11 – Blueprints and Configuration

Application package structure နဲ့ ပတ်သက်ပြီး၊ ယခင် သင်ခန်းစာ မှာ အတော်အသင့် လေ့လာပြီး structure ဆောက်ခဲ့ဘူးပါပြီ။ application package တွေဟာဆိုရင် application code အားလုံးနဲ့ templates, static files တွေကို လက်ခံပေးရတဲ့ နေရာတစ်ခုပါ။ application code အားလုံးကို application file တစ်ခုထဲမှာ ထည့်သွင်းထားမယ်ဆိုရင် အဆင်မပြေနိုင်သလို။ app ကို global scope မှာ ဖန်တီးပေးထားတာဟာလည်း configuration တွေကို dynamically ပြောင်းလဲဖို့ နည်းလမ်းမဟုတ်ပါဘူး။ script ကို run နေတဲ့အချိန် ရှိပြီးသား application instance ကို configuration ပြောင်းဖို့ နှောင့်နှေးပါလိမ့်မယ်။ unit test လုပ်ဖို့အတွက်တော့ ဒီ configruation ကို app run နေတဲ့အချိန်မှာ ပြောင်းလဲနိုင်တဲ့ အချက်ဟာ အရေးကြီးပါတယ်။ unit test တွေဟာ ရံဖန်ရံခါဆိုသလို အမျိုးမျိုးသော configuration settings တွေနဲ့ application ကို run ပေးဖို့ လိုအပ်လို့ပါ ။

အဲဒီပြဿနာ ဖြေရှင်းဖို့ နည်းလမ်းအနေနဲ့ application factory function တစ်ခုကို application package constructor ထဲမှာ တည်ဆောက်ပေးလိုက်ဖို့ပါပဲ။ အဲဒီအတွက် script ကို run နေတဲ့အချိန် configuration ကို settings ပြင်နိုင်သလို application instances များကို လိုသလို ဖန်တီးနိုင်စွမ်း ရှိပြီး testing အတွက်လည်း အသုံးတဲ့ပါတယ်။

နောက်ထည့်သွင်းစဉ်းစားရမဲ့ အချက်ကတော့ application instance က global scope မှာ ရှိနေတဲ့အတွက် routing မှာ app.route decorator သုံးပြီး၊ အလွယ်တကူ route လုပ်နိုင်ပေမဲ့ runtime မှာ application instance ကို create လုပ်ဖို့ နှောင့်နှေးကြန့်ကြာမှုနဲ့ အဆင်မပြေမှု ကို တွေ့ရပါမယ်။ အဲဒီအတွက် Flask ရဲ့ blueprints ကို သုံးပြီး ဖြေရှင်းကြပါမယ်။

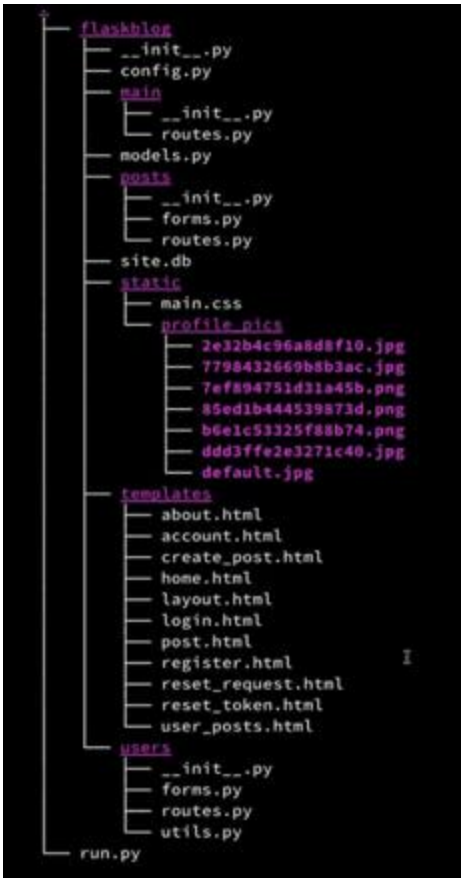
Blueprint ထဲမှာ routes နဲ့ error handlers တွေကို define လုပ်ပေးနိုင်တဲ့အတွက် application တစ်ခုနဲ့ အလားတူပါတယ်။ global scope မှာ blueprint ကို define လုပ် အသုံးပြုခြင်းဟာ application ရဲ့ routes နဲ့ error handlers တွေကို application တစ်ခုထဲမှာ အသုံးပြုတဲ့ နည်းလမ်းနဲ့ တူညီပါတယ်။

