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

Deploy Deep Learning models with FastAPI

Extra Class

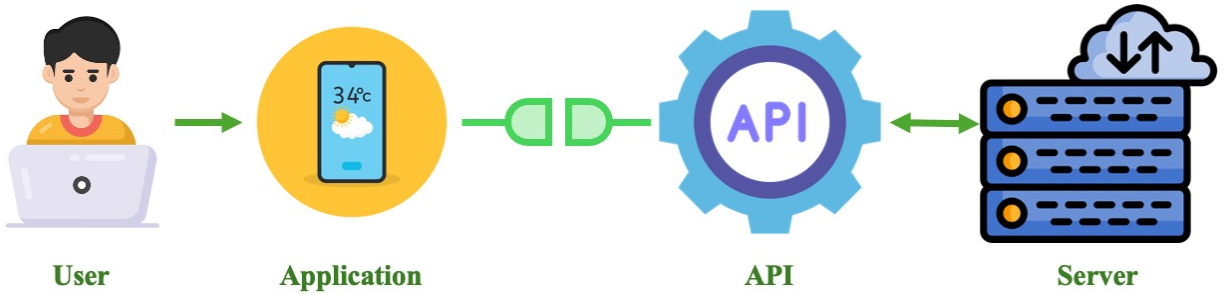
Dinh-Thang Duong – TA

Nguyen-Thuan Duong – STA

Year 2024

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.

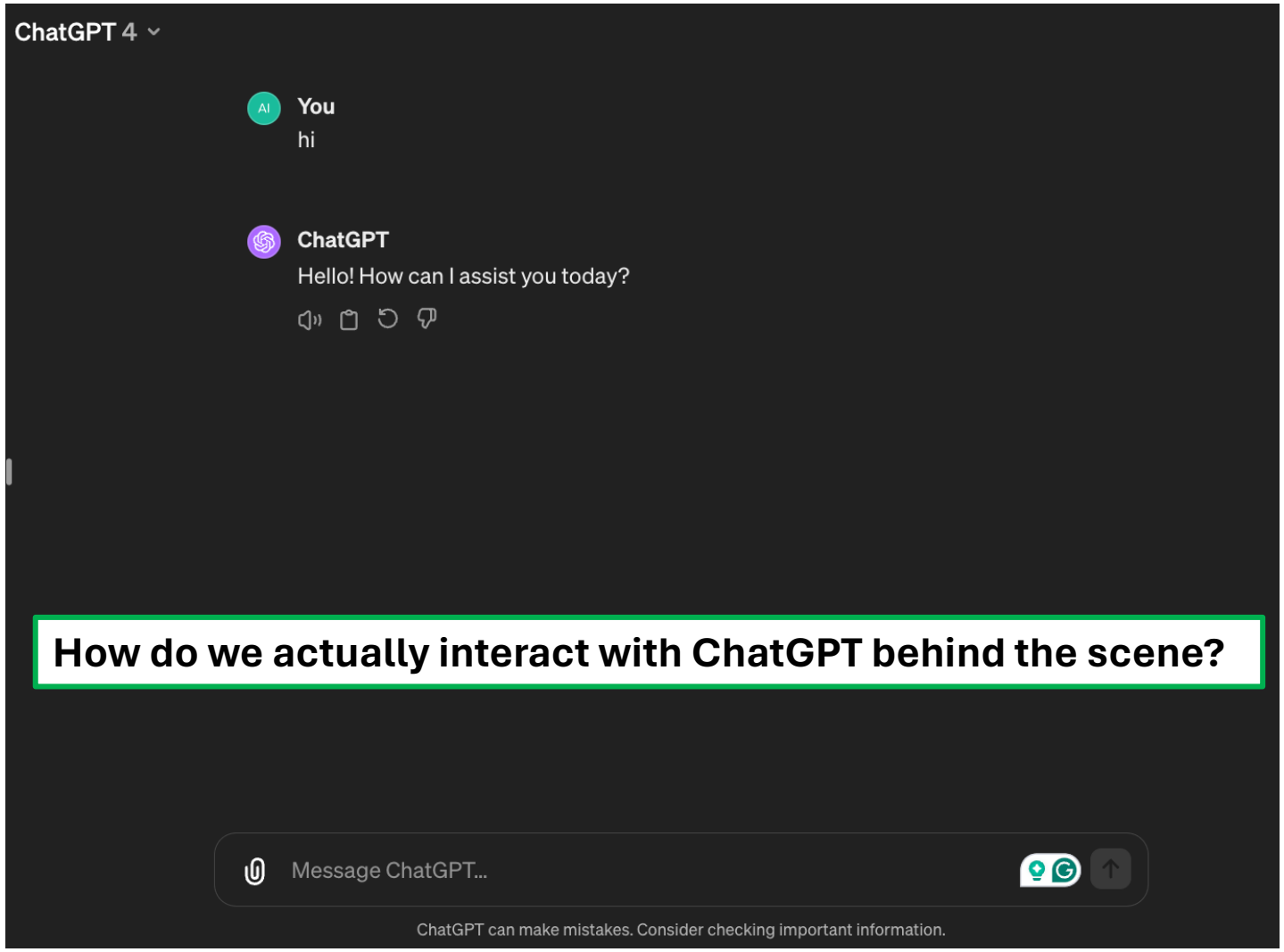
Outline

- Introduction
- API
- FastAPI
- Model Deployment
- Question

Introduction

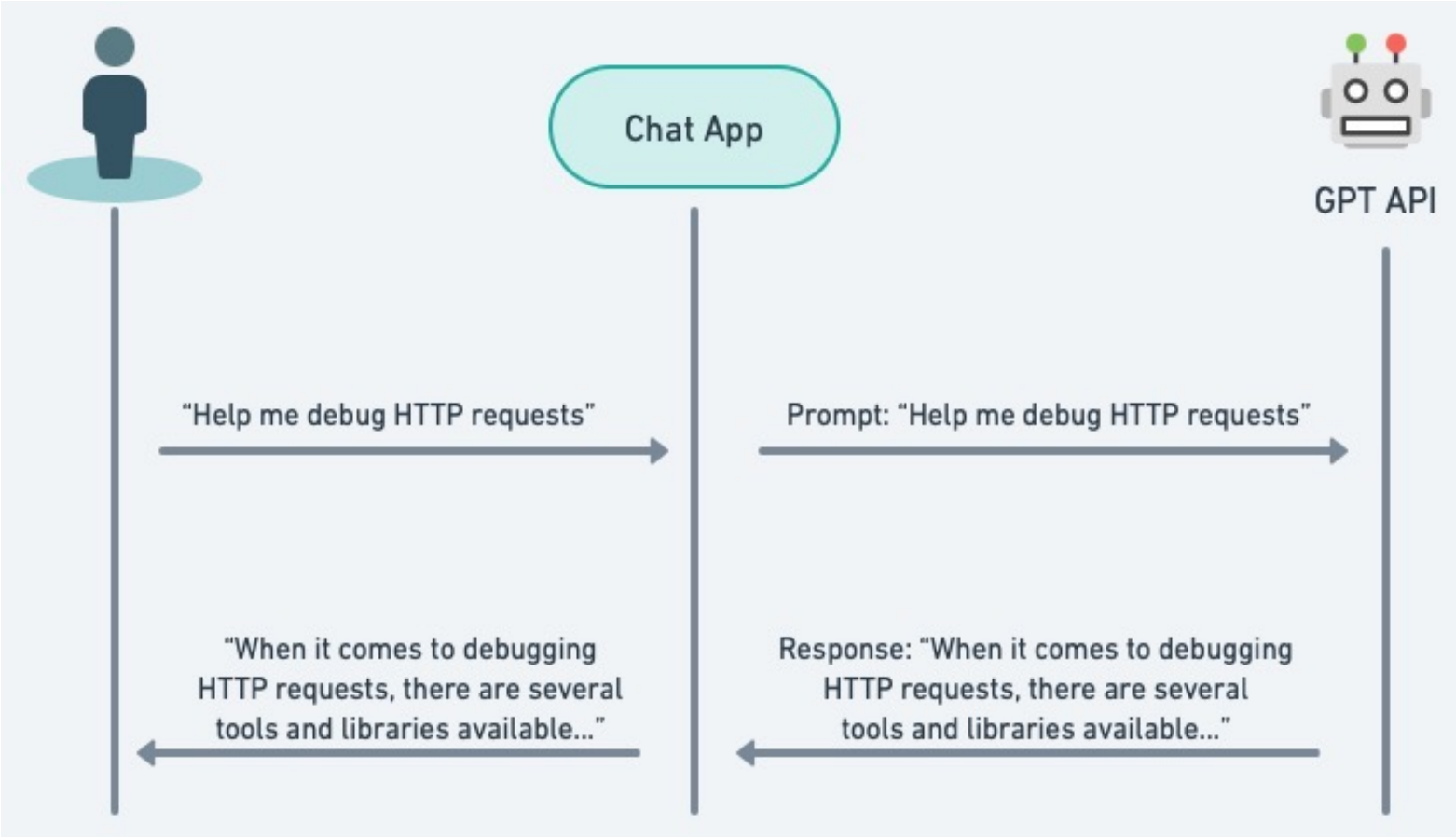
Introduction

❖ Getting Started



Introduction

❖ Getting Started



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

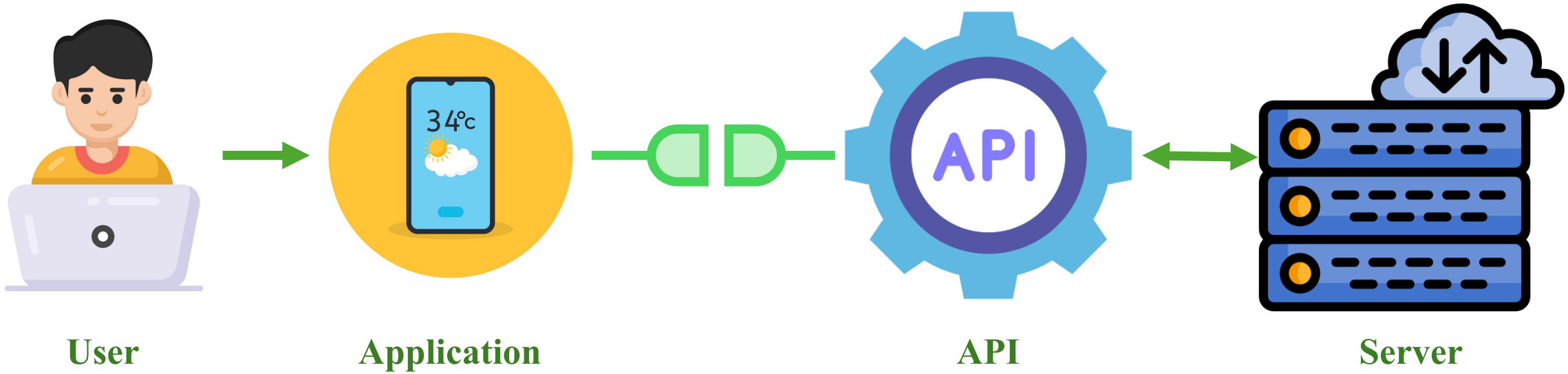
https://semaphoreci.com/blog/function-calling

API

API

❖ Getting Started

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



API

❖ Getting Started

Make the order

Delivery of order

Take the order

Bringing from kitchen

Customer

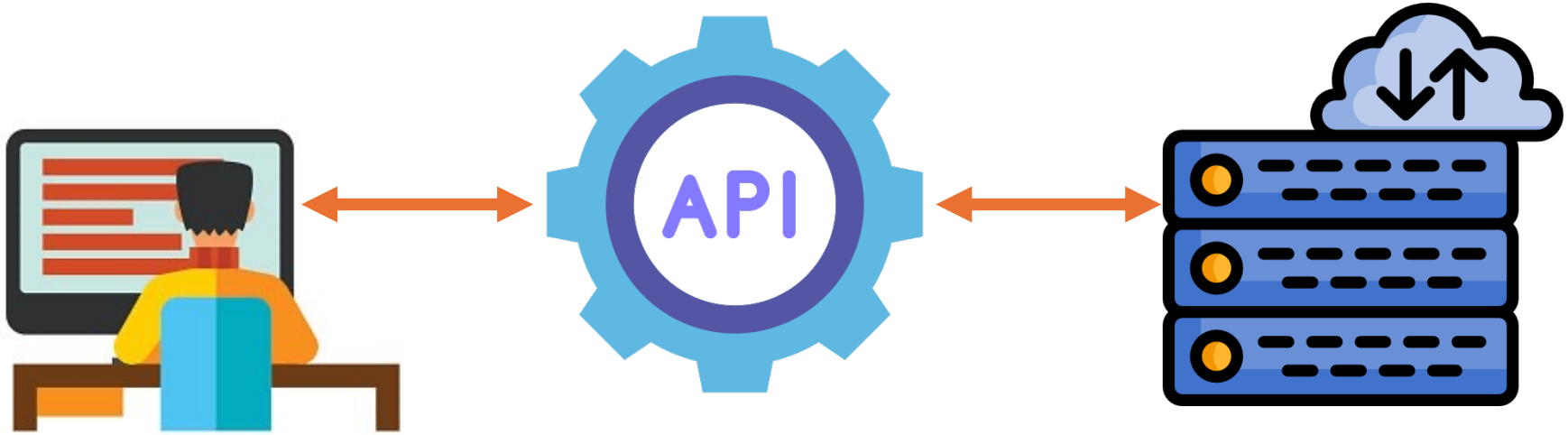
Waiter

Chef

API

❖ Getting Started

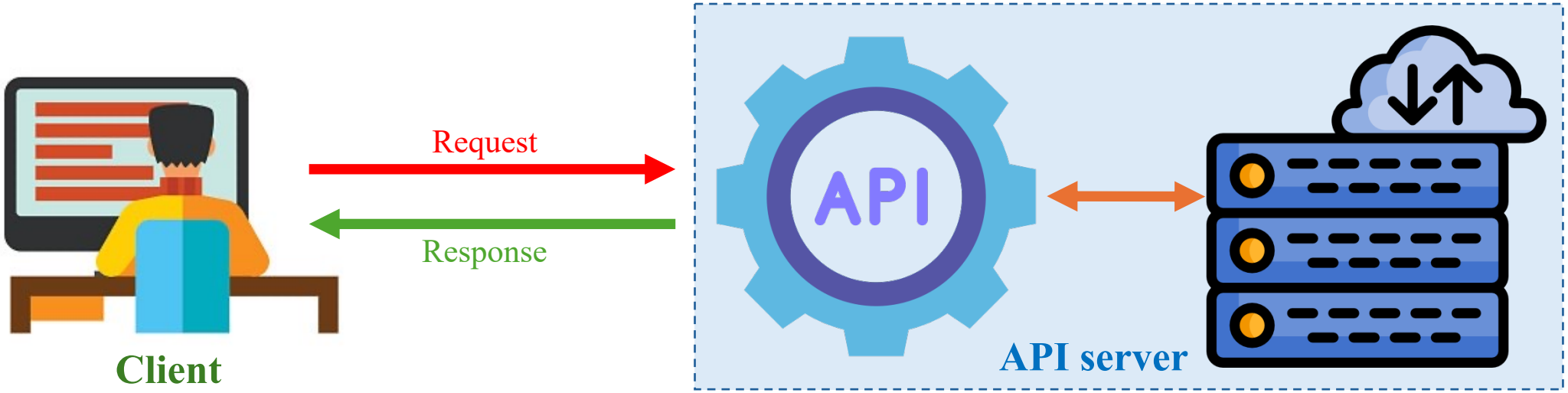
- The application sending the request is called the client, and the application sending the response is called the server.



API

❖ 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.

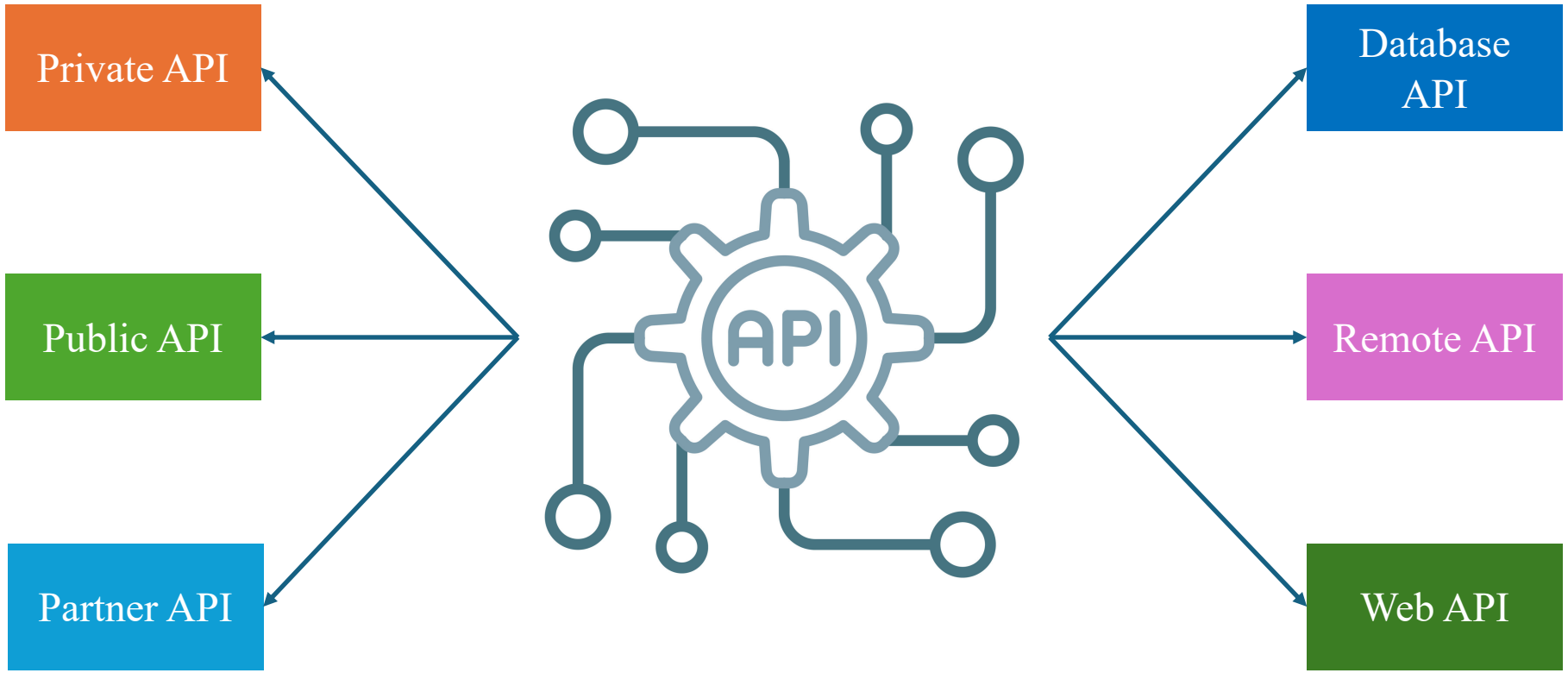


- **The API server** receives the request and processes it. The API server sends a response back to the client, which may include data, an error message, or a status code indicating the result of the operation.

- **The client** receives the response and processes it.

API

❖ Types of APIs

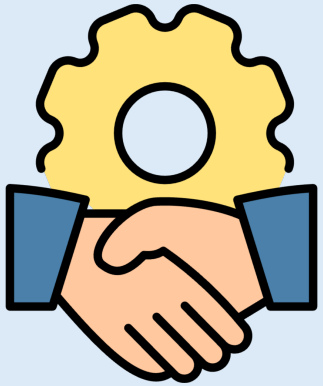


API

❖ 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

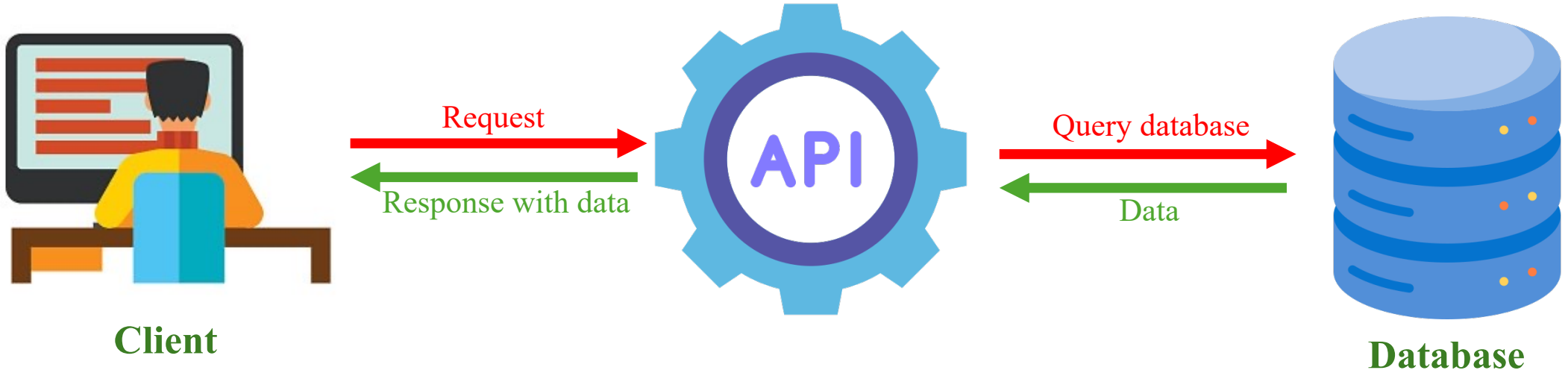
APIs are exposed only to existing **developers within the enterprise**

App are usually targeted at employees of the enterprise

API

❖ Database API

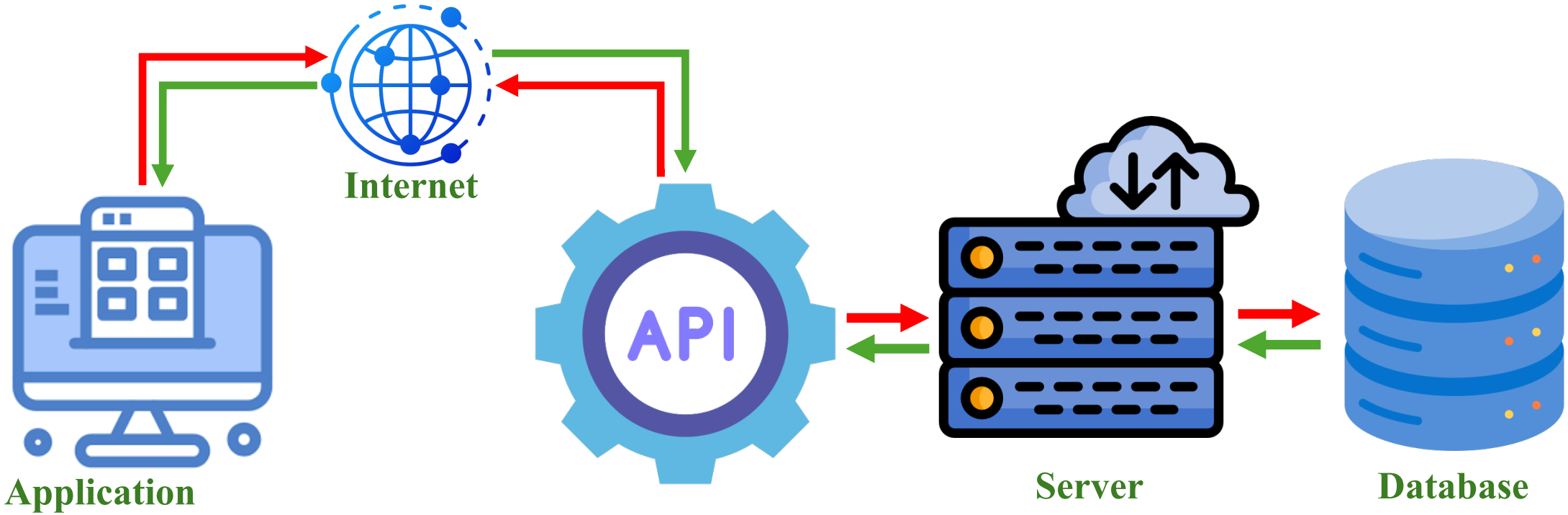
- A database API allows applications to interact with a database to access and manipulate data.



API

❖ Remote API

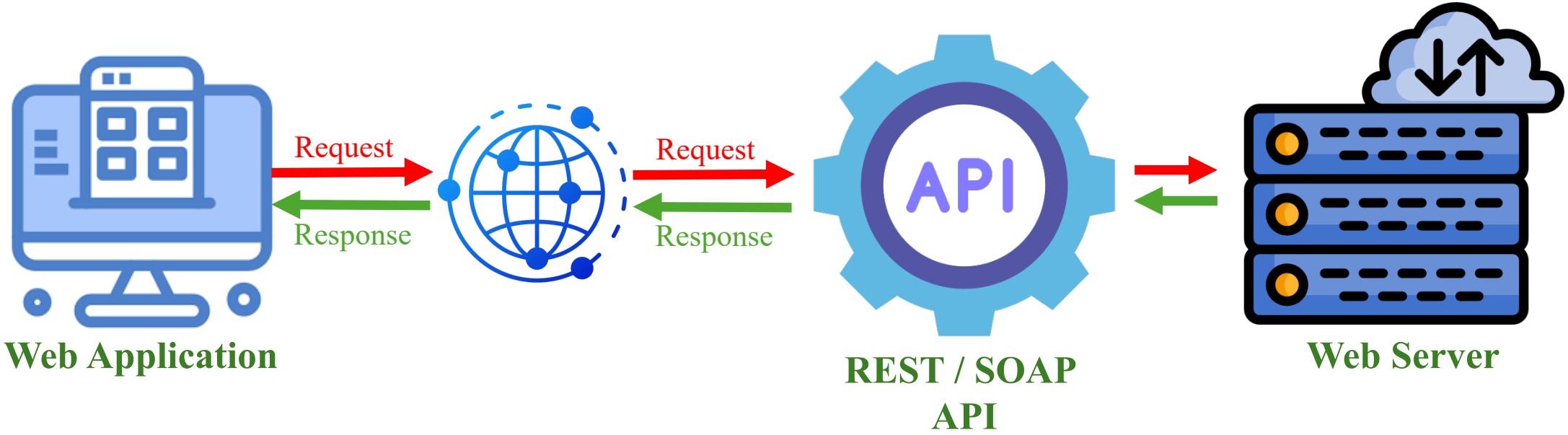
- Using a remote API, two distant applications communicate across a communications network, primarily the internet.



