Introduction to FastAPI

Fast, modern web framework for building APIs with Python 3.10

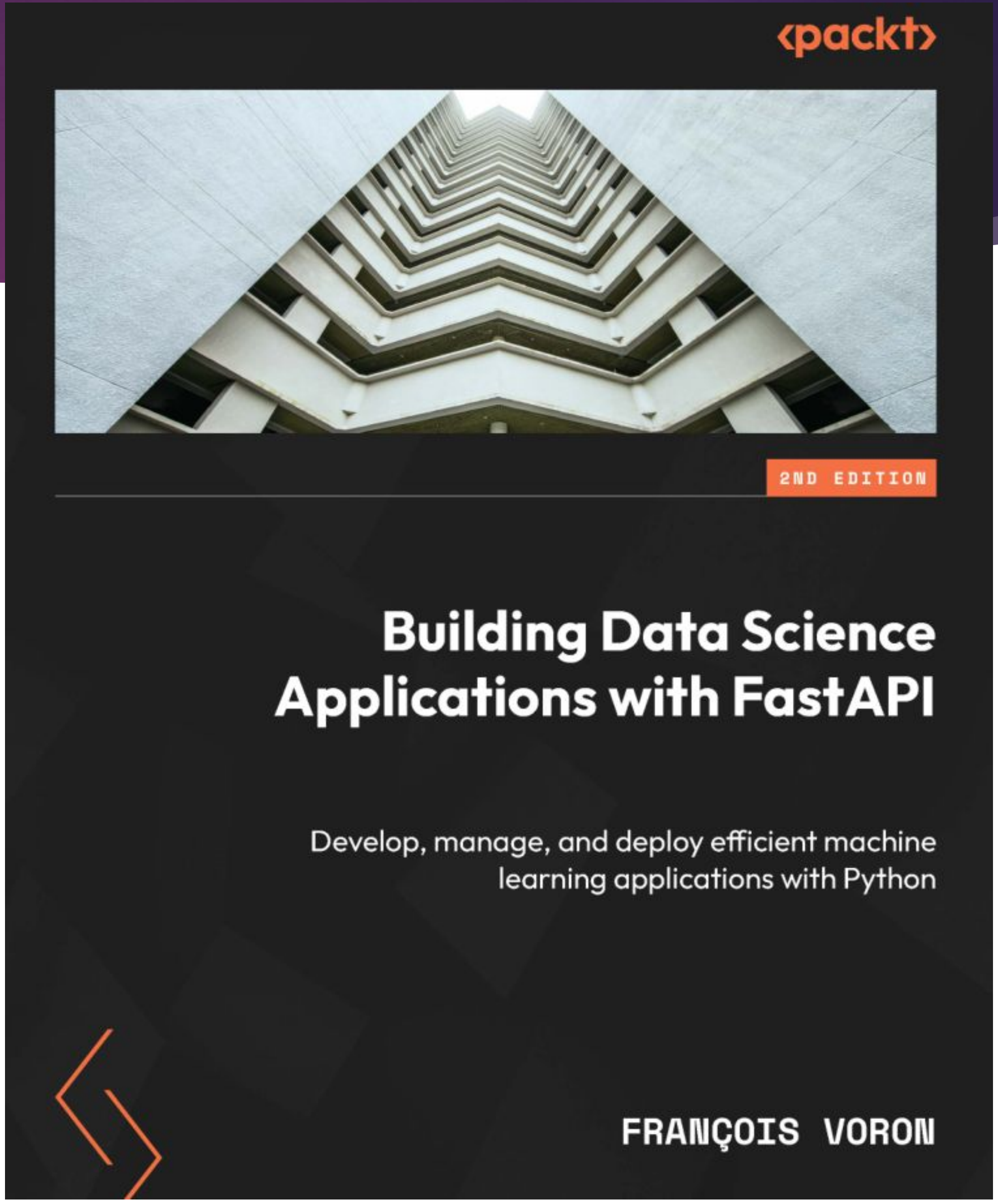
DR. MOHAMMED AL-HUBAISHI

Book Reference

https://fastapi.tiangolo.com/

Book codes

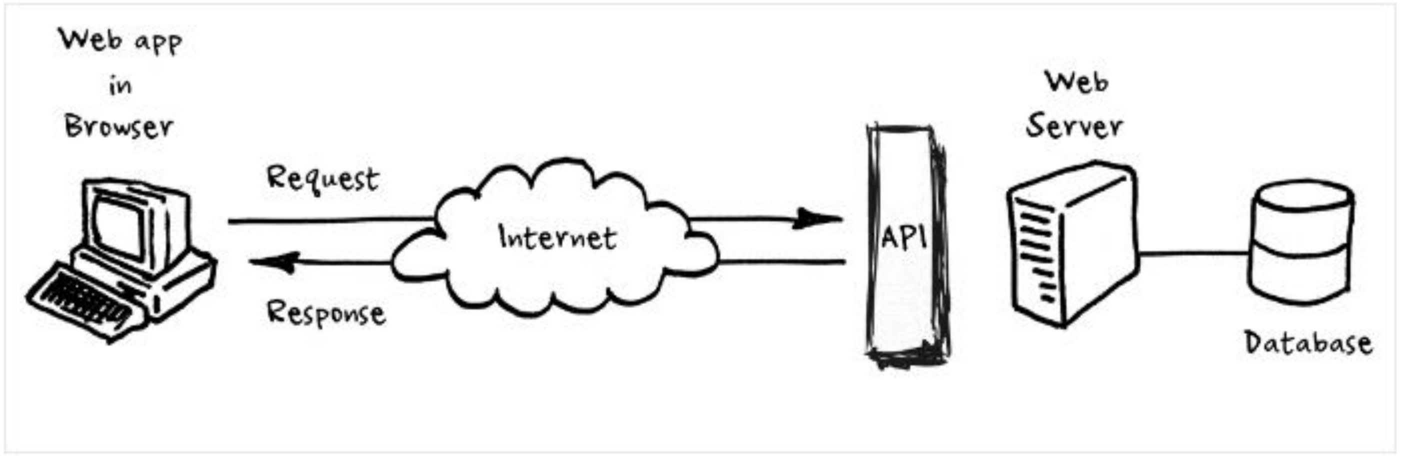
https://github.com/PacktPublishing/Building-Data-Science-Applications-with-FastAPI-Second-Edition/tree/main


‹packt›
2ND EDITION
Building Data Science Applications with FastAPI
Develop, manage, and deploy efficient machine learning applications with Python
FRANÇOIS VORON

what is API?

An API, or Application Programming Interface, is a set of rules and protocols that allows different software applications to communicate with each other.

It defines the methods and data formats that applications can use to request services from one another and exchange data.

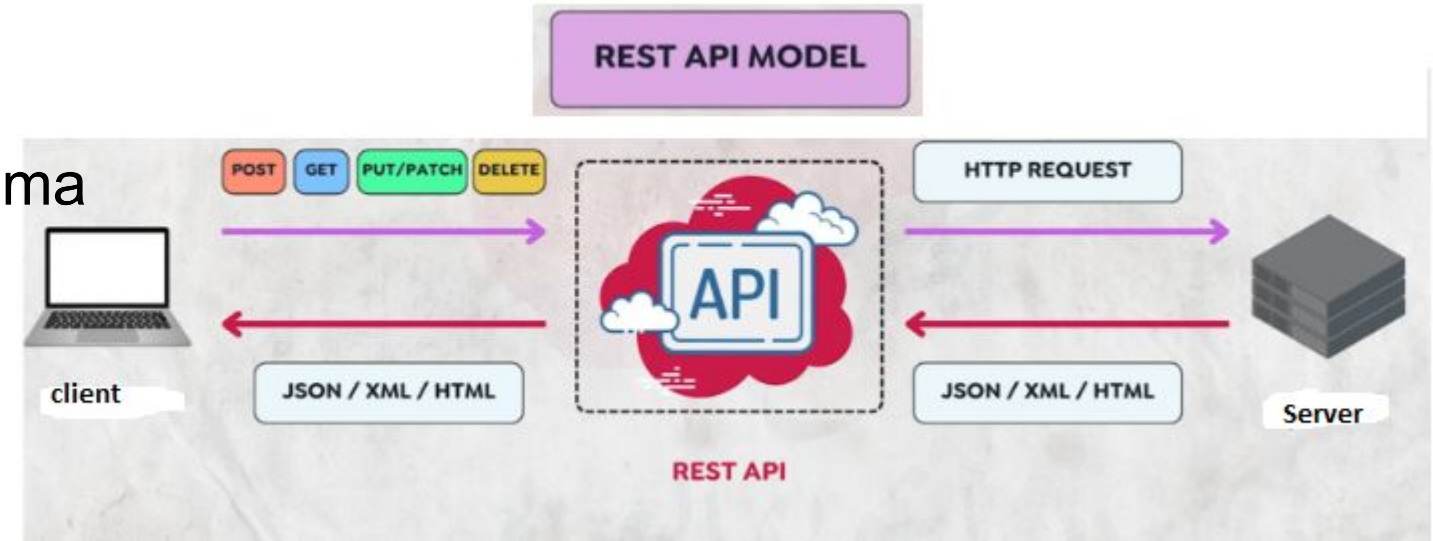


What is FastAPI?

Definition: FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.10 based on standard Python type hints.

Key Features:

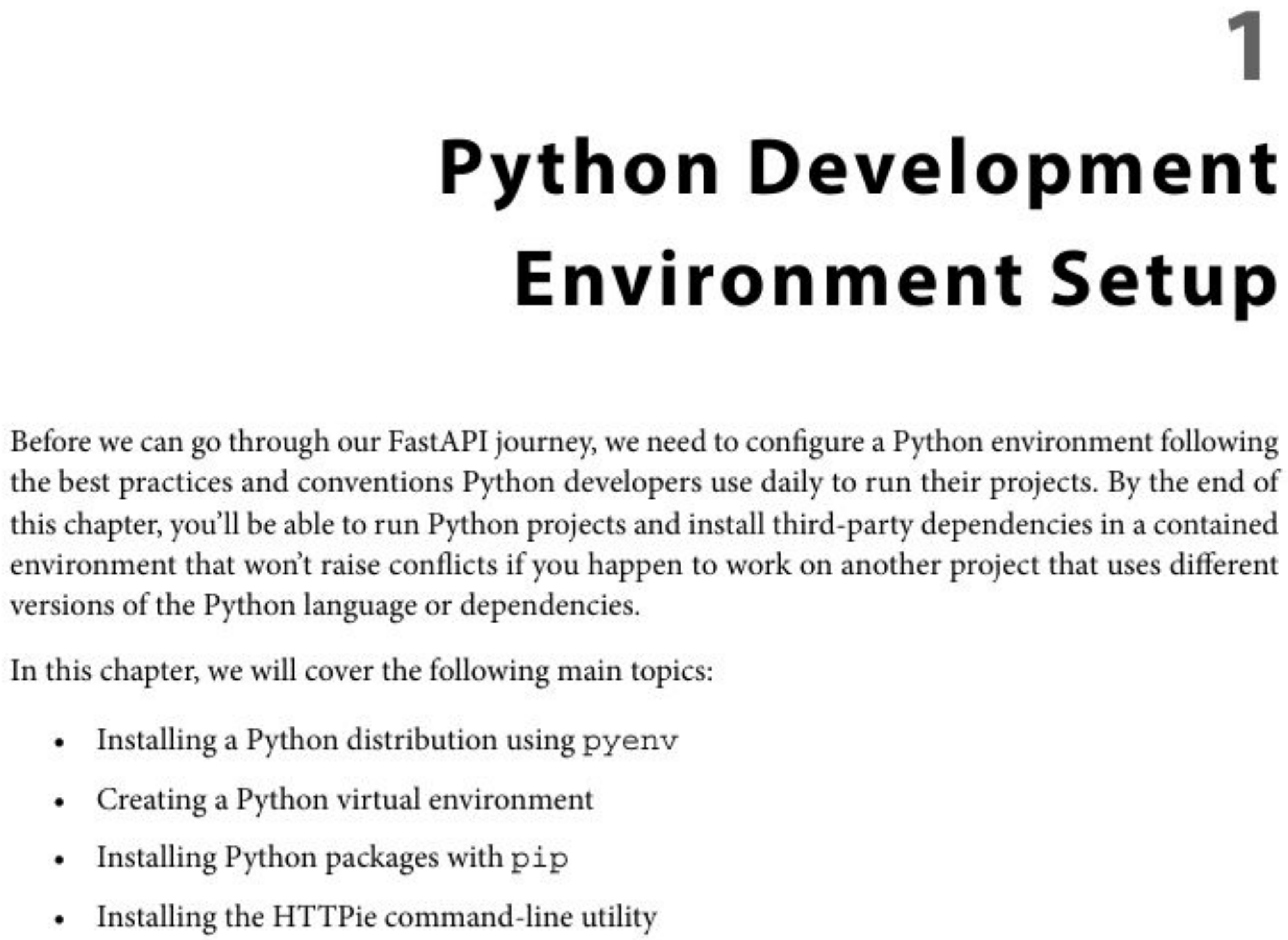
- Fast to code
- High performance
- Easy to use
- Based on OpenAPI and JSON Schema



Why Use FastAPI?

- **Advantages:**
  - Auto-generated API docs (Swagger UI and ReDoc)
  - Built-in validation and serialization with Python type hints
  - Asynchronous support (async/await)
  - Dependency injection system

CH1



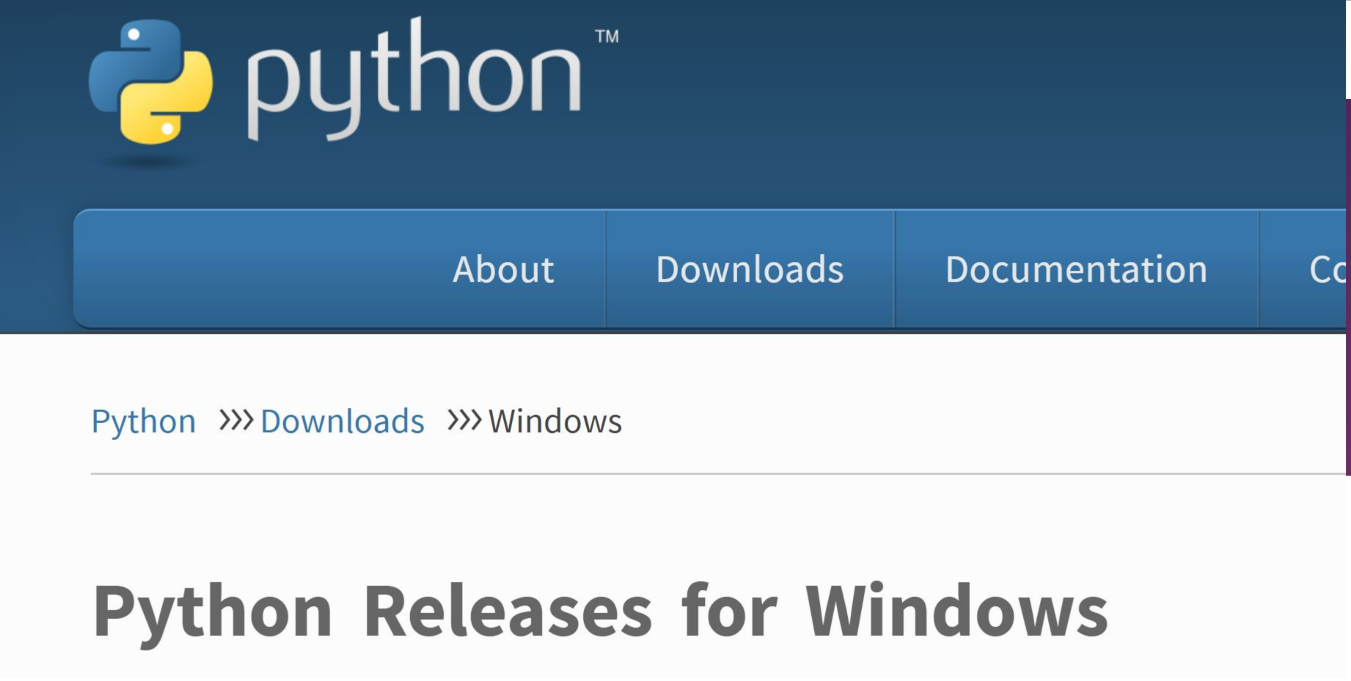
1

Python Development Environment Setup

Before we can go through our FastAPI journey, we need to configure a Python environment following the best practices and conventions Python developers use daily to run their projects. By the end of this chapter, you'll be able to run Python projects and install third-party dependencies in a contained environment that won't raise conflicts if you happen to work on another project that uses different versions of the Python language or dependencies.

In this chapter, we will cover the following main topics:

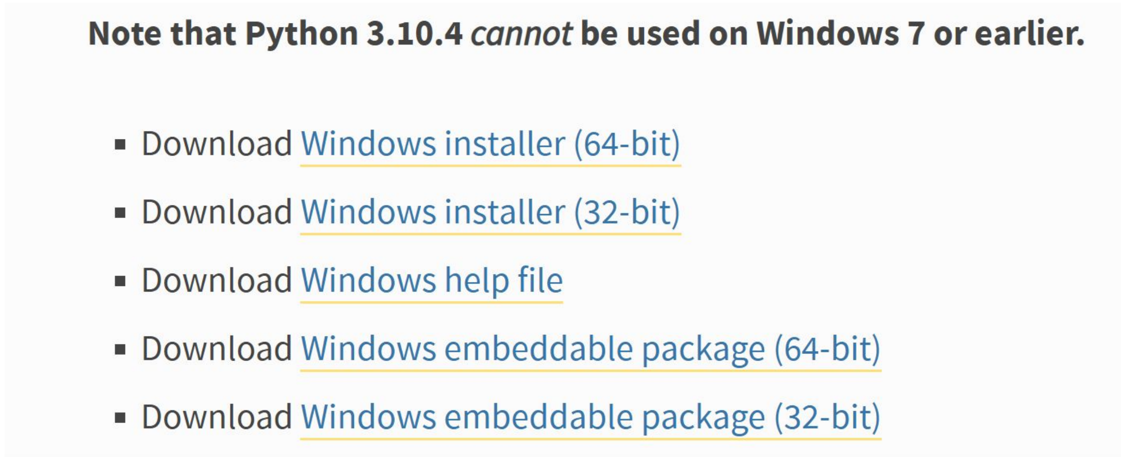
- Installing a Python distribution using `pyenv`
- Creating a Python virtual environment
- Installing Python packages with `pip`
- Installing the HTTPie command-line utility



Python ››› Downloads ››› Windows

Python Releases for Windows

https://www.python.org/downloads/windows/

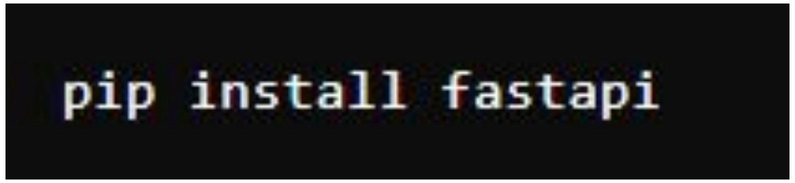


Note that Python 3.10.4 *cannot* be used on Windows 7 or earlier.

- Download Windows installer (64-bit)
- Download Windows installer (32-bit)
- Download Windows help file
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (32-bit)

Installation

- Install FastAPI:



```
pip install fastapi
```

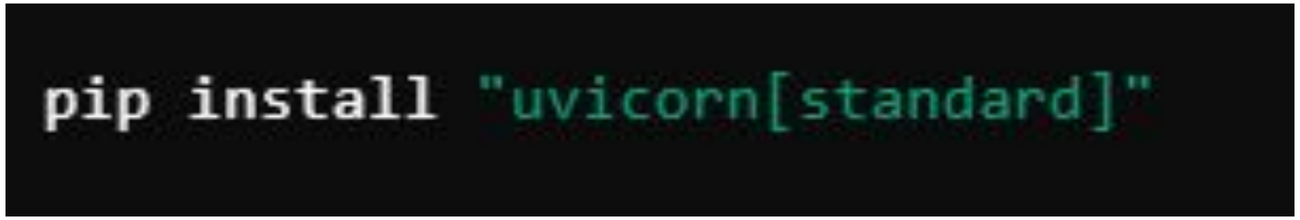
```
pip install "fastapi[standard]"
```

pip install fastapi

- Install ASGI Server (Uvicorn):

pip install "uvicorn[standard]"

https://fastapi.tiangolo.com/



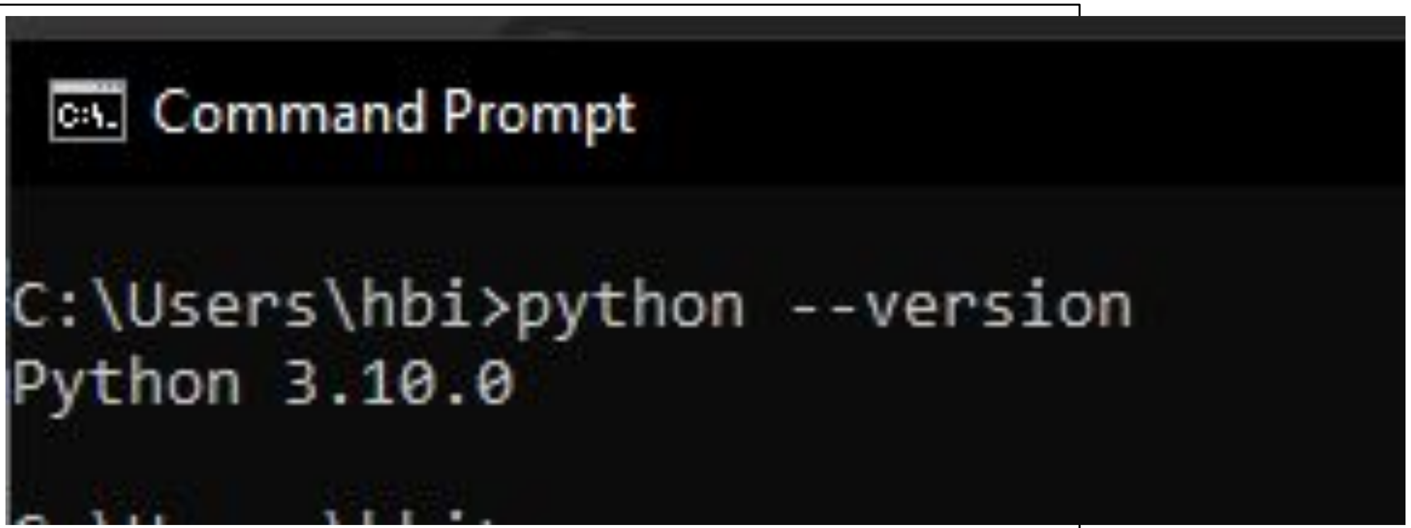
```
pip install "uvicorn[standard]"
```

Python version compatible

python 3.10

pip install fastapi

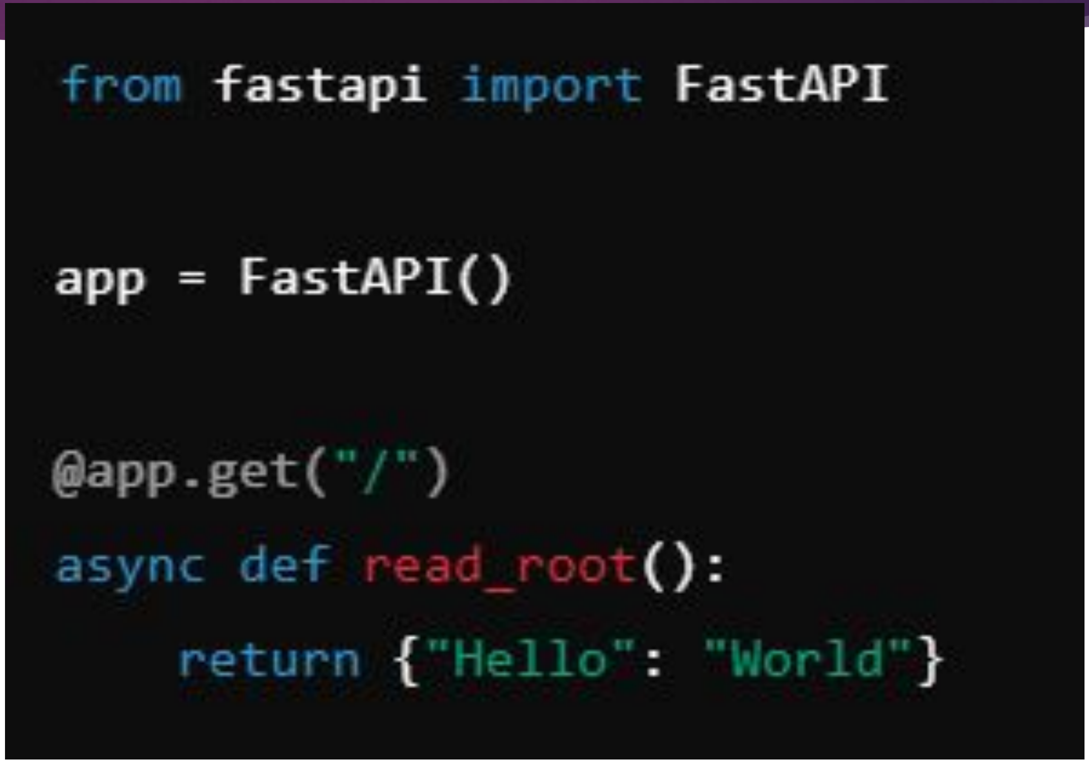
- fastapi
- pydantic
- pycaret
- pandas



```
C:\Users\hbi>python --version
Python 3.10.0
```

