**BULDING AN API IN PYTHON using FASTAPI**

Social Media Type App (CRUD Operations)

**Tools used:**

* Python
* Postman
* VS Code
* Postgres
* Psycopg 2(Python DB Adapted for Postgres)
* FastAPI
* Pydantic (Validation, defines schema)
* SQLALCHEMY(Object Relational Mapper)
* OAuth2
* Alembic (Migration Tool)
* Git/Github
* Heroku
* Gunicorn
* Docker
* NGINX
* CI/CD

**Setting up Virtual Environment**

py -3 -m venv <name>

**Change interpreter to virtual one**

View->Command Palette->Select Int Path->.\venv\Scripts\python.exe

**Make sure terminal is using virtual env**

venv\Scripts\activate.bat

**Install dependencies w/ pip (from FastAPI)**

pip install fastapi[all]

pip freeze (shows all dependencies install)

**Starting FastAPI (in VS Code)**

from fastapi import FastAPI

from fastapi.params import Body

app = FastAPI()

@app.get("/") (path operation)

async def root():

return {"message": "Hello World"}

uvicorn app.main:app -–reload (in command line, starts app and updates on saves)

**Use Postman to test app, not over the browser**

**HTTP Requests**

@app.get("/") (path operation)

def root():

return {"message": "Hello World"}

@app.get("/posts")

def get\_posts():

return {"data": "This is your posts"}

@app.post("/posts")

def create\_posts(payload: dict = Body(…)): #pulls body from message

return {"new post": f"title {payload[‘title’]} content: {payload[‘content’]}"}

**Creating a Schema(Model) with Pydantic**

from pydantic import BaseModel (in main.py)

from typing import Optional

class Post(BaseModel):

title: str

content: str

published: bool = True #optional for user, True set as default

rating = Optional[int] = None #optional

@app.post("/posts")

def create\_posts(post: Post): #uses Post Model as reference

return {“data”: post}

**CRUD Operations**

Create – Post - /posts

Read – Get(2) - /posts OR /posts/{id}

Update – Put/Patch - /posts/{id}

Delete – Delete - /posts/{id}

**Storing in Array (hard-coded values…)**

from random import randrange

my\_posts = [{“title”: “title for post 1”, “content”: “content for post 1”, “id”: 1},{“title”: “title for post 2”, “content”: “content for post 2”, “id”: 2}]

def find\_post(id):

for p in my\_posts:

if p[‘id’] == id:

return p

def find\_index\_post(id):

for i, p in enumerate(my\_posts):

if p[‘id’] == id:

return i

@app.get("/posts")

def get\_posts():

return {"data": my\_posts}

@app.post("/posts", status\_code=status.HTTP\_201\_CREATED)

def create\_posts(post: Post):

post\_dict = post.dict()

post\_dict[‘id’] = randrange(0, 1000000)

my\_posts.append(post\_dict)

return {"data": post\_dict}

@app.get("/posts/{id}")

def get\_post(id: int):

post = find\_id(id)

return {"post detail": post}

@app.delete(“/posts/{id}”, status\_code=status.HTTP\_204\_NOT\_FOUND)

def delete\_post(id: int):

index = find\_index\_post(id)

if index == None:

raise HTTPException(status\_code=status.HTTP\_404\_NO\_CONTENT, detail= f”post with id: {id} does not exist”)

my\_posts.pop(index)

return Response(status\_code=status.HTTP\_204\_NO\_CONTENT)

@app.put(“/posts/{id}”)

def update\_post(id: int, post: Post):

index = find\_index\_post(id)

if index == None:

raise HTTPException(status\_code=status.HTTP\_404\_NO\_CONTENT, detail= f”post with id: {id} does not exist”)

post\_dict = post.dict() #converts to python dictionary

post\_dict[‘id’] = id #adds id field to dictionary

my\_posts[index] = post\_dict #replaces post

return {“data”: post\_dict}

**Changing Response Status Codes**

from fastapi import FastAPI, Response, status, HTTPException

@app.get("/posts/{id}")

def get\_post(id: int):

post = find\_id(id)

if not post:

raise HTTPException(status\_code= status.HTTP\_404\_NOT\_FOUND, detail= f”post with id {id} was not found”)

#response.status\_code= status.HTTP\_404\_NOT\_FOUND

#return {‘message’. f”post with id: {id} was not found”}

return {"post detail": post}

@app.post("/posts", status\_code=status.HTTP\_201\_CREATED)

def create\_posts(post: Post):

post\_dict = post.dict()

post\_dict[‘id’] = randrange(0, 1000000)

my\_posts.append(post\_dict)

return {"data": post\_dict}

**Automatic Documentation with SwaggerUI**

path/docs or path/redoc

**Create a Folder(app), Create/Add \_\_init\_\_.py (creates package)**

uvicorn app.main:app –reload (in command line, starts app and updates on saves)