API

❖ Web API

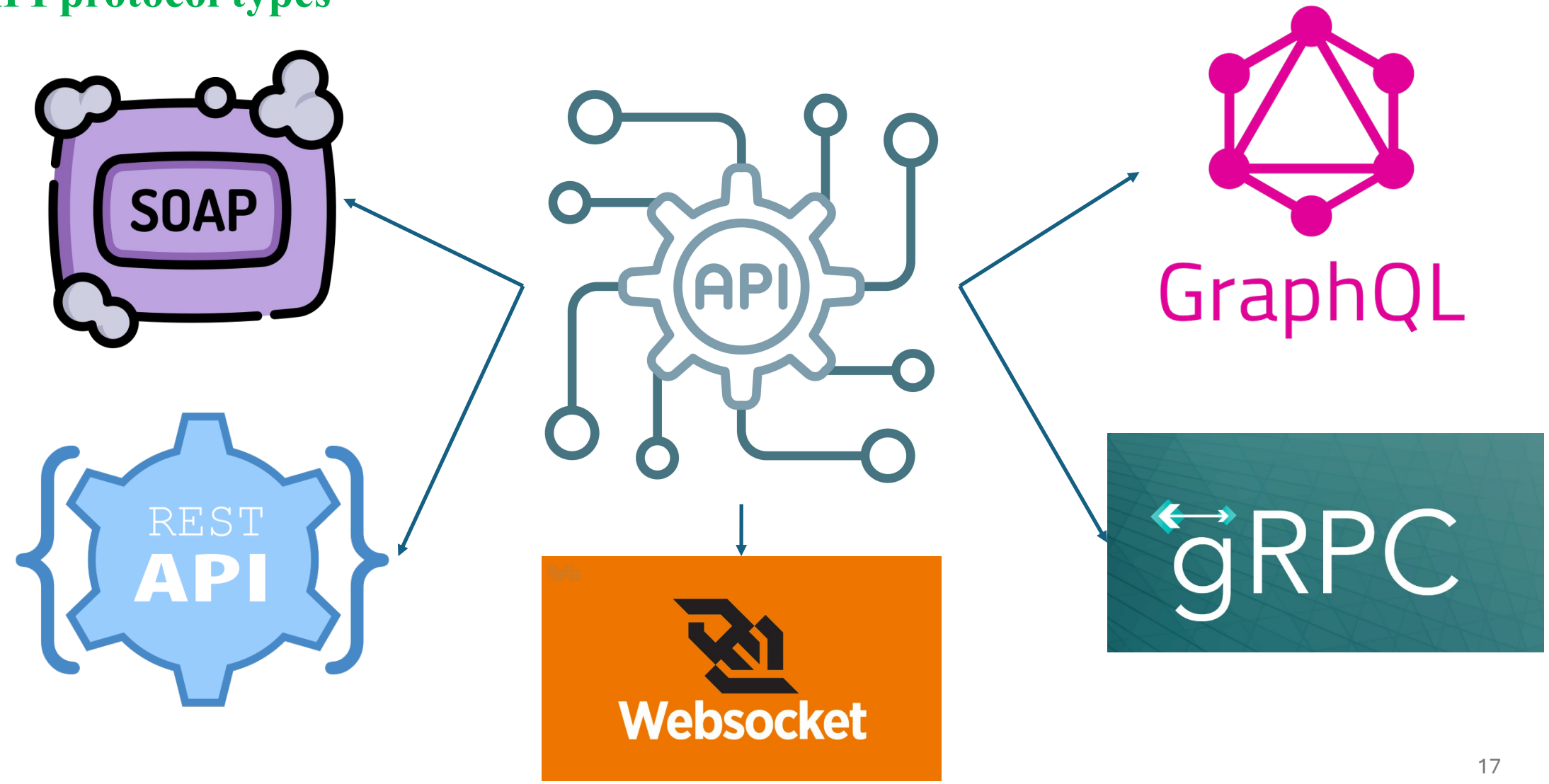
- Web API (or Web Service API) is an application inferface between a web browser and a web server.



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

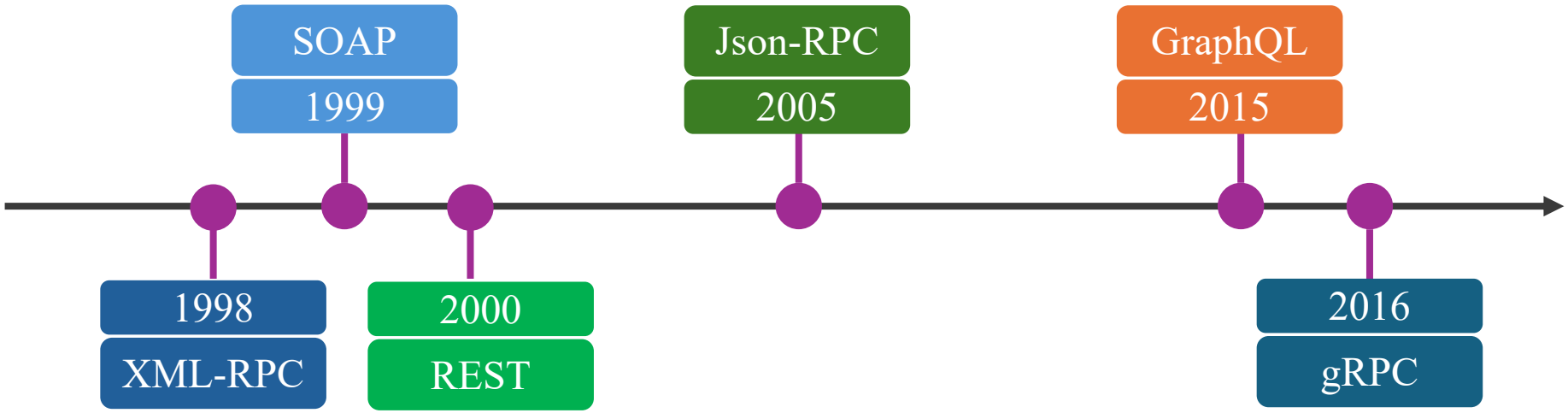
API

❖ API protocol types



API

❖ Timeline



API

❖ Conclusion

	REST	GraphQL	SOAP	RPC
Structure	Follows six architectural constraints	Schema and type	Message structure	Local procedural calls
Format	Json, XLM, HTML, plain text	Json	XML	Json, XML, Flatbuffers, etc
Advantages	Flexible terms of data format and structure	Sovles 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

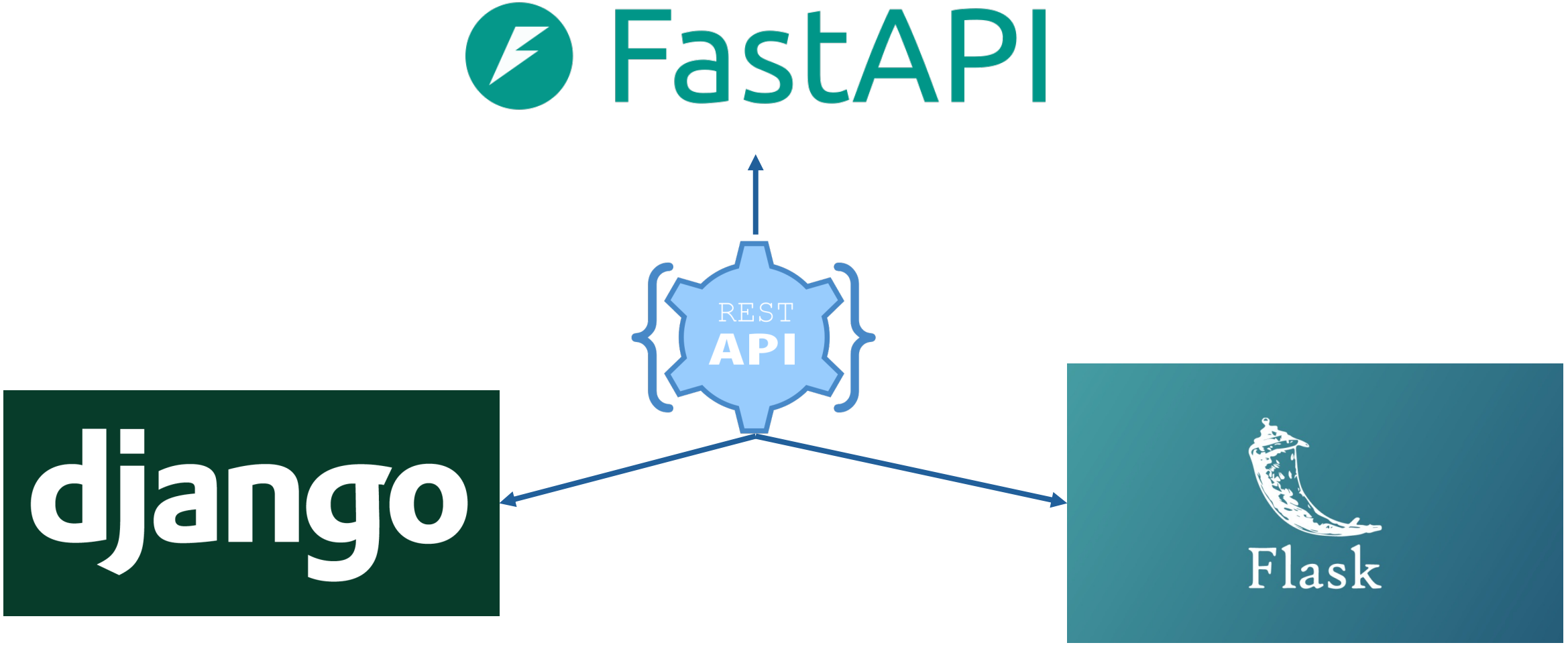
FastAPI

❖ Introduction

FastAPI 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

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

FastAPI

❖ Which companies use FastAPI ?

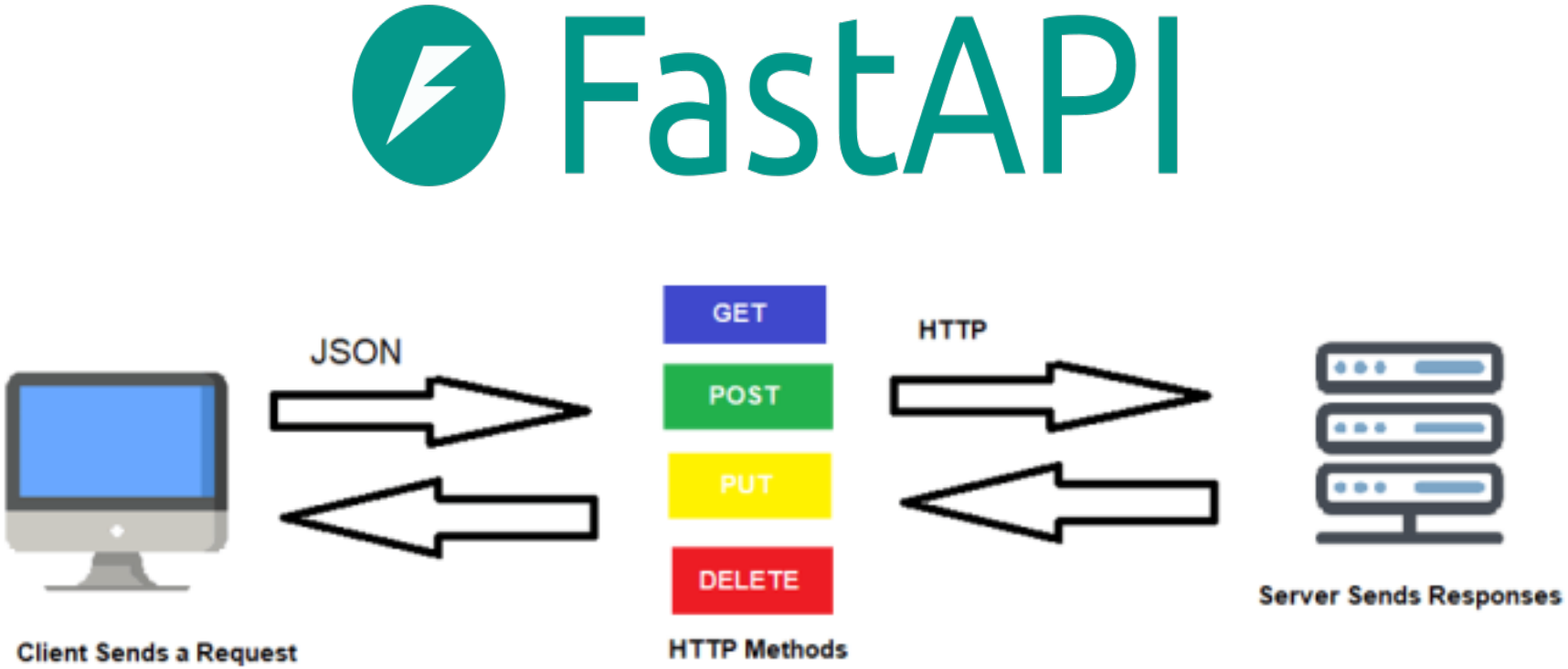
FastAPI

❖ 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

https://www.apisec.ai/blog/rest-api-and-its-significance-to-web-service-providers

FastAPI

❖ Practice: Initialize first API

```python
from fastapi import FastAPI

app = FastAPI()
```

```
(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)
```

1. Create a FastAPI instance

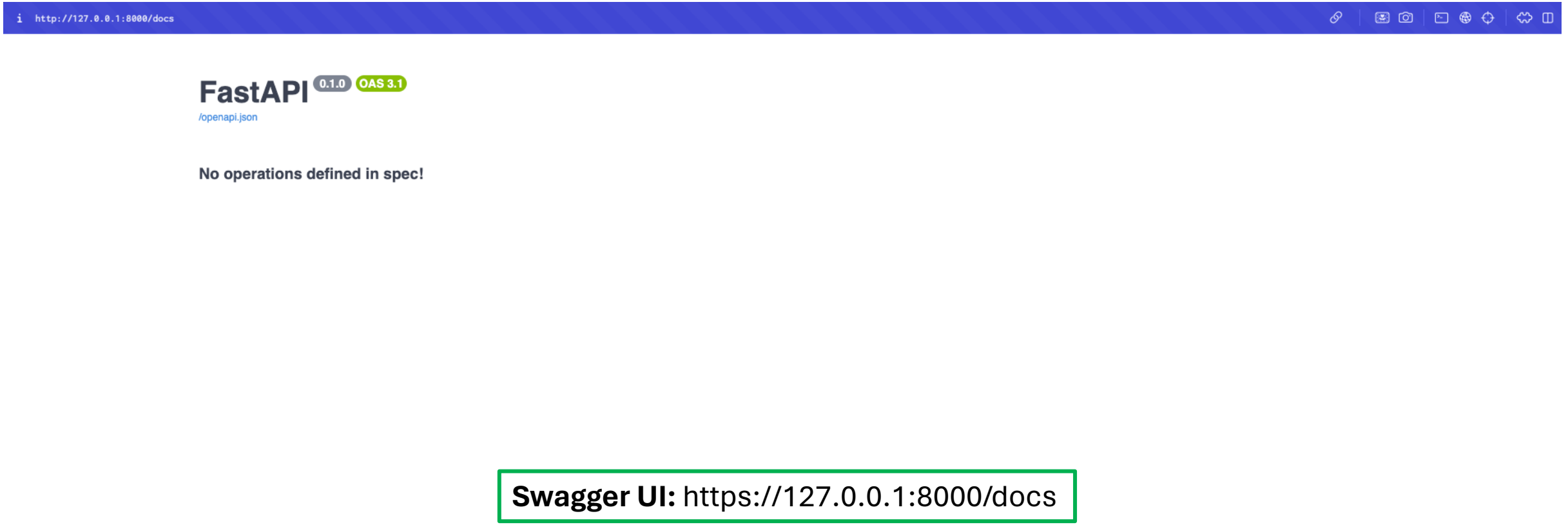
2. Run uvicorn to deploy the API

FastAPI

❖ Practice: Initialize first API

FastAPI

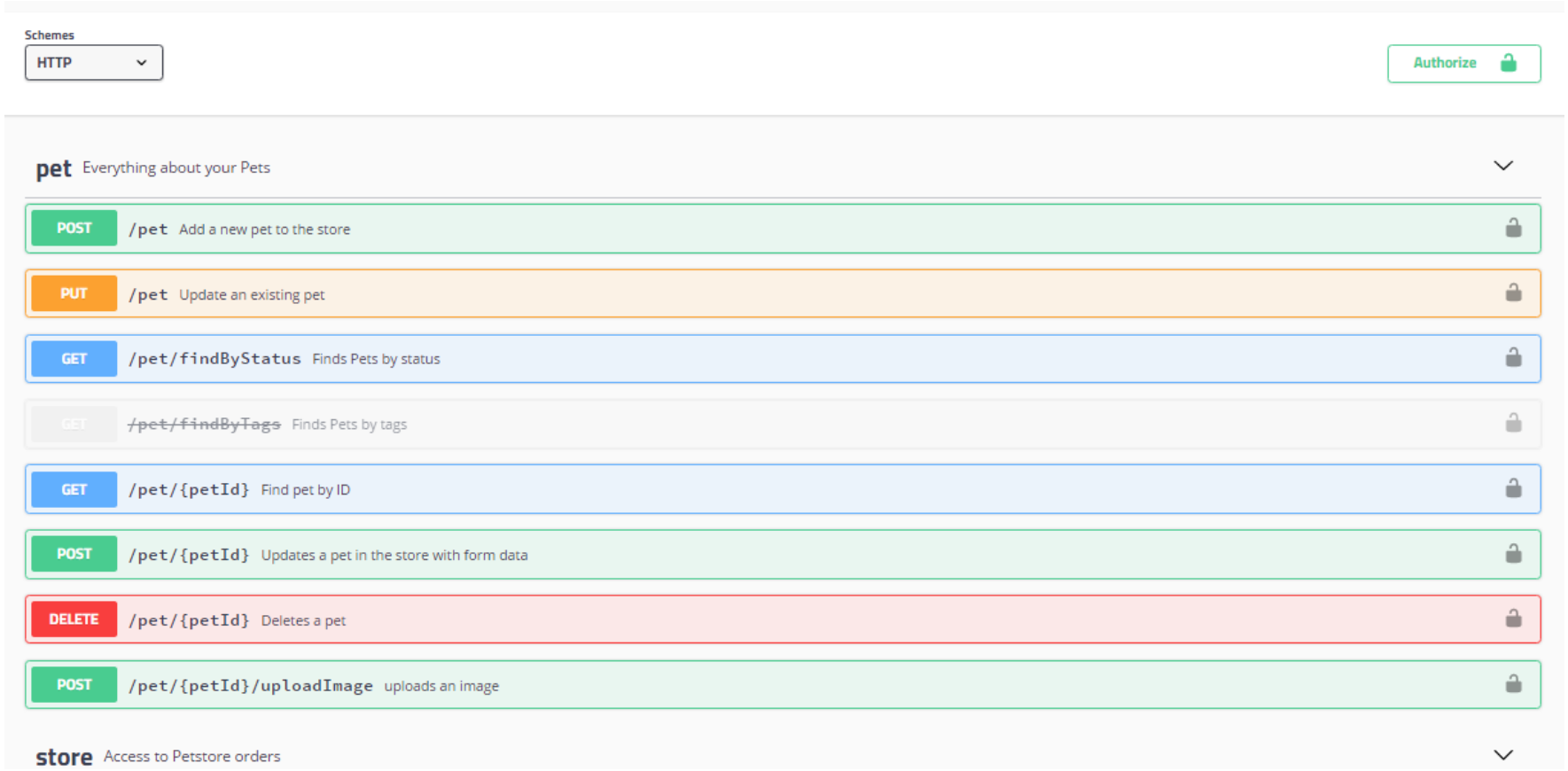
❖ Practice: Initialize first API



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

FastAPI

❖ Practice: Initialize first API

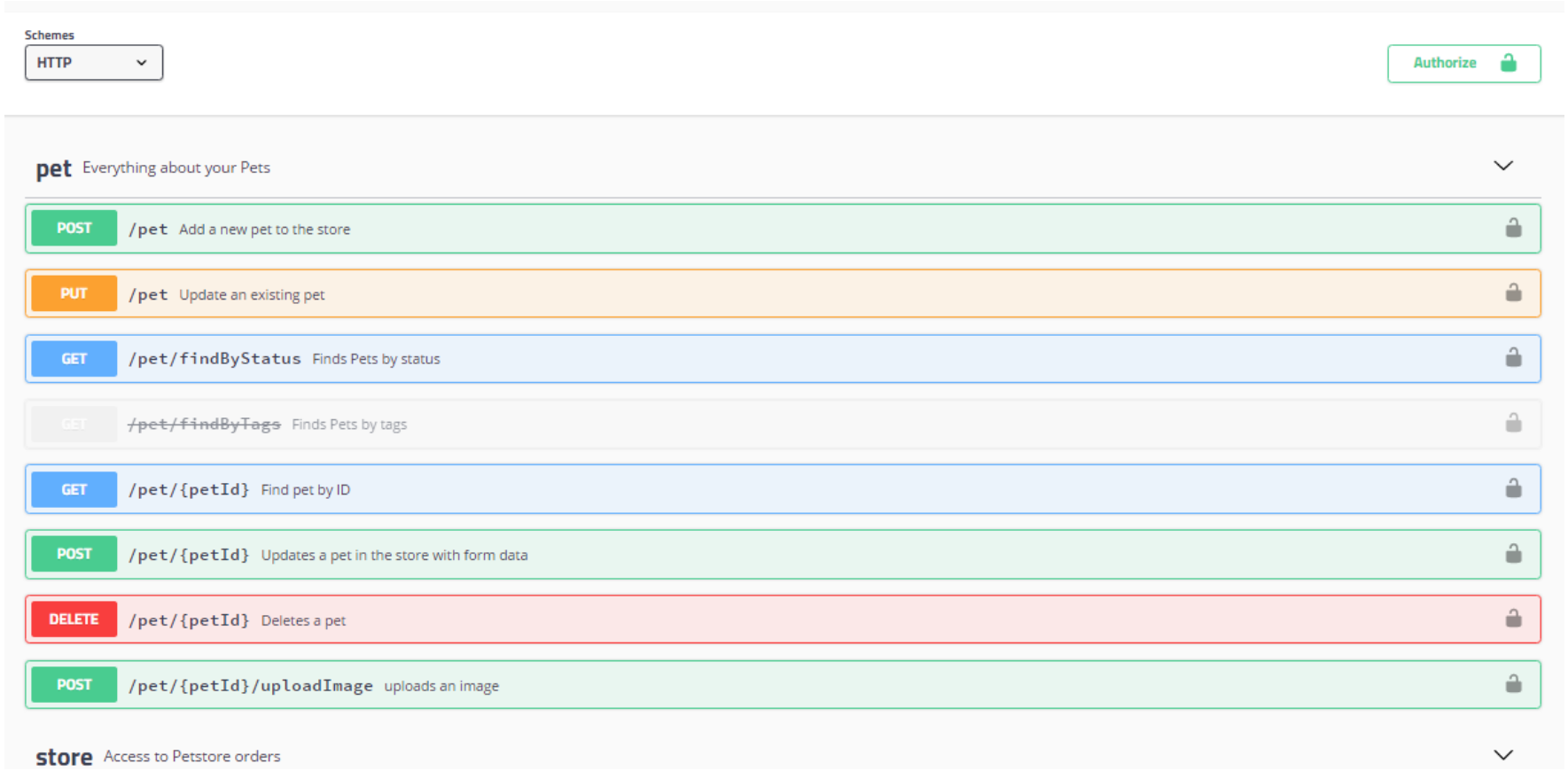


Swagger UI: An interface to visualize and interact with API. Can be accessed through subfix /docs.

https://swagger.io/tools/swagger-ui/

FastAPI

❖ Practice: Initialize first API

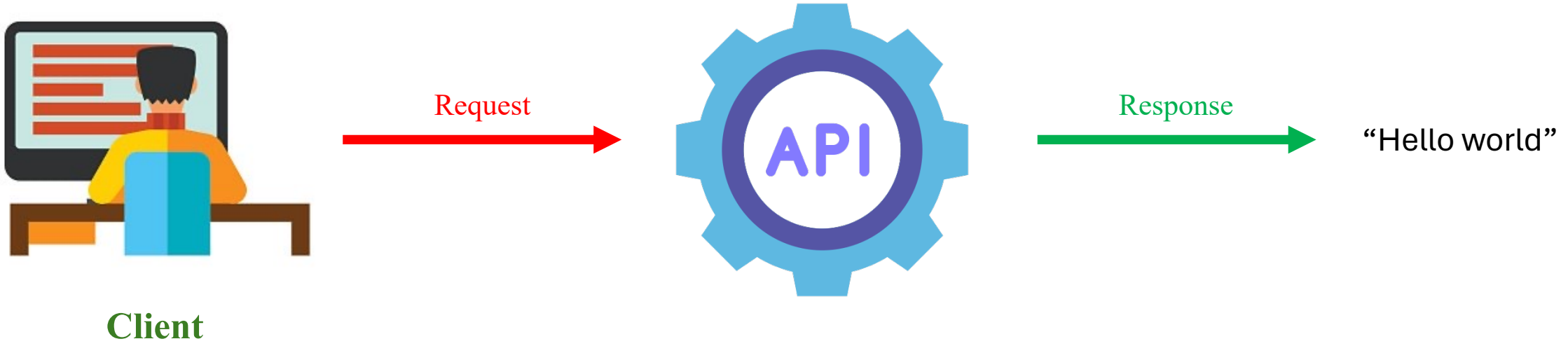


Swagger UI: An interface to visualize and interact with API. Can be accessed through subfix /docs.

https://swagger.io/tools/swagger-ui/

FastAPI

❖ Practice: Path Operations



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

FastAPI

❖ Practice: Path Operations GET Method

```python
from fastapi import FastAPI

app = FastAPI()

@app.get('/')
async def root():
    return 'Hello world'
```