ဒါဆို blueprint ကို သုံးကြည့်ရအောင်။ ပြထားတဲ့ package structure အတိုင်း လိုအပ်တဲ့ folder တွေကို application package folder အောက်မှာ တည်ဆောက်ပေးပါမယ်။

၁. users

၂. Posts

၃. main



folder တွေက package folder တွေဖြစ်တဲ့အတွက် folder တစ်ခုချင်းစီမှာ __init__.py file တစ်ခုစီ ထည့်ပေးရပါမယ်။ နောက်ပြီး သက်ဆိုင်ရာ module files တွေ ထည့်ပါမယ်။ routes.py file ကို folder အသီးသီးမှာ ထည့်ပေးပါ ။ posts နဲ့ users folder တွေထဲမှာတော့ forms.py file ကို ထည့်ပါ ။ routes တွေမှာ ခေါ်သုံးဖို့ သီးခြားရေးထားတဲ့ functions တွေထားဖို့ အတွက် utils.py file ကိုတော့ users folder မှာ ထည့်ပေးပါမယ်။ လိုအပ်တဲ့ files နဲ့ folders တွေ တည်ဆောက်ပြီးရင် blueprints ဖန်တီးပြီး၊ ဆိုင်ရာ code တွေကို file တွေထဲကို ရွှေ့ထည့်ပါမယ်။

Blueprint class ကို flask module မှ import လုပ်ပါ။ နောက် blueprint obj ကို create လုပ်ပါ။

```
from flask import Blueprint
users = Blueprint('users', __name__)
```

Blueprint constructor မှာ arguments နှစ်ခု ထည့်ပေးရပါမယ်။ blueprint name (users) နဲ့ blueprint တည်ရှိတဲ့ module ဒါမဟုတ် package name ပါ ။ second argument ကို location ရှာဖို့ အဆင်ပြေဆုံး ဖြစ်တဲ့ __name__ ကို သုံးထားပါတယ်။ blueprint ဆောက်ပြီးရင် ဆိုင်ရာ code တွေကို ယခင် routes.py file မှ ရွှေ့ထည့်ပြီး၊ app.route နေရာတွေမှာ blueprint name တွေအစားထိုးပေးရပါမယ်။
code files တွေမှာ ကြည့်ပါ။ ပြီးရင် file အပောင်းတွေကို ဖျက်ပါ။

ပြီးရင် __init__.py file ကို လိုအပ်တဲ့ blueprints တွေကို import လုပ်ပြီး register လုပ်ရပါမယ်။ ပုံမှန်အားဖြင့် package constructor မှာ လက်ရှိအသုံးပြုနေတဲ့ Flask extensions အများစုကို import လုပ်ပေးထားတဲ့ အနေအထားပေါ့။ ဒါပေမယ့် application အတွက် အသုံးပြုချင်တဲ့အချိန်မှာမှ configuration ကို သတ်မှတ်ချင်လို့ application factory function ကို တည်ဆောက်ပေးမှာ ဖြစ်တဲ့အတွက် config.py file ကို ဖန်တီးကြည့်ရအောင်။ flaskblog package folder ထဲမှာ config.py အမည်နဲ့ file တစ်ခု create လုပ်ပါ။ ဖော်ပြထားတဲ့အတိုင်း config class ကို ရေးပါ ။

```
import os
class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY')
    SQLALCHEMY_DATABASE_URI = os.environ.get('SQLALCHEMY_DATABASE_URI')
    MAIL_SERVER = 'smtp.googlemail.com'
    MAIL_PORT = 587
    MAIL_USE_TLS = True
    MAIL_USERNAME = os.environ.get('EMAIL_USER')
    MAIL_PASSWORD = os.environ.get('EMAIL_PASS')
```