**Install Postgres Database**

**Create Database**

**Install Psycopg 2 (Python DB Adapter)**

(main.py)

import psycopg2

from psycopg2.extras import RealDictCursor

import time

while True:

    try:

conn = psycopg2.connect(host='localhost', database='fastapi', user='postgres',password='TGates215!!', cursor\_factory=RealDictCursor)

      cursor = conn.cursor()  # used to make SQL queries

      print("Database connection was successful")

      break

    except Exception as error:

      print("Connecting to database failed")

      print("Error", error)

      time.sleep(2)

**Database CRUD (code update)**

**READ ALL**

@app.get("/posts")

def get\_posts():

    cursor.execute("""SELECT \* FROM posts """)

    posts = cursor.fetchall()

    return {"data": posts}

**CREATE**

@app.post("/posts", status\_code=status.HTTP\_201\_CREATED)

def create\_posts(post: Post):

cursor.execute("""INSERT INTO posts (title, content, published) VALUES (%s, %s, %s) RETURNING \*""", (post.title, post.content, post.published))

    new\_post = cursor.fetchone()

    conn.commit()

    return {"data": new\_post}

**READ ONE**

@app.get("/posts/{id}")

def get\_post(id: str):

    cursor.execute("""SELECT \* FROM posts WHERE id = %s """, (str(id),))

    post = cursor.fetchone()

    if not post:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} was not found")

    return {"post\_detail": post}

**DELETE**

@ app.delete("/posts/{id}", status\_code=status.HTTP\_204\_NO\_CONTENT)

def delete\_post(id: int):

    cursor.execute(

        """DELETE FROM posts WHERE id = %s RETURNING \*""", (str(id),))

    deleted\_post = cursor.fetchone()

    conn.commit()

    if deleted\_post == None:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} does not exist")

    return Response(status\_code=status.HTTP\_204\_NO\_CONTENT)

**UPDATE**

@ app.put("/posts/{id}")

def update\_post(id: int, post: Post):

cursor.execute("""UPDATE posts SET title = %s, content = %s, published = %s WHERE id = %s RETURNING \*""", (post.title, post.content, post.published, (str(id))))

    updated\_post = cursor.fetchone()

    conn.commit()

    if updated\_post == None:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} does not exist")

    return {"data": updated\_post}

**Object Relational Mapper(ORM) – with SQLALCHEMY**

**Install SQLALCHEMY**

pip install sqlalchemy (in command line)

**Create database.py file (just copy and paste)**

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

#SQLALCHEMY\_DATABASE\_URL = 'postgresql://<username>:<password>@<ip-#add/hostanem>/<database\_name>'

SQLALCHEMY\_DATABASE\_URL = 'postgresql://postgres:TGates215!!@localhost/fastapi'

engine = create\_engine(SQLALCHEMY\_DATABASE\_URL)

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

Base = declarative\_base()

#Dependency needed for sqlalchemy

def get\_db():

    db = SessionLocal()

    try:

        yield db

    finally:

        db.close()

**CREATE models.py (creates tables in Postgres)**

from datetime import timezone

from sqlalchemy import Column, Integer, String, Boolean

from sqlalchemy.sql.expression import text

from sqlalchemy.sql.sqltypes import TIMESTAMP

from .database import Base

class Post(Base):

    \_\_tablename\_\_ = "posts"

    id = Column(Integer, primary\_key=True, nullable=False)

    title = Column(String, nullable=False)

    content = Column(String, nullable=False)

    published = Column(Boolean, server\_default='TRUE', nullable=False)

    created\_at = Column(TIMESTAMP(timezone=True),

                        nullable=False, server\_default=text('now()'))

**(in main.py)**

from . import models

from .database import engine, get\_db

from sqlalchemy.orm import Session

from fastapi import FastAPI, Response, status, HTTPException, Depends

models.Base.metadata.create\_all(bind=engine)

**SQLALCHEMY READ ALL**

@app.get("/posts")

def get\_posts(db: Session = Depends(get\_db)):

    posts = db.query(models.Post).all()

    return {"data": posts}

**SQLALCHEMY POST**

@app.post("/posts", status\_code=status.HTTP\_201\_CREATED)

def create\_posts(post: Post, db: Session = Depends(get\_db)):

    new\_post = models.Post(\*\*post.dict())

    db.add(new\_post)

    db.commit()

    db.refresh(new\_post)

    return {"data": new\_post}

**SQLALCHEMY GET ONE**

@app.get("/posts/{id}")

def get\_post(id: int, db: Session = Depends(get\_db)):

    post = db.query(models.Post).filter(models.Post.id == id).first()

    if not post:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} was not found")