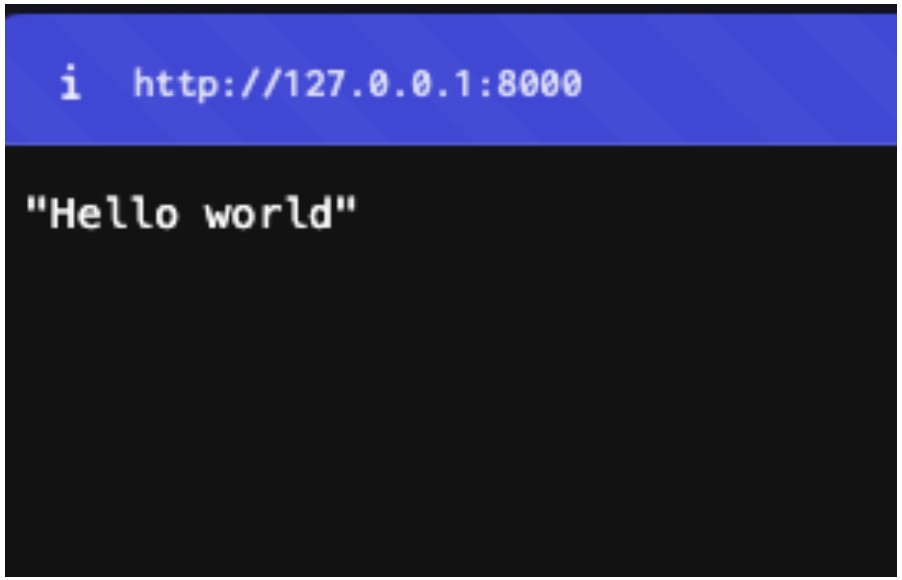
Defining a route

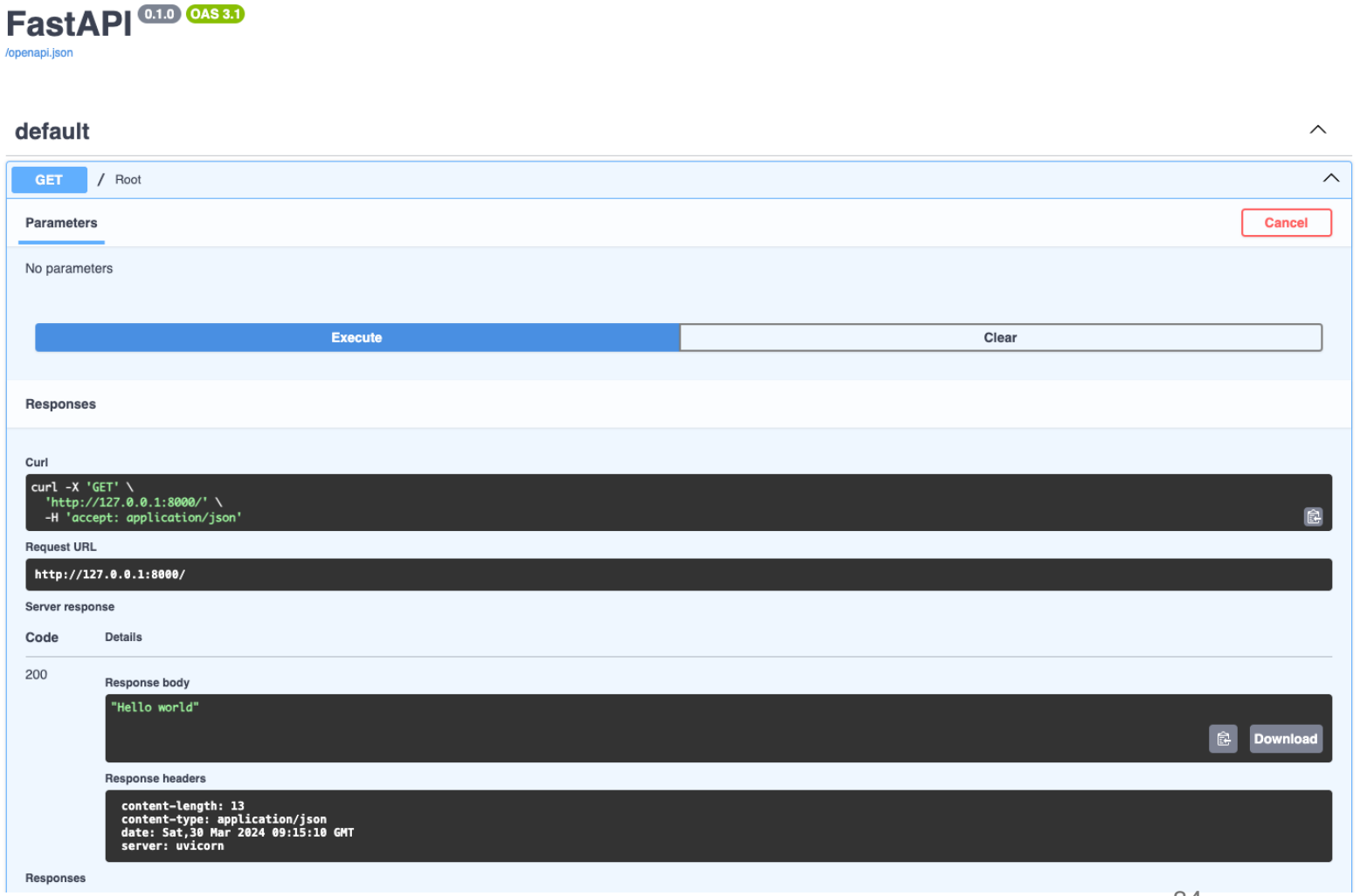
Define a function with decorator "@app.get('/')":

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

FastAPI

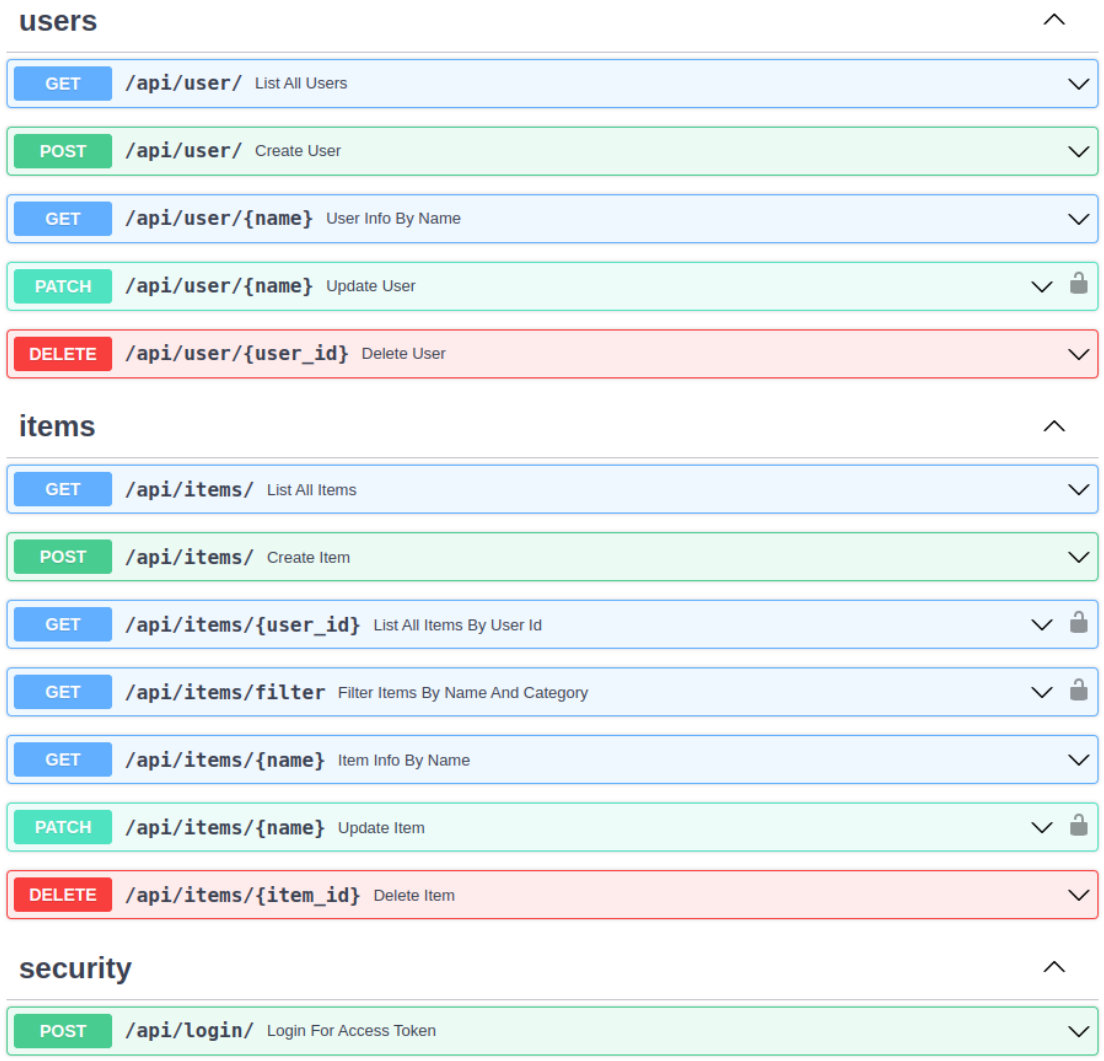
❖ Practice: Path Operations GET Method





FastAPI

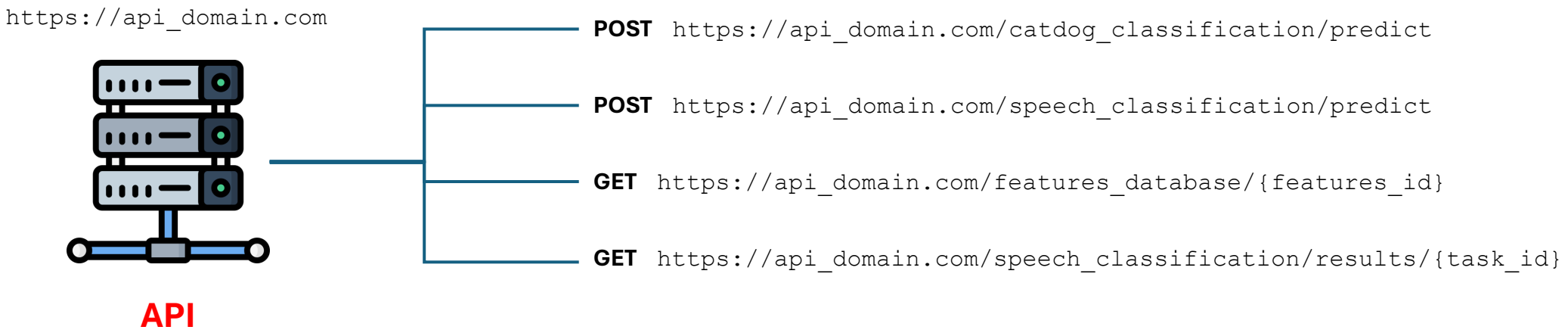
❖ Practice: Path Operations



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

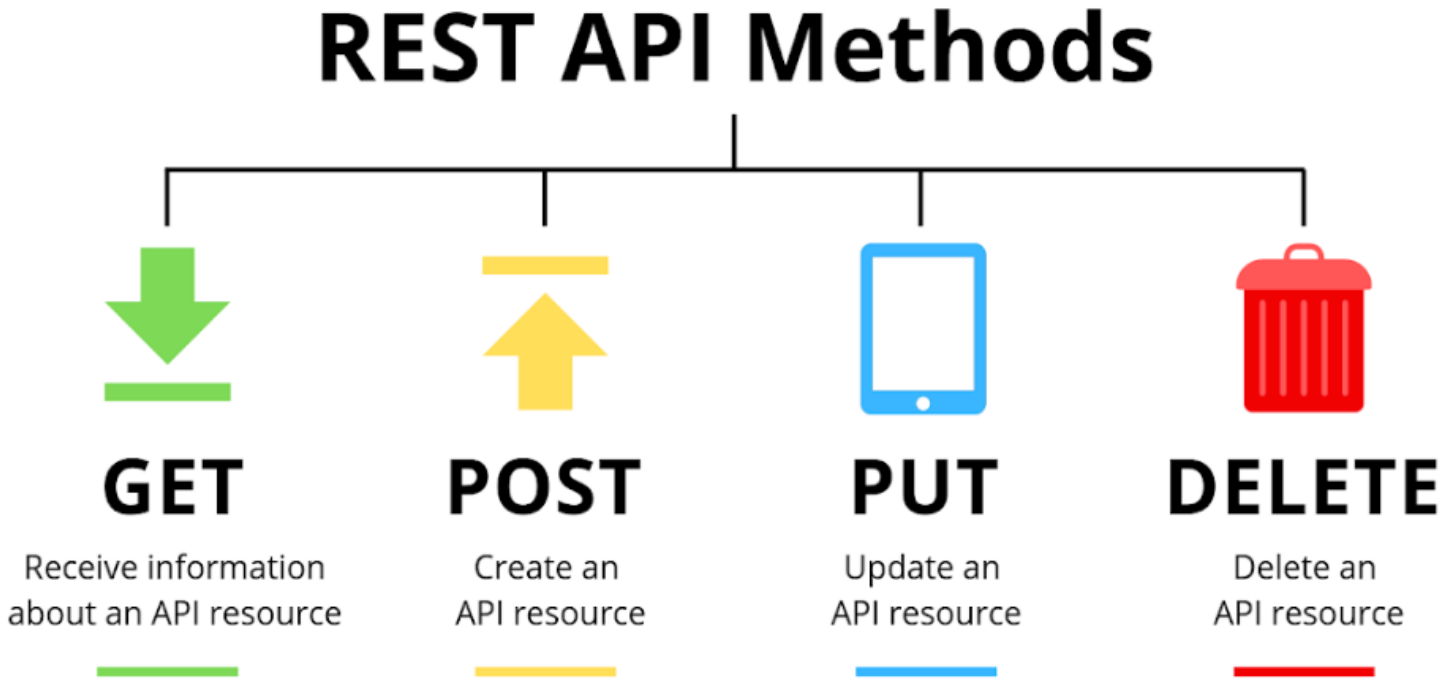
FastAPI

❖ Practice: Path Operations



FastAPI

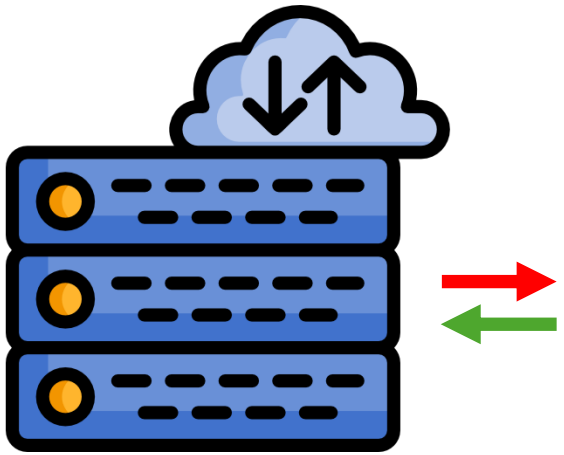
❖ Practice: Path Operations



https://api.arcadier.com/introduction-to-arcadier-api

FastAPI

❖ Practice: Path Operations



Database

How to retrieve and upload data from a database through API ?

FastAPI

❖ Practice: Path Operations GET Method

```python
from fastapi import FastAPI

db = [
    {
        'student_id': '20240001',
        'student_name': 'A'
    },
    {
        'student_id': '20240002',
        'student_name': 'B'
    }
]

app = FastAPI()
```

```python
# Declare a GET method
@app.get('/')
def root():
    return {
        'message': 'Hello AIVN'
    }

@app.get('/predict')
def predict():
    return {
        'predict_id': 0
    }

@app.get('/student_db/{student_id}')
def get_student(student_id: str):
    for record in db:
        if record['student_id'] == student_id:
            return record
```

FastAPI

❖ Practice: Path Operations (Path Parameters)

```python
# Declare a GET method
@app.get('/')
def root():
    return {
        'message': 'Hello AIVN'
    }

@app.get('/predict')
def predict():
    return {
        'predict_id': 0
    }

@app.get('/student_db/{student_id}')
def get_student(student_id: str):
    for record in db:
        if record['student_id'] == student_id:
            return record
```

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/greet/{name}")
async def greet(name: str):
    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.

FastAPI

❖ Practice: Path Operations POST Method

```python
from fastapi import FastAPI

db = [
    {
        'student_id': '20240001',
        'student_name': 'A'
    },
    {
        'student_id': '20240002',
        'student_name': 'B'
    }
]

app = FastAPI()
```

```python
@app.get('/student_db/{student_id}')
def get_student(student_id: str):
    for record in db:
        if record['student_id'] == student_id:
            return record


@app.post('/student_db')
def create_student(student_id: str, student_name: str):
    record = {
        'student_id': student_id,
        'student_name': student_name
    }

    db.append(record)
```

FastAPI

❖ Practice: Path Operations PUT Method

```python
from fastapi import FastAPI

db = [
    {
        'student_id': '20240001',
        'student_name': 'A'
    },
    {
        'student_id': '20240002',
        'student_name': 'B'
    }
]

app = FastAPI()
```

```python
@app.get('/student_db/{student_id}')
def get_student(student_id: str):
    for record in db:
        if record['student_id'] == student_id:
            return record

@app.put('/student_db/{student_id}')
def update_student(student_id: str, student_name: str):
    for record in db:
        if record['student_id'] == student_id:
            record['student_name'] = student_name
```

FastAPI

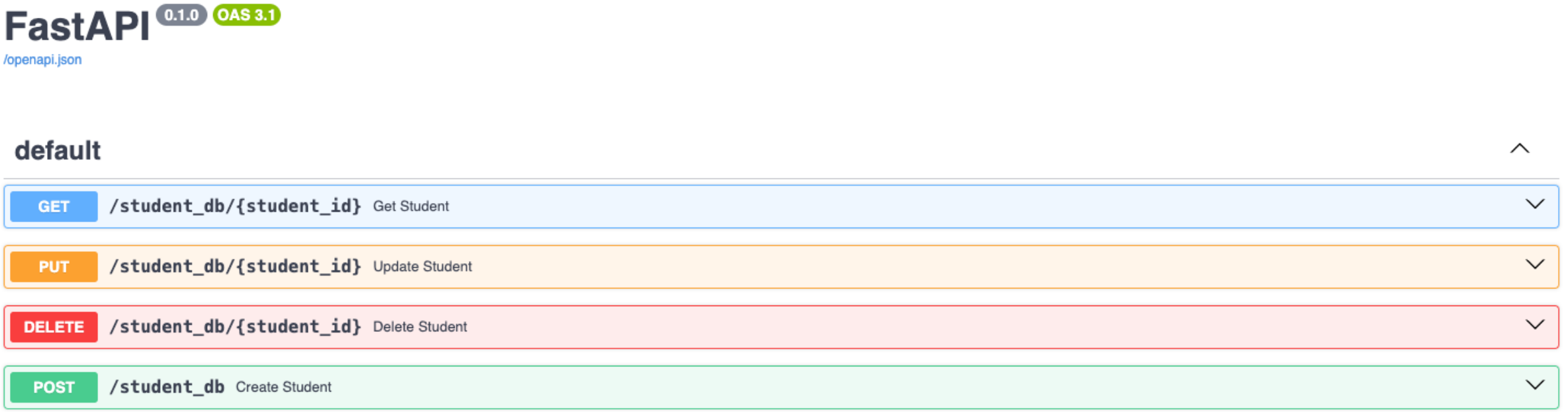
❖ Practice: Path Operations DELETE Method

```python
from fastapi import FastAPI

db = [
    {
        'student_id': '20240001',
        'student_name': 'A'
    },
    {
        'student_id': '20240002',
        'student_name': 'B'
    }
]

app = FastAPI()
```

```python
@app.get('/student_db/{student_id}')
def get_student(student_id: str):
    for record in db:
        if record['student_id'] == student_id:
            return record

@app.delete('/student_db/{student_id}')
def delete_student(student_id: str):
    for record in db:
        if record['student_id'] == student_id:
            db.remove(record)
```

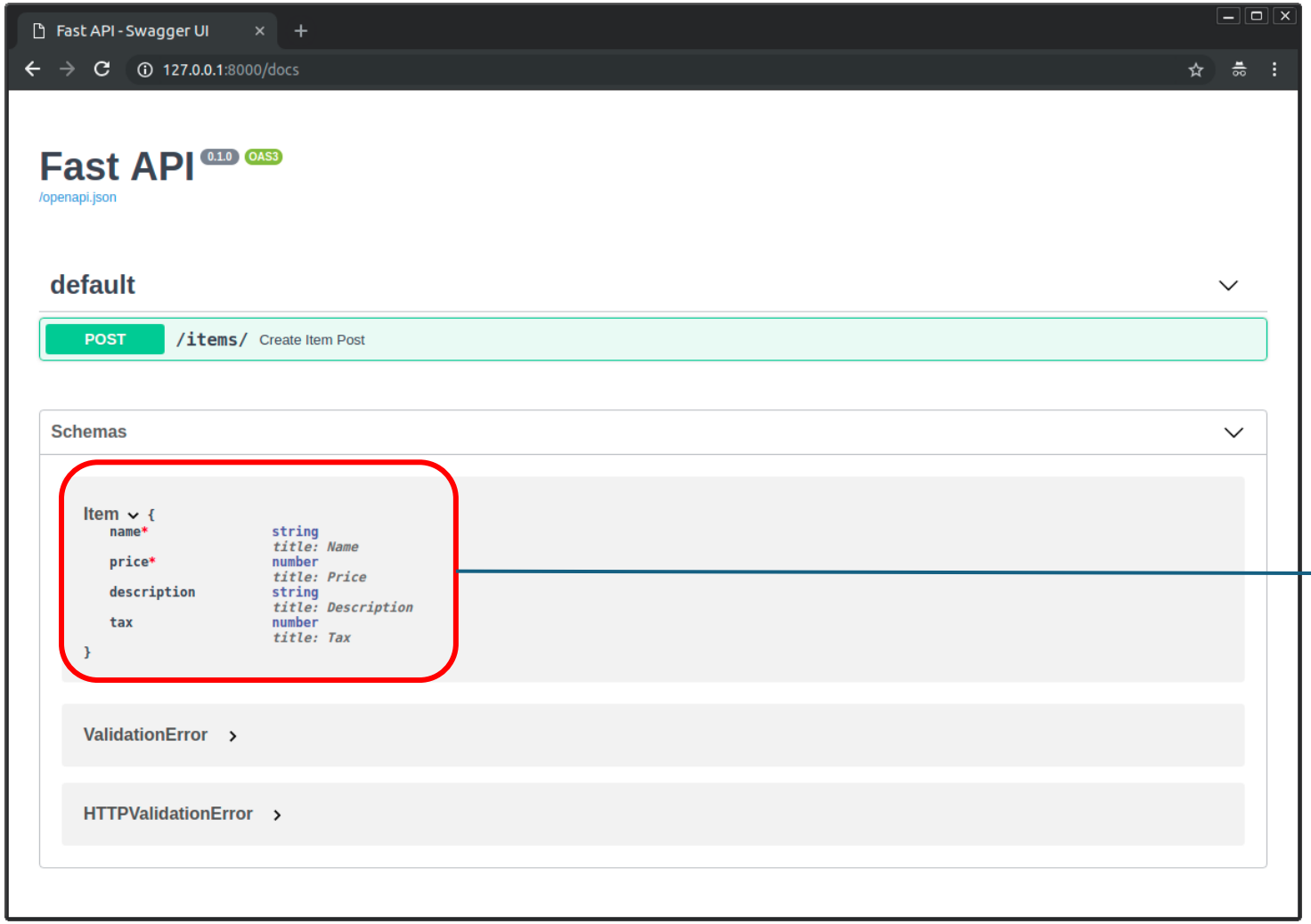
FastAPI

❖ Practice: Path Operations



FastAPI

❖ Practice: Pydantic Model



Request body: Data that is sent by Client.

Response body: Data that is sent by API.

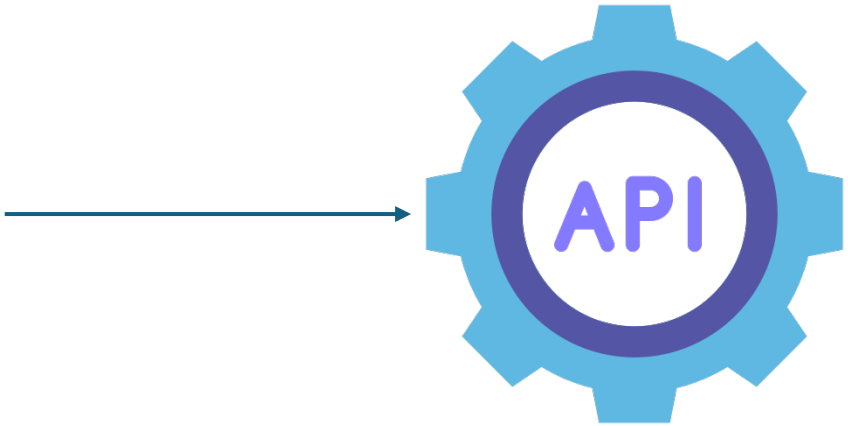
Item: {
”name”: string,
”price”: number,
“description”: string,
“tax”: number
}

An example of request body

FastAPI

❖ Practice: Pydantic Model

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



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

An example of request body

FastAPI

❖ Practice: Pydantic Model

```python
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    description: str
    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.

FastAPI

❖ Practice: Pydantic Model

E.g: Define student information record:

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

```python
from pydantic import BaseModel

class Student(BaseModel):
    id: str
    name: str
    gender: str
    age: int

a_student = Student(
    id="20240001",
    name="A",
    gender="Male",
    age=20
)

print(a_student)
# id='20240001' name='A' gender='Male' age=20
```

FastAPI

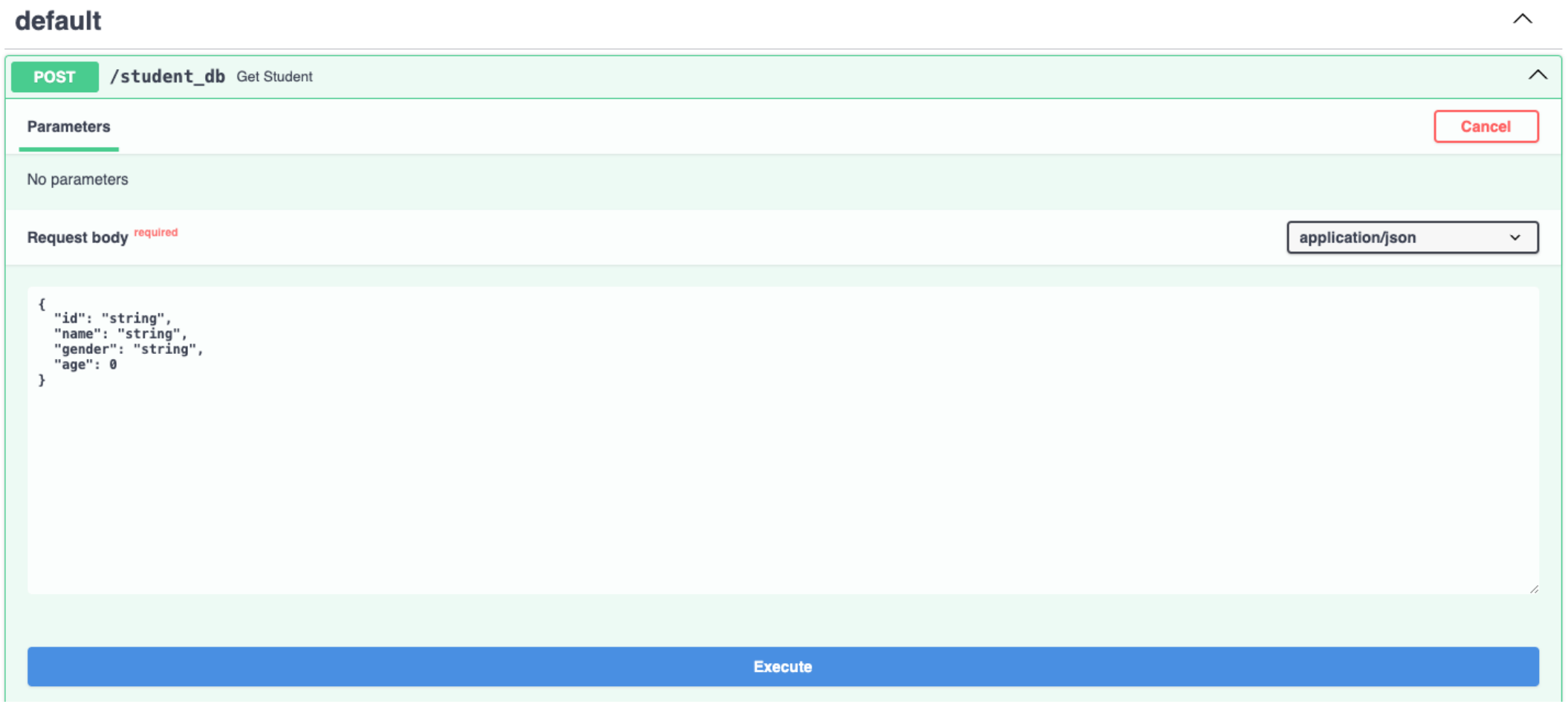
❖ Practice: Pydantic Model

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel

class Student(BaseModel):
    id: str
    name: str
    gender: str
    age: int

db = [
    Student(
        id="20240001",
        name="A",
        gender="Male",
        age=20
    )
]

app = FastAPI()
```

```python
@app.post('/student_db')
def get_student(student: Student):
    return student

@app.get('/student_db/{student_id}')
def get_student(student_id: str):
    for s in db:
        if s.id == student_id:
            return s

    raise HTTPException(status_code=404, detail='Student not found')
```

