MINDSONAI

MACHINE LEARNING OPERATIONS

rishabh.io

Setting Up Research Env 001

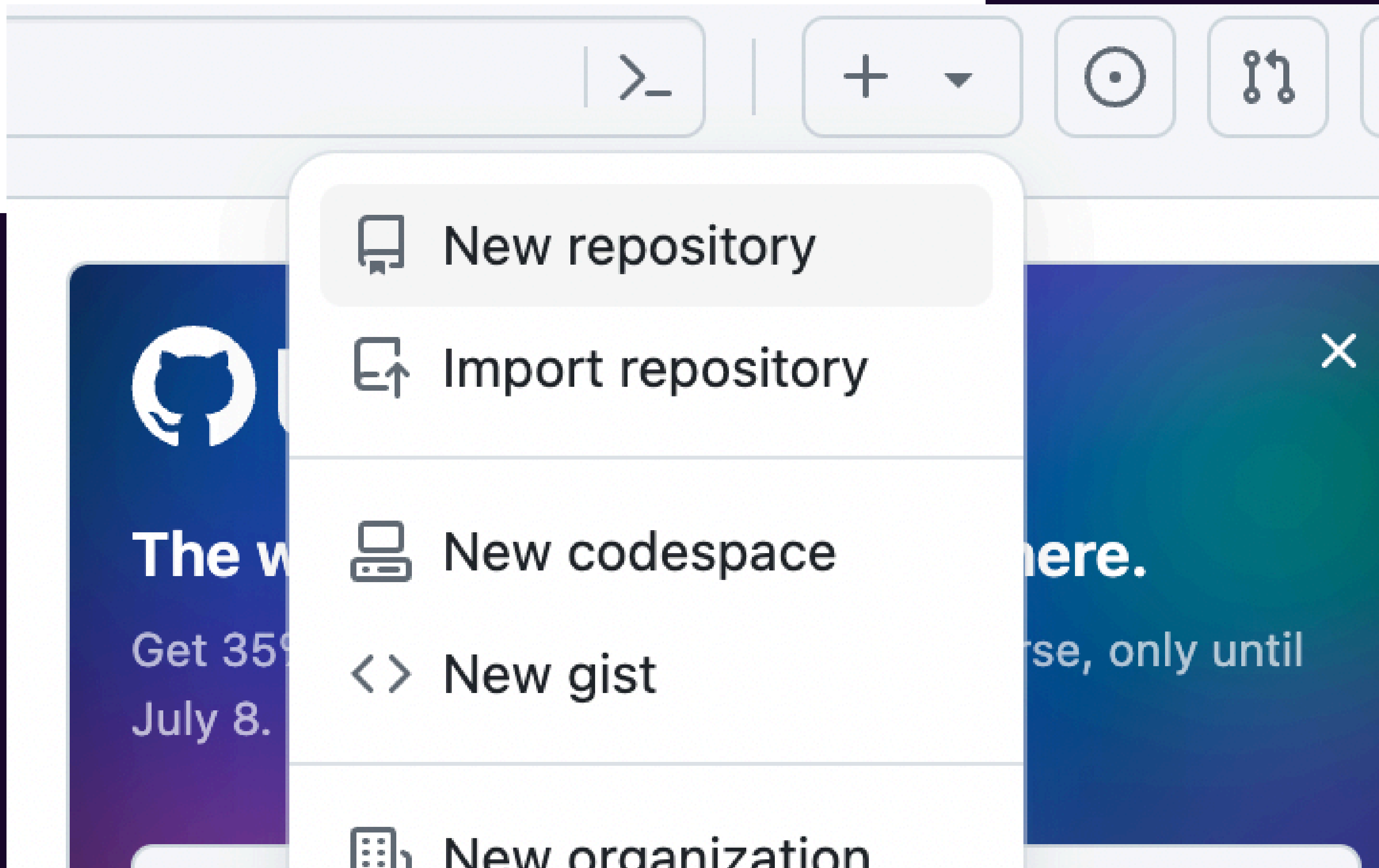
1

Go to https://www.github.com

Setting Up Research Env 002

2

Create a New Repository
ml_ops_24



Setting Up Research Env 003

3

Clone the Repository to Local Machine at Desired Location.

```
git clone https://<repo_url>
```

Setup Virtual Env 004

4

01 -> python -m venv ops_env

02 -> source ops_env/bin/activate

What's venv and how does it work ?

Add the Project Dependencies 005

5

Create a file **requirements.txt**

```
pandas
numpy
matplotlib
jupyterlab
sklearn
transformers
fastapi
```