        # response.status\_code = status.HTTP\_404\_NOT\_FOUND

        # return {'Message': f"post with id {id} was not found"}

    return {"post\_detail": post}

**SQLALCHEMY DELETE**

@ app.delete("/posts/{id}", status\_code=status.HTTP\_204\_NO\_CONTENT)

def delete\_post(id: int, db: Session = Depends(get\_db)):

    post = db.query(models.Post).filter(models.Post.id == id)

    if post.first() == None:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} does not exist")

    post.delete(synchronize\_session=False)

    db.commit()

    return Response(status\_code=status.HTTP\_204\_NO\_CONTENT)

**SQLALCHEMY UPDATE**

@ app.put("/posts/{id}")

def update\_post(id: int, updated\_post: Post, db: Session = Depends(get\_db)):

    post\_query = db.query(models.Post).filter(models.Post.id == id)

    post = post\_query.first()

    if post == None:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} does not exist")

    post\_query.update(updated\_post.dict(), synchronize\_session=False)

    db.commit()

    return {"data": post\_query.first()}

**CREATE schemas.py file, ADD Post(BaseModel) class**

from pydantic import BaseModel

#Schema used by Pydantic

class PostBase(BaseModel):

    title: str

    content: str

    published: bool = True

class PostCreate(PostBase):

    pass

**REMOVE Post(BaseModel) from main.py, ADD “schemas” import**

**UPDATE Post references in main.py**

def create\_posts(**post: schemas.PostCreate**, db: Session = Depends(get\_db)):

def update\_post(id: int, **updated\_post: schemas.PostCreate,** db: Session = Depends(get\_db)):

**CREATE RESPONSE to Send back to User(in schemas.py)**

from datetime import datetime

class Post(PostBase): #inherits from PostBase class

id: int

created\_at: datetime

class Config: #needed to work

orm\_mode = True

**UPDATE PATH OPERATIONS(in main.py)**

@app.post("/posts", status\_code=status.HTTP\_201\_CREATED, **response\_model=schemas.Post**)

**RESPONSE for GET/READ ALL, Need to Import LIST**

from typing import Optional, List

@app.get("/posts", **response\_model=List[schemas.Post]**)

**CREATING USERS**

**Create User Table in models.py**

class User(Base):

    \_\_tablename\_\_ = "users"

    id = Column(Integer, primary\_key=True, nullable=False)

    email = Column(String, nullable=False, unique=True)

    password = Column(String, nullable=False)

    created\_at = Column(TIMESTAMP(timezone=True),

                        nullable=False, server\_default=text('now()'))

**User Registration Path Operation**

@app.post("/users", status\_code=status.HTTP\_201\_CREATED)

def create\_user(user: schemas.UserCreate, db: Session = Depends(get\_db)):

    new\_user = models.User(\*\*user.dict())

    db.add(new\_user)

    db.commit()

    db.refresh(new\_user)

    return new\_user

**Email Validation**

pip install email-validator (command-line)

(in schemas.py)

from pydantic import BaseModel, **EmailStr**

class UserCreate(BaseModel):

    email: **EmailStr**

    password: str

**RESPONSE: USER CREATION & ENCRYPT PASSWORD**

(schemas.py)

class UserOut(BaseModel):

    id: int

    email: EmailStr

created\_at: datetime

    class Config:

        orm\_mode = True

(main.py)

@app.post("/users", status\_code=status.HTTP\_201\_CREATED,  **response\_model=schemas.UserOut**)

**Hashing a Password when User Registers**

**Create a utils.py file to store all Utilities**

pip install passlib[bcrypt] (command-line)

(**utils.py**)

from passlib.context import CryptContext

pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def hash(password: str):

    return pwd\_context.hash(password)

(main.py)

from . import models, schemas, **utils**

@app.post("/users", status\_code=status.HTTP\_201\_CREATED,  response\_model=schemas.UserOut)

def create\_user(user: schemas.UserCreate, db: Session = Depends(get\_db)):

    # hash the password - user password

**hashed\_password = utils.hash(user.password)**

**user.password = hashed\_password**

    new\_user = models.User(\*\*user.dict())

    db.add(new\_user)

    db.commit()

    db.refresh(new\_user)

    return new\_user

**GET USER by ID**

(main.py)

