Flask를 활용한 웹서비스 구현

파이썬 기반 웹 프레임워크

- Flask VS Django

- Flask와 Django는 파이썬에서 가장 널리 사용되는 오픈소스 기반 웹 프레임워크이다.
- 무엇이 좋다 나쁘다의 문제가 아닌 용도에 따른 적절한 선택의 문제이다.
- Django는 full stack web framework인 반면, Flask는 가볍고 확장 가능한 웹 프레임워크이다.

프레임워크

- 일반적으로 사용되는 코드를 체계화하여 쉽게 사용할 수 있도록 해주는 코드 집합
- 라이브러리보다는 좀 더 규모가 크다.
- 건축물에 비유하면 구조를 만드는 뼈대가 프레임워크이다.

파이썬 Flask

- Flask는 Python기반의 마이크로 프레임웍으로 단순하고 매우 가볍다.
- URL 라우팅, Template, Cookie, Debugger 및 개발서버 등 기본 기능만을 제공한다.
- 그러기 때문에 Django의 1/10밖에 안 되는 코드 (code 28,677 lines)로 구현되어 있어, 내부적으로 일어나는 동작을 직접 소스코드를 통해 확인할 수 있다.
- Flask는 크게 WSGI용 Library인 Werkzeug와 HTML 렌더링 엔진인 Jinja2 template로 구성 된다. 즉, Flask가 제공하는 기본 기능에 다양한 확장 모듈을 이용할 수 있는 구조 이다.

Django에서는 특수한 경우 내부 로직에서 어떠한 기능을 지원하지 않거나 장애가 발생했을 때 이를 해결하기위해 큰 비용이 들게 되지만, Flask는 정해진 확장 모듈이 없기때문에 다양한 방법으로 해결이 가능하다. (양면성)

Flask는 단 10줄도 안되는 코드로 웹 서버를 구동할 수 있다.
from flask import Flask 를 시작으로 서버를 시작하는 코드까지 10줄이 안된다.
물론 그만큼 최소한의 패키지로 구성되어 있기 때문에 Hello World는 간단히 구현 되지만, 상용 웹 서버를 구현할 때는 단순하지 않다.

파이썬 Django

- Django는 python기반 웹 프레임웍 중 가장 많이 사용되는 것 중 하나이다.
- Flask보다 약 10배 많은 코드 라인으로 개발 되었다.
- 구글 앱 엔진에서 Django를 사용하게 되면서 많은 사람들이 사용하게 되었으며, 웹 어플리케이션을 개발하기 위한 대부분의 기능들이 갖추어져 있기 때문에 외부 도구 및 라이브러리를 사용하지 않고도 어느 정도 규모가 있는 웹 어플리케이션 개발에도 문제가 없다.

- **Django는 ORM (Object Relational Mapping) 기능이 내장되어 있다.**
  객체 관계 매핑이라고도 하며 데이터베이스 시스템과 데이터 모델 클래스를 연결하는 역할을 한다. ORM을 이용해 다양한 데이터베이스를 지원하며, SQL 의존적인 코드를 벗어나 생산적인 코딩이 가능하게 되어 유지보수가 편하다.

- **Django는 자동으로 관리자 화면을 구성해 준다.**
  Django는 데이터베이스에 대한 관리 기능을 위하여 프로젝트를 시작하는 시점에 관리자 화면을 제공한다. 이런 관리자 화면을 이용하여 웹 어플리케이션에서 사용하는 데이터들을 쉽게 생성하거나 변경이 가능하다.

Flask First Example

New File을 선택하여 app.py이라는 파일명으로 아래 내용을 작성한다.

app.py

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

@app.route('/')
def hello():
    return 'Hello World!'
```

터미널 창에 아래의 내용을 입력한다.