Install Project Dependencies 006

6

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

Launch Jupyter 007

7

> jupyer lab

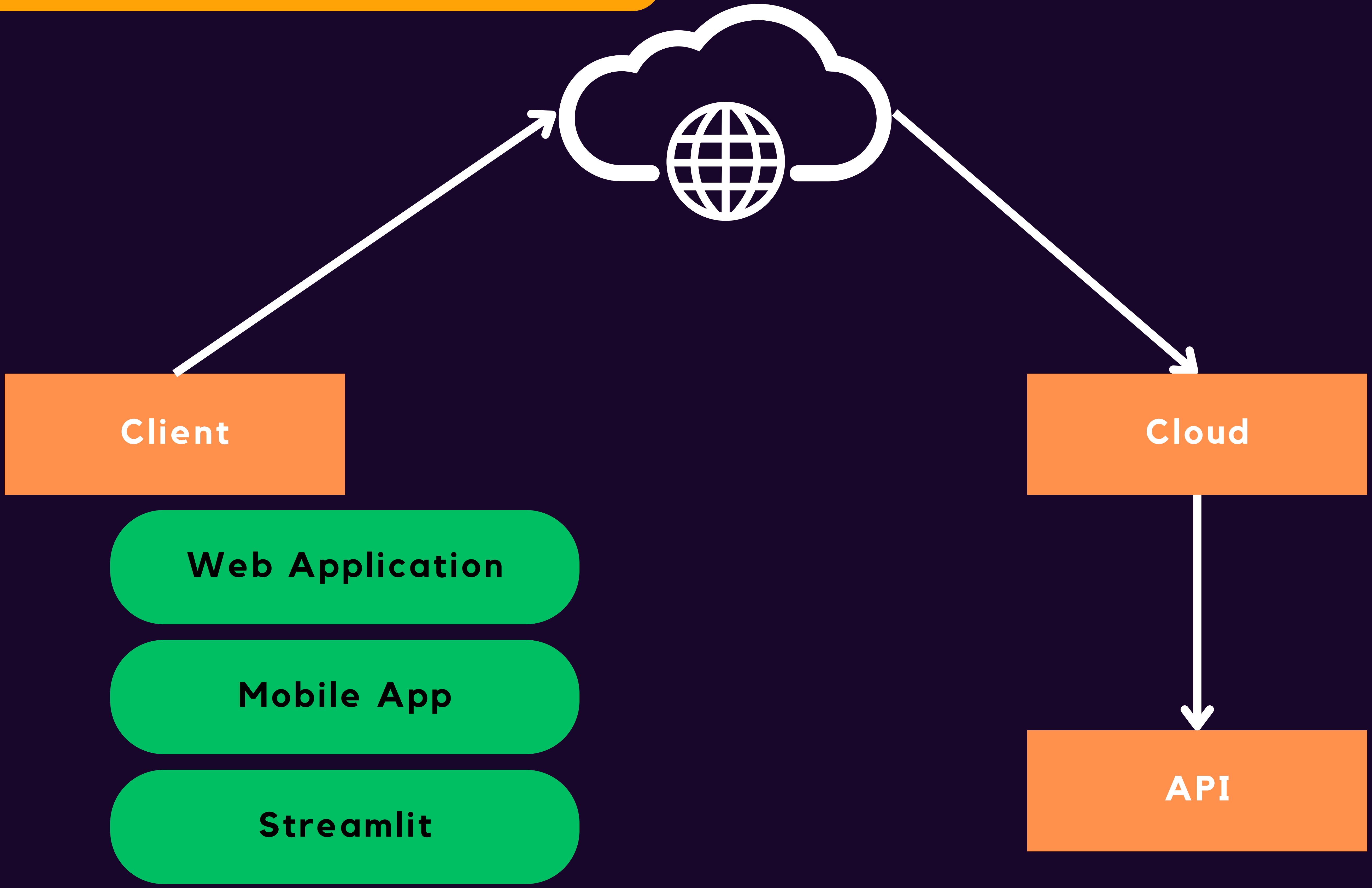
> navigate to: http://localhost:8000

8

> Convert Your Project into a Modular Template

> Once your model is performing fairly well, you're ready to deploy it.

Path towards Production

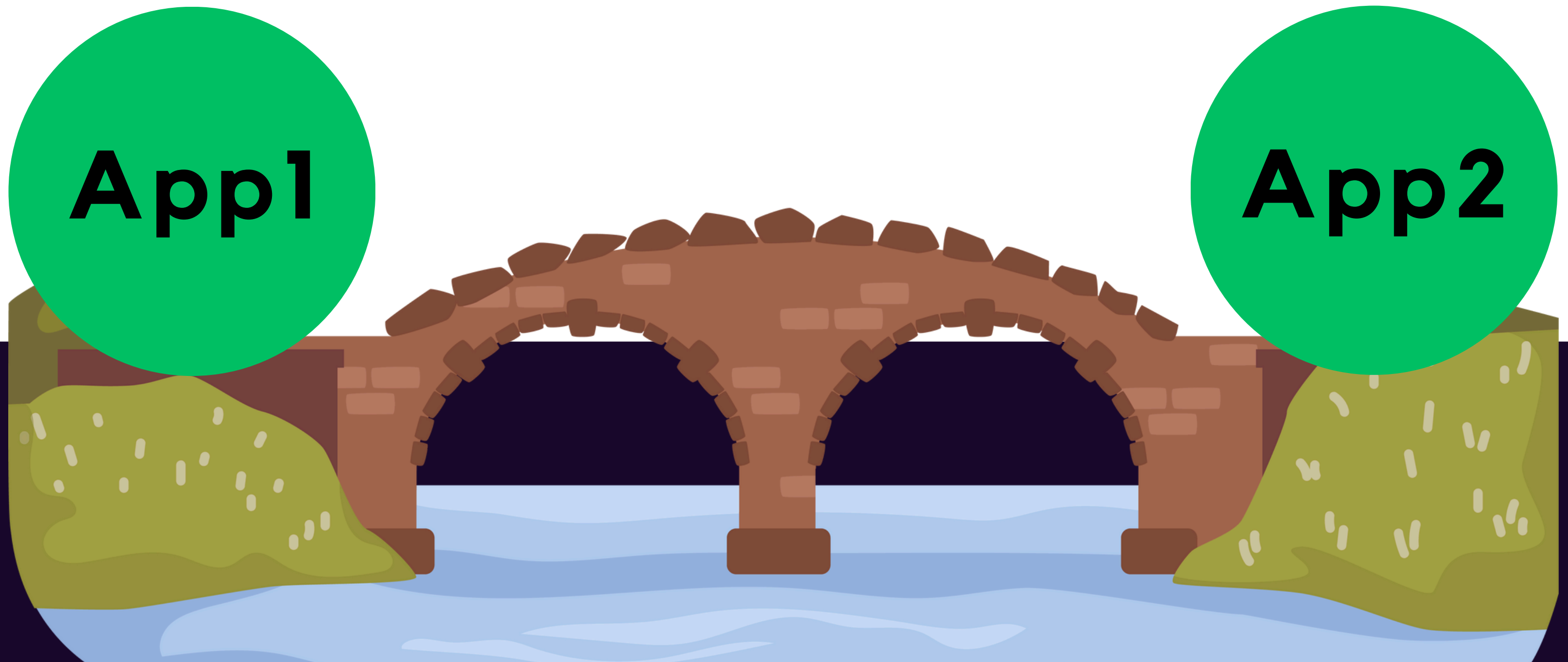


What is an API?