FastAPI

❖ Practice: Pydantic Model



FastAPI

❖ Practice: Pydantic Model

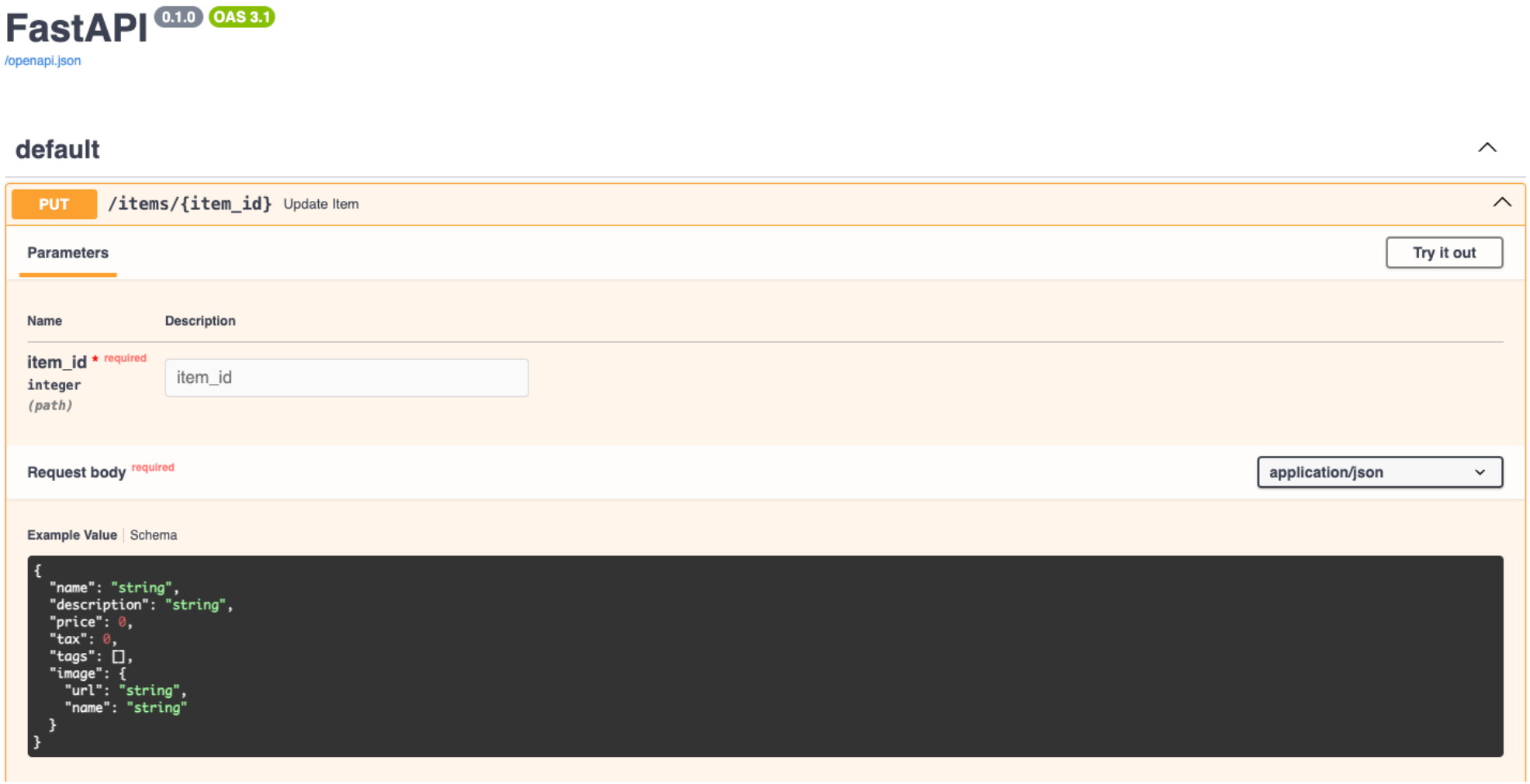
```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Image(BaseModel):
    url: str
    name: str

class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None
    tags: set[str] = set()
    image: Image | None = None
```

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

A pydantic model can be nested

FastAPI

❖ Practice: Pydantic Model



FastAPI

❖ Practice: Parameters Annotation

```python
from typing import Optional, Union

def add(x: int, y: int, op: Optional[str]) -> int:
    return x + y
add(3, 5, None)

def add2(x: int, y: int, op: Union[None, str]) -> int:
    return x + y
add2(3, 5, None)
```

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

FastAPI

❖ Practice: Parameters Annotation

```python
from typing import Annotated

from fastapi import FastAPI, Path, Query

app = FastAPI()

@app.get("/items/{item_id}")
async def read_items(
    item_id: Annotated[int, Path(title="The ID of the item to get")],
    q: Annotated[str | None, Query(alias="item-query")] = None,
):
    results = {"item_id": item_id}
    if q:
        results.update({"q": q})
    return results
```

Use Annotated to give additional information for read_items() function.

FastAPI

❖ Practice: Response Model

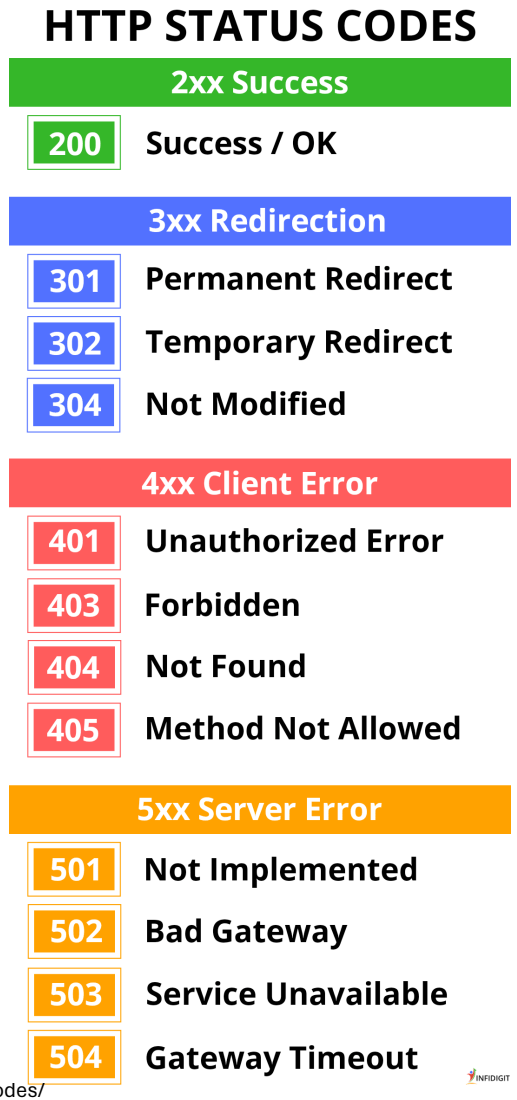
```python
from typing import Any

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None
    tags: list[str] = []
```

```python
@app.post("/items/", response_model=Item)
async def create_item(item: Item) -> Any:
    return item

@app.get("/items/", response_model=list[Item])
async def read_items() -> Any:
    return [
        {"name": "Portal Gun", "price": 42.0},
        {"name": "Plumbus", "price": 32.0},
    ]
```

We can specify the response type of the route

FastAPI

❖ Practice: Status Code



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.

https://www.infidigit.com/blog/http-status-codes/

FastAPI

❖ 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

https://www.infidigit.com/blog/http-status-codes/

```
(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
```

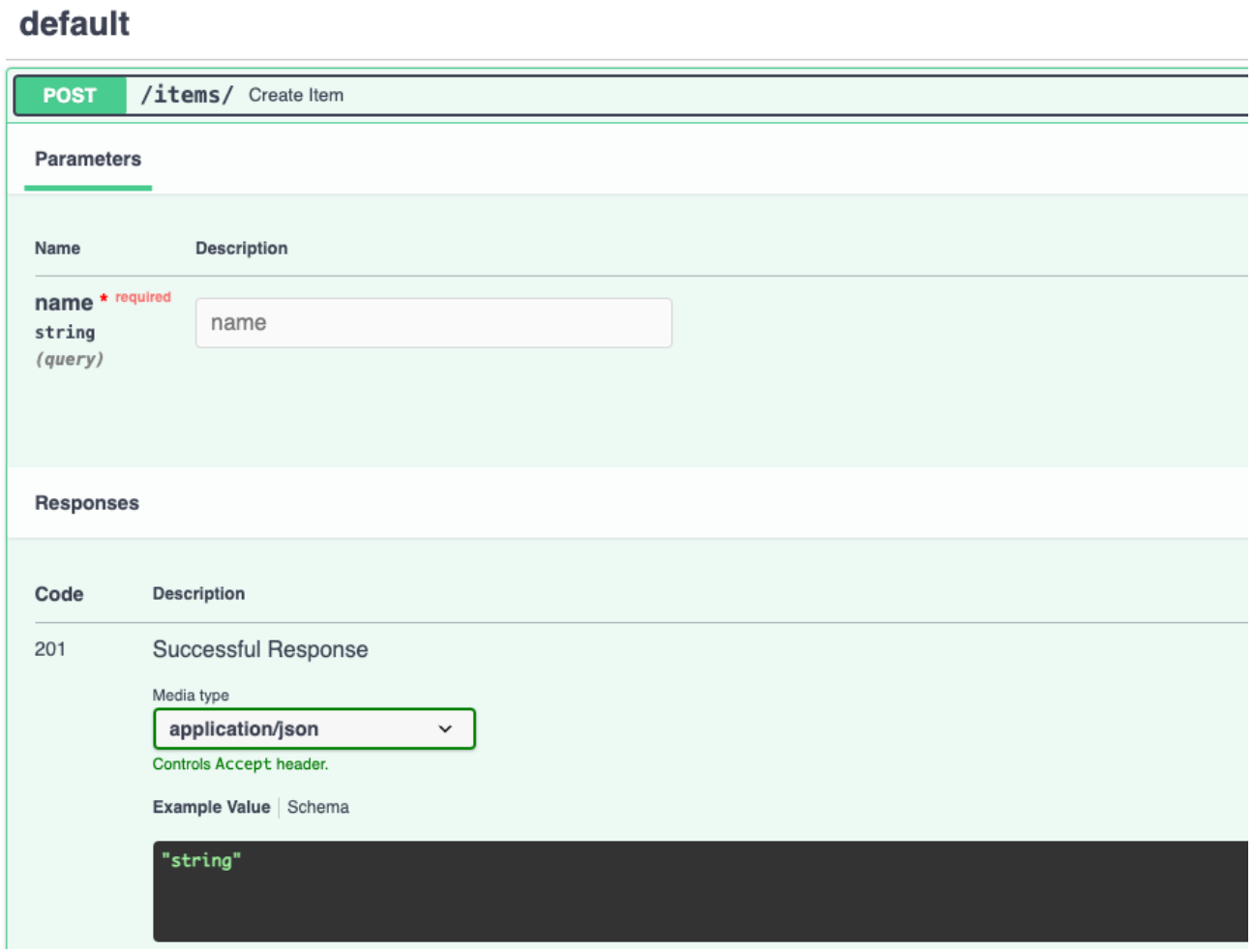
When a request is completed successfully, we got 200 OK

FastAPI

❖ Practice: Status Code

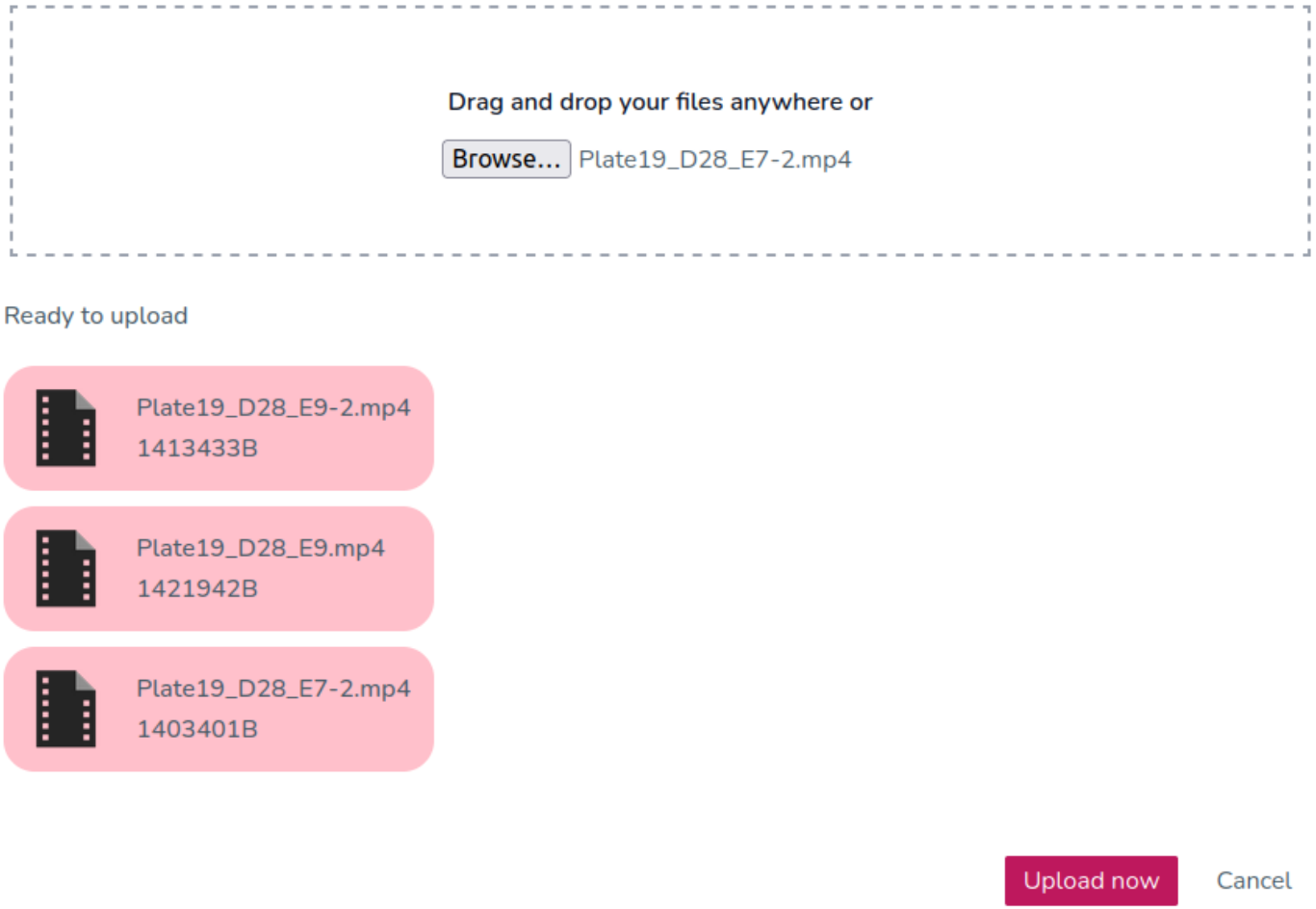
```python
from fastapi import FastAPI

app = FastAPI()

@app.post("/items/", status_code=201)
async def create_item(name: str):
    return {"name": name}
```

We can define the status code return in decorator



FastAPI

❖ Practice: Upload File

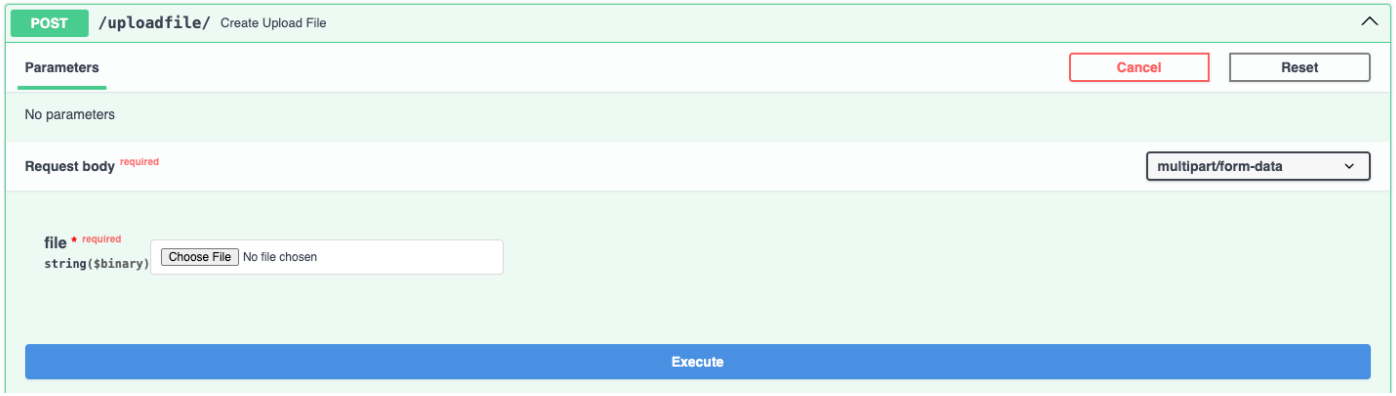


Beside Pydantic model, we can also have request body as a uploaded file

FastAPI

❖ Practice: Upload File

```python
from fastapi import FastAPI, File, UploadFile
from typing_extensions import Annotated

app = FastAPI()

@app.post("/files/")
async def create_file(file: Annotated[bytes, File()]):
    return {"file_size": len(file)}

@app.post("/uploadfile/")
async def create_upload_file(file: UploadFile):
    return {"filename": file.filename}
```



Use File or UploadFile module in FastAPI

FastAPI

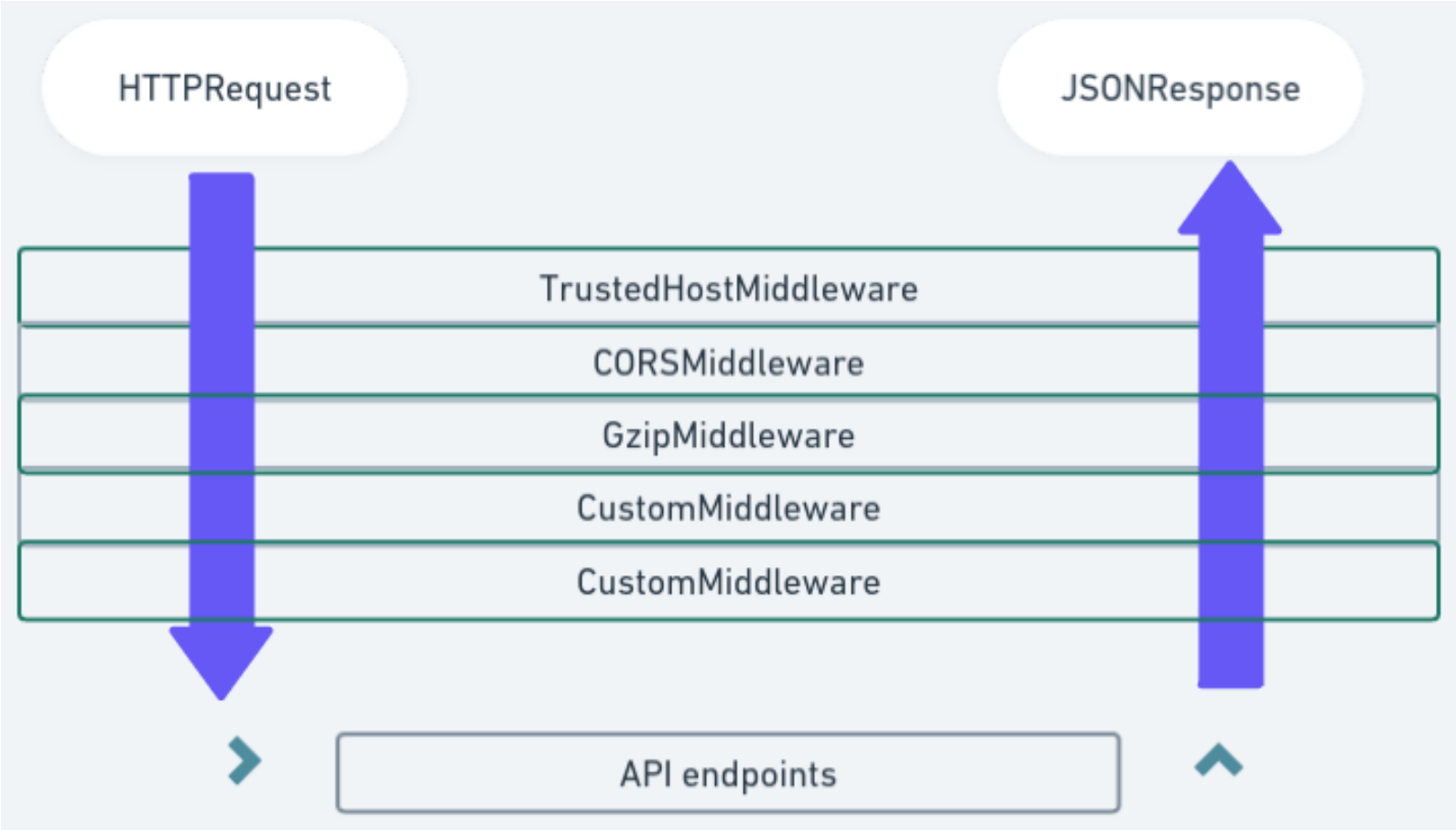
❖ Practice: Middleware

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.

https://semaphoreci.com/blog/custom-middleware-fastapi

FastAPI

❖ Practice: Middleware



https://semaphoreci.com/blog/custom-middleware-fastapi

FastAPI

❖ Practice: Middleware

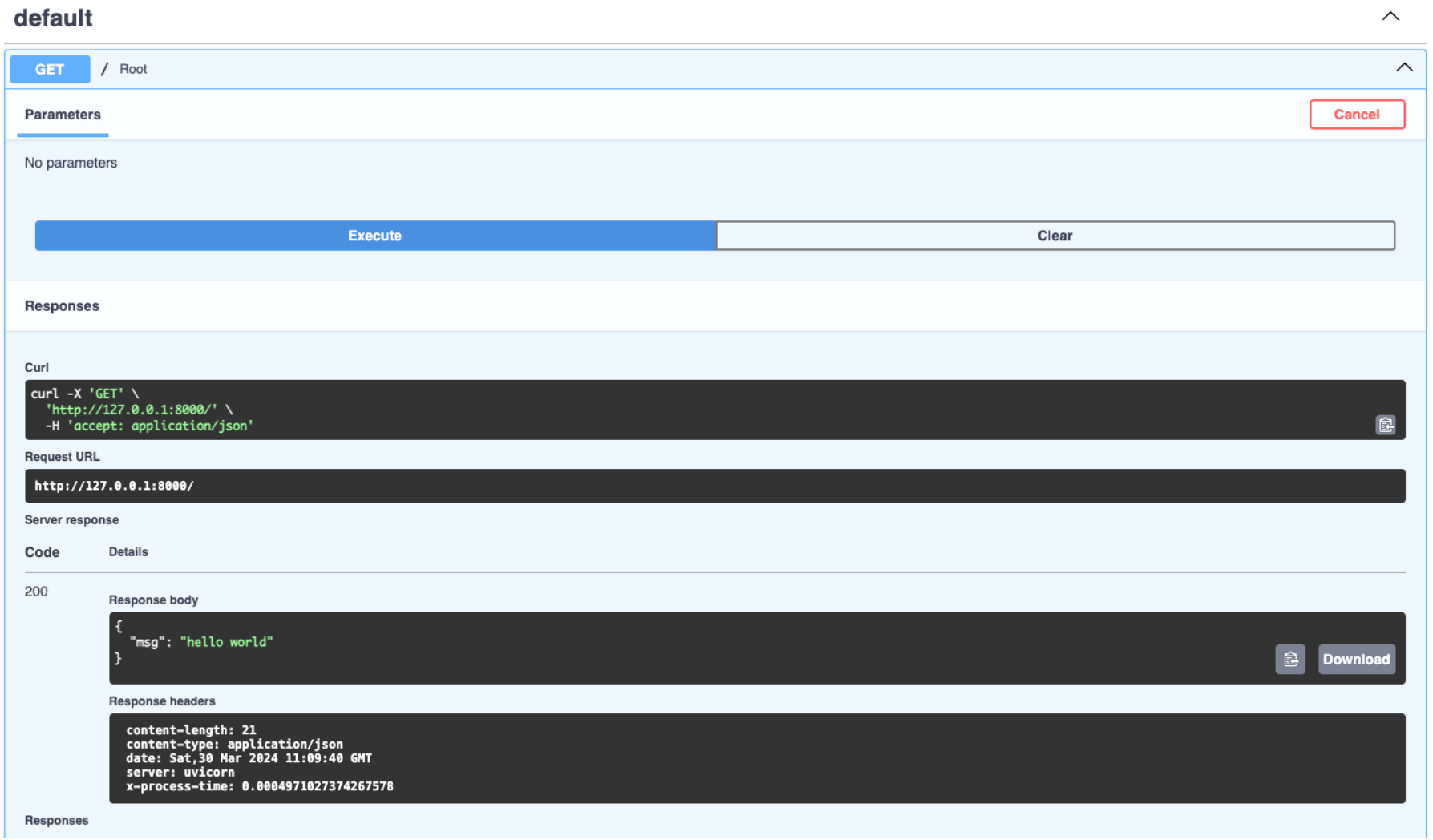
```python
import time

from fastapi import FastAPI, Request

app = FastAPI()

@app.get('/')
async def root():
    return {
        'msg': 'hello world'
    }
```

```python
@app.middleware("http")
async def add_process_time_header(request: Request, call_next):
    start_time = time.time()
    response = await call_next(request)
    process_time = time.time() - start_time
    response.headers["X-Process-Time"] = str(process_time)

    return response
```

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

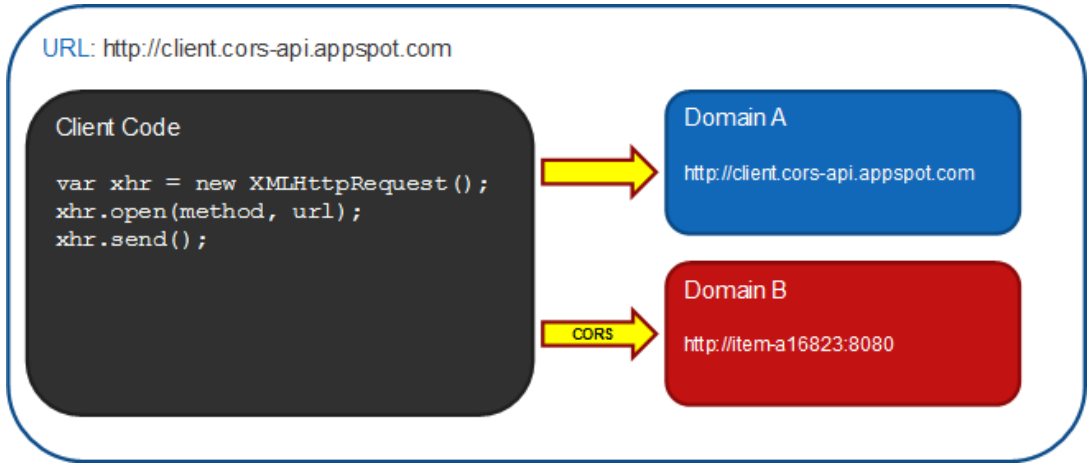
FastAPI

❖ Practice: Middleware



FastAPI

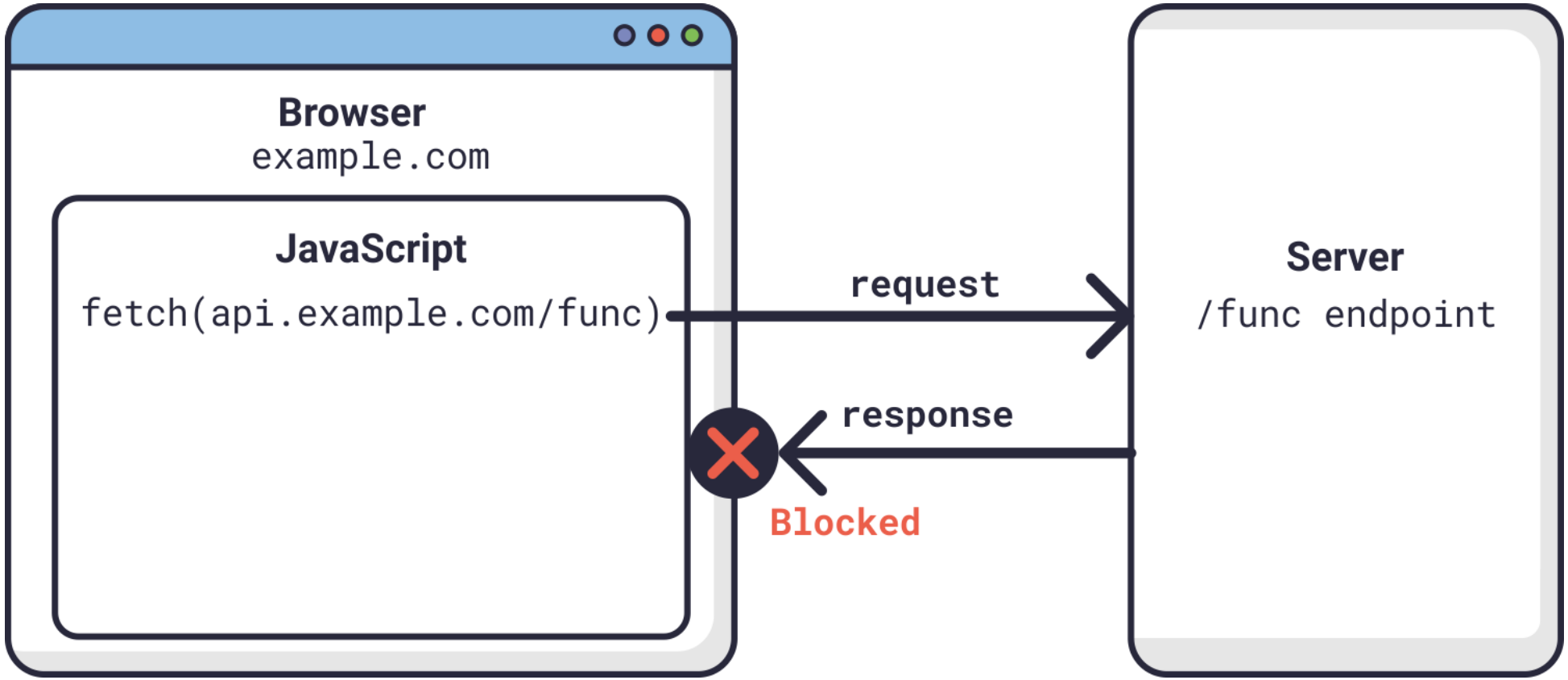
❖ Practice: CORS Middleware



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.

