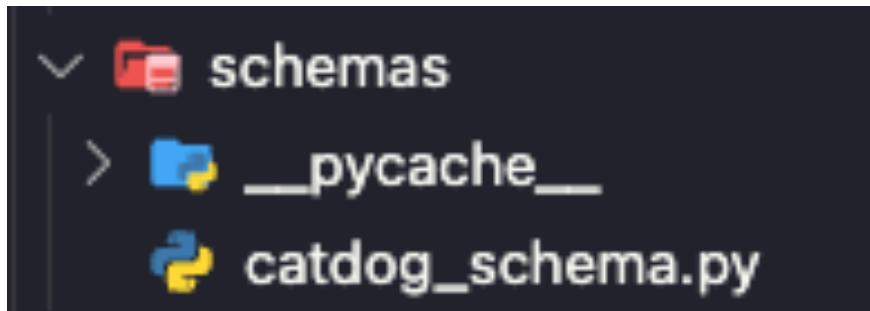
## ❖ Build API Step 6: Define middleware folder

```
 1 import time
 2 import sys
 3
 4 from pathlib import Path
 5 sys.path.append(str(Path(__file__).parent.parent))
 6
 7 from starlette.middleware.base import BaseHTTPMiddleware
 8 from starlette.requests import Request
 9 from utils.logger import Logger
10
11 LOGGER = Logger(__file__, log_file="http.log")
12
13 class LogMiddleware(BaseHTTPMiddleware):
14     async def dispatch(self, request: Request, call_next):
15         start_time = time.time()
16         response = await call_next(request)
17         process_time = time.time() - start_time
18         LOGGER.log.info(
19             f"{request.client.host} - \'{request.method} {request.url.path}\'
{request.scope['http_version']}\' {response.status_code} {process_time:.2f}s")
20
21     return response
```

http.py

# Model Deployment

## ❖ Build API Step 7: Define Schemas



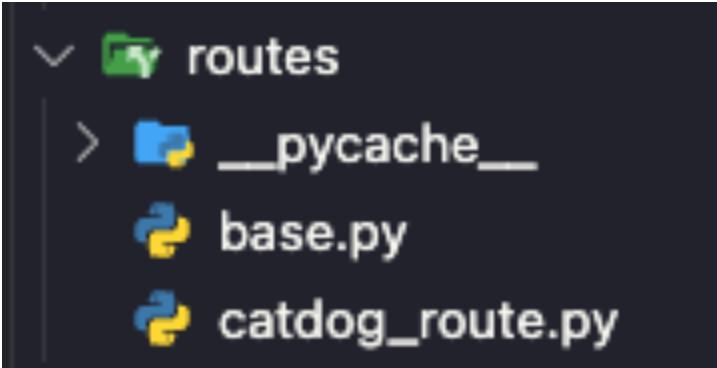
We define all Pydantic model in schemas folder



```
1 from pydantic import BaseModel  
2  
3 class CatDogResponse(BaseModel):  
4     probs: list = []  
5     best_prob: float = -1.0  
6     predicted_id: int = -1  
7     predicted_class: str = ""  
8     predictor_name: str = ""
```

# Model Deployment

## ❖ Build API Step 8: Define routes



base.py 1 X

Basic\_Structure > routes > base.py > ...  
💡 Click here to ask Blackbox to help you code faster

```
1 from fastapi import APIRouter
2 from .catdog_route import router as catdog_cls_route
3
4 router = APIRouter()
5 router.include_router(catdog_cls_route, prefix="/catdog_classification")
```

```
...
1 import sys
2 from pathlib import Path
3 sys.path.append(str(Path(__file__).parent))
4
5 from fastapi import File, UploadFile
6 from fastapi import APIRouter
7 from schemas.catdog_schema import CatDogResponse
8 from config.catdog_cfg import ModelConfig
9 from models.catdog_predictor import Predictor
10
11 router = APIRouter()
12 predictor = Predictor(
13     model_name=ModelConfig.MODEL_NAME,
14     model_weight=ModelConfig.MODEL_WEIGHT,
15     device=ModelConfig.DEVICE
16 )
17
18 @router.post("/predict")
19 async def predict(file_upload: UploadFile = File(...)):
20     response = await predictor.predict(file_upload.file)
21
22     return CatDogResponse(**response)
```

# Model Deployment

## ❖ Build API Step 9: Define app.py and server.py



```
1 import sys
2 from pathlib import Path
3 sys.path.append(str(Path(__file__).parent))
4
5 from fastapi import FastAPI
6 from middleware import LogMiddleware, setup_cors
7 from routes.base import router
8
9 app = FastAPI()
10
11 app.add_middleware(LogMiddleware)
12 setup_cors(app)
13 app.include_router(router)
```



```
1 import uvicorn
2
3 if __name__ == "__main__":
4     uvicorn.run("app:app", host="0.0.0.0", port=8000, reload=True)
```

# Model Deployment

## ❖ Result

default

POST /catdog\_classification/predict Predict

Parameters

No parameters

Request body required

multipart/form-data

file\_upload \* required string(\$binary)  
Choose File Dog Puppy Garden.jpg

Execute Clear

Responses

Curl

```
curl -X 'POST' \
'http://0.0.0.0:8000/catdog_classification/predict' \
-H 'accept: application/json' \
-H 'Content-type: multipart/form-data' \
-F 'file_upload=@Dog Puppy Garden.jpg;type=image/jpeg'
```

Request URL

http://0.0.0.0:8000/catdog\_classification/predict

Server response

Code Details

200

Response body

```
{
  "probs": [
    0.013586386106908321,
    0.9864135980606079
  ],
  "best_prob": 0.986414,
  "predicted_id": 1,
  "predicted_class": "Dog",
  "predictor_name": "resnet18"
}
```