@app.get("/users/{id}", response\_model=schemas.UserOut)

def get\_user(id: int, db: Session = Depends(get\_db)):

    user = db.query(models.User).filter(models.User.id == id).first()

    if not user:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"User with id: {id} does not exist")

    return user

**Reorganize Post AND User Routes**

**Create Router Folder in App folder**

**CREATE post.py AND user.py in ROUTER folder**

**Add Remove Routes from main.py to correct file**

**Update main.py**

(post.py)

from .. import models, schemas

from sqlalchemy.orm import Session

from fastapi import FastAPI, Response, status, HTTPException, Depends, APIRouter

from ..database import get\_db

from typing import List

router = APIRouter(

    prefix="/posts",

    tags=["Posts"]

)

@router.get("/", response\_model=List[schemas.Post])

def get\_posts(db: Session = Depends(get\_db)):

    # cursor.execute("""SELECT \* FROM posts """)

    # posts = cursor.fetchall()

    posts = db.query(models.Post).all()

    return posts

@router.post("/", status\_code=status.HTTP\_201\_CREATED, response\_model=schemas.Post)

def create\_posts(post: schemas.PostCreate, db: Session = Depends(get\_db)):

    # cursor.execute("""INSERT INTO posts (title, content, published) VALUES (%s, %s, %s) RETURNING \*""",

    #                (post.title, post.content, post.published))

    # new\_post = cursor.fetchone()

    # conn.commit()

    # new\_post = models.Post(

    #     title=post.title, content=post.content, published=post.published)

    # unpacks dictionary into correct format

    new\_post = models.Post(\*\*post.dict())

    db.add(new\_post)

    db.commit()

    db.refresh(new\_post)

    return new\_post

@router.get("/{id}", response\_model=schemas.Post)

def get\_post(id: int, db: Session = Depends(get\_db)):

    # cursor.execute("""SELECT \* FROM posts WHERE id = %s """, (str(id),))

    # post = cursor.fetchone()

    post = db.query(models.Post).filter(models.Post.id == id).first()

    if not post:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} was not found")

        # response.status\_code = status.HTTP\_404\_NOT\_FOUND

        # return {'Message': f"post with id {id} 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)):

    # cursor.execute(

    #     """DELETE FROM posts WHERE id = %s RETURNING \*""", (str(id),))

    # deleted\_post = cursor.fetchone()

    # conn.commit()

    post = db.query(models.Post).filter(models.Post.id == id)

    if post.first() == None:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} does not exist")

    post.delete(synchronize\_session=False)

    db.commit()

    return Response(status\_code=status.HTTP\_204\_NO\_CONTENT)

@router.put("/{id}", response\_model=schemas.Post)

def update\_post(id: int, updated\_post: schemas.PostCreate, db: Session = Depends(get\_db)):

    # cursor.execute(

    #     """UPDATE posts SET title = %s, content = %s, published = %s WHERE id = %s RETURNING \*""", (post.title, post.content, post.published, (str(id))))

    # updated\_post = cursor.fetchone()

    # conn.commit()

    post\_query = db.query(models.Post).filter(models.Post.id == id)

    post = post\_query.first()

    if post == None:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} does not exist")

    post\_query.update(updated\_post.dict(), synchronize\_session=False)

    db.commit()

    return post\_query.first()

(user.py)

from .. import models, schemas, utils

from sqlalchemy.orm import Session

from fastapi import FastAPI, Response, status, HTTPException, Depends, APIRouter

from ..database import get\_db

router = APIRouter(

    prefix="/users",

    tags=["Users"]

)

@router.post("/", status\_code=status.HTTP\_201\_CREATED,  response\_model=schemas.UserOut)

def create\_user(user: schemas.UserCreate, db: Session = Depends(get\_db)):

    # hash the password - user password

    hashed\_password = utils.hash(user.password)

    user.password = hashed\_password

    new\_user = models.User(\*\*user.dict())

    db.add(new\_user)

    db.commit()

    db.refresh(new\_user)

    return new\_user

@router.get("/{id}", response\_model=schemas.UserOut)

def get\_user(id: int, db: Session = Depends(get\_db)):

    user = db.query(models.User).filter(models.User.id == id).first()

    if not user:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"User with id: {id} does not exist")

    return user

(main.py)

from typing import List

from fastapi import FastAPI, Response, status, HTTPException, Depends

# Database Adapter for python

import psycopg2

# also needed for database connection

from psycopg2.extras import RealDictCursor

from sqlalchemy.orm import Session

import time

from sqlalchemy.sql.functions import user

from . import models, schemas, utils

**from .database import engine, get\_db**

**from .routers import post, user**

models.Base.metadata.create\_all(bind=engine)

app = FastAPI()

