02 – Templates

Web browser လိုမျိုး clients ကနေ web server ကို requests ပို့လိုက်ပြီဆိုတာနဲ့ server ကနေတဆင့် flask application instance စီသို့ requests ကို ထပ်ဆင့်ပို့လွှတ်ပါတယ်။ application instance အနေနဲ့ သက်ဆိုင်ရာ URL request တစ်ခုစီအတွက် ဘာလုပ်ရမယ် (ဘယ် code run ရမယ်) ဆိုတာ သိဖို့ လိုပြီး အဲဒီ URL နဲ့ သက်ဆိုင်ရာ view function ကို map လုပ်ပေးထားရပါမယ်။

View funtion ရဲ့ အထင်ရှားဆုံးအလုပ်ကတော့ request ကို response လုပ်ပေးတာပါ။ request က application ရဲ့ အနေအထားကို ပြောင်းဖို့ trigger လုပ်လိုက်ပြီဆိုတာနဲ့ view function က အပြောင်းအလဲကို generate လုပ်ပေးရပါတယ်။ ဒီနေရာမှာ view function က business logic နဲ့ presentation logic ဆိုတဲ့ လုပ်ငန်းနှစ်မျိုးကို အသီးသီး လုပ်ပေးရတာပါ။

ဥပမာ – website ကို account အသစ်တစ်ခု register လုပ်တယ်ဆိုပါတော့၊ view function က ဘာလုပ်ပေးရမလဲဆိုရင် database ထဲကို အချက်အလက် ပေါင်းထည့်ဖို့ ပြောရမယ်။ နောက် success ဖြစ်မဖြစ်ကို browser မှာ message ပြပေးရမယ်။ အဲဒီ နောက်ကွယ်ကလုပ်ငန်းနဲ့ ရှေ့ထွက်လုပ်ငန်းနှစ်ခုကို ပေါင်းလုပ်မယ်ဆိုရင် code တွေဟာ နားလည်ရခက်ခဲပြီး၊ maintain လုပ်ရတာလဲ မလွယ်ပါဘူး။

ဒီလုပ်ငန်းနှစ်ခုထဲက presentation logic ကို template အဖြစ် လွှဲပြောင်းပေးခြင်းအားဖြင့် application ရဲ့ maintainability ကို တိုးတက်စေပါတယ်။

Template ဆိုတာကတော့ dynamic parts တွေအတွက် variables တွေနဲ့ response text တွေ ပါဝင်တဲ့ file တစ်ခုပဲ ဖြစ်ပါတယ်။ variables ထဲကို actual values တွေအစားထိုးပြီး၊ နောက်ဆုံးပိတ် response string တွေပြန်ပေးတဲ့ လုပ်ငန်းစဉ်ကတော့ rendering ပါ။ rendering templates လုပ်နိုင်ဖို့အတွက် Flask မှာ jinja2 ဆိုတဲ့စွမ်းအားမြင့် template engine ကို အသုံးပြုပါတယ်။

အဲဒါကြောင့် Templating language ဖြစ်တဲ့ Jinja2 ကိုသုံးပြီး Templates တွေနဲ့ application ရေးကြည့်ပါမယ်။

flaskblog.py file ရဲ့ ပထမဆုံး route ကို ကြည့်ရအောင်။

```
@app.route("/")
    def home():
                return "<h1>Home Page</h1>"
```

View function ကနေ HTML content နဲ့ return ပြန်ခဲ့ပါတယ်။ အဲဒီမှာ content အတွက် HTML code ကို အပြည့်အစုံရေးမထားပါဘူး။ ရေးချင်ရင်လည်း ရပါတယ်။

ဥပမာ –

```
return ''' <!DOCTYPE html>
                <html>
                ....
                </html>
            '''
```

ဒါပေမဲ့ အဲလို return မှာရေးမဲ့အစား HTML files တွေကို သီးခြားခွဲထုတ်ပြီး template files တွေအနေနဲ့ ဖန်တီးပါမယ်။

Template files တွေက တစ်ကယ်တမ်းမတော့ HTML content files တွေပါ။ Templete files တွေကို application folder အောက်မှာ templetes အမည်နဲ့ folder ဆောက်ပြီး သီးခြားထားပေးရပါတယ်။ ဘာကြောင့်လဲဆိုတော့ Flask ကလိုအပ်တဲ့ templates တွေကို main folder ရဲ့အောက်က templates subfolderမှာချလို့ပါ။

Templates တွေကို return ပြန်တဲ့အခါ render_template function ရဲ့ first argument မှာ filename ပေးပြီး return ပြန်ရပါတယ်။ ရေးပြပေးထားတဲ့ code files တွေကို ကြည့်ရအောင်။ ပထမ home() က home.html နဲ့ ဒုတိယ about() က about.html ကို render လုပ်ပေးပါတယ်။ ဒါကြောင့်လိုအပ်နေတဲ့ home.html နဲ့ about.html files တွေကို templates folder မှာ ဖန်တီးပေးထားရမှာပါ။ files အားလုံး ရေးပြီးရင်တော့ run ကြည့်ပါမယ်။

flaskblog.py

```
from flask import Flask, render_template
app = Flask(__name__)
```

```
@app.route("/")
def home():
    return render_template('home.html')

@app.route("/about")
def about():
    return render_template('about.html')

if __name__ == '__main__':
    app.run(debug=1)
```

templates/home.html

```
<!DOCTYPE html>
<html>
<head>
        <title><title>
</head>
<body>
        <h1>Home Page</h1>
</body>
</html>
```

templates/about.html

```
<!DOCTYPE html>
<html>
<head>
        <title><title>
</head>
<body>
        <h1>About Page</h1>
</body>
</html>
```