Cancel Reset

1

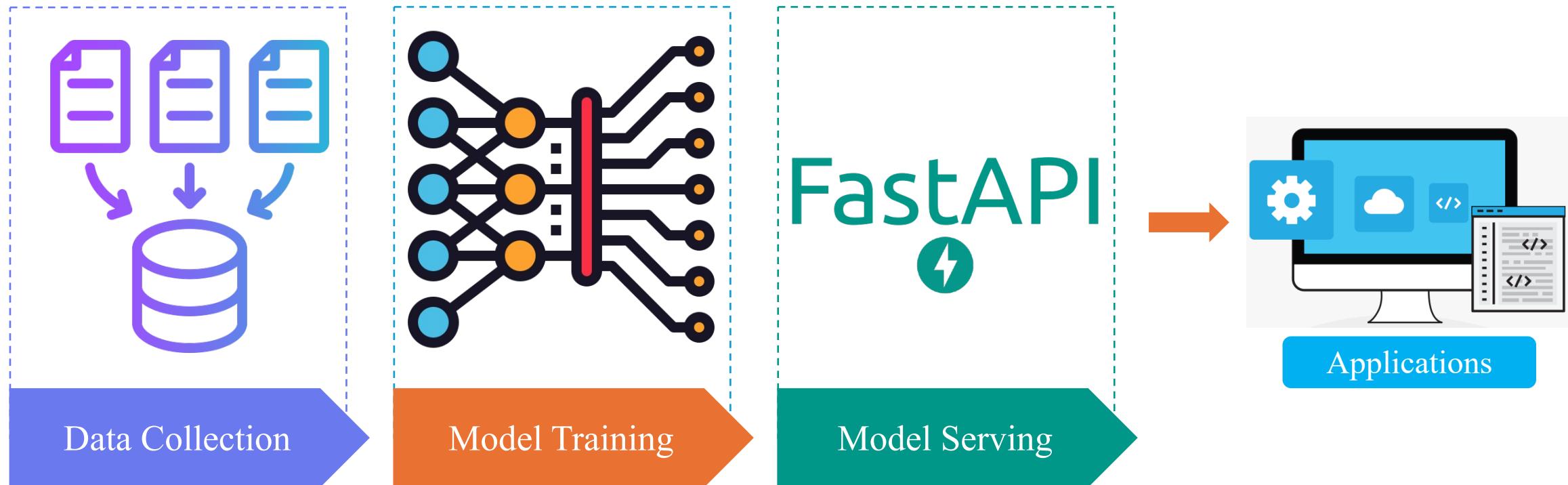
2

3

# Model Deployment (MLOps)

# Model Deployment (MLOps)

## ❖ Getting Started



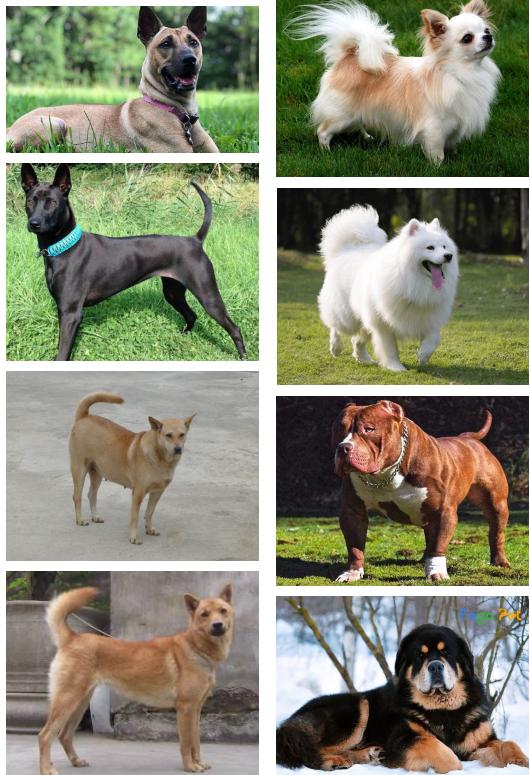
# Model Deployment (MLOps)

