Technical Documentation

REST API Service

Created By: Nilraj Shrestha

Date: 04/25/2022

Table of Content

1.Setup

In order to build the REST Service, you will need to install several software.

- Anaconda Distribution
  https://www.anaconda.com/products/distribution

- Visual Studio Code
  https://code.visualstudio.com/

- Postman API platform
  https://www.postman.com/downloads/

- pgAdmin
  https://www.pgadmin.org/download/

After the required Environment is setup, create a folder 'apiservice' and install the FASTAPI package in the python terminal

pip install fastapi[all]

Inside the 'apiservice' create another folder 'app' and inside this create a __init__.py file. Leave the __init__.py blank and create another file main.py. Inside the main.py file copy

```
from fastapi import FastAPI

app = FastAPI()

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

Now we need to start the live server to start the server. Go to the terminal and give the following command

uvicorn app.main:app --reload

*Notes: In the above code app is the folder name and main is the python file (app.main). The --reload command automatically refresh the server when any changes are made.

After you run the code you should see something that look like this:

←[32mINFO←[0m: Started server process [←[36m15448←[0m]

←[32mINFO←[0m: Waiting for application startup.

←[32mINFO←[0m: Application startup complete.

←[32mINFO←[0m: Uvicorn running on ←[1mhttp://127.0.0.1:8000←[0m (Press CTRL+C to quit)

Copy http://127.0.0.1:8000 and paste it in your browser. You should see the return statement

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

Now let's setup Postman.

Inside the Postman, please create a new collection("Service"), and then create a new request. As our method is "GET", select the method and paste the server url http://127.0.0.1:8000 and when you press send you should see the return statement {"message":"Hello World"}.

2.Setup Database

Once you have successfully setup the pgAdmin software, create a new server. Inside the server create a new database "fastapi'. Now in the folder 'apiservice' create a .env file. And inside the .env file declare the following variable:

```
DATABASE_HOSTNAME=localhost # hostname
DATABASE_PORT=5432 # your port number when installing the pgAdmin
DATABASE_PASSWORD=*****
DATABASE_NAME=fastapi # database name where you want to work
DATABASE_USERNAME=postgres #your username
```

Now inside the ‘app’ folder create a config.py file and paste the following code:

```
from pydantic import BaseSettings

class Settings(BaseSettings):
    database_hostname: str
    database_port: str
    database_password: str
    database_name: str
    database_username: str

    class Config:
        env_file =".env"

settings = Settings()
```

*Note: pydantic is case insensitive

In the terminal give the following command:

pip install pyscopg2

pip install sqlalchemy

In the ‘app’ folder create a database.py file. Inside the database file copy the following command

```
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import create_engine, false
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from .config import settings

SQLALCHEMY_DATABASE_URL = f'postgresql://{settings.
database_username}:{settings.database_password}@{settings.
database_hostname}:{settings.database_port}/{settings.database_name}'

# Engine is responsible for connection to database
engine = create_engine(SQLALCHEMY_DATABASE_URL)

SessionLocal = sessionmaker(autocommit=False,autoflush=False,bind=engine)

Base = declarative_base()

def get_db():
    db=SessionLocal()
    try:
        yield db
```

```
    finally:
        db.close()
```

Now create a models.py file. Here we create the table define the columns of the table in our database.
copy the following commands.

```
from .database import Base
from sqlalchemy import Column,Integer,String,ARRAY,BIGINT

class Users(Base):
    __tablename__ = "user"
    id = Column(Integer,primary_key=True,nullable= False)
    lastName = Column(String,nullable=False)
    firstName = Column(String,nullable=False)
    email = Column(ARRAY(String(30)),nullable=False,unique=True)
    phonenumber = Column(ARRAY(BIGINT),nullable=False)


class email(Base):
    __tablename__ = "email"
    id = Column(Integer,primary_key= True,nullable= False)
    email = Column(String,nullable=False)

class phonenumber(Base):
    __tablename__ = "phonenumber"
    id = Column(Integer,primary_key= True,nullable= False)
    phonenumber = Column(BIGINT,nullable=False)
```

*Note: __table__name is used to declare the table name.

Now we need to define the schemas. Inside the ‘app’ create a schemas.py file. Here we declare the datatype of our variable that will be used in the HTTP methods and validate the data that the user creates.

```python
from pydantic import BaseModel,EmailStr


class CreateUser(BaseModel):
    lastName: str
    firstName: str
    email:list[EmailStr]
    phonenumber:list[int]

    class Config:
        orm_mode =True

class PostResponse(CreateUser):
    class Config:
        orm_mode =True


class CreateEmail(BaseModel):
    email:EmailStr

    class Config:
        orm_mode =True

class CreatePhoneNumber(BaseModel):
    phonenumber:int

    class Config:
        orm_mode =True
```

The class Config is to ensure that the data is returned as json.

3.Create User

Inside the “app” folder create a new folder “router”. As we did previously create a __init__.py file, leave the file blank and create a user.py file. Inside the user.py file copy the following code. We are going to CREATE the user and return the user by id and Name and also DELETE the user by id.