ပြီးရင် __init__.py file မှာ config class ကို import လုပ်ပြီး၊ create_app function ကို create လုပ်ပါ။ create_app() က application factory ဖြစ်ပြီး၊ application instance တစ်ခုကို return ပြန်ပေးပါမယ်။ နောက် init_app() method ကိုလည်း invoked လုပ်ပေးပါလိမ့်မယ်။

```
def create_app(config_class=Config):
    app = Flask(__name__)
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

```
    app.config.from_object(Config)

    db.init_app(app)
    bcrypt.init_app(app)
    login_manager.init_app(app)
    mail.init_app(app)
    from flaskblog.users.routes import users
    from flaskblog.posts.routes import posts
    from flaskblog.main.routes import main
    app.register_blueprint(users)
    app.register_blueprint(posts)
    app.register_blueprint(main)

    return app
```

အားလုံးသော code files တွေမှာ လိုအပ်တာတွေအားလုံး ပြင်ဆင် ရေးသားပြီးရင် run ကြည့်လို့ရပါပြီ။

__init__.py

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_bcrypt import Bcrypt
from flask_login import LoginManager
from flask_mail import Mail
from flaskblog.config import Config

db = SQLAlchemy()
```

```
bcrypt = Bcrypt()
login_manager = LoginManager()
login_manager.login_view = 'users.login'
login_manager.login_message_category = 'info'
mail = Mail()



def create_app(config_class=Config):
    app = Flask(__name__)
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
    app.config.from_object(Config)

    db.init_app(app)
    bcrypt.init_app(app)
    login_manager.init_app(app)
    mail.init_app(app)

    from flaskblog.users.routes import users
    from flaskblog.posts.routes import posts
    from flaskblog.main.routes import main
    app.register_blueprint(users)
    app.register_blueprint(posts)
    app.register_blueprint(main)

    return app
```

models.py

```
from datetime import datetime
from itsdangerous import TimedJSONWebSignatureSerializer as Serializer
from flask import current_app
from flaskblog import db, login_manager
from flask_login import UserMixin

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
class User(db.Model, UserMixin):
    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 get_reset_token(self, expires_sec=1800):
        s = Serializer(current_app.config['SECRET_KEY'], expires_sec)
        return s.dumps({'user_id': self.id}).decode('utf-8')

    @staticmethod
    def verify_reset_token(token):
        s = Serializer(current_app.config['SECRET_KEY'])
        try:
```

```
            user_id = s.loads(token)['user_id']
        except:
            return None
        return User.query.get(user_id)

    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}')"
```

config.py

```
import os
# set SQLALCHEMY_DATABASE_URI=sqlite:///site.db
```

```
# set SECRET_KEY=5791628bb0b13ce0c676dfde280ba245
# set EMAIL_USER=test@gmail.com
# set EMAIL_PASS=xxxxxxx

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY')
    SQLALCHEMY_DATABASE_URI = os.environ.get('SQLALCHEMY_DATABASE_URI')
    MAIL_SERVER = 'smtp.googlemail.com'
    MAIL_PORT = 587
    MAIL_USE_TLS = True
    MAIL_USERNAME = os.environ.get('EMAIL_USER')
    MAIL_PASSWORD = os.environ.get('EMAIL_PASS')
```