FastAPI

❖ Practice: CORS Middleware



https://www.haproxy.com/blog/enabling-cors-in-haproxy

FastAPI

❖ Practice: CORS Middleware

```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

origins = [
    "http://localhost.tiangolo.com",
    "https://localhost.tiangolo.com",
    "http://localhost",
    "http://localhost:8080",
]
```

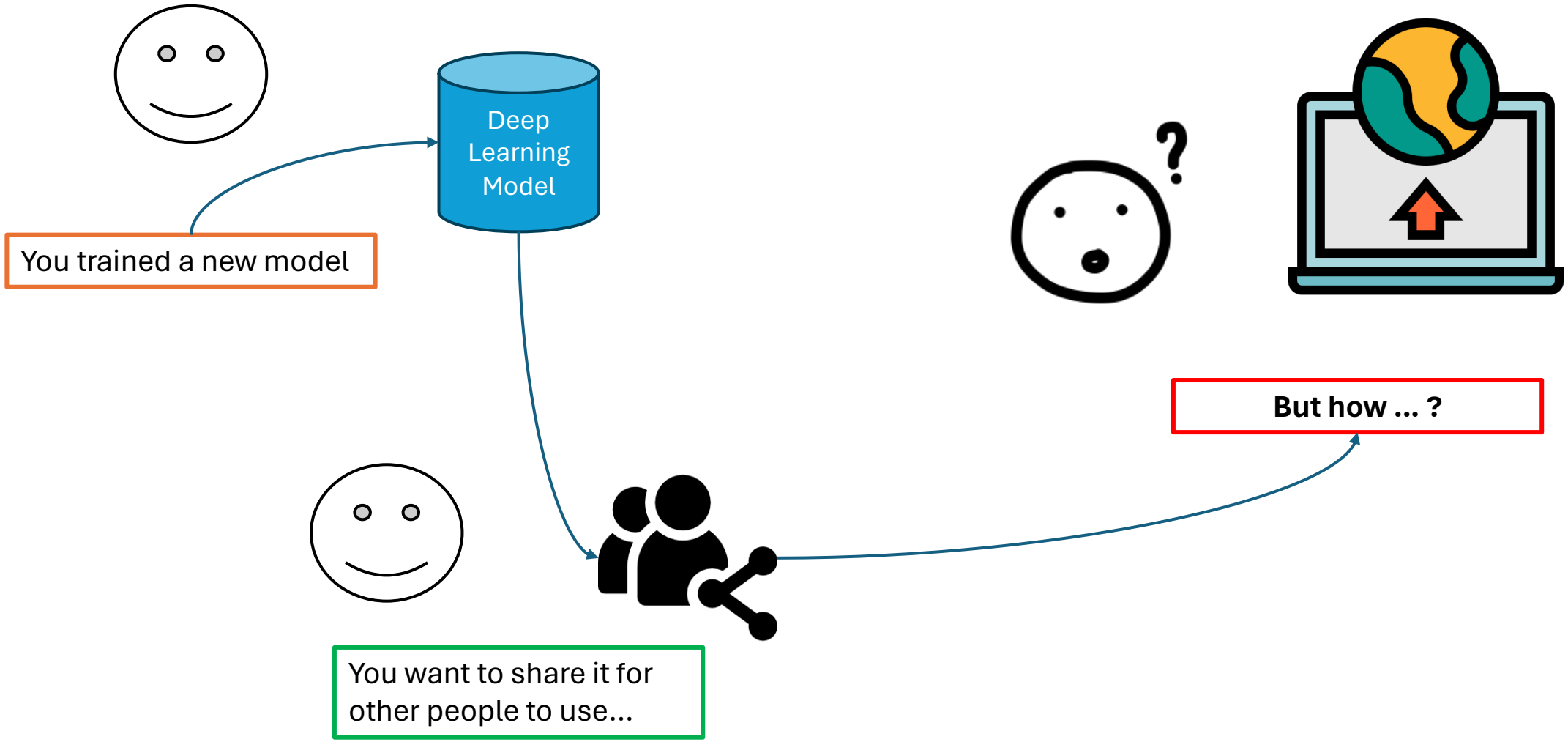
```python
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)


@app.get("/")
async def main():
    return {"message": "Hello World"}
```

Quiz

Model Deployment

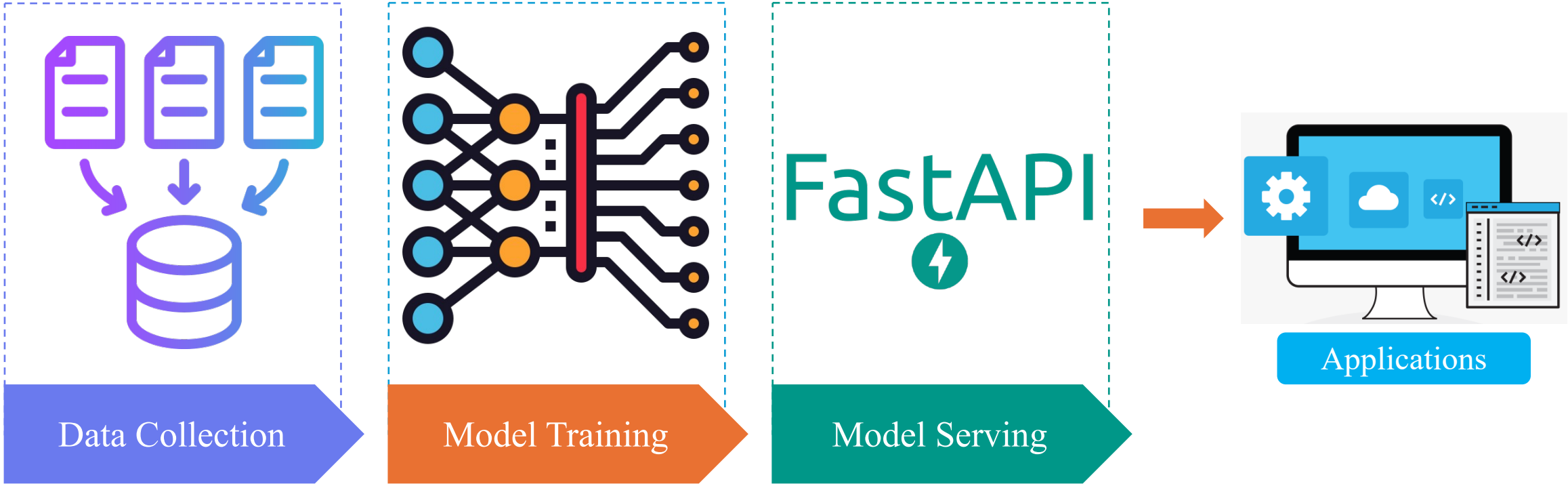
Model Deployment

❖ Introduction



Model Deployment

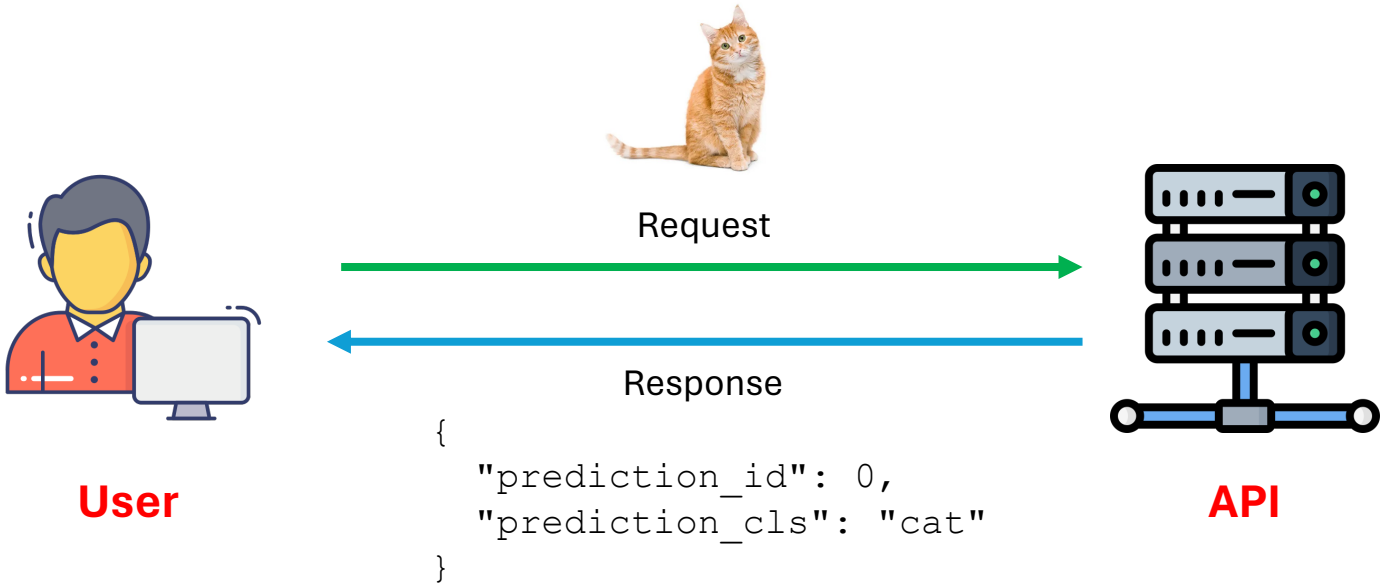
❖ Pipeline



Model Deployment

❖ Practice

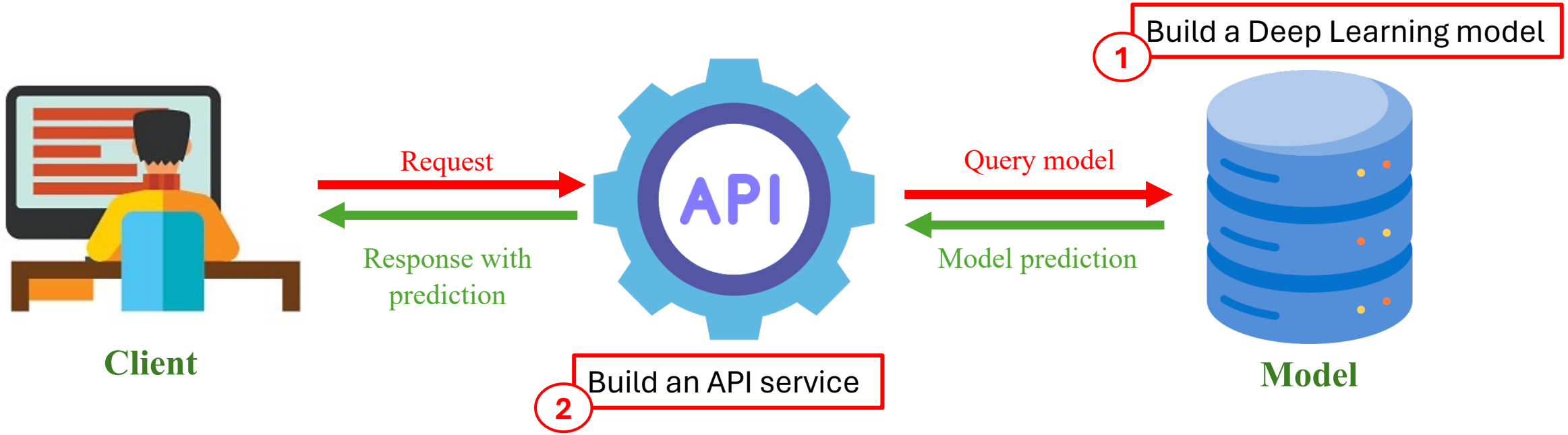
Problem statement: Deploy a Cat Dog Image Classification model as an API service using FastAPI.



Model Deployment

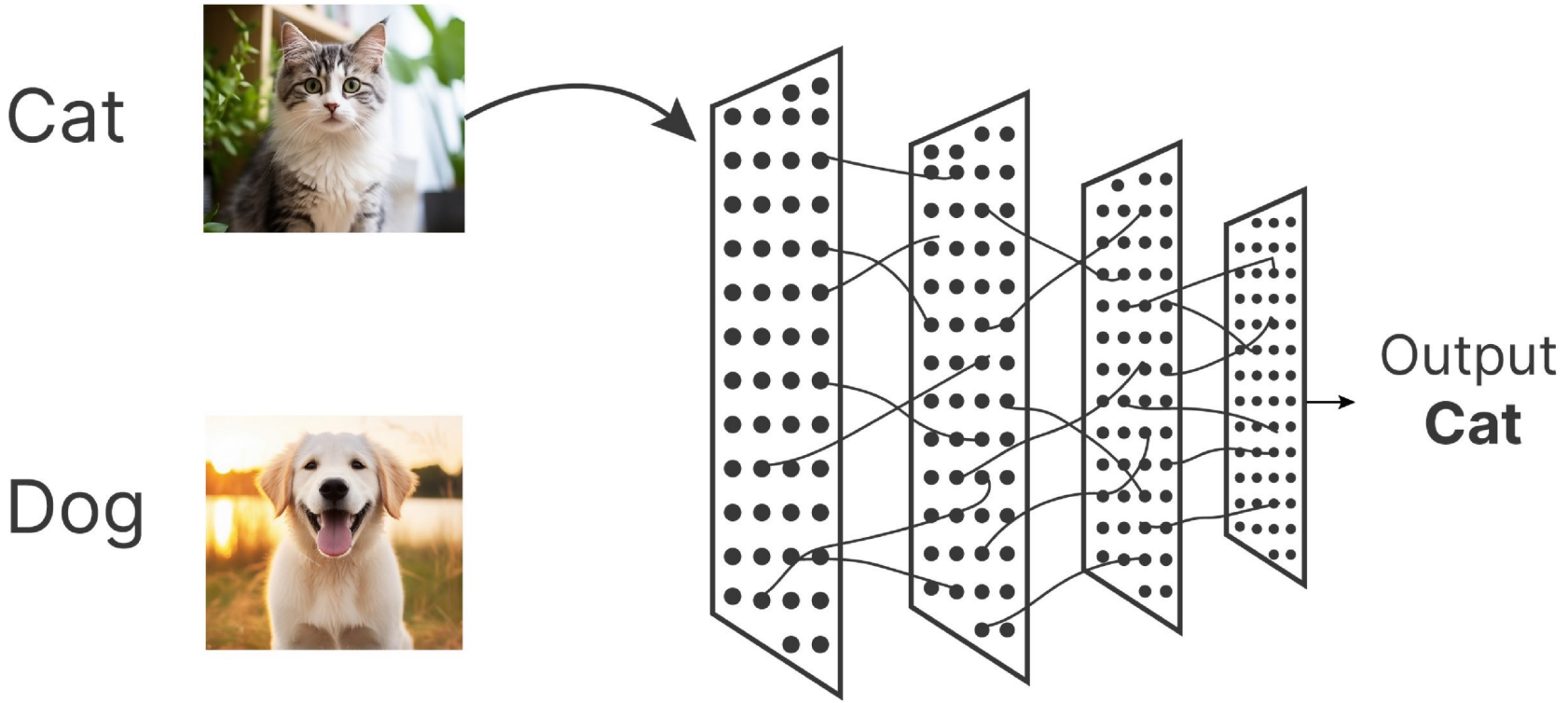
❖ Practice

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



Model Deployment

❖ Build model



https://www.analyticsvidhya.com/blog/2021/06/beginner-friendly-project-cat-and-dog-classification-using-cnn/

Model Deployment

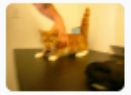
❖ Build model: Dataset

Dataset Preview ⓘ

</> API | View in Dataset Viewer

Split (1)
train

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

image image	labels 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)
```

```
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://hu
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 repositc
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 majc
  warnings.warn(
DatasetDict({
    train: Dataset({
        features: ['image', 'labels'],
        num_rows: 23410
    })
})
```

```
1 TEST_SIZE = 0.2
2 datasets = datasets['train'].train_test_split(test_size=TEST_SIZE)
```

Model Deployment

❖ Build model step 3: Create DataLoader

```python
IMG_SIZE = 64
img_transforms = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(
        [0.485, 0.456, 0.406],
        [0.229, 0.224, 0.225]
    )
])
```

```python
class CatDogDataset(Dataset):
    def __init__(self, data, transform=None):
        self.data = data
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        images = self.data[idx]['image']
        labels = self.data[idx]['labels']

        if self.transform:
            images = self.transform(images)

        labels = torch.tensor(labels, dtype=torch.long)

        return images, labels
```

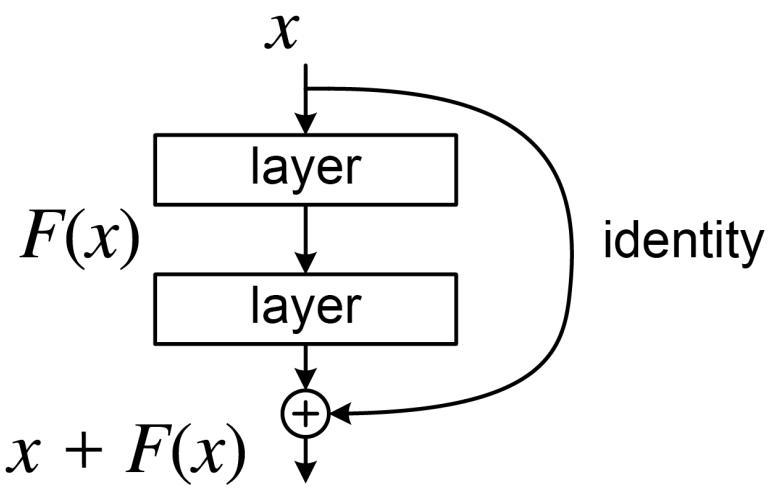
```python
TRAIN_BATCH_SIZE = 512
VAL_BATCH_SIZE = 256

train_dataset = CatDogDataset(
    datasets['train'], transform=img_transforms
)
test_dataset = CatDogDataset(
    datasets['test'], transform=img_transforms
)

train_loader = DataLoader(
    train_dataset,
    batch_size=TRAIN_BATCH_SIZE,
    shuffle=True
)
test_loader = DataLoader(
    test_dataset,
    batch_size=VAL_BATCH_SIZE,
    shuffle=False
)
```

Model Deployment

❖ Build model step 4: Build model

```python
class CatDogModel(nn.Module):
    def __init__(self, n_classes):
        super(CatDogModel, self).__init__()

        resnet_model = resnet18(weights='IMAGENET1K_V1')
        self.backbone = nn.Sequential(*list(resnet_model.children())[:-1])
        for param in self.backbone.parameters():
            param.requires_grad = False

        in_features = resnet_model.fc.in_features
        self.fc = nn.Linear(in_features, n_classes)

    def forward(self, x):
        x = self.backbone(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)

        return x

device = 'cuda' if torch.cuda.is_available() else 'cpu'
N_CLASSES = 2
model = CatDogModel(N_CLASSES).to(device)
test_input = torch.rand(1, 3, 224, 224).to(device)
with torch.no_grad():
    output = model(test_input)
    print(output.shape)
```

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



Model Deployment

❖ Build model step 5: Training and saving model weights

```
EPOCHS = 10
LR = 1e-3
WEIGHT_DECAY = 1e-5

optimizer = torch.optim.Adam(model.parameters(), lr=LR, weight_decay=WEIGHT_DECAY)
criterion = torch.nn.CrossEntropyLoss()
```

```
for epoch in range(EPOCHS):
    train_losses = []
    model.train()
    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)

        optimizer.zero_grad()
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        train_losses.append(loss.item())

    train_loss = sum(train_losses) / len(train_losses)

    val_losses = []
    model.eval()
    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)

            val_losses.append(loss.item())

    val_loss = sum(val_losses) / len(val_losses)

    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
```

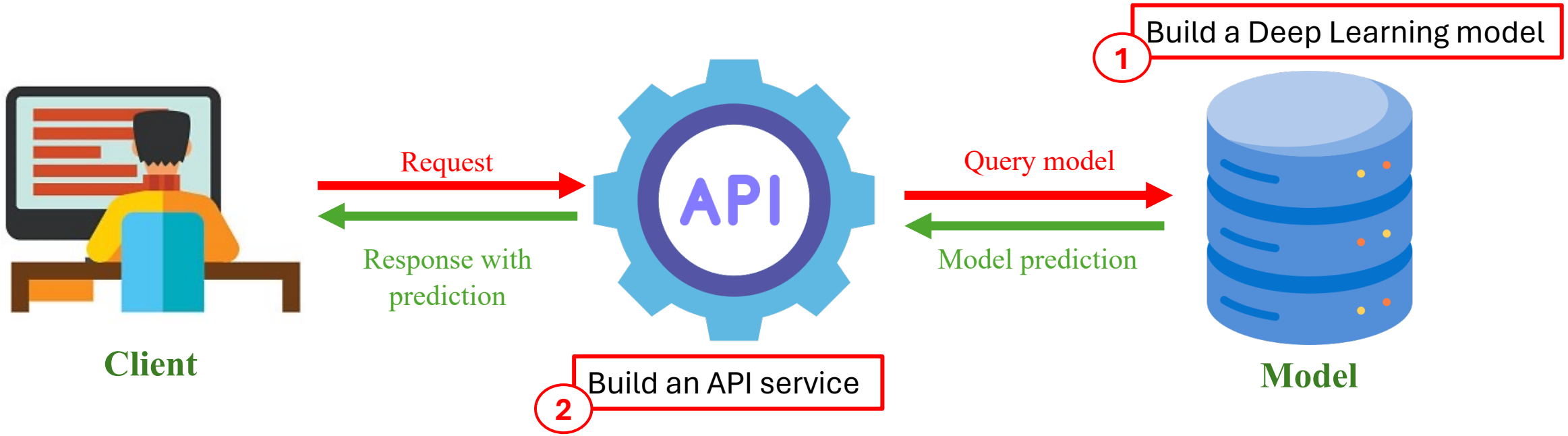
7. Save model

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

Model Deployment

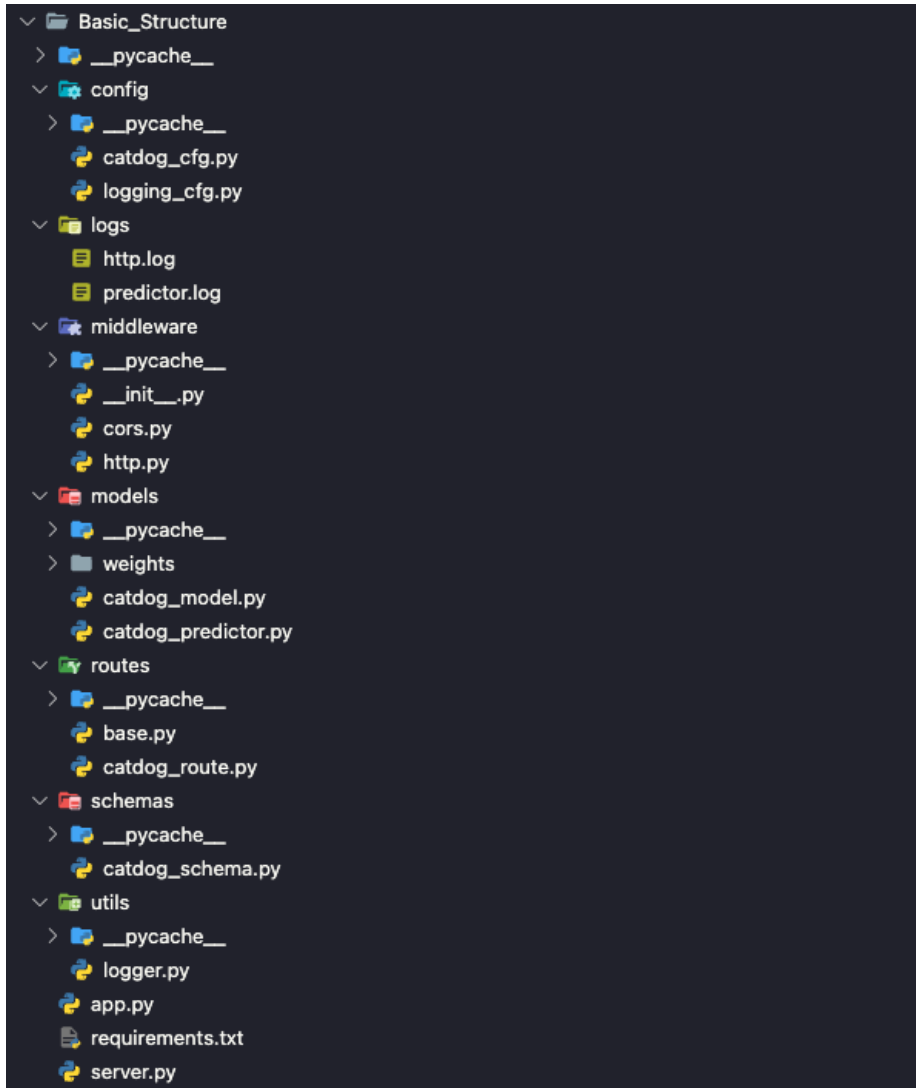
❖ Build API

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



Model Deployment

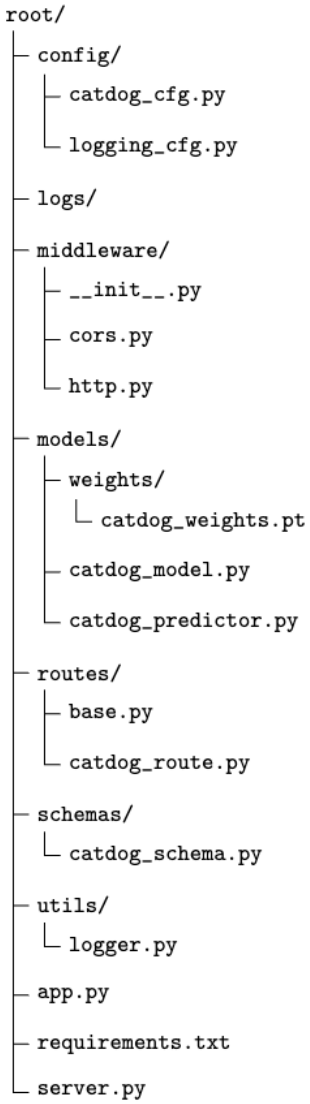
❖ Build API Step 1: Organize source code



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

Model Deployment

❖ Build API Step 1: Organize source code



- **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 soruce code.
- **server.py:** Python file containing codes to host the API service.

Model Deployment

❖ Build API Step 2: List packages requirements

requirements.txt

Basic_Structure > requirements.txt

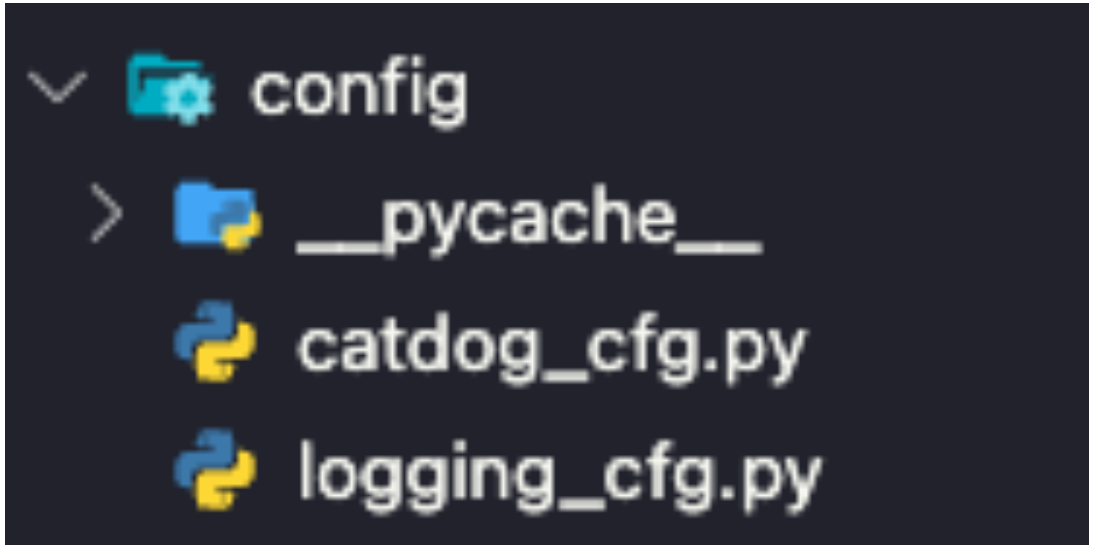
Click here to ask Blackbox to help you code faster

