

Dynamic Model을 위한 Flask Web App 개발

Samkeun Kim <skim@hknu.ac.kr>

<http://cyber.hknu.ac.kr/>

Dynamic Model을 위한 Flask Web App 개발

SavedModel => 계산 그래프 저장!!

- 오로지 TensorFlow 오퍼레이션에 기반한 모델에만 사용 가능

Dynamic tf.keras 모델 => 계산 그래프로 변환될 수 없다!!

- Dynamic 모델 => 다른 툴(예: **Flask**)을 이용하여 서비스 되어야

Installation

Flask

- "마이크로 프레임워크"라고 불리울 정도로 아주 작은 프레임워크
- 확장 가능하게 설계된 프레임워크
- 3가지 주요 종속 모듈(dependencies):
 - ✓ The routing, debugging, and Web Server Gateway Interface (WSGI) subsystems come from [Werkzeug](#);
 - ✓ the template support is provided by [Jinja2](#); and
 - ✓ the command-line integration comes from [Click](#).

These dependencies are all authored by Armin Ronacher, the author of Flask.

나머지 모듈들은 사용자가 자신의 프로젝트에 최상으로 적합한 모듈들을 골라 사용할 수 있다!

Creating the Application Directory

Anaconda Prompt: 18-1

```
(base) mkdir flasky  
(base) cd flasky  
# using private Python interpreter  
(base) activate aisam  
(aisam) pip install flask  
(aisam) pip freeze > requirements.txt
```

```
(aisam) E:\practice\wait\handson(pycharm)\flasky>pip freeze  
absl-py==0.9.0  
asn1crypto==1.3.0  
astor==0.8.0  
attrs==19.3.0  
backcall==0.1.0  
bleach==3.1.0  
blinker==1.4  
cachetools==3.1.1  
certifi==2019.11.28  
cffi==1.14.0  
chardet==3.0.4  
Click==7.0  
colorama==0.4.3  
cryptography==2.8  
cycler==0.10.0  
decorator==4.4.1
```

참조: 제 16.1강 Creating a Flask Project in PyCharm.pdf

Basic Application Structure

첫 번째 Flask 웹 애플리케이션을 만들어 보자!

Initialization

모든 Flask 애플리케이션은 **애플리케이션 인스턴스**를 만들어야 한다.

웹 서버는 WSGI(Web Server Gateway Interface)("wiz-ghee"라고 발음)라는 프로토콜을 사용하여 클라이언트로부터 수신한 모든 요청을 이 애플리케이션 인스턴스로 전달한다.

애플리케이션 인스턴스는 Flask 클래스의 객체이며 일반적으로 다음과 같이 생성한다:

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

- Flask 클래스 생성자의 **__name__** 인자: 애플리케이션의 메인 모듈 또는 패키지의 이름
- 애플리케이션의 위치를 결정하는데 사용
- 기타 파일들(예: 이미지)을 찾을 수 있게 해 줌

Routes and View Functions

클라이언트(웹 브라우저) => 웹 서버에 요청을 보냄 => 웹 서버는 요청을 Flask 애플리케이션 인스턴스에 보냄

⇒ Flask 애플리케이션 인스턴스는 각 요청(URL)에 대해 어떤 코드를 실행해야 할 지 알아야 하기 때문에 URL과 Python 함수를 매핑

⇒ URL과 이를 처리하는 함수 간의 연결을 **라우트(route)**라고 함

Flask 애플리케이션에서 라우트를 정의하는 가장 편리한 방법:

⇒ 애플리케이션 인스턴스가 노출시켜 주는 **app.route** 데코레이터를 사용하는 것

⇒ 데코레이터를 사용하여 라우트를 선언하는 예:

```
@app.route('/')  
def index():  
    return '<h1>Hello World!</h1>'
```

애플리케이션의 루트 URL을 위한 핸들러로서 index() 함수를 등록해 준다!

애플리케이션 URL을 처리하는 `index()`와 같은 함수를 뷰(view) 함수라고 한다:

- 애플리케이션이 `www.example.com` 도메인 이름의 서버에 배포된 경우 브라우저에서 `http://www.example.com/`으로 요청하면 서버에서 `index()`가 실행됨
- 이 뷰 함수의 반환 값은 클라이언트가 받게 되는 응답(***response***)이 됨
- ***Response*** 는 클라이언트의 브라우저 창에 표시되는 문서
- View 함수가 반환하는 ***response*** 는 HTML로 된 간단한 문자열일 수도 있고 더 복잡한 형식을 취할 수도 있다.

매일 사용하는 서비스의 URL을 자세히 살펴보면 많은 곳에서 가변적인 부분들이 있음을 알 수 있다.

예를 들어, Facebook 프로필 페이지의 URL 형식은 `https://www.facebook.com/<your-name>`이며 사용자 이름이 포함되어 있고 각 사용자마다 다르다.

Flask는 `app.route` 데코레이터라는 특수 구문을 사용하여 이러한 유형의 URL을 지원한다.

동적 구성 요소가 있는 라우트:

```
@app.route('/user/<name>')
def user(name):
    return '<h1>Hello, {}!</h1>'.format(name)
```

- ✓ 정적 부분과 일치하는 모든 URL이 이 라우트에 매핑, 뷰 함수가 호출될 때 동적 구성 요소가 인자로 전달됨
- ✓ name 인자 => 디폴트로 string 타입
- ✓ `/user/<int:id>`와 같은 라우트도 가능 (예: `/user/123`)
- ✓ Flask는 라우트를 위해 `string`, `int`, `float` 및 `path`를 지원함
- ✓ Path 유형은 `string` 유형과 달리 슬래시(`\`/`/`)를 포함할 수 있는 특수 문자열 유형임

A Complete Application

애플리케이션 인스턴스, 라우트 및 뷰 함수 정의:

Example 2-1. hello.py: A complete Flask application

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

@app.route('/')
def index():
    return '<h1>Hello World!</h1>'
```

flasky > hello.py

Project

- flasky F:\practice\ai\handson(pycharm)\flasky
 - static
 - templates
 - hello.py
 - hello2.py
 - requirements.txt
- External Libraries
- Scratches and Consoles

hello.py

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5
6 @app.route('/')
7 def hello_world():
8     return '<h1>Hello World!</h1>'
9
10
11 if __name__ == '__main__':
12     app.run()
13
```

Run: Flask (hello1.py) ×

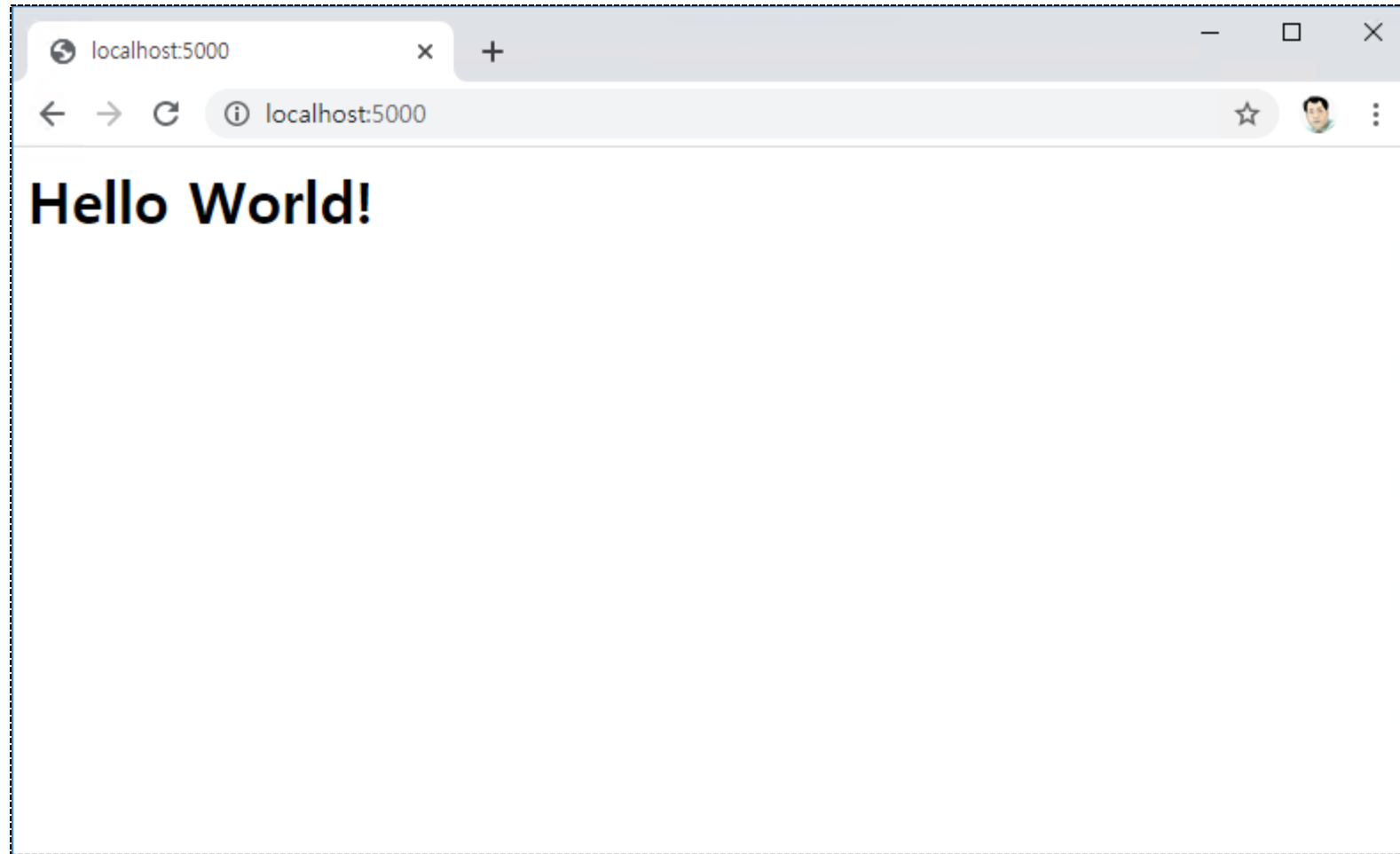
- FLASK_APP = hello1.py
- FLASK_ENV = development
- FLASK_DEBUG = 0
- In folder F:/practice/ai/handson(pycharm)/flasky
- C:\Users\user\Anaconda3\envs\aisam\python.exe -m flask run
- * Serving Flask app "hello1.py"
- * Environment: development
- * Debug mode: off
- * Running on <http://127.0.0.1:5000/> (Press CTRL+C to quit)

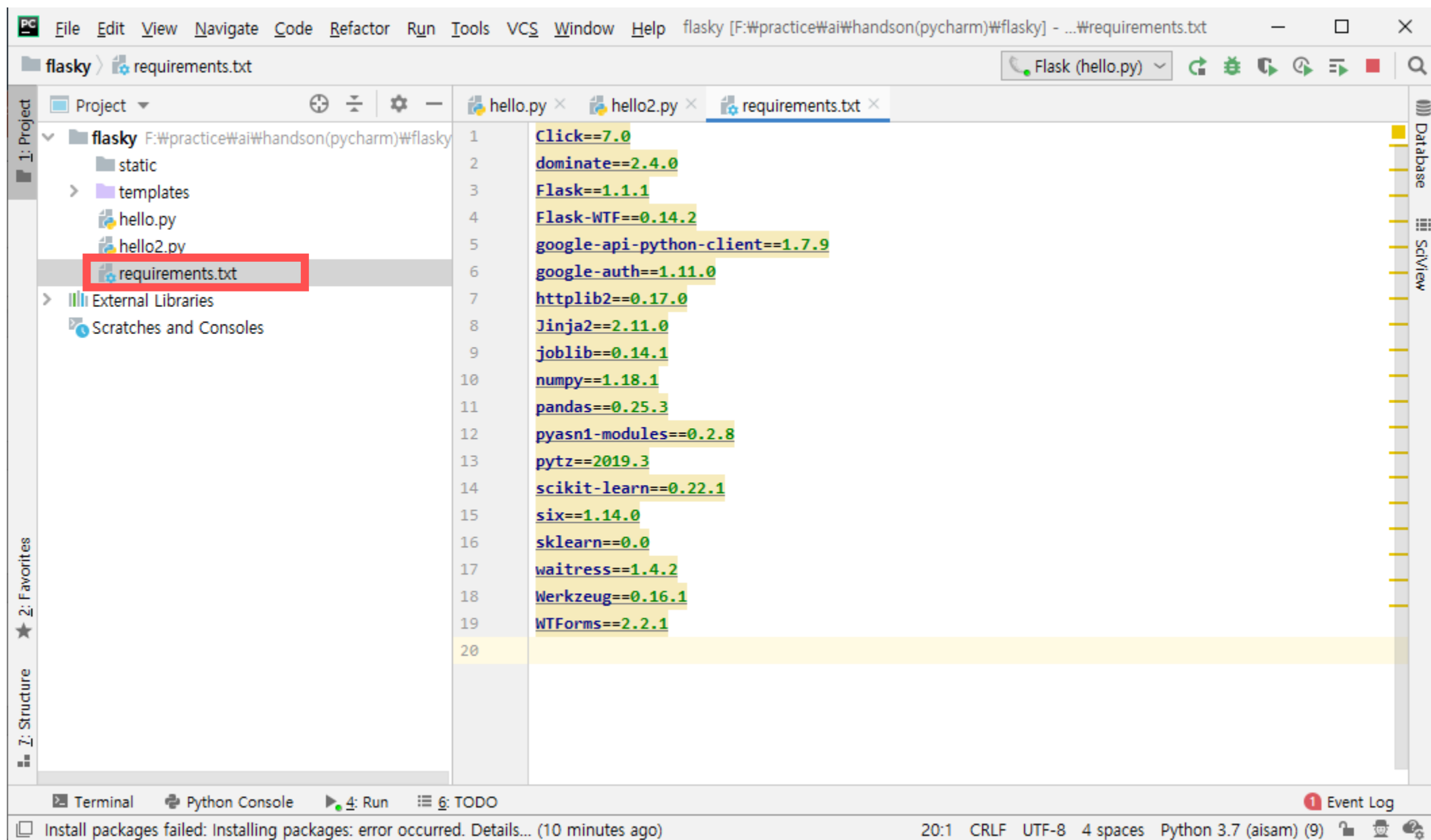
Flask command가 없을 때의 Flask 구버전에 필요.
지금은 flask run command가 있어서 불필요.
Unit testing에 유용!!

