FastAPI Quick Start

FastAPI is a Python module used alongside an ASGI server to provide a blazingly fast API easily with Python.

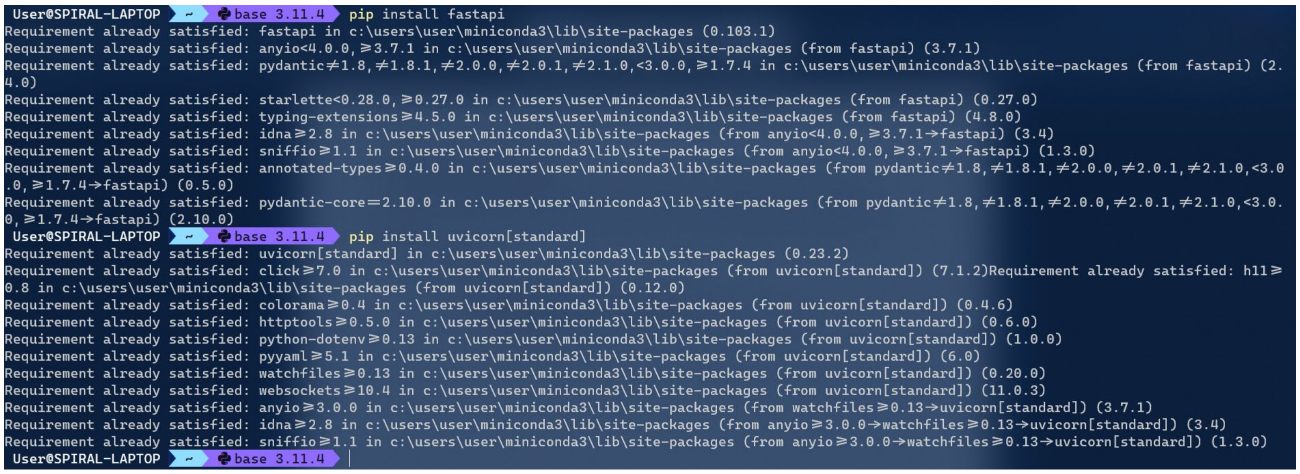
To start using FastAPI, Get the following packages:

- FastAPI using pip command:

  ```
  pip install fastapi
  ```

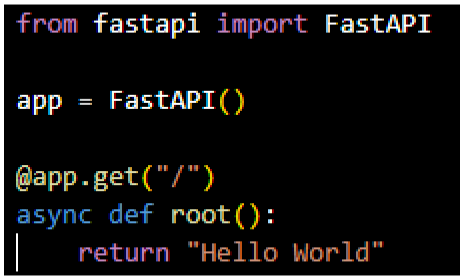
- Uvicorn ASGI server, using pip cpmmand:

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



```
 User@SPIRAL-LAPTOP  ~  base 3.11.4  pip install fastapi
Requirement already satisfied: fastapi in c:\users\user\miniconda3\lib\site-packages (0.103.1)
Requirement already satisfied: anyio<4.0.0,≥3.7.1 in c:\users\user\miniconda3\lib\site-packages (from fastapi) (3.7.1)
Requirement already satisfied: pydantic≠1.8,≠1.8.1,≠2.0.0,≠2.0.1,≠2.1.0,<3.0.0,≥1.7.4 in c:\users\user\miniconda3\lib\site-packages (from fastapi) (2.
4.0)
Requirement already satisfied: starlette<0.28.0,≥0.27.0 in c:\users\user\miniconda3\lib\site-packages (from fastapi) (0.27.0)
Requirement already satisfied: typing-extensions≥4.5.0 in c:\users\user\miniconda3\lib\site-packages (from fastapi) (4.8.0)
Requirement already satisfied: idna≥2.8 in c:\users\user\miniconda3\lib\site-packages (from anyio<4.0.0,≥3.7.1→fastapi) (3.4)
Requirement already satisfied: sniffio≥1.1 in c:\users\user\miniconda3\lib\site-packages (from anyio<4.0.0,≥3.7.1→fastapi) (1.3.0)
Requirement already satisfied: annotated-types≥0.4.0 in c:\users\user\miniconda3\lib\site-packages (from pydantic≠1.8,≠1.8.1,≠2.0.0,≠2.0.1,≠2.1.0,<3.0
.0,≥1.7.4→fastapi) (0.5.0)
Requirement already satisfied: pydantic-core═2.10.0 in c:\users\user\miniconda3\lib\site-packages (from pydantic≠1.8,≠1.8.1,≠2.0.0,≠2.0.1,≠2.1.0,<3.0.
0,≥1.7.4→fastapi) (2.10.0)
 User@SPIRAL-LAPTOP  ~  base 3.11.4  pip install uvicorn[standard]
Requirement already satisfied: uvicorn[standard] in c:\users\user\miniconda3\lib\site-packages (0.23.2)
Requirement already satisfied: click≥7.0 in c:\users\user\miniconda3\lib\site-packages (from uvicorn[standard]) (7.1.2)Requirement already satisfied: h11≥
0.8 in c:\users\user\miniconda3\lib\site-packages (from uvicorn[standard]) (0.12.0)
Requirement already satisfied: colorama≥0.4 in c:\users\user\miniconda3\lib\site-packages (from uvicorn[standard]) (0.4.6)
Requirement already satisfied: httptools≥0.5.0 in c:\users\user\miniconda3\lib\site-packages (from uvicorn[standard]) (0.6.0)
Requirement already satisfied: python-dotenv≥0.13 in c:\users\user\miniconda3\lib\site-packages (from uvicorn[standard]) (1.0.0)
Requirement already satisfied: pyyaml≥5.1 in c:\users\user\miniconda3\lib\site-packages (from uvicorn[standard]) (6.0)
Requirement already satisfied: watchfiles≥0.13 in c:\users\user\miniconda3\lib\site-packages (from uvicorn[standard]) (0.20.0)
Requirement already satisfied: websockets≥10.4 in c:\users\user\miniconda3\lib\site-packages (from uvicorn[standard]) (11.0.3)
Requirement already satisfied: anyio≥3.0.0 in c:\users\user\miniconda3\lib\site-packages (from watchfiles≥0.13→uvicorn[standard]) (3.7.1)
Requirement already satisfied: idna≥2.8 in c:\users\user\miniconda3\lib\site-packages (from anyio≥3.0.0→watchfiles≥0.13→uvicorn[standard]) (3.4)
Requirement already satisfied: sniffio≥1.1 in c:\users\user\miniconda3\lib\site-packages (from anyio≥3.0.0→watchfiles≥0.13→uvicorn[standard]) (1.3.0)
 User@SPIRAL-LAPTOP  ~  base 3.11.4 
```

To create the project, start by create a blank python file and put in FastAPI example. Such as:

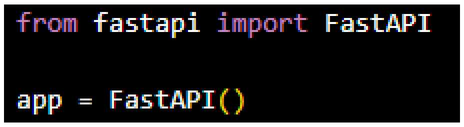


```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return "Hello World"
```

Explanation:

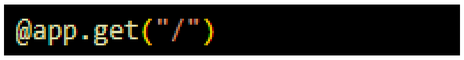


```python
from fastapi import FastAPI

app = FastAPI()
```

Import the FastAPI module and create an app object for **FastAPI service**.
In this example we'll do a simple Hello World GET API.

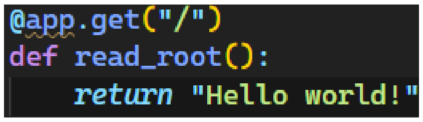
By writing the decorator on what **FastAPI service** this method should be in, and where the route is.



```
@app.get("/")
```

@app.get means this function is for the GET method. "/" means the route is in the root directory.

