Flask로 만드는 블로그

공동공부 (2명)

생산자

a476548

토픽 5 / 봤어요 0

템플릿

2018-08-07 02:50:29

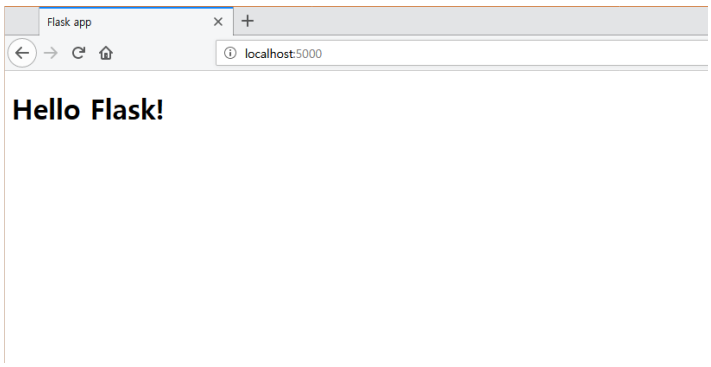
이전에 우리는 간단한 Flask 어플리케이션을 만들어보았습니다.

이 어플리케이션은 단순히 `Hello Flask!`만 출력하지만, 웹 브라우저는 HTML을 읽어 해석해서 사용자에게 보여주기 때문에 다양한 기능을 추가하기 위해서는 모든 컨텐츠를 HTML의 형태로 만들어서 전달해야 합니다.

HTML을 보여주기 위해 app.py를 다음과 같이 수정합니다.

```
from flsk import Flask

App = Flask(__name__)

@app.route('/')
def index():
  return '''<!DOCTYPE HTML><html>
  <head>
    <title>Flask app</title>
  </head>
  <body>
    <h1>Hello Flask!</h1>
  </body>
</html>'''

@app.route('/about')
def about():
    return 'About 페이지'
```

웹 브라우저에서 인덱스 페이지를 들어가보면 다음과 같이 나타납니다.



하지만 이렇게 긴 HTML 코드 블럭을 어플리케이션 코드 내에 삽입한다는 것은 가독성이 너무 떨어져서 유지보수하기 어렵게 만듭니다. 어플리케이션의 규모가 커지면서 점점 더 많은 기능들을 제공하게 되고, 점점 더 많은 HTML을 처리해야 하는데 이를 모두 하나의 코드에 삽입한다면 관리하기 어렵게 됩니다.

따라서 구조적으로 사용자에게 보여질 부분(HTML)과 실제로 처리하는 부분(어플리케이션 코드)을 나눌 필요가 있습니다.

템플릿

Flask에서는 보여지는 부분과 처리하는 부분을 나누기 위해 템플릿이라는 기능을 제공합니다.

템플릿에 사용되는 파일들은 `templates` 디렉터리에 저장되며 일반적으로 .html 파일을 사용합니다. 또한 css 같은 파일들은 `static` 디렉터리에 저장합니다. 어플리케이션 상에서 이러한 html 파일들을 렌더링할 수 있도록 Flask에서는 `render_template`를 제공하는데요. Jinja2 템플릿 엔진을 사용해서 html 문서 내에 코드 조각들을 삽입하여 웹 페이지를 동적으로 생성할 수 있습니다.

템플릿 기능을 사용하기 위해 먼저 프로젝트 디렉터리 하위에 `templates`과 `static`디렉터리를 생성합니다.

```
flask_blog/
├── templates/
├── static/
├── venv/
└── app.py
```

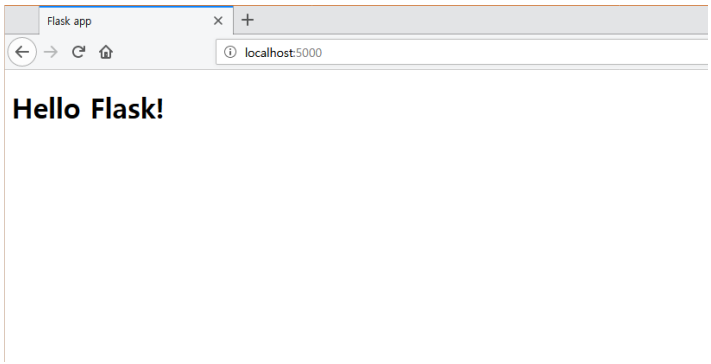
템플릿 디렉터리 하위에 `index.html`을 만들어 위에서 작성했던 html 코드를 그대로 가져옵니다.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Flask app</title>
  </head>
  <body>
    <h1>Hello Flask!</h1>
  </body>
</html>
```

`index.html`을 렌더링 하기 위해서 `app.py`파일을 수정합니다. html 문서가 분리된 파일로 이동했기 때문에 코드는 훨씬 간결하게 됩니다.