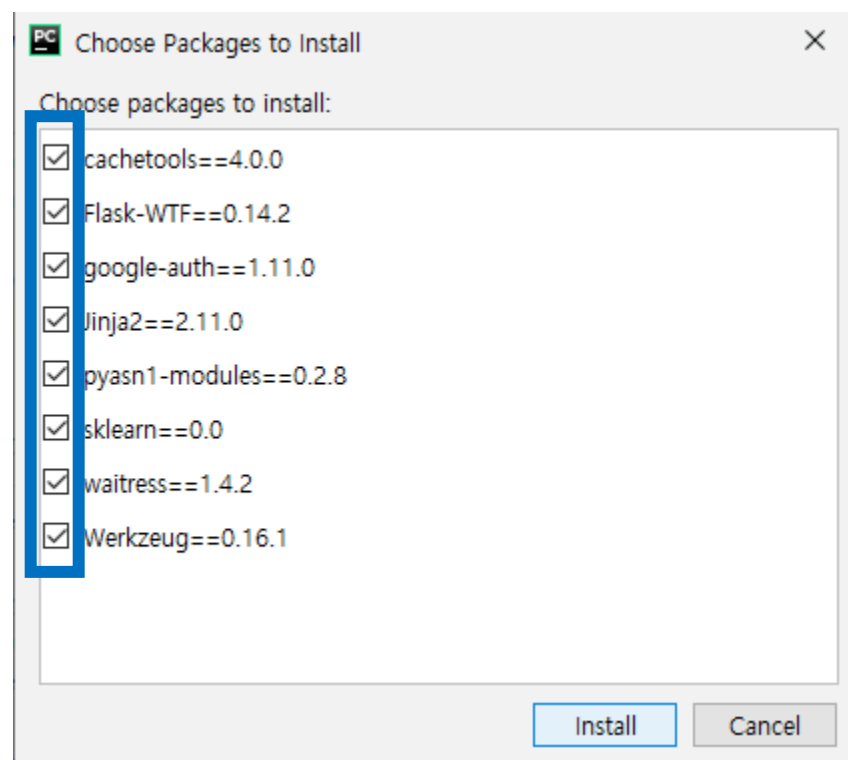
Terminal Python Console Run TODO

No occurrences found

12:14 CRLF UTF-8 4 spaces Python 3.7 (aisam) (9)







Dynamic Routes

Example 2-2에 표시된 두 번째 버전의 애플리케이션: 동적인 두 번째 경로 추가

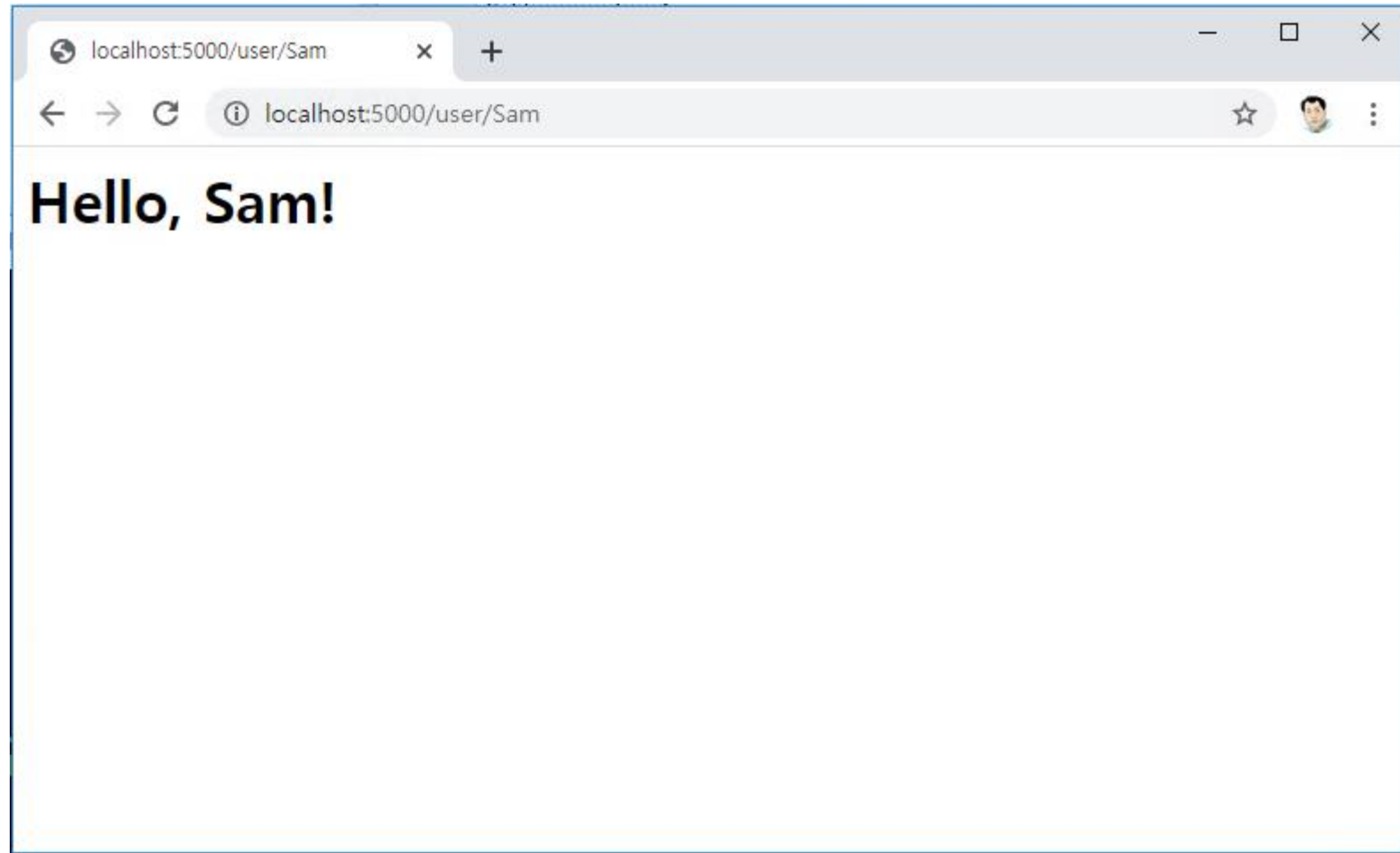
브라우저에서 동적 URL을 방문하면 URL에 제공된 이름이 포함된 개인화된 인사말이 표시된다:

Example 2-2. hello.py: Flask application with a dynamic route

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

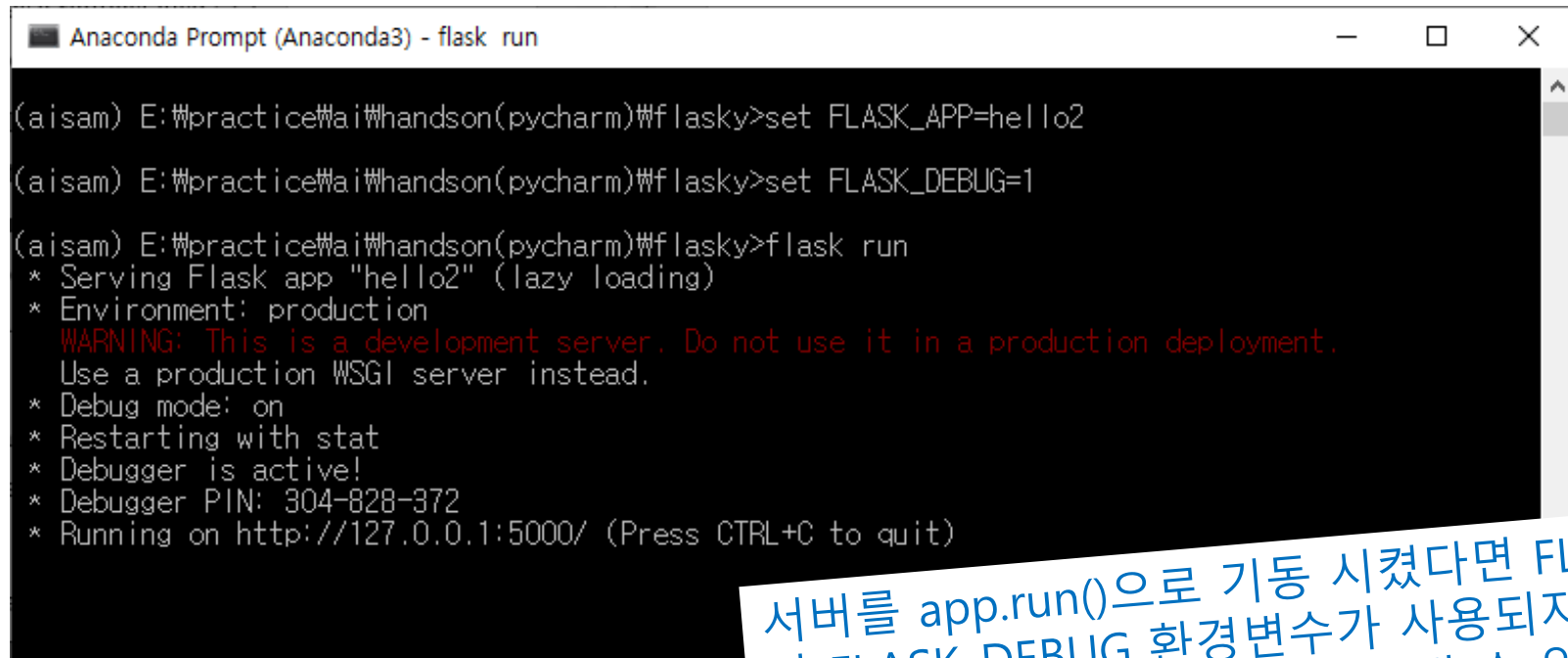
@app.route('/')
def index():
    return '<h1>Hello World!</h1>'

@app.route('/user/<name>')
def user(name):
    return '<h1>Hello, {}!</h1>'.format(name)
```



Debug Mode

(aisam) E:\practice\wai\handson(pycharm)\flasky>set FLASK_DEBUG=1



```
Anaconda Prompt (Anaconda3) - flask run

(aisam) E:\practice\wai\handson(pycharm)\flasky>set FLASK_APP=hello2
(aisam) E:\practice\wai\handson(pycharm)\flasky>set FLASK_DEBUG=1
(aisam) E:\practice\wai\handson(pycharm)\flasky>flask run
* Serving Flask app "hello2" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 304-828-372
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

서버를 `app.run()`으로 기동 시켰다면 `FLASK_APP`과 `FLASK_DEBUG` 환경변수가 사용되지 않음.
프로그래밍 방식으로 활성화시킬 수 있음:
`app.run(debug=True)`.

Templates

유지보수가 용이한 애플리케이션 => clean & well-structured code로 작성!

앞 예의 Flask 뷰 함수 => 두 가지 완전히 독립된 목적을 하나인 것처럼 취급함 => 문제를 야기할 수 있음

예: 웹 사이트에 새 계정을 등록하는 사용자의 경우

사용자: 웹 폼 작성, Submit 클릭

서버에서: 사용자가 제공한 데이터를 포함한 요청 도착, Flask가 등록 요청을 처리하는 뷰함수에게 위임,

뷰함수는 새 사용자를 DB에 추가, 성공 또는 실패 메시지를 포함하는 응답을 브라우저에게 반환.

