FastAPI with SQLAlchemy, PostgreSQL and Alembic and of course Docker [Part-1]

Ahmed Nafies · Follow
6 min read · Mar 1, 2020

The comprehensive guide (tutorial) to using relational databases with FastAPI

Intro

The purpose of this article is to create a simple guide on how to use FastAPI with relational databases and use Alembic for migrations. An implementation that can be used in production

Installation

I will be using `pipenv` to manage both my packages and the virtual environment. Feel free to manage your own packages in any way you like

Dependencies

- python ≥ 3.5
- fastapi
- pydantic
- fastapi-sqlalchemy
- alembic
- psycopg2
- uvicorn

lets create a new directory (you can call it whatever you want)

i will call it `fastapi_sqlalchemy_alembic`

open the terminal and write

```
cd fastapi_sqlalchemy_alembic
```

```
pipenv install --three fastapi fastapi-sqlalchemy pydantic alembic psycopg2 uvicorn
```

I will be using docker compose to manage the database, you might get some errors regarding install `psycopg2` if you are using Mac-OS but since we will use docker anyways, we would not care so much about it.

Main.py

lets start with a simple `main.py` for Fastapi

main.py

```
import uvicorn
from fastapi import FastAPI

app = FastAPI()

@app.post("/user/", response_model=User)
def create_user(user: User):
    return user

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

Docker Configuration

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 "alembic upgrade head && 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
```

The above configuration will create a cluster with 3 containers

1. web container — where the actual code will run
2. db container
3. pgadmin container

Now in your current directory you should see 5 files

1. Pipfile
2. Pipfile.lock
3. Dockerfile
4. docker-compose.yml
5. main.py

Now lets build the docker cluster, by running the following command in the terminal

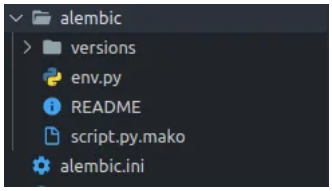
```
docker-compose build
```

Alembic

Let initialize Alembic by running the following cmd in the terminal in the same directory

```
alembic init alembic
```

that will create following a directory called `alembic` and config file `alembic.ini`



open the `alembic.ini` file with your editor and change line 38 from

```
sqlalchemy.url = driver://user:pass@localhost/dbname
```

to

```
sqlalchemy.url =
```

and we will add the postgres db url in the `alembic/env.py` file

since we are creating configuration that should work in production, we will not be able to push the `alembic.ini` file with database username and password in it. We will need to read that from the environment variables instead and that we can do in `alembic/env.py`

lets install `python-dotenv`

```
pipenv install python-dotenv
```

since we added a new package lets rebuild docker to include it

```
docker-compose build
```

lets create a `.env` file

and add the following to it

```
DATABASE_URL = postgresql+psycopg2://postgres:postgres@db:5432
```

How did we figure out the database url?

```
DATABASE_URL = postgresql+psycopg2://{user}:{password}@{host}:{port}
```

if we check the docker-compose.yml config for the database

```
db:
    image: postgres:11
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=test_db
```

we will see that `user=postgres` , `password=postgres` and since we are in the docker world, the database host will not be `localhost` but the name of the container, in our case we named it `db`

so lets add this line to our `.env`

```
DATABASE_URL = postgresql+psycopg2://postgres:postgres@db:5432
```

Open `alembic\env.py` and it should look like this

```
from logging.config import fileConfig

from sqlalchemy import engine_from_config
from sqlalchemy import pool
```

```
from alembic import context

# this is the Alembic Config object, which provides
# access to the values within the .ini file in use.
config = context.config

# Interpret the config file for Python logging.
# This line sets up loggers basically.
fileConfig(config.config_file_name)

# add your model's MetaData object here
# for 'autogenerate' support
# from myapp import mymodel
# target_metadata = mymodel.Base.metadata
target_metadata = None
```

We will need to make changes as follows

```
from logging.config import fileConfig

from sqlalchemy import engine_from_config
from sqlalchemy import poolfrom alembic import context

# ---------------- added code here -------------------------#
import os, sys
from dotenv import load_dotenv

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

#-------------------------------------------------------------#

# this is the Alembic Config object, which provides
# access to the values within the .ini file in use.
config = context.config

# ---------------- added code here -------------------------#
# this will overwrite the ini-file sqlalchemy.url path
# with the path given in the config of the main code
config.set_main_option("sqlalchemy.url", os.environ["DATABASE_URL"])

#-------------------------------------------------------------#

# Interpret the config file for Python logging.
# This line sets up loggers basically.
fileConfig(config.config_file_name)

# add your model's MetaData object here
# for 'autogenerate' support
# from myapp import mymodel
# target_metadata = mymodel.Base.metadata

# ---------------- added code here -------------------------#
import models
#-------------------------------------------------------------#

# ---------------- changed code here -------------------------#
# here target_metadata was equal to None

target_metadata = models.Base.metadata
#-------------------------------------------------------------#
```