## ❖ Data Drift Problem

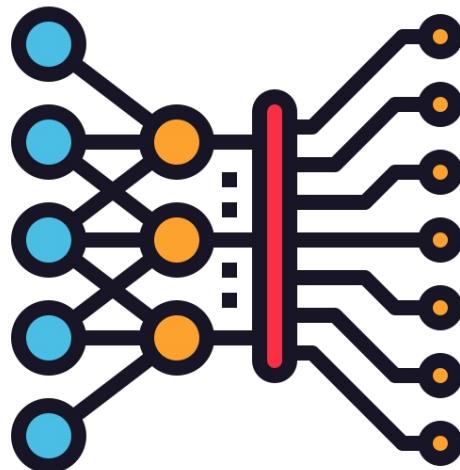


# Model Deployment (MLOps)

## ❖ Data Drift Solution



More Data Collection



Model Training

FastAPI

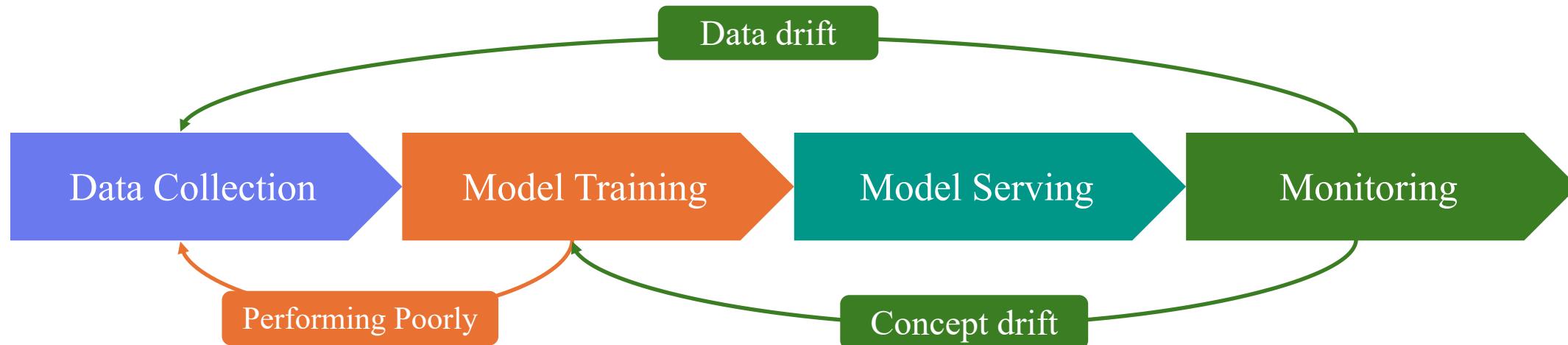


Applications

Model Serving

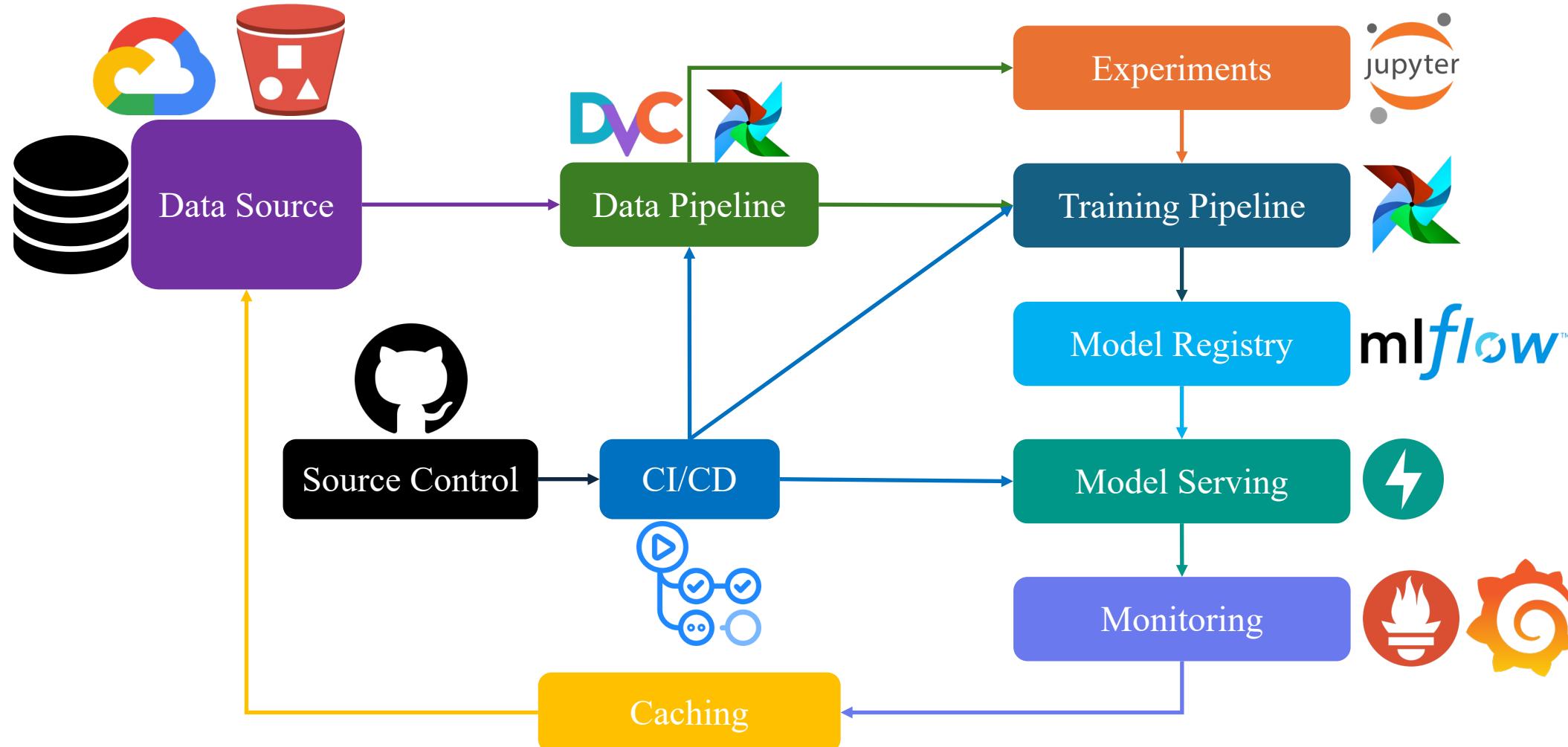
# Model Deployment (MLOps)

## ❖ Machine Learning Life Cycle



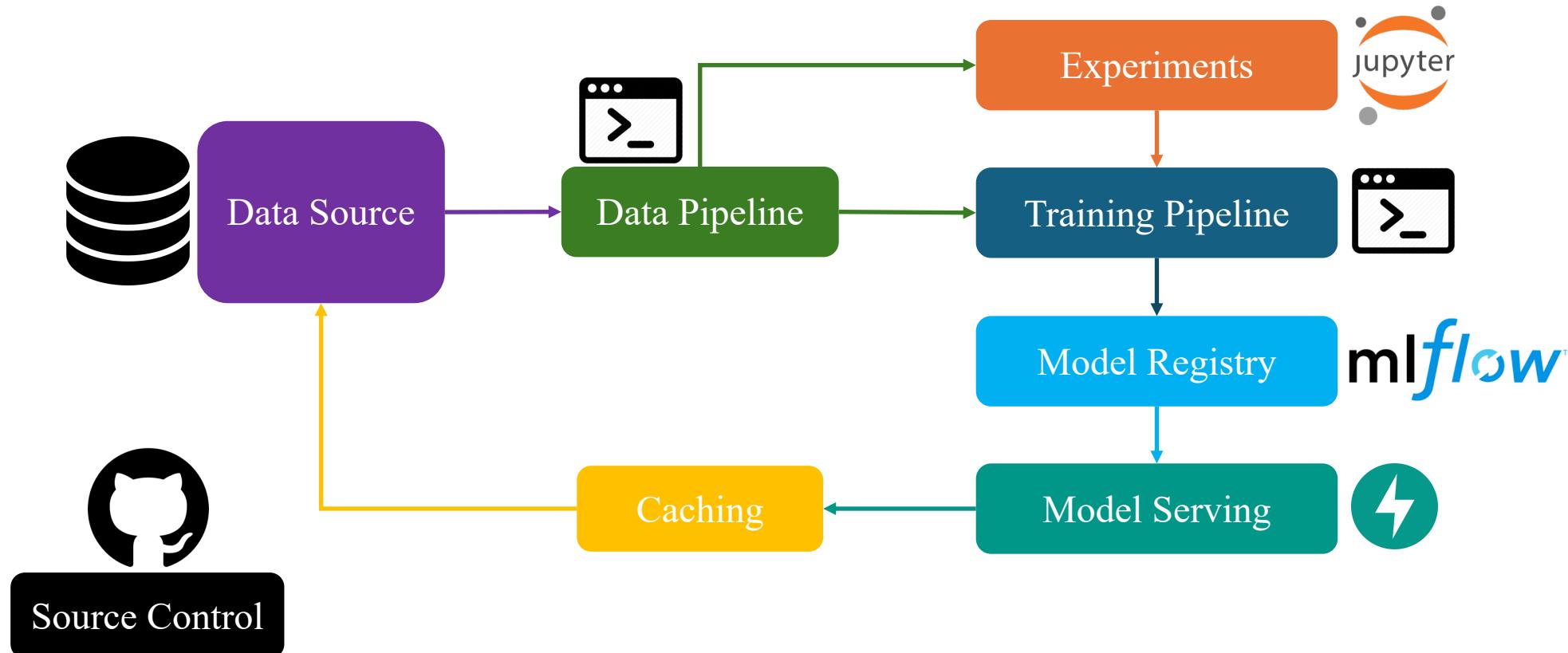
# Model Deployment (MLOps)