(비즈니스 로직과 프리젠테이션 로직이 혼용되어 있음)

비즈니스 로직과 프리젠테이션 로직을 분리 => 이해하기 쉽고 유지보수성 향상

- 프리젠테이션 로직을 **템플릿**으로 이동하면 애플리케이션의 유지 보수성이 향상됨
- 템플릿 => placeholder 변수가 포함된 응답 텍스트를 포함하는 파일
- Placeholder 변수를 실제 값으로 치환하는 과정 => 렌더링(rendering)
- 템플릿 렌더링 작업을 위해 **Jinja2**라는 강력한 템플릿 엔진 제공!!

The Jinja2 Template Engine

Jinja2 템플릿 => 응답 텍스트를 포함하고 있는 파일

- `index()` 뷰 함수의 응답에 매칭되는 Jinja2 템플릿:

Example 3-1. templates/index.html: Jinja2 template

```
<h1>Hello World!</h1>
```

- `user()` 뷰 함수의 응답에 매칭되는 Jinja2 템플릿:

Example 3-2. templates/user.html: Jinja2 template

```
<h1>Hello, {{ name }}!</h1>
```

Rendering Templates

Flask => 디폴트로 애플리케이션 루트 디렉토리 안에서 **templates** 디렉토리에 있는 템플릿을 찾는다!!

Example 3-3. hello.py: rendering a template

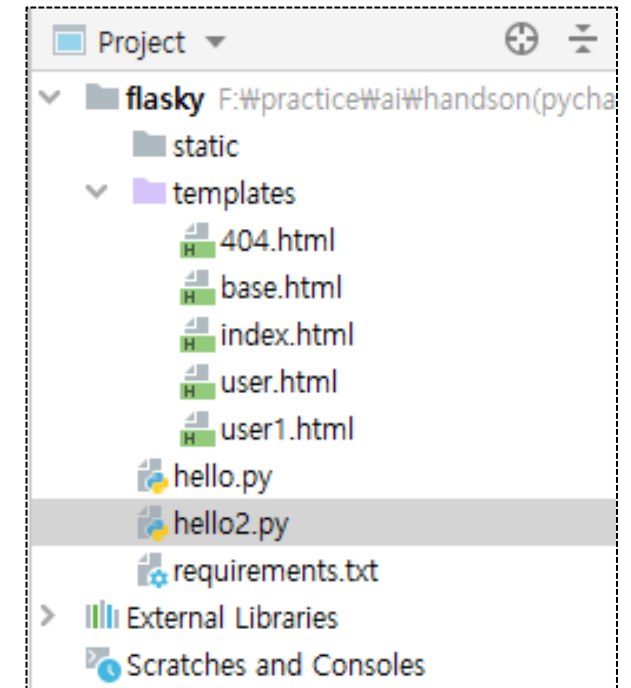
```
from flask import Flask, render_template

# ...

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

@app.route('/user/<name>')
def user(name):
    return render_template('user.html', name=name)
```

Flask에 의해 제공되는 `render_template()` 함수는 Jinja2 템플릿 엔진을 애플리케이션과 통합시켜 준다!



Variables

`{{ name }}` => 템플릿이 렌더링 될 때 제공되는 데이터로 name 변수가 치환됨:

`<p>A value from a dictionary: {{ mydict['key'] }}.</p>`

`<p>A value from a list: {{ mylist[3] }}.</p>`

`<p>A value from a list, with a variable index: {{ mylist[myintvar] }}.</p>`

`<p>A value from an object's method: {{ myobj.somemethod() }}.</p>`

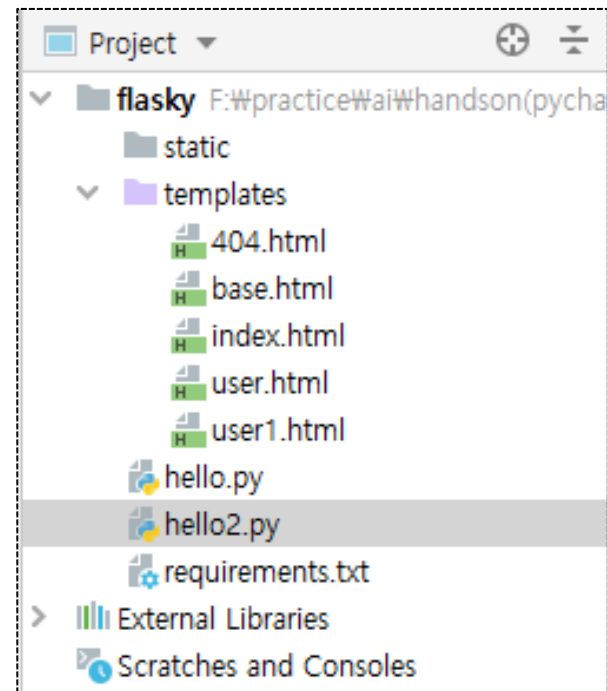
필터링:

`Hello, {{ name|capitalize }}`

Control Structure

Base 템플릿: base.html

```
<html>
<head>
    {% block head %}
    <title>{% block title %}{% endblock %} - My Application</title>
    {% endblock %}
</head>
<body>
    {% block body %}
    {% endblock %}
</body>
</html>
```



Base 템플릿은 파생 템플릿으로 재정의할 수 있는 블록을 정의한다.

Jinja2의 block 및 endblock 지시문은 base 템플릿에 추가되는 콘텐츠 블록을 정의한다.

이 예에는 head, title 및 body라는 블록이 있다.

다음 예제는 base 템플릿의 파생 템플릿이다:

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }}
    <style>
    </style>
{% endblock %}
{% block body %}
    <h1>Hello, World!</h1>
{% endblock %}
```


Bootstrap Integration with Flask-Bootstrap

Bootstrap

- Twitter의 오픈 소스 웹 브라우저 프레임워크
 - ✓ 데스크탑 및 모바일 플랫폼에서 사용되는 모든 최신 웹 브라우저와 호환되는 깨끗하고 매력적인 웹 페이지를 만드는데 도움이 되는 사용자 인터페이스 구성 요소를 제공한다.
- Bootstrap은 클라이언트 측 프레임워크
 - ✓ 서버가 직접 관련되지 않는다. 서버는 Bootstrap의 CSS(Cascading Style Sheets) 및 JavaScript 파일을 참조하는 HTML 응답을 제공하고 HTML, CSS 및 JavaScript 코드를 통해 원하는 사용자 인터페이스 요소를 인스턴스화하기만 하면 된다. 이 모든 작업을 수행하기에 이상적인 장소는 템플릿이다.
- Bootstrap을 애플리케이션과 통합하는 기본적인 방법
 - ✓ 필요한 모든 변경사항을 HTML 템플릿으로 만들
 - ✓ 그러나 **Flask 확장 기능**을 사용하면 통합 작업을 훨씬 간단하게 할 수 있다!!

Flask extension: **Flask-Bootstrap**

Anaconda Prompt (Anaconda3)

```
(base) C:\Users\User>cd F:\practice\ai\handson(pycharm)\flasky
(base) C:\Users\User>f:
(base) F:\practice\ai\handson(pycharm)\flasky>activate aisam
(aisam) F:\practice\ai\handson(pycharm)\flasky>pip install flask-bootstrap
Requirement already satisfied: flask-bootstrap in c:\users\user\anaconda3\envs\flasky\lib\site-packages (0.10.1)
Requirement already satisfied: visitor in c:\users\user\anaconda3\envs\flasky\lib\site-packages (0.1.3)
```

- Flask 확장기능 => 애플리케이션 인스턴스가 생성될 때 초기화된다:

Example 3-4. hello.py: Flask-Bootstrap initialization

```
from flask_bootstrap import Bootstrap
# ...
bootstrap = Bootstrap(app)
```

Bootstrap Integration with Flask-Bootstrap

Example 3-5. templates/user.html: template that uses Flask-Bootstrap

```
{% extends "bootstrap/base.html" %}

{% block title %}Flasky{% endblock %}

{% block navbar %}
<div class="navbar navbar-inverse" role="navigation">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle"
        data-toggle="collapse" data-target=".navbar-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="/">Flasky</a>
    </div>
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav">
        <li><a href="/">Home</a></li>
      </ul>
    </div>
  </div>
</div>

{% endblock %}

{% block content %}
<div class="container">
  <div class="page-header">
    <h1>Hello, {{ name }}!</h1>
  </div>
</div>

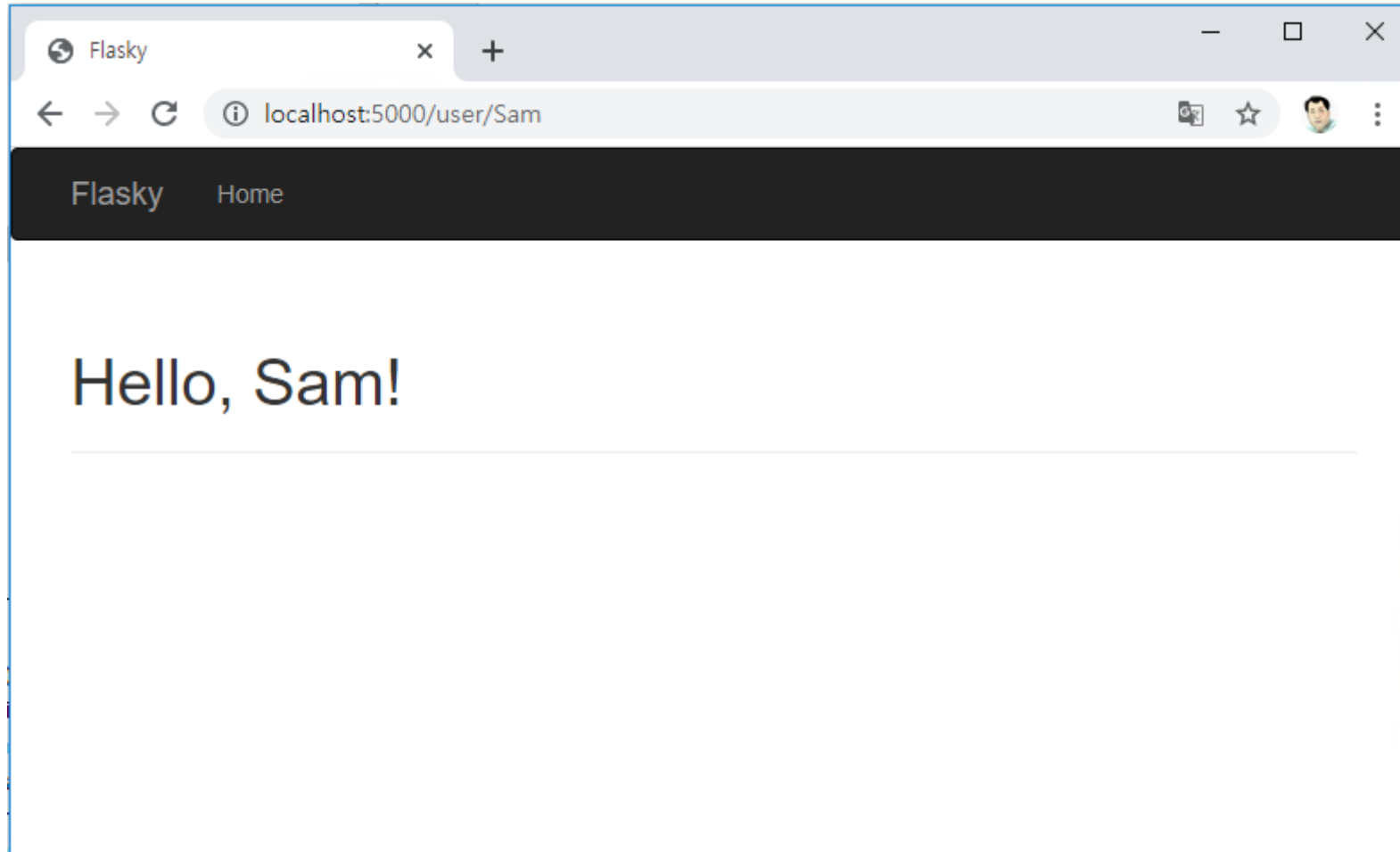
{% endblock %}
```

일단 Flask-Bootstrap이 초기화되면:
모든 Bootstrap 파일을 포함하는 base 템플릿을
애플리케이션에서 이용할 수 있다.

애플리케이션은 Jinja2의 템플릿 상속기능을 이용하여 base
템플릿을 확장한다.

Flask-Bootstrap의 base 템플릿은 모든 Bootstrap CSS 및
JavaScript 파일이 포함된 기본 웹 페이지를 제공한다.

Bootstrap templates:



Flask-Bootstrap에서 이용가능한
전체 block 리스트

Table 3-2. Flask-Bootstrap’s base template blocks

Block name	Description
doc	The entire HTML document
html_attribs	Attributes inside the <html> tag
html	The contents of the <html> tag
head	The contents of the <head> tag
title	The contents of the <title> tag
metas	The list of <meta> tags
styles	CSS definitions
body_attribs	Attributes inside the <body> tag
body	The contents of the <body> tag
navbar	User-defined navigation bar
content	User-defined page content
scripts	JavaScript declarations at the bottom of the document

Custom Error Pages

브라우저의 주소 바에 잘못된 라우트를 입력했을 경우:

- Flask: 커스텀 에러 페이지를 정의할 수 있도록 해 줌

Example 3-6. hello.py: custom error pages