Models

Now lets create our Models to be migrated to PostgreSQL

models.py

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String

Base = declarative_base()

class User(Base):
    __tablename__ = "users"
```

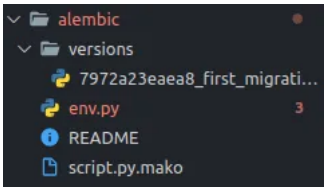
```
    id = Column(Integer, primary_key=True, index=True)
    first_name = Column(String,)
    last_name = Column(String)
    age = Column(Integer)
```

Now lets make our first migration

```
docker-compose run web alembic revision--autogenerate -m "First
migration"
```

```
Starting fastapi_sqlalchemy_alembic_db_1 ... done
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.autogenerate.compare] Detected added table 'users'
INFO  [alembic.autogenerate.compare] Detected added index 'ix_users_id' on '['id']'
  Generating /code/alembic/versions/7972a23eaea8_first_migration.py ...  done
```

if the command ran successfully you should see a new file generated under versions



lets run the migration

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

```
Starting fastapi_sqlalchemy_alembic_db_1 ... done
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> 7972a23eaea8, First migration
```

Pgadmin

in order to check our created migrations

run in terminal

`docker-compose up` and wait a little, it takes some time to load.

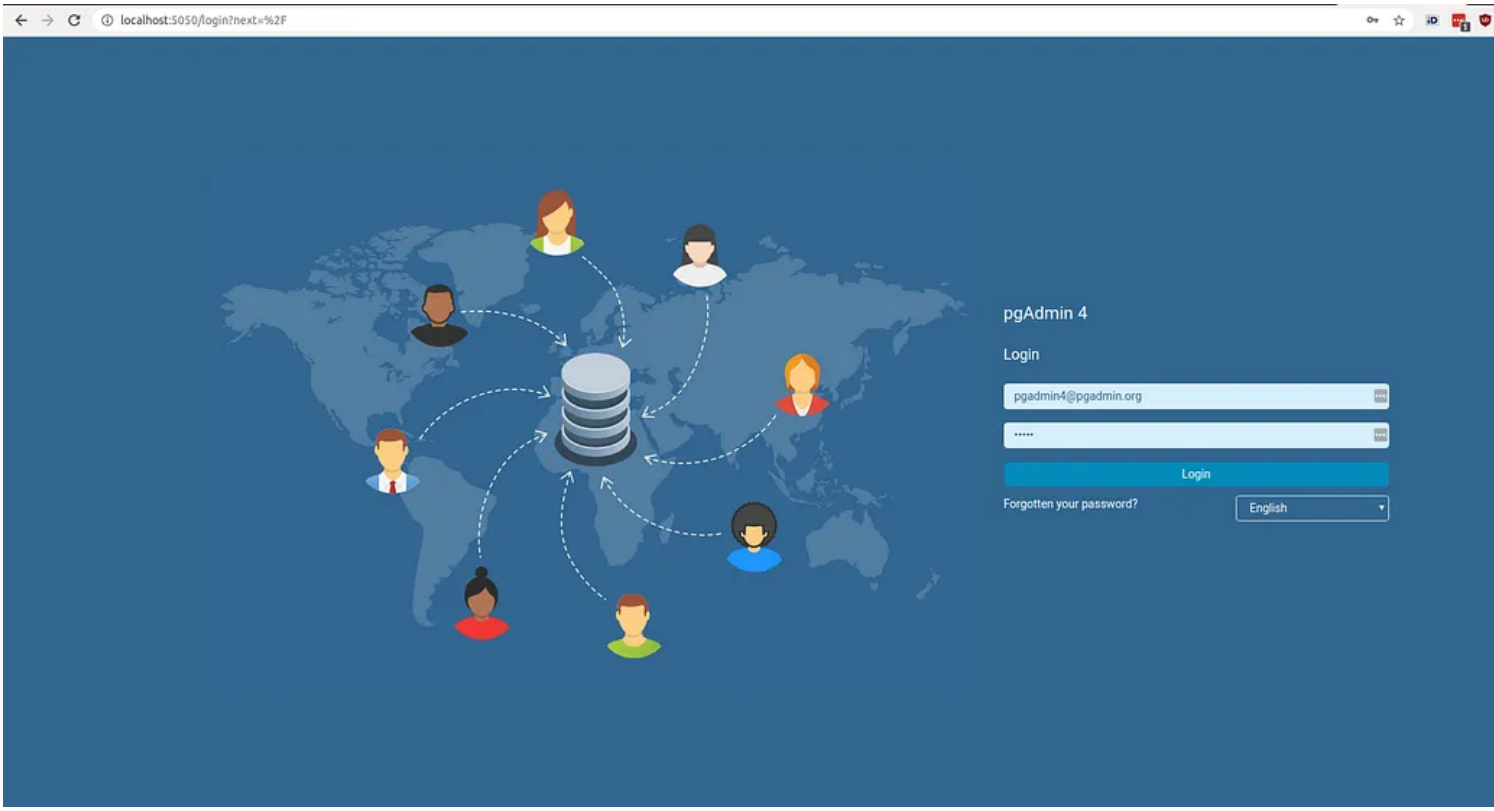
go to `localhost:5050`

and login with `user=pgadmin4@pgadmin.org` and `password=admin`

again we did set this up before in our `docker-compose.yml`

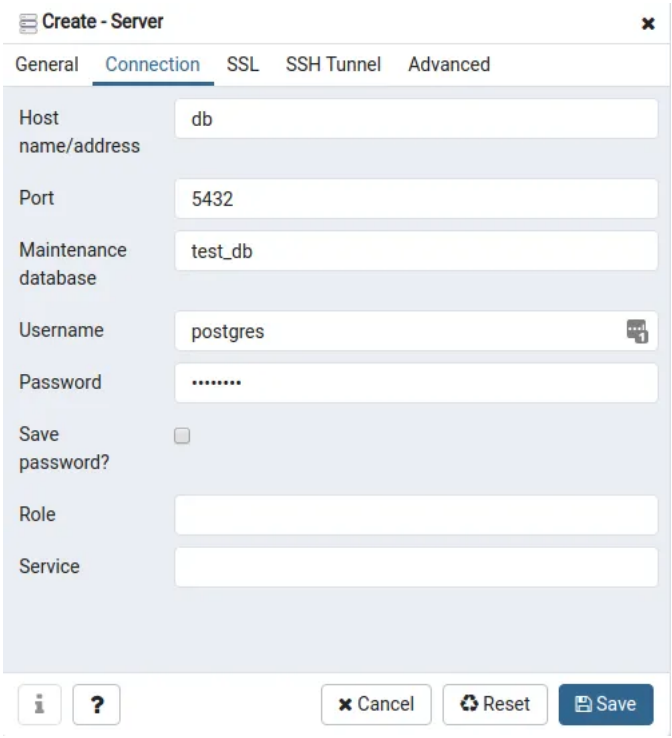
```
pgadmin:
    container_name: pgadmin
    image: dpage/pgadmin4
    environment:
      - PGADMIN_DEFAULT_EMAIL=pgadmin4@pgadmin.org
      - PGADMIN_DEFAULT_PASSWORD=admin
```