## ❖ Machine Learning Operations (MLOps) pipeline



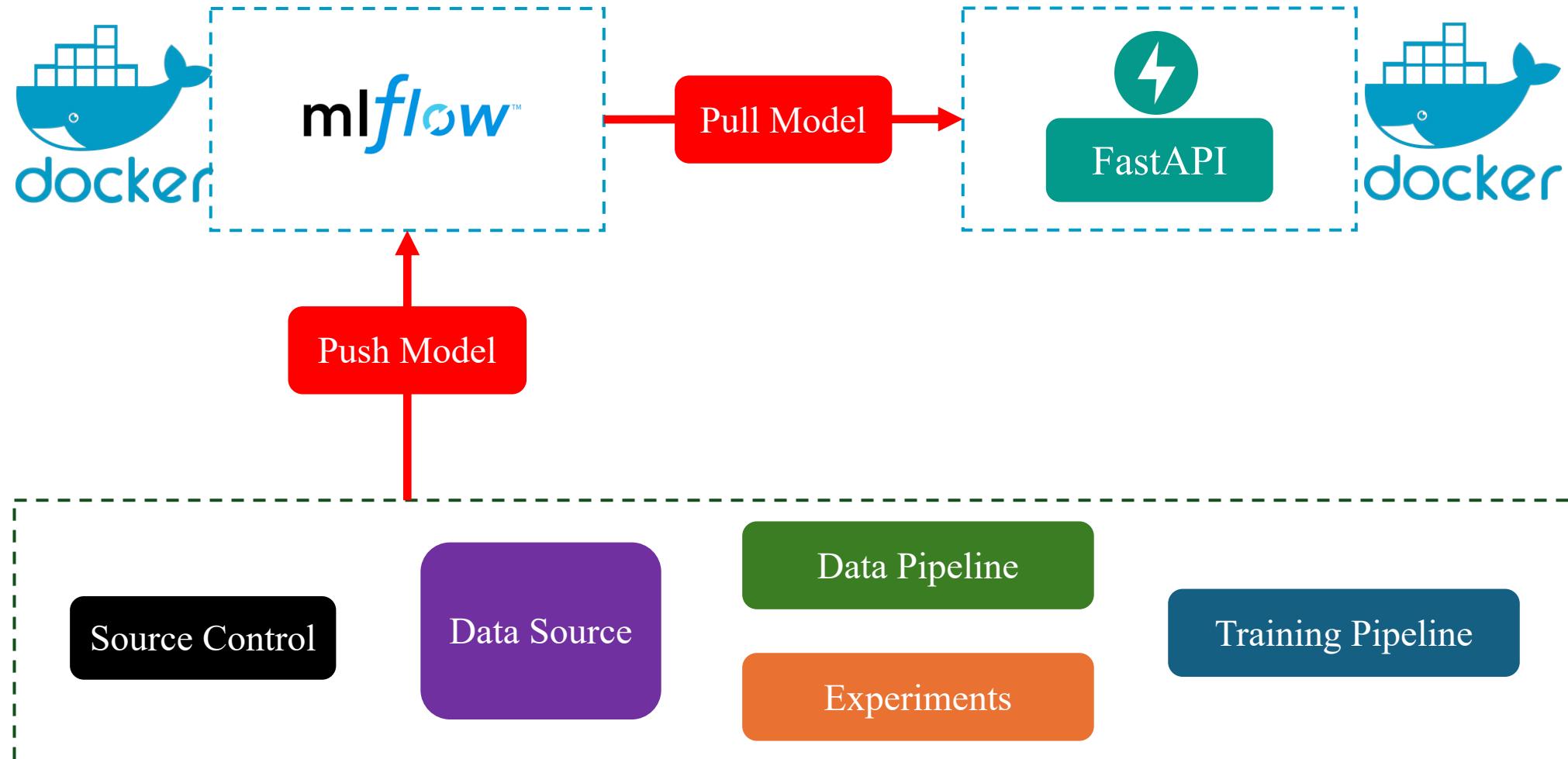
# Model Deployment (MLOps)

## ❖ MLOps simple pipeline



# Model Deployment (MLOps)

## ❖ Running steps



# Model Deployment (MLOps)

❖ Source code: <https://github.com/ThuanNaN/mlops-simple-pipeline.git>

The screenshot shows the GitHub repository page for 'mlops-simple-pipeline'. The repository is public and has 72 commits. The README file contains the following content:

```
Machine Learning Operations (MLOps) Simple Pipeline

1. Tools
  - Model tracking: MLFlow - https://mlflow.org
  - Containerization: Docker/Docker compose

2. Backend API
  - FastAPI - https://fastapi.tiangolo.com
  - Unicorn - https://www.unicorn.org

3. Frontend Interface
  - Gradio - https://www.gradio.app
```

The repository stats on the right show Python as the primary language (96.9%), followed by Makefile (2.1%) and Dockerfile (1.0%).

# Model Deployment (MLOps)

## ❖ Start MLflow server

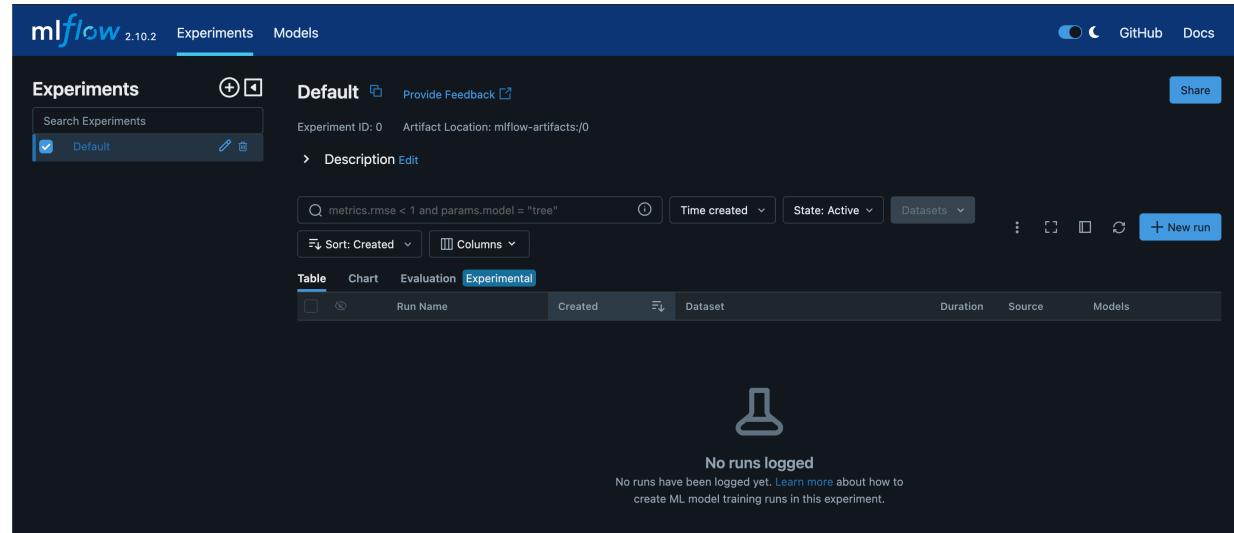
Makefile command

make mlflow\_up

Use MLflow for tracking  
and versioning the  
trained model.