user/routes.py

```
from flask import render_template, url_for, flash, redirect, request, Blueprint
from flask_login import login_user, current_user, logout_user, login_required
from flaskblog import db, bcrypt
from flaskblog.models import User, Post
from flaskblog.users.forms import (RegistrationForm, LoginForm, UpdateAccountForm,
                                   RequestResetForm, ResetPasswordForm)
from flaskblog.users.utils import save_picture, send_reset_email

users = Blueprint('users', __name__)

@users.route("/register", methods=['GET', 'POST'])
```

```
def register():
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
    form = RegistrationForm()
    if form.validate_on_submit():
        hashed_password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(username=form.username.data, email=form.email.data,
password=hashed_password)
        db.session.add(user)
        db.session.commit()
        flash('Your account has been created! You are now able to log in', 'success')
        return redirect(url_for('users.login'))
    return render_template('register.html', title='Register', form=form)


@users.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user, remember=form.remember.data)
            next_page = request.args.get('next')
            return redirect(next_page) if next_page else redirect(url_for('main.home'))
```

```
        else:
            flash('Login Unsuccessful. Please check email and password', 'danger')
    return render_template('login.html', title='Login', form=form)

@users.route("/logout")
def logout():
    logout_user()
    return redirect(url_for('main.home'))

@users.route("/account", methods=['GET', 'POST'])
@login_required
def account():
    form = UpdateAccountForm()
    if form.validate_on_submit():
        if form.picture.data:
            picture_file = save_picture(form.picture.data)
            current_user.image_file = picture_file
        current_user.username = form.username.data
        current_user.email = form.email.data
        db.session.commit()
        flash('Your account has been updated!', 'success')
        return redirect(url_for('users.account'))
    elif request.method == 'GET':
        form.username.data = current_user.username
        form.email.data = current_user.email
    image_file = url_for('static', filename='profile_pics/' + current_user.image_file)
    return render_template('account.html', title='Account',
```

```
                        image_file=image_file, form=form)

@users.route("/user/<string:username>")
def user_posts(username):
    page = request.args.get('page', 1, type=int)
    user = User.query.filter_by(username=username).first_or_404()
    posts = Post.query.filter_by(author=user)\
        .order_by(Post.date_posted.desc())\
        .paginate(page=page, per_page=5)
    return render_template('user_posts.html', posts=posts, user=user)


@users.route("/reset_password", methods=['GET', 'POST'])
def reset_request():
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
    form = RequestResetForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        send_reset_email(user)
        flash('An email has been sent with instructions to reset your password.', 'info')
        return redirect(url_for('users.login'))
    return render_template('reset_request.html', title='Reset Password', form=form)


@users.route("/reset_password/<token>", methods=['GET', 'POST'])
def reset_token(token):
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
```

```
    user = User.verify_reset_token(token)
    if user is None:
        flash('That is an invalid or expired token', 'warning')
        return redirect(url_for('users.reset_request'))
    form = ResetPasswordForm()
    if form.validate_on_submit():
        hashed_password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user.password = hashed_password
        db.session.commit()
        flash('Your password has been updated! You are now able to log in', 'success')
        return redirect(url_for('users.login'))
    return render_template('reset_token.html', title='Reset Password', form=form)
```

user/forms.py

```
from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileAllowed
from wtforms import StringField, PasswordField, SubmitField, BooleanField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError
from flask_login import current_user
from flaskblog.models import User

class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=2,
max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])
```

```
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                                     validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError('That username is taken. Please choose a different one.')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('That email is taken. Please choose a different one.')



class LoginForm(FlaskForm):
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember = BooleanField('Remember Me')
    submit = SubmitField('Login')

class UpdateAccountForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=2,
max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    picture = FileField('Update Profile Picture', validators=[FileAllowed(['jpg', 'png'])])
```

```
    submit = SubmitField('Update')

    def validate_username(self, username):
        if username.data != current_user.username:
            user = User.query.filter_by(username=username.data).first()
            if user:
                raise ValidationError('That username is taken. Please choose a different one.')

    def validate_email(self, email):
        if email.data != current_user.email:
            user = User.query.filter_by(email=email.data).first()
            if user:
                raise ValidationError('That email is taken. Please choose a different one.')




class RequestResetForm(FlaskForm):
    email = StringField('Email', validators=[DataRequired(), Email()])
    submit = SubmitField('Request Password Reset')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user is None:
            raise ValidationError('There is no account with that email. You must register first.')
```

```
class ResetPasswordForm(FlaskForm):
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                                     validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Reset Password')
```