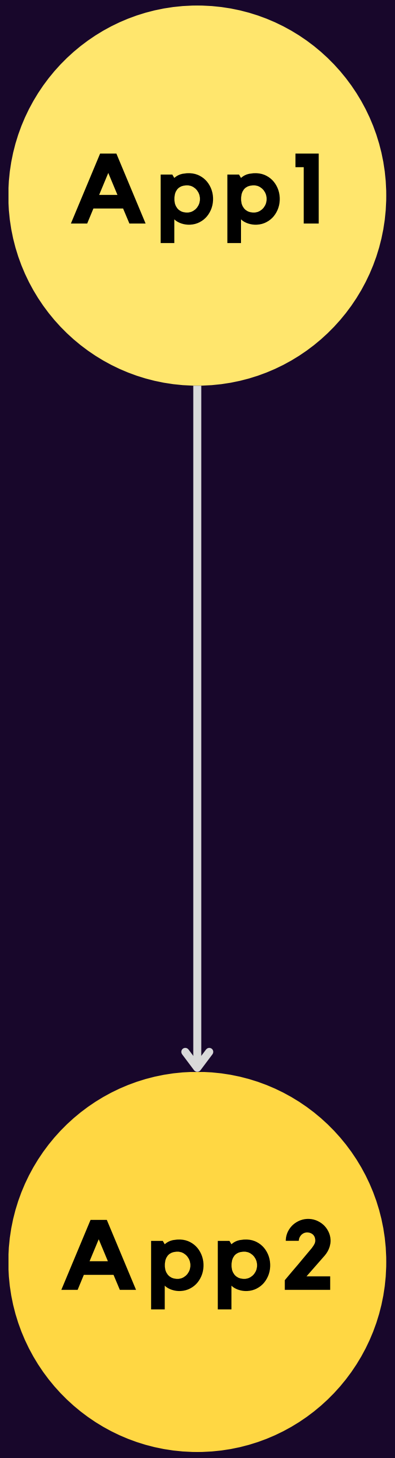
API

API stands for Application Programming Interface

Bridge between different Apps



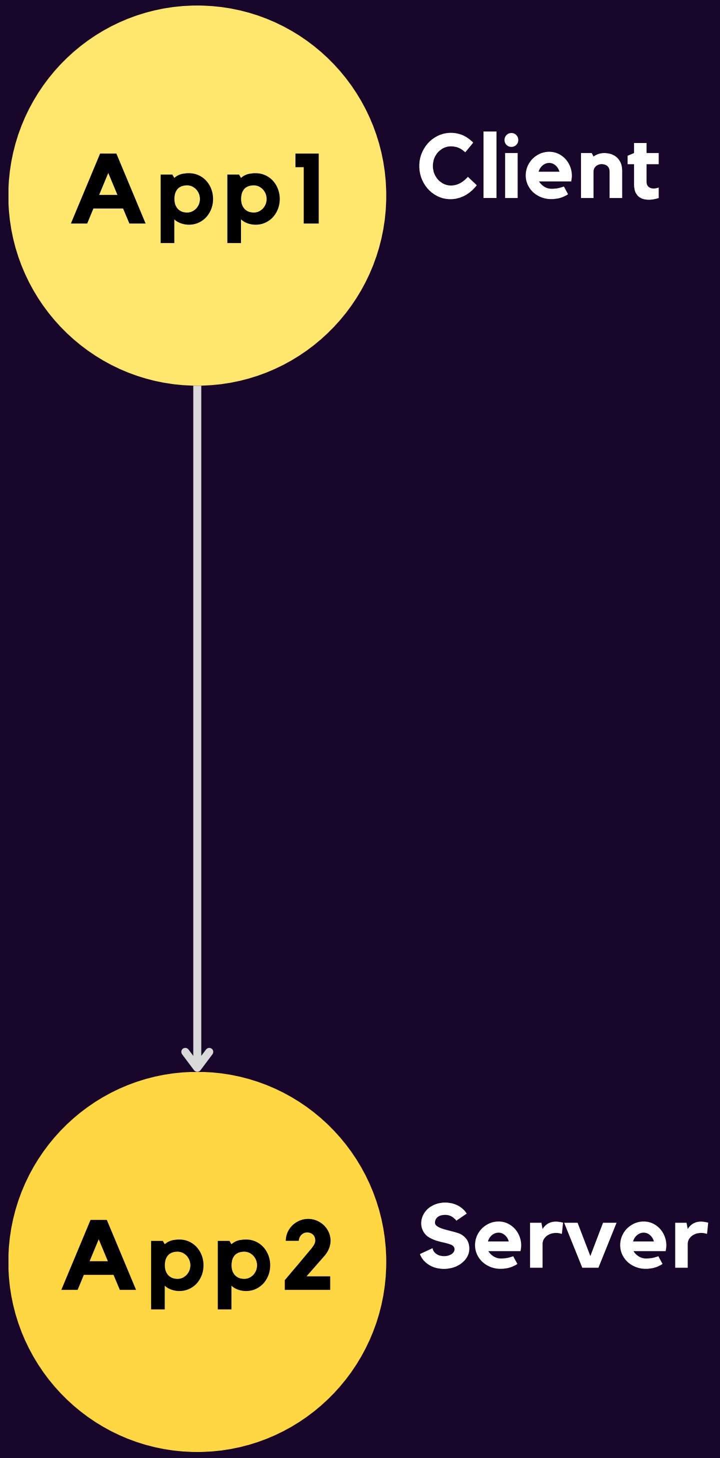
What is an API?



API allows Application (client) to access resources (digital / data) exposed by server through some end-points.

JSON Format is Used for exchange of data

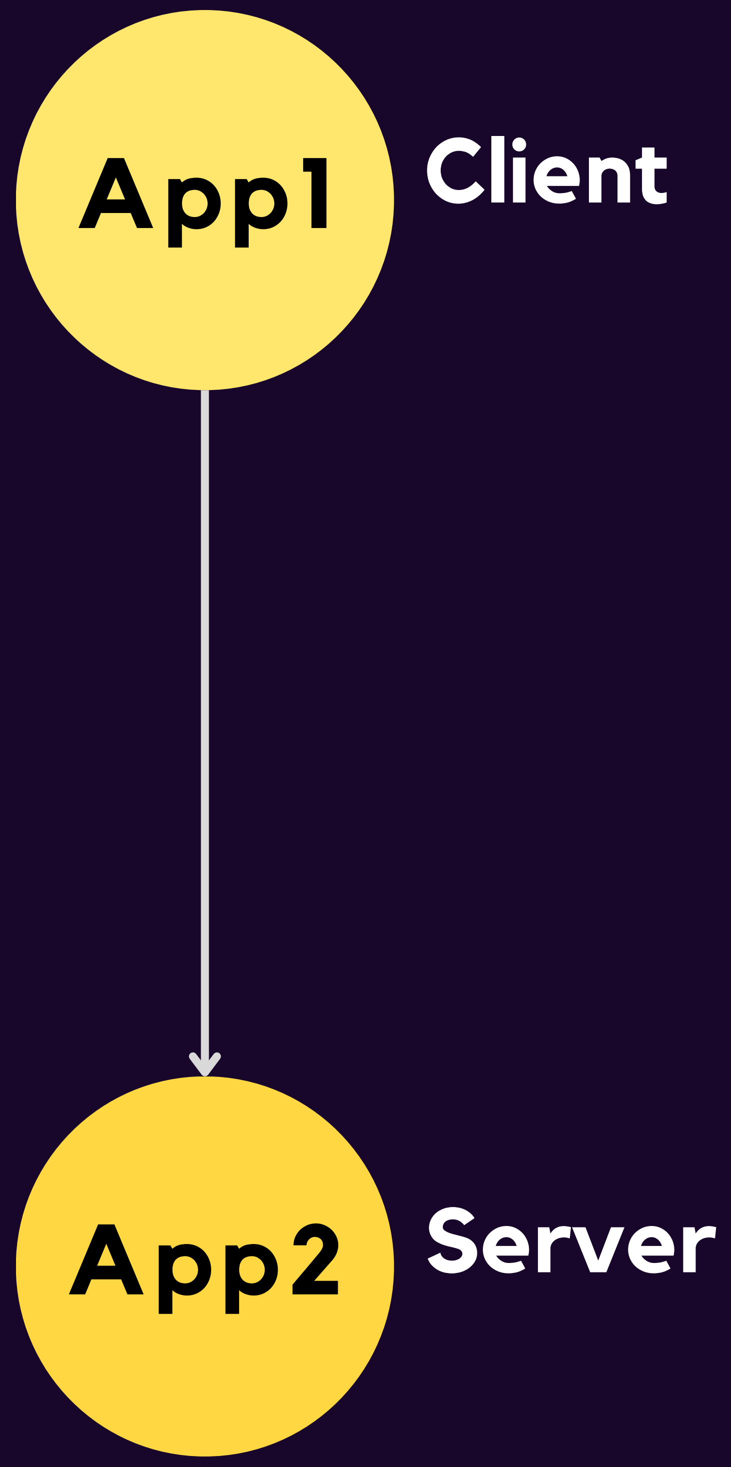
Restful API



The Server exposes certain functions on certain urls called as end_points.