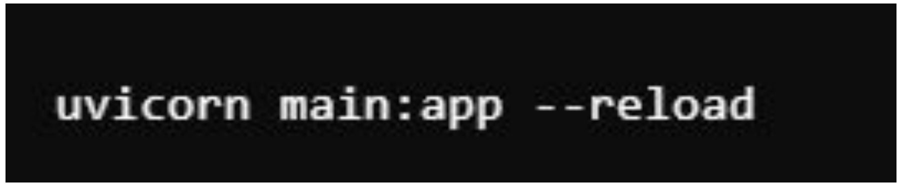
Setting Up a Simple FastAPI Application

```
from fastapi import FastAPI

app = FastAPI()

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



```
from fastapi import FastAPI

app = FastAPI()

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

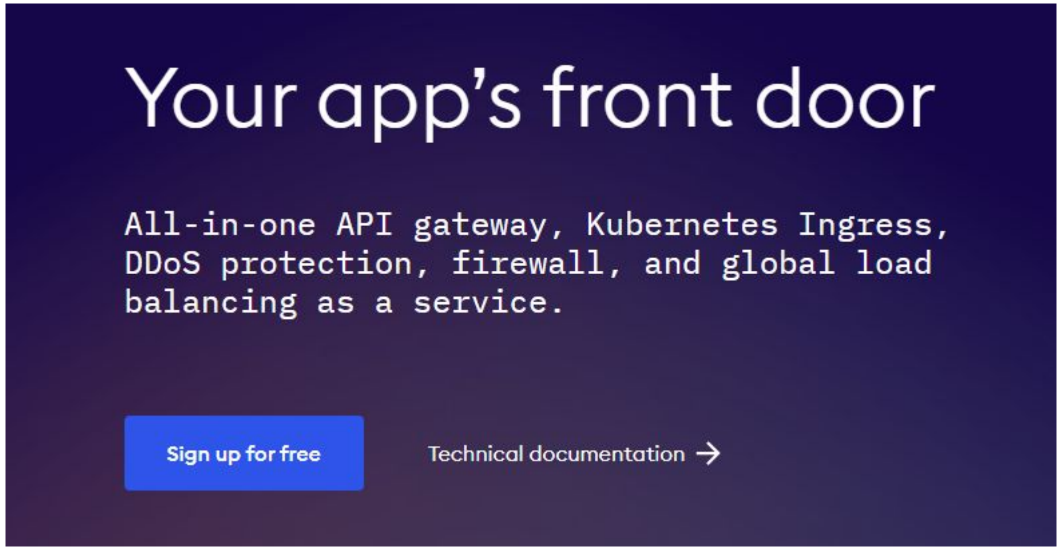
- Run the App:



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

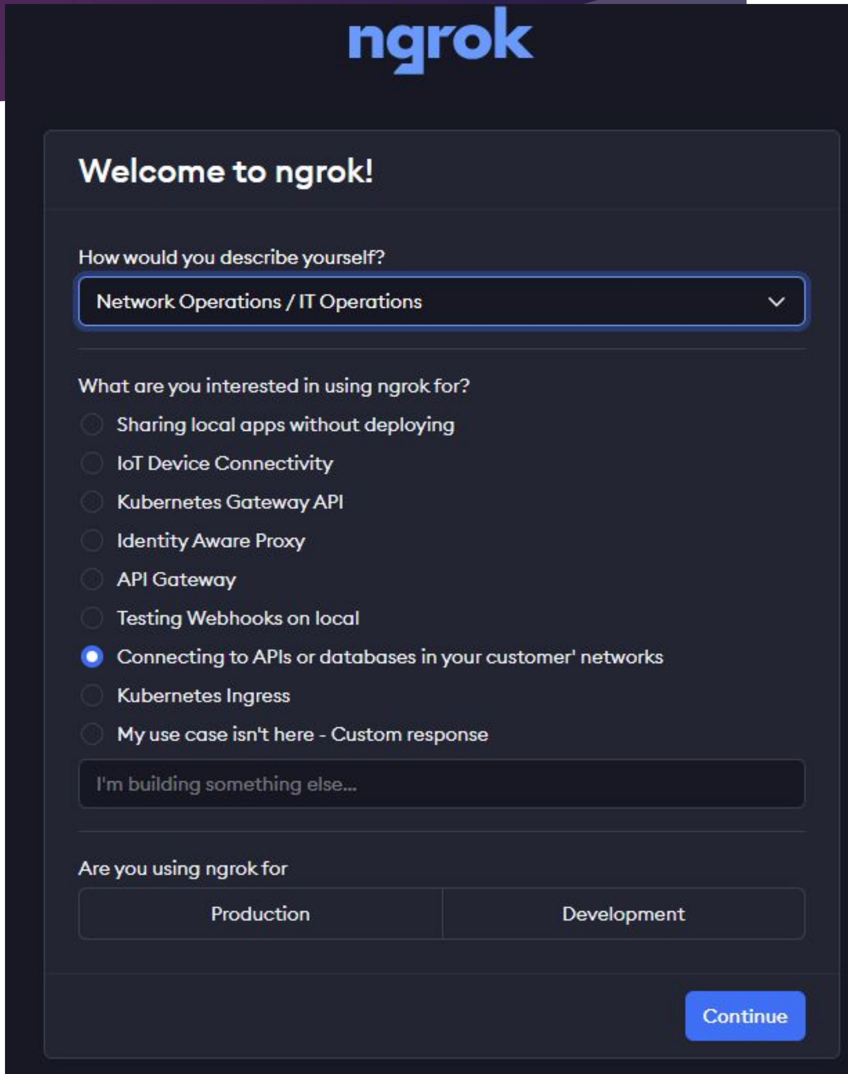
reload to update the app during the running time ngrok

ngrok | API Gateway, IoT Device Gateway, Secure Tunnels for Containers, Apps & APIs



Your app's front door

All-in-one API gateway, Kubernetes Ingress, DDoS protection, firewall, and global load balancing as a service.



ngrok

Welcome to ngrok!

How would you describe yourself?

Network Operations / IT Operations

What are you interested in using ngrok for?

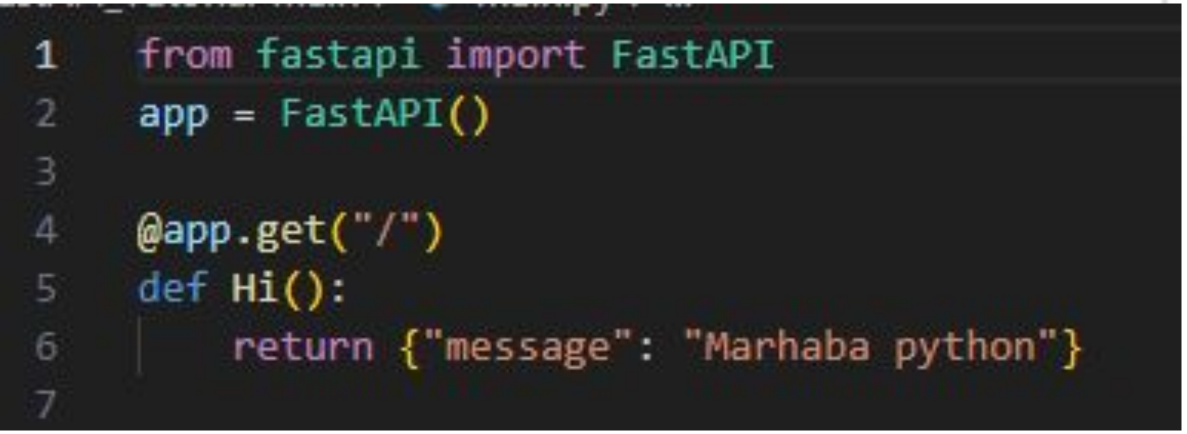
- Sharing local apps without deploying
- IoT Device Connectivity
- Kubernetes Gateway API
- Identity Aware Proxy
- API Gateway
- Testing Webhooks on local
- [✓ Connecting to APIs or databases in your customer' networks]
- Kubernetes Ingress
- My use case isn't here - Custom response

I'm building something else...

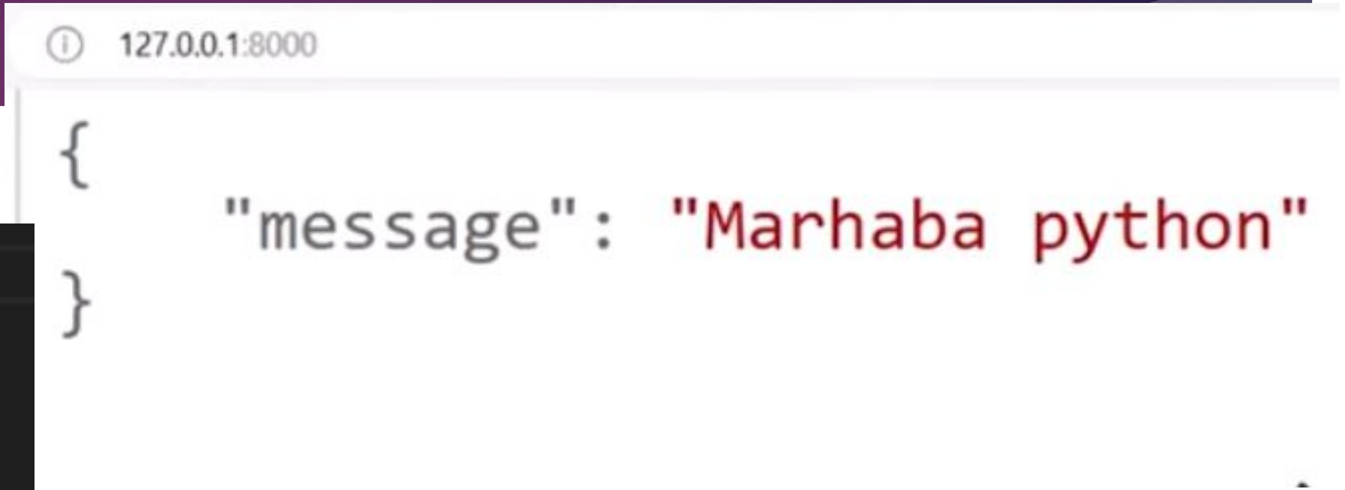
Are you using ngrok for

Production | Development

How does it work ?

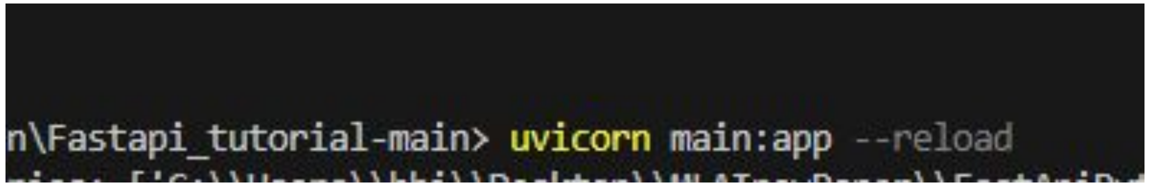


```python
from fastapi import FastAPI
app = FastAPI()

@app.get("/")
def Hi():
    return {"message": "Marhaba python"}

```



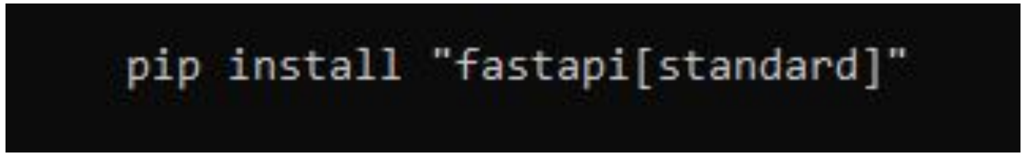
```
127.0.0.1:8000
{
    "message": "Marhaba python"
}
```

fastapi dev p2.py

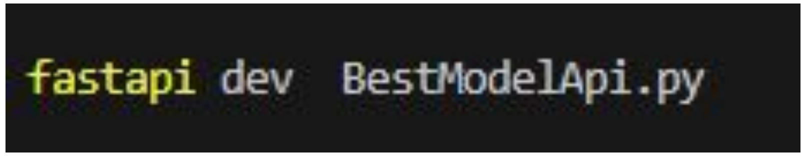


```
n\Fastapi_tutorial-main> uvicorn main:app --reload
```

Documents

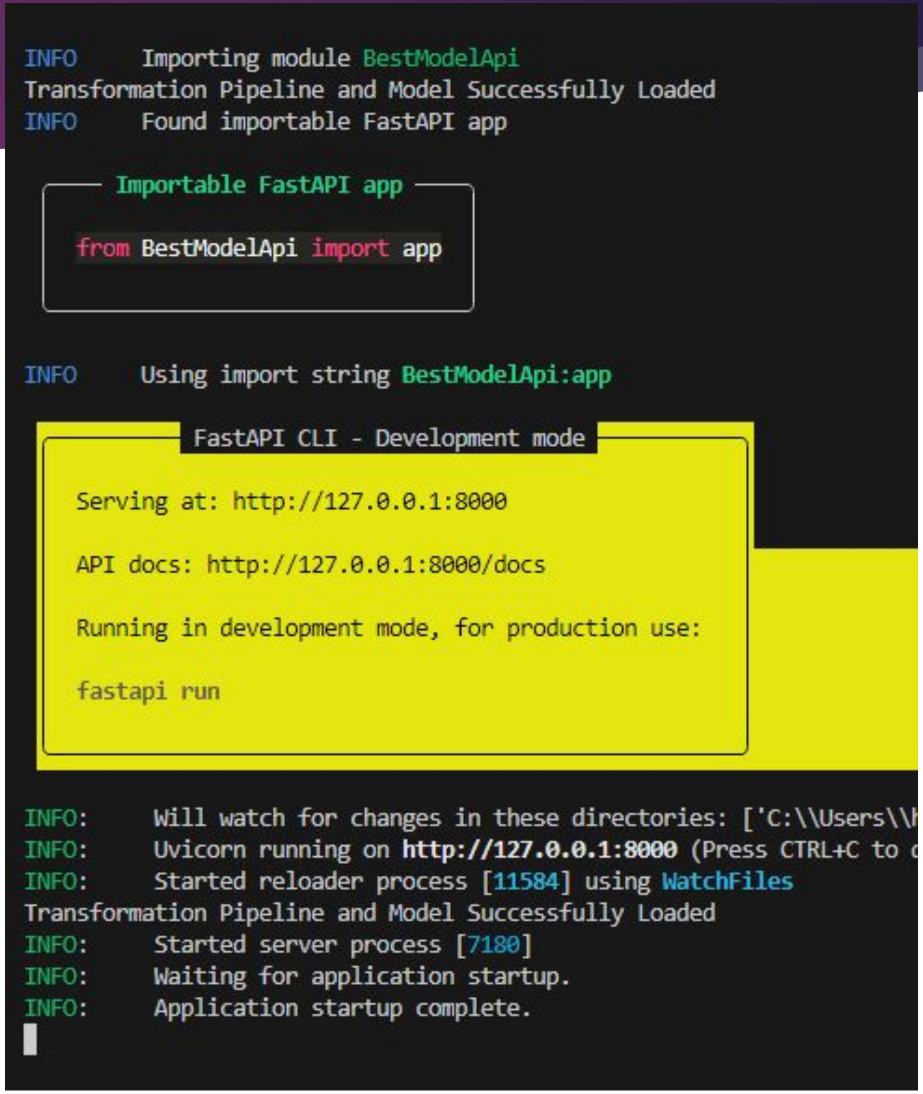


```
pip install "fastapi[standard]"
```



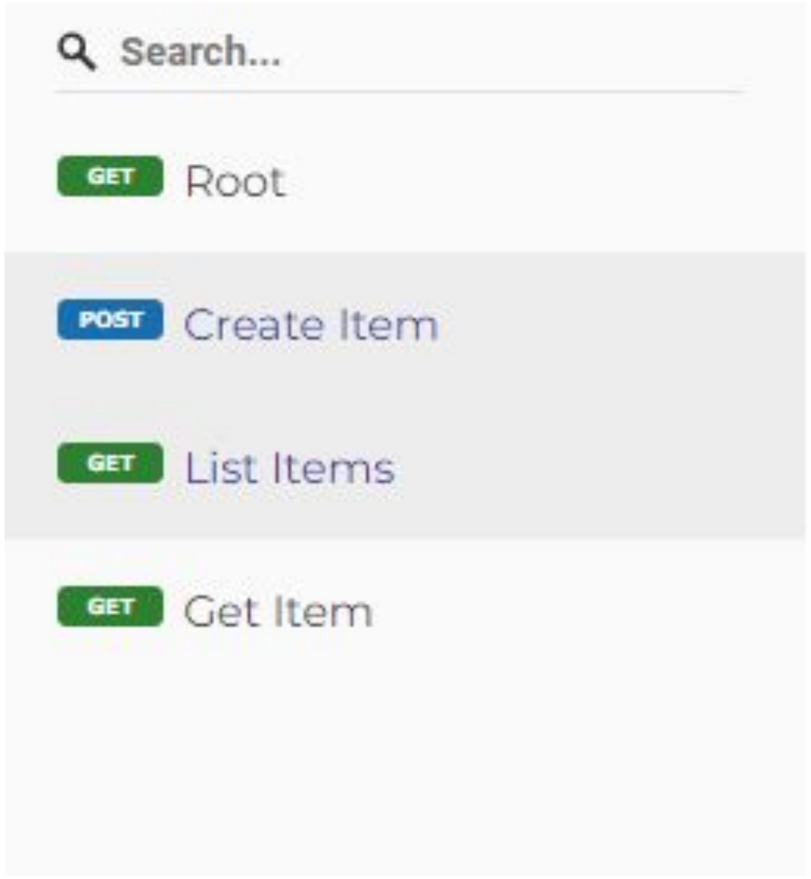
```
fastapi dev  BestModelApi.py
```

http://127.0.0.1:8000/redoc

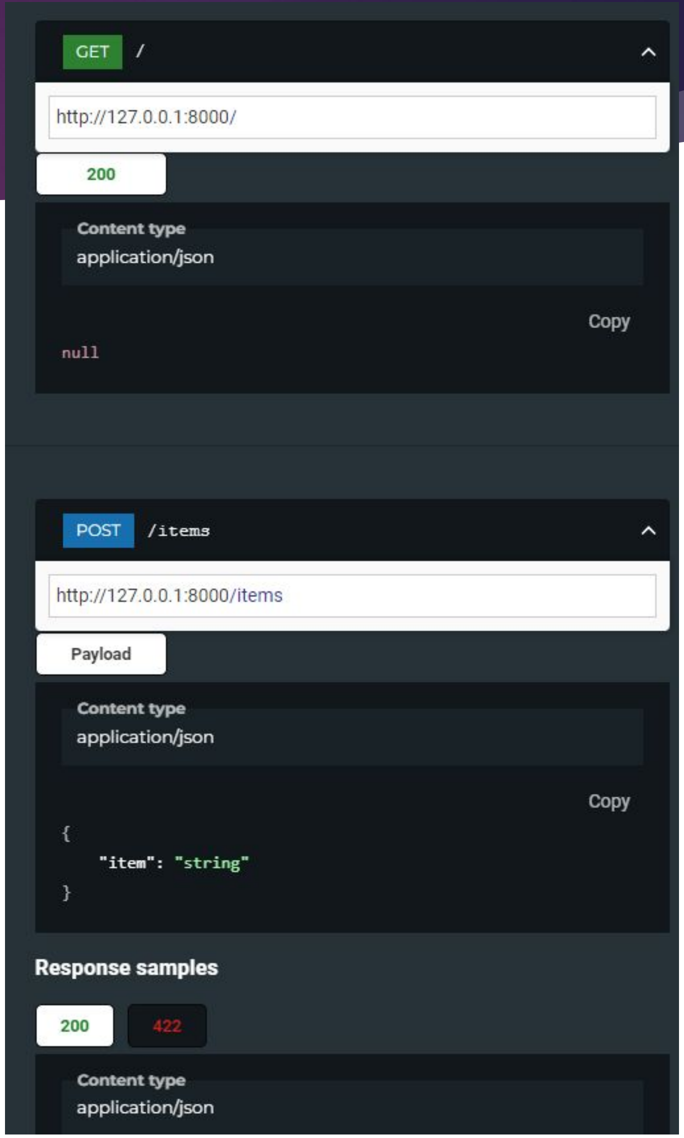


```
INFO     Importing module BestModelApi
Transformation Pipeline and Model Successfully Loaded
INFO     Found importable FastAPI app

 ╭─ Importable FastAPI app ─╮
 │                          │
 │  from BestModelApi import app │
 │                          │
 ╰──────────────────────────╯

INFO     Using import string BestModelApi:app

 ╭────────── FastAPI CLI - Development mode ──────────╮
 │                                                     │
 │  Serving at: http://127.0.0.1:8000                  │
 │                                                     │
 │  API docs: http://127.0.0.1:8000/docs               │
 │                                                     │
 │  Running in development mode, for production use:   │
 │                                                     │
 │  fastapi run                                        │
 │                                                     │
 ╰─────────────────────────────────────────────────────╯

INFO:     Will watch for changes in these directories: ['C:\\Users\\
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to
INFO:     Started reloader process [11584] using WatchFiles
Transformation Pipeline and Model Successfully Loaded
INFO:     Started server process [7180]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

http://127.0.0.1:8000/redoc



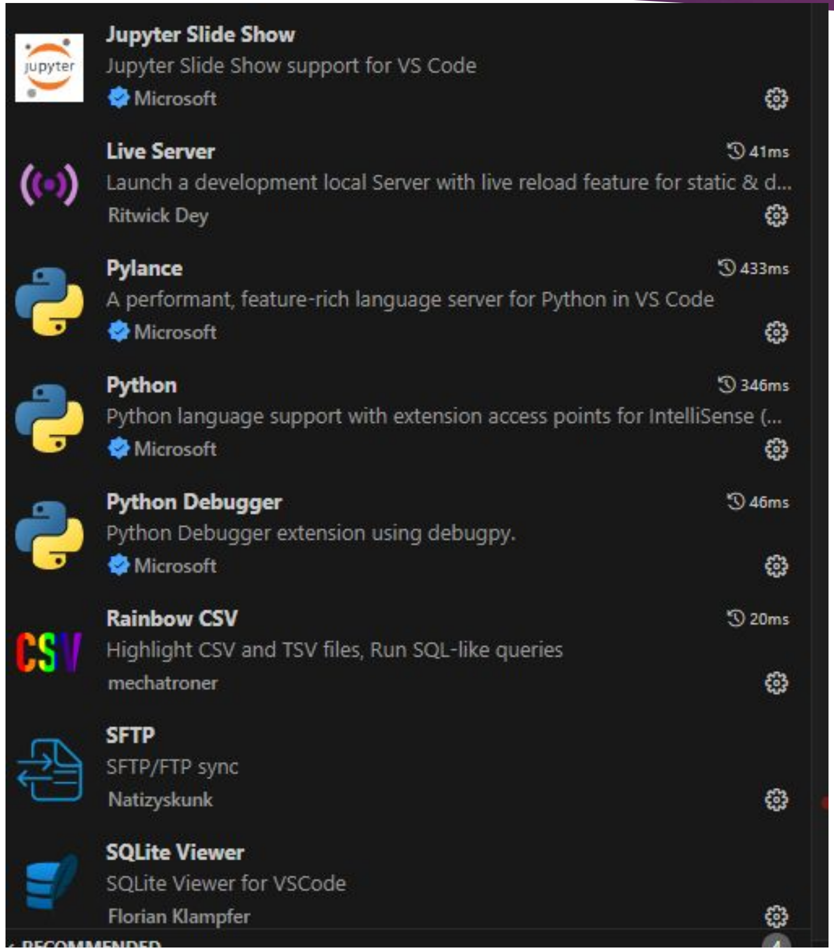
Non-interactive Docs



