04 – Database with Flask–SQLAlchemy

ရှေ့အခန်းတွေမှာ data အတွက် dummy data နဲ့ အသုံးပြုခဲ့ပါတယ်။ ယခု database ကို သုံးပြီး အချက်အလက်တွေကို စီမံပါမယ်။ database က file တစ်ခုဖြစ်ပြီး စီစဉ်ထားတဲ့ပုံစံအတိုင်း application data တွေကို သိမ်းဆည်းပေးထားပါတယ်။ application က လိုအပ်လာတဲ့အချိန်မှာ သီးခြားလိုတဲ့ data အပိုင်းတွေကို ဆွဲထုတ်အသုံးပြုနိုင်ပါတယ်။ web application တွေအတွက် အများအားဖြင့် အသုံးပြုတဲ့ database ကတော့ Structured Query Language ကိုသုံးထားတဲ့ SQL database လို့ ခေါ်တဲ့ relational model တွေကို အခြေပြုသုံးလေ့ရှိပါတယ်။ ဒါပေမယ့် မကြာမီကာလတွေမှာ NoSQL database လို့ အသိများလာတဲ့ document–oriented and key–value database တွေကိုလည်း ပြောင်းလဲ အသုံးပြုလာကြပါတယ်။

Python မှာ open–source နဲ့ commercial နှစ်မျိုးစလုံးနဲ့ဆိုင်တဲ့ database engine packages တွေ အများအပြားရှိပါတယ်။ Flask framework ကလည်း ဘယ် database package ကို သုံးရမယ်ဆိုပြီး ကန့်သတ်ထားတာမရှိပဲ ကြိုက်နှစ်သက်ရာနဲ့ အလုပ်လုပ်နိုင်ပါတယ်။ ဒီ project မှာတော့ Flask extension wrapper ဖြစ်တဲ့ Flask–SQLAlchemy database framework ကို သုံးပါမယ်။

အရင်ဆုံး flask–sqlalchemy extension ကို install လုပ်ပါမယ်။

```
(venv)pip install flask–sqlalchemy
```

Flask–SQLAlchemy ဟာ Flask applications တွေထဲမှာ SQLAlchemy ကို အရိုးရှင်းဆုံး အသုံးပြုတဲ့ Flask extension တစ်ခုပါ ။ SQLAlchemy ကတော့ နောက်ကွယ်မှ အမျိုးမျိုး ကွဲပြားသော database (MySQL, PostgreSQL, IBMDB2, Microsoft SQL server,...) များကို ထောက်ပံ့ပေးတဲ့ စွမ်းအားကောင်း relational database framework တစ်ခုဖြစ်ပါတယ်။ Flask–SQLAlchemy မှာ အသုံးပြုမဲ့ database တစ်ခုကို URL တစ်ခုအနေနဲ့ သတ်မှတ်ပေးထားရပါမယ်။ ဒီမှာတော့ အများကြိုက်တဲ့ database engines တွေထဲကမှ ပေါ့ပေါ့ပါးပါးအသုံးပြုလို့ ကောင်းတဲ့ SQLite ကို သုံးပါမယ်။

SQLite ကို သုံးမယ်ဆိုရင် သတ်မှတ်ပေးရမဲ့ URL ပုံစံဖြစ်တဲ့ sqlite:///databasename ကို

SQLALCHEMY_DATABASE_URI key အဖြစ် Flask config object မှာ configure လုပ်ပေးရပါမယ်။

SQLite database က server မလိုတဲ့အတွက် အခြား database engines တွေလို hostname, username, password တွေမလိုပဲ database name ကို location နဲ့ ပြောရင် ရပါ ပြီ။ Flask–SQLAlchemy documentation မှာတော့ SQLALCHEMY_TRACK_MODIFICATIONS key ကို False

သတ်မှတ်ပေးဖို့ အကြံပြုထားပါတယ်။ မဖြစ်မနေ object အပြောင်းအလဲအတွက် signals တွေမဟုတ်ရင် memory ကို အနည်းငယ်သာ အသုံးပြုစေဖို့ပါ ။ စရေးကြည့်ရအောင်။

flaskblog.py ကိုဖွင့်ပါ။

1. SQLAlchemy ကို import လုပ်ပါ။
   from flask_sqlalchemy import SQLAlchemy
2. config object ကို assign လုပ်ပါ။
   app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'

SQLAlchemy class နဲ့ object တစ်ခု instantiate လုပ်ပြီးပြီဆိုတာနဲ့ အဲဒီ object က database ကို project folder မှာ ဖော်ပြပေးသလို Flask-SQLAlchemy ရဲ့ လုပ်ဆောင်ချက်အားလုံးကိုလည်း access လုပ်လို့ရပါပြီ။