Templates ရဲ့ response text မှာ dynamic component တွေ ပါလာမယ်ဆိုရင် Jinja2 ရဲ့ variable block {{ }} နဲ့ response ကို အကောင်အထည်ဖော်ပေးရပါမယ်။

Templates တွေမှာ reference လုပ်မဲ့ variables တွေအတွက် actual values တွေ ဖော်ပြချင်ရင်တော့ key–value pairs ပုံစံနဲ့ render_template function မှာ arguments တွေ ထပ်ထည့်နိုင်ပါတယ်။ flaskblog.py မှာ posts ဆိုတဲ့ dummy data ထည့်ပြီး templates မှာ variable block {{ }} နဲ့ reference လုပ်သုံးကြည့်ပါမယ်။ file မှာ ကြည့်ပါ။

flaskblog.py

```
from flask import Flask, render_template
app = Flask(__name__)

posts = [# list of dicts (dummy data)
        {
                'author': 'Hein Htet Oo',
                'title': 'Blog Post 1',
                'content': 'First Blog Post',
                'date_posted': 'April 26, 2020'
    },
    {
                'author': 'Wai Yan Tun',
                'title': 'Blog Post 2',
                'content': 'Second Blog Post',
                'date_posted': 'April 27, 2020'
    }
]

@app.route("/")
def home():
    return render_template('home.html', posts=posts)
@app.route("/about")
def about():
    return render_template('about.html', title='About')
if __name__ == '__main__':
    app.run(debug=1)
```

render_template function ရဲ့ argument (posts = posts) မှာ ဘယ်ဘက်က posts က argument name ဖြစ်ပြီး၊ ညာဘက်က posts ကတော့ variable ပါ။ Jinja2 ရဲ့ variable block {{}} နဲ့ templates မှာ ဒီ variable ကို ယူသုံးပါမယ်။ file ကို ကြည့်လိုက်ပါ။

templates/home.html

```
<!DOCTYPE html>
<html>
<head>
	<title><title>
</head>
<body>
	{% for post in posts %}
		<h1>{{ post.title }}</h1>
		<p>By {{ post.author }} on {{ post.date_posted }}</p>
		<p>{{ post.content }}</p>
	{% endfor %}
</body>
</html>
```

posts တွေက တစ်ခုထက်မကနိုင်တဲ့အတွက် post တစ်ခုစီကို loop ပတ်ပြီး ယူသုံးပါမယ်။ ဒါကြောင့် Jinja2 ရဲ့ control block {% %}တွေထဲက loops ကို သုံးထားပါတယ်။ အဲလိုပဲ conditional statement ကိုလည်း သုံးကြည့်ပါမယ်။

title argument နဲ့ title ပါလာရင်ပေးထားတဲ့ title ကိုသုံးပြီး၊ ပေးမထားရင်default title သုံးဖို့ ရေးကြည့်ပါမယ်။ file ကိုကြည့်ပါ။

templates/about.html

```
<!DOCTYPE html>
<html>
<head>
```

```
        {% if title %}
        <title>Flask Blog – {{ title }}<title>
        {% else %}
                <title>Flask Blog</title>
        {% endif %}
</head>
<body>
        <h1>About Page</h1>
</body>
</html>
```

Templete file တွေမှာ အားလုံး တူညီတဲ့ အပိုင်းတချို့ (header and footer) တွေပါတဲ့အတွက် code မထပ်ရအောင် templete inheritence ကို သုံးပါမယ်။ အဲဒီအတွက် base template နဲ့ child templates တွေ ဖန်တီးပေးရပါမယ်။ Base template ထဲမှာ derived templates တွေက override လုပ်နိုင်ဖို့ Jinja2 block and endblock တွေ သတ်မှတ်ပေးထားရပါမယ်။

ဥပမာ – {% block content %}{% endblock content %}

{% block title %}{% endblock %}

အစရှိသဖြင့်ပေါ့။

base template ကိုတော့ အများအားဖြင့် layout.html or base.html or home.html စသဖြင့် အမည်ပေးလေ့ရှိပါတယ်။ ရေးကြည့်ရအောင်။

templates/layout.html

```
<!DOCTYPE html>
<html>
<head>
        <meta charset='utf-8'>
```

```
        {% if title %}
        <title>Flask Blog – {{ title }}<title>
        {% else %}
                <title>Flask Blog</title>
        {% endif %}
</head>
<body>
        {% block content %} {% endblock %}
</body>
</html>
```

templates/home.html

```
{% extends "layout.html" %}
{% block content %}
        {% for post in posts %}
                <h1>{{ post.title }}</h1>
                <p>By {{ post.author }} on {{ post.date_posted }}</p>
                <p>{{ post.content }}</p>
        {% endfor %}
{% endblock content %}
```

templates/about.html

```
{% extends "layout.html" %}
{% block content %}
        <h1>About Page</h1>
{% endblock content %}
```

layout.html မှ drive လုပ်ဖို့ templates တွေမှာ extends directive ကို declare လုပ်ပေးရပါမယ်။