```
null
```

```
{
    "item": "string"
}
```

VScode extensions



Jupyter Slide Show
Jupyter Slide Show support for VS Code
Microsoft

Live Server — 41ms
Launch a development local Server with live reload feature for static & d...
Ritwick Dey

Pylance — 433ms
A performant, feature-rich language server for Python in VS Code
Microsoft

Python — 346ms
Python language support with extension access points for IntelliSense (...
Microsoft

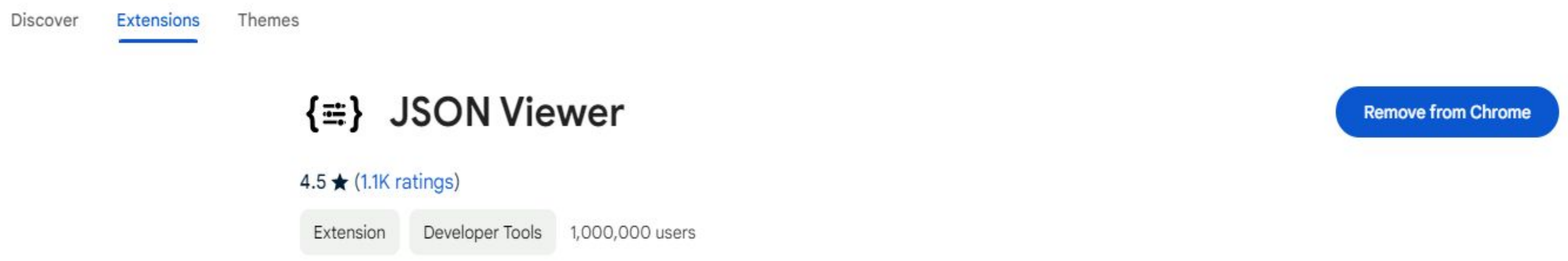
Python Debugger — 46ms
Python Debugger extension using debugpy.
Microsoft

Rainbow CSV — 20ms
Highlight CSV and TSV files, Run SQL-like queries
mechatroner

SFTP
SFTP/FTP sync
Natizyskunk

SQLite Viewer
SQLite Viewer for VSCode
Florian Klampfer

Chrome Web Store



JSON Viewer

4.5 ★ (1.1K ratings)

Extension | Developer Tools | 1,000,000 users

Building a FastAPI Application in Colab

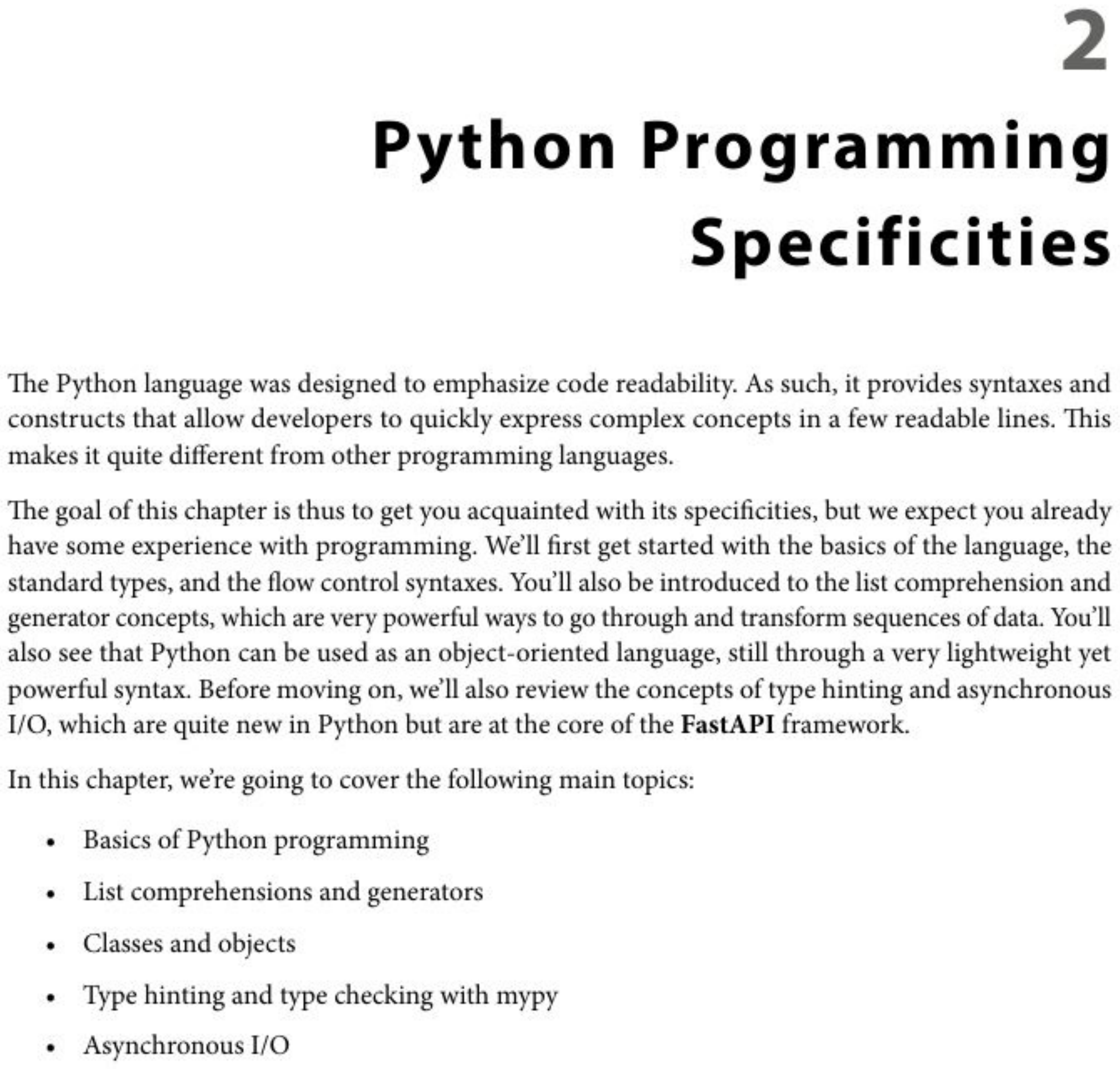
https://colab.research.google.com/drive/1VyCyyciFmOu7c8d4GL3uH0xVmiHmOHbt?hl=en#scrollTo=8K--r9DcbKzm

https://dashboard.ngrok.com/authtokens

https://youtu.be/w2htmabBMuk

https://github.com/DrMohammedhbi/fastapi_p1

CH2



2

Python Programming Specificities

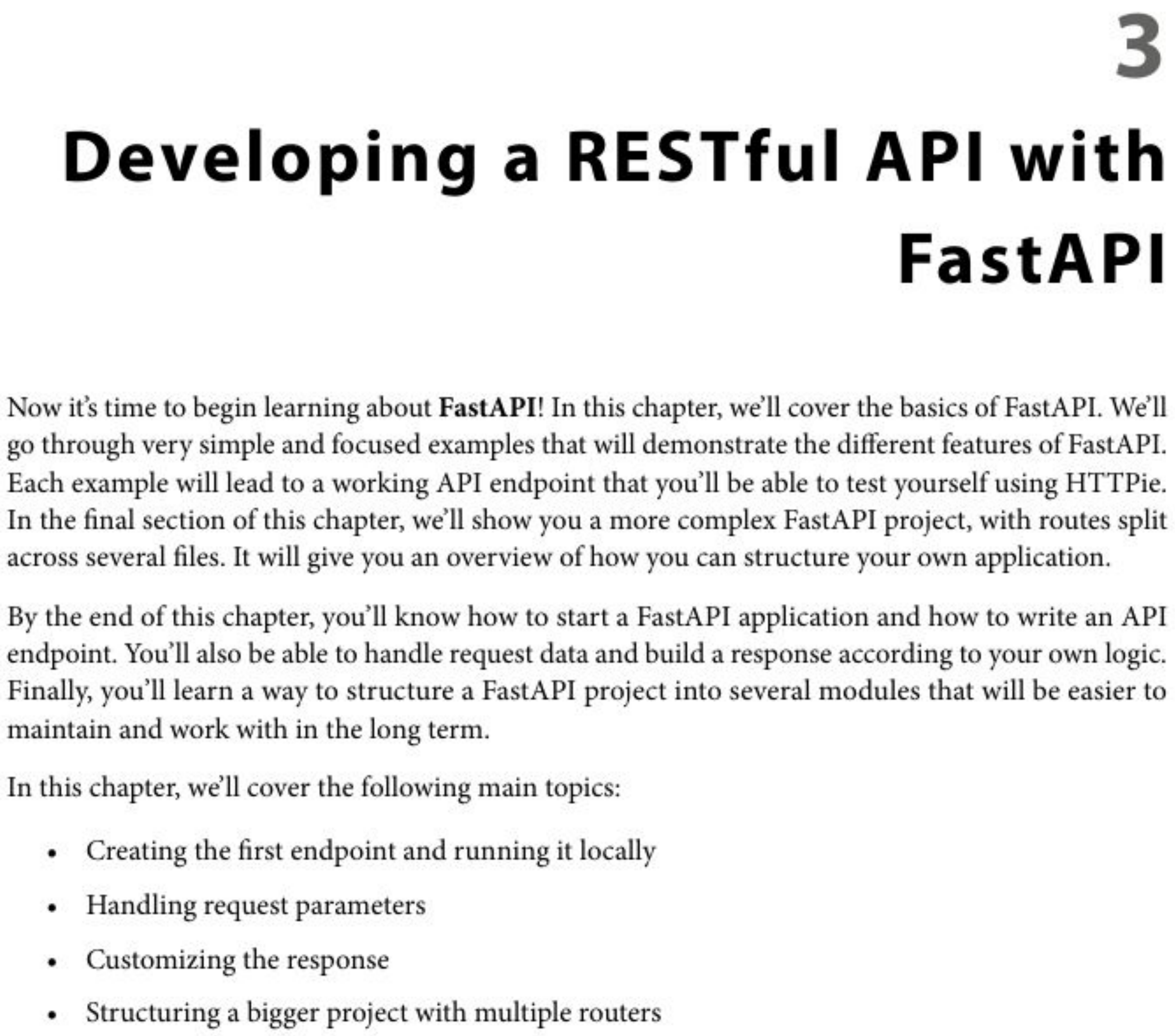
The Python language was designed to emphasize code readability. As such, it provides syntaxes and constructs that allow developers to quickly express complex concepts in a few readable lines. This makes it quite different from other programming languages.

The goal of this chapter is thus to get you acquainted with its specificities, but we expect you already have some experience with programming. We'll first get started with the basics of the language, the standard types, and the flow control syntaxes. You'll also be introduced to the list comprehension and generator concepts, which are very powerful ways to go through and transform sequences of data. You'll also see that Python can be used as an object-oriented language, still through a very lightweight yet powerful syntax. Before moving on, we'll also review the concepts of type hinting and asynchronous I/O, which are quite new in Python but are at the core of the **FastAPI** framework.

In this chapter, we're going to cover the following main topics:

- Basics of Python programming
- List comprehensions and generators
- Classes and objects
- Type hinting and type checking with mypy
- Asynchronous I/O

CH3



3

Developing a RESTful API with FastAPI

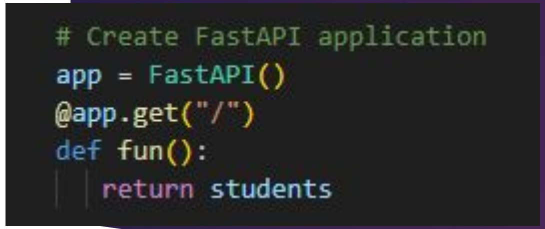
Now it's time to begin learning about **FastAPI**! In this chapter, we'll cover the basics of FastAPI. We'll go through very simple and focused examples that will demonstrate the different features of FastAPI. Each example will lead to a working API endpoint that you'll be able to test yourself using HTTPie. In the final section of this chapter, we'll show you a more complex FastAPI project, with routes split across several files. It will give you an overview of how you can structure your own application.

By the end of this chapter, you'll know how to start a FastAPI application and how to write an API endpoint. You'll also be able to handle request data and build a response according to your own logic. Finally, you'll learn a way to structure a FastAPI project into several modules that will be easier to maintain and work with in the long term.

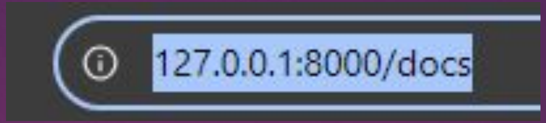
In this chapter, we'll cover the following main topics:

- Creating the first endpoint and running it locally
- Handling request parameters
- Customizing the response
- Structuring a bigger project with multiple routers

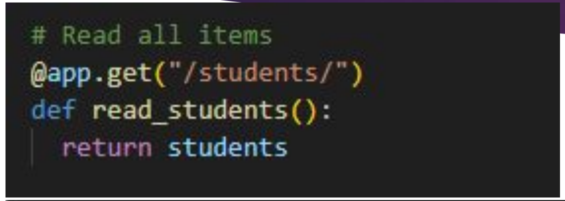
create routes



```python
# Create FastAPI application
app = FastAPI()
@app.get("/")
def fun():
    return students
```

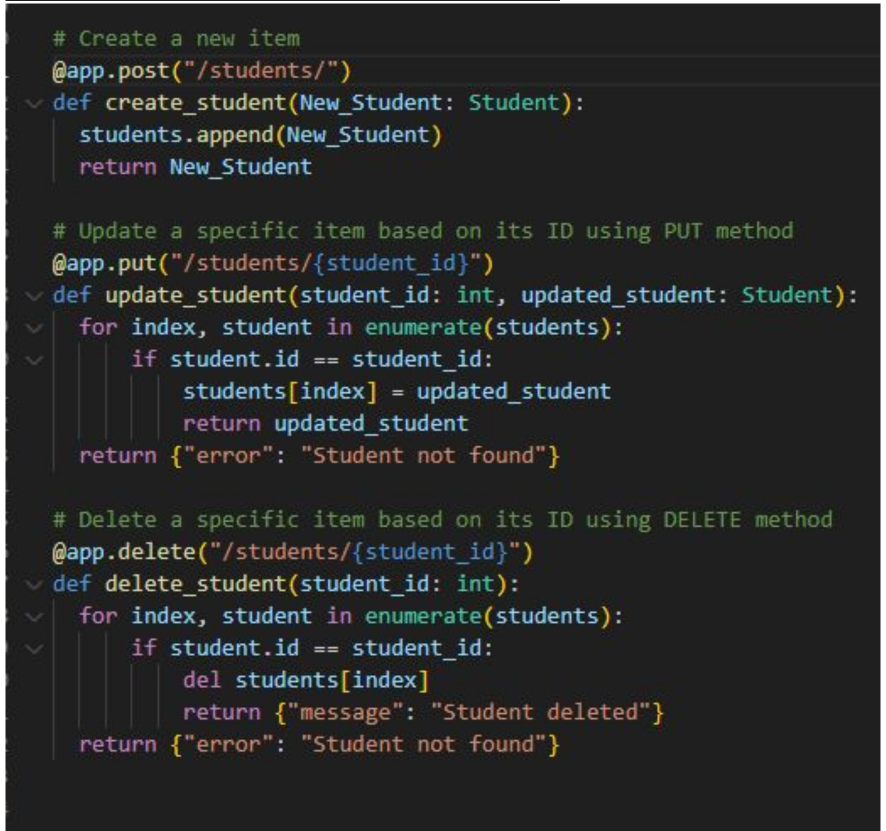


127.0.0.1:8000/docs

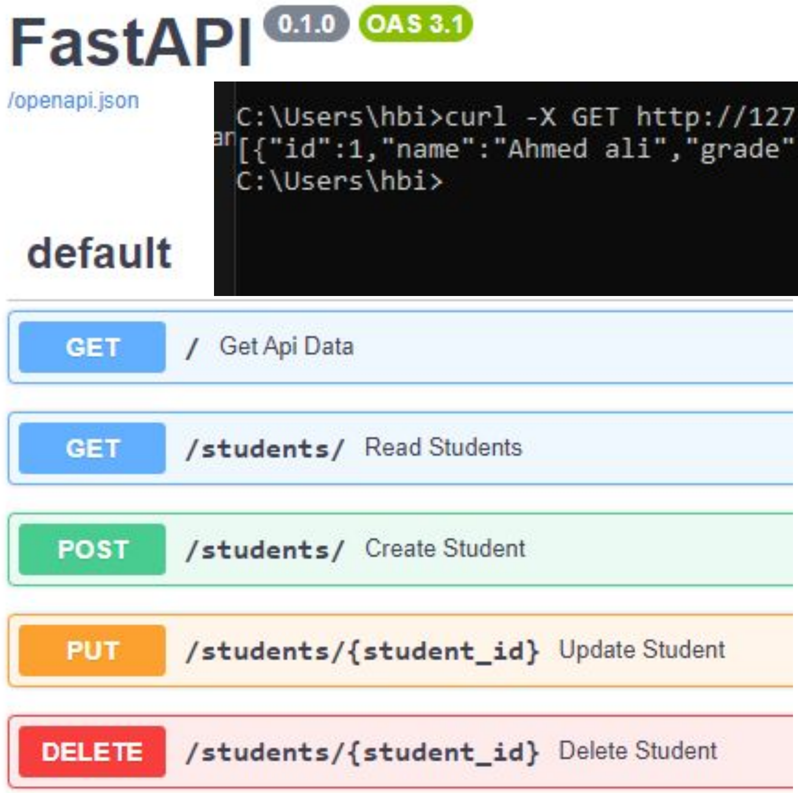


```python
# Read all items
@app.get("/students/")
def read_students():
    return students
```

routes used to define different URL that your app should respond to



```python
# Create a new item
@app.post("/students/")
def create_student(New_Student: Student):
    students.append(New_Student)
    return New_Student

# Update a specific item based on its ID using PUT method
@app.put("/students/{student_id}")
def update_student(student_id: int, updated_student: Student):
    for index, student in enumerate(students):
        if student.id == student_id:
            students[index] = updated_student
            return updated_student
    return {"error": "Student not found"}

# Delete a specific item based on its ID using DELETE method
@app.delete("/students/{student_id}")
def delete_student(student_id: int):
    for index, student in enumerate(students):
        if student.id == student_id:
            del students[index]
            return {"message": "Student deleted"}
    return {"error": "Student not found"}
```

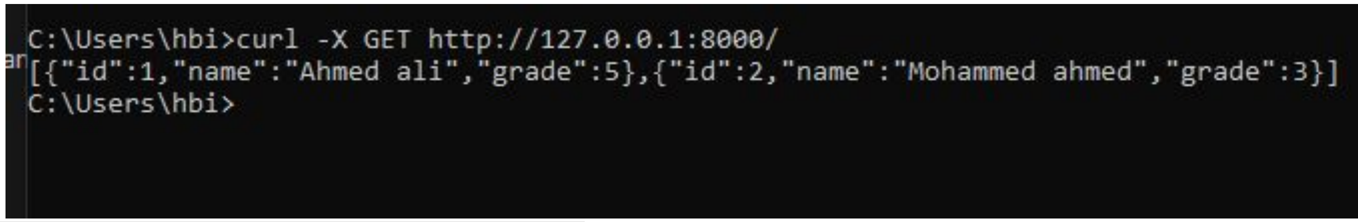


FastAPI 0.1.0 OAS 3.1

/openapi.json

default

GET	/	Get Api Data
GET	/students/	Read Students
POST	/students/	Create Student
PUT	/students/{student_id}	Update Student
DELETE	/students/{student_id}	Delete Student



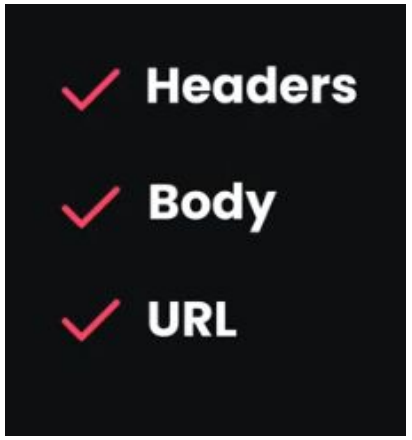
```
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/
[{"id":1,"name":"Ahmed ali","grade":5},{"id":2,"name":"Mohammed ahmed","grade":3}]
C:\Users\hbi>
```

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

curl

curl is a command line tool that allows data exchange between a device and a server.

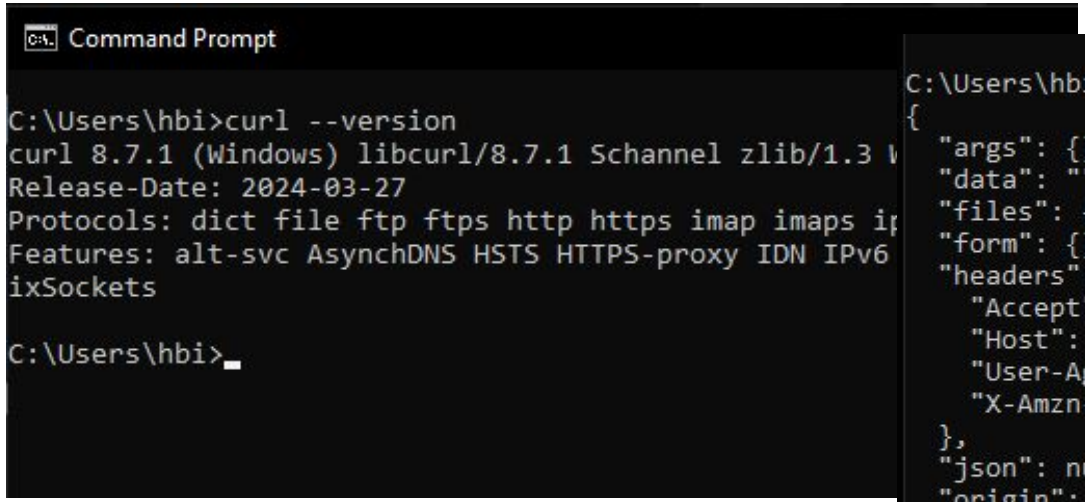
curl allows you to make requests through methods such as GET, which retrieves data



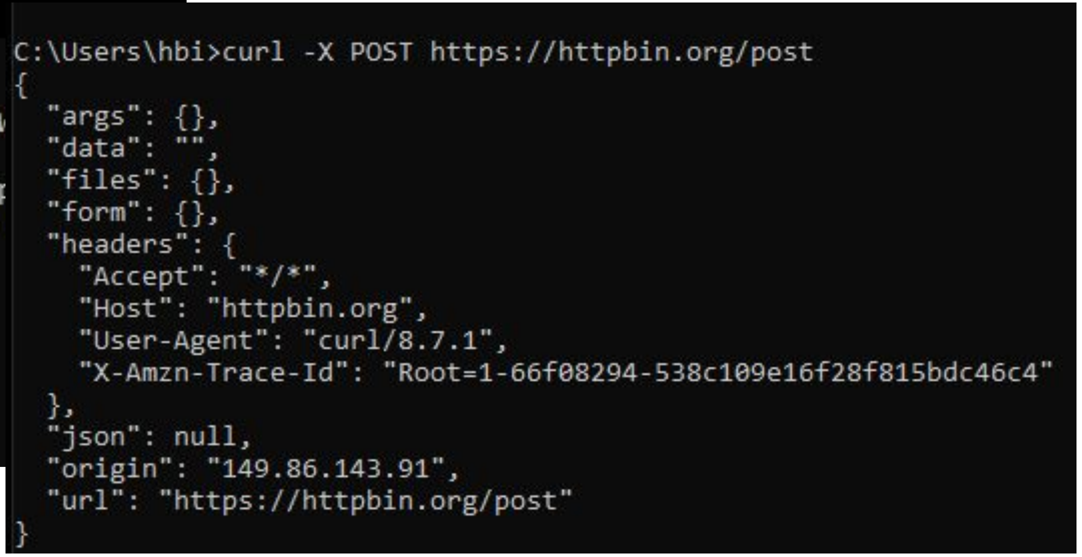
✓ **Headers**

✓ **Body**

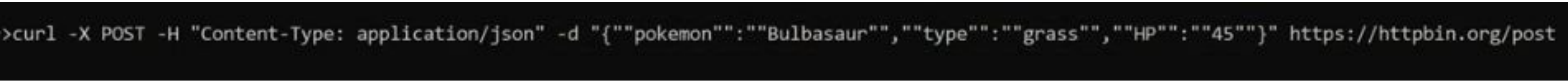
✓ **URL**