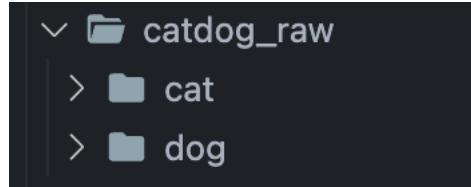
localhost:8000

```
[mlops_pipeline] thuannd@thuan-ubuntu:/mnt/HDD/Repositories/MLOps_simple_pipeline/backend$ make mlflow_up
docker compose -f ./docker/mlflow/docker-compose.yml up -d
[!] Building 54.2s (7/7) FINISHED
=> [mlflow internal] load build definition from Dockerfile
=> [mlflow internal] transferring dockerfile: 123B
=> [mlflow internal] load metadata for docker.io/library/python:3.11-slim
=> [mlflow internal] load .dockerrignore
=> [mlflow internal] transferring context: 64B
=> [mlflow 1/3] FROM docker.io/library/python:3.11-slim@sha256:90f8795536170fd08236d2ceb74f738d8b84bfbf263656654dc9b
=> => resolve docker.io/library/python:3.11-slim@sha256:90f8795536170fd08236d2ceb74f738d8b84bfbf263656654dc9b
=> => sha256:346e2b922dbd8f853cd1a63f142299fc7449b0a59b0e2b600ef7dc98ca5ab436 1.37kB / 1.37kB
=> => sha256:23bae5f8ae9607056414daaa84966ce8bb5663c370204a0ef0596ed81ef03 6.93kB / 6.93kB
=> => sha256:8a1e25ce7c4f75e372e9884fb8f7b1bedcfea47a7d452eb4b9a1c7477c9a00345 29.12MB / 29.12MB
=> => sha256:1103112ebfc46e01c0f35f3586e5a39c6a9ffa32c1a362d4d5f20e3783c6fd7 3.51MB / 3.51MB
=> => sha256:b4b80ef7128dc9bd114e4f7ab83bd62287fd091494c12ab793563211fcc32e84 12.87MB / 12.87MB
=> => sha256:90f8795536170fd08236d2ceb74f74738d8b84bfbf263656654dc9b 1.65kB / 1.65kB
=> => extracting sha256:8a1e25ce7c4f75e372e9884fb8f7b1bedcfea47a7d452eb4b9a1c7477c9a00345
=> => sha256:cc7f04ac52f8a3bad5b4d1f6179041d3ccca1967aaece3faa38303c99f9ed7e0c 243B / 243B
=> => sha256:87b8bf94a2ace29c44331381ce313388 3.41MB / 3.41MB
=> => extracting sha256:c7f04ac52f8a3bad5b4d1f6179041d3ccca1967aaece3faa38303c99f9ed7e0c
=> => extracting sha256:b4b80ef7128dc9bd114e4f7ab83bd62287fd091494c12ab793563211fcc32e84
=> => extracting sha256:c7f04ac52f8a3bad5b4d1f6179041d3ccca1967aaece3faa38303c99f9ed7e0c
=> => extracting sha256:87b8bf94a2ace2b005da41fb268f80a63c02887d1aeeb29c44331381ce313388
=> [mlflow 2/3] WORKDIR /mlflow/
=> [mlflow 3/3] RUN pip install --no-cache-dir mlflow==2.10.2
=> [mlflow] exporting to image
=> => writing image sha256:3ab775d300d28ad4660092a8eec2515d69526879fb58f66be2dedf4da7d5620
=> => naming to docker.io/library/mlflow-mlflow
[!] Running 1/2
  Network mlflow_default Created
  Container mlflow_server Started
```



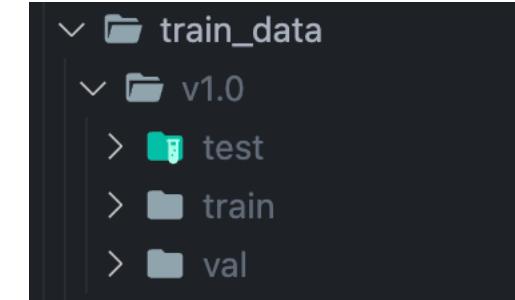
# Model Deployment (MLOps)

## ❖ Prepare dataset for training



Bash command

```
python3 src/data_processing.py --version v1.0
```



Split data into train/val/test for training and versioning based on folder name.

# Model Deployment (MLOps)

## ❖ Training model (resnet\_18)

Bash command

```
python3 src/model_training.py --data_version  
v1.0 --model_name resnet_18 --device cpu
```

The screenshot shows the mlflow UI interface. At the top, there are tabs for 'Experiments' (which is selected) and 'Models'. Below the tabs, the 'Experiments' section has a search bar and two entries: 'Default' and 'Image\_Classification'. The 'Image\_Classification' entry is checked. To the right, the 'Image\_Classification' experiment details are shown, including its ID, artifact location, and a description edit link. A search bar at the bottom allows filtering by metrics like 'metrics.rmse < 1' and parameters like 'params.model = "tree"'. The main table displays runs, with one row for 'resnet\_18\_v1.0' created 56 seconds ago.

Run Name	Created	Dataset
resnet_18_v1.0	56 seconds ago	-

localhost:8000

Training model with dataset version 1.0. The model trained will be saved to Mlflow server.

# Model Deployment (MLOps)

## ❖ Registry model trained

Bash command

```
python3 src/model_registry.py --best_metric  
best_val_loss --model_alias Production --  
config_name raw_data
```

The screenshot shows the mlflow UI with the 'Models' tab selected. The page title is 'Registered Models'. There is a search bar with placeholder text 'Filter registered models by name or tags' and icons for help and search. A table lists one registered model:

Name	Latest version	Aliased versions
resnet_18	Version 1	@ Production: Version 1

localhost:8000

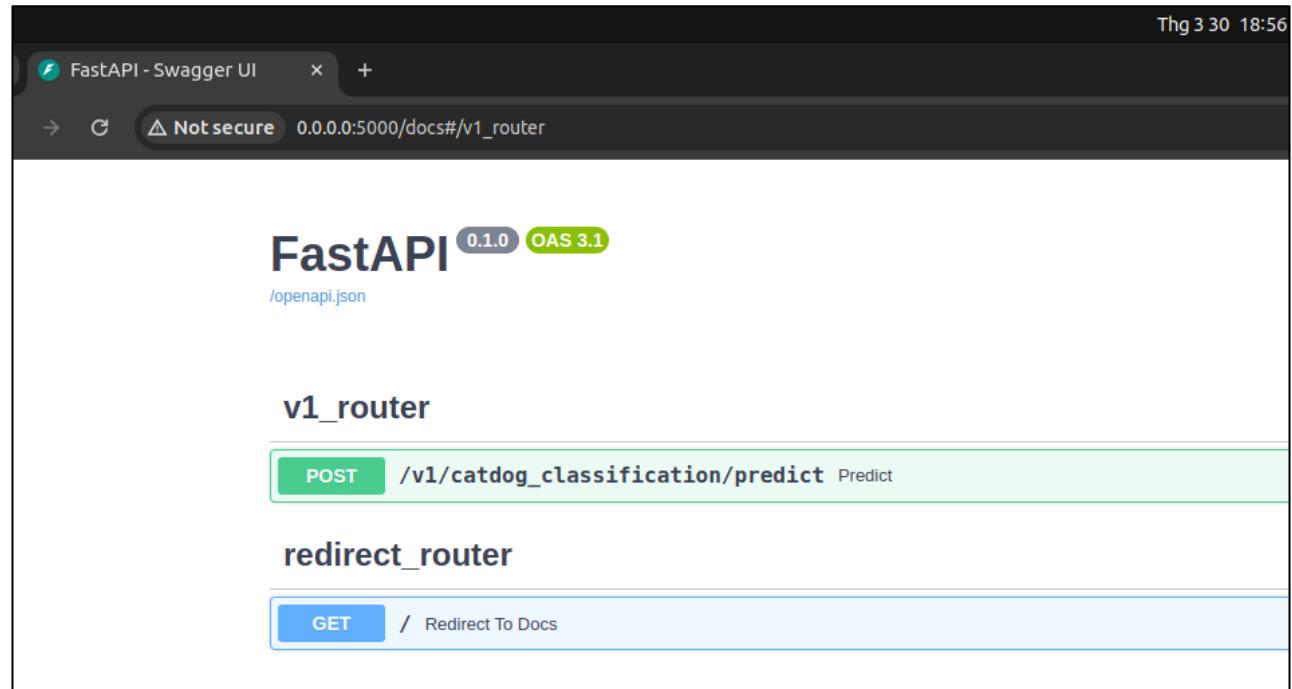
Registry model trained which have highest metric to serving.