```
from flask import Flask, render_template

App = Flask(__name__)

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

@app.route('/about')
def about():
    return 'About 페이지'
```

웹 브라우저에서 인덱스 페이지를 접속합니다.



브라우저에서의 결과는 이전과 동일하지만 코드를 보기가 훨씬 깔끔한 것을 알 수 있습니다.

또한 Flask 템플릿은 강력한 기능들이 있는데 그 중 하나가 바로 계층 구조를 지원한다는 점입니다.

하나의 웹사이트는 일반적으로 동일한 테마를 갖습니다. 각각의 페이지는 동일한 구성요소를 갖는데 이를 다
일에 삽입하게 되면 구성요소를 변경할 때, 모든 파일에 수정사항을 반영해야 합니다. 이번 프로젝트에서는 사

사용되는 동일한 구성요소는 `layout.html`파일에 넣어두고, 나머지 페이지들은 이를 상속 받아서 사용하도록 하겠습니다.

이를 통해 `layout.html` 파일 하나를 변경함으로써 전 페이지에 변경사항을 적용할 수 있습니다.

템플릿 디렉터리 하위에 `layout.html` 파일을 만듭니다.

```
<!DOCTYPE HTML>
<html lang="kr">
  <head>
    <meta charset="utf-8">

    <title>Flask 블로그</title>
  </head>

  <body>
    <header>
      <nav>
        <a href="/">Flask 블로그</a>
        <a href="/">Home</a>
        <a href="/about">About</a>
      </nav>
      <hr>
    </header>

    <main role="main">
    <!-- Main content block -->
    {% block content %}{% endblock %}
    </main>
  </body>
</html>
```

Jinja2 템플릿 엔진은 구문(Statement)의 경우 `{% %}`로, 표현(Expression)은 `{{ }}`로 감싸서 표시합니다. 주석의 경우에는 `{# #}`을 이용해 작성할 수 있습니다.

파일을 보면 `{% block <이름> %}` 태그가 보이는데 이 태그는 상속받은 템플릿에서 사용할 수 있는 공간을 정의합니다.

상속 받은 템플릿에서 부모의 block 태그와 동일한 이름의 block 태그를 사용하면 템플릿이 렌더링 될 때, 부모의 block 태그는 상속 받은 템플릿에서 작성한 코드로 대치됩니다.

이제 나머지 모든 템플릿은 `layout.html`을 상속받아 사용할 것입니다. 그렇게 되면 모든 템플릿이 `layout.html`에 정의되어 있는 요소들 상속받게 됩니다.

먼저 `index.html` 파일을 다음과 같이 수정하도록 하겠습니다.

```
{% extends 'layout.html' %}

{% block content %}
<div>
  <div>
    <article>
      <h1>첫 번째 포스트 제목</h1>
      <p>첫 번째 포스트 내용입니다.</p>
      <div>
        <p>작성자</p>
        <p>2018-08-01</p>
      </div>
    </article>
    <article>
      <h1>두 번째 포스트 제목</h1>
```

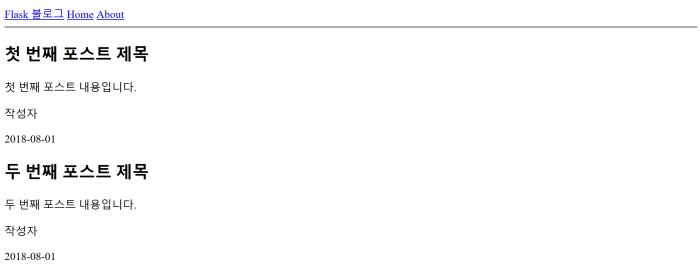
```
        <p>두 번째 포스트 내용입니다.</p>
        <div>
          <p>작성자</p>
          <p>2018-08-01</p>
        </div>
      </article>
    </div>
</div>
{% endblock %}
```

`index.html` 파일을 보면 {% extends 'layout.html' %}` 태그가 보입니다.

이는 우리가 작성했던 `layout.html` 파일을 상속 받겠다는 의미입니다.

그리고 `{% block content %} {% endblock %}`사이에 우리가 페이지에 표현할 내용들을 작성했습니다.

이 상태로 파일을 저장하고 웹 사이트를 다시 열어 봅니다.



이와 동일하게 about 페이지도 한 번 만들어 보겠습니다.

템플릿 디렉터리 하위에 `about.html` 파일을 만들고 다음과 같은 코드를 작성합니다.