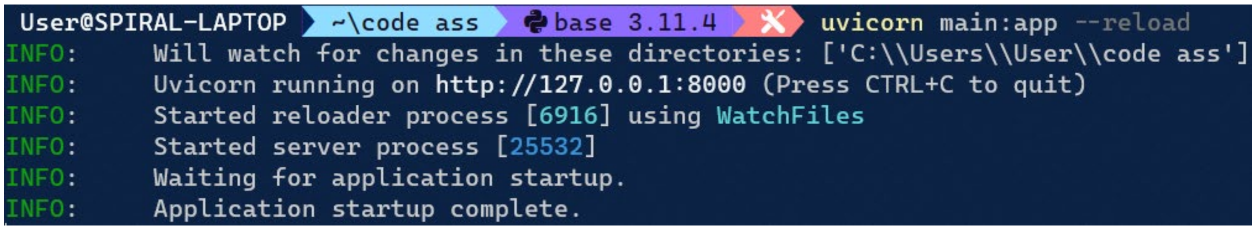
Then, define a function. Make it return a simple hello world.



```
@app.get("/")
def read_root():
    return "Hello world!"
```

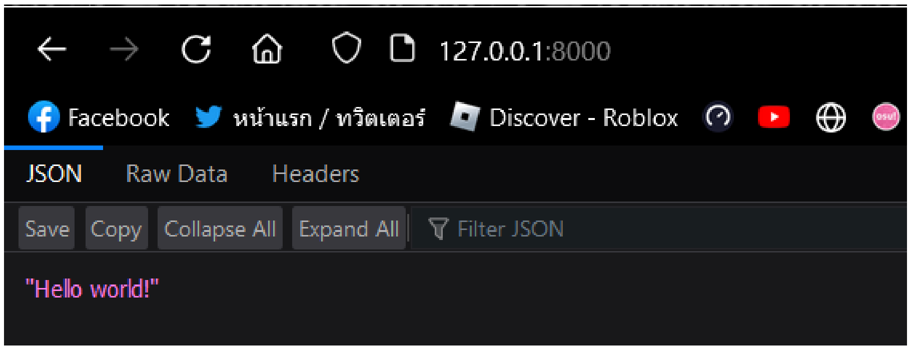
To start the server, save the file and run it on the terminal or command line using:

```
uvicorn [filename without the .py]:[the FastAPI server you
named it] --reload
```



```
User@SPIRAL-LAPTOP  ~\code ass  base 3.11.4    uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\User\\code ass']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [6916] using WatchFiles
INFO:     Started server process [25532]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

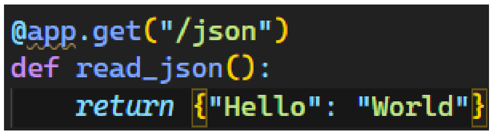
To test the result, open your web browser and enter the server's address (in this case, localhost) You'll find the hello world returned as JSON.