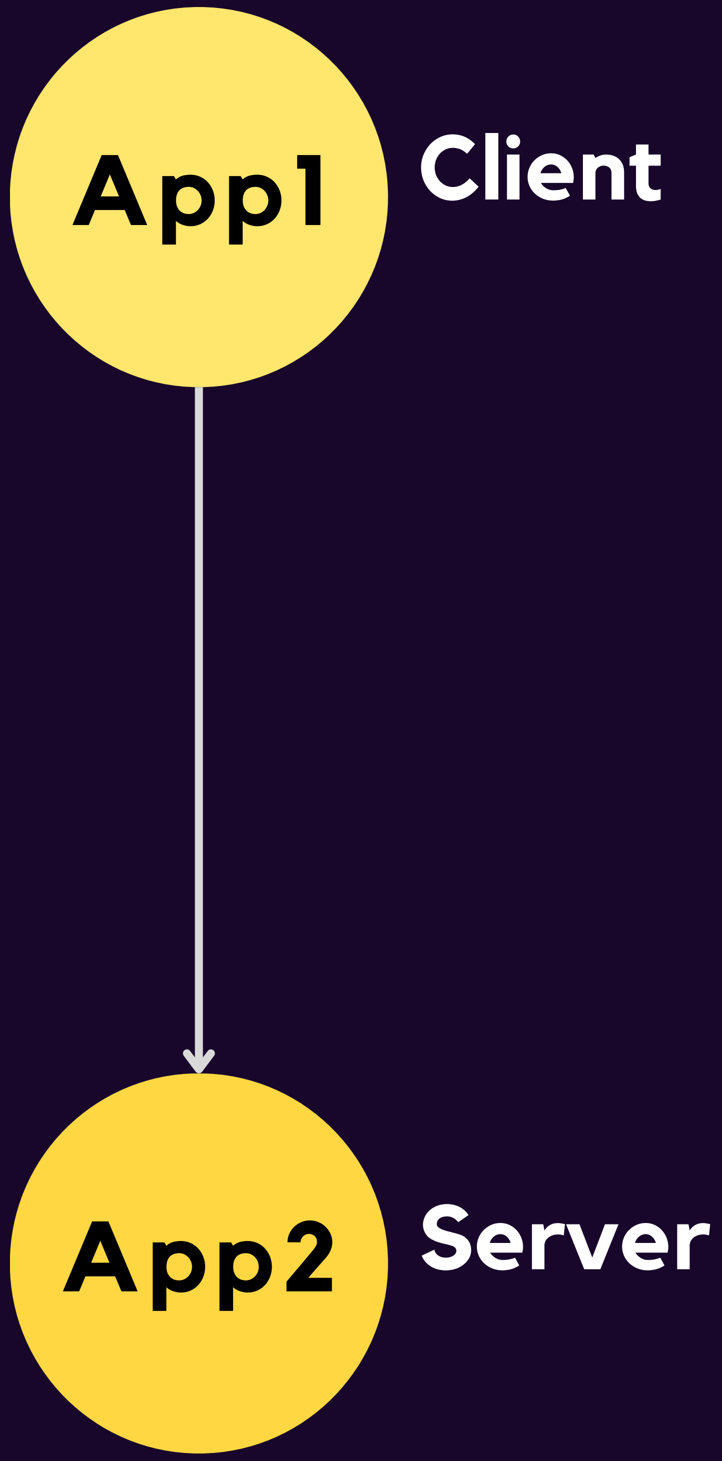
<base_url>/endpoint GET: Read Data
<base_url>/endpoint POST: Write Data
<base_url>/endpoint PUT / PATCH: Update Data
<base_url>/endpoint DELETE: Delete Data

Framework to build APIs



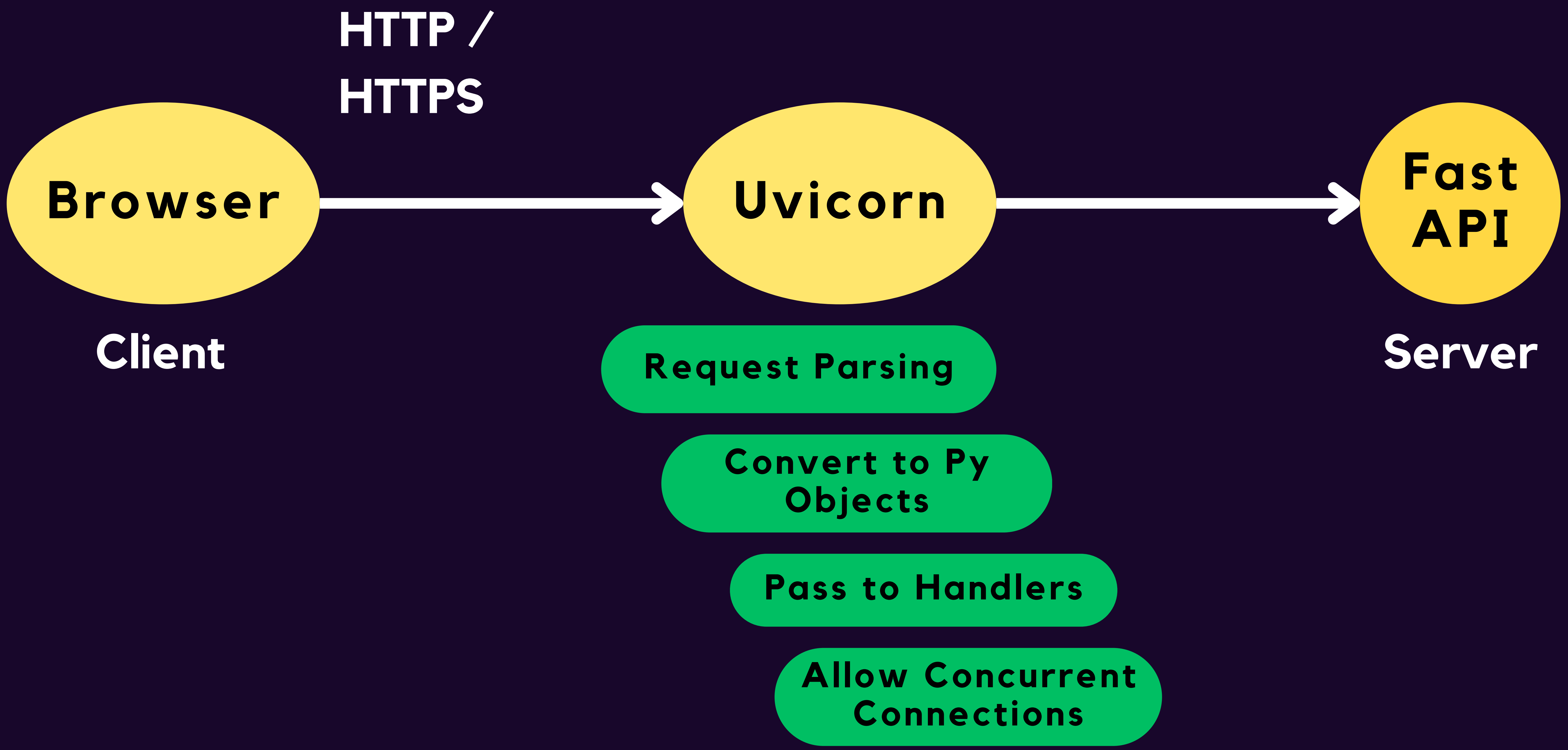
There are Python packages which abstract away the complexities of implementation providing a smooth way of exposing API end-points.