```
{% extends 'layout.html' %}

{% block content %}
<div>
  <h1>About</h1>
</div>
{% endblock %}
```

파일들을 다 만들면 디렉터리 구조는 다음과 같습니다.

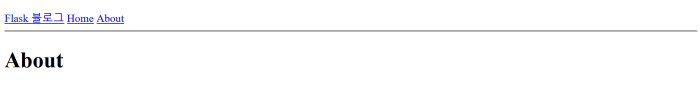
```
flask_blog/
├── templates/
│   ├── about.html
│   ├── index.html
│   └── layout.html
├── static/
├── venv/
└── app.py
```

about 페이지가 `about.html`를 렌더링해서 보여주기 위해 `app.py` 파일의 `about()` 함수를 변경합니다.

```
from flask import Flask, render_template

app = Flask(__name__)

```

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

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

파일을 수정한 다음 웹 브라우저에서 확인하면 다음과 같습니다.



현재까지 작성된 전체 코드는 Flask blog tutorial Chapter2-#1에서 확인할 수 있습니다.

템플릿에 파라미터 전송

제목 동적 생성

`layout.html`을 보면 `<title>`이 고정값입니다. 이를 페이지마다 다르게 지정하기 위해 `{{ title }}`변수를 사용하고 페이지마다 이를 넘겨주도록 하겠습니다.

`layout.html`에 `<title>`태그가 있는 곳을 다음과 같이 변경합니다.

```
<!DOCTYPE HTML>
<html lang="kr">
  <head>
    <meta charset="utf-8">

    {% if title %}
      <title>Flask 블로그 - {{ title }}</title>
    {% else %}
      <title>Flask 블로그</title>
    {% endif %}
  </head>

  <body>
    <header>
      <nav>
        <a href="/">Flask 블로그</a>
        <a href="/">Home</a>
        <a href="/about">About</a>
      </nav>
      <hr>
    </header>
```

```
22
23      <main role="main">
24      <!-- Main content block -->
25      {% block content %}{% endblock %}
26      </main>
27    </body>
28  </html>
```

jinja2에서는 다음과 같은 형태로 조건문을 작성할 수 있습니다.

```
{% if "조건문1" %}
  조건문1이 참인 경우 실행되는 문장
{% elif "조건문2" %}
  조건문2이 참인 경우 실행되는 문장
{% else %}
  조건식이 만족하지 않는 경우 실행되는 문장
{% endif %}
```

이를 통해 `title`에 값이 들어오면 웹 사이트의 제목은 `Flask 블로그 - {{ title }}`이 되고 그렇지 않으면 그냥 `Flask 블로그`만 출력하게 됩니다.

템플릿에 파라미터를 전달하려면 `render_template` 함수에 템플릿에 정의된 변수 이름과 동일한 이름을 넘겨주면 됩니다.

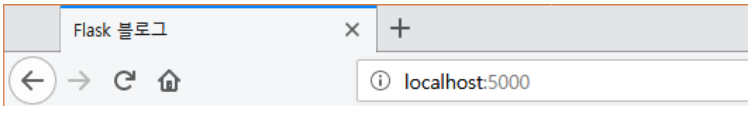
메인 페이지는 제목을 그대로 두고 About 페이지만 제목을 변경해보겠습니다.

`app.py`의 `/about` 라우트 함수를 수정합니다.

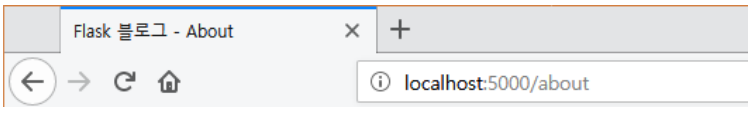
```
from flask import Flask, render_template

app = Flask(__name__)

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

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

웹 브라우저 상에서도 제목이 페이지에 따라 달라집니다.

- index 페이지



- about 페이지



게시물 동적 생성

`index.html` 파일을 보면 게시물 정보가 하드코드 되어 있는 것을 확인할 수 있습니다.
실제 웹 페이지는 db에서 가져온 포스트를 기반으로 보여줘야 하기 때문에 `app.py`에서 포스트에 대한 정보를 넘겨줄 수 있도록 작성하겠습니다.

게시물은 개수가 얼마든지 가능하기 때문에 템플릿상에서 동적으로 생성할 수 있도록 작성해야 합니다.
따라서 반복문을 사용하도록 하겠습니다.