# Model Deployment (MLOps)

## ❖ Serving best model

Makefile command

```
make model_name=resnet_18  
model_alias=Production port=5000 serving_up
```



localhost:5000

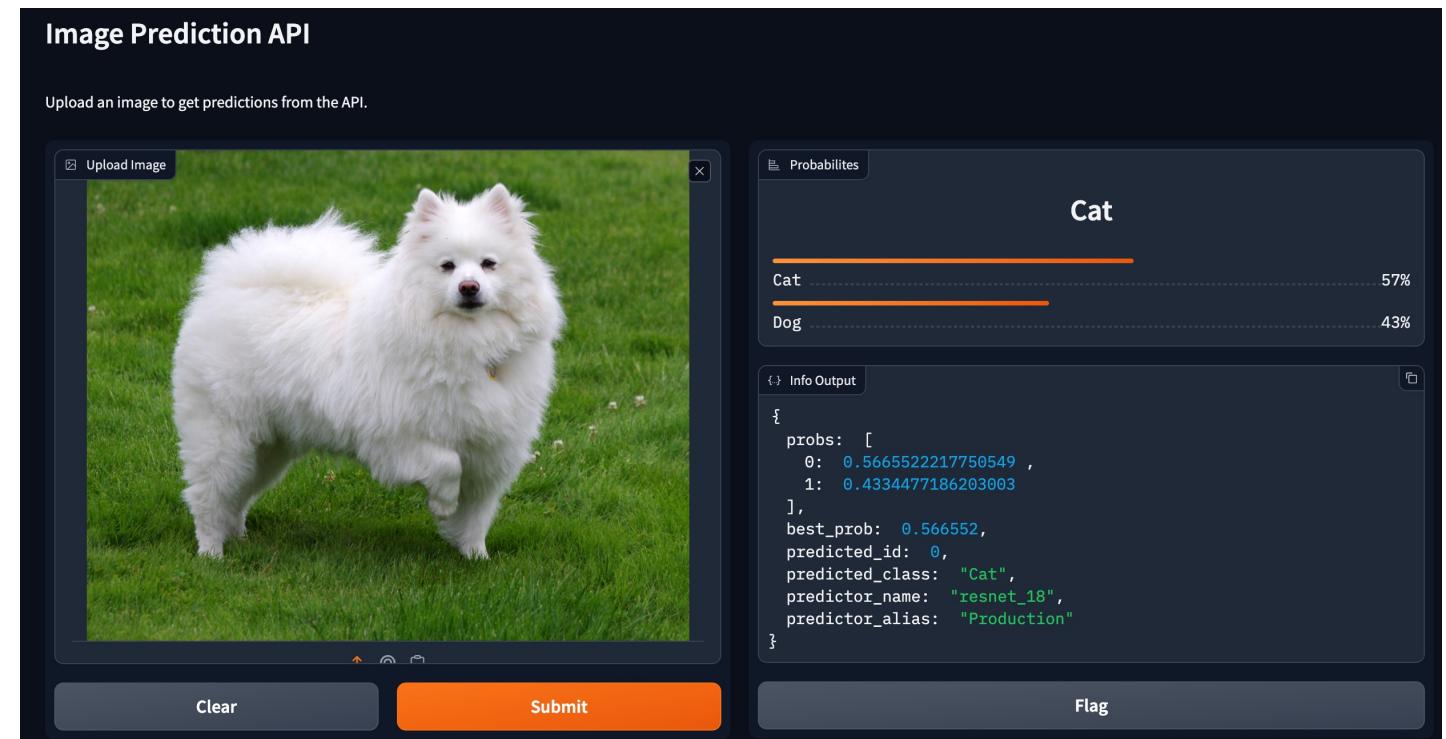
Pulling the best model from MLflow and serving at port 5000 by FastAPI (take a minute for building docker container).

# Model Deployment (MLOps)

## ❖ Test in web application

Bash command

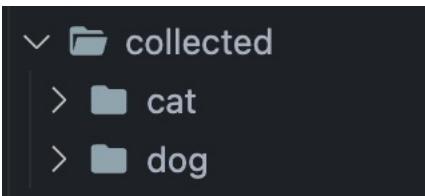
```
cd /frontend  
pip3 install -r requirements.txt  
python app.py
```



localhost:3000

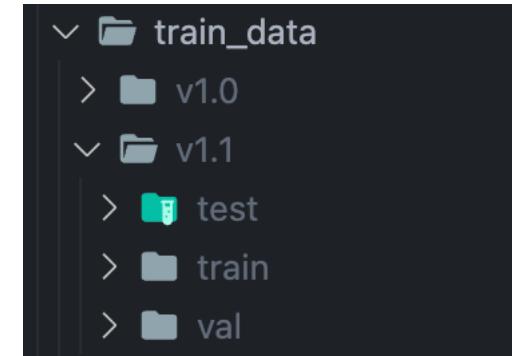
# Model Deployment (MLOps)

## ❖ Collect more data to improve performance



Bash command

```
python3 src/data_processing.py --  
merge_collected --version v1.1
```



Merge raw data with collected data and split into train/val/test  
and named v1.1 for training new model.

# Model Deployment (MLOps)

## ❖ Training new model with new dataset

Bash command

```
python3 src/model_training.py --data_version  
v1.1 --model_name resnet_18 --device cpu
```

Train resnet\_18 with v1.1 dataset

Image\_Classification [Provide Feedback](#)

Experiment ID: 1 Artifact Location: mlflow-artifacts:/1

> Description [Edit](#)

