FastAPI with SQLAlchemy, PostgreSQL, Alembic and Docker [Part-2] (asynchronous version)

Ahmed Nafies · Follow

4 min read · Jul 21, 2020

100 2

An imitation of a database using boxes

Intro

The purpose of this article is to create a simple guide on how to use FastAPI with relational databases asynchronously and use Alembic for migrations. Before you go ahead with this tutorial please check **part-1.**

Here is the full working code on github

Update! SQLAlchemy ORM now has support for async, check this tutorial.

Lets Start

Install the required package `databases`

`databases` is a lightweight package with asyncio support for many relational databases and uses the `sqlalchemy` core queries.

for the purpose of this tutorial I will be using **pipenv**, but you can use **pip** or **poetry** or **conda** or whatever package manager you like.

```
pipenv install databases
pipenv install databases[postgresql]
pipenv install asyncpg
```

we will use the same `docker` config as before

Dockerfile

```
# Pull base image
FROM python:3.7

# Set environment varibles
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

WORKDIR /code/

# Install dependencies
RUN pip install pipenv
COPY Pipfile Pipfile.lock /code/
```

```
RUN pipenv install --system --dev

COPY . /code/

EXPOSE 8000
```

docker-compose.yml

```
version: "3"

services:
  db:
    image: postgres:11
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=test_db
  web:
    build: .
    command: bash -c "uvicorn main:app --host 0.0.0.0 --port 8000 --
reload"
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    depends_on:
      - db

  pgadmin:
    container_name: pgadmin
    image: dpage/pgadmin4
    environment:
      - PGADMIN_DEFAULT_EMAIL=pgadmin4@pgadmin.org
      - PGADMIN_DEFAULT_PASSWORD=admin
    ports:
      - "5050:80"
    depends_on:
      - db
```

we will keep ***schema.py*** the way it is

```
from pydantic import BaseModel


class User(BaseModel):
    first_name: str
    last_name: str
    age: int

    class Config:
        orm_mode = True
```

we will keep *alembic.ini* the same as well.

we will change ***.env*** to

```
DATABASE_URL=postgresql://postgres:postgres@db:5432/postgres
```

add ***db.py*** where we will initialise our database

```
import os
from databases import Database
from dotenv import load_dotenv
import sqlalchemy

BASE_DIR = os.path.dirname(os.path.abspath(__file__))
load_dotenv(os.path.join(BASE_DIR, ".env"))

db = Database(os.environ["DATABASE_URL"])

metadata = sqlalchemy.MetaData()
```

now we add ***app.py,*** we will handle app initialisation with database connection and termination

```
from db import db
from fastapi import FastAPI


app = FastAPI(title="Async FastAPI")


@app.on_event("startup")
async def startup():
    await db.connect()
```

```
@app.on_event("shutdown")
async def shutdown():
    await db.disconnect()
```

change ***models.py*** accordingly

```
from db import db

users = sqlalchemy.Table(
    "users",
    metadata,
    sqlalchemy.Column("id", sqlalchemy.Integer, primary_key=True),
    sqlalchemy.Column("first_name", sqlalchemy.String),
    sqlalchemy.Column("last_name", sqlalchemy.String),
    sqlalchemy.Column("age", sqlalchemy.Integer),
)
```

lets enhance our `models` but creating a simple model manager class `User`

```
import sqlalchemy
from db import db, metadata, sqlalchemy


users = sqlalchemy.Table(
    "users",
    metadata,
    sqlalchemy.Column("id", sqlalchemy.Integer, primary_key=True),
    sqlalchemy.Column("first_name", sqlalchemy.String),
    sqlalchemy.Column("last_name", sqlalchemy.String),
    sqlalchemy.Column("age", sqlalchemy.Integer),
)


class User:
    @classmethod
    async def get(cls, id):
        query = users.select().where(users.c.id == id)
        user = await db.fetch_one(query)
        return user

    @classmethod
    async def create(cls, **user):
        query = users.insert().values(**user)
        user_id = await db.execute(query)
        return user_id
```

The manager class will provide a simpler implementation for `get` and `create`

lets now change ***main.py*** accordingly

```
import uvicorn
from models import User as ModelUser
from schema import User as SchemaUser
from app import app
from db import db


@app.post("/user/")
async def create_user(user: SchemaUser):
    user_id = await ModelUser.create(**user.dict())
    return {"user_id": user_id}


@app.get("/user/{id}", response_model=SchemaUser)
async def get_user(id: int):
    user = await ModelUser.get(id)
    return SchemaUser(**user).dict()


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

You should now notice that we are using `async/await` with the database

It is time to modify our `Alembic` config.

change

```
import models

target_metadata = models.Base.metadata
```

to

```
import models
from db import metadata
```

```
target_metadata = metadata
```

Hint: It is important to import models before metadata.

build