```
C:\Users\hbi>curl --version
curl 8.7.1 (Windows) libcurl/8.7.1 Schannel zlib/1.3 W
Release-Date: 2024-03-27
Protocols: dict file ftp ftps http https imap imaps ip
Features: alt-svc AsynchDNS HSTS HTTPS-proxy IDN IPv6
ixSockets

C:\Users\hbi>_
```

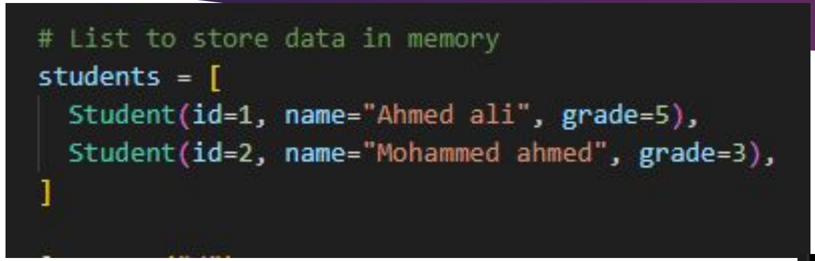


```
C:\Users\hbi>curl -X POST https://httpbin.org/post
{
  "args": {},
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/8.7.1",
    "X-Amzn-Trace-Id": "Root=1-66f08294-538c109e16f28f815bdc46c4"
  },
  "json": null,
  "origin": "149.86.143.91",
  "url": "https://httpbin.org/post"
}
```

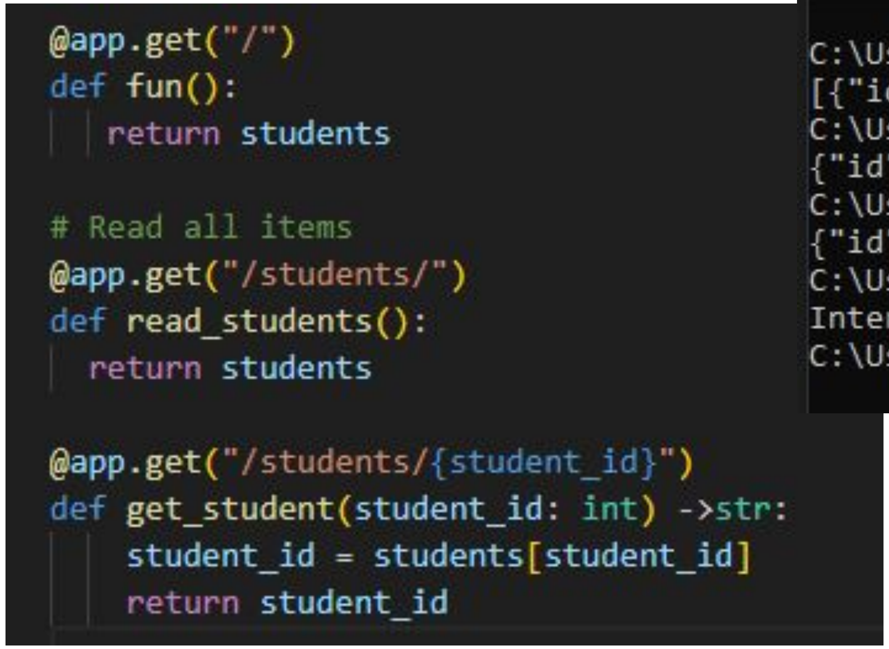


```
>curl -X POST -H "Content-Type: application/json" -d "{""pokemon"":""Bulbasaur"",""type"":""grass"",""HP"":""45""}" https://httpbin.org/post
```

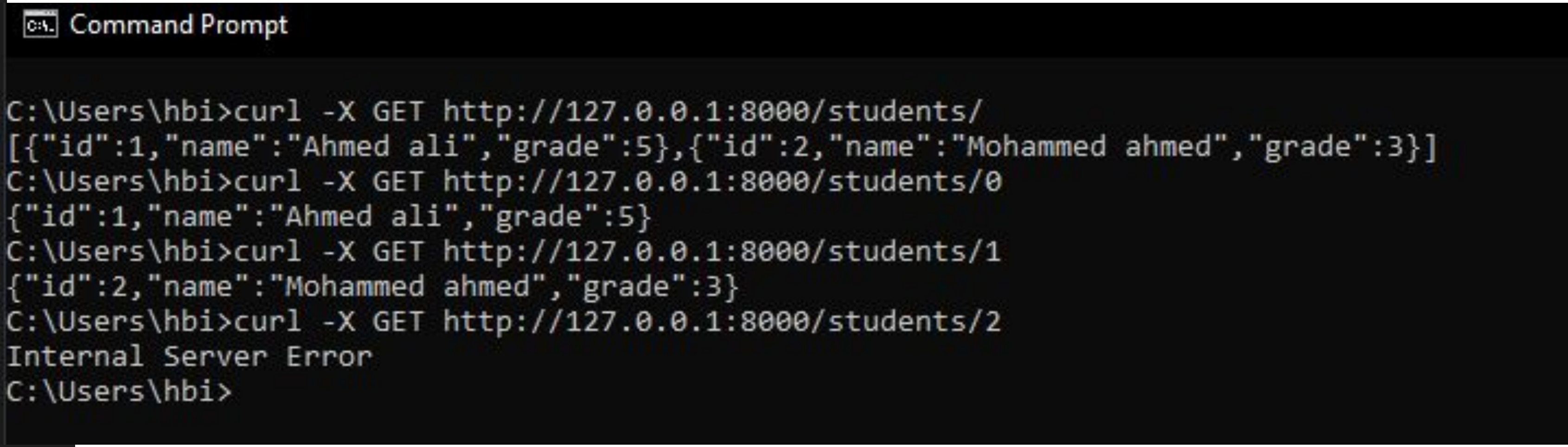
get info from routes



```python
# List to store data in memory
students = [
    Student(id=1, name="Ahmed ali", grade=5),
    Student(id=2, name="Mohammed ahmed", grade=3),
]
```

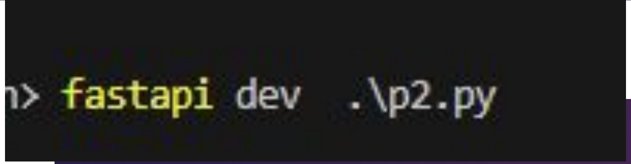


```python
@app.get("/")
def fun():
        return students

# Read all items
@app.get("/students/")
def read_students():
    return students

@app.get("/students/{student_id}")
def get_student(student_id: int) ->str:
        student_id = students[student_id]
        return student_id
```

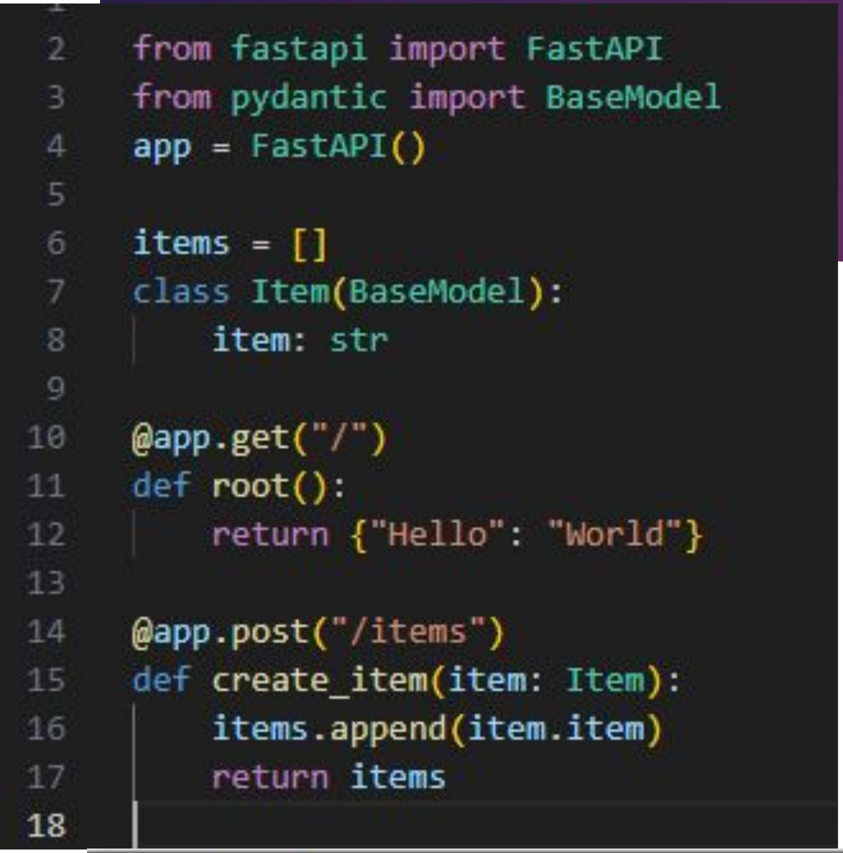


```
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/students/
[{"id":1,"name":"Ahmed ali","grade":5},{"id":2,"name":"Mohammed ahmed","grade":3}]
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/students/0
{"id":1,"name":"Ahmed ali","grade":5}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/students/1
{"id":2,"name":"Mohammed ahmed","grade":3}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/students/2
Internal Server Error
C:\Users\hbi>
```

POST data to server



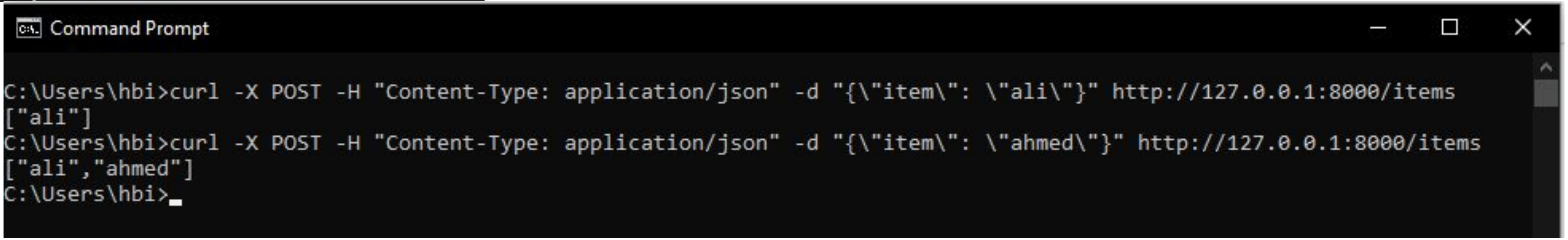
```
> fastapi dev  .\p2.py
```



```python
from fastapi import FastAPI
from pydantic import BaseModel
app = FastAPI()

items = []
class Item(BaseModel):
    item: str

@app.get("/")
def root():
    return {"Hello": "World"}

@app.post("/items")
def create_item(item: Item):
    items.append(item.item)
    return items
```

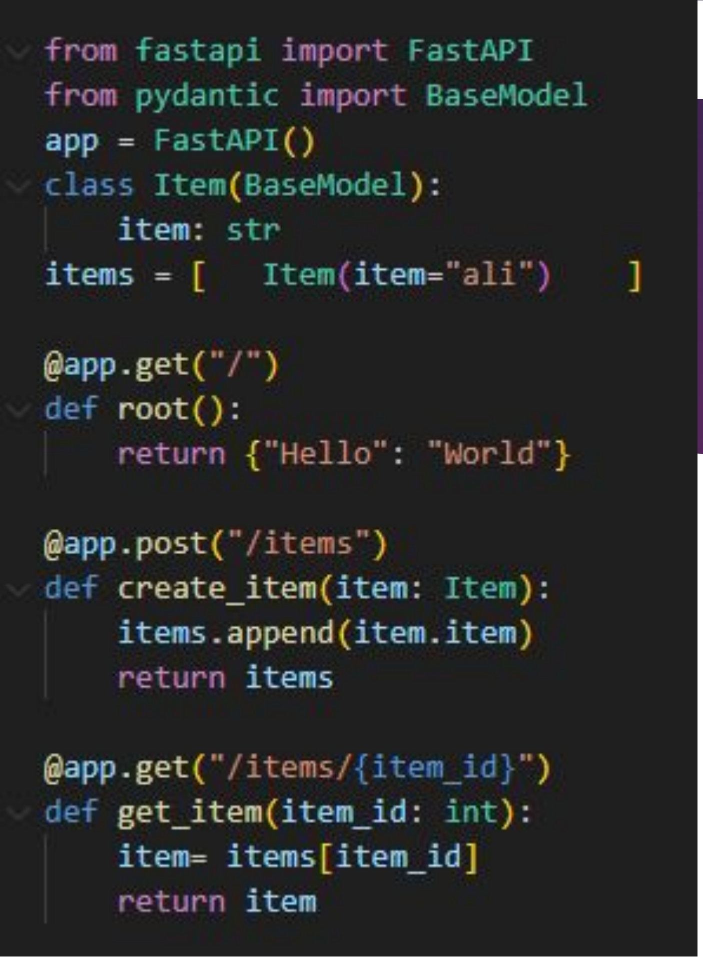
curl -X POST -H "Content-Type: application/json" -d "{\"item\": \"ahmed\"}" http://127.0.0.1:8000/items

https://ec.haxx.se/http/post/expect100.html
https://edoceo.com/sys/curl



```
C:\Users\hbi>curl -X POST -H "Content-Type: application/json" -d "{\"item\": \"ali\"}" http://127.0.0.1:8000/items
["ali"]
C:\Users\hbi>curl -X POST -H "Content-Type: application/json" -d "{\"item\": \"ahmed\"}" http://127.0.0.1:8000/items
["ali","ahmed"]
C:\Users\hbi>
```

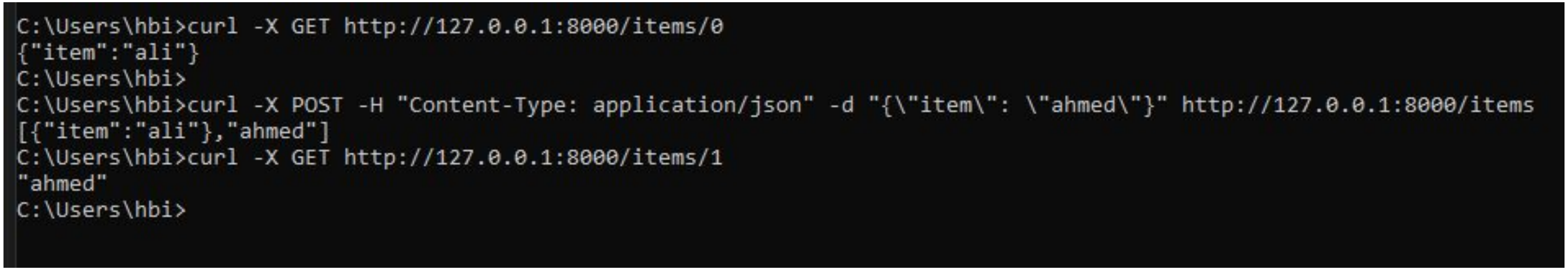
GET and POST



```python
from fastapi import FastAPI
from pydantic import BaseModel
app = FastAPI()
class Item(BaseModel):
    item: str
items = [   Item(item="ali")     ]

@app.get("/")
def root():
    return {"Hello": "World"}

@app.post("/items")
def create_item(item: Item):
    items.append(item.item)
    return items

@app.get("/items/{item_id}")
def get_item(item_id: int):
    item= items[item_id]
    return item
```

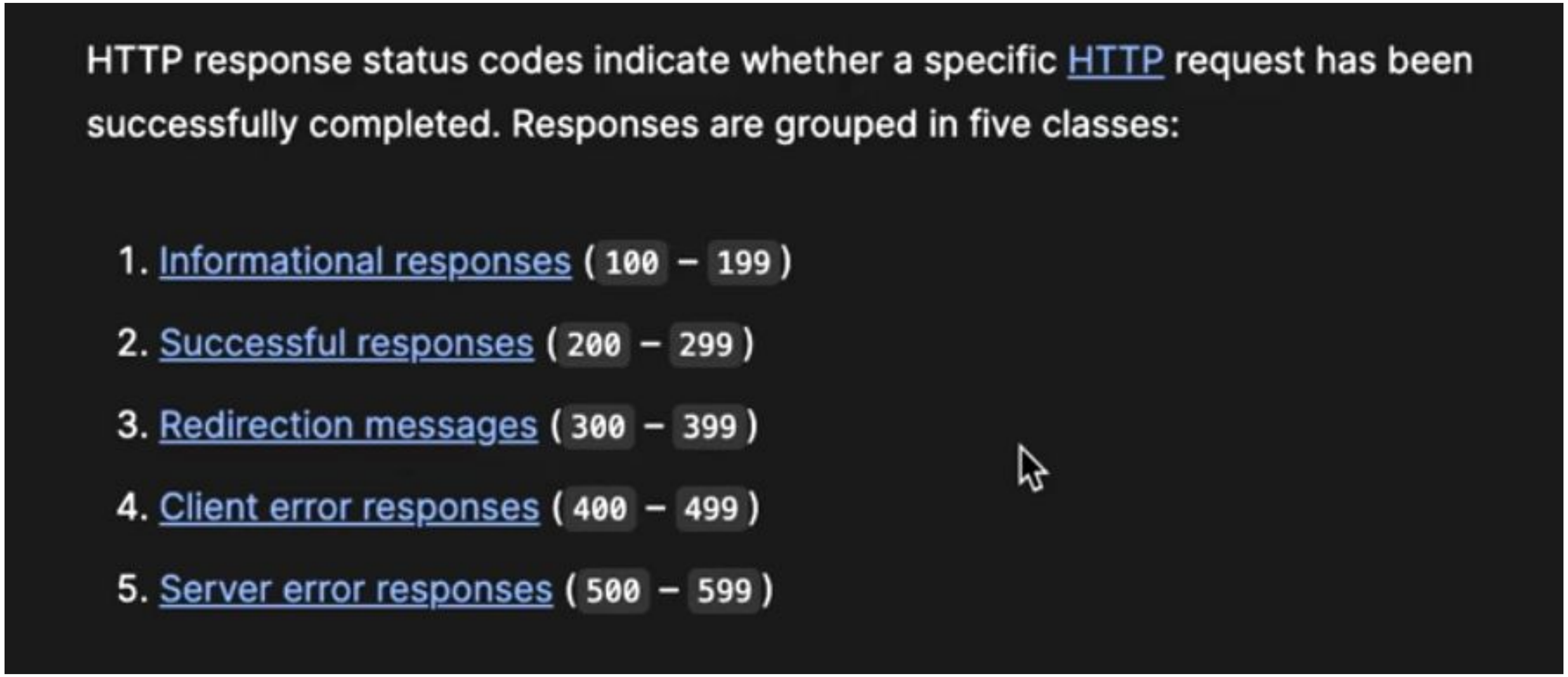
curl -X GET http://127.0.0.1:8000/items/0

curl -X POST -H "Content-Type: application/json" -d "{\"item\": \"ahmed\"}" http://127.0.0.1:8000/items



```
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/items/0
{"item":"ali"}
C:\Users\hbi>
C:\Users\hbi>curl -X POST -H "Content-Type: application/json" -d "{\"item\": \"ahmed\"}" http://127.0.0.1:8000/items
[{"item":"ali"},"ahmed"]
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/items/1
"ahmed"
C:\Users\hbi>
```

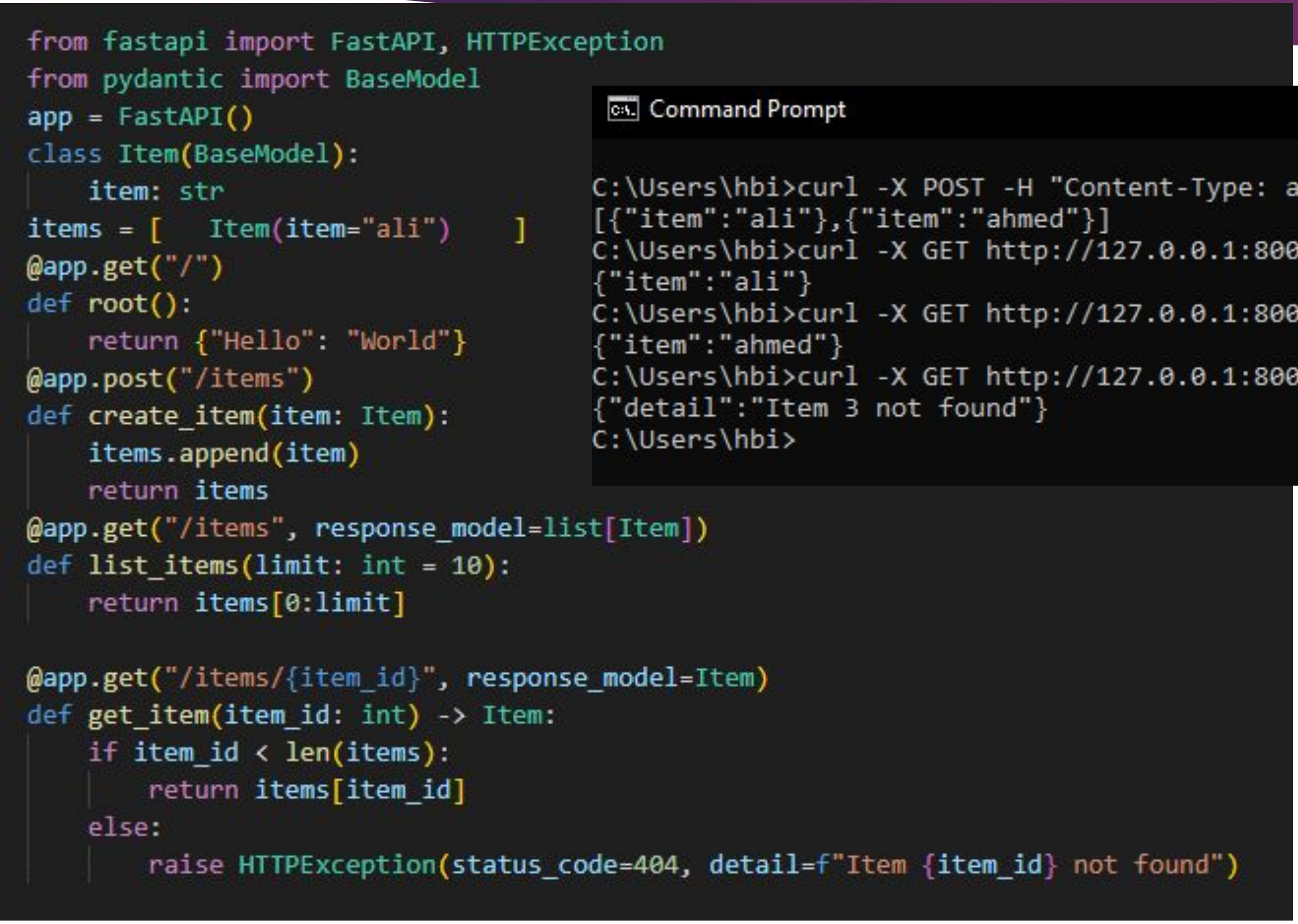
HTTP response status codes



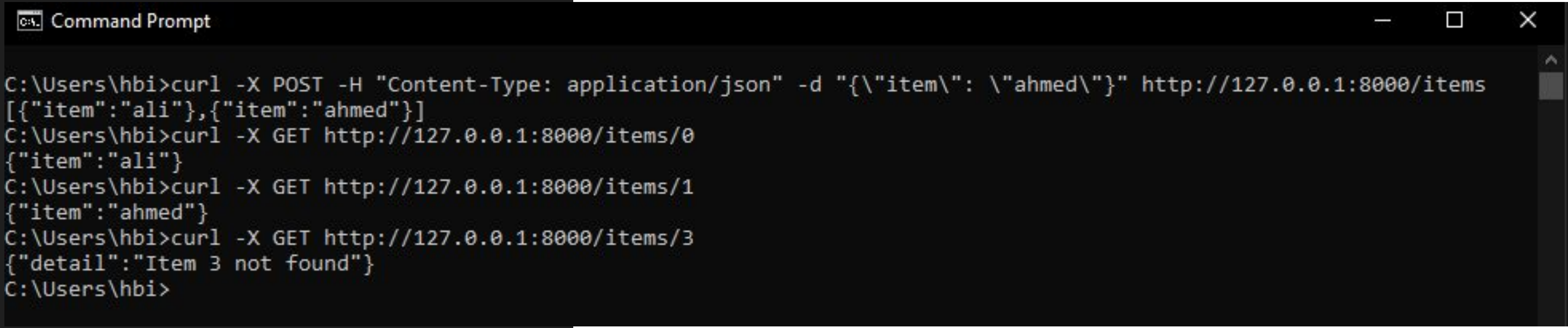
HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. Informational responses (`100` – `199`)
2. Successful responses (`200` – `299`)
3. Redirection messages (`300` – `399`)
4. Client error responses (`400` – `499`)
5. Server error responses (`500` – `599`)

HTTPException: client error 404 not found