```
ports:
  - "5050:80"
depends_on:
  - db
```



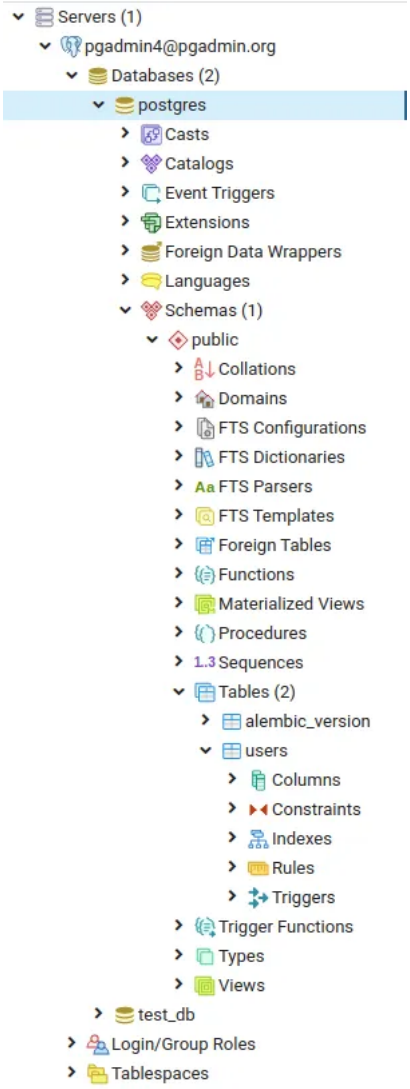
In general change choose a name for the connection

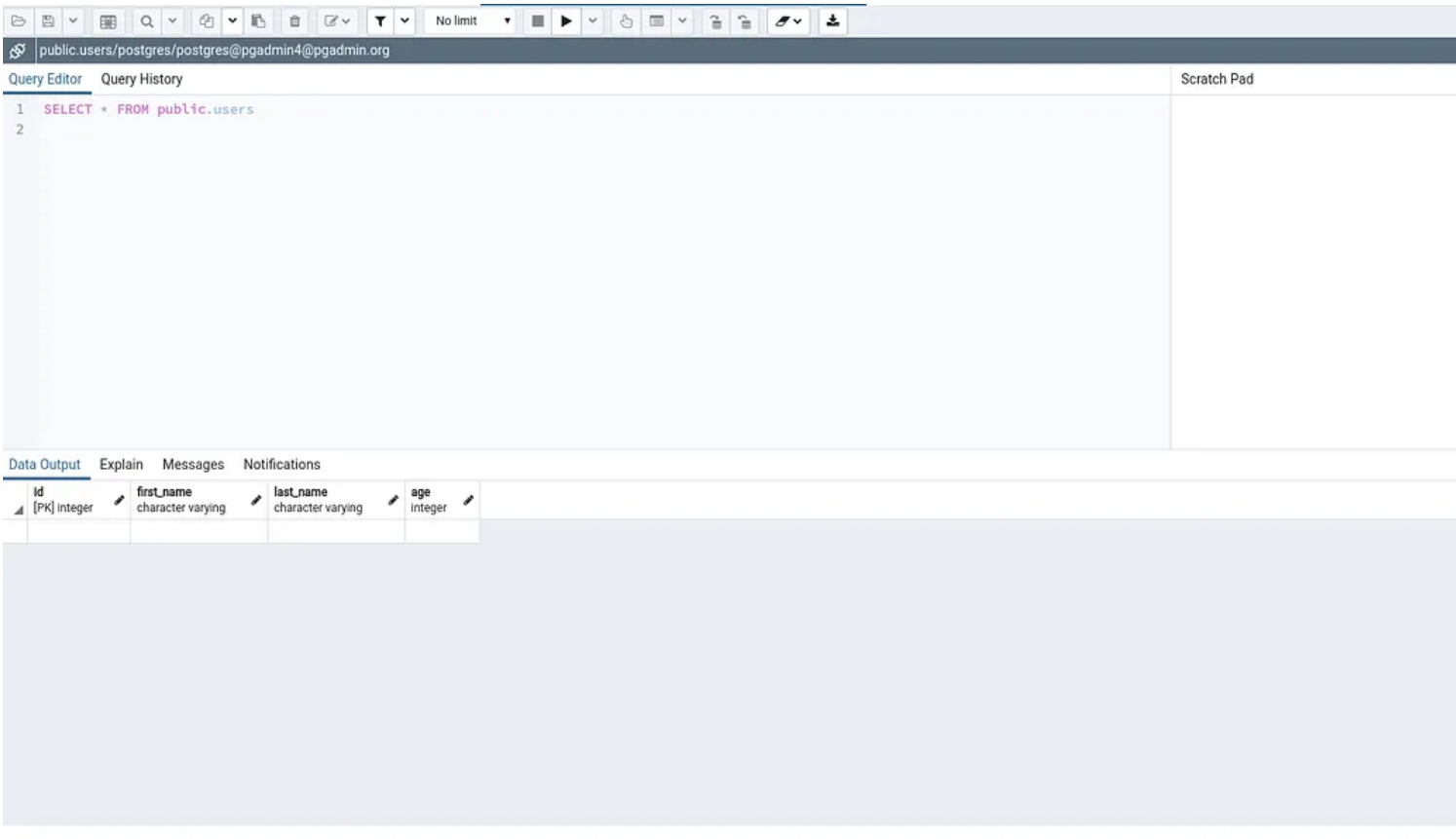
Connect to database server, password is `postgres` as well



lets find our users table

Servers > {your-connection-name} >Databases > postgres >Schemas >public>Tables >users


Servers (1)
pgadmin4@pgadmin.org
Databases (2)
postgres
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Schemas (1)
public
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Procedures
Sequences
Tables (2)
alembic_version
users
Columns
Constraints
Indexes
Rules
Triggers
Trigger Functions
Types
Views
test_db
Login/Group Roles
Tablespaces


No limit
public.users/postgres/postgres@pgadmin4@pgadmin.org
Query Editor
Query History
Scratch Pad
1 SELECT * FROM public.users
2
Data Output
Explain
Messages
Notifications
id
[PK] integer
first_name
character varying
last_name
character varying
age
integer

Now we can surely say that our migration is a success.

now we have fully implemented ORM with Alembic migrations, Awesome. Next step is to connect it to Pydantic schema.

Schema — Pydantic Model

schema.py

