

File: models.py

```
from sqlalchemy import Column, Date, Integer, String, ForeignKey
from sqlalchemy.orm import relationship
from ..database import Base
from ..auth.models import User

class Profile(Base):
    __tablename__ = "profiles"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), unique=True,
nullable=False)

    date_of_birth = Column(Date, nullable=True)
    gender = Column(String, nullable=True)
    location = Column(String, nullable=True)
    interests = Column(String, nullable=True) # Store as a
comma-separated string

    profile_pic_url = Column(String, nullable=True)

    user = relationship("User", back_populates="profile")
```

File: schemas.py

```
from pydantic import BaseModel

from typing import Optional

from datetime import date

from .enums import Gender


class ProfileBase(BaseModel):

    date_of_birth: Optional[date]

    gender: Optional[Gender]

    location: Optional[str]

    interests: Optional[str]


class ProfileCreate(ProfileBase):

    pass


class ProfileUpdate(ProfileBase):

    pass


class Profile(BaseModel):

    id: int

    user_id: int

    date_of_birth: Optional[date]

    gender: Optional[Gender]

    location: Optional[str]

    interests: Optional[str]

    profile_pic_url: Optional[str]
```

```
class Config:
    from_attributes = True
```

File: services.py

```
from sqlalchemy.orm import Session

from .models import Profile

from .schemas import ProfileCreate, ProfileUpdate


def create_profile(db: Session, user_id: int, profile_data:
ProfileCreate):

    profile = Profile(user_id=user_id, **profile_data.dict())

    db.add(profile)

    db.commit()

    db.refresh(profile)

    print("created profile: ", profile)

    return profile


def update_profile(db: Session, user_id: int, profile_data:
ProfileUpdate):

    profile = db.query(Profile).filter(Profile.user_id == user_id).first()

    if not profile:

        return None

    for key, value in profile_data.dict(exclude_unset=True).items():

        setattr(profile, key, value)

    db.commit()

    db.refresh(profile)

    return profile


def get_profile(db: Session, user_id: int):

    return db.query(Profile).filter(Profile.user_id == user_id).first()
```

File: views.py

```
### File: views.py

from fastapi import APIRouter, Depends, HTTPException, UploadFile, File,
status

from sqlalchemy.orm import Session

from ..database import get_db

from .schemas import ProfileCreate, ProfileUpdate, Profile as
ProfileSchema

from .services import create_profile, update_profile, get_profile

from ..auth.services import get_current_user

import shutil

import os


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


@router.post("/", status_code=status.HTTP_201_CREATED)

async def create_user_profile(

    profile_data: ProfileCreate,

    db: Session = Depends(get_db),

    current_user: dict = Depends(get_current_user),

):

    if not current_user:

        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,

detail="Unauthorized")

        profile = create_profile(db, user_id=current_user.id,

profile_data=profile_data)

        return ProfileSchema.from_orm(profile)
```

```

@router.put("/", status_code=status.HTTP_200_OK)

async def update_user_profile(
    profile_data: ProfileUpdate,
    db: Session = Depends(get_db),
    current_user: dict = Depends(get_current_user),
):
    if not current_user:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Unauthorized")

        profile = update_profile(db, user_id=current_user.id,
profile_data=profile_data)

        if not profile:
            raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Profile not found")

        return profile


@router.post("/upload-pic", status_code=status.HTTP_200_OK)

async def upload_profile_picture(
    file: UploadFile = File(...),
    db: Session = Depends(get_db),
    current_user: dict = Depends(get_current_user),
):
    if not current_user:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Unauthorized")

```

```
file_dir = "static/profile_pics/"

os.makedirs(file_dir, exist_ok=True) # Ensure directory exists


file_location = os.path.join(file_dir,
f"{current_user.id}_{file.filename}")

with open(file_location, "wb") as buffer:
    shutil.copyfileobj(file.file, buffer)


profile = get_profile(db, user_id=current_user.id)

if not profile:
    raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Profile not found")


profile.profile_pic_url = file_location

db.commit()

db.refresh(profile)

return {"profile_pic_url": file_location}
```

File: enums.py

```
from enum import Enum
```

```
class Gender(str, Enum):
```

```
    MALE = "male"
```

```
    FEMALE = "female"
```

```
    OTHER = "others"
```