```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
app = FastAPI()
class Item(BaseModel):
    item: str
items = [   Item(item="ali")    ]
@app.get("/")
def root():
    return {"Hello": "World"}
@app.post("/items")
def create_item(item: Item):
    items.append(item)
    return items
@app.get("/items", response_model=list[Item])
def list_items(limit: int = 10):
    return items[0:limit]

@app.get("/items/{item_id}", response_model=Item)
def get_item(item_id: int) -> Item:
    if item_id < len(items):
        return items[item_id]
    else:
        raise HTTPException(status_code=404, detail=f"Item {item_id} not found")
```



Command Prompt

```
C:\Users\hbi>curl -X POST -H "Content-Type: application/json" -d "{\"item\": \"ahmed\"}" http://127.0.0.1:8000/items
[{"item":"ali"},{"item":"ahmed"}]
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/items/0
{"item":"ali"}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/items/1
{"item":"ahmed"}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/items/3
{"detail":"Item 3 not found"}
C:\Users\hbi>
```

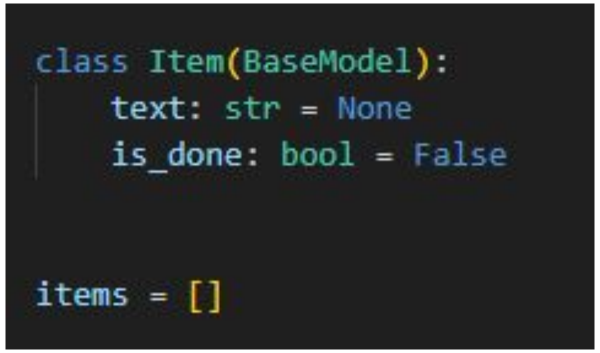
curl -X POST -H "Content-Type: application/json" -d "{\"item\": \"ahmed\"}" http://127.0.0.1:8000/items

curl -X GET http://127.0.0.1:8000/items/3

curl -X GET http://127.0.0.1:8000/items?limit=5

pydantic: validate requests

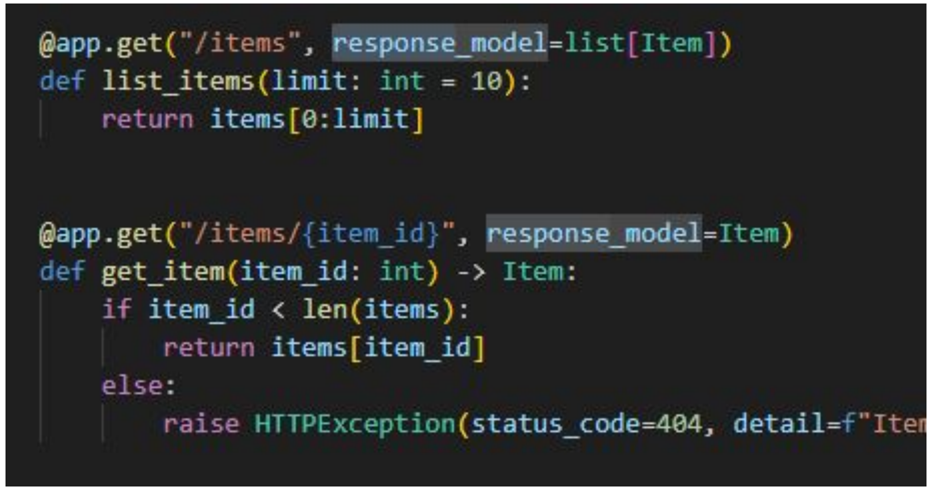
in the class we can set the item required or not (=None) or by default (=False)



```python
class Item(BaseModel):
    text: str = None
    is_done: bool = False


items = []
```

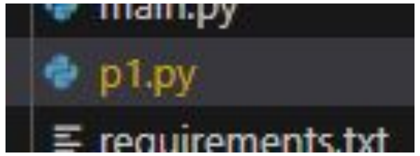
pydantic: validate response

we can control response to list or class from server

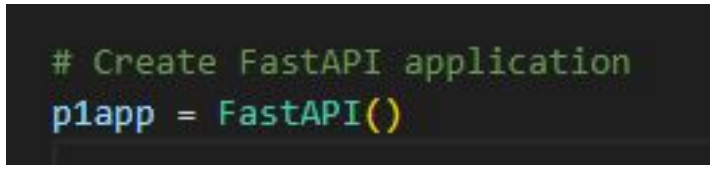


```
@app.get("/items", response_model=list[Item])
def list_items(limit: int = 10):
    return items[0:limit]


@app.get("/items/{item_id}", response_model=Item)
def get_item(item_id: int) -> Item:
    if item_id < len(items):
        return items[item_id]
    else:
        raise HTTPException(status_code=404, detail=f"Iten
```

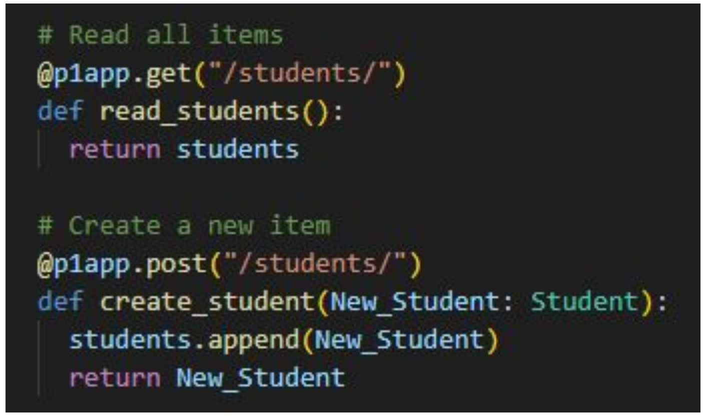
create and run a new app “p1app”



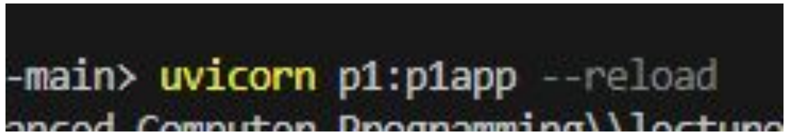
```
main.py
p1.py
requirements.txt
```



```
# Create FastAPI application
p1app = FastAPI()
```



```
# Read all items
@p1app.get("/students/")
def read_students():
    return students

# Create a new item
@p1app.post("/students/")
def create_student(New_Student: Student):
    students.append(New_Student)
    return New_Student
```



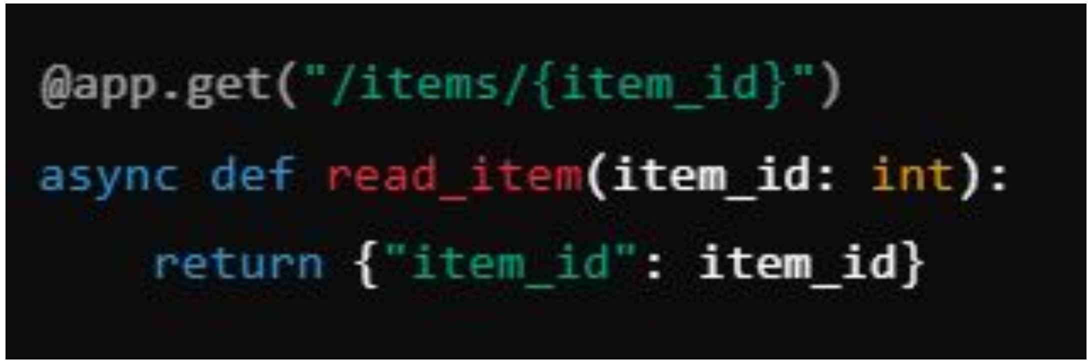
```
-main> uvicorn p1:p1app --reload
```

FastAPI Request Handling (RESTful)

Explaining HTTP methods:

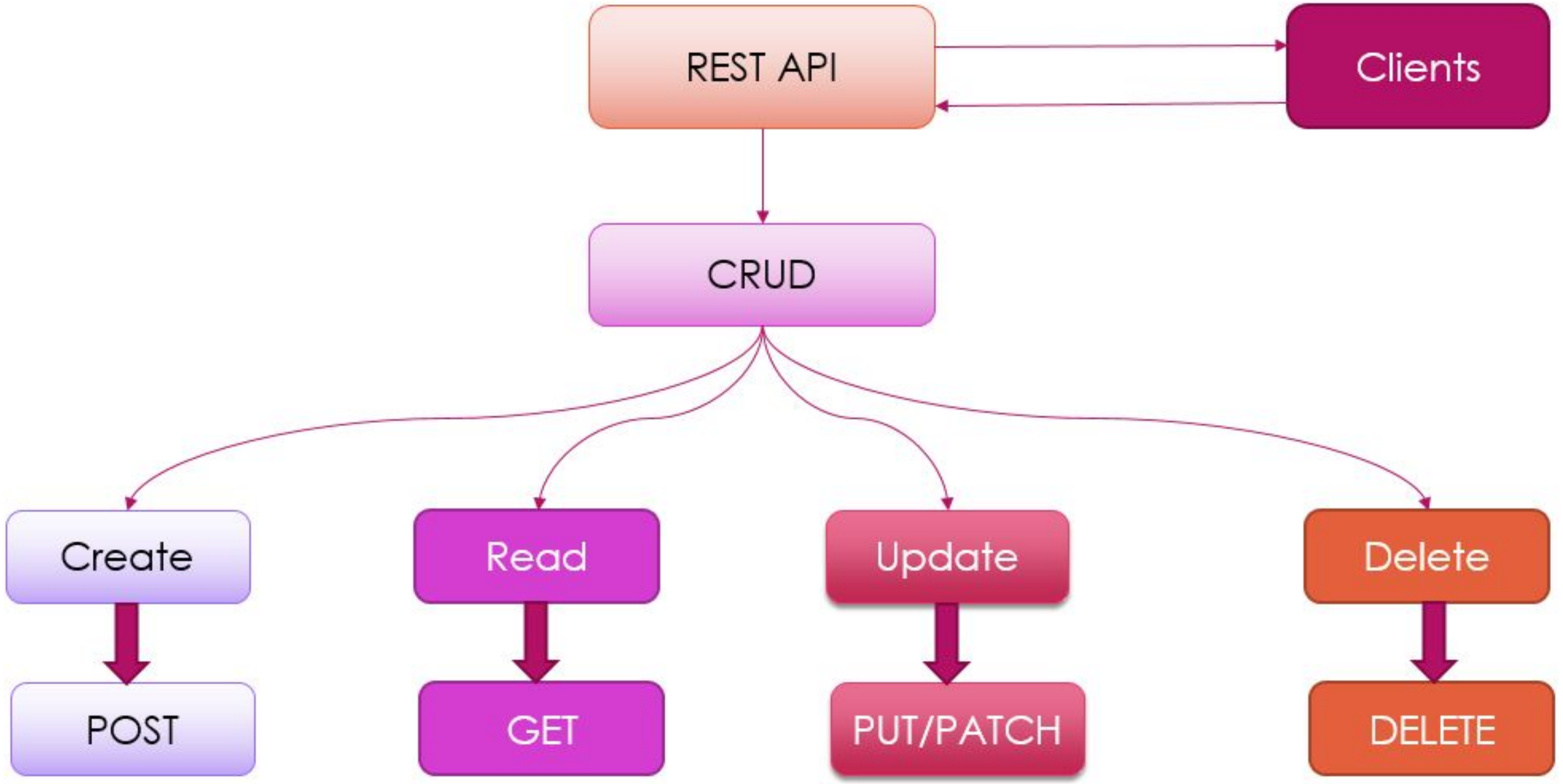
- `GET`: Fetch data
- `POST`: Send data
- `PUT`: Update data
- `DELETE`: Remove data

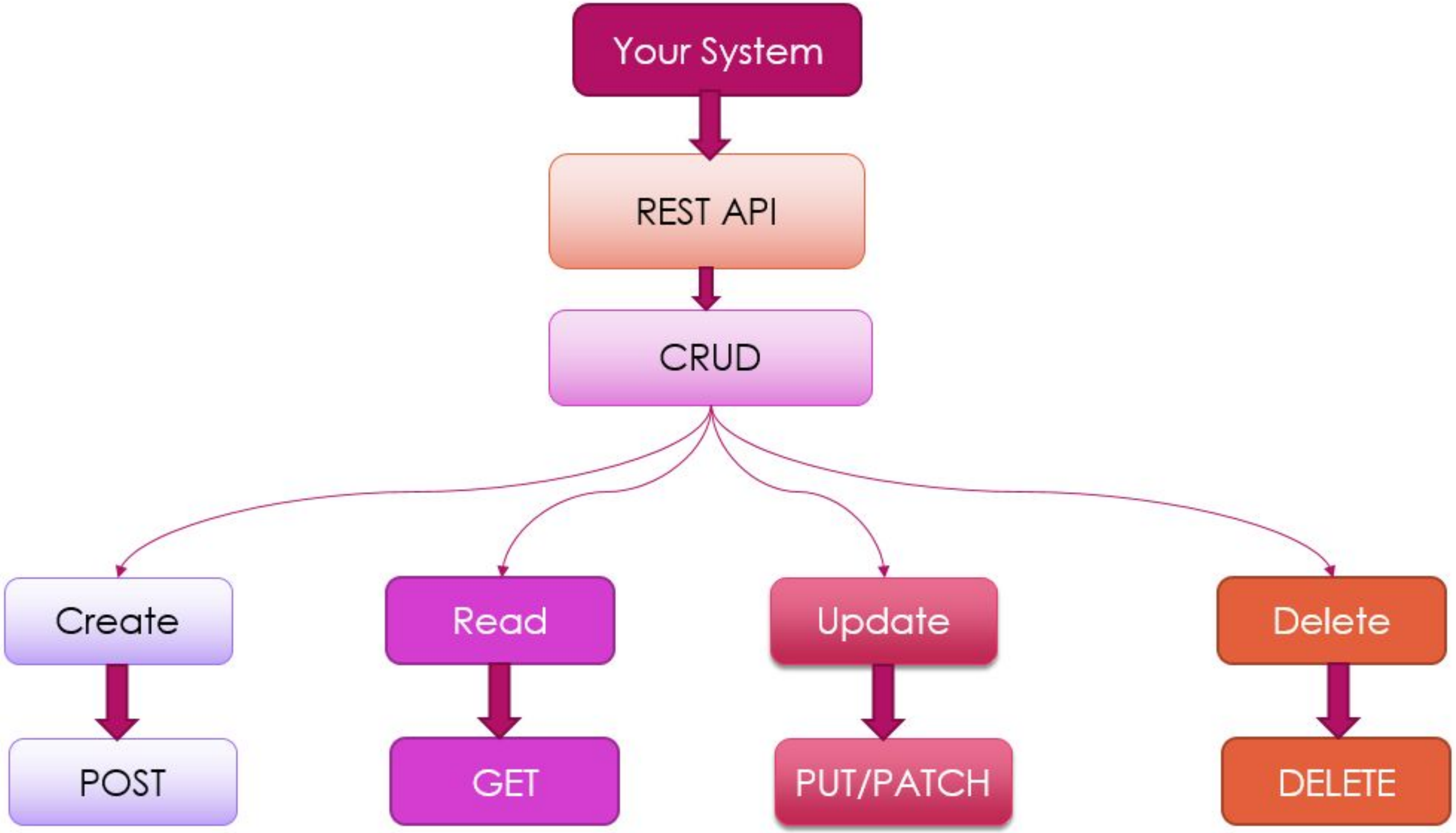
- Basic Example:

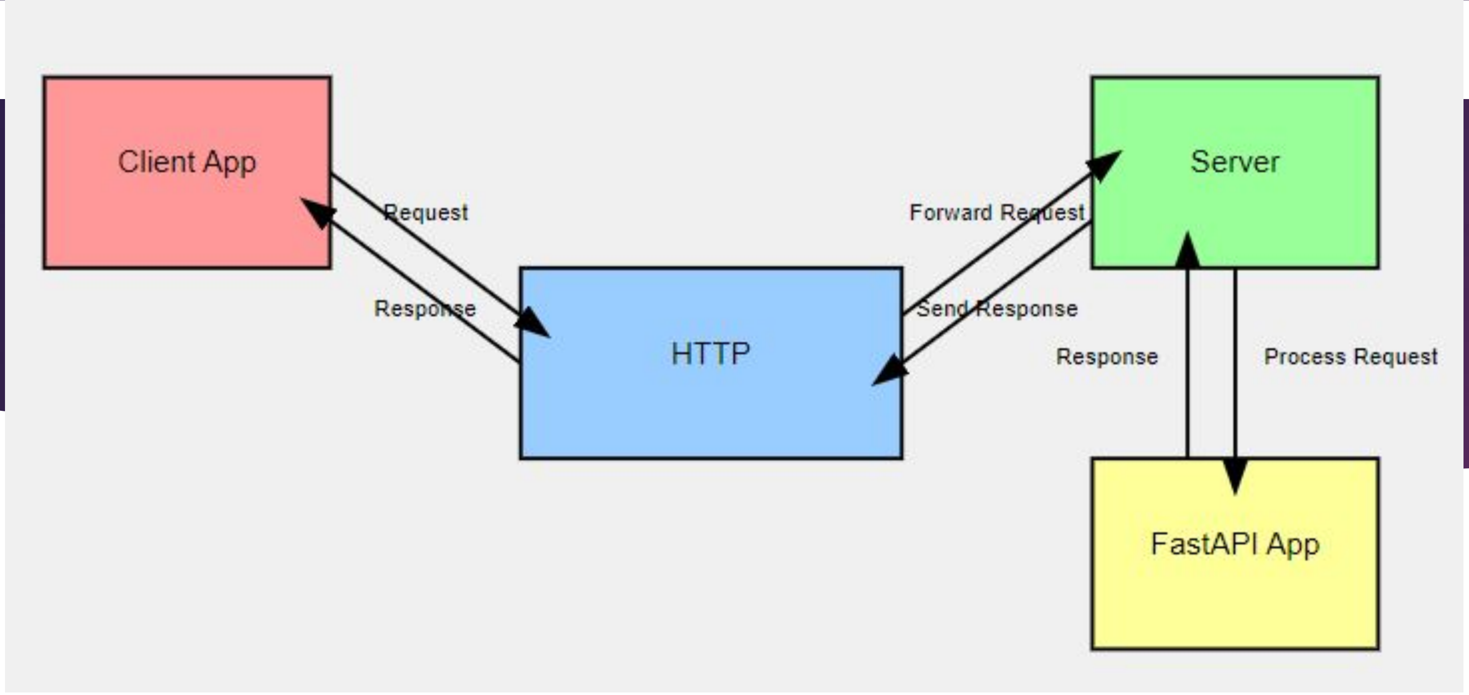