```
docker-compose build
```

make migrations

```
docker-compose run web alembic revision --autogenerate
```

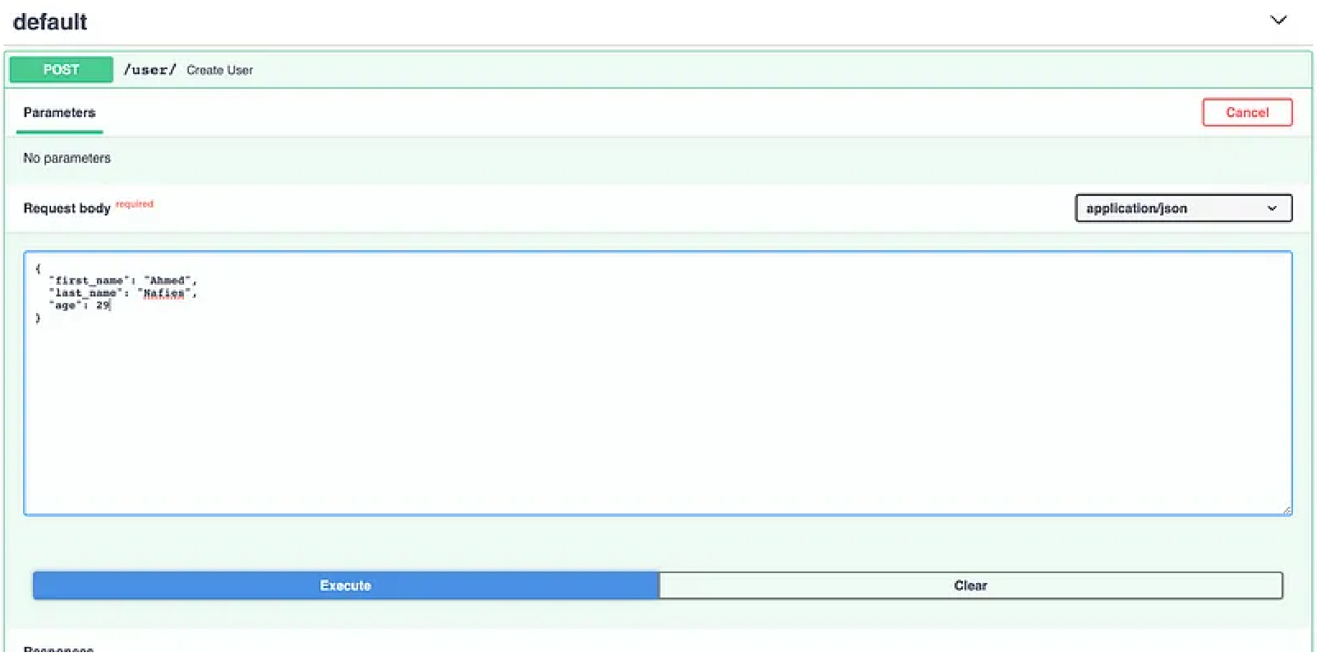
migrate

```
docker-compose run web alembic upgrade head
```

now lets run it

```
docker-compose up
```

open your browser and go to `http://localhost:8000`



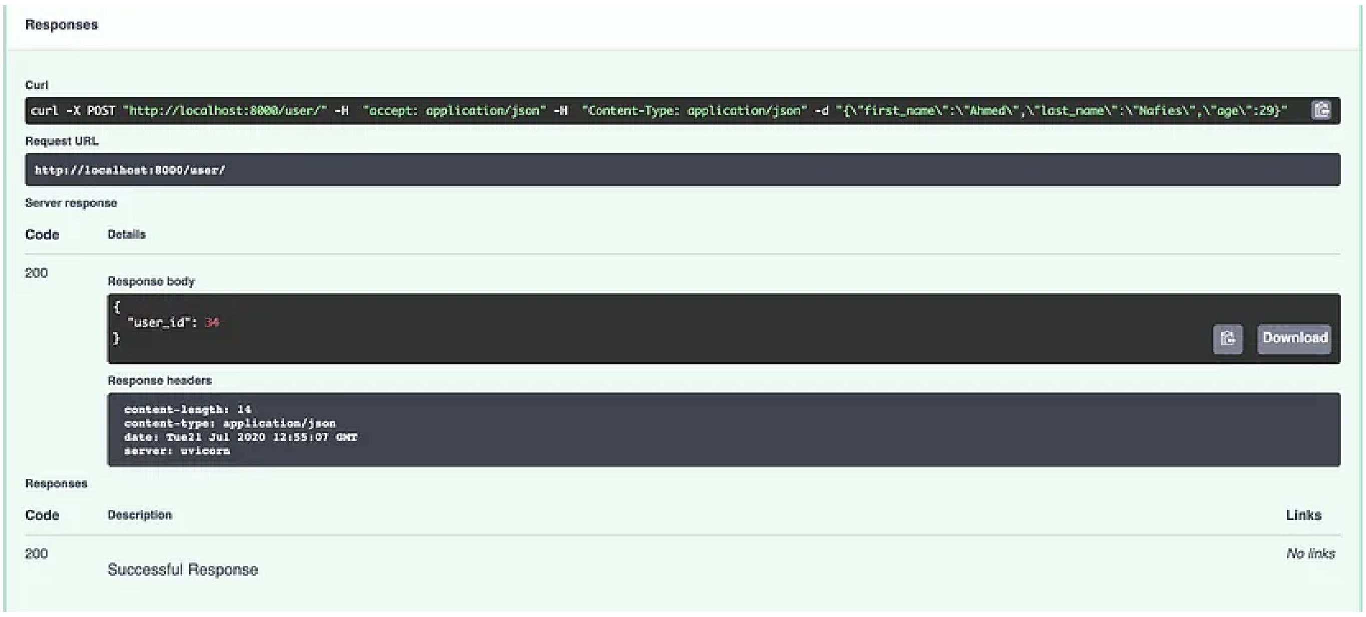
default

No parameters

Request body required