```
1  fastapi==0.110.0
2  uvicorn==0.27.1
3  torch==2.2.1
4  torchvision==0.17.1
5  python-multipart==0.0.9
```

```
1 $ 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

```python
import sys

from pathlib import Path
sys.path.append(str(Path(__file__).parent))

class CatDogDataConfig:
    N_CLASSES = 2
    IMG_SIZE = 64
    ID2DLABEL = {0: 'Cat', 1: 'Dog'}
    LABEL2ID = {'Cat': 0, 'Dog': 1}
    NORMALIZE_MEAN = [0.485, 0.456, 0.406]
    NORMALIZE_STD = [0.229, 0.224, 0.225]

class ModelConfig:
    ROOT_DIR = Path(__file__).parent.parent
    MODEL_NAME = 'resnet18'
    MODEL_WEIGHT = ROOT_DIR / 'models' / 'weights' / 'catdog_weights.pt'
    DEVICE = 'cpu'
```

logging_cfg.py

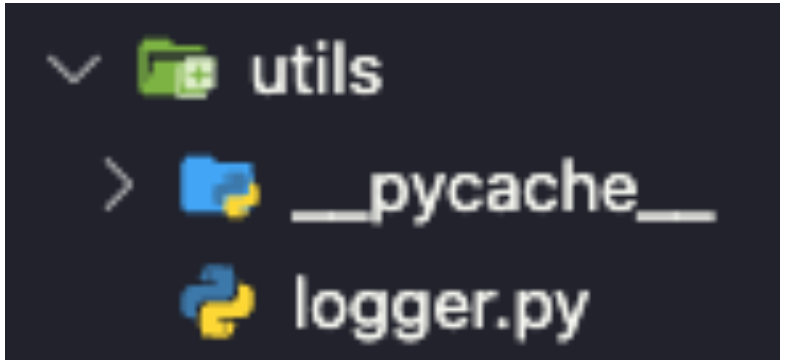
Basic_Structure > config > logging_cfg.py > ...

Click here to ask Blackbox to help you code faster

```python
from pathlib import Path

class LoggingConfig:
    ROOT_DIR = Path(__file__).parent.parent

    LOG_DIR = ROOT_DIR / "logs"

LoggingConfig.LOG_DIR.mkdir(parents=True, exist_ok=True)
```

Model Deployment

❖ Build API Step 4: Build logging function




```python
import sys
import logging

from pathlib import Path
sys.path.append(str(Path(__file__).parent))

from logging.handlers import RotatingFileHandler
from config.logging_cfg import LoggingConfig

class Logger:
    def __init__(self, name="", log_level=logging.INFO, log_file=None) -> None:
        self.log = logging.getLogger(name)
        self.get_logger(log_level, log_file)

    def get_logger(self, log_level, log_file):
        self.log.setLevel(log_level)
        self._init_formatter()
        if log_file is not None:
            self._add_file_hander(LoggingConfig.LOG_DIR / log_file)
        else:
            self._add_stream_hander()
```

Model Deployment

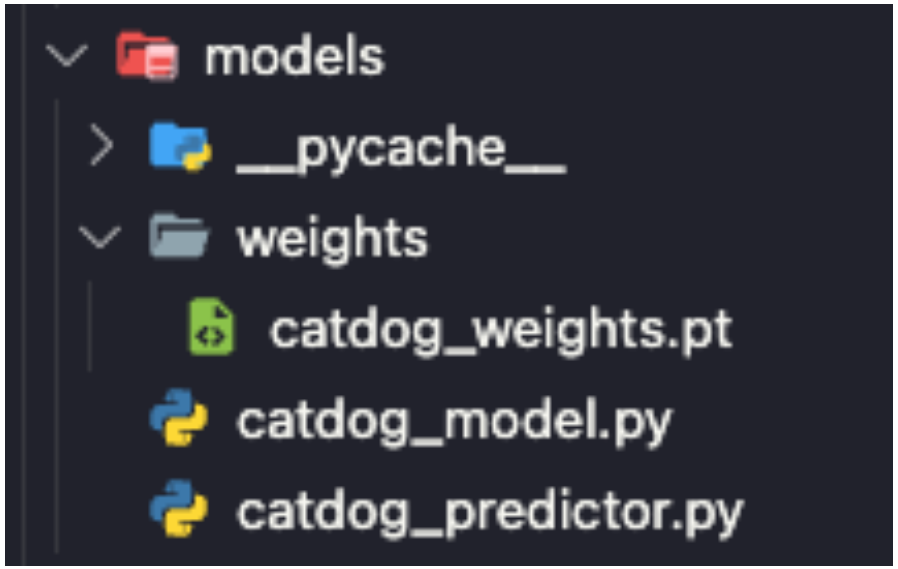
❖ Build API Step 4: Build logging function

```python
    # Continue previous code

    def _init_formatter(self):
        self.formatter = logging.Formatter(
            "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
        )
    def _add_stream_hander(self):
        stream_handler = logging.StreamHandler(sys.stdout)
        stream_handler.setFormatter(self.formatter)
        self.log.addHandler(stream_handler)

    def _add_file_hander(self, log_file):
        file_handler = RotatingFileHandler(log_file, maxBytes=10000, backupCount=10)
        file_handler.setFormatter(self.formatter)
        self.log.addHandler(file_handler)

    def log_model(self, predictor_name):
        self.log.info(f"Predictor name: {predictor_name}")

    def log_response(self, pred_prob, pred_id, pred_class):
        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



catdog_model.py

```python
import torch
import torch.nn as nn

from torchvision.models import resnet18

class CatDogModel(nn.Module):
    def __init__(self, n_classes):
        super(CatDogModel, self).__init__()

        resnet_model = resnet18(weights='IMAGENET1K_V1')
        self.backbone = nn.Sequential(*list(resnet_model.children())[:-1])
        for param in self.backbone.parameters():
            param.requires_grad = False

        in_features = resnet_model.fc.in_features
        self.fc = nn.Linear(in_features, n_classes)

    def forward(self, x):
        x = self.backbone(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)

        return x
```

Model Deployment

❖ Build API Step 5: Define models folder

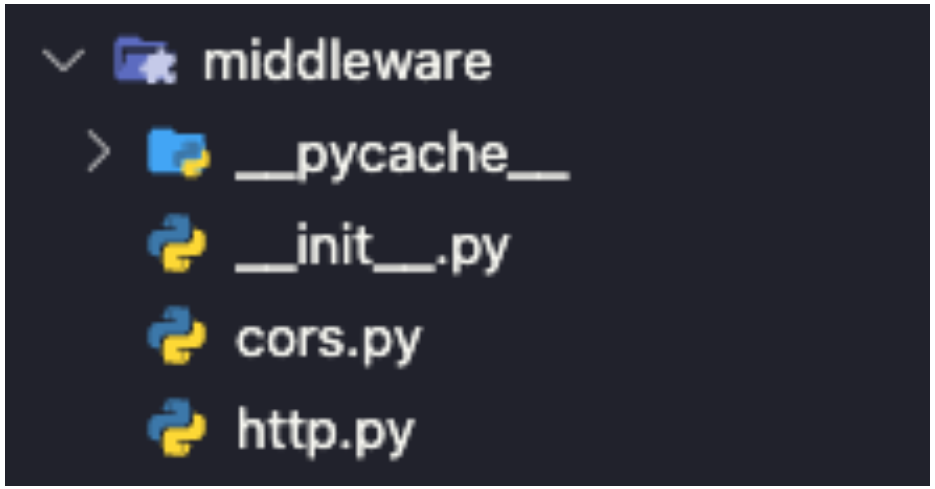
```python
import sys
import torch
import torchvision

from pathlib import Path
sys.path.append(str(Path(__file__).parent.parent))

from PIL import Image
from torch.nn import functional as F
from utils.logger import Logger
from config.catdog_cfg import CatDogDataConfig
from .catdog_model import CatDogModel

LOGGER = Logger(__file__, log_file="predictor.log")
LOGGER.log.info("Starting Model Serving")
```

catdog_predictor.py

```python
class Predictor:
    def __init__(self, model_name: str, model_weight: str, device: str = "cpu"):
        self.model_name = model_name
        self.model_weight = model_weight
        self.device = device
        self.load_model()
        self.create_transform()

    async def predict(self, image):
        pil_img = Image.open(image)

        if pil_img.mode == 'RGBA':
            pil_img = pil_img.convert('RGB')

        transformed_image = self.transforms_(pil_img).unsqueeze(0)
        output = await self.model_inference(transformed_image)
        probs, best_prob, predicted_id, predicted_class = self.output2pred(output)

        LOGGER.log_model(self.model_name)
        LOGGER.log_response(best_prob, predicted_id, predicted_class)

        torch.cuda.empty_cache()

        resp_dict = {
            "probs":probs,
            "best_prob": best_prob,
            "predicted_id": predicted_id,
            "predicted_class": predicted_class,
            "predictor_name": self.model_name
        }

        return resp_dict
```

Model Deployment

❖ Build API Step 6: Define middleware folder



We declare each Middleware in separated files

```python
from fastapi.middleware.cors import CORSMiddleware

def setup_cors(app):
    app.add_middleware(
        CORSMiddleware,
        allow_origins=["*"],
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )
```

cors.py

Model Deployment

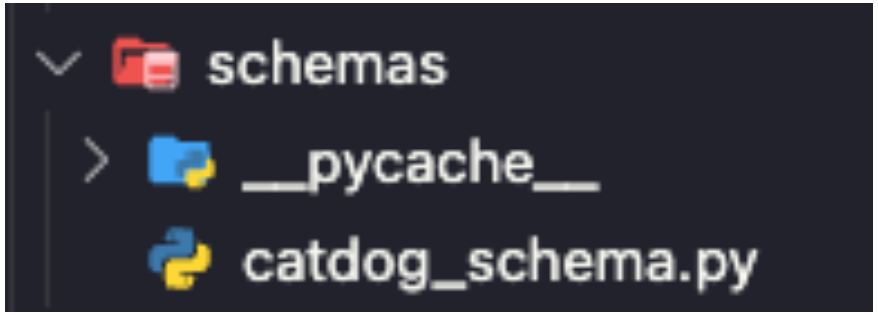
❖ Build API Step 6: Define middleware folder

http.py

```python
import time
import sys

from pathlib import Path
sys.path.append(str(Path(__file__).parent.parent))

from starlette.middleware.base import BaseHTTPMiddleware
from starlette.requests import Request
from utils.logger import Logger

LOGGER = Logger(__file__, log_file="http.log")

class LogMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request: Request, call_next):
        start_time = time.time()
        response = await call_next(request)
        process_time = time.time() - start_time
        LOGGER.log.info(
            f"{request.client.host} - \"{request.method} {request.url.path} {request.scope['http_version']}\" {response.status_code} {process_time:.2f}s")

        return response
```

Model Deployment

❖ Build API Step 7: Define Schemas

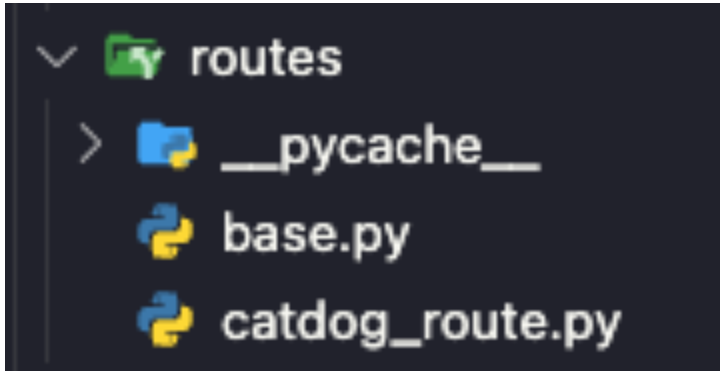


We define all Pydantic model in schemas folder

```python
from pydantic import BaseModel

class CatDogResponse(BaseModel):
    probs: list = []
    best_prob: float = -1.0
    predicted_id: int = -1
    predicted_class: str = ""
    predictor_name: str = ""
```

Model Deployment

❖ Build API Step 8: Define routes



base.py 1 ×

Basic_Structure > routes > base.py > ...

Click here to ask Blackbox to help you code faster

```python
from fastapi import APIRouter
from .catdog_route import router as catdog_cls_route

router = APIRouter()
router.include_router(catdog_cls_route, prefix="/catdog_classification")
```

```python
import sys
from pathlib import Path
sys.path.append(str(Path(__file__).parent))

from fastapi import File, UploadFile
from fastapi import APIRouter
from schemas.catdog_schema import CatDogResponse
from config.catdog_cfg import ModelConfig
from models.catdog_predictor import Predictor

router = APIRouter()
predictor = Predictor(
    model_name=ModelConfig.MODEL_NAME,
    model_weight=ModelConfig.MODEL_WEIGHT,
    device=ModelConfig.DEVICE
)

@router.post("/predict")
async def predict(file_upload: UploadFile = File(...)):
    response = await predictor.predict(file_upload.file)

    return CatDogResponse(**response)
```

Model Deployment

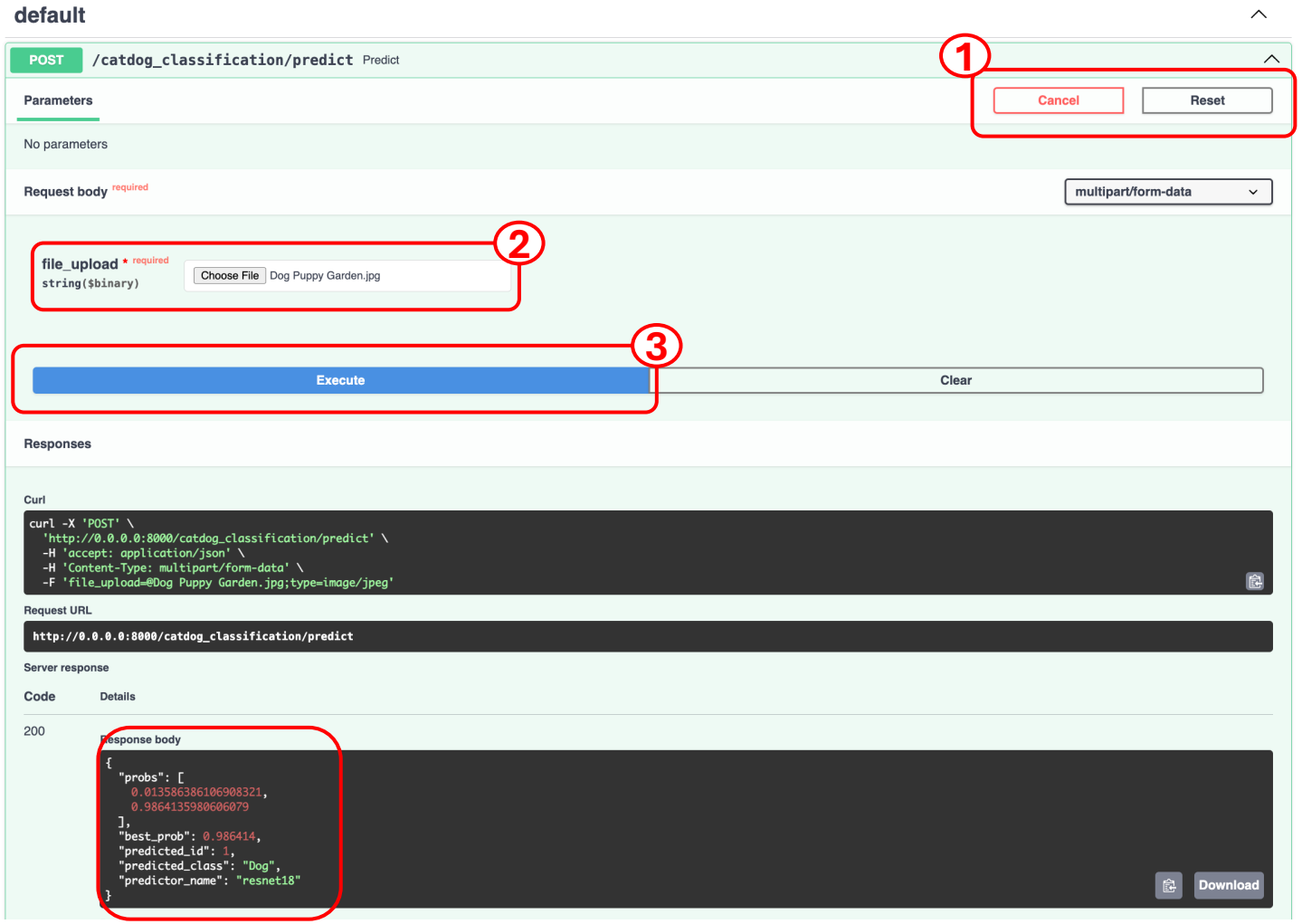
❖ Build API Step 9: Define app.py and server.py

```python
import sys
from pathlib import Path
sys.path.append(str(Path(__file__).parent))

from fastapi import FastAPI
from middleware import LogMiddleware, setup_cors
from routes.base import router

app = FastAPI()

app.add_middleware(LogMiddleware)
setup_cors(app)
app.include_router(router)
```

```python
import uvicorn

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

Model Deployment

❖ Result

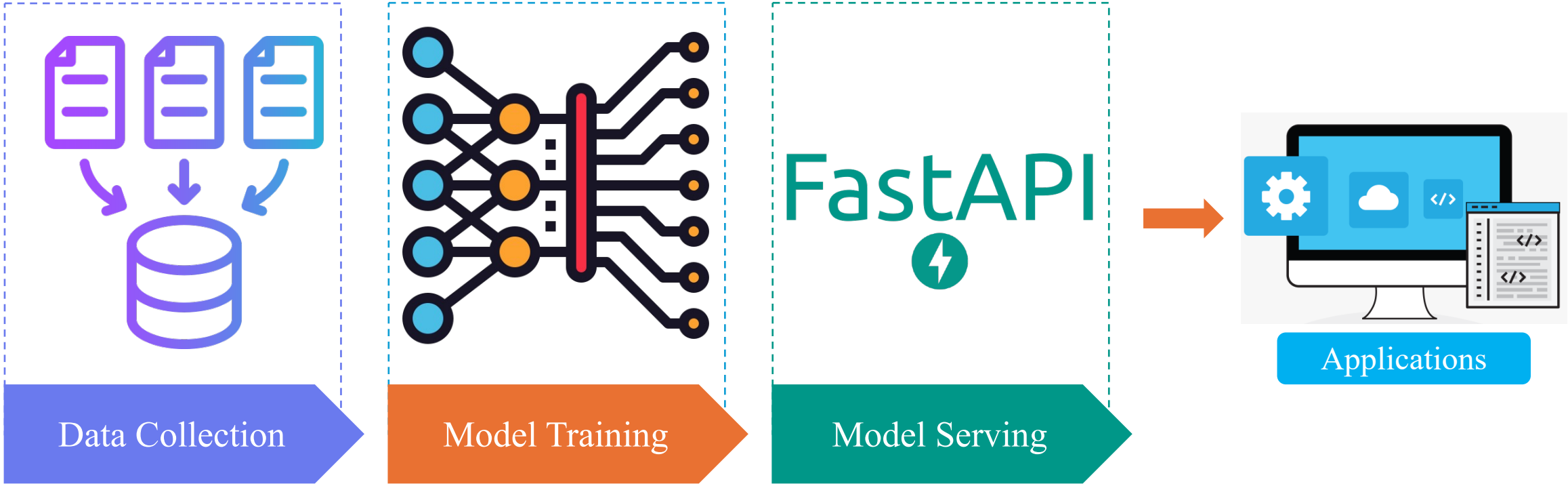


AI VIETNAM
All-in-One Course
(TA Session)

Model Deployment (MLOps)

Model Deployment (MLOps)

❖ Getting Started



Model Deployment (MLOps)

❖ Data Drift Problem









Data Collection

User input

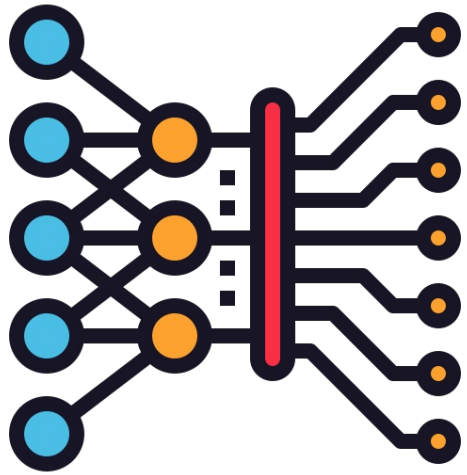
User input

Applications

Model Deployment (MLOps)

❖ Data Drift Solution

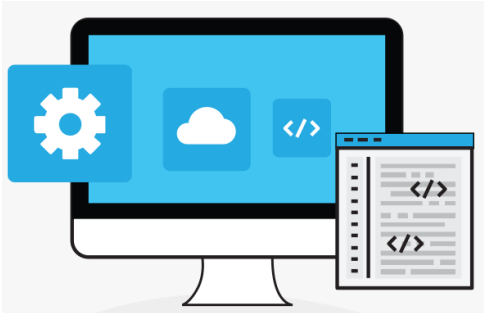
More Data Collection



Model Training

FastAPI

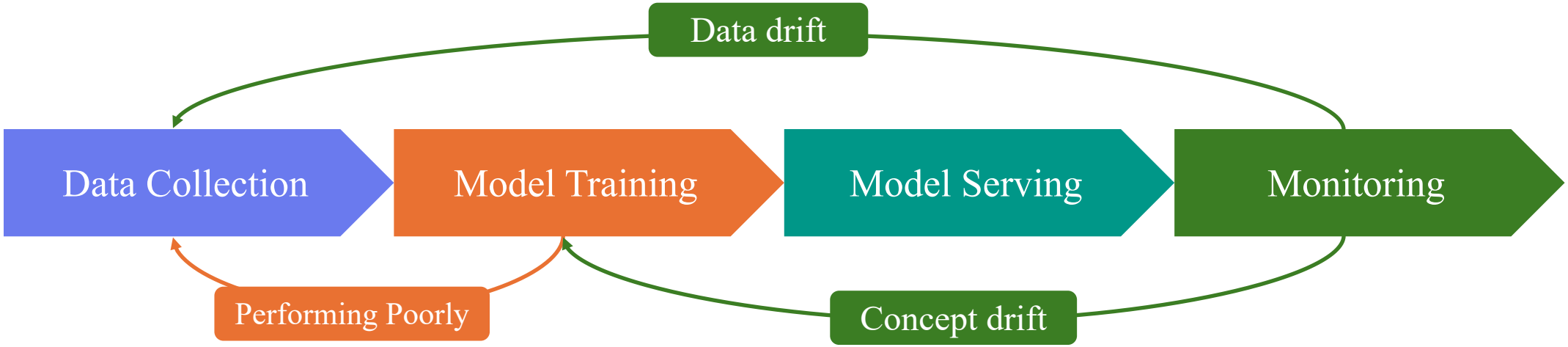
Model Serving



Applications

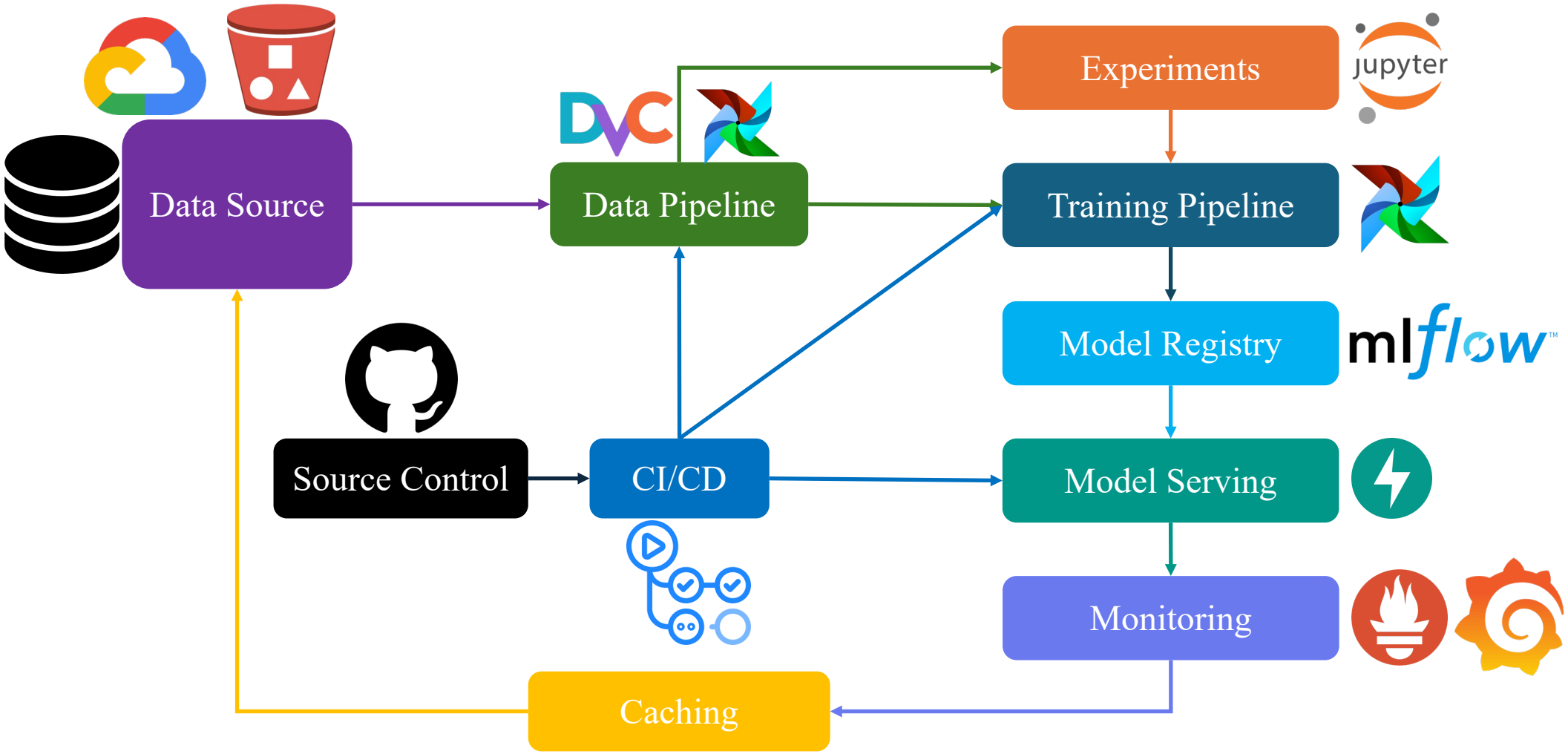
Model Deployment (MLOps)