APIs with Fast API

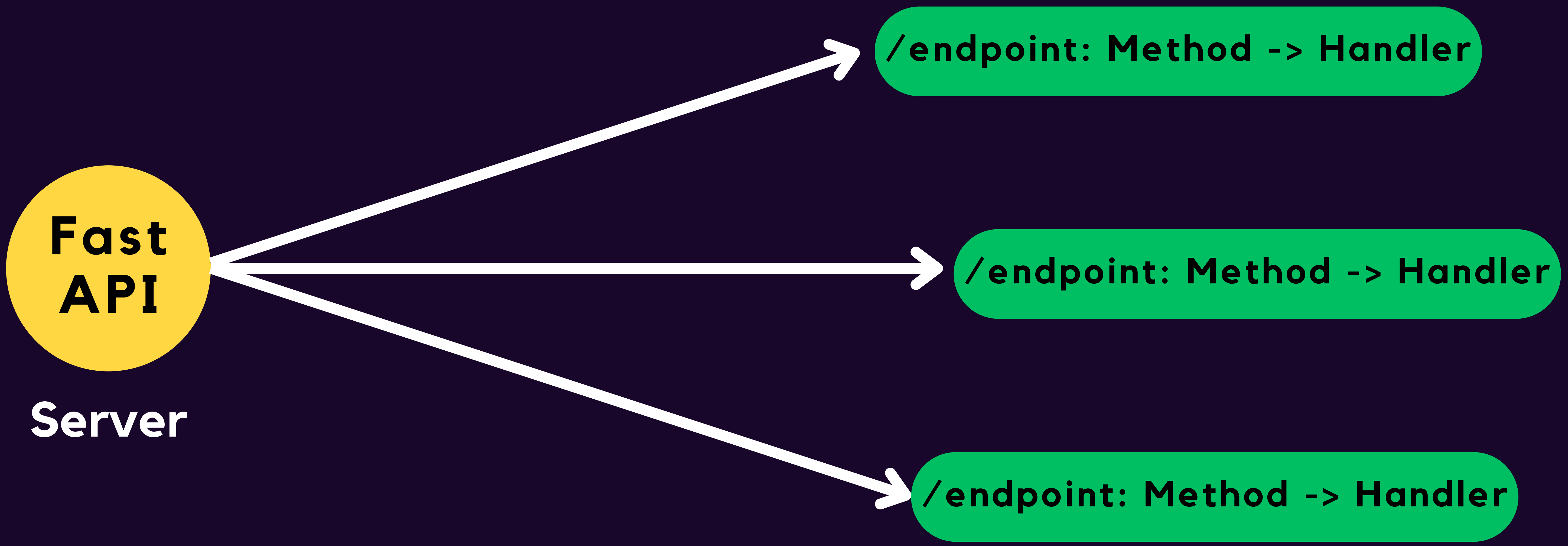


```
pip install fastapi uvicorn
```

What is UviCorn



What is fastAPI


Fast API
Server
/endpoint: Method -> Handler
/endpoint: Method -> Handler
/endpoint: Method -> Handler

Writing handlers is as easy as writing Python functions.