```
@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

@app.errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500
```

- 에러 핸들러에서 참조되는 페이지(템플릿) 정의 필요
- 에러 템플릿: 정상적인 페이지와 동일한 레이아웃으로 만들어야 함(즉 네비게이션 바와 페이지 헤더를 포함시켜야 함)
- 간단한 방법: `templates/user.html`을 복사해서 커스터마이징 하는 방법
⇒ 많은 중복 코드 발생!!

Jinja2의 템플릿 상속기능 이용:

애플리케이션이 자신의 base 템플릿을 가질 수 있음

Example 3-7: templates/base.html 정의

⇒ bootstrap/base.html을 상속받는 새 템플릿 정의

⇒ templates/user.html, templates/404.html,
templates/500.html에 second-level 템플릿 제공

Example 3-7. templates/base.html: base application template with navigation bar

```
{% extends "bootstrap/base.html" %}

{% block title %}Flasky{% endblock %}

{% block navbar %}
<div class="navbar navbar-inverse" role="navigation">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle"
        data-toggle="collapse" data-target=".navbar-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="/">Flasky</a>
    </div>
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav">
        <li><a href="/">Home</a></li>
      </ul>
    </div>
  </div>
</div>
{% endblock %}

{% block content %}
<div class="container">
  {% block page_content %}{% endblock %}
</div>
{% endblock %}
```

Example 3-8. templates/404.html: custom code 404 error page using template inheritance

```
{% extends "base.html" %}
```

```
{% block title %}Flasky - Page Not Found{% endblock %}
```

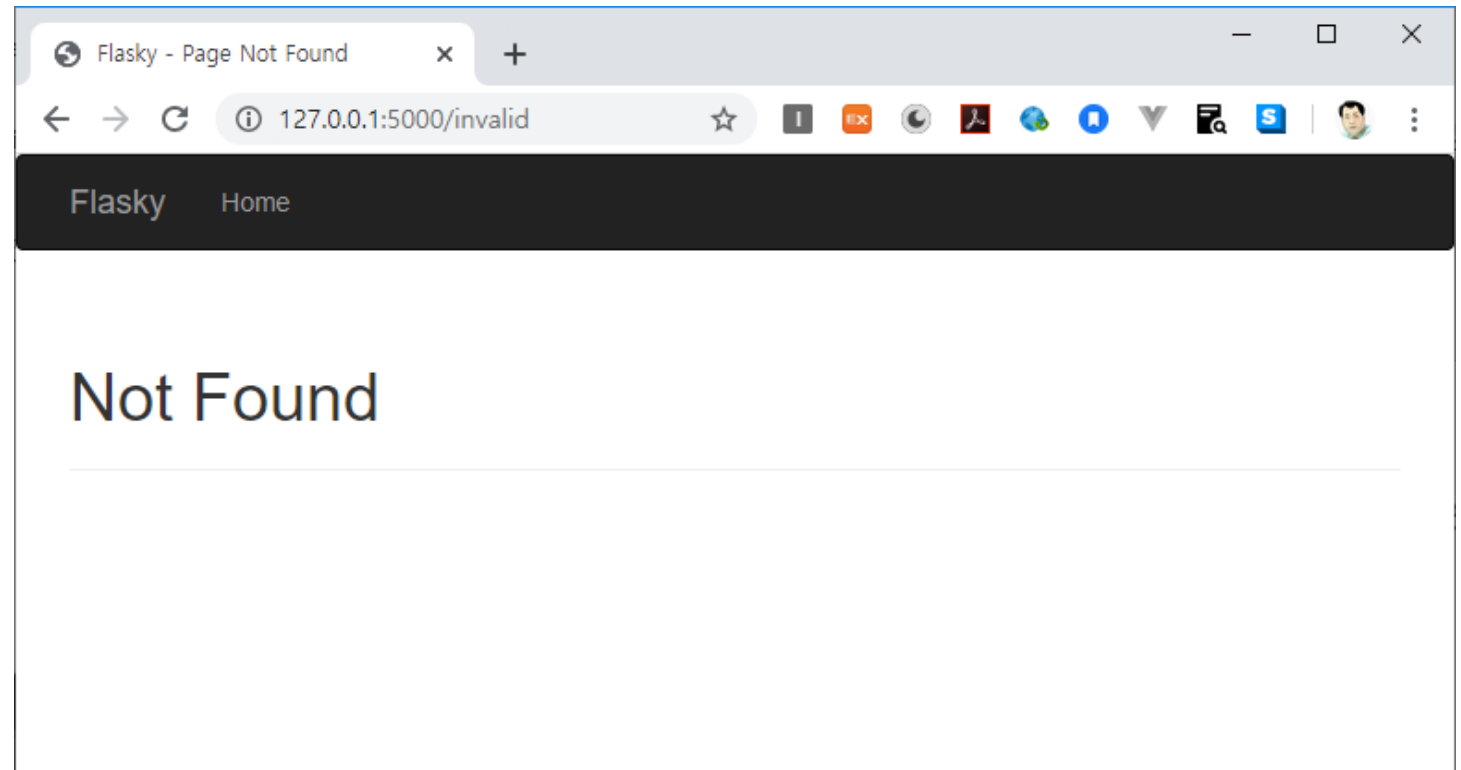
```
{% block page_content %}
```

```
<div class="page-header">
```

```
<h1>Not Found</h1>
```

```
</div>
```

```
{% endblock %}
```

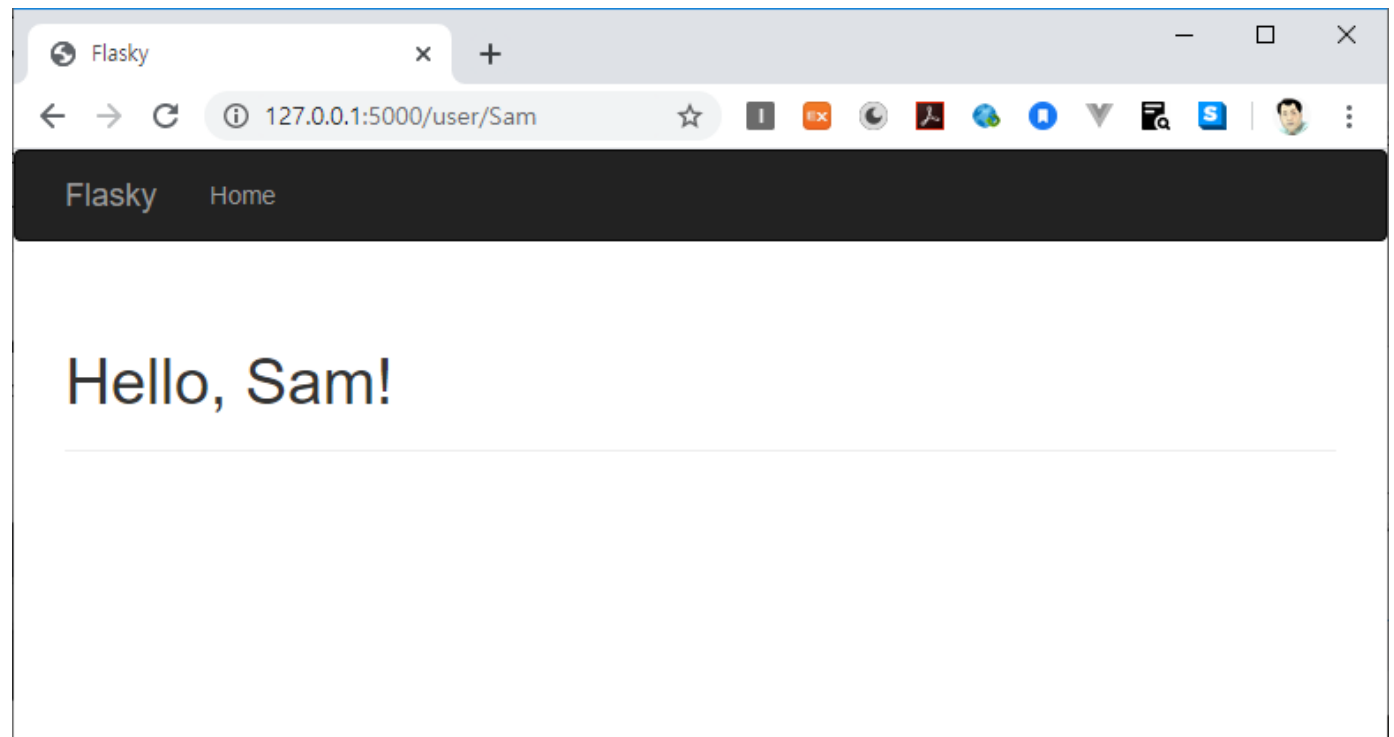


Example 3-9. templates/user.html: simplified page template using template inheritance

```
{% extends "base.html" %}

{% block title %}Flasky{% endblock %}

{% block page_content %}
<div class="page-header">
  <h1>Hello, {{ name }}!</h1>
</div>
{% endblock %}
```



Web Forms

지금까지 서버에서 클라이언트에게 정보를 보내는 템플릿에 대해 알아 보았음

이제 클라이언트(웹 브라우저)에서 서버로 보내는 폼 데이터에 대해 알아보자:

⇒ POST request: `request.form`을 이용해 정보에 접근 가능

⇒ 웹 폼을 처리하기 위해 Flask의 request 객체에 대한 지원기능을 이용해도 충분하지만...

Flask-WTF 확장기능 => 웹 양식 작업을 훨씬 더 즐거운 경험이 되도록!

- 프레임워크-독립적인 [WTForms](#) 패키지를 감싸는 Flask 통합 래퍼
- Flask-WTF and its dependencies can be installed with pip:

```
(aisam) ~> pip install flask-wtf
```

Configuration

Flask-WTF

- 대부분의 다른 확장과 달리 애플리케이션 수준에서 초기화할 필요는 없지만 애플리케이션에 비밀 키 (secret key)가 설정되어 있어야 함
- 비밀 키는 암호화 또는 서명 키로 사용되는 임의의 고유한 콘텐츠가 포함된 문자열
- Flask는 이 키를 사용하여 사용자 세션의 내용이 변조되지 않도록 보호
- Flask 응용 프로그램에서 비밀 키를 구성하는 방법:

Example 4-1. hello.py: Flask-WTF configuration

```
app = Flask(__name__)  
app.config['SECRET_KEY'] = 'hard to guess string'
```

app.config dictionary는 Flask, extensions 또는 애플리케이션 자체에서 사용하는 설정 변수를 저장하는 범용 장소

Form Classes

Flask-WTF를 사용할 때:

- 각 웹 Form은 FlaskForm 클래스를 상속하는 클래스에 의해 서버에 표시됨
- 폼 클래스 – 폼의 필드 리스트 정의
- 각 필드 객체에는 하나 이상의 유효성 검사기가 연결되어 있을 수 있다.
- 유효성 검사기 => 사용자가 제출한 데이터가 유효한지 확인하는 기능:

Example 4-2. hello.py: form class definition

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

class NameForm(FlaskForm):
    name = StringField('What is your name?', validators=[DataRequired()])
    submit = SubmitField('Submit')
```

Table 4-1. WTForms standard HTML fields

Field type	Description
BooleanField	Checkbox with <code>True</code> and <code>False</code> values
DateField	Text field that accepts a <code>datetime.date</code> value in a given format
DateTimeField	Text field that accepts a <code>datetime.datetime</code> value in a given format
DecimalField	Text field that accepts a <code>decimal.Decimal</code> value
FileField	File upload field
HiddenField	Hidden text field
MultipleFileField	Multiple file upload field
FieldList	List of fields of a given type
FloatField	Text field that accepts a floating-point value
FormField	Form embedded as a field in a container form
IntegerField	Text field that accepts an integer value
PasswordField	Password text field
RadioField	List of radio buttons
SelectField	Drop-down list of choices
SelectMultipleField	Drop-down list of choices with multiple selection
SubmitField	Form submission button
StringField	Text field
TextAreaField	Multiple-line text field

Table 4-2. WTForms validators


Validator	Description		
DataRequired	Validates that the field contains data after type conversion	NumberRange	Validates that the value entered is within a numeric range
Email	Validates an email address	Optional	Allows empty input in the field, skipping additional validators
EqualTo	Compares the values of two fields; useful when requesting a password to be entered twice for confirmation	Regexp	Validates the input against a regular expression
InputRequired	Validates that the field contains data before type conversion	URL	Validates a URL
IPAddress	Validates an IPv4 network address	UUID	Validates a UUID
Length	Validates the length of the string entered	AnyOf	Validates that the input is one of a list of possible values
MacAddress	Validates a MAC address	NoneOf	Validates that the input is none of a list of possible values

