Getting Started

In this sub-repository we are going to learn the basics of implementing api's using the fast `FastAPI`. First things first we need to create a virtual environment, activate it and install fastapi.

```
virtualenv venv

# then
.\venv\Scripts\activate

# then
pip install fastapi[standard]
```

Basic Hello World

In the first example we are going to create a basic hello world api using the fastapi. First we are going to create a file called `app.py` and add the following code to it.

```
from fastapi import FastAPI
from typing import Union

app = FastAPI()


@app.get("/")
def hello():
    return {"message": "Hello world!"}


@app.get("/items/{item_id}")
def read_item(item_id: int, name: Union[str, None] = None):
    return {"item_id": item_id, "name": name}
```

To start the server we will open the terminal and run the following command.

```
fastapi dev app.py
```

If we visit `http://127.0.0.1:8000` with a `GET` method we will get the following json response:

```
{
  "message": "Hello world!"
}
```

If we visit `http://127.0.0.1:8000/items/1?name=brits` with a `GET` method we will get the following json response

```
{
  "item_id": 1,
  "name": "brits"
}
```

We can also use the `async` and `await`. Let's say we want to send a `put` request to the server so that we can update our items. To make this typed we are going to create a type `TItem` which will look as follows:

```
from pydantic import BaseModel

class TItem(BaseModel):
    name: str
    price: float
    id: int
```

Now we are going to update our `app.py` to:

```
from fastapi import FastAPI
from typing import Union
from pydantic import BaseModel


class TItem(BaseModel):
    name: str
    price: float
    id: int


app = FastAPI()

@app.put("/item/{id}")
async def update_item(id: int, item: TItem):
    return  {"id": id, "name": item.name, "price": item.price}
```

If we send a `PUT` request to `http://127.0.0.1:8000/item/1` with the following `json` body:

```
{
  "id": 1,
  "name": "Dog",
  "price": 12.7
}
```

We will get the following json response:

```
{
  "id": 1,
  "name": "Dog",
  "price": 12.7
}
```

Interactive Docs

To use the interactive docs in fastapi you can visit this url: http://127.0.0.1:8000/docs

API Router

We can create different routers in fast api by using the `APIRouter` class. Let's create a new package called `routers` and inside that package we are going to create another package called `users` and add the following code to it:

```
from fastapi import APIRouter
userRouter = APIRouter(
    prefix="/api/v1/users",

)

people = [
    {'name': "Jonh", "gender": "M", "age": 16},
    {'name': "Peter", "gender": "M", "age": 16},
]

@userRouter.get('/')
def users():
    return people
```

Now in the `app.py` we are then going to add the following:

```
from routers.user import userRouter

class TItem(BaseModel):
    name: str
    price: float
    id: int

app = FastAPI()
app.include_router(userRouter)

# .....
```

Now if we visit GET: `http://127.0.0.1:8000/api/v1/users/` we will get the following response.

```
[
  {
    "name": "Jonh",
    "gender": "M",
    "age": 16
  },
  {
    "name": "Peter",
    "gender": "M",
    "age": 16
  }
]
```

FastAPI Class

Ypu can change the default parameter values of the `FastAPI` class. Here is an example showing how that can be done:

```
app = FastAPI(
    title="My API",
    description="This is a simple api",
    version="0.0.1",
)
```

Custom Response Class

With FastAPI you can return specific response. There are different responses that can be found from the `fastapi.responses` the code below shows some examples:

```
from fastapi.responses import PlainTextResponse, HTMLResponse, JSONResponse


class TItem(BaseModel):
    name: str
    price: float
    id: int


app = FastAPI(
    title="My API",
    description="This is a simple api",
    version="0.0.1",
)
```

```
@app.get("/")
def plain():
    return PlainTextResponse("Hello there", status_code=200)


@app.get("/html")
def html():
    return HTMLResponse("<h1>Hello there</h1>", status_code=200)


@app.get("/json")
def json():
    return JSONResponse({"name": "Hello", "age": 12}, status_code=200)
```

Request Parameters

FastAPI comes with some special functions that can be used to get data from the `Annotated` requests which are:

```
from fastapi import Body, Cookie, File, Form, Header, Path, Query
```

1. `Query()`

```
@app.get('/')
def get_query(
q: Annotated[str, Query()]
):
return {'q': q}
```

2. `Path()`

```
@app.get('/{name}/{id}')
def get_path(
    id: Annotated[int, Path()],
    name:  Annotated[str, Path()]
):
    return {'name': name, 'id': id}
```

3. `Body()`

You can get the request body from the requests using the `Body()` method. Below is an example that demonstrates that:

```
@app.post('/login')
async def login(
   username: Annotated[str, Body()],
   password: Annotated[str, Body()],
):
   return {
       'username' : username,
       'password' : password,
   }
```

4. `Cookie()`

You can get request cookies as follows:

```
@app.get("/")
async def read_cookie(cookie: Annotated[str | None, Cookie()] = None):
    return {"cookie": cookie}
```

5. `Form()`

   To use form data we need to install `python-multipart` by running the following command:

   ```
   pip install python-multipart
   ```

```
@app.post('/login')
async def login(
   username: Annotated[str, Form()],
   password: Annotated[str, Form()],
):
   return {
       'username' : username,
       'password' : password,
   }
```

Then you can send the data as form data from the client.

