File: models.py

```
from sqlalchemy import Column, Date, DateTime, Integer, String, ForeignKey



from ..database import Base

from datetime import datetime

from sqlalchemy.orm  import relationship


class User(Base):

    __tablename__ = "users"


    # basic details

    id = Column(Integer, primary_key=True, index=True)

    email = Column(String, unique=True)

    username = Column(String, unique=True)

    firstname = Column(String, nullable=False)

    lastname = Column(String, nullable=False)

    hashed_password = Column(String, nullable=False)

    created_at = Column(DateTime, default=datetime.utcnow())
```

File: schemas.py

```
from pydantic import BaseModel, EmailStr
from datetime import date, datetime
from typing import Optional


class UserBase(BaseModel):
    email: EmailStr
    username: str
    firstname: str
    lastname: str


class UserCreate(UserBase):
    password: str


class User(UserBase):
    id: int
    created_at: datetime
```

File: services.py

```
from fastapi import Depends
from sqlalchemy.orm import Session
from passlib.context import CryptContext
from fastapi.security import OAuth2PasswordBearer
from jose import jwt, JWTError
from datetime import timedelta, datetime
from ..config import settings

from .models import User
from .schemas import UserCreate

bcrypt_context  =  CryptContext(schemes=["bcrypt"],  deprecated="auto")   #
hasing password
oauth2_bearer = OAuth2PasswordBearer(tokenUrl="v1/auth/token")
SECRET_KEY = settings.secret_key
ALGORITHM = settings.algorithm  # encoding our jwt
TOKEN_EXPIRE_MINS =  settings.access_token_expire_minutes

# check for existing user
async def existing_user(db: Session, username: str, email: str):
                db_user    =    db.query(User).filter((User.username    ==
username)|(User.email==email)).first()
    return db_user

# create access token
async def create_access_token(username: str, id: int):
```

```
    encode = {"sub": username, "id": id}
    expires = datetime.utcnow() + timedelta(minutes=TOKEN_EXPIRE_MINS)
    encode.update({"exp": expires})
    return jwt.encode(encode, SECRET_KEY, algorithm=ALGORITHM)


# get current user from token
async def get_current_user(db: Session, token: str = Depends(oauth2_bearer)):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        id: str = payload.get("id")
        expires: datetime = payload.get("exp")
        if datetime.fromtimestamp(expires) < datetime.now():
            return None
        if username is None or id is None:
            return None
        return db.query(User).filter(User.id == id).first()
    except JWTError:
        return None


# get user from user id
async def get_user_from_user_id(db: Session, user_id: int):
    return db.query(User).filter(User.id == user_id).first()


# create user
async def create_user(db: Session, user: UserCreate):
```

```
    db_user = User(
        firstname=user.firstname,
        lastname=user.lastname,
        email=user.email.lower().strip(),
        username=user.username.lower().strip(),
        hashed_password=bcrypt_context.hash(user.password)
    )
    db.add(db_user)
    db.commit()

    return db_user

# authentication
async def authenticate(db: Session, username: str, password: str):
    db_user = db.query(User).filter(User.username == username).first()
    if not db_user:
        print("no user")
        return None
    if not bcrypt_context.verify(password, db_user.hashed_password):
        return None
    return db_user
```

File: views.py

```
from fastapi import APIRouter, Depends, status, HTTPException
from fastapi.security import OAuth2PasswordRequestForm
from sqlalchemy.orm import Session
from datetime import datetime
from .schemas import UserCreate, User as UserSchema
from ..database import get_db
from  .services import (
    existing_user,
    create_access_token,
    get_current_user,
    create_user as  create_user_svc,
    authenticate
)

router = APIRouter(prefix="/auth", tags=["auth"])

@router.post("/signup", status_code=status.HTTP_201_CREATED)
async def create_user(user: UserCreate, db: Session = Depends(get_db)):
    # check existing user
    db_user = await existing_user(db, user.username, user.email)
    if db_user:
        raise HTTPException(
            status_code=status.HTTP_409_CONFLICT,
            detail="username or email already in use",
        )
```

```
    db_user = await create_user_svc(db, user)
    access_token = await create_access_token(user.username, db_user.id)

    return {
        "access_token": access_token,
        "token_type": "bearer",
        "username": user.username,
    }

# login to generate token
@router.post("/token", status_code=status.HTTP_201_CREATED)
async def login(
       form_data: OAuth2PasswordRequestForm = Depends(), db: Session = Depends(get_db)
):
            db_user = await authenticate(db, form_data.username, form_data.password)
    if not db_user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="incorrect username or password",
        )

    access_token = await create_access_token(db_user.username, db_user.id)
    return {"access_token": access_token, "token_type": "bearer"}

# get current user
```

```
@router.get("/current_user",                status_code=status.HTTP_200_OK,
response_model=UserSchema)
async def current_user(token: str, db: Session = Depends(get_db)):
    db_user = await get_current_user(db, token)
    if not db_user:
        raise HTTPException(
                    status_code=status.HTTP_401_UNAUTHORIZED,  detail="token
invalid"
        )

    return db_user
```