HTML Rendering of Forms

폼 필드: HTML로 렌더링 될 수 있게 호출 가능함

뷰 함수가 NameForm 인스턴스를 **form**이라는 인자로 템플릿에 전달한다고 가정하면

```
@app.route('/', methods=['GET', 'POST'])
def index():
    name = None
    form = NameForm()
    if form.validate_on_submit():
        name = form.name.data
        form.name.data = ''
    return render_template('index.html', form=form, name=name)
```



템플릿은 아래와 같은 HTML 폼을 생성할 수 있다:

```
<form method="POST">
  {{ form.hidden_tag() }}
  {{ form.name.label }} {{ form.name() }}
  {{ form.submit() }}
</form>
```

Flask-Bootstrap 확장기능: Flask-WTF 폼을 Bootstrap의 미리 정의된 스타일로 전체 Flask-WTF 폼을 렌더링 해주는 기능


- `import` 지시문 => 템플릿에서 템플릿 요소를 가져오고 사용할 수 있다.
- `bootstrap/wtf.html` 파일 => Bootstrap을 사용하여 Flask-WTF 양식을 렌더링하는 기능을 정의
- `wtf.quick_form()` 함수 => Flask-WTF 양식 객체를 가져와 기본 Bootstrap 스타일을 사용하여 렌더링:

Example 4-3. templates/index.html: using Flask-WTF and Flask-Bootstrap to render a form

```
{% extends "base.html" %}
{% import "bootstrap/wtf.html" as wtf %}

{% block title %}Flasky{% endblock %}

{% block page_content %}
<div class="page-header">
    <h1>Hello, {% if name %}{{ name }}{% else %}Stranger{% endif %}!</h1>
</div>
{{ wtf.quick_form(form) }}
{% endblock %}
```



Form Handling in View Functions

`hello.py`의 새 버전:

- `index()` 뷰 함수의 2가지 작업 : 폼 렌더링 & 사용자 입력 폼 데이터 수신

Example 4-4. `hello.py`: handle a web form with GET and POST request methods

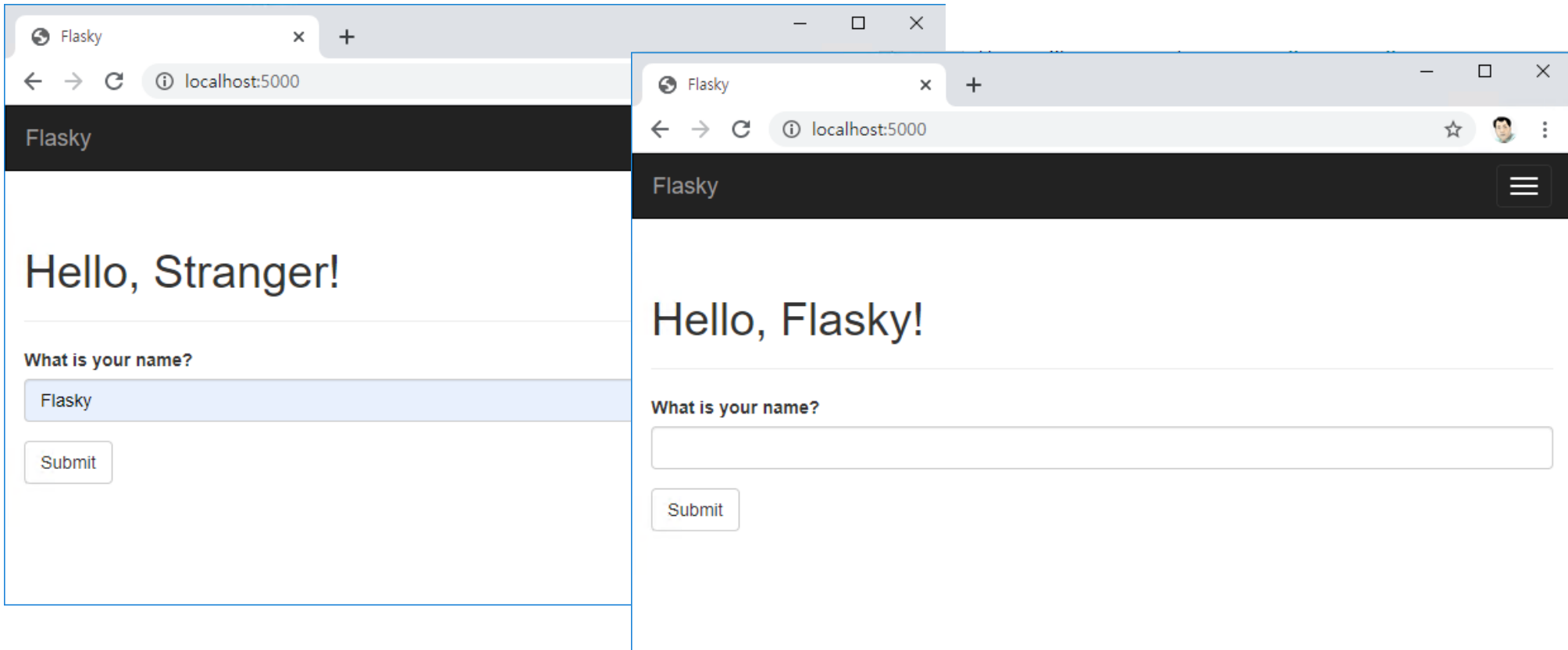
```
@app.route('/', methods=['GET', 'POST'])
def index():
    name = None
    form = NameForm()
    if form.validate_on_submit():
        name = form.name.data
        form.name.data = ''
    return render_template('index.html', form=form, name=name)
```

A yellow arrow points from the right towards the `form = NameForm()` line in the code block, highlighting the form object used for validation and rendering.

`validate_on_submit()` 메소드 : 폼 제출 & 데이터가 모든 유효성 검사기 통과 성공 시 => True!!

실습과제 16-1

Flask Web App 개발 - Hello Flasky!



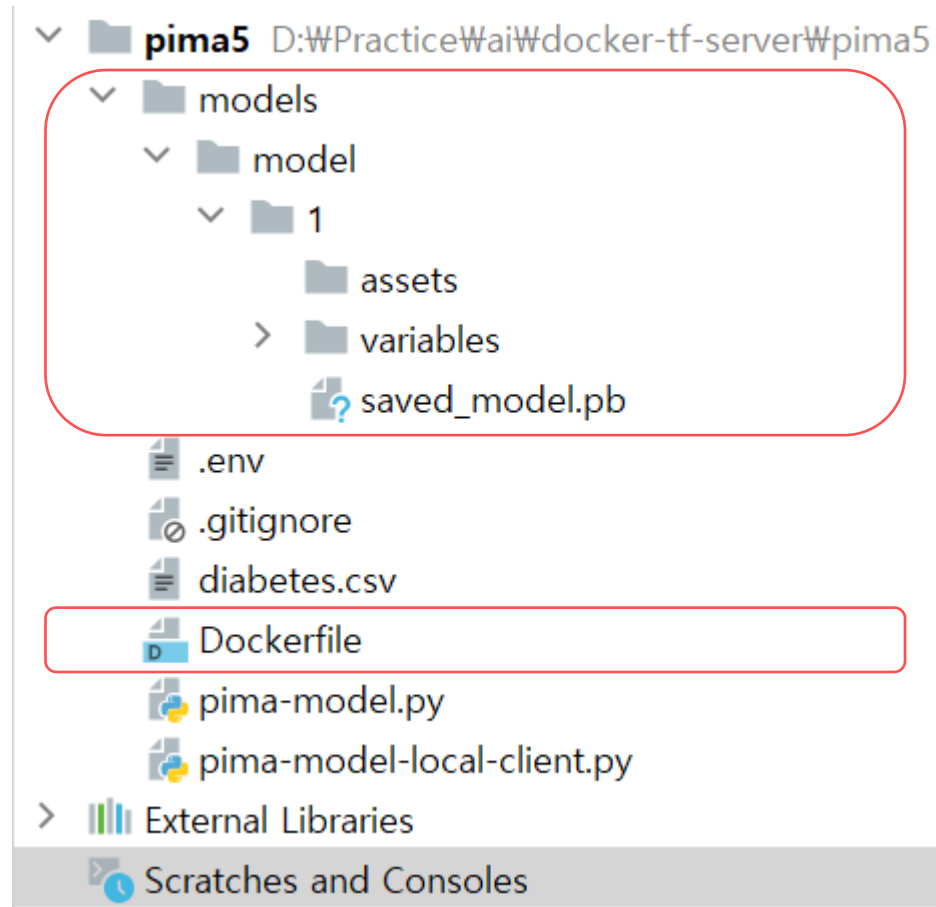
TensorFlow / Docker / Heroku를 이용하여 Deep Learning 모델을 클라우드(Heroku)에 배치하고 로컬에서 Flask 웹앱 실행하기

먼저 사이버캠퍼스 강의자료실의 '**Docker 설치.pdf**' 파일을 참조하여 Docker 설치한다!!

작업 순서:

1. TensorFlow Serving과 Docker를 이용하여 로컬하게 Saved model 서빙하기
2. Heroku 상의 Docker container에 호스팅하기

앱의 디렉토리 구조를 아래처럼 만든다. (Dockerfile, SavedModel, ...)



TensorFlow Serving을 위한 official docker image는 Heroku에서 동작하지 않으므로 아래처럼 수정한다:

Dockerfile:

```
# Copyright 2018 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#   https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

```
ARG TF_SERVING_VERSION=latest
ARG TF_SERVING_BUILD_IMAGE=tensorflow/serving:${TF_SERVING_VERSION}-devel
```

```
FROM ${TF_SERVING_BUILD_IMAGE} as build_image
FROM ubuntu:18.04
```

```
ARG TF_SERVING_VERSION_GIT_BRANCH=master
ARG TF_SERVING_VERSION_GIT_COMMIT=head
```

```
LABEL maintainer="gvasudevan@google.com"
LABEL tensorflow_serving_github_branchtag=${TF_SERVING_VERSION_GIT_BRANCH}
LABEL tensorflow_serving_github_commit=${TF_SERVING_VERSION_GIT_COMMIT}
```

```
RUN apt-get update && apt-get install -y --no-install-recommends \
    ca-certificates \
    && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
```

```
# Install TF Serving pkg
COPY --from=build_image /usr/local/bin/tensorflow_model_server
/usr/bin/tensorflow_model_server
```

```
# Expose ports
# gRPC
#EXPOSE 8500
```

```
# REST
#EXPOSE 8501
```

```
# Set where models should be stored in the container
ENV MODEL_BASE_PATH=/models
RUN mkdir -p ${MODEL_BASE_PATH}
```

```
# The only required piece is the model name in order to differentiate endpoints
ENV MODEL_NAME=model
```

```
COPY models/model models/model
# Create a script that runs the model server so we can use environment variables
# while also passing in arguments from the docker command line
RUN echo '#!/bin/bash \n\n\ntensorflow_model_server --rest_api_port=$PORT \n\n--model_name=${MODEL_NAME} --model_base_path=${MODEL_BASE_PATH}/${MODEL_NAME} \n\n"$@"' > /usr/bin/tf_serving_entrypoint.sh \
&& chmod +x /usr/bin/tf_serving_entrypoint.sh
```

```
CMD ["/usr/bin/tf_serving_entrypoint.sh"]
#CMD ["/usr/bin/tf_serving_entrypoint.sh"]
```

Copy/Paste 가능!!

1. 터미널을 열어서 앱의 루트 디렉토리로 이동
2. 아래 커맨드를 실행하여 Docker container 구축:

```
> docker build -t app_name .
```

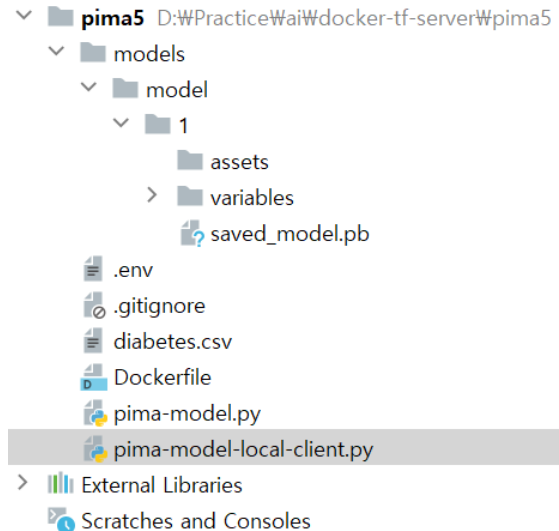
```
( 예: docker build -t pima5 . )
```

3. Docker container 테스트:

```
> docker run -p 8501:8501 -e PORT=8501 -t app_name
```

```
( 예: docker run -p 8501:8501 -e PORT=8501 -t pima5 )
```

이제 이미지가 포함된
json으로
[http://localhost:8501/v1
/models/model:predict](http://localhost:8501/v1/models/model:predict)
에 옆 코드와 같이
POST 요청하여 모델을
사용할 수 있다:



```
1 import numpy as np
2 import pandas as pd
3 import requests
4 import json
5 from sklearn.preprocessing import MinMaxScaler
6 from sklearn.model_selection import train_test_split
7
8 # read the file containing the pima indians diabetes data set
9 data = pd.read_csv('./diabetes.csv', sep=',')
10
11 # extract the X and y from the imported data
12 X = data.values[:,0:8]
13 y = data.values[:,8]
14
15 # use MinMaxScaler to fit a scaler object
16 scaler = MinMaxScaler()
17 scaler.fit(X)
18
19 # min max scale the X data
20 X = scaler.transform(X)
21
22 # split the test set into the train and test set
23 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
24
25 X_new = X_test[:5]
26
27 input_data_json = json.dumps({
28     "signature_name": "serving_default",
29     "instances": X_new.tolist(),
30 })
31
32 SERVER_URL = 'http://localhost:8501/v1/models/model:predict'
33 # SERVER_URL = 'https://sanbul5.herokuapp.com/v1/models/model:predict'
34
35 response = requests.post(SERVER_URL, data=input_data_json)
36 response.raise_for_status() # raise an exception in case of error
37 response = response.json()
38
39 y_proba = np.array(response["predictions"])
40 print("\ny_proba.round(2): \n", y_proba.round(2))
```

```
Run: pima-model-local-client x
C:\Users\user\anaconda3\envs\aisam\python.exe D:/Practice/ai/docker-tf-server/pi
2021-06-01 12:11:03.012597: I tensorflow/stream_executor/platform/default/dso_lo

y_proba.round(2):
[[0.38]
 [0.1 ]
 [0.06]
 [0.25]
 [0.5 ]]

Process finished with exit code 0
```


SavedModel 을 Heroku의 Docker container에 호스팅하기

SavedModel 을 Heroku의 Docker container에 호스팅하기:

Heroku CLI에서 heroku 계정에 로그인

```
> heroku login
```

앱의 루트 디렉토리로 이동:

Heroku container registry에 로그인:

```
> heroku container:login
```

Heroku에 앱 생성:

```
> heroku create app_name    (예: pima5)
```

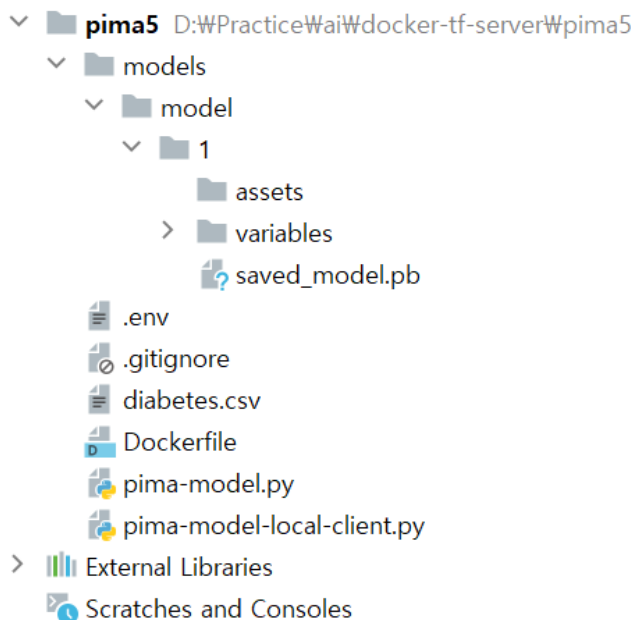
Heroku에 Docker image를 Push & Release:

```
> heroku container:push web -a app_name
```

```
> heroku container:release web -a app_name
```

이제 모델이 Heroku에 살아있다!!

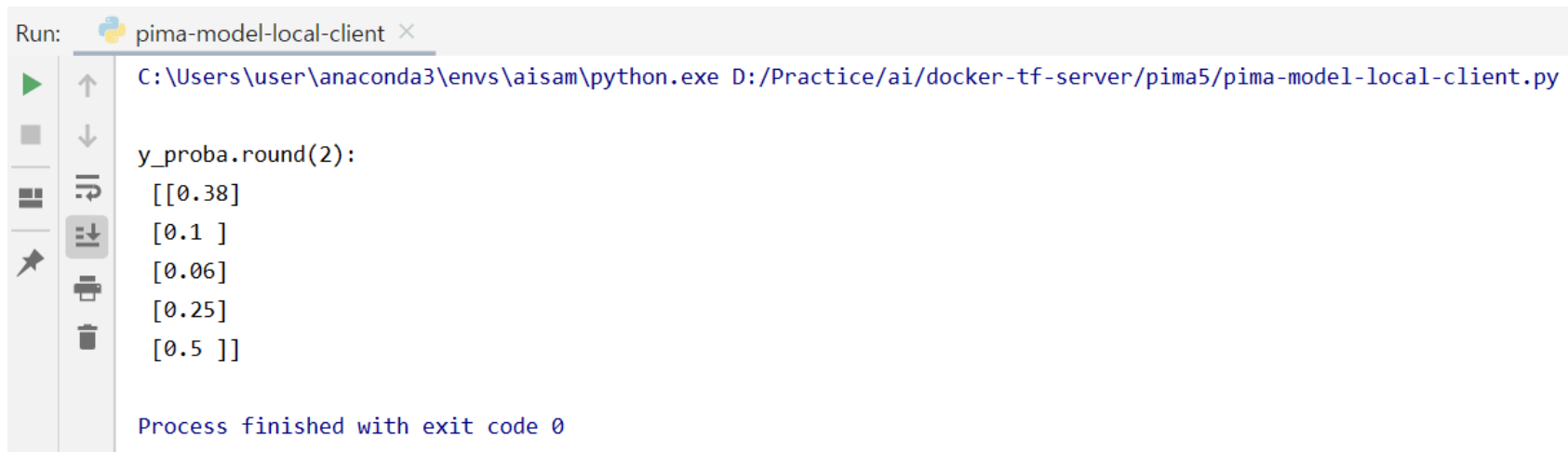
SERVER_URL을 http://app_name.herokuapp.com/v1/models/model:predict 으로 변경하여 Heroku 상의 모델에 접근 가능:



```
21
22 # split the test set into the train and test set
23 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
24
25 X_new = X_test[:5]
26
27 input_data_json = json.dumps({
28     "signature_name": "serving_default",
29     "instances": X_new.tolist(),
30 })
31
32 # SERVER_URL = 'http://localhost:8501/v1/models/model:predict'
33 SERVER_URL = 'https://pima5.herokuapp.com/v1/models/model:predict'
34
35 response = requests.post(SERVER_URL, data=input_data_json)
36 response.raise_for_status() # raise an exception in case of error
37 response = response.json()
38
39 y_proba = np.array(response["predictions"])
40 print("\ny_proba.round(2): \n", y_proba.round(2))
```

실습과제 16-2

- TensorFlow Serving / Docker / Heroku를 이용한 모델 배치
- 클라우드 모델을 이용한 로컬 클라이언트 개발



The screenshot shows a terminal window titled "Run: pima-model-local-client". The command executed is `C:\Users\user\anaconda3\envs\aisam\python.exe D:/Practice/ai/docker-tf-server/pima5/pima-model-local-client.py`. The output shows the result of `y_proba.round(2):` as a list of five arrays: `[[0.38]`, `[0.1]`, `[0.06]`, `[0.25]`, and `[0.5]]`. The terminal also indicates "Process finished with exit code 0".

```
Run: pima-model-local-client ×  
C:\Users\user\anaconda3\envs\aisam\python.exe D:/Practice/ai/docker-tf-server/pima5/pima-model-local-client.py  
  
y_proba.round(2):  
[[0.38]  
 [0.1 ]  
 [0.06]  
 [0.25]  
 [0.5 ]]  
  
Process finished with exit code 0
```

Flask Web App 개발

Flask 앱을 위한 디렉토리 생성 후 루트 디렉토리로 이동:

```
(base) cd (자신의 루트 디렉토리)
      (예: D:\Practice\ai\docker-tf-server\pimaclient5)
```

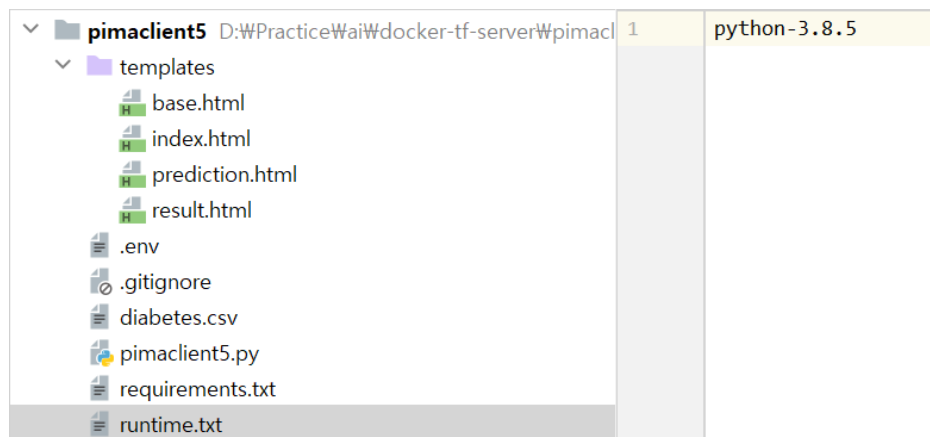
```
(base) activate aisam
```

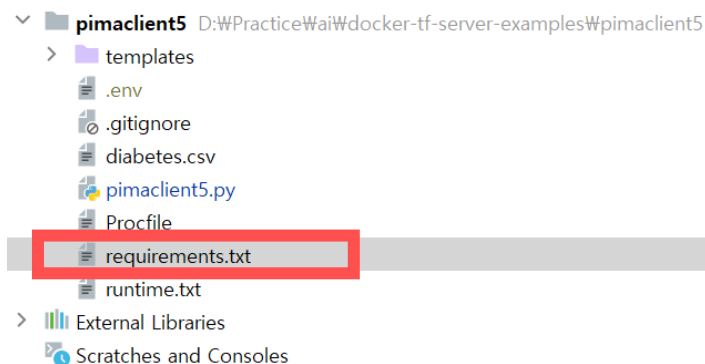
```
(aisam) pip freeze > requirements.txt
```

(다음 페이지 내용 복사/붙여넣기)