# Database connection with Adapter

while True:

    try:

        conn = psycopg2.connect(host='localhost', database='fastapi', user='postgres',

                                password='TGates215!!', cursor\_factory=RealDictCursor)

        cursor = conn.cursor()  # used to make SQL queries

        print("Database connection was successful!!")

        break

    except Exception as error:

        print("Connecting to database failed")

        print("Error", error)

        time.sleep(2)

my\_posts = [{"title": "title of post 1", "content": "content of post 1", "id": 1}, {

    "title": "favorite foods", "content": "i like pizza", "id": 2}]

def find\_post(id):

    for p in my\_posts:

        if p["id"] == id:

            return p

def find\_index\_post(id):

    for i, p in enumerate(my\_posts):

        if p['id'] == id:

            return i

**app.include\_router(post.router)**

**app.include\_router(user.router)**

@app.get("/")

def root():

    return {"message": "Welcome"}

**JWT Token Authentication (stateless, no encryption)**

**Login Process: Create auth.py in Router folder**

(auth.py)

from fastapi import Response, status, HTTPException, Depends, APIRouter

from sqlalchemy.orm import Session

from .. import database, schemas, models, utils

router = APIRouter(tags=['Authentication'])

@router.post('/login')

def login(user\_credentials: schemas.UserLogin, db: Session = Depends(database.get\_db)):

    user = db.query(models.User).filter(

        models.User.email == user\_credentials.email).first()

    if not user:

        raise HTTPException(status\_code=status.HTTP\_403\_FORBIDDEN,

                            detail=f"Invalid credentials")

    if not utils.verify(user\_credentials.password, user.password):

        raise HTTPException(status\_code=status.HTTP\_403\_FORBIDDEN,

                            detail=f"Invalid credentials")

    # create token

    # return token

    return {"token": "example token"}

**UPDATE utils.py to include Password Verification**

(utils.py)

def verify(plain\_password, hashed\_password):

    return pwd\_context.verify(plain\_password, hashed\_password)

**Create UserLogin Schema**

(schema.py)

class UserLogin(BaseModel):

    email: EmailStr

    password: str

**Update main.py with new Route**

(main.py)

app.include\_router(auth.router)

**Create Token**

pip install python-jose[cryptography] (command-line)

**Create oauth.py file in App folder**

from jose import JWTError, jwt

from datetime import datetime, timedelta

# SECRET\_KEY

# Algorithm

# Expiration time

# to get a string like this run:

# openssl rand -hex 32

SECRET\_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"

ALGORITHM = "HS256"

ACCESS\_TOKEN\_EXPIRE\_MINUTES = 30

def create\_access\_token(data: dict):

    to\_encode = data.copy()

expire = datetime.utcnow() + timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)

    to\_encode.update({"exp": expire})

    encoded\_jwt = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)

    return encoded\_jwt

(auth.py)

from .. import database, schemas, models, utils, **oauth2**

access\_token = oauth2.create\_access\_token(data={"user\_id": user.id})

    return {"access\_token": access\_token, "token\_type": "bearer"}

**Change getting user credentials using OAuth2PasswordRequestForm**

pip install python-multipart (command-line)

(auth.py)

from fastapi.security.oauth2 import OAuth2PasswordRequestForm

@router.post('/login', **response\_model=schemas.Token**)

def login(user\_credentials: **OAuth2PasswordRequestForm = Depends()**, db: Session = Depends(database.get\_db)):

    user = db.query(models.User).filter(

        models.User.email == user\_credentials.**username**).first()

***In Postman, enter credentials under “form-data”***

***(username, password)***

**Verifying Token is Valid**

**Create Schemas**

(schemas.py)

class Token(BaseModel):

    access\_token: str

    token\_type: str

class TokenData(BaseModel):

    id: Optional[str] = None

**Create functions**

(oauth2.py)

from . import schemas

from fastapi.security import OAuth2PasswordBearer

oauth2\_scheme = OAuth2PasswordBearer('login')

def verify\_access\_token(token: str, credentials\_exception):

    try:

        payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])

        id: str = payload.get("user\_id")

        if id is None:

            raise credentials\_exception

token\_data = schemas.TokenData(id=id)

    except JWTError:

        raise credentials\_exception

return token\_data

def get\_current\_user(token: str = Depends(oauth2\_scheme)):

    credentials\_exception = HTTPException(status\_code=status.HTTP\_401\_UNAUTHORIZED, detail=f"Could not valiadte credentials", headers={"WWW-Authenticate": "Bearer"})

    return verify\_access\_token(token, credentials\_exception)