❖ Machine Learning Life Cycle



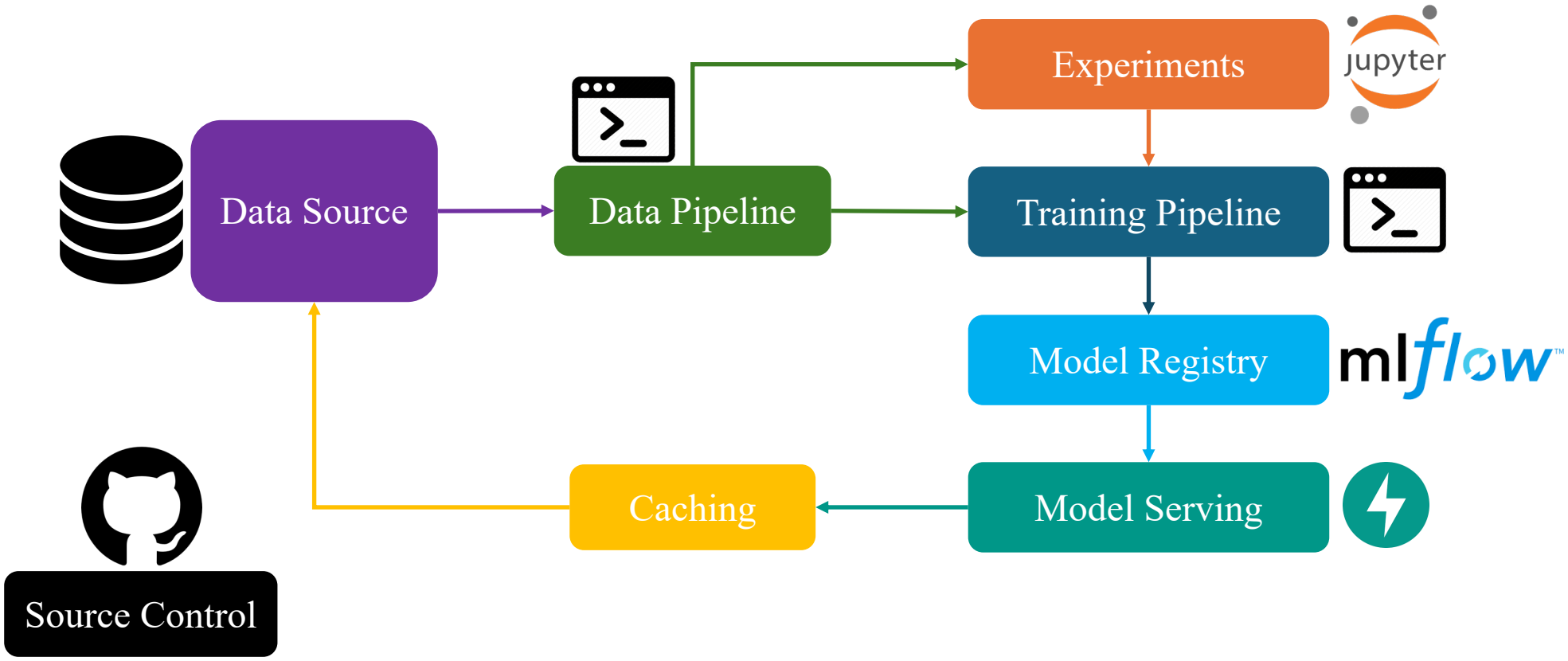
Model Deployment (MLOps)

❖ Machine Learning Operations (MLOps) pipeline



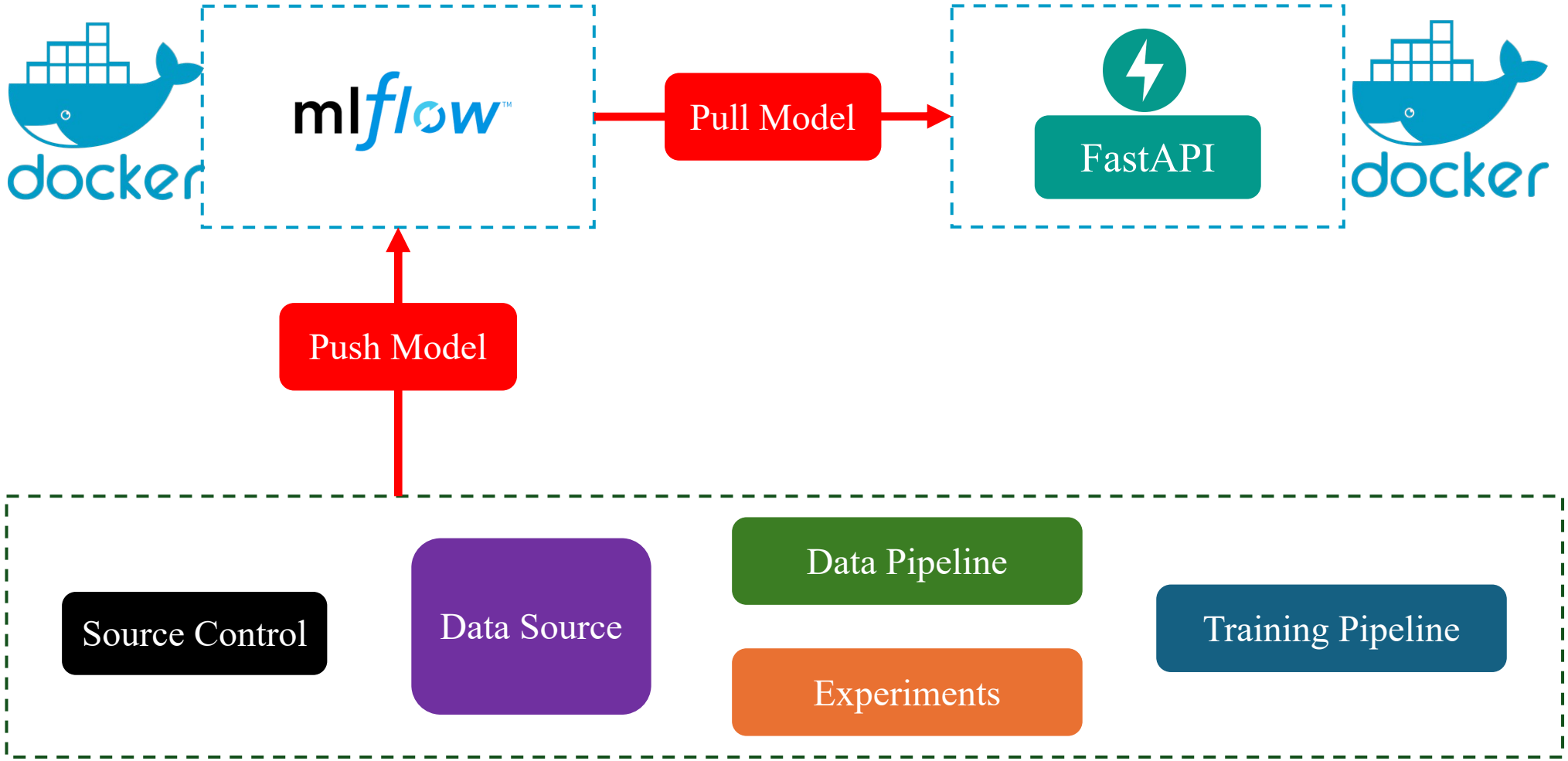
Model Deployment (MLOps)

❖ MLOps simple pipeline



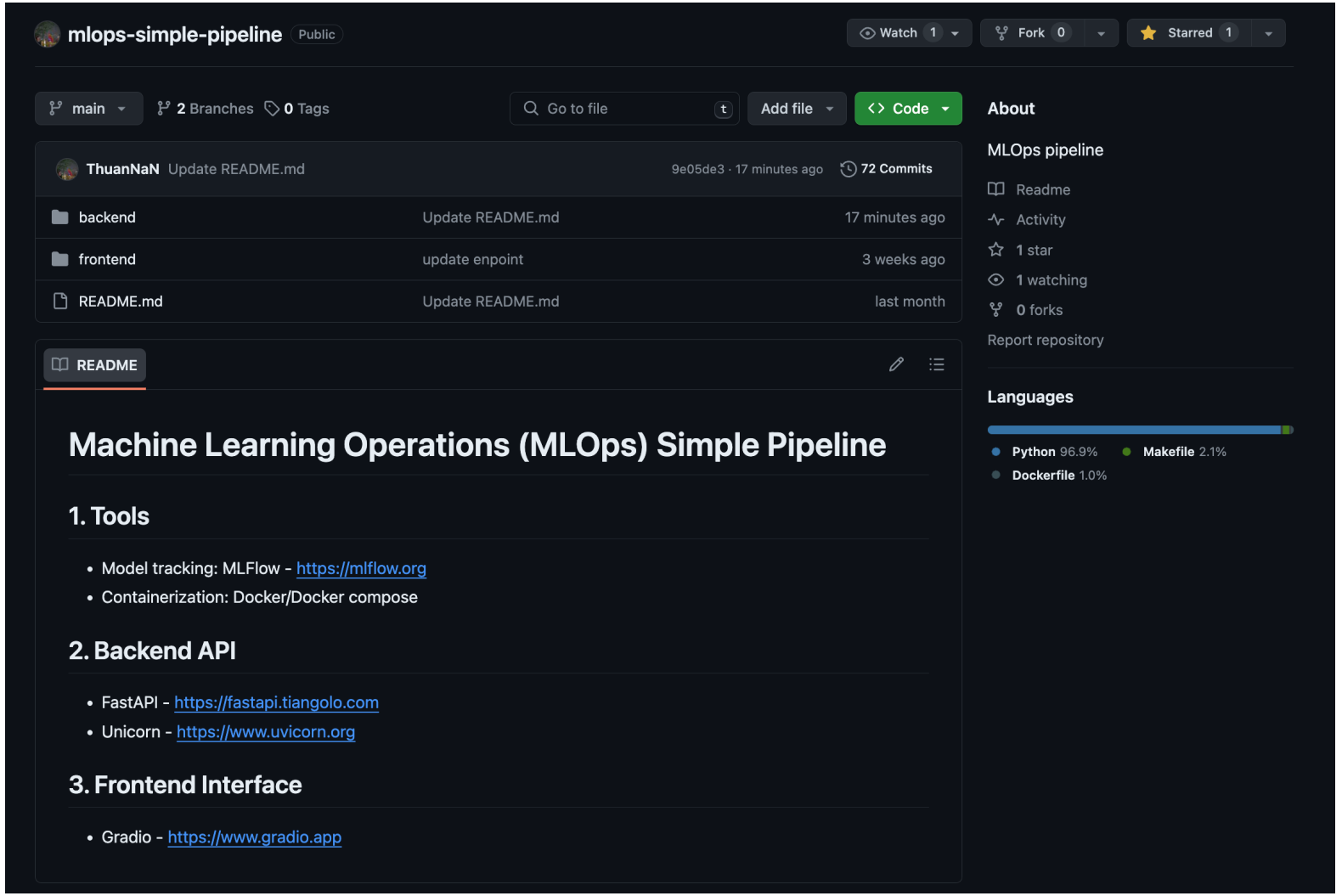
Model Deployment (MLOps)

❖ Runing steps



Model Deployment (MLOps)

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



Model Deployment (MLOps)

❖ Start MLflow server

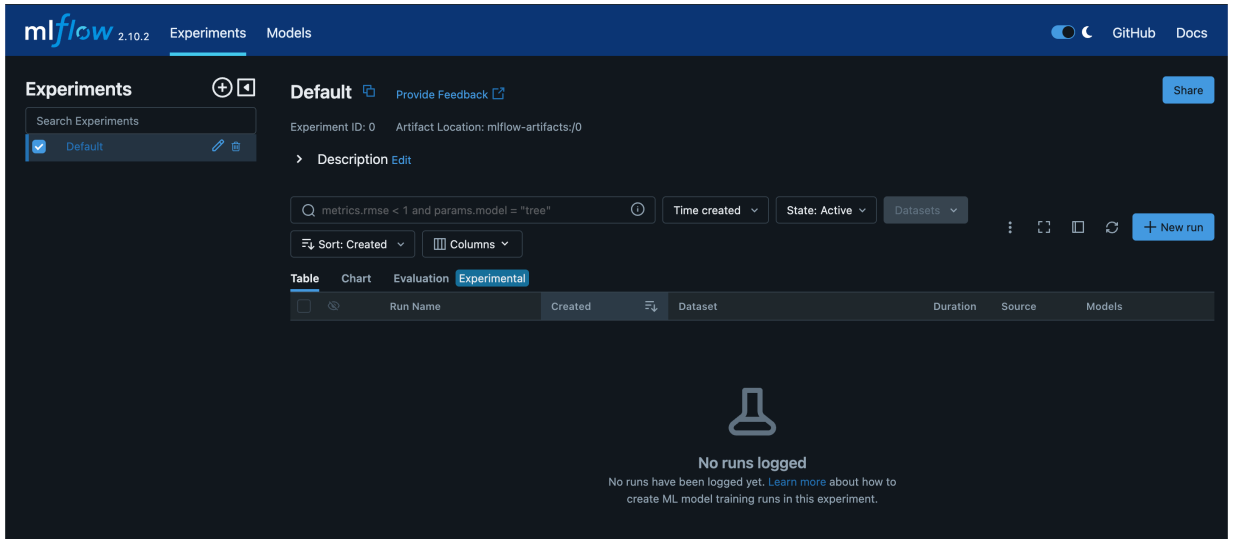
Makefile command

make mlflow_up

Use MLflow for tracking and versioning the trained model.

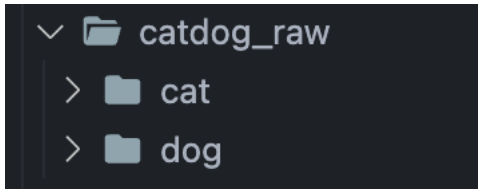
localhost:8000

```
(mlops_pipleline) thuannd@thuan-ubuntu:/mnt/HDD/Repositories/MLOps_simple_pipeline/backend$ make mlflow_up
docker compose -f ./docker/mlflow/docker-compose.yml up -d
WARN[0000] /mnt/HDD/Repositories/MLOps_simple_pipeline/backend/docker/mlflow/docker-compose.yml: `version` is obsolete
[+] Building 54.2s (7/7) FINISHED                                                                                    docker:default
 => [mlflow internal] load build definition from Dockerfile                                                                   0.0s
 => => transferring dockerfile: 123B                                                                                          0.0s
 => [mlflow internal] load metadata for docker.io/library/python:3.11-slim                                                    2.7s
 => [mlflow internal] load .dockerignore                                                                                      0.0s
 => => transferring context: 64B                                                                                              0.0s
 => [mlflow 1/3] FROM docker.io/library/python:3.11-slim@sha256:90f8795536170fd08236d2ceb74fe7065dbf74f738d8b84bfbf263656654dc9b  4.1s
 => => resolve docker.io/library/python:3.11-slim@sha256:90f8795536170fd08236d2ceb74fe7065dbf74f738d8b84bfbf263656654dc9b      0.0s
 => => sha256:346e2b922dbd8f853cd1a63f142290fc7449b0a59b0e2b600ef7dc98ca5ab436 1.37kB / 1.37kB                                0.0s
 => => sha256:23bae5f8ae960b7056414daaa84966ce8bb5663c370204adef0596ed8d1e7f03 6.93kB / 6.93kB                                0.0s
 => => sha256:8a1e25ce7c4f75e372e9884f8f7b1bedcfe4a7a7d452eb4b0a1c7477c9a90345 29.12MB / 29.12MB                              2.3s
 => => sha256:1103112ebfc46e01c0f35f3586e5a39c6a9ffa32c1a362d4d5f20e3783c6fdd7 3.51MB / 3.51MB                                2.4s
 => => sha256:b4b80ef7128dc9bd114e4f7ab83bdb6228f6d91494c12ab793563211fcc32e84 12.87MB / 12.87MB                              3.8s
 => => sha256:90f8795536170fd08236d2ceb74fe7065dbf74f738d8b84bfbf263656654dc9b 1.65kB / 1.65kB                                0.0s
 => => extracting sha256:8a1e25ce7c4f75e372e9884f8f7b1bedcfe4a7a7d452eb4b0a1c7477c9a90345                                      0.4s
 => => sha256:cc7f04ac52f8a3bad5b4d1f6179041d3ccca1967aace3faa38303c99f9ed7e0c 243B / 243B                                    2.7s
 => => sha256:87b8bf94a2ace2b005da41fb26fa80a63c02887d1aebe29c44331381ce313388 3.41MB / 3.41MB                                2.9s
 => => extracting sha256:1103112ebfc46e01c0f35f3586e5a39c6a9ffa32c1a362d4d5f20e3783c6fdd7                                      0.0s
 => => extracting sha256:b4b80ef7128dc9bd114e4f7ab83bdb6228f6d91494c12ab793563211fcc32e84                                      0.1s
 => => extracting sha256:cc7f04ac52f8a3bad5b4d1f6179041d3ccca1967aace3faa38303c99f9ed7e0c                                      0.0s
 => => extracting sha256:87b8bf94a2ace2b005da41fb26fa80a63c02887d1aebe29c44331381ce313388                                      0.1s
 => [mlflow 2/3] WORKDIR /mlflow/                                                                                             0.1s
 => [mlflow 3/3] RUN pip install --no-cache-dir mlflow==2.10.2                                                               46.0s
 => [mlflow] exporting to image                                                                                               1.2s
 => => exporting layers                                                                                                       1.2s
 => => writing image sha256:3ab775d3d0d28ad4660092a8eeec2515d69526879fb58f66be2dedf4da7d5620                                  0.0s
 => => naming to docker.io/library/mlflow-mlflow                                                                              0.0s
[+] Running 1/2
 ⠿ Network mlflow_default    Created                                                                                          0.3s
 ✔ Container mlflow_server   Started                                                                                          0.2s
```



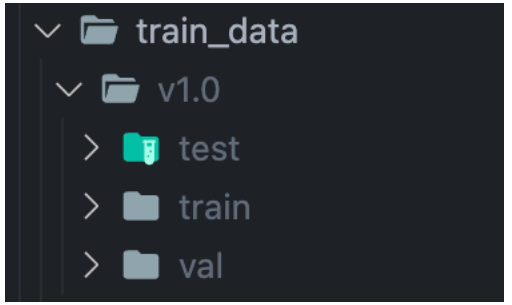
Model Deployment (MLOps)

❖ Prepare dataset for training



Bash command

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



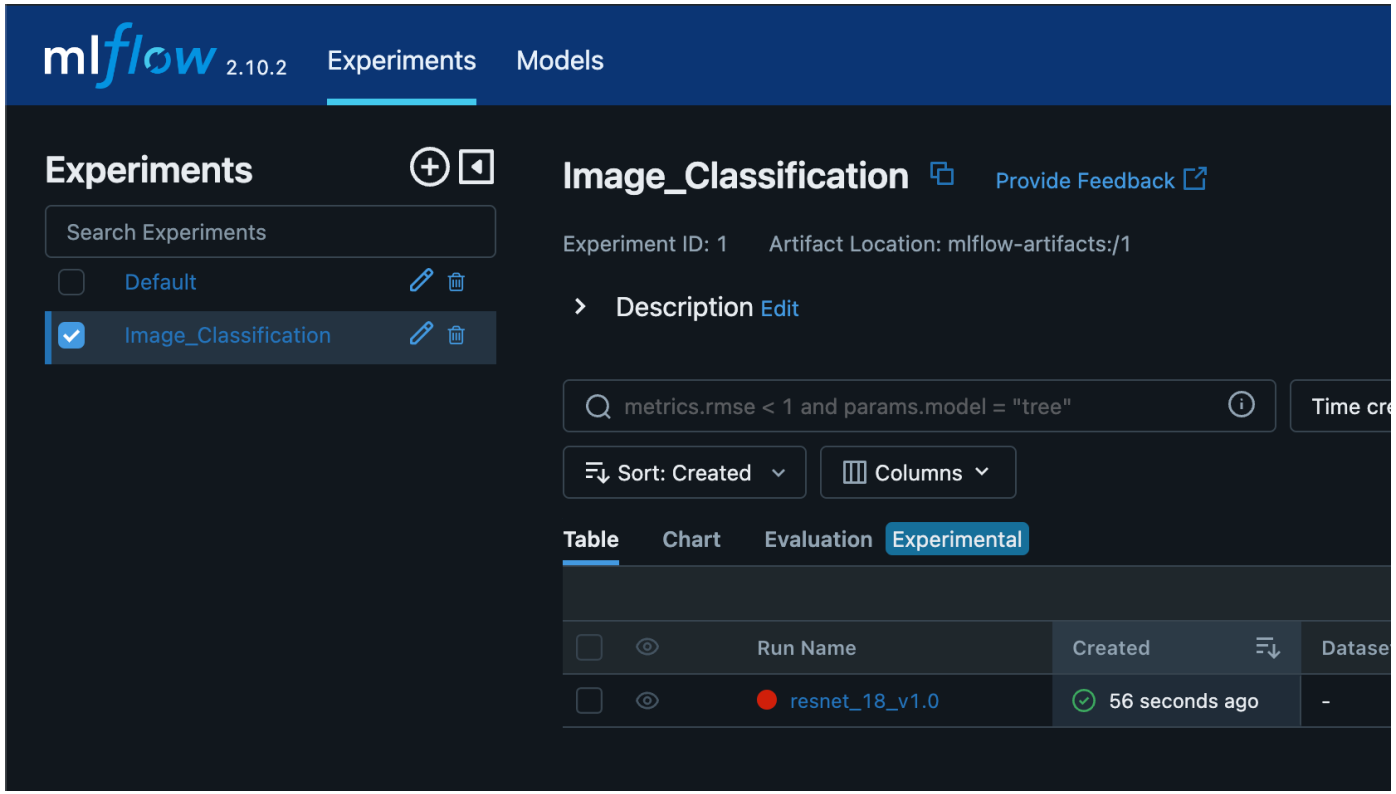
Split data into train/val/test for training and versioning based on folder name.

Model Deployment (MLOps)

❖ Training model (resnet_18)

Bash command

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



localhost:8000

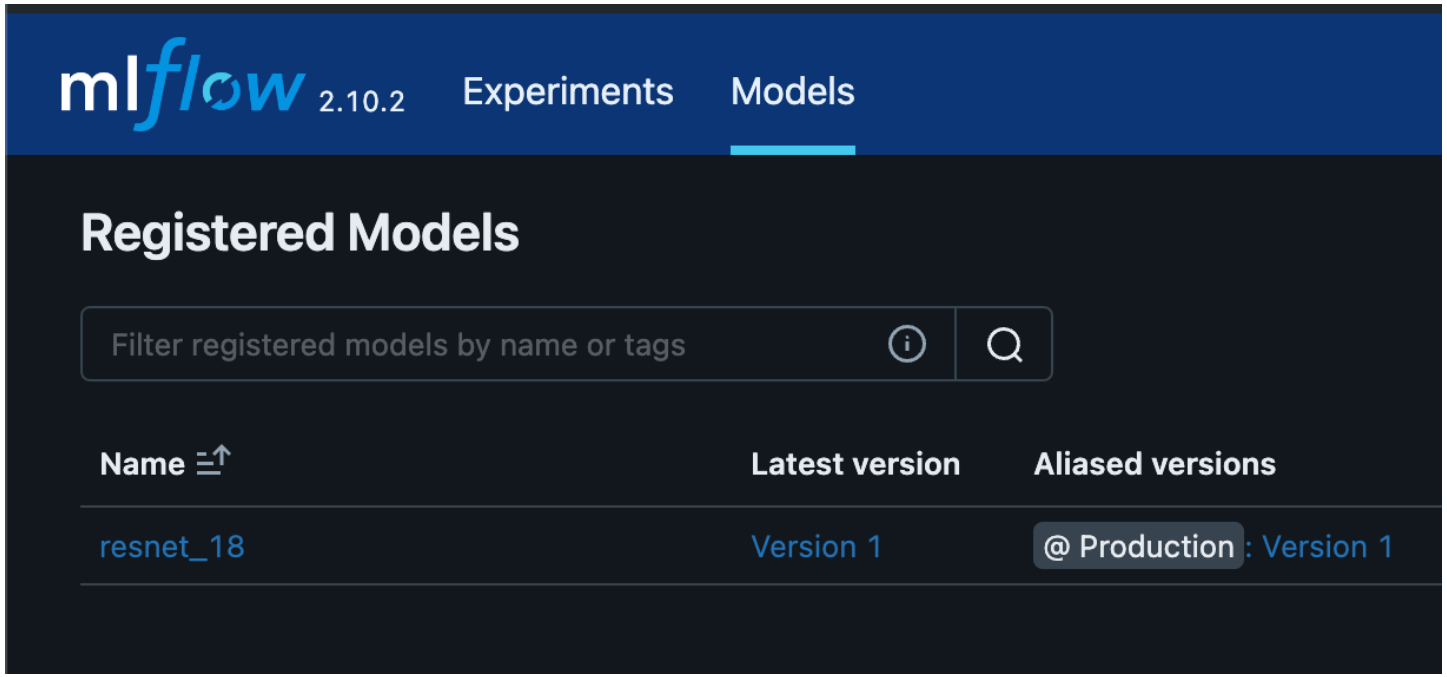
Training model with dataset version 1.0. The model trained will be saved to Mlflow server.