```
@app.get("/items/{item_id}")
async def read_item(item_id: int):
    return {"item_id": item_id}
```

Representational State Transfer (REST)

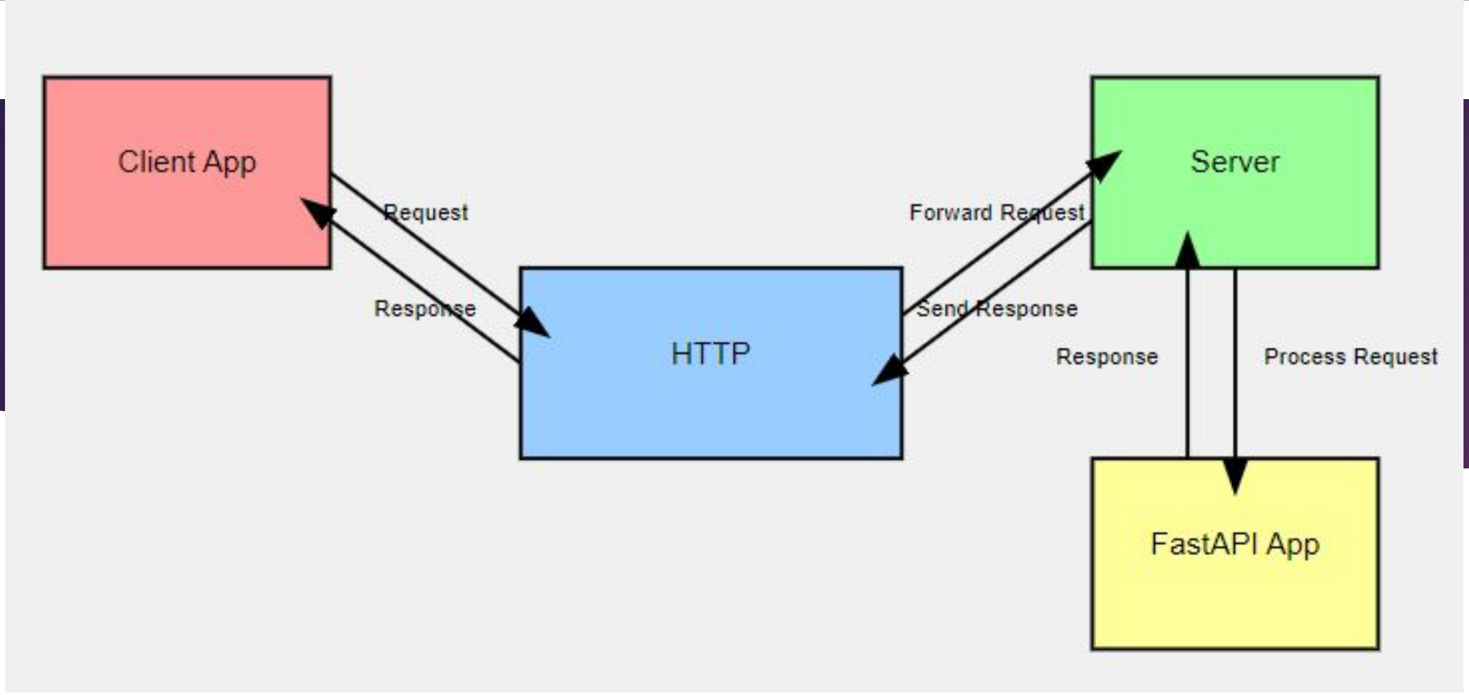



Your System
REST API
CRUD
Create
Read
Update
Delete
POST
GET
PUT/PATCH
DELETE



The diagram shows the interactions between a FastAPI app, client app, HTTP, and server :

1. Client App: This is the application that initiates requests, typically a web browser or a mobile app.
2. HTTP: This represents the HTTP protocol, which facilitates communication between the client and server.
3. Server: This is the web server that handles incoming HTTP requests and routes them to the appropriate application.
4. FastAPI App: This is your FastAPI application running on the server, which processes requests and generates responses.

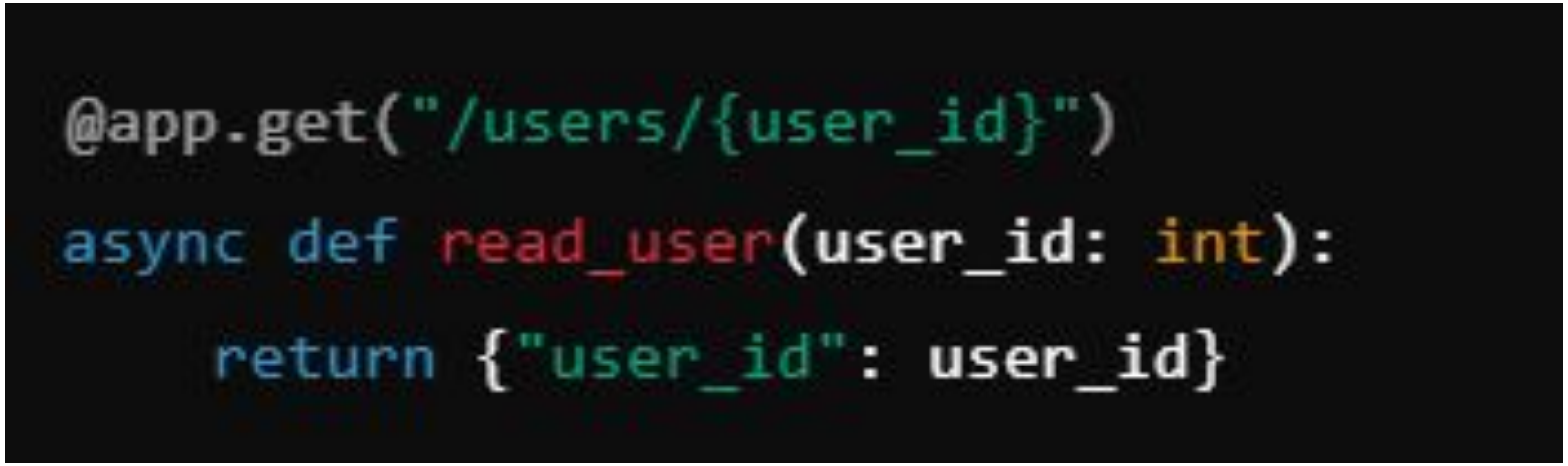


The arrows in the diagram show the flow of communication:

1. The Client App sends a request via HTTP.
2. HTTP forwards this request to the Server.
3. The Server passes the request to the FastAPI App for processing.
4. The FastAPI App processes the request and sends a response back to the Server.
5. The Server sends the response back through HTTP.
6. HTTP delivers the response to the Client App.

Path Parameters

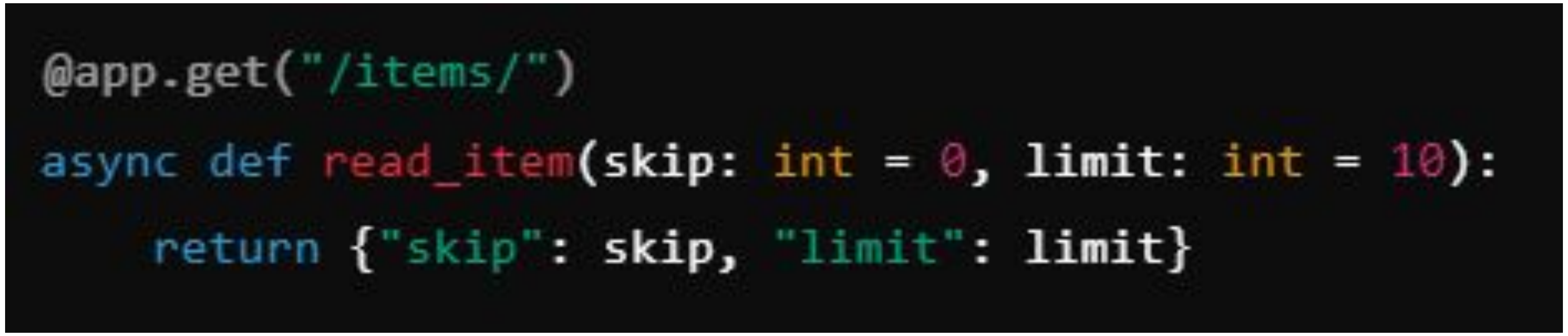
- **Explanation:** Path parameters are used to capture dynamic segments of a URL.



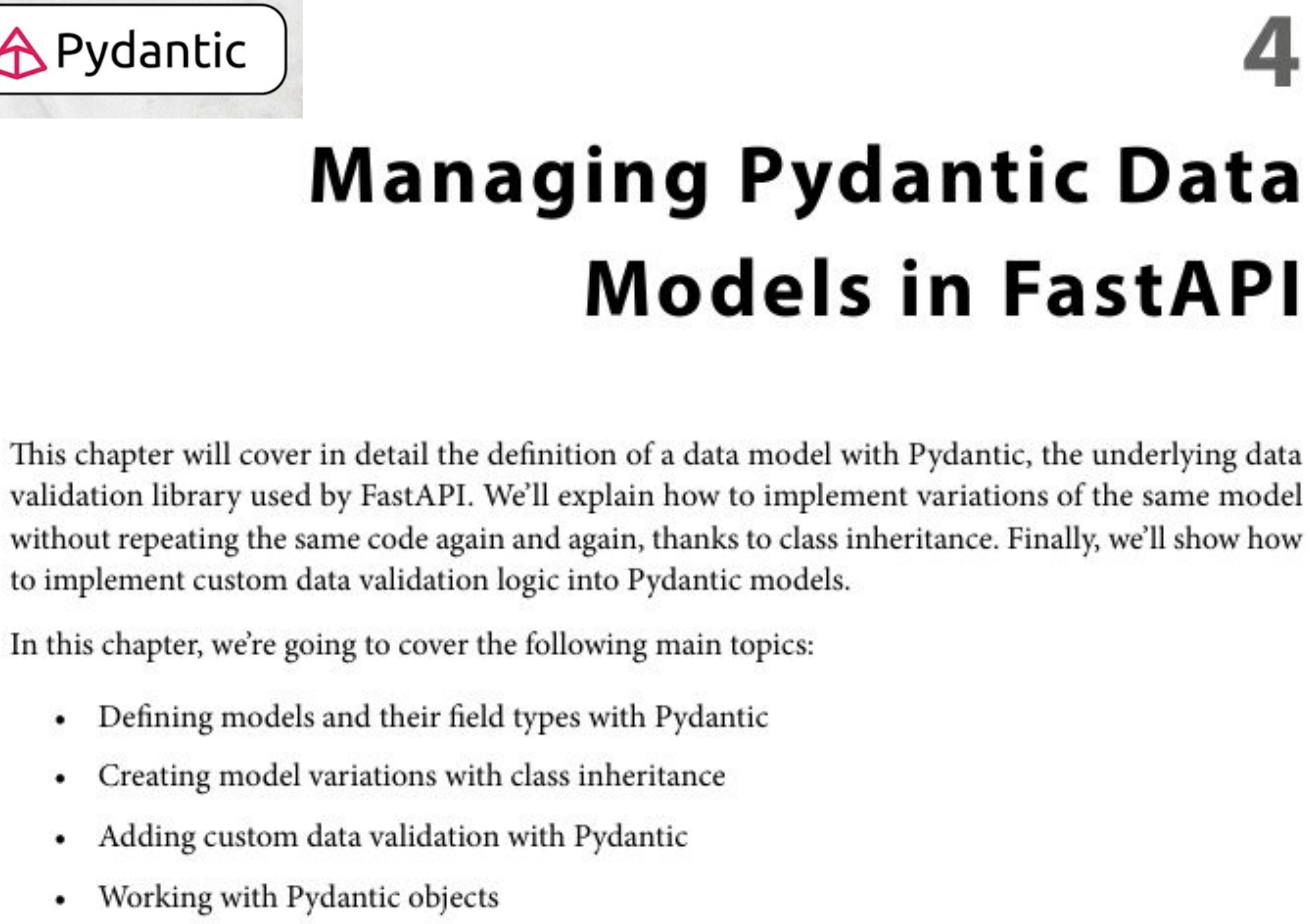
```
@app.get("/users/{user_id}")
async def read_user(user_id: int):
    return {"user_id": user_id}
```

Query Parameters

- **Explanation:** Optional query parameters allow filtering or pagination.



```python
@app.get("/items/")
async def read_item(skip: int = 0, limit: int = 10):
    return {"skip": skip, "limit": limit}
```



4

Managing Pydantic Data Models in FastAPI

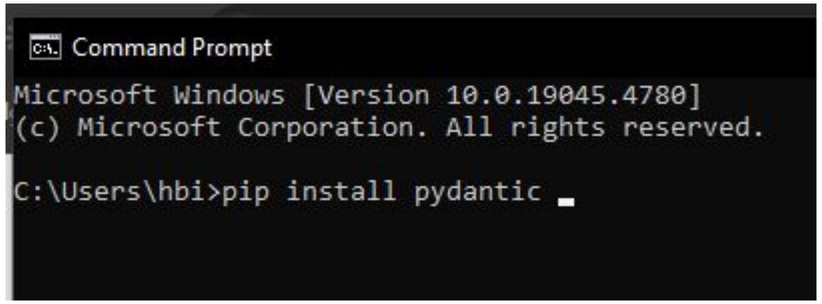
This chapter will cover in detail the definition of a data model with Pydantic, the underlying data validation library used by FastAPI. We'll explain how to implement variations of the same model without repeating the same code again and again, thanks to class inheritance. Finally, we'll show how to implement custom data validation logic into Pydantic models.

In this chapter, we're going to cover the following main topics:

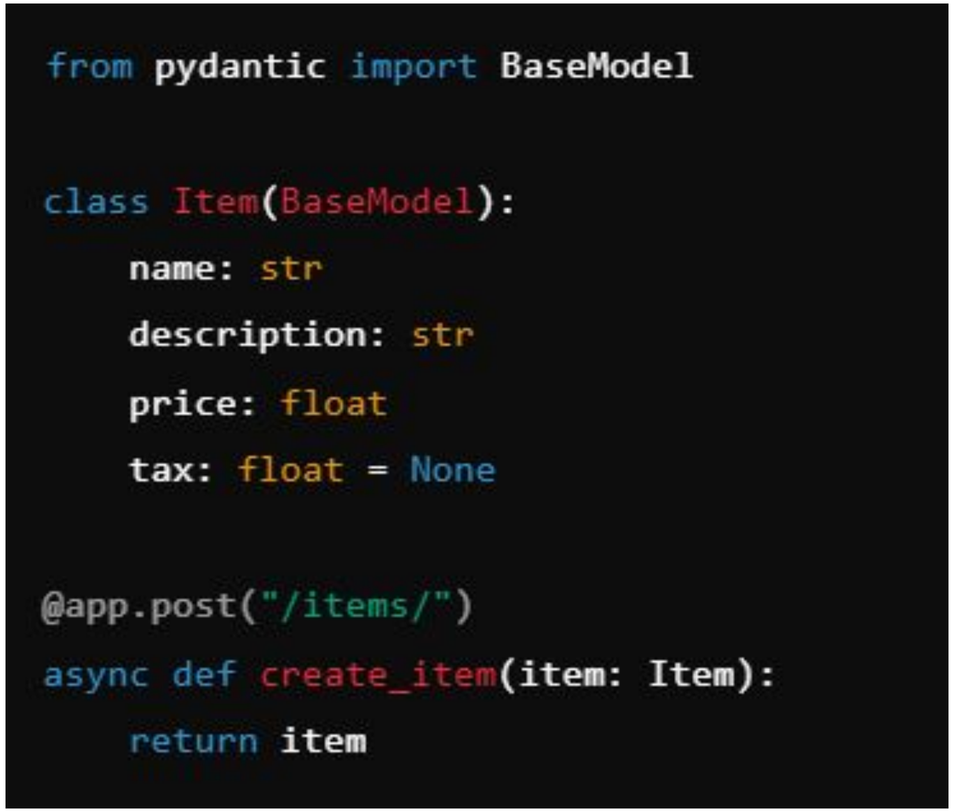
- Defining models and their field types with Pydantic
- Creating model variations with class inheritance
- Adding custom data validation with Pydantic
- Working with Pydantic objects

Request Body (POST Method)

- **Explanation:** Uses Pydantic models to define and validate the request body.



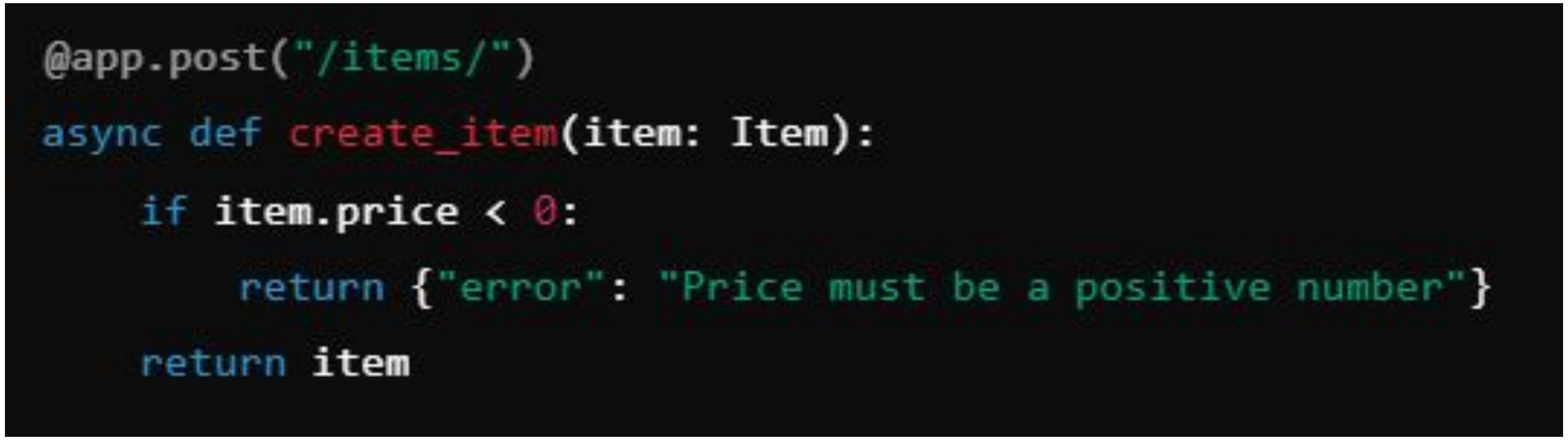
```
Command Prompt
Microsoft Windows [Version 10.0.19045.4780]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hbi>pip install pydantic _
```



```python
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    description: str
    price: float
    tax: float = None

@app.post("/items/")
async def create_item(item: Item):
    return item
```

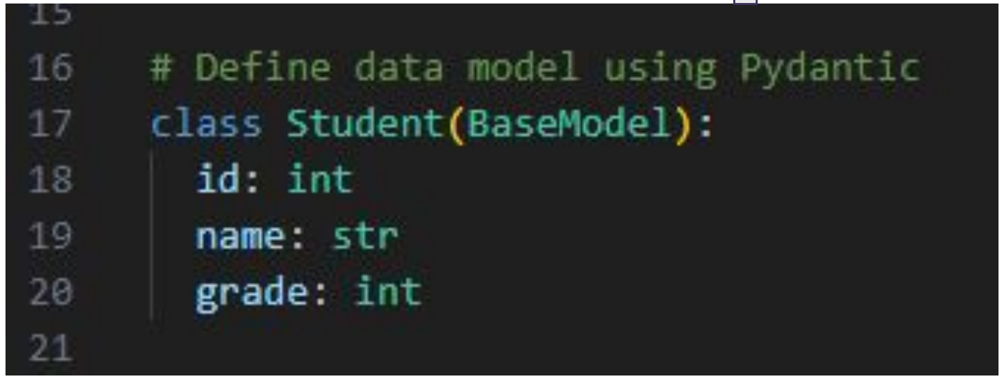
Data Validation with Pydantic

- **Explanation:** Pydantic ensures data validation with Python type.
- Code Example:

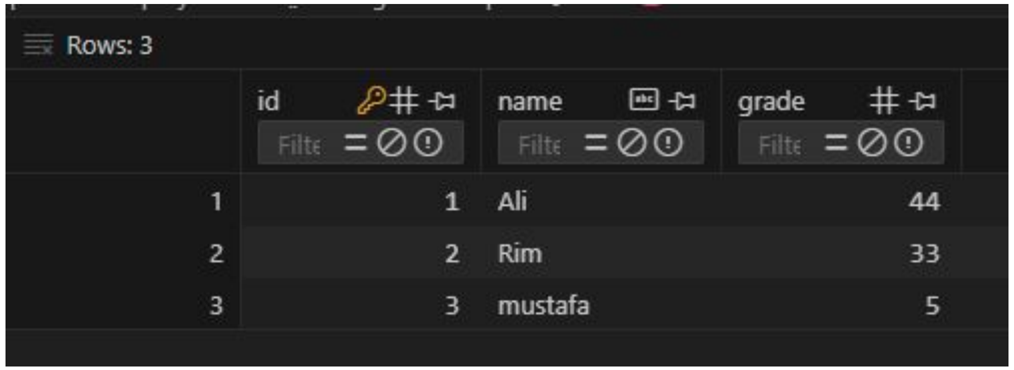


```
@app.post("/items/")
async def create_item(item: Item):
    if item.price < 0:
        return {"error": "Price must be a positive number"}
    return item
```

pedantic is a tool to model your data to solve python dynamic data types



```python
# Define data model using Pydantic
class Student(BaseModel):
    id: int
    name: str
    grade: int
```

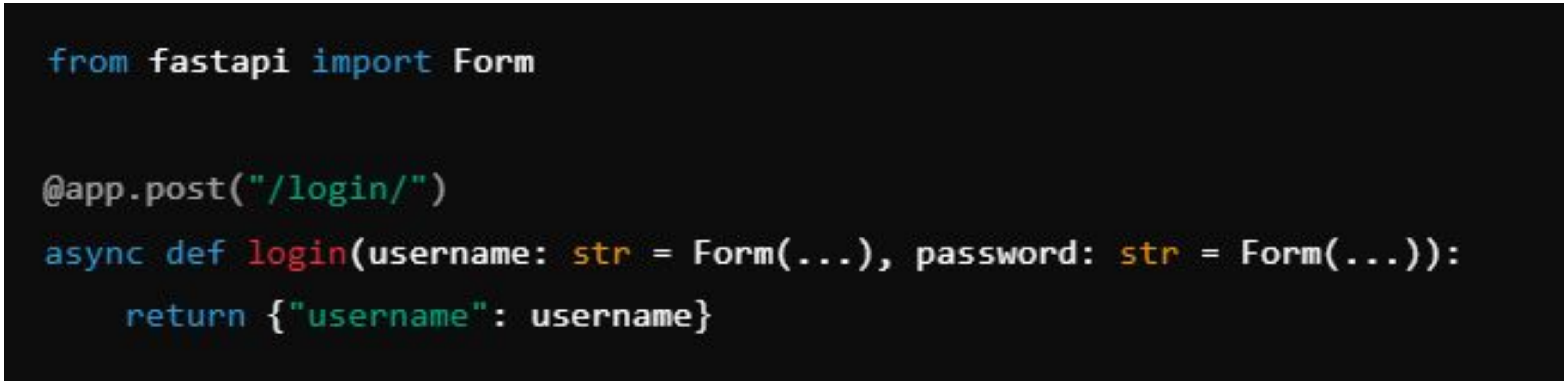


Rows: 3

	id	name	grade
1	1	Ali	44
2	2	Rim	33
3	3	mustafa	5

Form Data

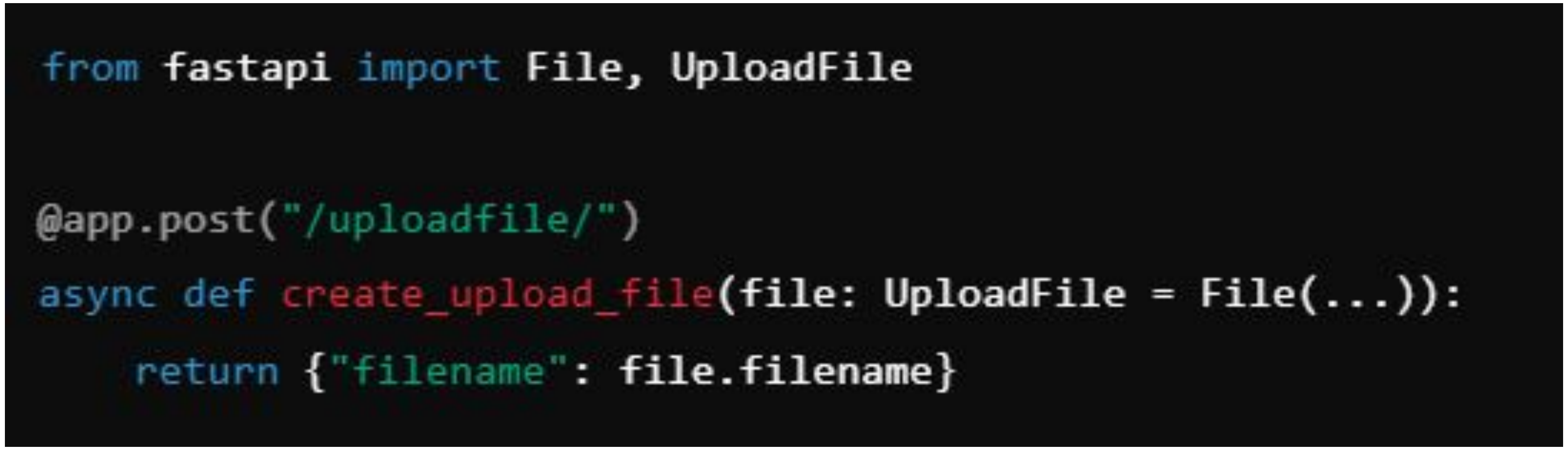
- **Explanation:** FastAPI supports form data inputs with the `Form` class.
- Code Example:



```
from fastapi import Form

@app.post("/login/")
async def login(username: str = Form(...), password: str = Form(...)):
    return {"username": username}
```

File Uploads

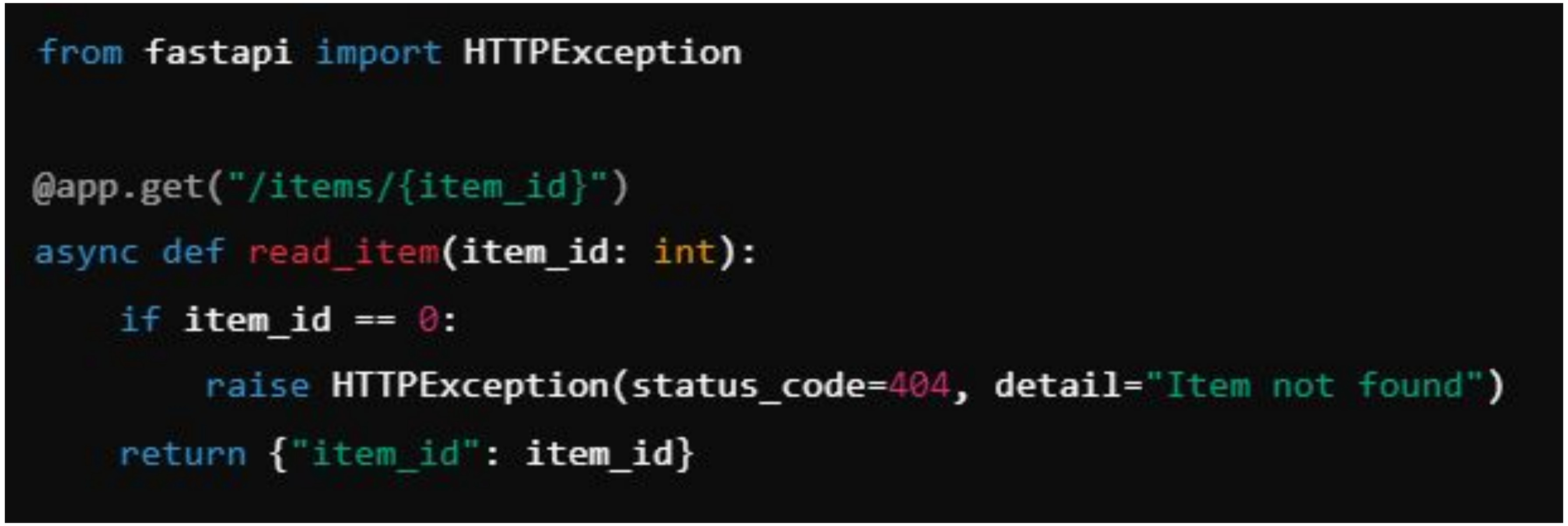
- Code Example:



```
from fastapi import File, UploadFile

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

Handling Errors

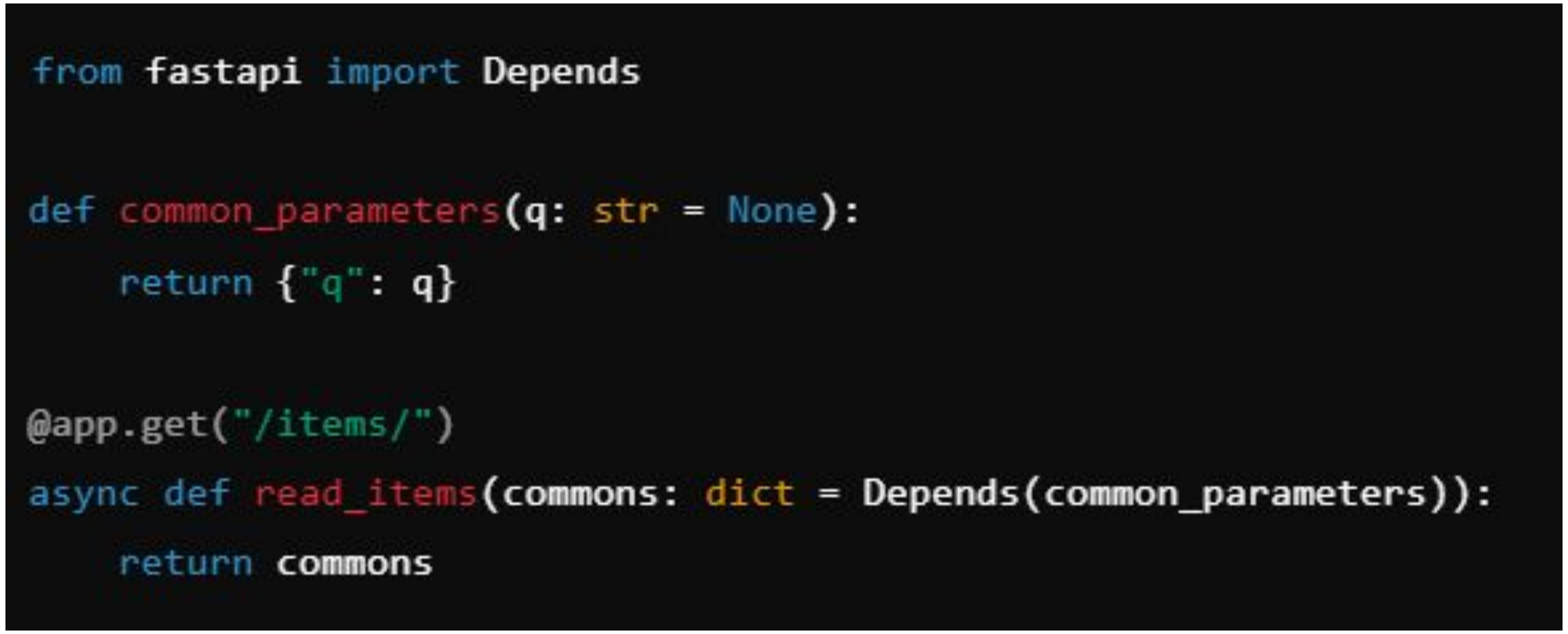
- **Explanation:** Use `HTTPException` for proper error handling.
- Code Example:



```python
from fastapi import HTTPException

@app.get("/items/{item_id}")
async def read_item(item_id: int):
    if item_id == 0:
        raise HTTPException(status_code=404, detail="Item not found")
    return {"item_id": item_id}
```

Dependency Injection

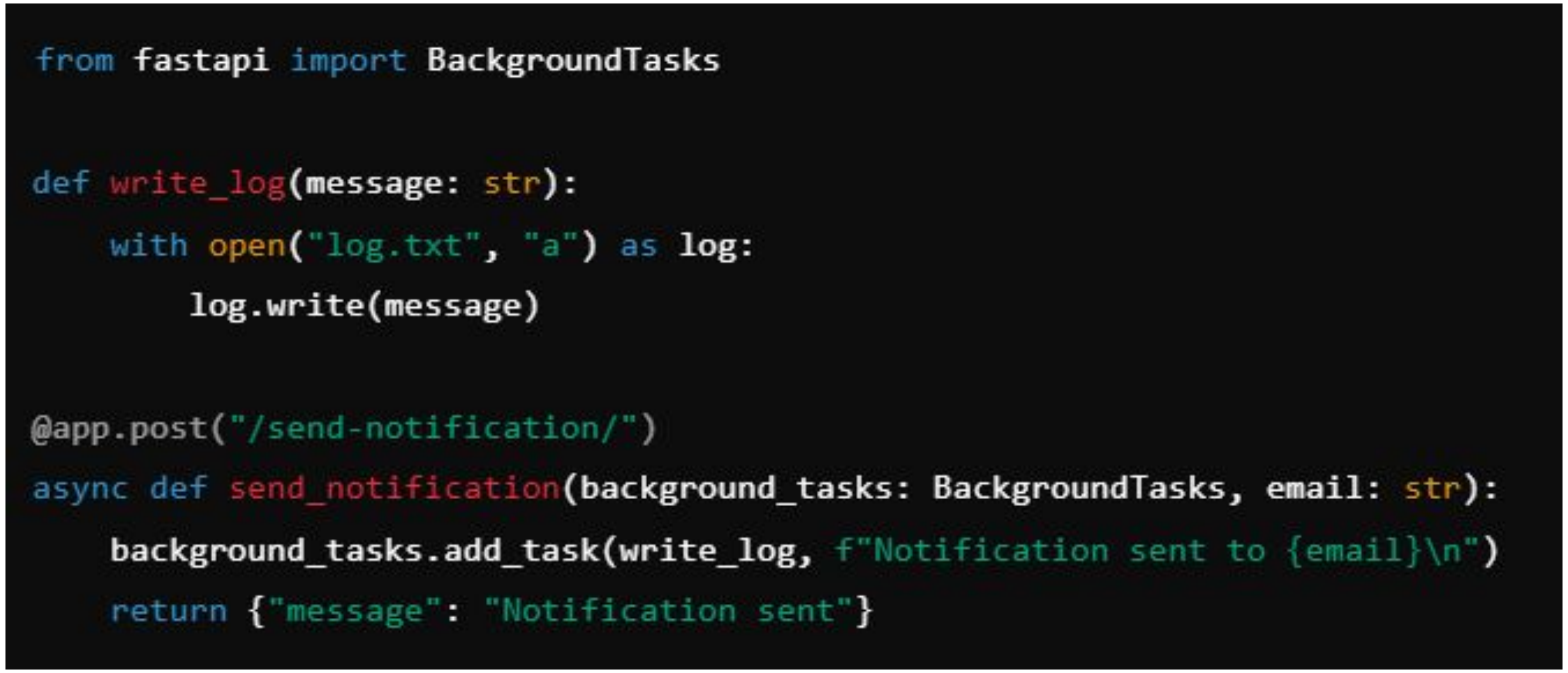
- **Explanation:** Dependencies can be reused, reducing code duplication.
- Code Example:



```python
from fastapi import Depends

def common_parameters(q: str = None):
    return {"q": q}

@app.get("/items/")
async def read_items(commons: dict = Depends(common_parameters)):
    return commons
```

Background Tasks

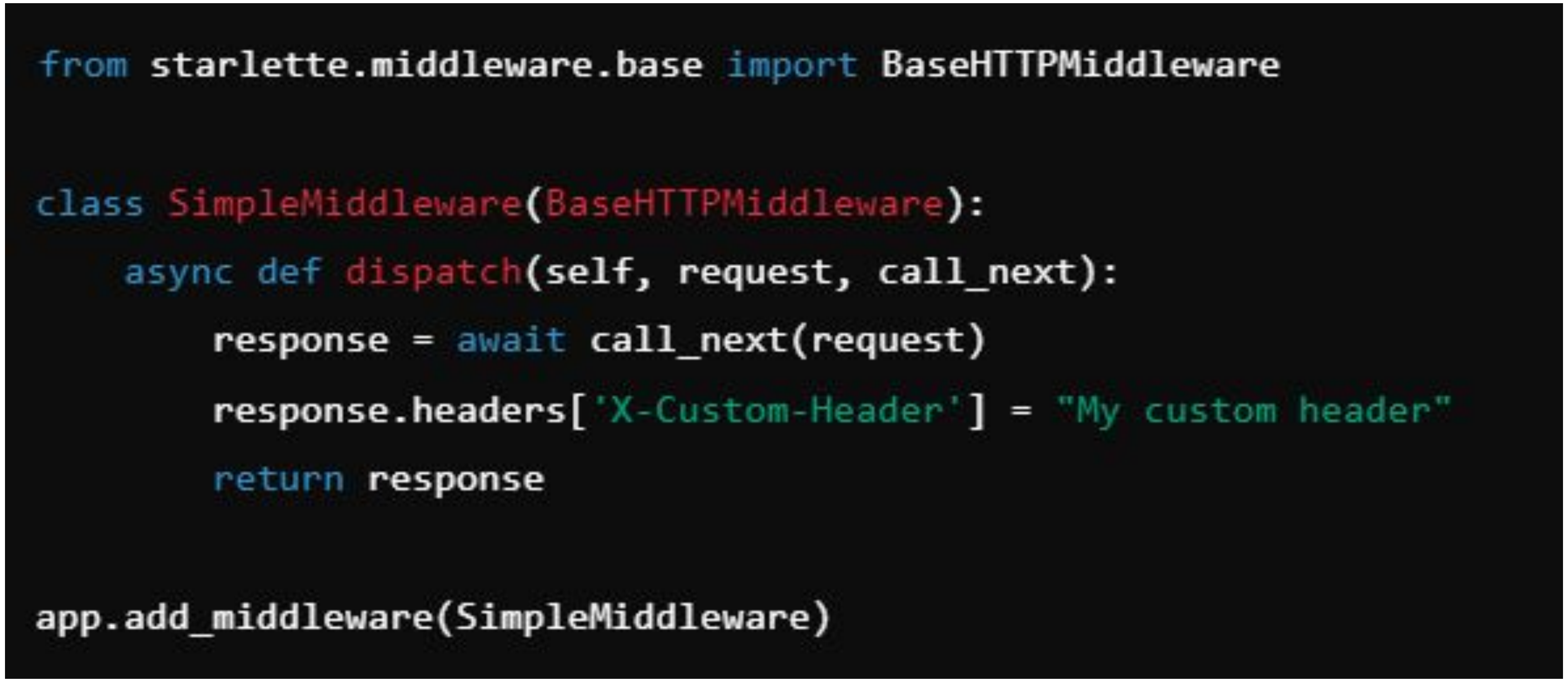
- **Explanation:** Use background tasks for tasks that don't need to be completed during the request-response cycle.



```python
from fastapi import BackgroundTasks

def write_log(message: str):
    with open("log.txt", "a") as log:
        log.write(message)

@app.post("/send-notification/")
async def send_notification(background_tasks: BackgroundTasks, email: str):
    background_tasks.add_task(write_log, f"Notification sent to {email}\n")
    return {"message": "Notification sent"}
```

Middleware

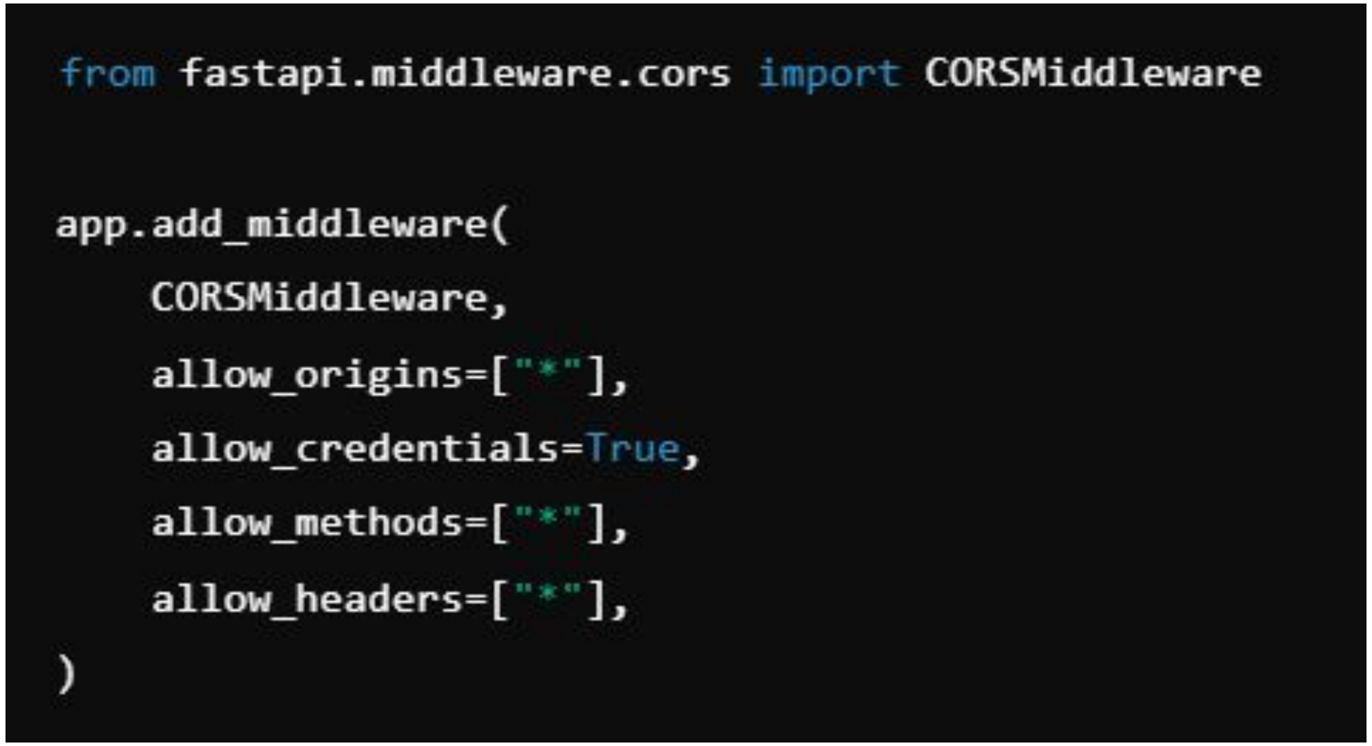
- **Explanation:** Middleware can modify requests and responses globally.
  - **Code Example:**



```
from starlette.middleware.base import BaseHTTPMiddleware

class SimpleMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request, call_next):
        response = await call_next(request)
        response.headers['X-Custom-Header'] = "My custom header"
        return response

app.add_middleware(SimpleMiddleware)
```

CORS (Cross-Origin Resource Sharing)

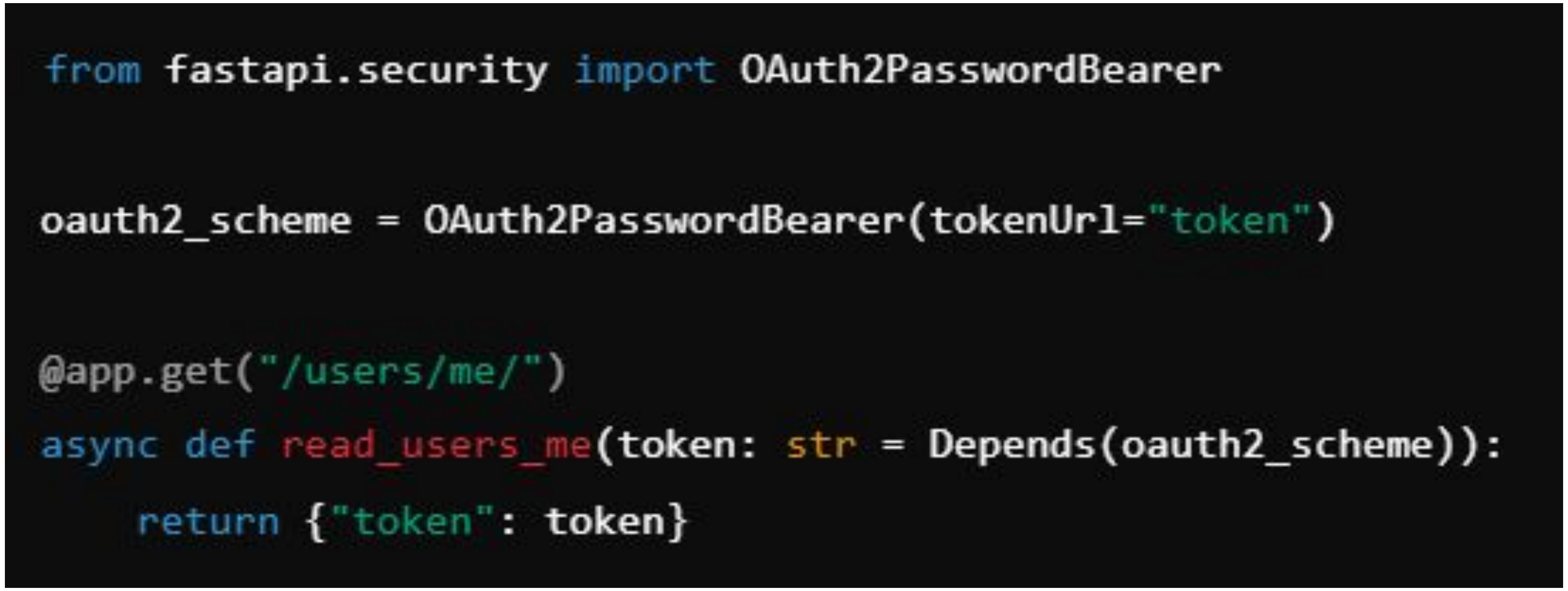
- **Explanation:** Use middleware to handle CORS issues in modern web apps.
  - **Code Example:**



```
from fastapi.middleware.cors import CORSMiddleware

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

Authentication - OAuth2 Password Flow

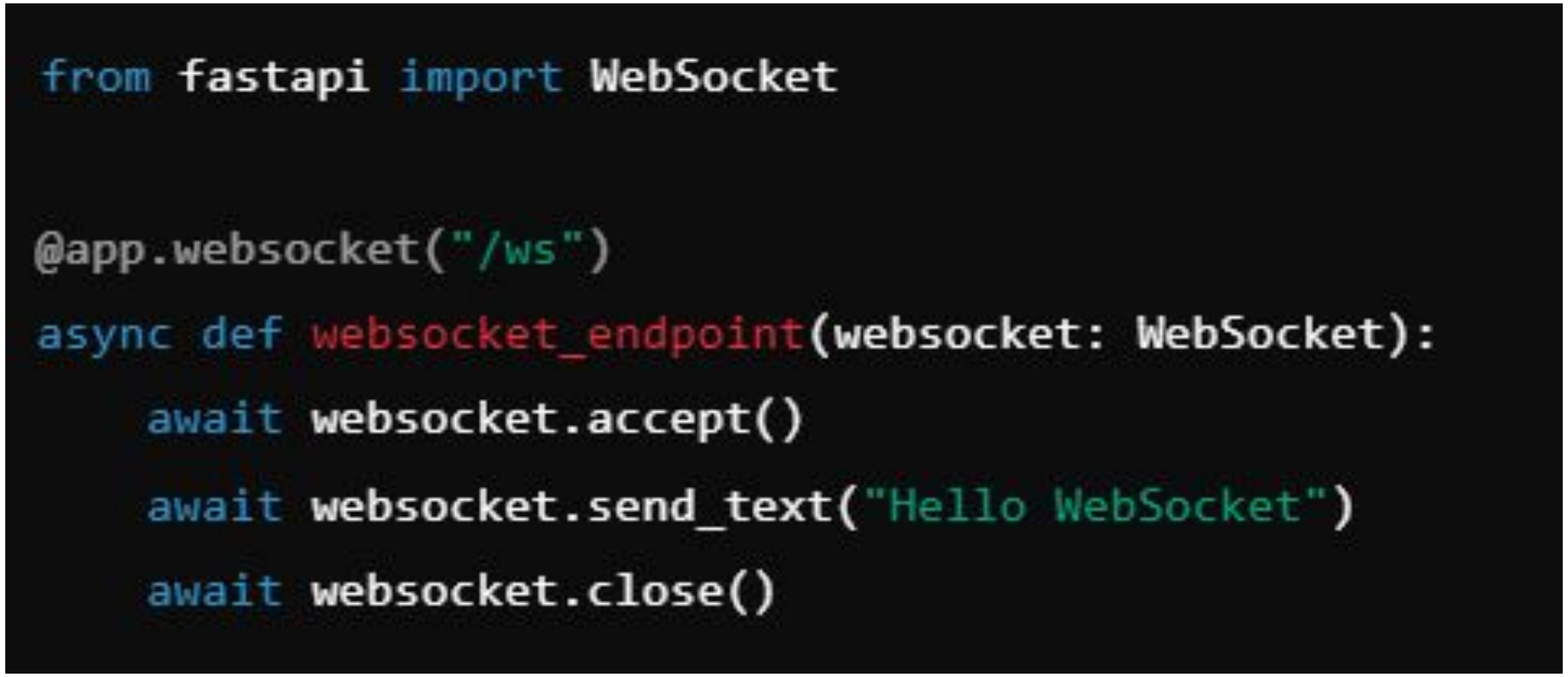
- **Explanation:** FastAPI provides OAuth2 support with password flow.
- **Code Example:**



```python
from fastapi.security import OAuth2PasswordBearer

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

@app.get("/users/me/")
async def read_users_me(token: str = Depends(oauth2_scheme)):
    return {"token": token}
```

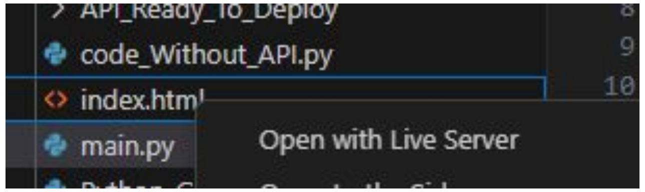
FastAPI with WebSockets

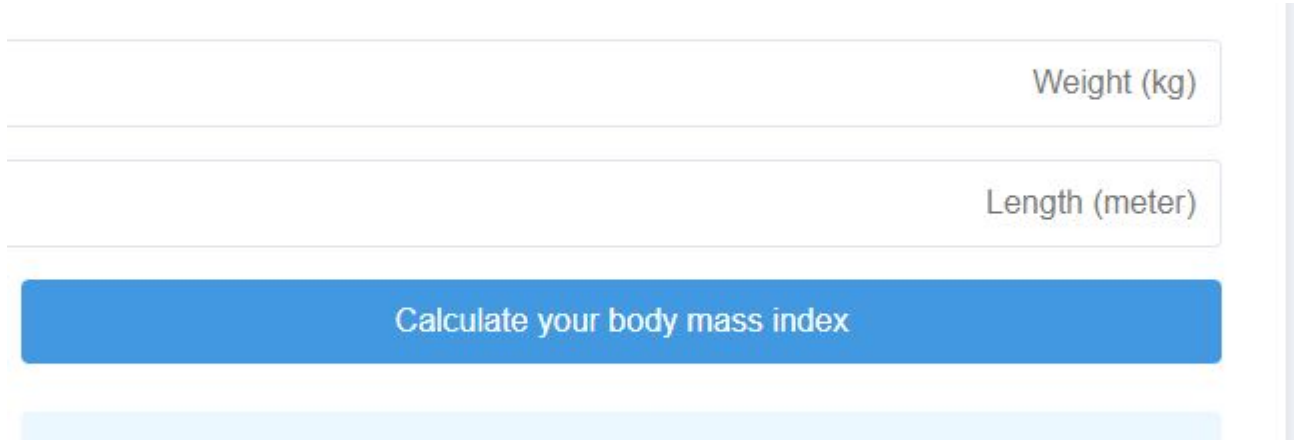
- **Code Example:**