**Add dependency where User Login is required (on all paths)**

@router.post("/", status\_code=status.HTTP\_201\_CREATED, response\_model=schemas.Post)

def create\_posts(post: schemas.PostCreate, db: Session = Depends(get\_db), **current\_user: int = Depends(oauth2.get\_current\_user)**):

**Test in Postman (Login User with valid credentials which creates token, then add “Bearer + token” under header in “create post)**

**Fetching User in Protected Routes**

(oauth2.py)

from sqlalchemy.orm import Session

from . import schemas, **database, models**

def get\_current\_user(token: str = Depends(oauth2\_scheme), **db: Session = Depends(database.get\_db)**):

credentials\_exception = HTTPException(status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail=f"Could not valiadte credentials", headers={"WWW-Authenticate": "Bearer"})

**token = verify\_access\_token(token, credentials\_exception)**

**user = db.query(models.User).filter(models.User.id == token.id).first()**

    return user

**Creating Environment Variables in Postman**

**Creating Foreign Key with SQLALCHEMY**

(models.py)

from sqlalchemy.sql.schema import ForeignKey

class Post(Base):

    \_\_tablename\_\_ = "posts"

    id = Column(Integer, primary\_key=True, nullable=False)

    title = Column(String, nullable=False)

    content = Column(String, nullable=False)

    published = Column(Boolean, server\_default='TRUE', nullable=False)

    created\_at = Column(TIMESTAMP(timezone=True),

                        nullable=False, server\_default=text('now()'))

**owner\_id = Column(Integer, ForeignKey(**

**"users.id", ondelete="CASCADE"), nullable=False)**

**Update Schema to include User ID in Create\_Post Response**

(schemas.py)

class Post(PostBase):

    id: int

    created\_at: datetime

**owner\_id: int**

(post.py)

new\_post = models.Post(**owner\_id=current\_user.id**, \*\*post.dict())

**Delete and Update only your own**

def delete\_post(id: int, db: Session = Depends(get\_db), current\_user: int = Depends(oauth2.get\_current\_user)):

    # cursor.execute(

    #     """DELETE FROM posts WHERE id = %s RETURNING \*""", (str(id),))

    # deleted\_post = cursor.fetchone()

    # conn.commit()

    post\_query = db.query(models.Post).filter(models.Post.id == id)

    post = post\_query.first()

    if post == None:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} does not exist")

**if post.owner\_id != current\_user:**

**raise HTTPException(status\_code=status.HTTP\_403\_FORBIDDEN,**

**detail=f"Not authorized to perform requested** **action")**

    post\_query.delete(synchronize\_session=False)

    db.commit()

    return Response(status\_code=status.HTTP\_204\_NO\_CONTENT)

**Returns only own Posts (optional)**

(post.py)

@router.get("/", response\_model=List[schemas.Post])

def get\_posts(db: Session = Depends(get\_db), current\_user: int = Depends(oauth2.get\_current\_user)):

    # cursor.execute("""SELECT \* FROM posts """)

    # posts = cursor.fetchall()

**posts = db.query(models.Post).filter(models.Post.owner\_id == current\_user.id).all()**

    return posts

def get\_post(id: int, db: Session = Depends(get\_db), current\_user: int = Depends(oauth2.get\_current\_user)):

    # cursor.execute("""SELECT \* FROM posts WHERE id = %s """, (str(id),))

    # post = cursor.fetchone()

    post = db.query(models.Post).filter(models.Post.id == id).first()

    if not post:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                            detail=f"post with id: {id} was not found")

        # response.status\_code = status.HTTP\_404\_NOT\_FOUND

        # return {'Message': f"post with id {id} was not found"}

**if post.owner\_id != current\_user.id:**

**raise HTTPException(status\_code=status.HTTP\_403\_FORBIDDEN,**

**detail=f"Not authorized to perform requested** **action")**

    return post

**SQLALCHEMY RELATIONSHIP (like a JOIN)**

(models.py)

**from sqlalchemy.orm import relationship**

class Post(Base):

    \_\_tablename\_\_ = "posts"

    id = Column(Integer, primary\_key=True, nullable=False)

    title = Column(String, nullable=False)

    content = Column(String, nullable=False)

    published = Column(Boolean, server\_default='TRUE', nullable=False)

    created\_at = Column(TIMESTAMP(timezone=True),

                        nullable=False, server\_default=text('now()'))

    owner\_id = Column(Integer, ForeignKey(

        "users.id", ondelete="CASCADE"), nullable=False)

**owner = relationship("User")**

(schemas.py)