```python
from ..  import models,schemas
from fastapi import FastAPI,status,Depends,HTTPException,APIRouter
from sqlalchemy.orm import Session
from ..database import engine,get_db

router = APIRouter(
    prefix = "/users"
)

@router.post("/",status_code=status.HTTP_201_CREATED,response_model=schemas.PostR
esponse)
def create_user(user:schemas.CreateUser,db:Session = Depends(get_db)):
    new_post = models.Users(**user.dict())
    db.add(new_post)
    db.commit()
    db.refresh(new_post)
    return new_post

@router.get("/{id}",response_model=schemas.PostResponse)
def get_user_id(id:int, db: Session = Depends(get_db)):
    user = db.query(models.Users).filter(models.Users.id == id).first()
    if not user:
        raise HTTPException(status_code= status.HTTP_404_NOT_FOUND,
                            detail=f"post with id: {id} was not found")
    return user


@router.get("/name/{name}")
def get_user_name(name:str,db: Session = Depends(get_db)):
    post = db.query(models.Users).filter(models.Users.firstName == name).first()
    if not post:
        raise HTTPException(status_code= status.HTTP_404_NOT_FOUND,
                            detail=f"post with id: {name} was not found")
    return post

@router.delete("/{id}",status_code=status.HTTP_204_NO_CONTENT)
def delete_post(id: int, db:Session = Depends(get_db)):
    delpost = db.query(models.Users).filter(models.Users.id == id)
    if delpost.first() == None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f"post with id {id} does not exist ")
    delpost.delete(synchronize_session = False)
    db.commit()
```

*Note: The commented-out code is to get the user data according to the input name. This is commented out because the get method will look for the first get method and implement which is in our case “return user by id”.

Now we need to call the module in main.py

```
from fastapi import FastAPI

from . import models
from .database import engine
from .routers import user,email,phonenumber
from .config import settings

#Comment out when creating a new database
models.Base.metadata.create_all(bind=engine)

app = FastAPI()

app.include_router(user.router)

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

Go to postman and create new request and select the POST method and then send data and check if new user is created also check the other HTTP method.

4.Create Email

In the ‘routers’ folder create a new file email.py. We are going to CREATE the email and also UPDATE the email. Copy the following code

```
from ..  import models,schemas
from fastapi import FastAPI,status,Depends,HTTPException,APIRouter
from sqlalchemy.orm import Session
from ..database import engine,get_db

router = APIRouter(
    prefix = "/email"
)

@router.post("/",status_code=status.HTTP_201_CREATED,response_model=schemas.Creat
eEmail)
```

```
def create_email(email:schemas.CreateEmail,db:Session = Depends(get_db)):
    additionalemail = models.email(**email.dict())
    db.add(additionalemail)
    db.commit()
    db.refresh(additionalemail)
    return additionalemail

@router.put("/{id}",response_model=schemas.CreateEmail)
def update_email(id:int,updated_email:schemas.CreateEmail,
db:Session = Depends(get_db)):
    update_post = db.query(models.email).filter(models.email.id == id)

    if update_post == None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f"post with id {id} does not exist")
    update_post.update(updated_email.dict(),synchronize_session=False)
    db.commit()
    return update_post.first()
```

Again, go to main.py file and add the module. Similar to way we did in CREATE USER

```
app.include_router(email.router)
```

5.Create PhoneNumber

Inside the 'routers' folder creates a new phonenumber.py file. Here we CREATE additional phonenumber and UPDATE the phone number.

```
from ..  import models,schemas
from fastapi import FastAPI,status,Depends,HTTPException,APIRouter
from sqlalchemy.orm import Session
from ..database import engine,get_db

router = APIRouter(
    prefix = "/phonenumber"
)

@router.post("/",status_code=status.HTTP_201_CREATED,
response_model=schemas.CreatePhoneNumber)
def create_phone_number(number:schemas.CreatePhoneNumber,
db:Session = Depends(get_db)):
    additionalnumber = models.phonenumber(**number.dict())
    db.add(additionalnumber)
    db.commit()
```

```
    db.refresh(additionalnumber)
    return additionalnumber

@router.put("/{id}",response_model=schemas.CreatePhoneNumber)
def update_phone_number(id:int,updated_number:schemas.CreatePhoneNumber,
db:Session = Depends(get_db)):
    update_post = db.query(models.phonenumber).filter(models.phonenumber.id ==
id)

    if update_post == None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f"post with id {id} does not exist")

    update_post.update(updated_number.dict(),synchronize_session=False)
    db.commit()

    return update_post.first()
```

Similarly, we call the module in the main.py file.

```
app.include_router(phonenumber.router)
```

Go to postman create new request for every method we have use and test it. In SQLAlchemy once the table has been created and if we need to add any new column, we have to delete the database so, in order to go around this we use alembic to update our database. Install the alembic library using

pip install alembic

Now in the terminal initialize the alembic library

alembic init alembic

This should create an alembic folder and a alembic.ini file. Now we need to configure the database with alembic library. Go to env.py file in alembic folder and paste the following code

```
from app.models import Base
from app.config import settings
import psycopg2

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

```
config.set_main_option(
"sqlalchemy.url",
f'postgresql+psycopg2://{settings.database_username}:{settings.database_password}
@{settings.database_hostname}:{settings.database_port}/{settings.database_name}')

target_metadata = Base.metadata
```

This code is use to connect to the database and leave the rest of the code as is it in the env.py file. Also go to alembic.ini and in sqlalcehmy.url clear the line.

```
sqlalchemy.url =
```

Inside the terminal give the following command

alembic revision -m “update_user_table”

Check the folder ‘version’ a file should be created, with some number and update_user_table. If you want to update the table this where you can write the update code and also you can delete the table here.

```
from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision = '5e0a67b99a37'
down_revision = 'c7917d8cc241'
branch_labels = None
depends_on = None


def upgrade():
    #update the phonenumber table
    pass

def downgrade():
    pass
```

The upgrade code goes in the upgrade function and the downgrade function is where the delete code goes. Please refer to this documentation for using sql query for alembic.

https://alembic.sqlalchemy.org/en/latest/api/ddl.html

In order to make the upgrade give the following command in the terminal

alembic upgrade (revision number)

```
alembic upgrade 5e0a67b99a37
```