7. `Header()`

We can get the headers from the requests as follows:

```
@app.get("/")
async def read_header(authorization: Annotated[str | None, Header()] = None):
    print(authorization)
    return {"authorization": authorization}
```

Dependencies

Let's say we have an route that depends on certain query parameters. We can use the `Depends()` special function that takes a callable to achieve this.

```
from fastapi import FastAPI, Depends
from typing import Union, Annotated
from pydantic import BaseModel
# ....

async def common_parameters(
    q: str | None, limit: int = 0, offset: int = 0, skip: int = 0
):
    return {"q": q, "limit": limit, "offset": offset, "skip": skip}


@app.get("/todo/")
def todos(commons: Annotated[dict, Depends(common_parameters)]):
    return commons
```

If we visit `http://127.0.0.1:8000/todo/?q=name` we will get the following json response:

```
{
  "q": "name",
  "limit": 0,
  "offset": 0,
  "skip": 0
}
```

Exceptions

Let's say we have a list of users. And we want to find the user by id. If the user is not found we want to send a response as an exception. We can do that as follows:

```
# ....
users = [
    {'name': "Jonh", "gender": "M", "age": 16, 'id': 1},
    {'name': "Peter", "gender": "M", "age": 16, 'id': 2},
]
```

```
app = FastAPI(
    title="My API",
    description="This is a simple api",
    version="0.0.1",
)


@app.get("/user/{id}")
def user(id: int):
    me = list(filter(lambda x: x['id'] == id, users))
    if len(me) == 0:
        return HTTPException(
            status_code=404,
            detail= f"The user with '{id}' was not found"
        )
    return JSONResponse({
        **me[0]
    }, status_code=200)
```

CORS

Cross origin refers to two platforms that want to communicate with each other from different origins. Bellow is an example of how CORS can be configured on fastapi.

```
from fastapi import FastAPI, Path
from typing import Annotated
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI(
    title="My API",
    description="This is a simple api",
    version="0.0.1",
)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:8080", "*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)


@app.get("/{name}/{id}")
def get_path(id: Annotated[int, Path()], name: Annotated[str, Path()]):
    return {"name": name, "id": id}
```

Status Codes

We can have access to the status codes from the `status` package from `fastapi`.

```
from fastapi import FastAPI,  status


app = FastAPI(
    title="My API",
    description="This is a simple api",
    version="0.0.1",
)

@app.get("/", status_code=status.HTTP_418_IM_A_TEAPOT)
def read_items():
    return [{"name": "Plumbus"}, {"name": "Portal Gun"}]
```

File Upload

Let's implement a route path that allows us to upload files. Here is the code implementation for that.

```
from fastapi import FastAPI,  UploadFile, File
from typing import Annotated

app = FastAPI(
    title="My API",
    description="This is a simple api",
    version="0.0.1",
)

@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):
    with open(file.filename, "wb+") as file_object:
        file_object.write(file.file.read())
    return {"filename": file.filename}
```

If you navigate to the image `me.jpeg` and run the following `cURL` command you will be able to upload a file to the server and it will be saved in teh root directory of the project:

```
cURL -X POST -F file=@me.jpeg http://127.0.0.1:8000/uploadfile/
```

Static Files

Let's say we want to serve static files. For that we are going to need to mount an external application. Let's say we have an image named `me.jpeg` that is located in the `static/images` folder. We can serve that from this url: http://127.0.0.1:8000/storage/images/me.jpeg. Here is an example of how the static files can be served:

```
from fastapi import FastAPI
from fastapi.staticfiles import StaticFiles

app = FastAPI(
    title="My API",
    description="This is a simple api",
    version="0.0.1",
)

app.mount("/storage", StaticFiles(directory="static"), name="storage")
```

Test Client

We can test our api easily with fastapi by using the `TestClient`. First you will need to install `pytest` as follows:

```
pip install pytest
```

Then in the `app.py` we are going to have the following code:

```
from fastapi import FastAPI, Path
from typing import Annotated
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI(
    title="My API",
    description="This is a simple api",
    version="0.0.1",
)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:8080", "*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)


@app.get('/')
def hi():
    return {'message': 'hi'}
```

```
@app.get('/bye')
def bye():
    return {'message': 'bye'}

@app.get('/hello/{name}')
def hello(
    name: Annotated[str, Path()] | None = None
):
    if name is None:
        return {"message": "Hello World"}
    return {'message': f'Hello {name}.'}
```

In the `test_app.py` file we are going to have the following code:

```
from fastapi.testclient import TestClient
from app import app

client = TestClient(app)


class TestApp:
    def test_hello(self):
        res = client.get("/hello/jon")
        assert res.json() == {"message": "Hello jon."}

    def test_hi(self):
        res = client.get("/")
        assert res.json() == {"message": "hi"}

    def test_bye(self):
        res = client.get("/bye")
        assert res.json() == {"message": "bye"}
```

To run the tests you will run the following command:

```
pytest
```

Refs

1. FastAPI