```
from pydantic import BaseModel


class User(BaseModel):
    first_name: str
    last_name: str = None
    age: int

    class Config:
        orm_mode = True
```

Notice that we have `Config` class where we set `orm_mode=True` and is all what we need for Pydantic models, without Sqlalchemy model objects will not be serialized to JSON.

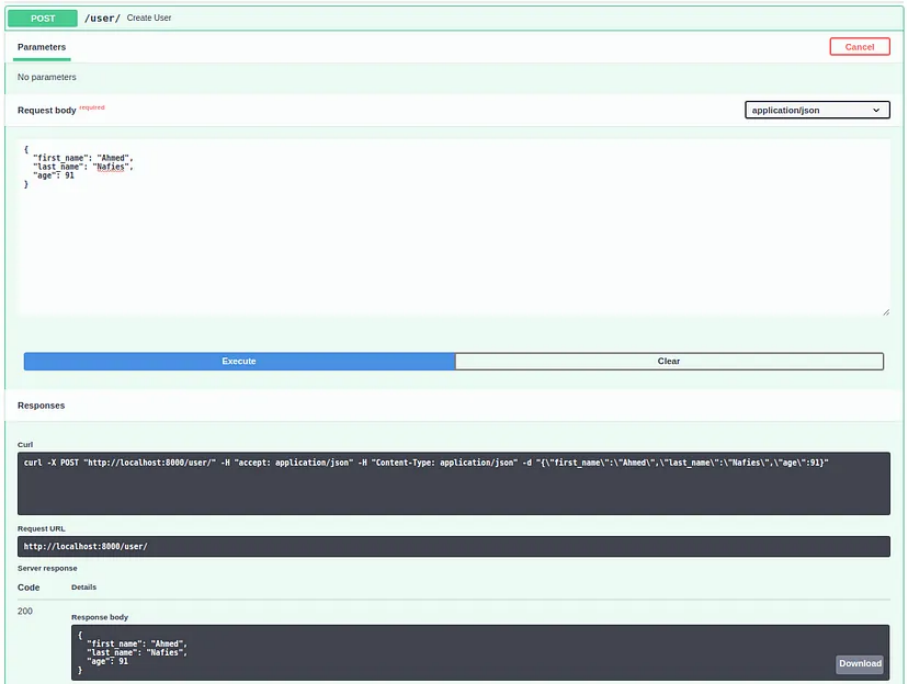
Lets connect everything in `main.py`

```
import uvicorn
from fastapi import FastAPI

#--------------- added code ------------------------#
import os
from fastapi_sqlalchemy import DBSessionMiddleware
from fastapi_sqlalchemy import db
from models import User as ModelUser
from schema import User as SchemaUser
from dotenv import load_dotenv

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

app = FastAPI()
#--------------- added code ------------------------#
app.add_middleware(DBSessionMiddleware,
db_url=os.environ["DATABASE_URL"])
#---------------------------------------------------#

#--------------- modified code ---------------------#
@app.post("/user/", response_model=SchemaUser)
def create_user(user: SchemaUser):
    db_user = ModelUser(
        first_name=user.first_name, last_name=user.last_name,
age=user.age
    )
    db.session.add(db_user)
    db.session.commit()
    return db_user
#---------------------------------------------------#

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

great lets run it again

```
docker-compose up
```

now lets go to `\docs` and invoke `Create User` endpoint



Awesome, lets check from `pgadmin` if it actually worked

go to `localhost:5050`

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

The full code for this article is here on github.

In **Part-2** we will discuss how to work with databases **asynchronously**.