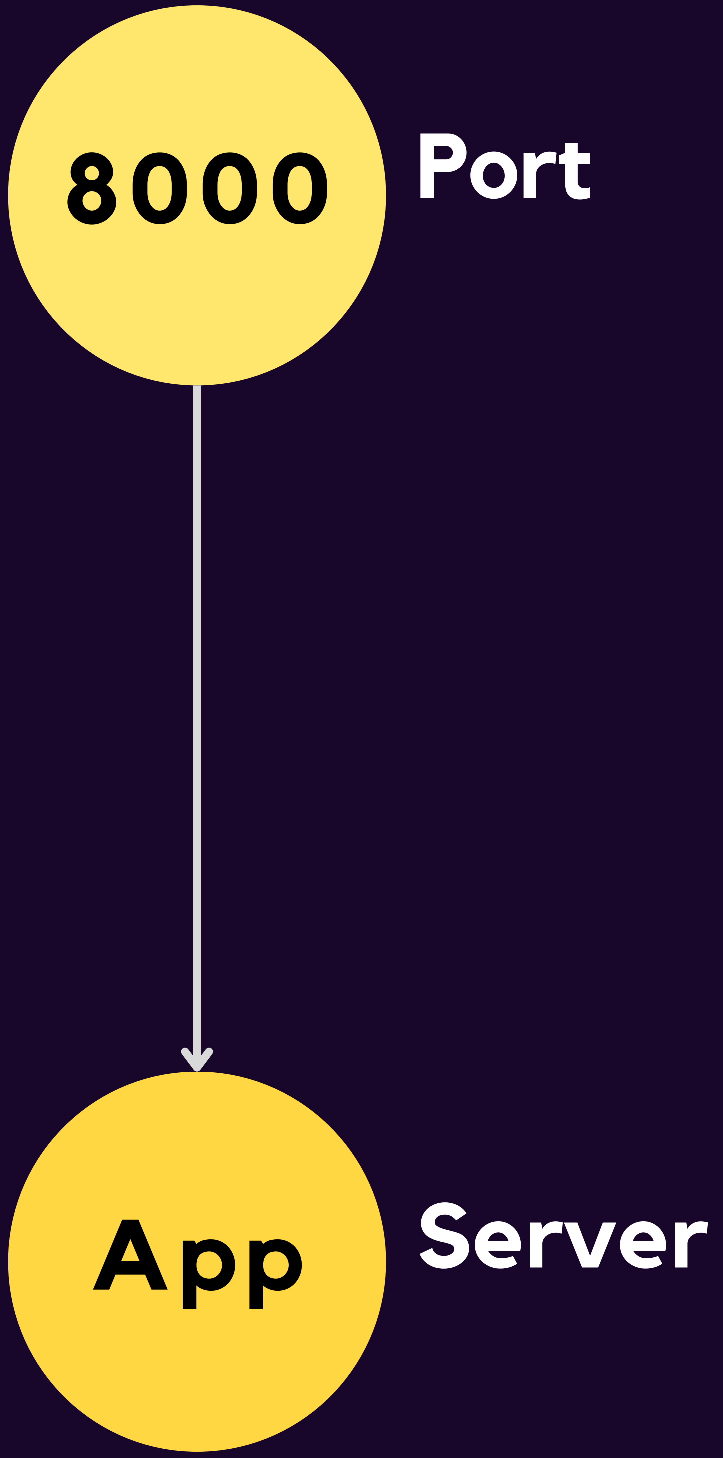
Writing your first Handler

Fast
API

Handler

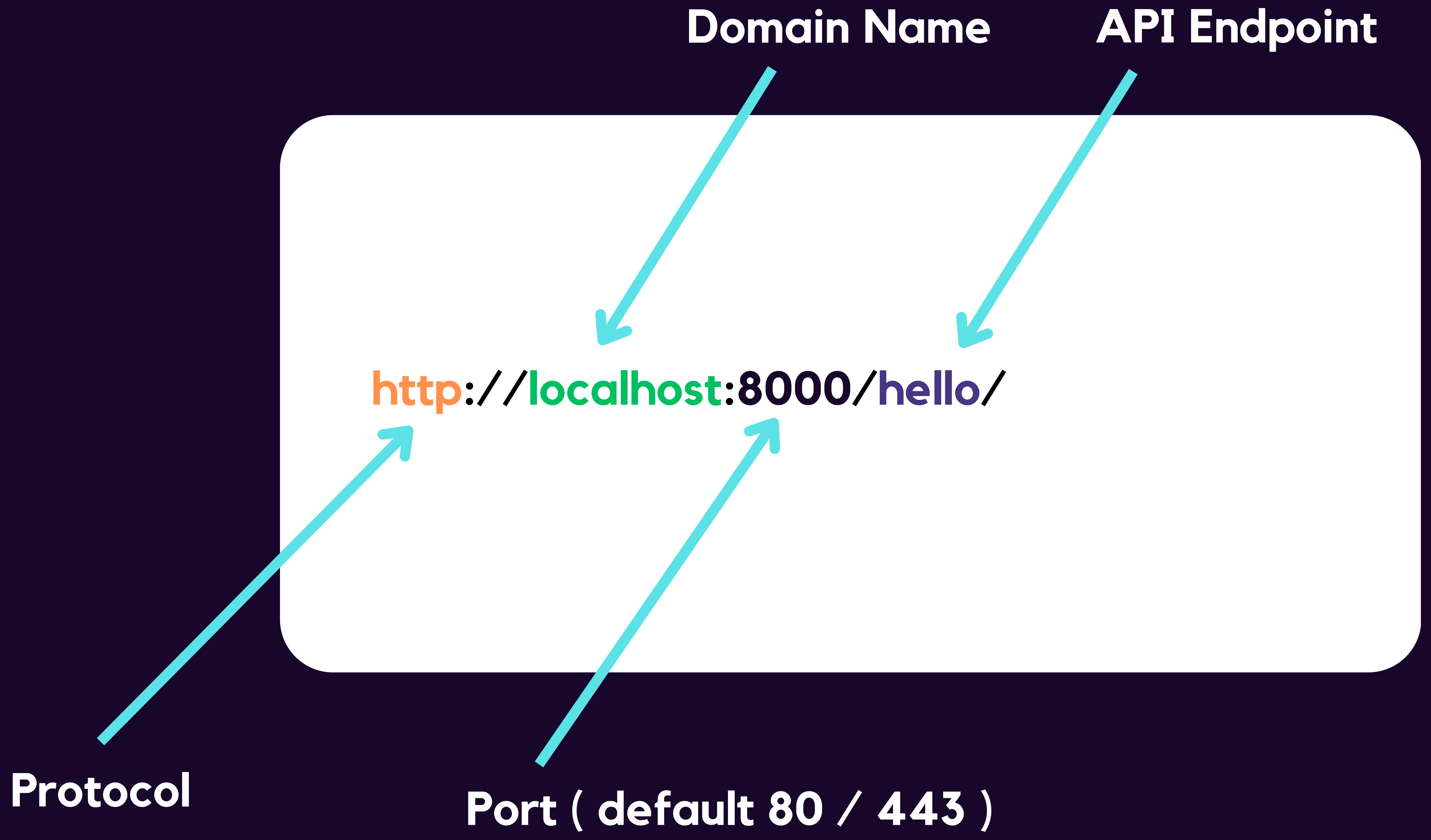
```
# create a file with name: main.py
# Writing your first handler

from fastapi import FastAPI, HTTPException
# Define FastAPI app
app = FastAPI()


# Define route to handle requests
@app.get("/hello")
async def say_hello():
    return {"result": "Hello from API"}
```