user/utils.py

```
import os
import secrets
from PIL import Image
from flask import url_for, current_app
from flask_mail import Message
from flaskblog import mail

def save_picture(form_picture):
    random_hex = secrets.token_hex(8)
    _, f_ext = os.path.splitext(form_picture.filename)
    picture_fn = random_hex + f_ext
    picture_path = os.path.join(current_app.root_path, 'static/profile_pics', picture_fn)


    output_size = (125, 125)
    i = Image.open(form_picture)
    i.thumbnail(output_size)
    i.save(picture_path)
    return picture_fn
```

```
def send_reset_email(user):
    token = user.get_reset_token()
    msg = Message('Password Reset Request',
                  sender='test@gmail.com',
                  recipients=[user.email])
    msg.body = f'''To reset your password, visit the following link:
{url_for('users.reset_token', token=token, _external=True)}
If you did not make this request then simply ignore this email and no changes will be
made.
'''
    mail.send(msg)
```

post/routes.py

```
from flask import (render_template, url_for, flash, redirect, request, abort, Blueprint)
from flask_login import current_user, login_required
from flaskblog import db
from flaskblog.models import Post
```

```
from flaskblog.posts.forms import PostForm

posts = Blueprint('posts', __name__)

@posts.route("/post/new", methods=['GET', 'POST'])
@login_required
def new_post():
    form = PostForm()
    if form.validate_on_submit():
        post = Post(title=form.title.data, content=form.content.data, author=current_user)
        db.session.add(post)
        db.session.commit()
        flash('Your post has been created!', 'success')
        return redirect(url_for('main.home'))
    return render_template('create_post.html', title='New Post',
                           form=form, legend='New Post')

@posts.route("/post/<int:post_id>")
def post(post_id):
    post = Post.query.get_or_404(post_id)
    return render_template('post.html', title=post.title, post=post)

@posts.route("/post/<int:post_id>/update", methods=['GET', 'POST'])
@login_required
def update_post(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
```

```
        abort(403)
    form = PostForm()
    if form.validate_on_submit():
        post.title = form.title.data
        post.content = form.content.data
        db.session.commit()
        flash('Your post has been updated!', 'success')
        return redirect(url_for('posts.post', post_id=post.id))
    elif request.method == 'GET':
        form.title.data = post.title
        form.content.data = post.content
    return render_template('create_post.html', title='Update Post',
                           form=form, legend='Update Post')


@posts.route("/post/<int:post_id>/delete", methods=['POST'])
@login_required
def delete_post(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        abort(403)
    db.session.delete(post)
    db.session.commit()
    flash('Your post has been deleted!', 'success')
    return redirect(url_for('main.home'))
```

post/forms.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField, TextAreaField
from wtforms.validators import DataRequired

class PostForm(FlaskForm):
    title = StringField('Title', validators=[DataRequired()])
    content = TextAreaField('Content', validators=[DataRequired()])
    submit = SubmitField('Post')
```

main/routes.py

```
from flask import render_template, request, Blueprint
from flaskblog.models import Post

main = Blueprint('main', __name__)

@main.route("/")
@main.route("/home")
def home():
    page = request.args.get('page', 1, type=int)
    posts = Post.query.order_by(Post.date_posted.desc()).paginate(page=page, per_page=5)
    return render_template('home.html', posts=posts)

@main.route("/about")
def about():
    return render_template('about.html', title='About')
```