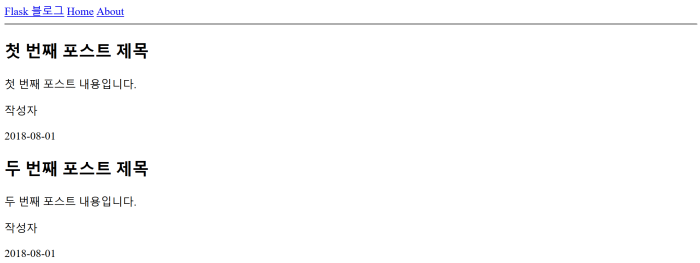
하드코드된 `<article>` 블럭을 다음과 같이 수정합니다.

```
{% extends 'layout.html' %}

{% block content %}
<div>
  <div>
    {% for post in posts %}
      <article>
        <h1>{{ post.title }}</h1>
        <p>{{ post.content}}</p>
        <div>
          <p>{{ post.author.username }}</p>
          <p>{{ post.date_posted.strftime('%Y-%m-%d') }}</p>
        </div>
      </article>
    {% endfor %}
  </div>
</div>
{% endblock %}
```

현재는 db가 구성되지 않았으므로 테스트를 위해 더미 데이터를 만들고 이를 템플릿에 넘겨주겠습니다. `app.py` 파일을 열고 다음과 같이 수정합니다.

```
from datetime import datetime

from flask import Flask, render_template

app = Flask(__name__)

posts = [
    {
        'author': {
            'username': 'test-user'
        },
        'title': '첫 번째 포스트',
        'content': '첫 번째 포스트 내용입니다.',
        'date_posted': datetime.strptime('2018-08-01', '%Y-%m-%d')
    },
    {
        'author': {
            'username': 'test-user'
        },
        'title': '두 번째 포스트',
        'content': '두 번째 포스트 내용입니다.',
        'date_posted': datetime.strptime('2018-08-03', '%Y-%m-%d')
    },
]

@app.route('/')
@app.route('/index')
def index():
    return render_template('index.html', posts=posts)

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

웹 브라우저를 이용해 홈페이지에 접속하면 아까와 동일한 페이지를 확인할 수 있습니다.
하지만 템플릿들이 모두 동적으로 생성되어서 나타나므로 나중에 데이터가 추가되더라도 템플릿은 변경할 필요가 없습니다.


Flask 블로그 Home About

첫 번째 포스트 제목

첫 번째 포스트 내용입니다.

작성자

2018-08-01

두 번째 포스트 제목

두 번째 포스트 내용입니다.

작성자

2018-08-01

URL 동적 생성

html파일을 보면 `href` 속성이 `/`, `/about`과 같이 하드코드 되어 있는 것을 확인할 수 있습니다. 지금이야 이러한 속성들이 문제되지 않지만 `app.py`에서 라우트가 변경되거나 전체 프로젝트가 특정 홈페이지의 하위로 이동하는 경우 URL이 달라질 수 있어 제대로 된 URL을 가르키지 못할 수 있습니다.

따라서 이를 동적으로 생성할 수 있도록 변경해야 하는데 flask에서는 `url_for`라는 함수로 지원합니다. 기본적으로 `url_for`함수는 엔드포인트 함수명을 인자로 받습니다. 따라서 `/` 주소를 생성하고 싶다면 app.py에서 `/` 라우트에 대한 함수 이름을 `index`로 정의했으므로 `url_for('index')`를 사용해야 합니다.

만약 `/about` 주소를 생성하고 싶다면 `url_for('about')`을 사용해야 합니다.

또한 `url_for`함수로 `static` 폴더 내에 있는 리소스의 주소를 생성할 수도 있는데, 이는 `url_for('static', filename=<파일 이름>)`를 이용합니다.

이제 layout.html 파일에서 하드 코드 된 url들을 모두 `url_for` 함수를 이용해 동적생성 하도록 변경하겠습니다.

```
<!DOCTYPE HTML>
<html lang="kr">
  <head>
    <meta charset="utf-8">

    {% if title %}
      <title>Flask 블로그 - {{ title }}</title>
    {% else %}
      <title>Flask 블로그</title>
    {% endif %}
  </head>

  <body>
    <header>
      <nav>
        <a href="{{ url_for('index') }}">Flask 블로그</a>
        <a href="{{ url_for('index') }}">Home</a>
        <a href="{{ url_for('about') }}">About</a>
      </nav>
      <hr>
    </header>

    <main role="main">
    <!-- Main content block -->
```

```
25      {% block content %}{% endblock %}
26      </main>
27    </body>
28  </html>
```

현재까지 작성된 전체 코드는 Flask blog tutorial Chapter2-#2에서 확인할 수 있습니다.

봤어요 (2명) 이전 다음

댓글을 작성하려면 로그인하셔야 합니다.

학습이 1년 전

app.py 첫번째행 flsk 오타 있습니다. 잘 보고 있어요. 감사합니다

모바일 버전