3. dababase object တစ်ခု ကို instantiate လုပ်ပါ။
   db = SQLAlchemy(app) # site.db ကို application folder ထဲမှာ ကြည့်ပါ။

database ရပြီဆိုရင် Application ကနေ ထပ်တလဲလဲ အသုံးပြုမဲ့ entities တွေကို လွှဲပြောင်းဖို့ model ကို ဖန်တီးပေးရပါမယ်။ model ဆိုတာက attributes တွေနဲ့ Python class တစ်ခုပါပဲ။ database table အတွက် ကိုက်ညီတဲ့ columns အစုံအတွဲတွေ သတ်မှတ်ပေးထားတာပါ။။

ဒါဆို db instance ကနေ ထောက်ပံ့ပေးထားတဲ့ base class Model ကို သုံးပြီး မိမိလိုအပ်တဲ့ classes နဲ့ functions တွေ တည်ဆောက်ပါမယ်။ database မှာ table အမည်ကို __tablename__ class variable နဲ့ သတ်မှတ်ပေးရပါတယ်။ အကယ်၍ အမည်မပေးခဲ့ရင်တော့ Flask-SQLAlchemy မှ default သတ်မှတ်ပေးပါမယ်။ table name ကို plurals ပဲသုံးရပါမယ်။ ကျန်တဲ့ class variables တွေကတော့ table (model) ရဲ့ attributes တွေဖြစ်ပြီး db.Column class ရဲ့ instance အနေနဲ့ သတ်မှတ်ပေးရပါမယ်။

Flask-SQLAlchemy ရဲ့ model အားလုံးမှာ id လို့ အမည်ပေးထားတဲ့ primary key column တစ်ခုလည်း မဖြစ်မနေ သတ်မှတ်ပေးရပါမယ်။

နောက် model တိုင်းမှာ readable string representation အတွက် __repr__() method ပါရပါမယ်။ debugging and testing အတွက်လည်း အသုံးပြုနိုင်ဖို့ပါ။

Relational database ကိုသုံးထားတဲ့အတွက် relationships သုံးပြီး table rows တွေကြား connection တည်ဆောက်ထားလို့ရပါတယ်။ ဒါကြောင့် foreign key သုံးပြီး table နှစ်ခုရဲ့ row နှစ်ခုအတွင်း connection ဆောက်ပြီး relationships လုပ်ရပါမယ်။ ရေးထားတဲ့ code ကို ကြည့်ရအောင်ပါ။

```
class User(db.Model):
        id = db.Column(db.Integer, primary_key=True)
        username = db.Column(db.String(20), unique=True, nullable=False)
        email = db.Column(db.String(120), unique=True, nullable=False)
        image_file = db.Column(db.String(20), nullable=False, default='default.jpg')
        password = db.Column(db.String(60), nullable=False)
        posts = db.relationship('Post', backref='author', lazy=True)


    def __repr__(self):
        return f"User('{self.username}', '{self.email}', '{self.image_file}')"


class Post(db.Model):
        id = db.Column(db.Integer, primary_key=True)
        title = db.Column(db.String(100), nullable=False)
    date_posted = db.Column(db.DateTime, nullable=False,       default=datetime.utcnow)
        content = db.Column(db.Text, nullable=False)
        user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)


        def __repr__(self):
        return f"Post('{self.title}', '{self.date_posted}')"
```

အားလုံးနားလည်သွားရင် database နဲ့ operations စလုပ်ကြည့်ရအောင်ပါ။

Teminal ကနေ Python shell ကိုသွားပါမယ်။

```
>>> from flaskblog import db
>>> db.create_all()                    # to create table

>>> from flaskblog import User, Post
>>> user_1 = User(username="Hein Htet Oo", email="hho@gmail.com",

        password="hho2020")             # to create user
>>> db.session.add(user_1)          # to add database session

>>> user_2 = User(username="Wai Yan Tun", email="WYan@gmail.com",

        password="wai2020")
>>> db.session.add(user_2)

>>> db.session.commit()                # to save database session
>>> User.query.all()                   # to query all user
>>> User.query.first()                 # to query first user
>>> User.query.filter_by(username="Hein Htet Oo").all()
>>> User.query.filter_by(username="Hein Htet Oo").first()
>>> user = User.query.filter_by(username="Hein Htet Oo").first()

>>> user.id
>>> user = User.query.get(1)
>>> user.id
>>> user.posts

>>> post_1 = Post(title="Blog 1", content="First Post Content", user_id=user.id)
```

```
>>> post_2 = Post(title="Blog 2", content="Second Post Content", user_id=user.id)
>>> db.session.add(post_1)
>>> db.session.add(post_2)
>>> db.session.commit()

>>> user.posts
>>> for post in user.posts:
                print(post.title)



>>> post = Post.query.first()
>>> post
>>> post.user_id (post.id)
>>> post.author

>>> Clear all
>>> db.drop_all()                    # to delete tables
>>> db.create_all()
>>> User.query.all()
>>> Post.query.all()
```