```
{
  "first_name": "Ahmed",
  "last_name": "Nafies",
  "age": 29
}
```

POST request to create a user



Responses

Curl

```
curl -X POST "http://localhost:8000/user/" -H  "accept: application/json" -H  "Content-Type: application/json" -d "{\"first_name\":\"Ahmed\",\"last_name\":\"Nafies\",\"age\":29}"
```

Request URL

```
http://localhost:8000/user/
```

Server response

Code	Details
200	Response body

```
{
  "user_id": 34
}
```

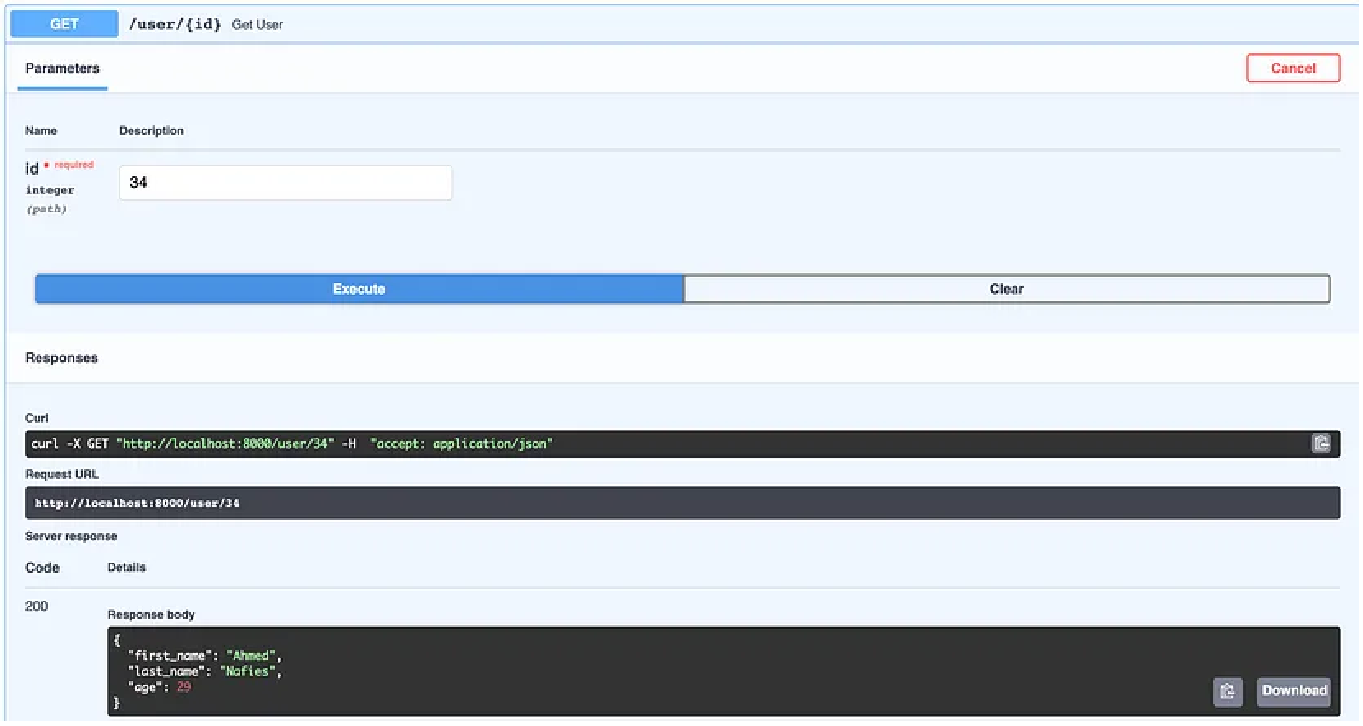
Response headers

```
content-length: 14
content-type: application/json
date: Tue21 Jul 2020 12:55:07 GMT
server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response	*No links*

response of the previous request



id * required
integer
(path)
34

Curl

```
curl -X GET "http://localhost:8000/user/34" -H  "accept: application/json"
```

Request URL

```
http://localhost:8000/user/34
```

Server response

Code Details

200 Response body

```
{
  "first_name": "Ahmed",
  "last_name": "Nafies",
  "age": 29
}
```

GET request of the same user with response

I hope that this tutorial was comprehensive enough on how to use FastAPI with PostgreSQL, SQLAlchemy, and Alembic utilizing the power of async.

The full code for this article is here on github.

Hint: A new article has been released with SQLAlchemy 2.0, check it out here