**class UserOut(BaseModel): # must be above Post Class**

**id: int**

**email: EmailStr**

**created\_at: datetime**

**class Config:**

**orm\_mode = True**

class Post(PostBase):

    id: int

    created\_at: datetime

    owner\_id: int

**owner: UserOut**

    class Config:

        orm\_mode = True

**Query Parameters (limit, skip, search)**

def get\_posts(db: Session = Depends(get\_db), current\_user: int = Depends(oauth2.get\_current\_user), **limit: int = 10, skip: int = 0, search: Optional[str] = ""):**

    # cursor.execute("""SELECT \* FROM posts """)

    # posts = cursor.fetchall()

    print(limit)

    posts = db.query(models.Post).**filter(**

**models.Post.title.contains(search)).limit(limit).offset(skip)**.all()

    return posts

(POSTMAN Get Posts)

{{URL}}posts?limit=2&skip=0&search=will%20be

? – before query

& - to add multiple queries

%20 – for spaces between words

**main.py clean-up, adapter moved to database.py(not needed)**

from fastapi import FastAPI

from sqlalchemy.sql.functions import user

from . import models

from .database import engine

from .routers import post, user, auth

models.Base.metadata.create\_all(bind=engine)

app = FastAPI()

app.include\_router(post.router)

app.include\_router(user.router)

app.include\_router(auth.router)

@app.get("/")

def root():

    return {"message": "Welcome"}

**Create Environment Variables (instead of hard-coding values)**

**Create config.py file in app & Create .env file in fastapi**

(config.py)

class Settings(BaseSettings):

    database\_hostname: str

    database\_port: str

    database\_password: str

    database\_name: str

    database\_username: str

    secret\_key: str

    algorithm: str

    access\_token\_expire\_minutes: int

    class Config:

        env\_file = ".env"

settings = Settings()

(.env)

DATABASE\_HOSTNAME=localhost

DATABASE\_PORT=5432

DATABASE\_PASSWORD=TGates215!!

DATABASE\_NAME=fastapi

DATABASE\_USERNAME=postgres

SECRET\_KEY=09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7

ALGORITHM=HS256

ACCESS\_TOKEN\_EXPIRE\_MINUTES=30

**Update database.py & oauth2.py**

(database.py)

from .config import settings

SQLALCHEMY\_DATABASE\_URL = f'postgresql://{settings.database\_username}:{settings.database\_password}@{settings.database\_hostname}:{settings.database\_port}/{settings.database\_name}'

(oatuh2.py)

from .config import settings

SECRET\_KEY = settings.secret\_key

ALGORITHM = settings.algorithm

ACCESS\_TOKEN\_EXPIRE\_MINUTES = settings.access\_token\_expire\_minutes

**Adding Voting/Likes**

**Create Table in models.py**

(models.py)

class Vote(Base):

    \_\_tablename\_\_ = "votes"

    user\_id = Column(Integer, ForeignKey(

        "users.id", ondelete="CASCADE"), primary\_key=True)

    post\_id = Column(Integer, ForeignKey(

        "posts.id", ondelete="CASCADE"), primary\_key=True)

**Create Vote Schema**

(schemas.py)

class Vote(BaseModel):

    post\_id: int

    dir: conint(le=1)

**Create vote.py in Router Folder**

(vote.py)

from fastapi import FastAPI, Response, status, HTTPException, Depends, APIRouter

from sqlalchemy.orm import Session

from .. import schemas, database, models, oauth2

router = APIRouter(

    prefix="/vote",

    tags=['Vote']

)

@router.post("/", status\_code=status.HTTP\_201\_CREATED)

def vote(vote: schemas.Vote, db: Session = Depends(database.get\_db), current\_user: int = Depends(oauth2.get\_current\_user)):

post = db.query(models.Post).filter(models.Post.id == vote.post\_id).first()

    if not post:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

              detail=f"Post with id: {vote.post\_id} does not exist")

    vote\_query = db.query(models.Vote).filter(

        models.Vote.post\_id == vote.post\_id, models.Vote.user\_id == current\_user.id)

    found\_vote = vote\_query.first()

    if (vote.dir == 1):

        if found\_vote:

            raise HTTPException(status\_code=status.HTTP\_409\_CONFLICT,

detail=f"user {current\_user.id} has already voted on post {vote.post\_id}")

        new\_vote = models.Vote(post\_id=vote.post\_id, user\_id=current\_user.id)

        db.add(new\_vote)

        db.commit()

        return {"message": "successfully added vote"}

    else:

        if not found\_vote:

            raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND,

                                detail="vote does not exist")

        vote\_query.delete(synchronize\_session=False)

        db.commit()

        return {"message": "successfully deleted vote"}

**Add Route to main.py**

(main.py)

from .routers import post, user, auth, **vote**

app.include\_router(vote.router)