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
```



localhost:8000

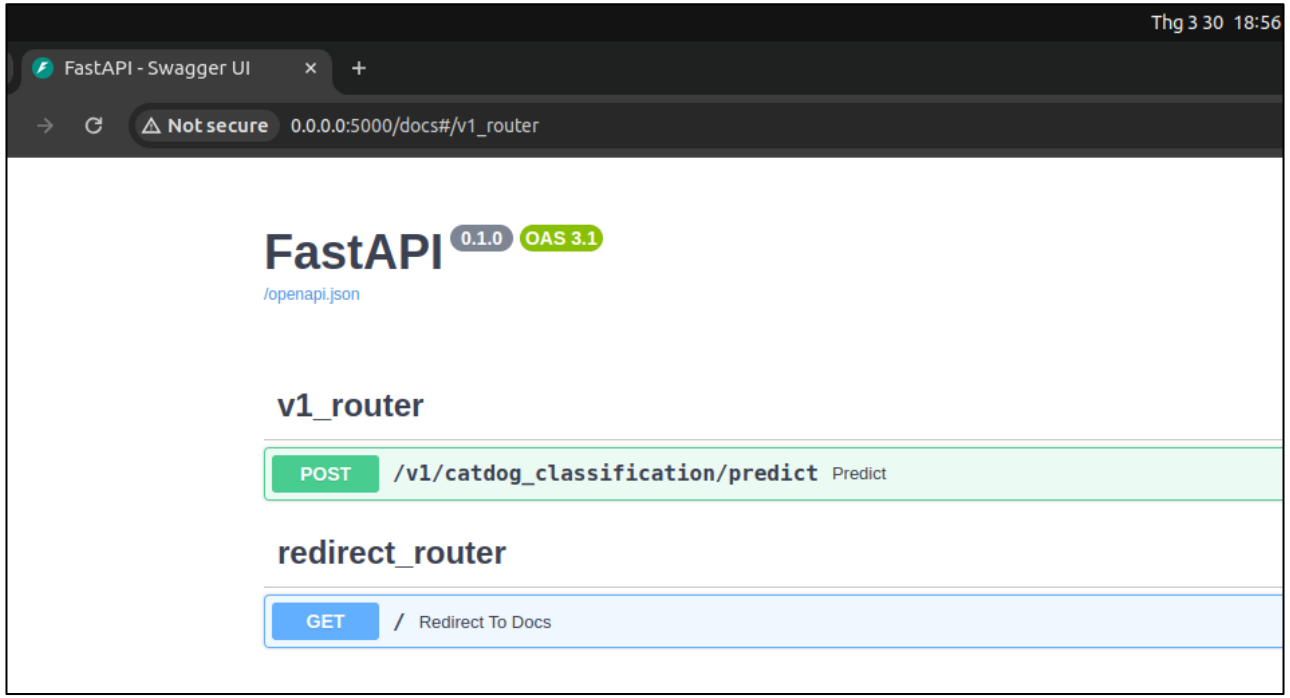
Registry model trained which have highest metric to serving.

Model Deployment (MLOps)

❖ Serving best model

Makefile command

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



localhost:5000

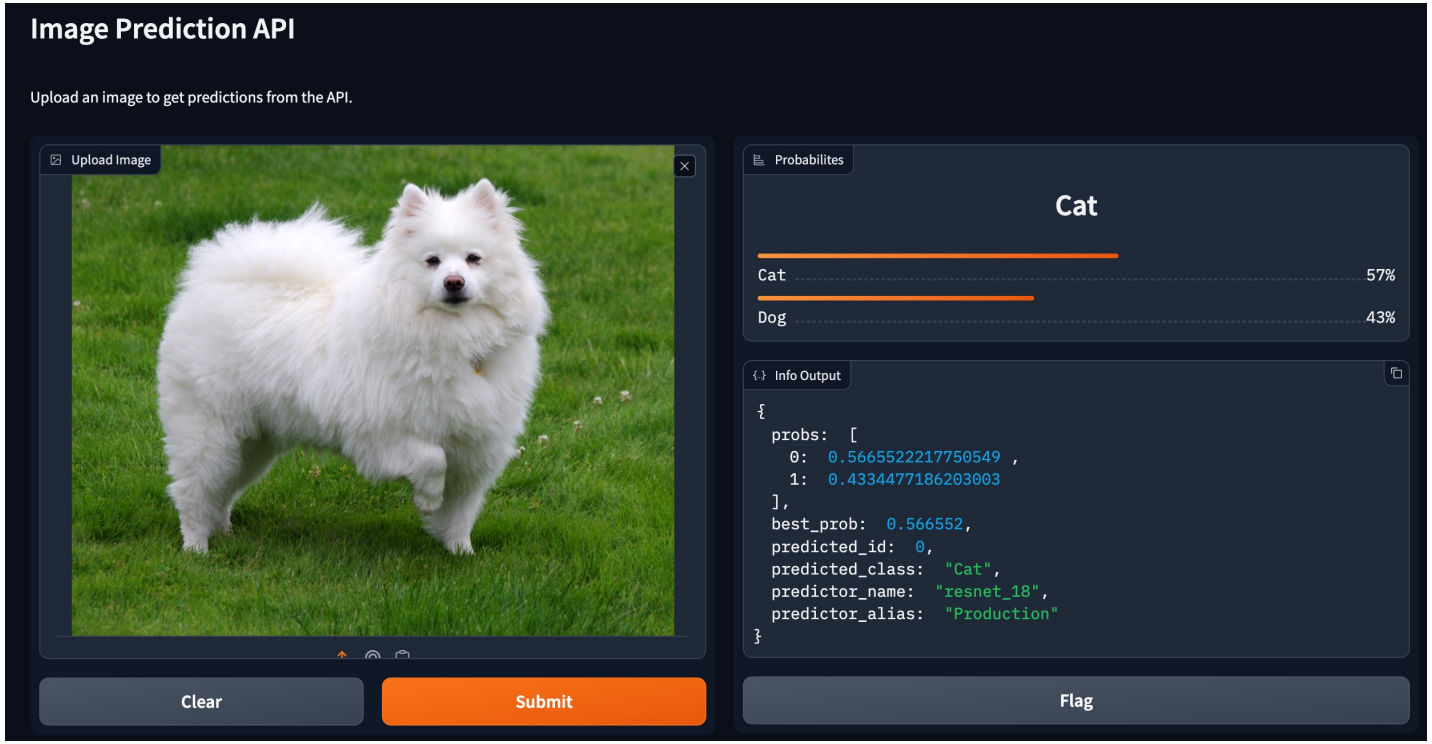
Pulling the best model from MLflow and serving at port 5000 by FastAPI (take a minute for building docker container).

Model Deployment (MLOps)

❖ Test in web application

Bash command

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



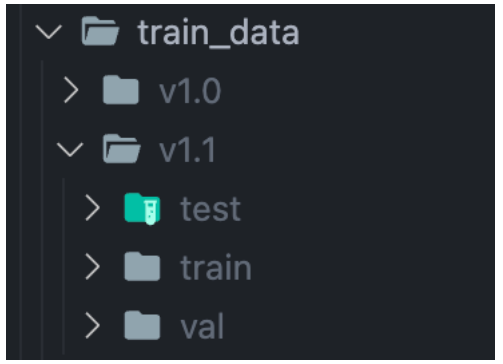
localhost:3000

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.

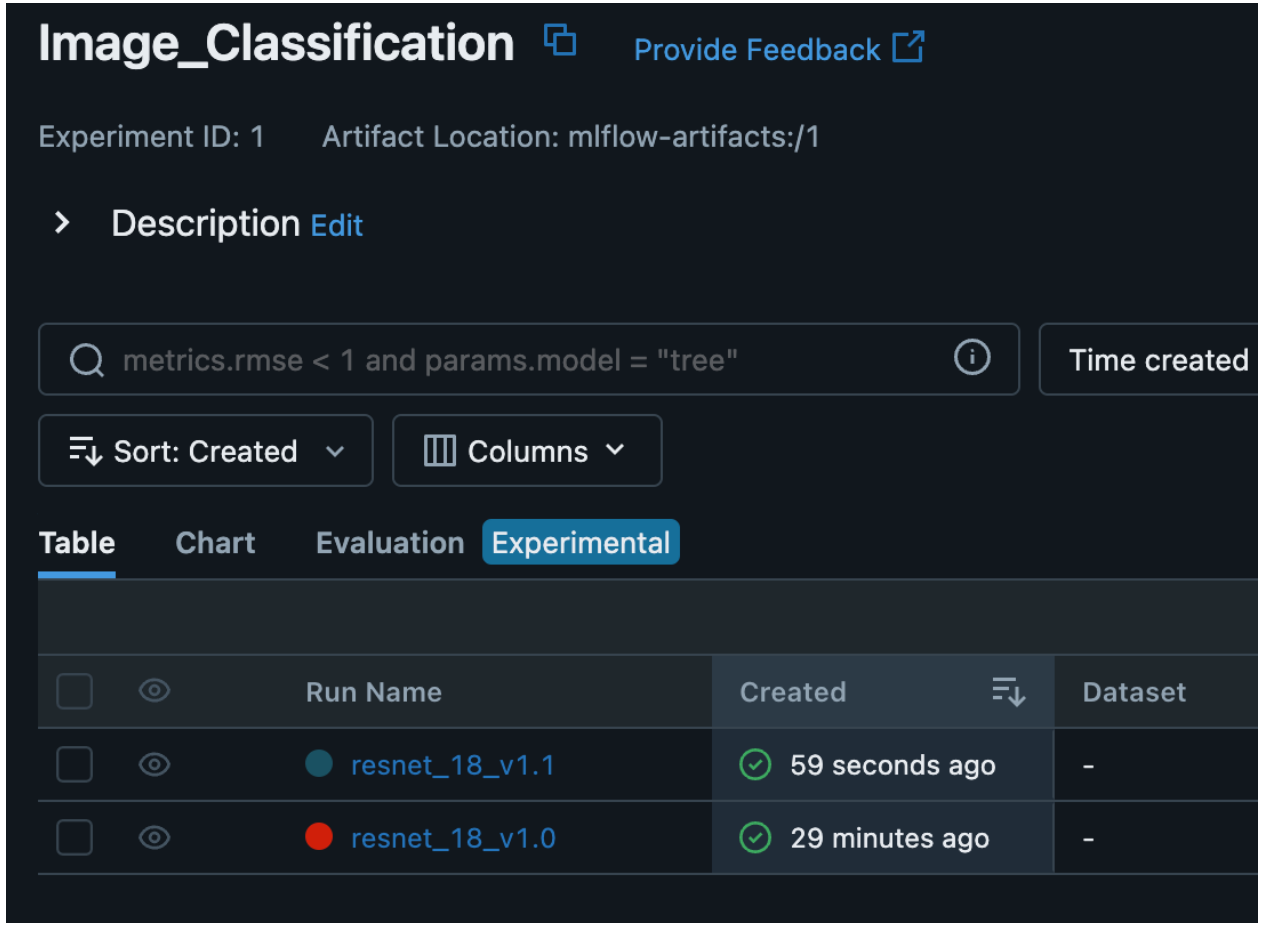
Model Deployment (MLOps)

❖ Training new model with new dataset

Bash command

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

Train renset_18 with v1.1 dataset



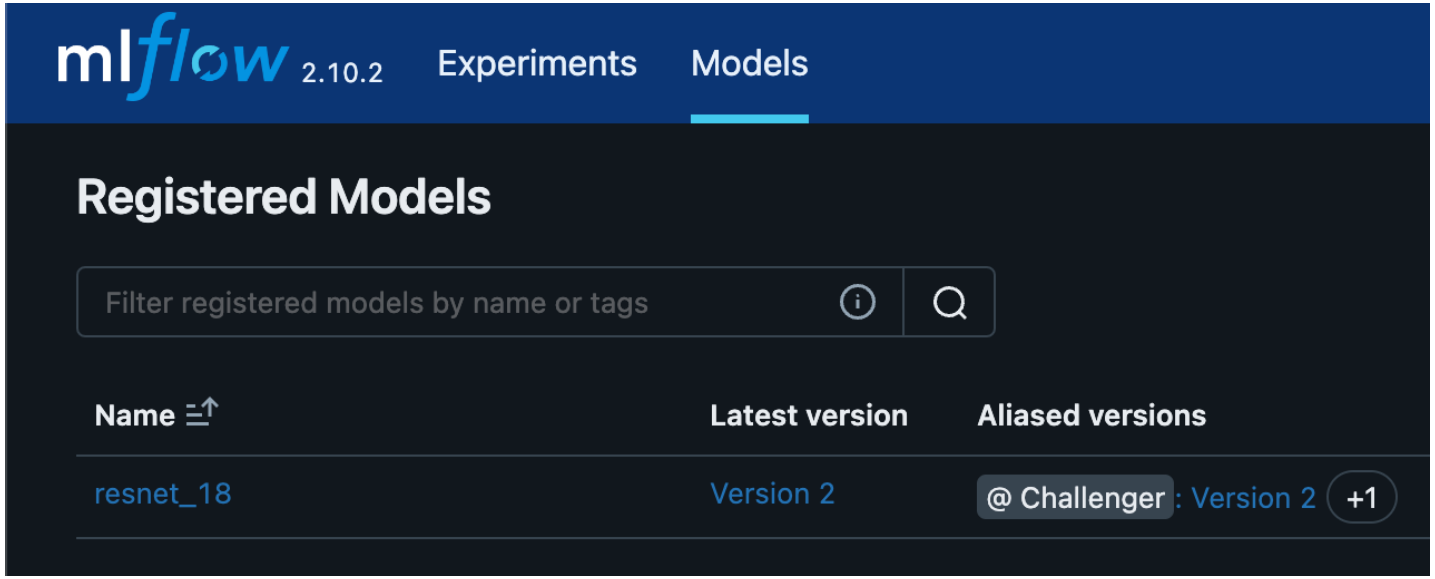
localhost:8000

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
```



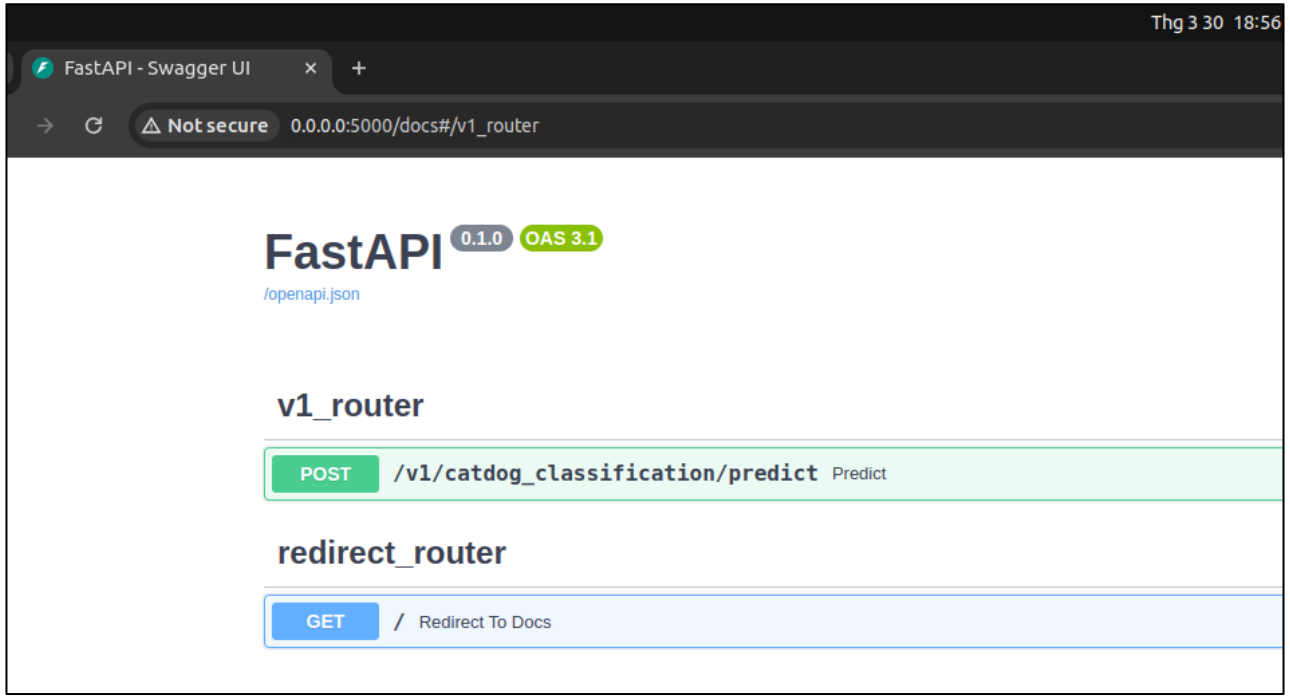
Registry model trained that version of dataset training is v1.1 and tagging it to “Challenger”.

Model Deployment (MLOps)

❖ Restart FastAPI

Makefile command

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

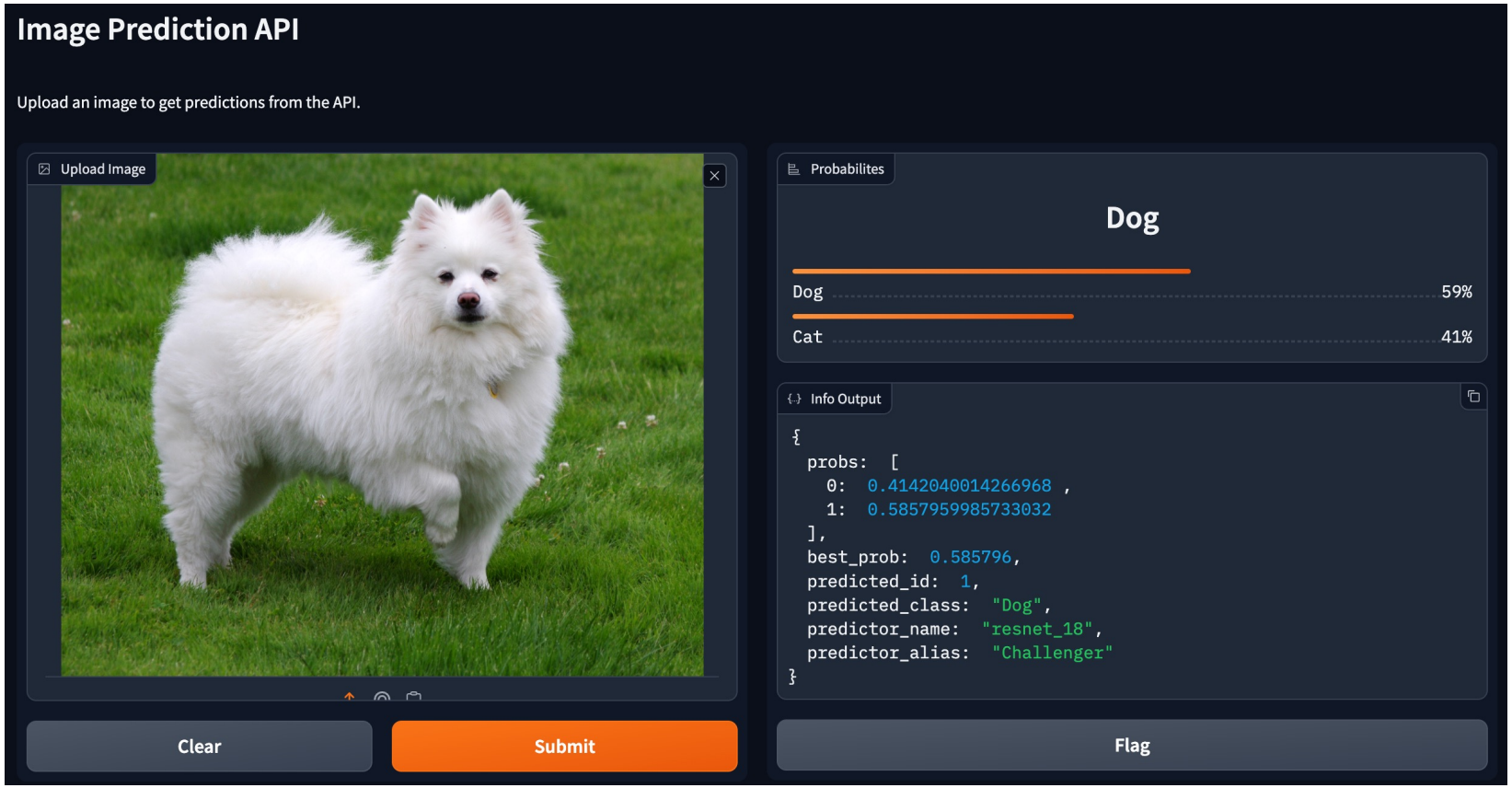


localhost:5000

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

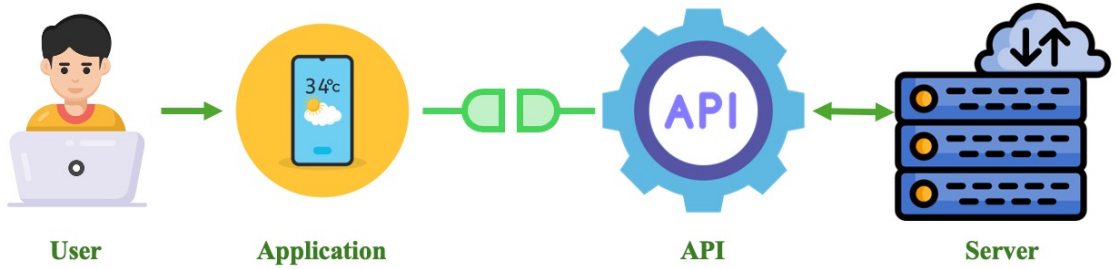


localhost:3000

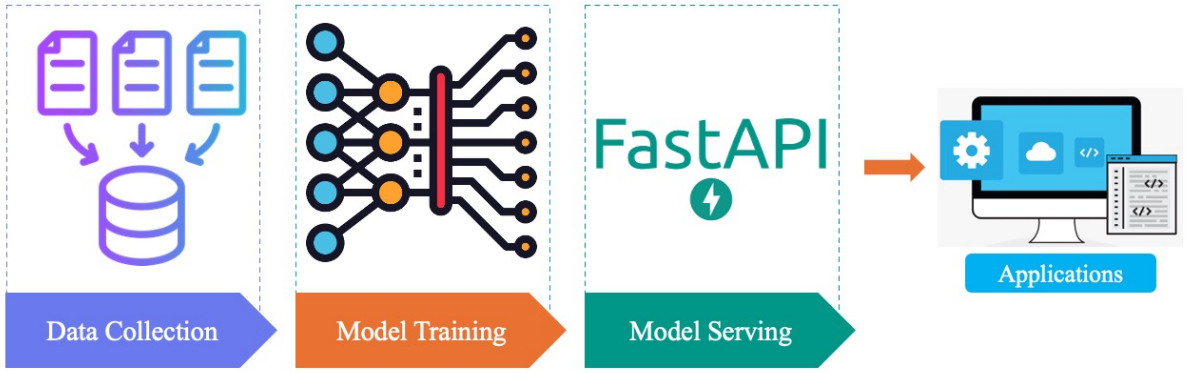
Summarization

Summarization

❖ What we have learned so far



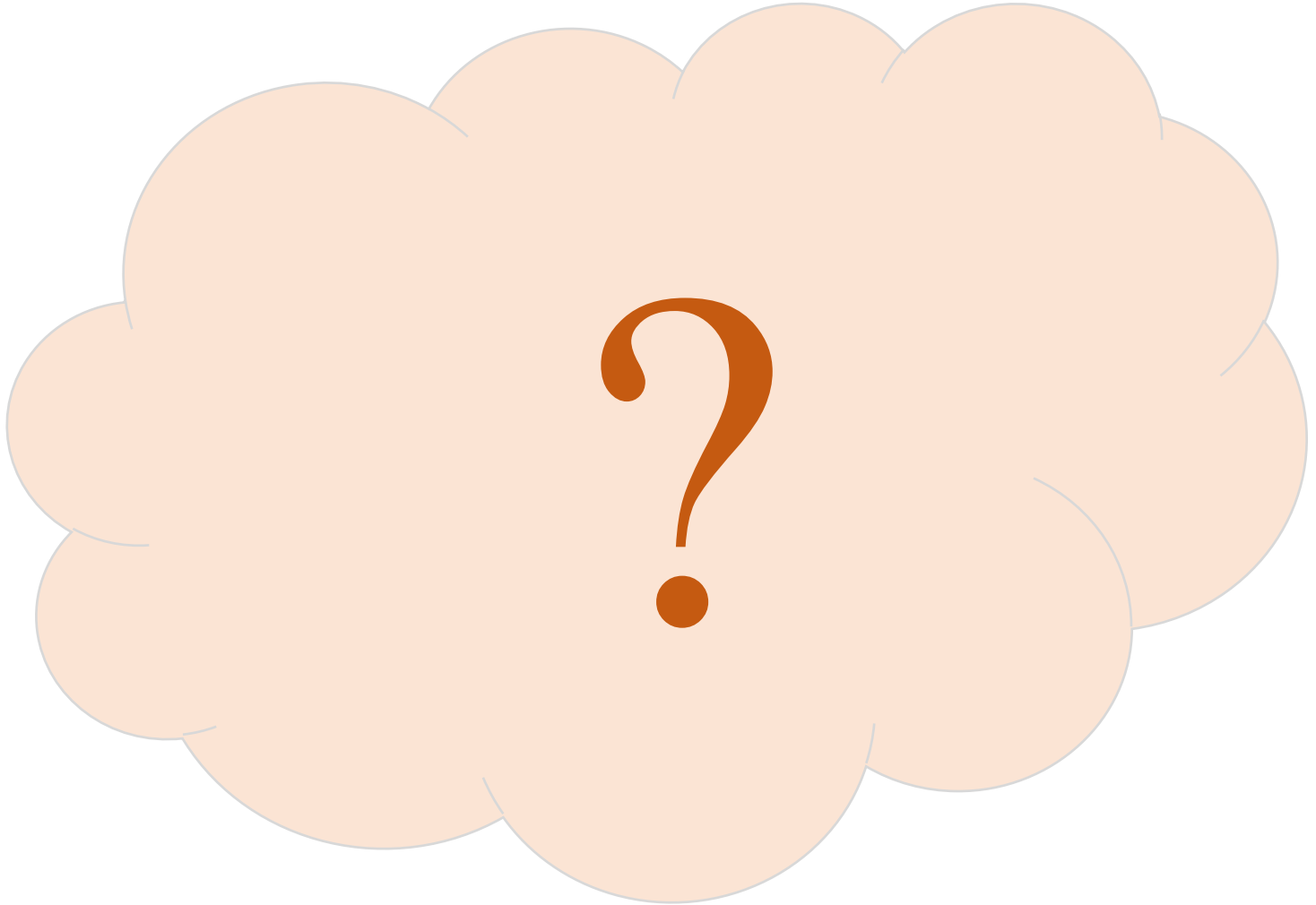
FastAPI



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




Thank you!