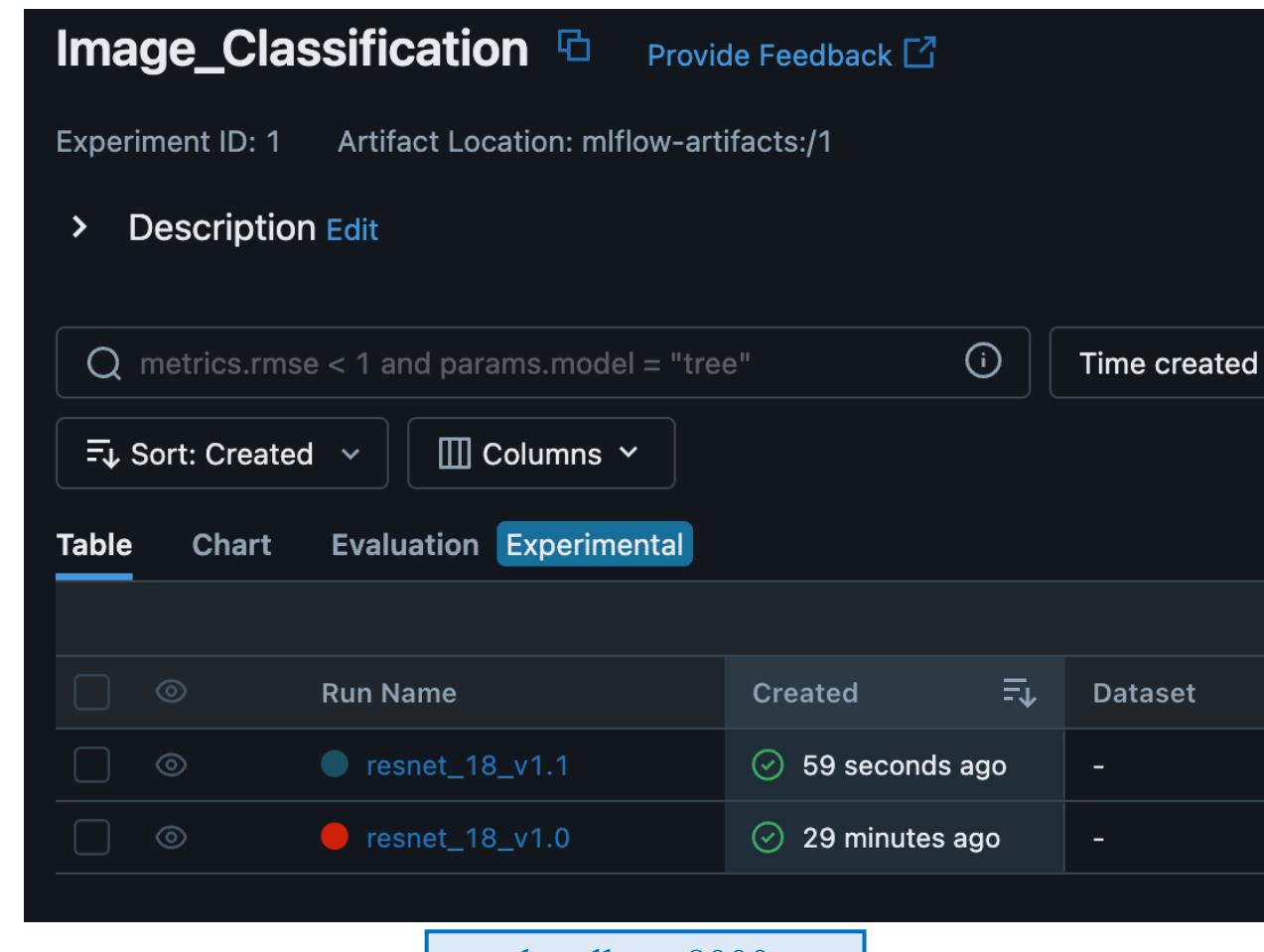
metrics.rmse < 1 and params.model = "tree" [Time created](#)

[Sort: Created](#) [Columns](#)

[Table](#) [Chart](#) [Evaluation](#) [Experimental](#)

Run Name	Created	Dataset
resnet_18_v1.1	59 seconds ago	-
resnet_18_v1.0	29 minutes ago	-

localhost:8000



# Model Deployment (MLOps)

## ❖ Registry model with a specific tag

Bash command

```
python3 src/model_registry.py --filter_string  
"tags.data_version LIKE 'v1.1'" --best_metric  
best_val_loss --model_alias Challenger --  
config_name add_collect
```

The screenshot shows the mlflow UI with the 'Models' tab selected. The title bar indicates 'mlflow 2.10.2'. Below the title bar, there are tabs for 'Experiments' and 'Models'. The main section is titled 'Registered Models'. A search bar at the top right allows filtering by name or tags. The table below lists registered models with columns for 'Name', 'Latest version', and 'Aliased versions'. One entry is visible: 'resnet\_18' with 'Version 2' and an alias '@ Challenger : Version 2 +1'.

Name	Latest version	Aliased versions
resnet_18	Version 2	@ Challenger : Version 2 +1

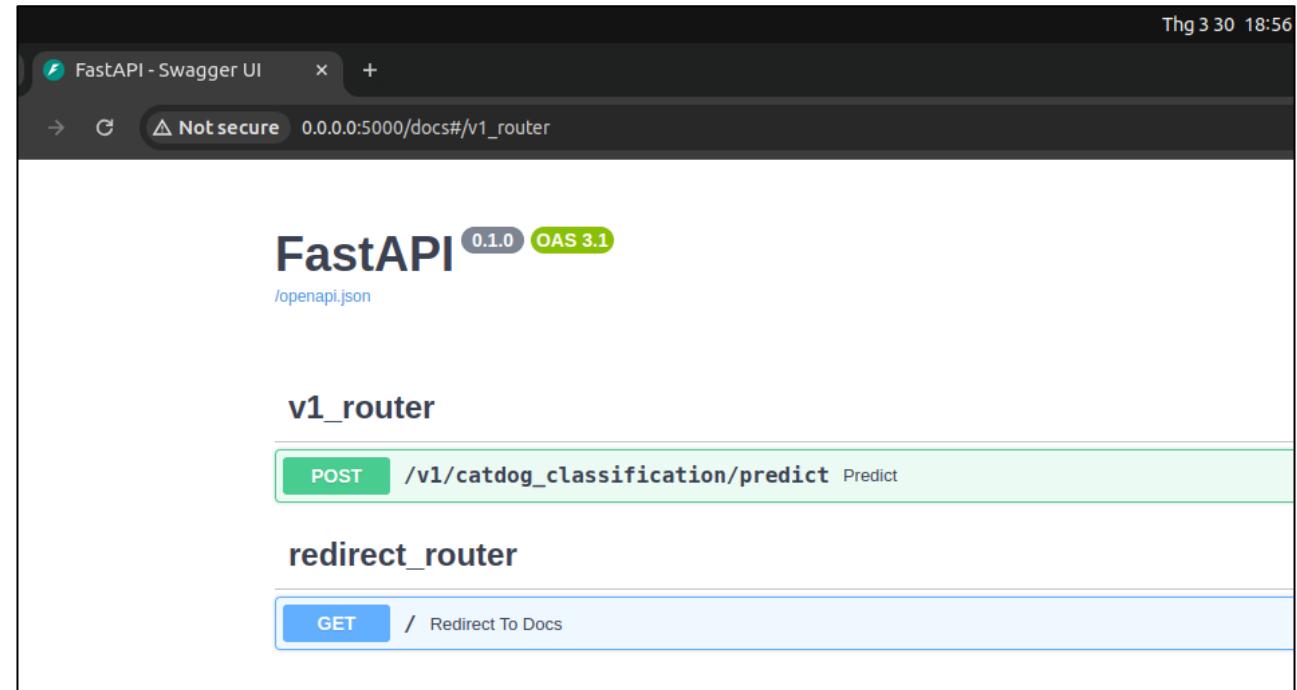
Registry model trained that version of dataset training is v1.1  
and tagging it to “Challenger”.