**SQL Joins with SQLALCHEMY (get all posts with # of votes)**

(post.py)

from sqlalchemy import func

@router.get("/", response\_model=List[**schemas.PostOut**])

results = db.query(models.Post,func.count(models.Vote.post\_id).label("votes")).join(

      models.Vote, models.Vote.post\_id == models.Post.id, isouter=True).group\_by(models.Post.id).all()

return results

(schemas.py)

class PostOut(BaseModel):

    Post: Post

    votes: int

**SQL Joins with SQLALCHEMY (get one post with # of votes)**

@router.get("/{id}", response\_model=**schemas.PostOut**)

post = db.query(models.Post, func.count(models.Vote.post\_id).label("votes")).join(

        models.Vote, models.Vote.post\_id == models.Post.id, isouter=True).group\_by(models.Post.id).first()

**Migration Tool (Alembic)**

pip install alembic (command-line)

alembic init alembic <-(whatever name you want)

**Import Base from database.py into env.py (gives access to models)**

(env.py)

from app.models import Base

from app.config import settings

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

**Add Database URL in alembic.ini**

(alembic.ini)

sqlalchemy.url =

**Setting up DB with Alembic**

alembic revision -m “create post table” (command-line)(creates versions folder under alembic folder with revision numbered file)

**Create “Posts” Table**

(…\_create\_posts\_table.py)

def upgrade():

    op.create\_table('posts', sa.Column('id', sa.Integer(), nullable=False, primary\_key=True), sa.Column('title', sa.String(), nullable=False))

    pass

def downgrade():

    op.drop\_table('posts')

    pass

**Implement Changes**

alembic upgrade <revision number> (command-line)

**Adding a Revision to Existing Table**

alembic revision -m “add content column to post table” (command-line)

**Add “Content” Column**

(…\_add\_content\_column\_to\_post\_table.py)

def upgrade():

    op.add\_column('posts', sa.Column('content', sa.String(), nullable=False))

    pass

def downgrade():

    op.drop\_column('posts', 'content')

    pass

**Implement New Changes**

alembic heads

alembic upgrade <revision number> or head (command-line)

To Rollback

alembic downgrade <revision number> or -1

**Add “Users” Table**

def upgrade():

    op.create\_table('users',

                    sa.Column('id', sa.Integer(), nullable=False),

                    sa.Column('email', sa.String(), nullable=False),

                    sa.Column('password', sa.String(), nullable=False),

                    sa.Column('created\_at', sa.TIMESTAMP(timezone=True),

                             server\_default=sa.text('now()'),nullable=False),

                    sa.PrimaryKeyConstraint('id'),

                    sa.UniqueConstraint('email')

                    )

    pass

def downgrade():

    op .drop\_table('users')

    pass

**Add “Owner\_ID” Foreign Key Column**

def upgrade():

    op.add\_column('posts', sa.Column('owner\_id', sa.Integer(), nullable=False))

    op.create\_foreign\_key('posts\_users\_fk', source\_table="posts", referent\_table="users", local\_cols=['owner\_id'], remote\_cols=['id'], ondelete="CASCADE")

    pass

def downgrade():

    op.drop\_constraint('post\_users\_fk', table\_name="posts")

    op.drop\_column('posts', 'owner\_id')

    pass

**Adding remaining Columns to Post Table**

def upgrade():

    op.add\_column('posts', sa.Column(

        'published', sa.Boolean(), nullable=False, server\_default='TRUE'),)

    op.add\_column('posts', sa.Column(

        'created\_at', sa.TIMESTAMP(timezone=True), nullable=False, server\_default=sa.text('NOW()')),)

    pass

def downgrade():

    op.drop\_column('posts', 'published')

    op.drop\_column('posts', 'created\_at')

    pass

**Allow Alembic to Automatically add Missing Models**

alembic revision **--autogenerate** -m "add votes automatically" (command-line)

**Can remove *models.Base.metadata.create\_all(bind=engine)* in main.py after using Alembic**

**CORS Setup (allows users outside domain to access API)**

(main.py)

from fastapi.middleware.cors import CORSMiddleware

origins = ["\*"]

app.add\_middleware(

    CORSMiddleware,

    allow\_origins=origins,

    allow\_credentials=True,

    allow\_methods=["\*"],

    allow\_headers=["\*"],

**GIT Setup**

**create .gitignore file in FASTAPI folder(\_\_pycache\_\_,venv/,.env)**

**pip freeze > requirements.txt (command-line)**

**pip install -r requirements.txt (to install required files)**