```
"Hello world!"
```

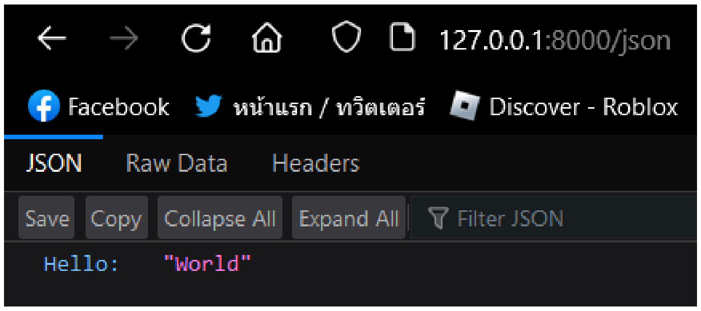
Note that it will return as text/javascript, so you can return a dictionary.

Now, try to make it return a JSON.



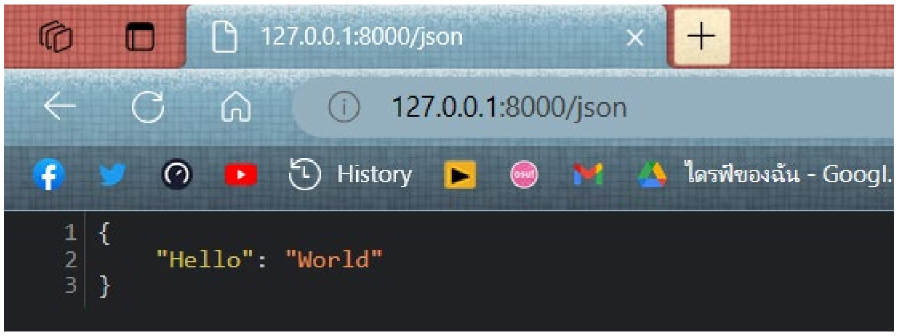
```
@app.get("/json")
def read_json():
    return {"Hello": "World"}
```

(We started uvicorn with the –reload flag, so we could just save it, then it'll automatically update the API.)



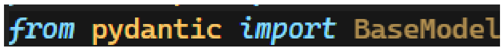
```
Hello:    "World"
```

(Mozilla Firefox automatically uses the JSON explorer if it detects that the response is JSON. If you're using Chromium-based, you'll just find a JSON as plaintext instead, like this:)



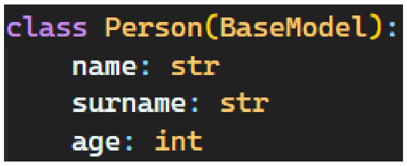
```
{
    "Hello": "World"
}
```

To use the CRUD operation in FastAPI service, we need to import **Pydantic** for type checking.



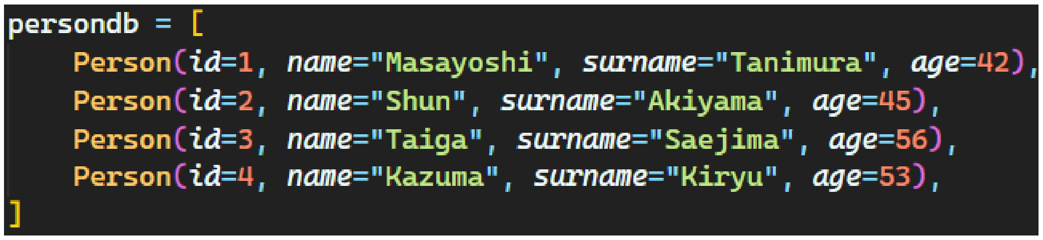
```python
from pydantic import BaseModel
```

Create a class that is inherited from BaseModel class