```
(aisam) python -version
Python 3.8.5
```

runtime.txt 파일에 아래 내용 추가:
Python-3.8.5

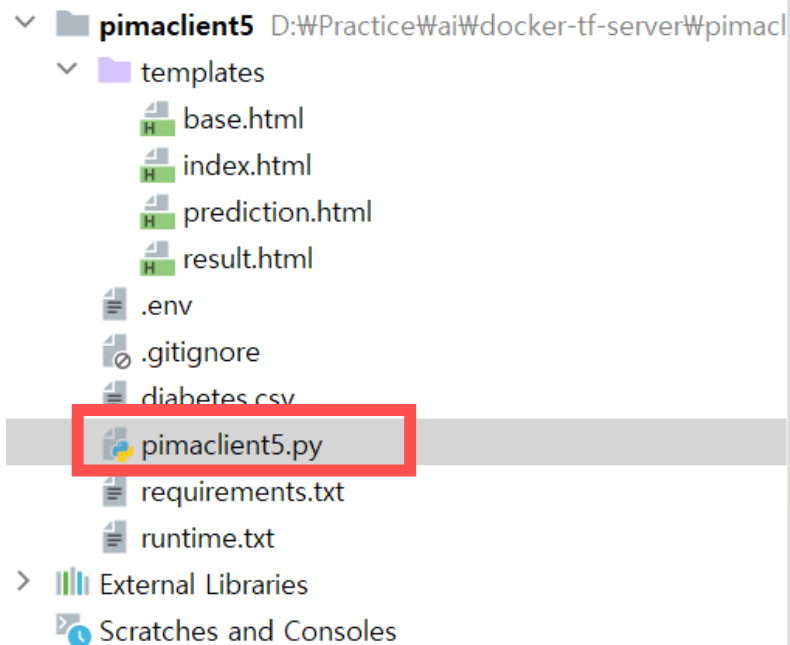




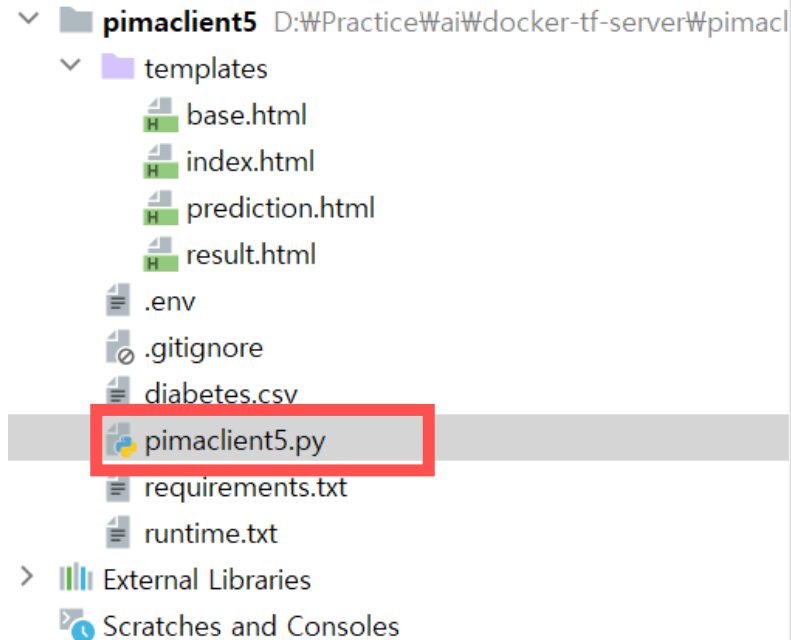
```
1  cachetools==4.0.0
2  Click==7.0
3  dominate==2.4.0
4  Flask==1.1.1
5  Flask-Bootstrap==3.3.7.1
6  Flask-WTF==0.14.2
7  google-api-python-client==1.7.9
8  google-auth==1.11.0
9  google-auth-http2==0.0.3
10 http2==0.17.0
11 itsdangerous==1.1.0
12 Jinja2==2.11.0
13 joblib==0.14.1
14 MarkupSafe==1.1.1
15 numpy==1.18.1
16 pandas==0.25.3
17 pyasn1==0.4.8
18 pyasn1-modules==0.2.8
19 python-dateutil==2.8.1
20 pytz==2019.3
21 rsa==4.0
22 scikit-learn==0.22.1
23 scipy==1.4.1
24 six==1.14.0
25 sklearn==0.0
26 uritemplate==3.0.1
27 visitor==0.1.3
28 waitress==1.4.2
29 Werkzeug==0.16.1
30 WTForms==2.2.1
31 matplotlib==3.0.2
32 requests==2.25.1
```

아래 내용 Copy/Paste!!

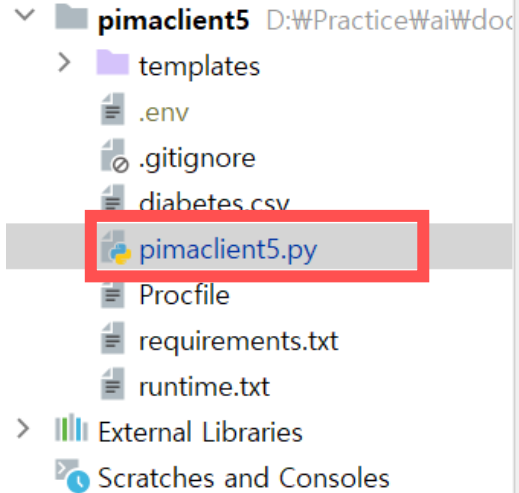
```
cachetools==4.0.0
Click==7.0
dominate==2.4.0
Flask==1.1.1
Flask-Bootstrap==3.3.7.1
Flask-WTF==0.14.2
google-api-python-client==1.7.9
google-auth==1.11.0
google-auth-http2==0.0.3
http2==0.17.0
itsdangerous==1.1.0
Jinja2==2.11.0
joblib==0.14.1
MarkupSafe==1.1.1
numpy==1.18.1
pandas==0.25.3
pyasn1==0.4.8
pyasn1-modules==0.2.8
python-dateutil==2.8.1
pytz==2019.3
rsa==4.0
scikit-learn==0.22.1
scipy==1.4.1
six==1.14.0
sklearn==0.0
uritemplate==3.0.1
visitor==0.1.3
waitress==1.4.2
Werkzeug==0.16.1
WTForms==2.2.1
matplotlib==3.0.2
requests==2.25.1
```



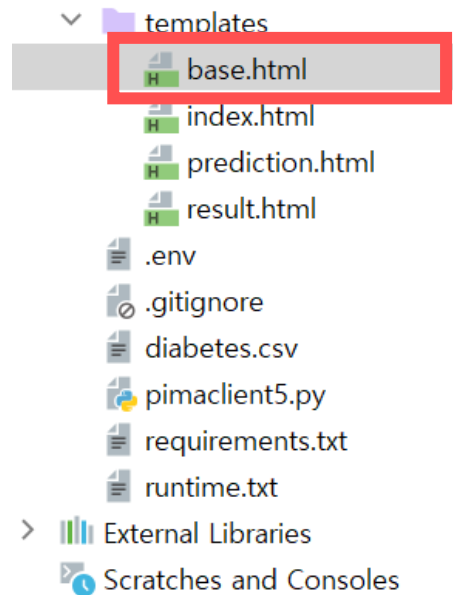
```
1 import numpy as np
2 import pandas as pd
3 import requests
4 import json
5 from sklearn.preprocessing import MinMaxScaler
6 from flask import Flask, render_template
7
8 app = Flask(__name__)
9 app.config['SECRET_KEY'] = 'hard to guess string'
10
11 from flask_bootstrap import Bootstrap
12
13 Bootstrap(app)
14
15 from flask_wtf import FlaskForm
16 from wtforms import StringField, SubmitField
17 from wtforms.validators import DataRequired
18
19 class LabForm(FlaskForm):
20     preg = StringField('# Pregnancies', validators=[DataRequired()])
21     glucose = StringField('Glucose', validators=[DataRequired()])
22     blood = StringField('Blood pressure', validators=[DataRequired()])
23     skin = StringField('Skin thickness', validators=[DataRequired()])
24     insulin = StringField('Insulin', validators=[DataRequired()])
25     bmi = StringField('BMI', validators=[DataRequired()])
26     dpf = StringField('DPF Score', validators=[DataRequired()])
27     age = StringField('Age', validators=[DataRequired()])
28     submit = SubmitField('Submit')
```



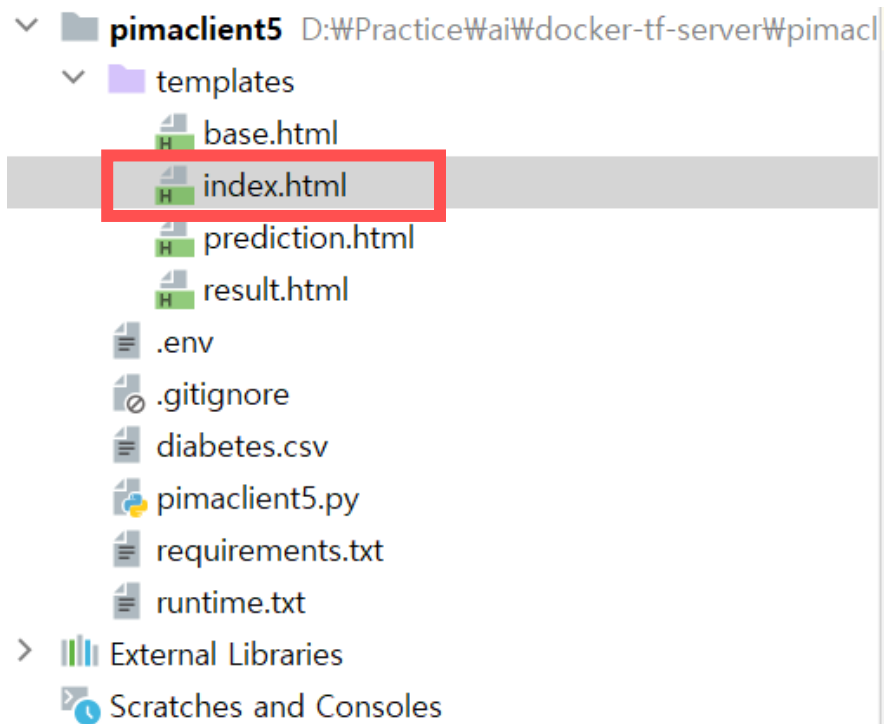
```
30 @app.route('/')
31 @app.route('/index')
32 def index():
33     return render_template('index.html')
34
35 @app.route('/prediction', methods=['GET', 'POST'])
36 def lab():
37     form = LabForm()
38     if form.validate_on_submit():
39         # get the form data for the patient data and put into a form for the
40         X_test = np.array([[float(form.preg.data),
41                             float(form.glucose.data),
42                             float(form.blood.data),
43                             float(form.skin.data),
44                             float(form.insulin.data),
45                             float(form.bmi.data),
46                             float(form.dpf.data),
47                             float(form.age.data)]])
48         print(X_test.shape)
49         print(X_test)
50
51         # get the data for the diabetes data.
52         data = pd.read_csv('./diabetes.csv', sep=',')
53
54         # extract the X and y from the imported data
55         X = data.values[:, 0:8]
56         y = data.values[:, 8]
57
58         # use MinMaxScaler to fit a scaler object
59         scaler = MinMaxScaler()
60         scaler.fit(X)
```



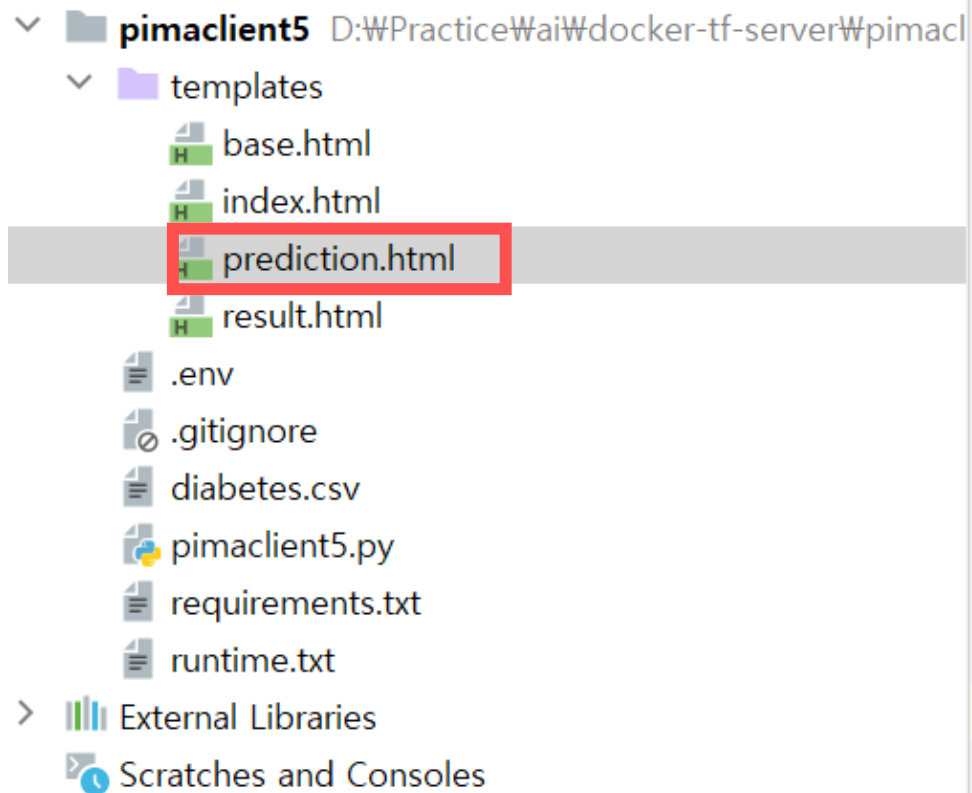
```
62 # min max scale the data for the prediction
63 X_test = scaler.transform(X_test)
64
65 input_data_json = json.dumps({
66     "signature_name": "serving_default",
67     "instances": X_test.tolist(),
68 })
69
70 # SERVER_URL = 'http://localhost:8501/v1/models/model:predict'
71 SERVER_URL = 'https://pima5.herokuapp.com/v1/models/model:predict'
72
73 response = requests.post(SERVER_URL, data=input_data_json)
74 response.raise_for_status() # raise an exception in case of error
75 response = response.json()
76
77 res = np.array(response["predictions"])
78 res = np.round(res, 2)
79
80 return render_template('result.html', res=res)
81
82 return render_template('prediction.html', form=form)
83
84 if __name__ == '__main__':
85     app.run()
```

```
1 <!doctype html>
2 <html>
3 <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <link rel="apple-touch-icon" href="/logo180.png" sizes="180x180" />
7     <link rel="icon" type="image/png" href="/logo180.png" sizes="192x192" />
8     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
9     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
10    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
11 </head>
12 <body>
13     {% block navbar %}
14     <div class="container">
15         <div class="page-header">
16             <div class="navbar navbar-fixed-top">
17                 <div class="container">
18                     <br>
19                     <a href="/index" class="btn btn-info" role="button">Home</a>
20                     <a href="/prediction" class="btn btn-info" role="button">Diagnostic</a>
21                 </div>
22             </div>
23         </div>
24     </div>
25     {% endblock %}
26     {% block content %}{% endblock %}
27     <div class="container">
28         <footer> Samkeun Kim</footer>
29     </div>
30 </body>
31 </html>
```