Launch your App

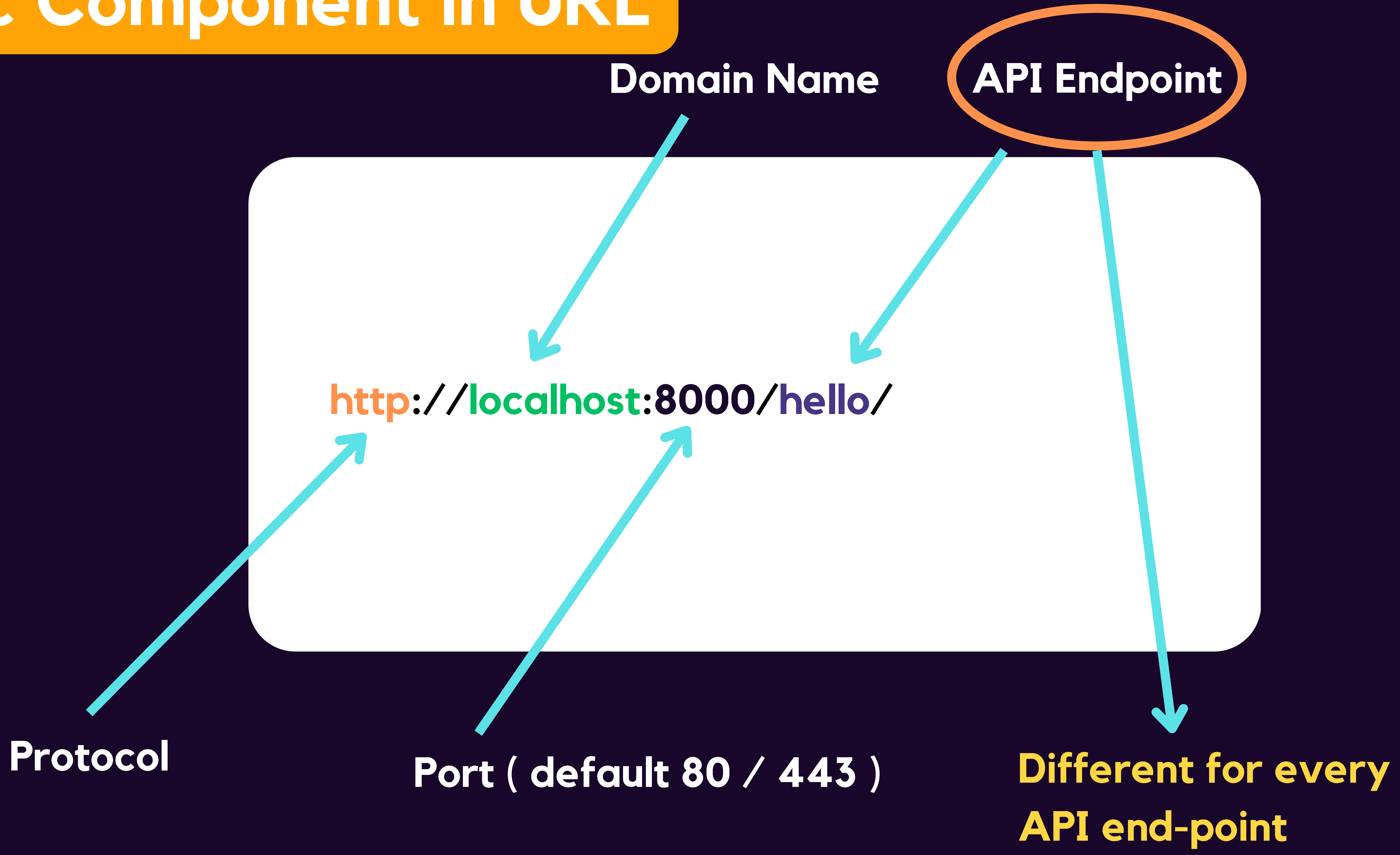


```
uvicorn main:app --reload
```

Url Components



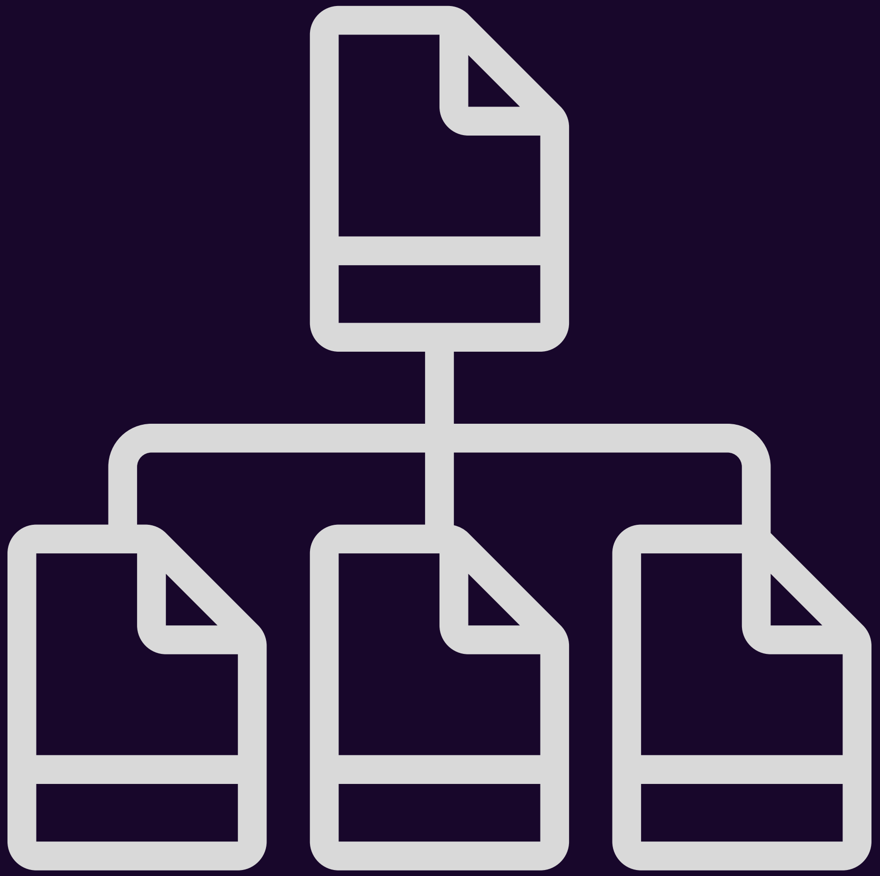
Dynamic Component in URL


Domain Name
API Endpoint
http://localhost:8000/hello/
Protocol
Port (default 80 / 443)
Different for every API end-point

Project Structure to Serve a Model

```
App/
    - main.py
    /predictors [ contains serialized models ]
    /models  [ for app data model objects ]
    /steps   [ to pre-process req data ]
    /utils   [ general utility functions ]
```

Predictors has all the serialized models



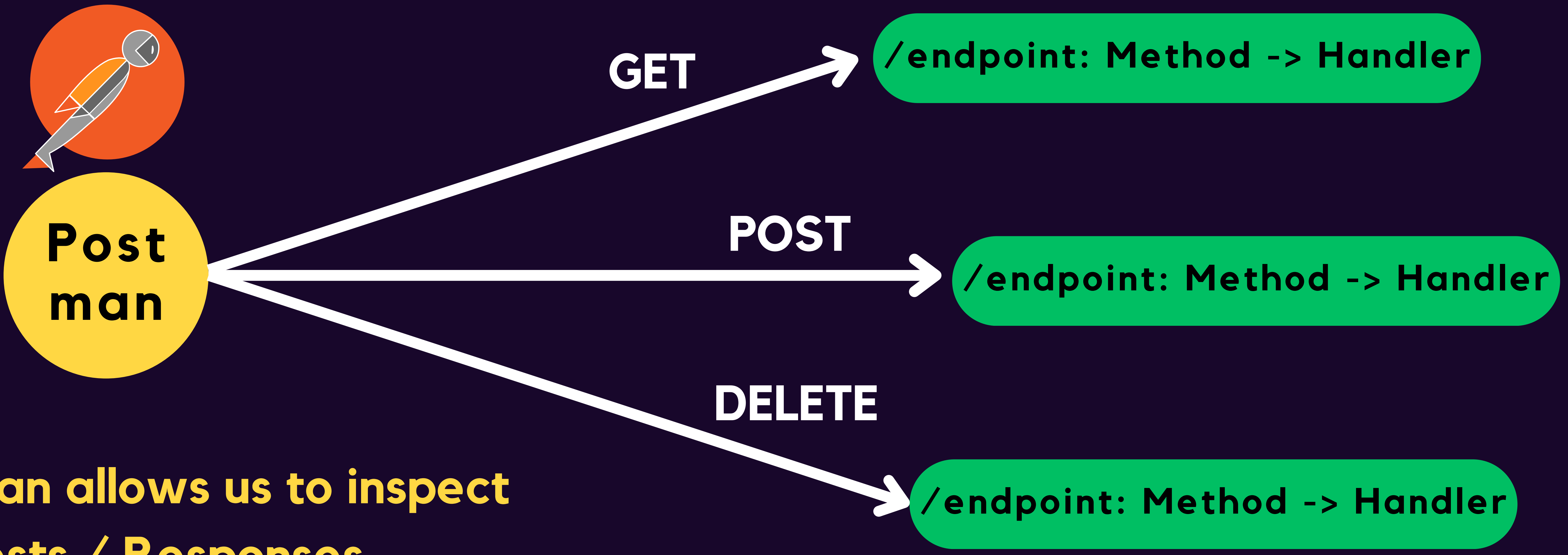
```
App/
    - main.py
    /predictors [ contains serialized models ]
        - model.keras
        - iris_predictor.py
    /models  [ for app data model objects ]
    /steps   [ to pre-process req data ]
    /utils   [ general utility functions ]
```

Predictor class for providing predictions

```
class IrisPrediction:

    def __init__(self, model_file):
        self.model_file = model_file

    def load_model( self ):
        pass

    def predict( self, input_parameters ):
        pass
```

Interruption! (Tour of Postman)