```python
from fastapi import WebSocket


@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    await websocket.send_text("Hello WebSocket")
    await websocket.close()
```

FastAPI with HTML



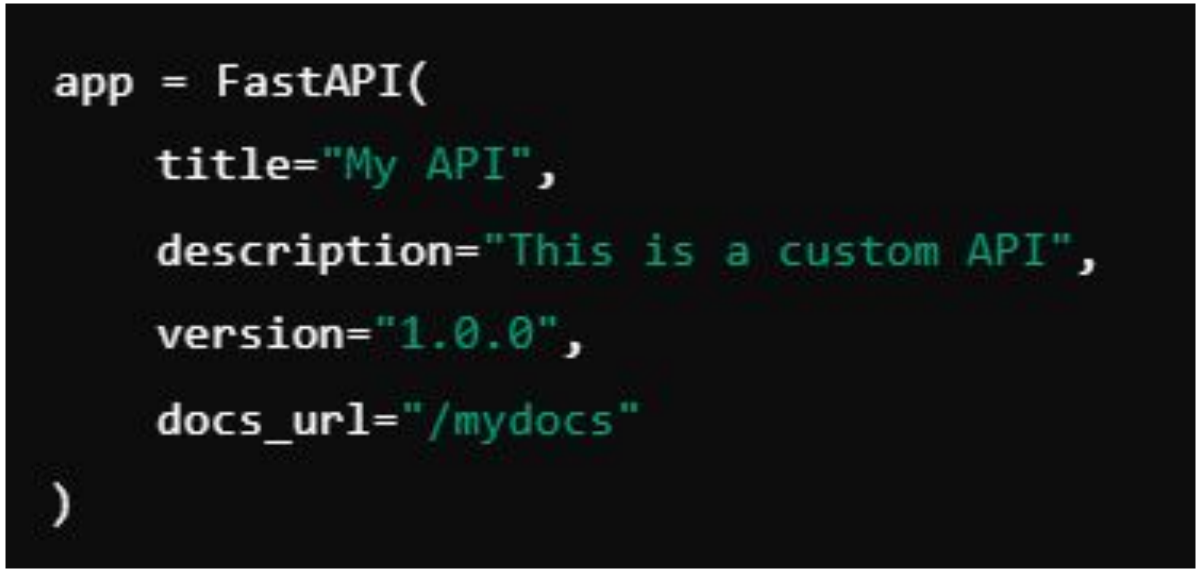


Auto-generated API Documentation

FastAPI Docs:

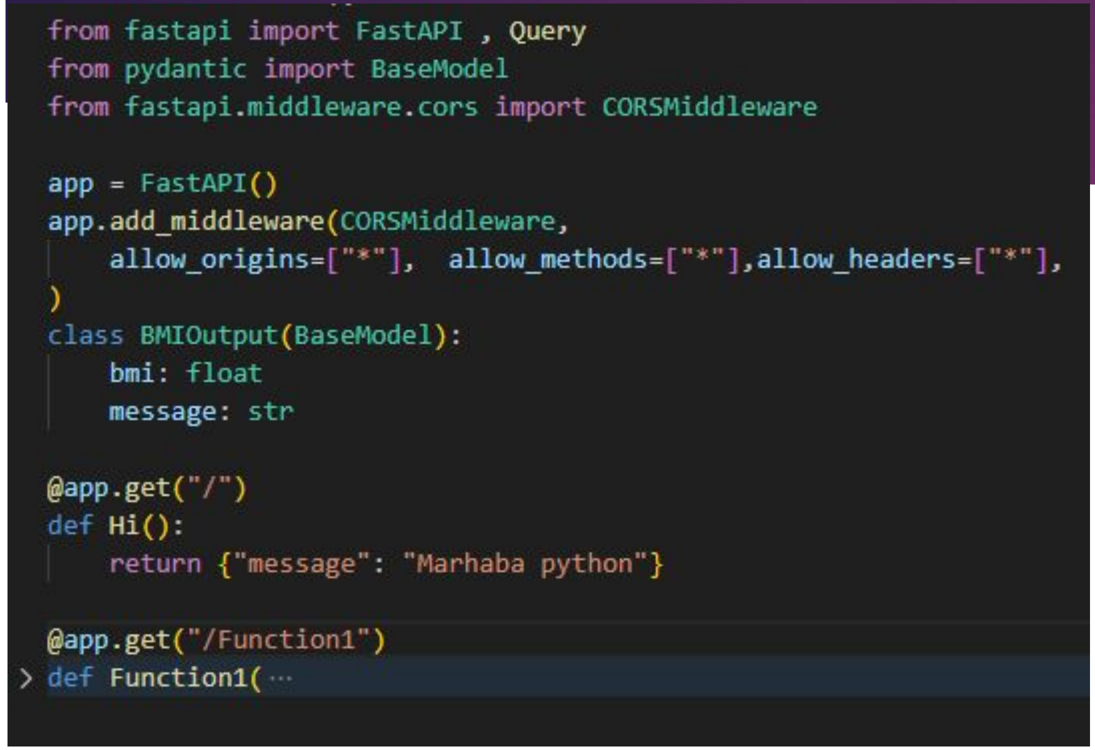
- Swagger UI: `/docs`
- ReDoc: `/redoc`

- **Customizing Docs:**

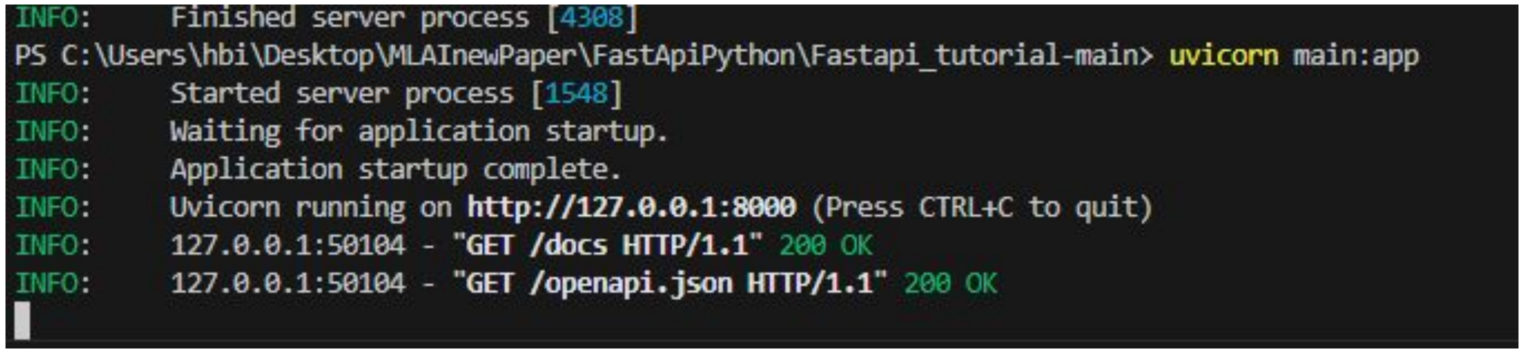


```python
app = FastAPI(
    title="My API",
    description="This is a custom API",
    version="1.0.0",
    docs_url="/mydocs"
)
```

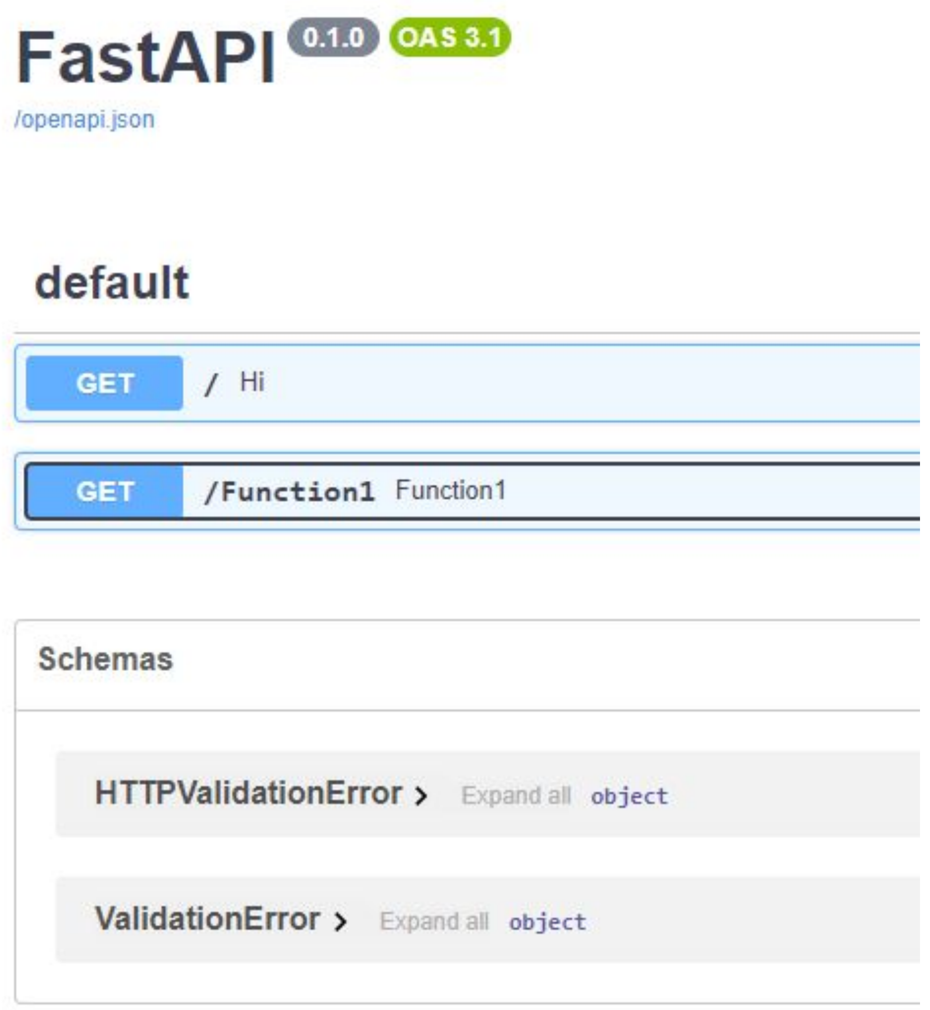
http://127.0.0.1:8000/docs#/



```
from fastapi import FastAPI , Query
from pydantic import BaseModel
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()
app.add_middleware(CORSMiddleware,
    allow_origins=["*"],  allow_methods=["*"],allow_headers=["*"],
)
class BMIOutput(BaseModel):
    bmi: float
    message: str

@app.get("/")
def Hi():
    return {"message": "Marhaba python"}

@app.get("/Function1")
def Function1(⋯
```



```
INFO:     Finished server process [4308]
PS C:\Users\hbi\Desktop\MLAInewPaper\FastApiPython\Fastapi_tutorial-main> uvicorn main:app
INFO:     Started server process [1548]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     127.0.0.1:50104 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:50104 - "GET /openapi.json HTTP/1.1" 200 OK
```



FastAPI 0.1.0 OAS 3.1

/openapi.json

default

GET / Hi

GET /Function1 Function1

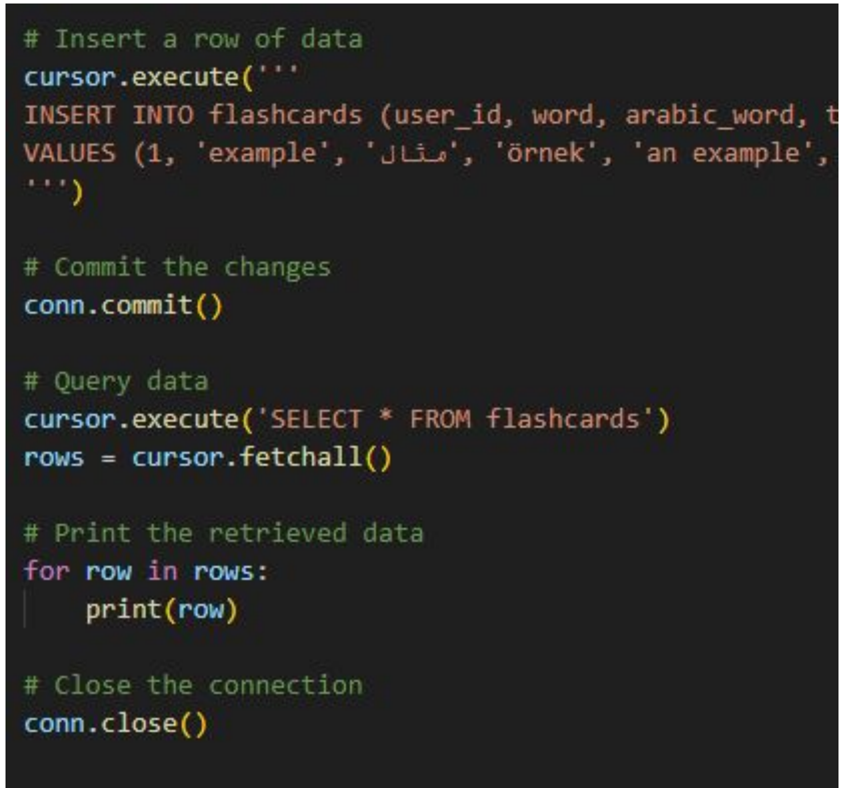
Schemas

HTTPValidationError › Expand all object

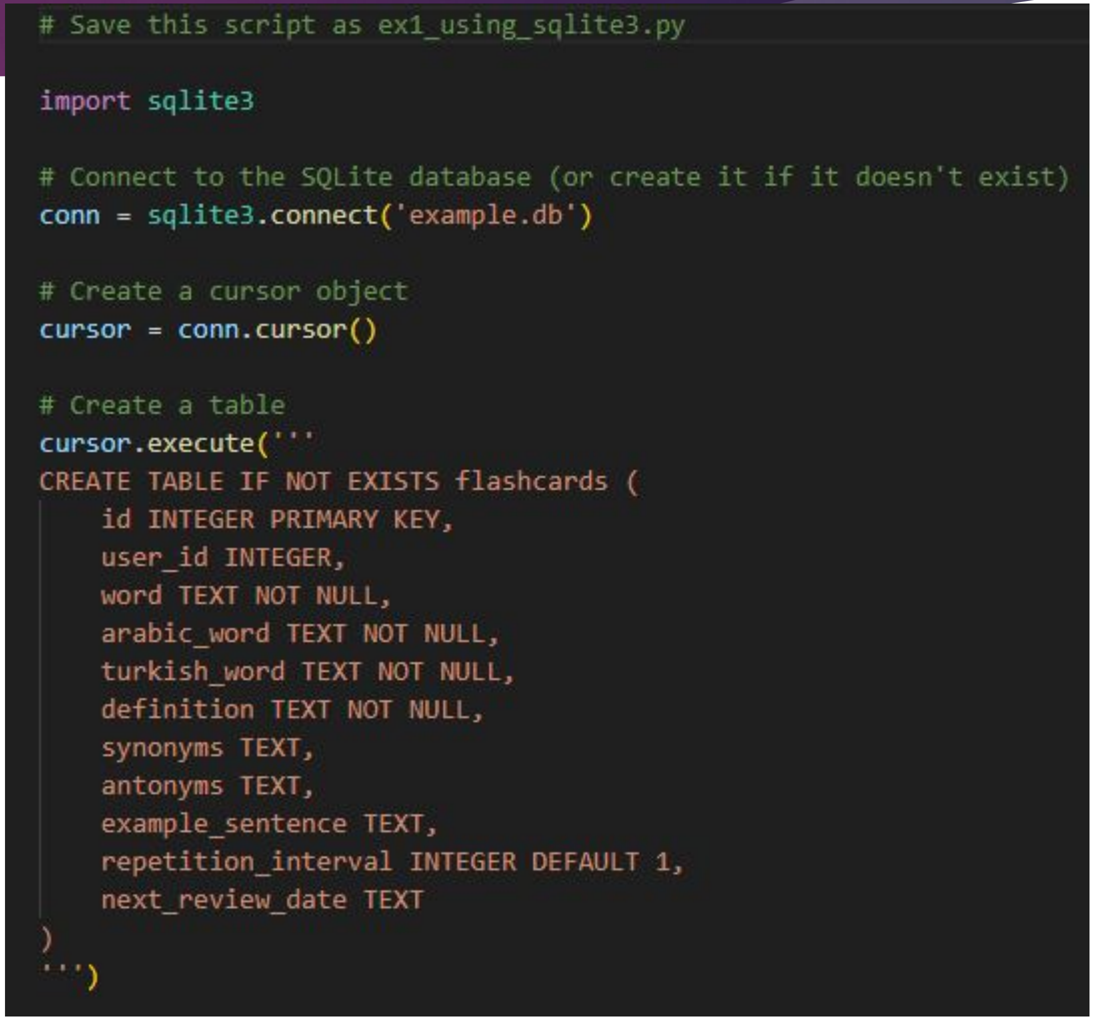
ValidationError › Expand all object

Database Integration - sqlite3

- sqlite3 is built into Python's standard library
- https://docs.python.org/3/library/sqlite3.html



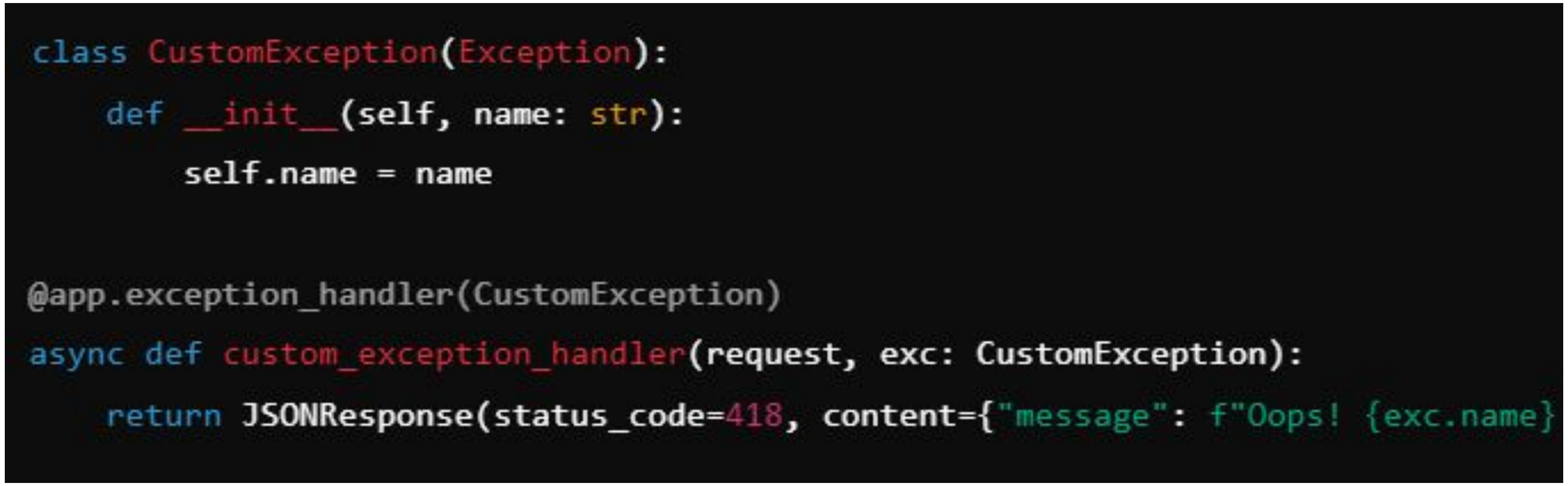
```
# Insert a row of data
cursor.execute('''
INSERT INTO flashcards (user_id, word, arabic_word, t
VALUES (1, 'example', 'مثال', 'örnek', 'an example',
''')

# Commit the changes
conn.commit()

# Query data
cursor.execute('SELECT * FROM flashcards')
rows = cursor.fetchall()

# Print the retrieved data
for row in rows:
    print(row)

# Close the connection
conn.close()
```



```
# Save this script as ex1_using_sqlite3.py

import sqlite3

# Connect to the SQLite database (or create it if it doesn't exist)
conn = sqlite3.connect('example.db')

# Create a cursor object
cursor = conn.cursor()

# Create a table
cursor.execute('''
CREATE TABLE IF NOT EXISTS flashcards (
    id INTEGER PRIMARY KEY,
    user_id INTEGER,
    word TEXT NOT NULL,
    arabic_word TEXT NOT NULL,
    turkish_word TEXT NOT NULL,
    definition TEXT NOT NULL,
    synonyms TEXT,
    antonyms TEXT,
    example_sentence TEXT,
    repetition_interval INTEGER DEFAULT 1,
    next_review_date TEXT
)
''')
```

Creating a Custom Exception

- **Explanation:** Define custom exceptions and handlers.



```python
class CustomException(Exception):
    def __init__(self, name: str):
        self.name = name

@app.exception_handler(CustomException)
async def custom_exception_handler(request, exc: CustomException):
    return JSONResponse(status_code=418, content={"message": f"Oops! {exc.name}
```

Summary

Recap of Key Features:

- High-performance API development
- Automatic validation with Pydantic
- Dependency injection
- Async support
- Auto-generated API docs

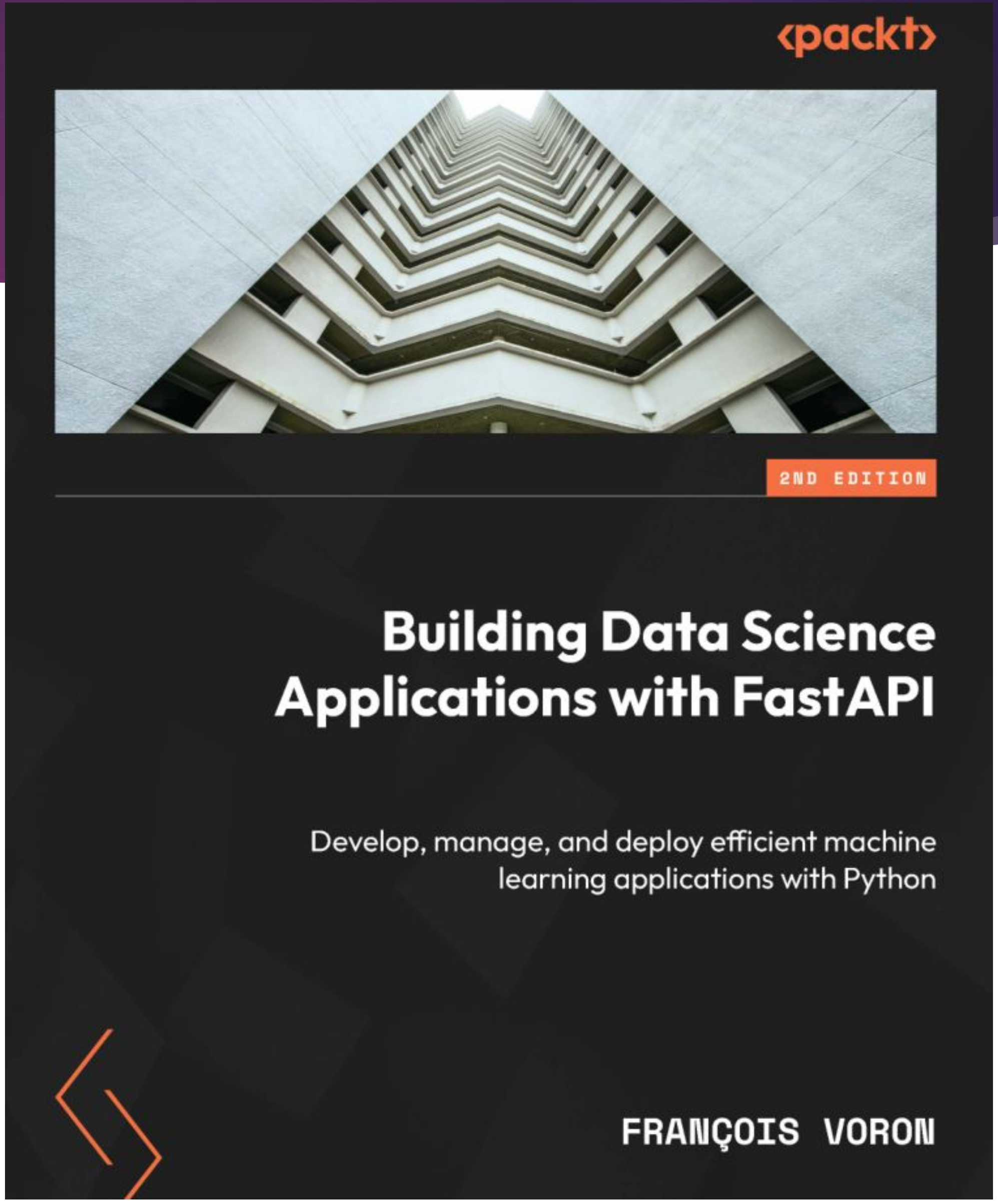
Next Steps: Experiment with FastAPI, integrate databases, and explore advanced topics like WebSockets and background tasks.

datasets

1. https://huggingface.co/datasets
2. https://www.kaggle.com/search
3. https://docs.google.com/document/d/1q8nfOEFkVM8u7i_cBaUweWf0mJFXkLzN52le5rvBKpg/edit?usp=sharing

References

https://fastapi.tiangolo.com/



<packt>

2ND EDITION

Building Data Science Applications with FastAPI

Develop, manage, and deploy efficient machine learning applications with Python

FRANÇOIS VORON