# Model Deployment (MLOps)

## ❖ Restart FastAPI

Makefile command

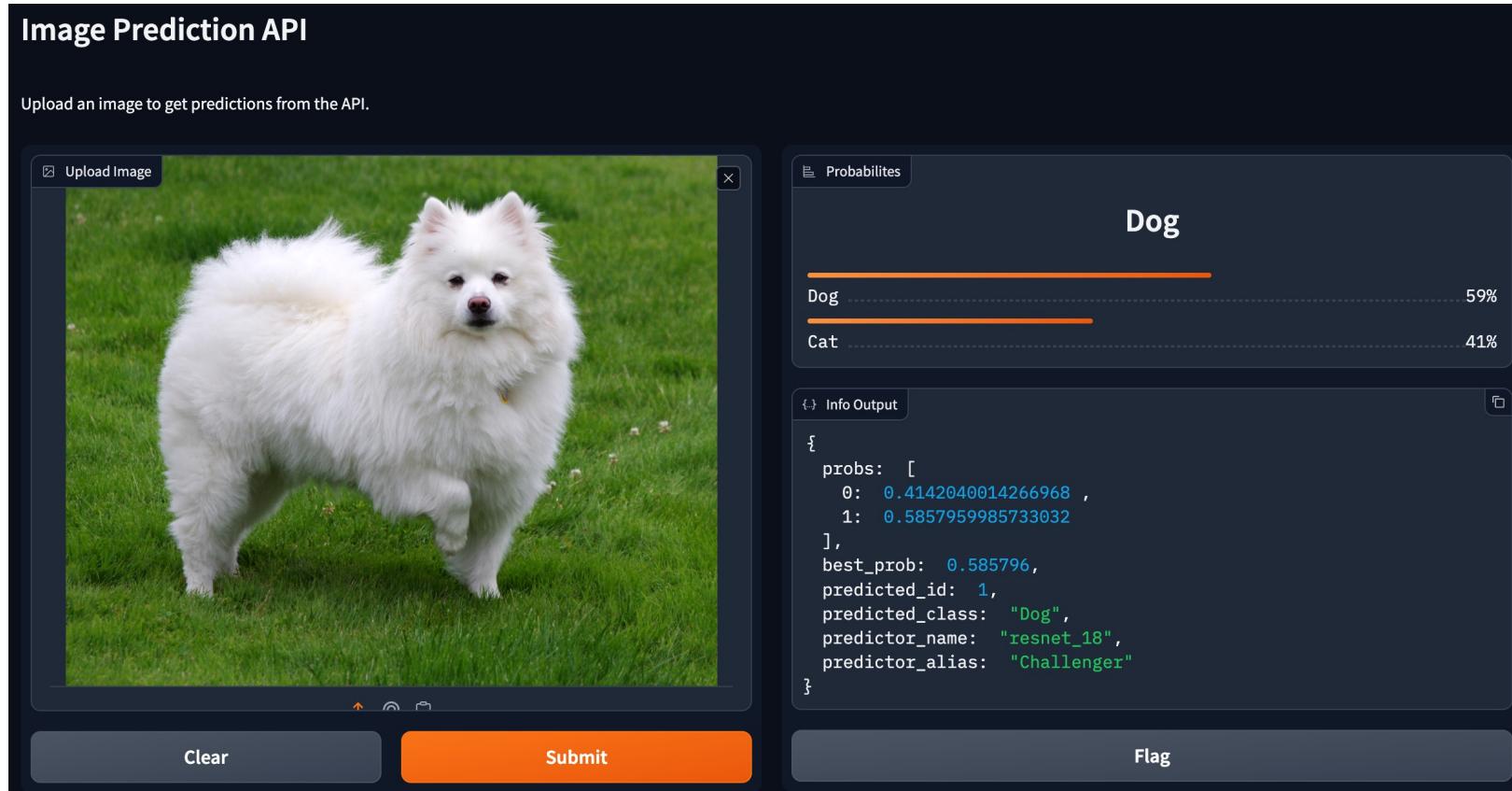
```
- make serving_down  
- make model_name=resnet_18  
model_alias=Challenger port=5000 serving_up
```



Restart serving container to load the new model with specific tag is “Challenger” for testing the performance of new model.

# Model Deployment (MLOps)

- ❖ Refresh the web app and predict again

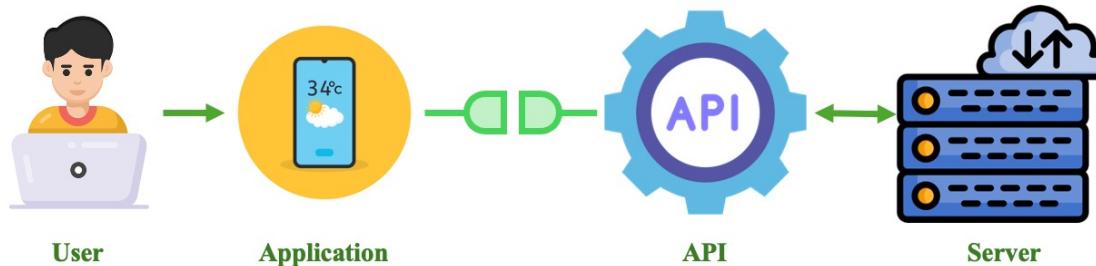


localhost:3000

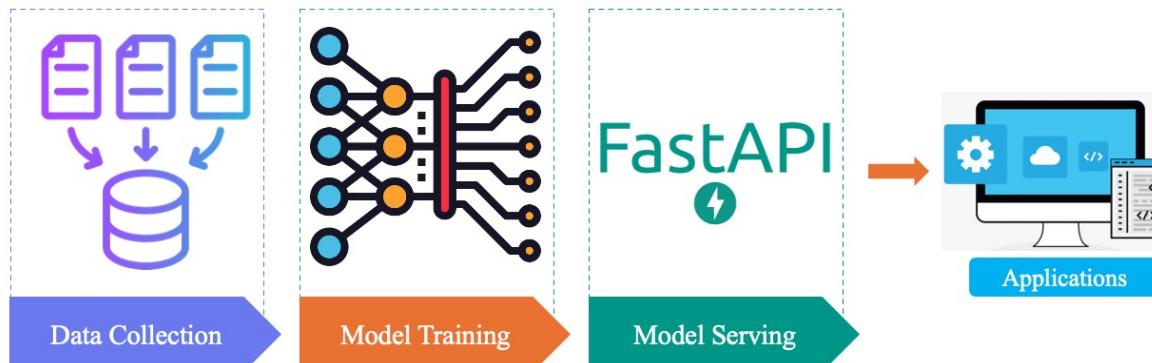
# Summarization

# Summarization

## ❖ What we have learned so far



## FastAPI



### Summarization:

- Discuss about the definition of API.
- Learn the basis of FastAPI:
  - Path Operations.
  - Pydantic Model.
  - Response Code.
  - Middleware.
- Learn how to deploy a Deep Learning model as an API service with FastAPI.
  - Deploy a Cat Dog Classification model.
  - Simple MLOps pipeline.

# Question