Browser's URL bar only supports get method

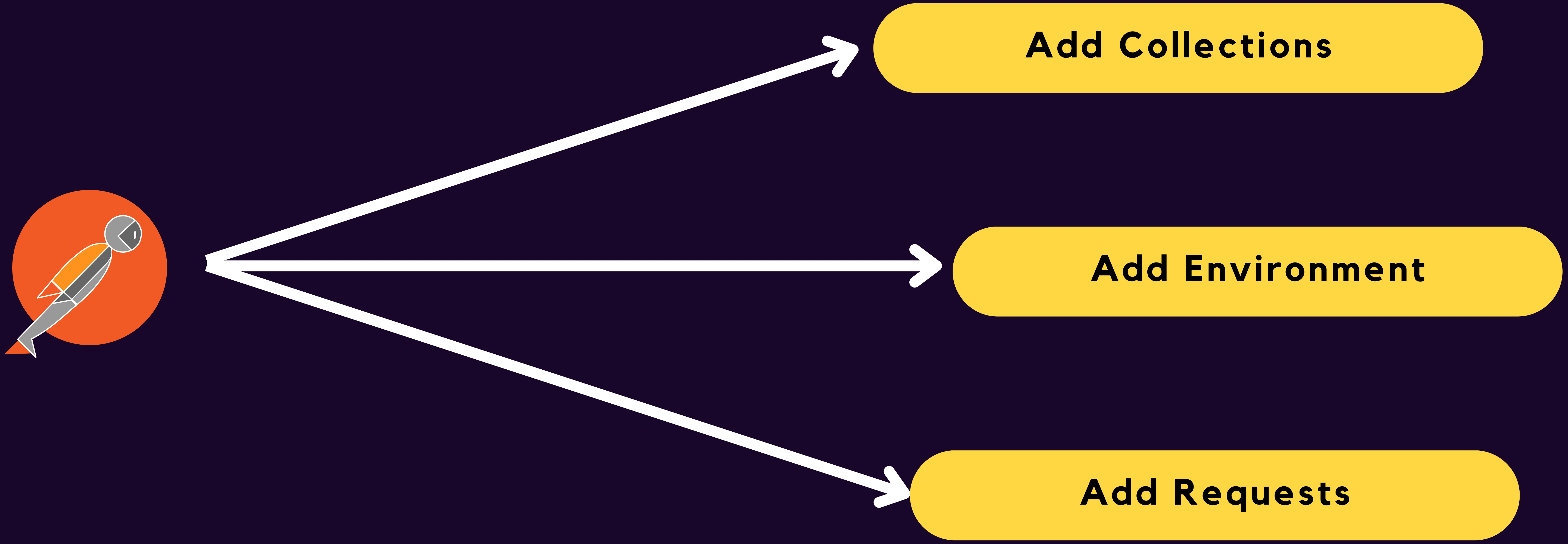

Post man
GET
POST
DELETE
/endpoint: Method -> Handler
/endpoint: Method -> Handler
/endpoint: Method -> Handler

Postman allows us to inspect Requests / Responses.

Install Postman (If not already having)

https://www.postman.com/downloads/

Postman -- Important Operations



Make a POST request via POSTMAN

```
POST: http://localhost:8000/predict

{
    "sepal_length": 2.5,
    "sepal_width": 3.0,
    "petal_length": 3.2,
    "petal_width": 2.6
}
```

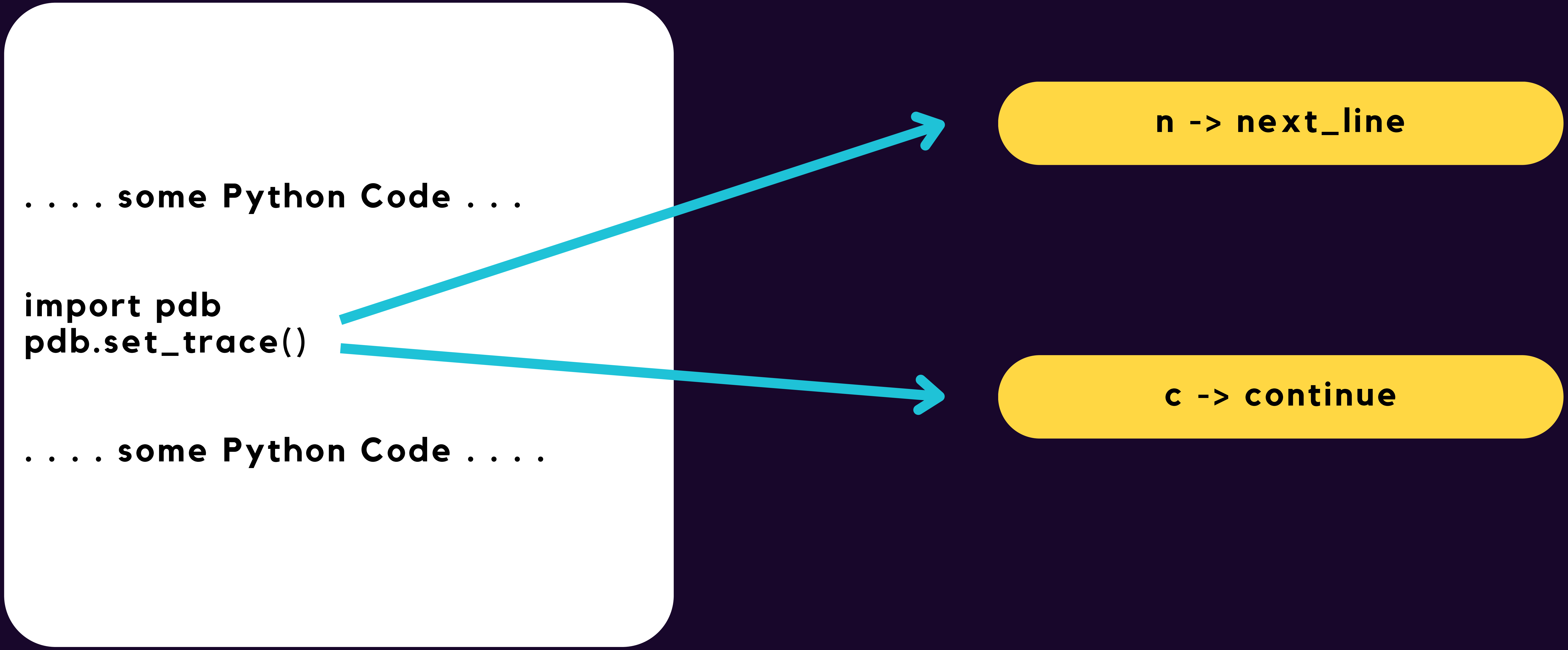
Response should give the class.

Response:

```
{
    "class": "setosa",
    "message": "Predicted class is setosa"
}
```

How to Debug your Application?

source_code.py



```
. . . . some Python Code . . .

import pdb
pdb.set_trace()

. . . . some Python Code . . . .
```

Add Your Project's root to Path

```
import sys
import os

# Get the root directory of your project
root_dir = os.path.dirname(os.path.abspath(__file__))

# Add the root directory to the Python path
sys.path.append(root_dir)
```

But Why ?

In order to be able to access modules and packages seamlessly.

Stepping towards modularity

```
my_fastapi_app/
│
├─── app/
│    ├─── __init__.py
│    ├─── main.py
│    └─── api/
│         ├─── __init__.py
│         ├─── package1/
│         │    ├─── __init__.py
│         │    ├─── endpoints/
│         │    │    ├─── __init__.py
│         │    │    └─── endpoint1.py
│         │    └─── models/
│         │         └─── __init__.py
│         ├─── package2/
│         │    ├─── __init__.py
│         │    ├─── endpoints/
│         │    │    ├─── __init__.py
│         │    │    └─── endpoint2.py
│         │    └─── models/
│         │         └─── __init__.py
│         └─── common/
│              ├─── __init__.py
│              └─── utils.py
└─── ...
```

Because it's not enough to deploy your model to just the localhost.

Stepping towards modularity

```
# file_name = main.py

from fastapi import FastAPI
from app.api.package1 import endpoint1
from app.api.package2 import endpoint2

app = FastAPI()

# Include routes from package1
app.include_router(endpoint1.router, prefix="/package1")

# Include routes from package2
app.include_router(endpoint2.router, prefix="/package2")



# uvicorn main:app --reload
# use this to run the app
```

Modular code is easy to:

1. Test
2. Scale
3. Maintain
4. Debug
5. Read

Stepping towards modularity

```
# file_name: endpoint1.py

from fastapi import APIRouter

router = APIRouter()

@router.get("/example")
async def get_example():
    return {"message": "This is an example endpoint"}

@router.post("/example")
async def create_example():
    return {"message": "Created an example"}
```

While designing a system, try to decouple as many components as possible so they can be dealth with individually.