```
1 {% extends "base.html" %}  
2  
3 {% block content %}  
4  
5 <div class="container">  
6     <div class="page-header">  
7         <h3 class="alert alert-success">딥러닝을 이용한 당뇨병 진단 예측</h3>  
8     </div>  
9 </div>  
10  
11 {% endblock %}  
12  
13
```



```
1 {% extends "base.html" %}
2 {% import "bootstrap/wtf.html" as wtf %}
3
4 {% block content %}
5     <div class="container">
6         <div class="page-header">
7             <h1>Fill in the medical data of the patient</h1>
8
9             {{ wtf.quick_form(form) }}
10        </div>
11    </div>
12 {% endblock %}
13
```

▼

pimaclient5

D:\Practice\ai\docker-tf-server\pimaclient5

▼

templates

base.html

index.html

prediction.html

result.html

.env

.gitignore

diabetes.csv

pimaclient5.py

requirements.txt

runtime.txt

>

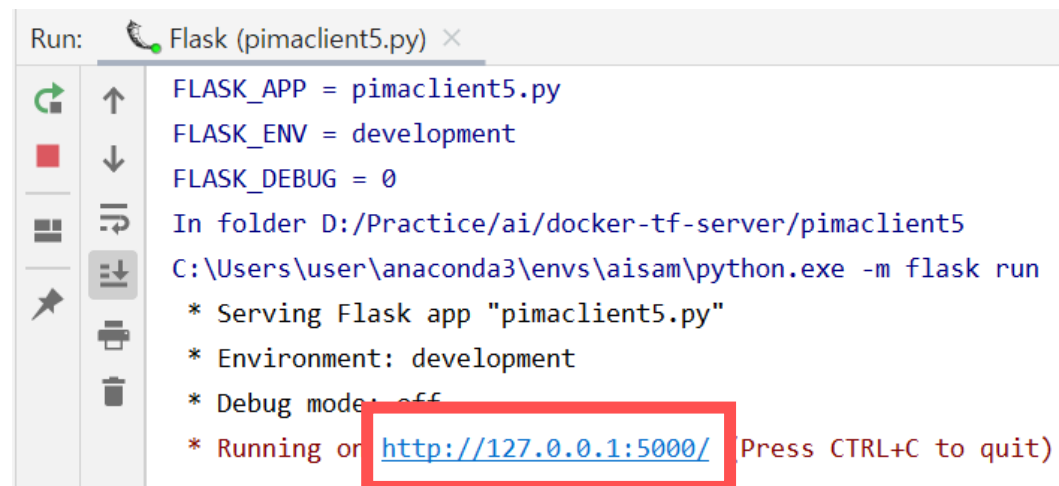
External Libraries

Scratches and Consoles

```
1 {% extends "base.html" %}
2
3 {% block content %}
4 <div class="container">
5     <div class="page-header">
6         <h2 class="alert alert-success"> 진단 예측 결과</h2>
7         <br>
8         <h4 class="alert alert-light">당뇨병 확률:</h4>
9         <h4 class="alert alert-dark"><p><b>{{ res }}</b></p></h4>
10    </div>
11 </div>
12 {% endblock %}
13
14
```

로컬에서 앱 실행하기:

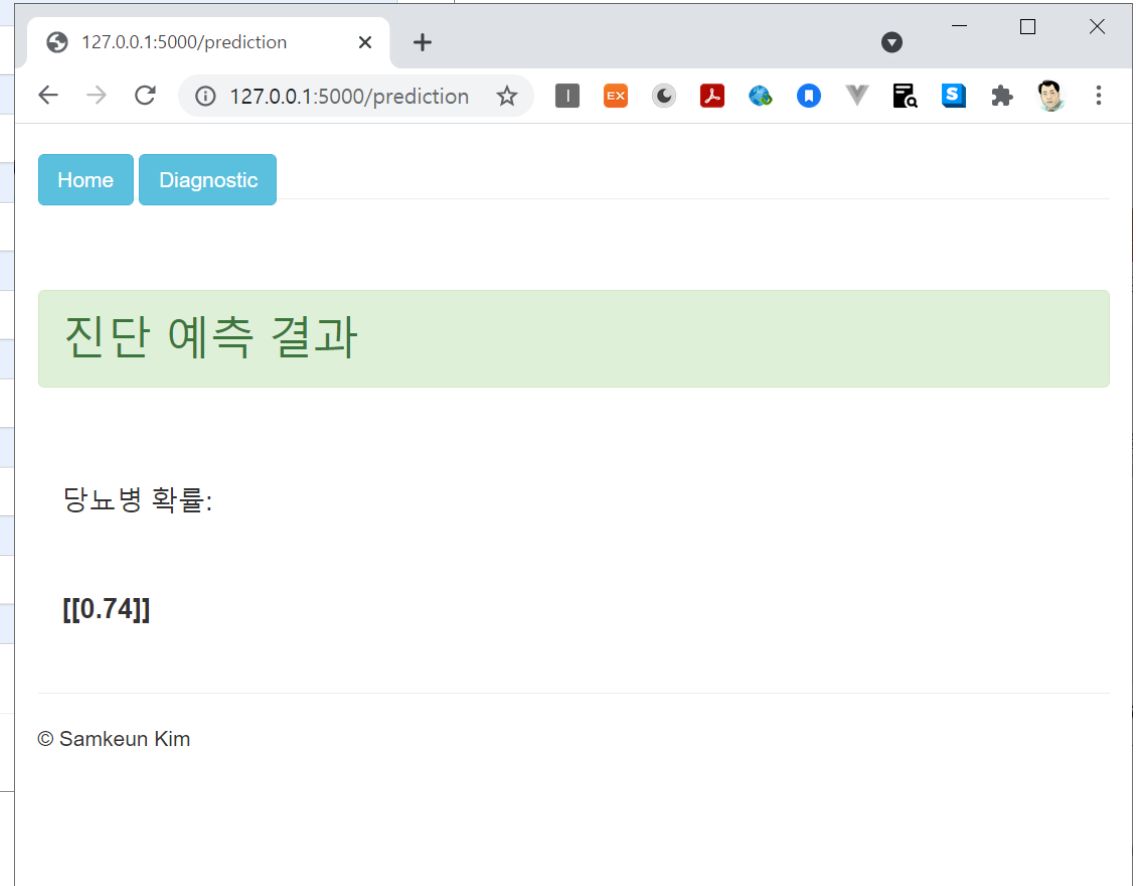
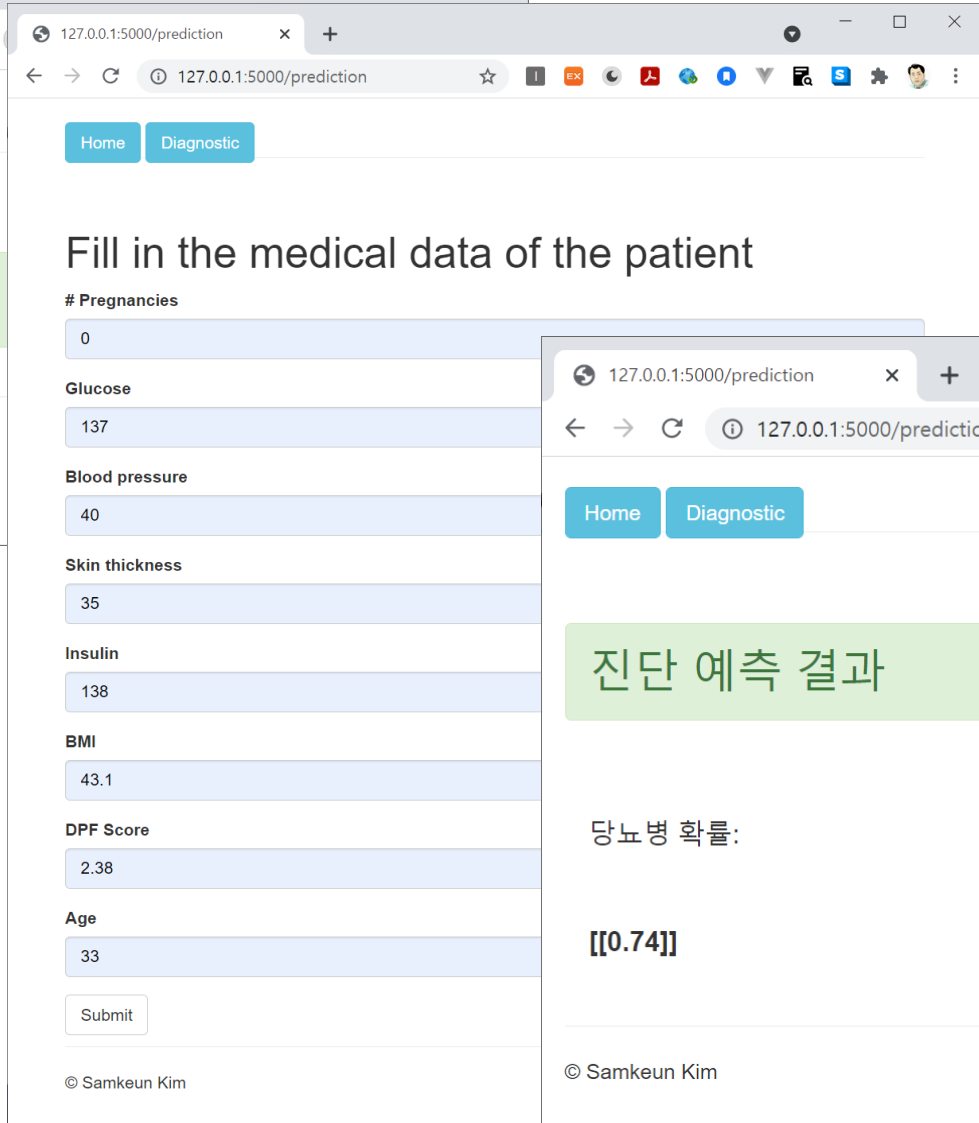
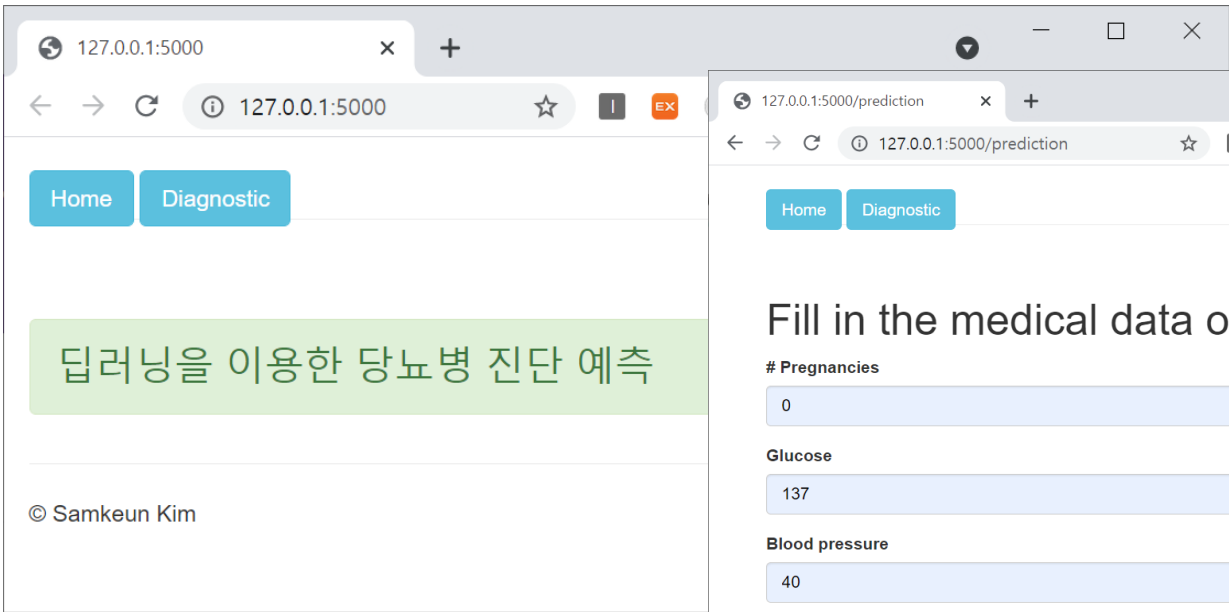
pimaclient5.py 실행 -> 브라우저에서 <http://127.0.0.1:5000/>으로 실행



The screenshot shows a terminal window titled "Run: Flask (pimaclient5.py)". The terminal output displays the following information:

```
FLASK_APP = pimaclient5.py
FLASK_ENV = development
FLASK_DEBUG = 0
In folder D:/Practice/ai/docker-tf-server/pimaclient5
C:\Users\user\anaconda3\envs\aisam\python.exe -m flask run
* Serving Flask app "pimaclient5.py"
* Environment: development
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

The URL <http://127.0.0.1:5000/> is highlighted with a red rectangle.



실습과제 16-3

Dynamic Model을 위한 Flask Web App 개발 (로컬에서 실행)

- Diabetes diagnostic web app (pimaclient5)