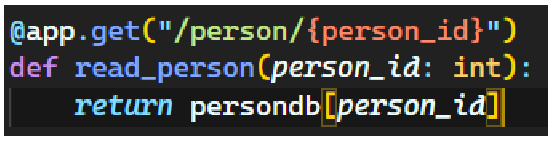
```python
class Person(BaseModel):
    name: str
    surname: str
    age: int
```

For this example, we'll create a simple list to store people.



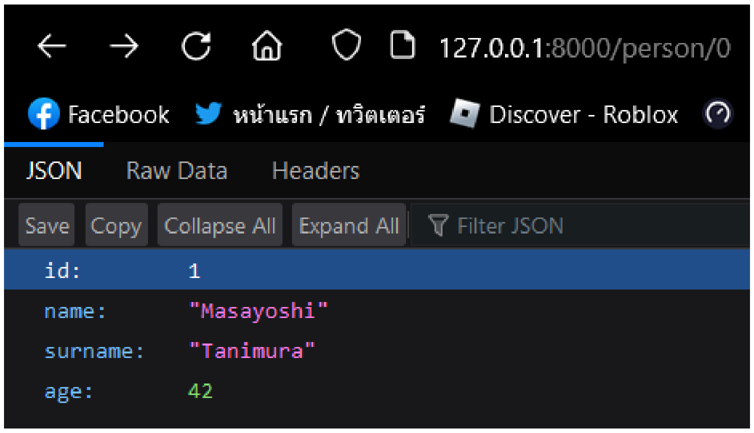
```python
persondb = [
    Person(id=1, name="Masayoshi", surname="Tanimura", age=42),
    Person(id=2, name="Shun", surname="Akiyama", age=45),
    Person(id=3, name="Taiga", surname="Saejima", age=56),
    Person(id=4, name="Kazuma", surname="Kiryu", age=53),
]
```

Then, create a **GET** method that accepts a dynamic parameter that returns the person.



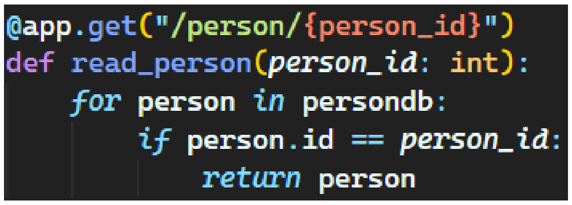
```python
@app.get("/person/{person_id}")
def read_person(person_id: int):
    return persondb[person_id]
```

Now you can access the person's data by the ID.



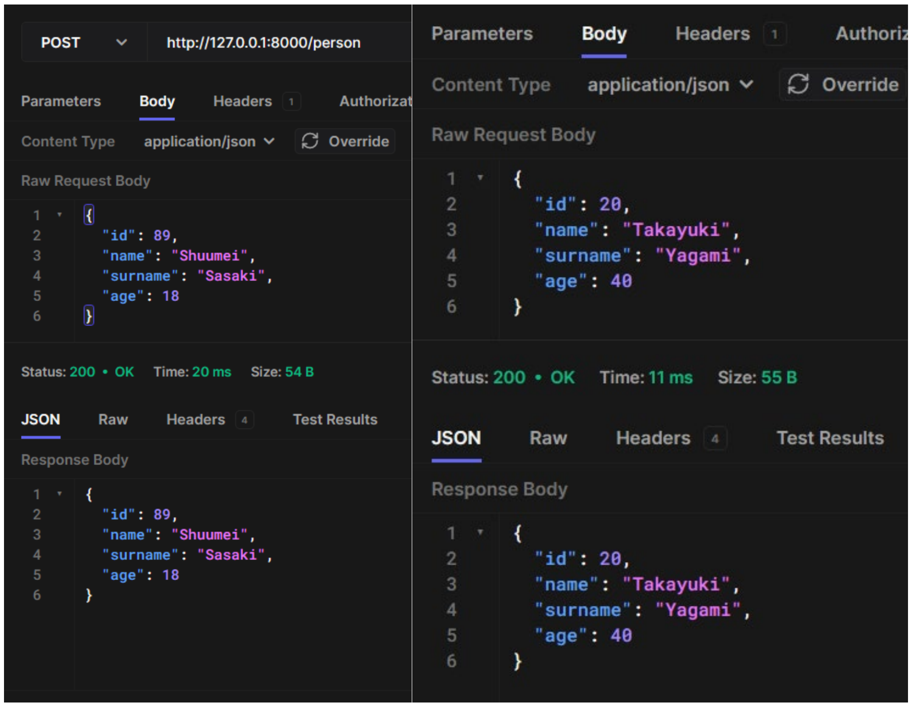
```
id:        1
name:      "Masayoshi"
surname:   "Tanimura"
age:       42
```

To add a person's data's into the "database", we need to create a POST method such as:



```python
@app.get("/person/{person_id}")
def read_person(person_id: int):
    for person in persondb:
        if person.id == person_id:
            return person
```

Note: The Person class can be inputted as a JSON object, or as query parameters.
Now, send a POST request the same way as you did in the previous lab.
In this example, we use **Hoppscotch** as an API tester. The process is the exact same with Insomnia and Postman.



```
{
    "id": 89,
    "name": "Shuumei",
    "surname": "Sasaki",
    "age": 18
}
```

Status: 200 • OK   Time: 20 ms   Size: 54 B

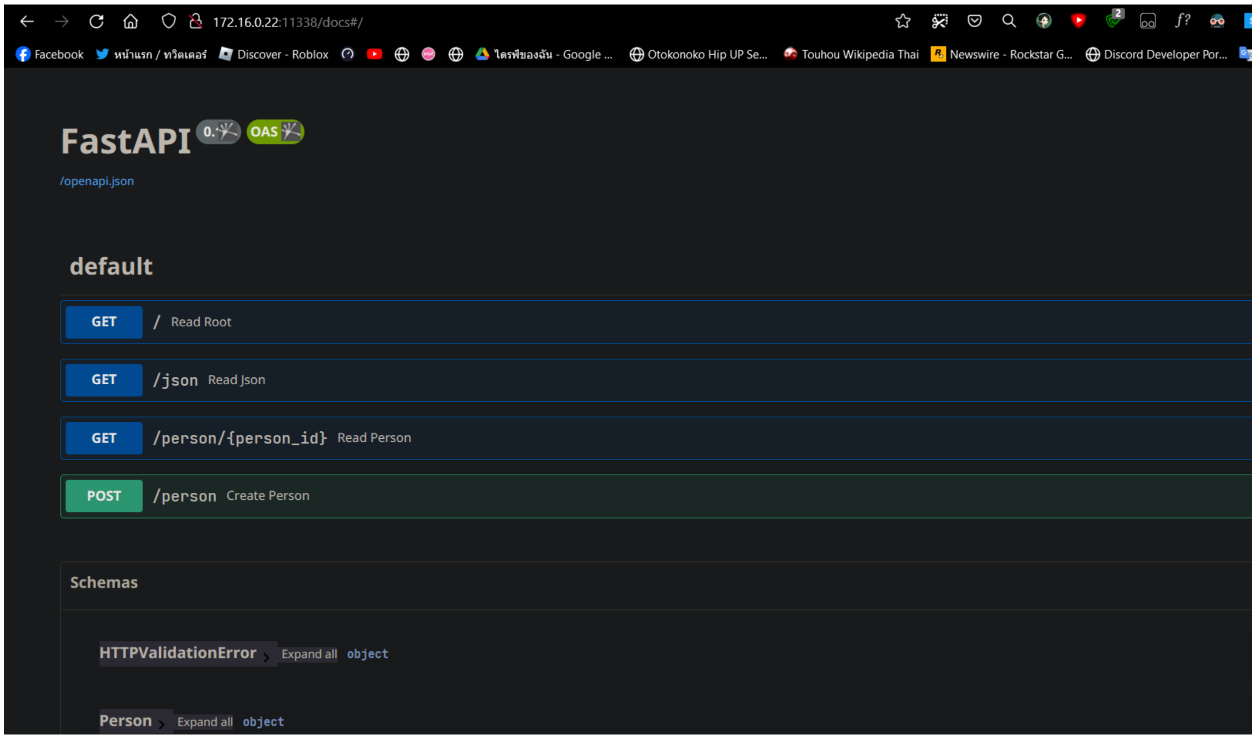
```
{
    "id": 89,
    "name": "Shuumei",
    "surname": "Sasaki",
    "age": 18
}
```

```
{
    "id": 20,
    "name": "Takayuki",
    "surname": "Yagami",
    "age": 40
}
```

Status: 200 • OK   Time: 11 ms   Size: 55 B

```
{
    "id": 20,
    "name": "Takayuki",
    "surname": "Yagami",
    "age": 40
}
```

The automatically generated Swagger UI API Documentation can be accessed by the /docs endpoint.


FastAPI 0.1.0 OAS 3.1

/openapi.json

default

GET / Read Root

GET /json Read Json

GET /person/{person_id} Read Person

POST /person Create Person

Schemas

HTTPValidationError > Expand all object

Person > Expand all object

For more information that can be conveyed more than this piece of paper, refer to https://fastapi.tiangolo.com/.