```
set FLASK_APP=app

flask run
```

FLASK_APP은 Flask 서버를 기동할 때, 실행할 어플리케이션을 지정해주는 역할을 한다.

FLASK_DEBUG는 Flask 어플리케이션에 디버그 모드를 활성화 하여 문제가 발생한 경우 어떤 문제가 발생했는지 자세히 알려 준다. 또한 어플리케이션 코드가 변경되었을 때, 자동으로 Flask 서버를 재시작하여 매번 서버를 재시작해야 하는 귀찮음을 덜어 준다. 만약 자동으로 서버를 재시작하고 싶지 않은 경우에는 **flask run --no-restart** 옵션을 추가하면 된다.

참고 : 직접 실행 VS 임포트 실행

HelloTop.py

```
import helloModule

helloModule.func()
```

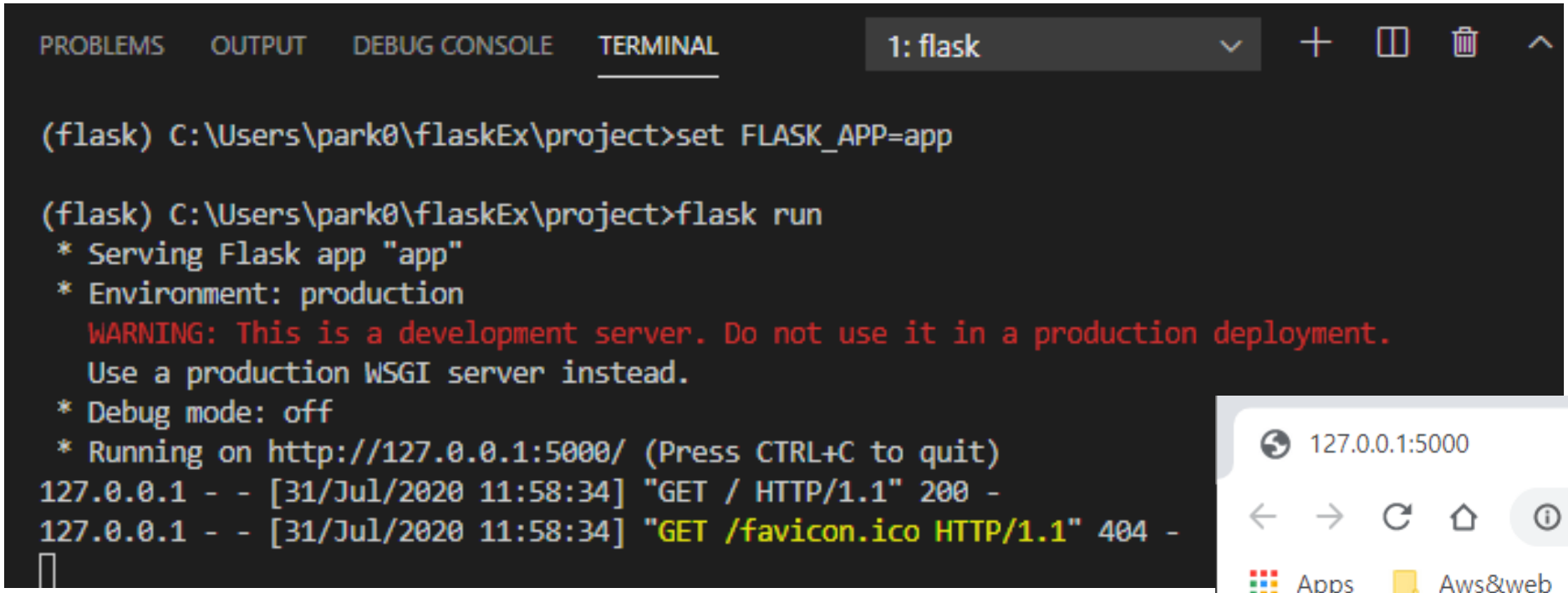
```
(flask) C:₩User₩park₩flaskEx₩> python HelloTop.py
임포트되어 사용됨
helloModule
function working
```

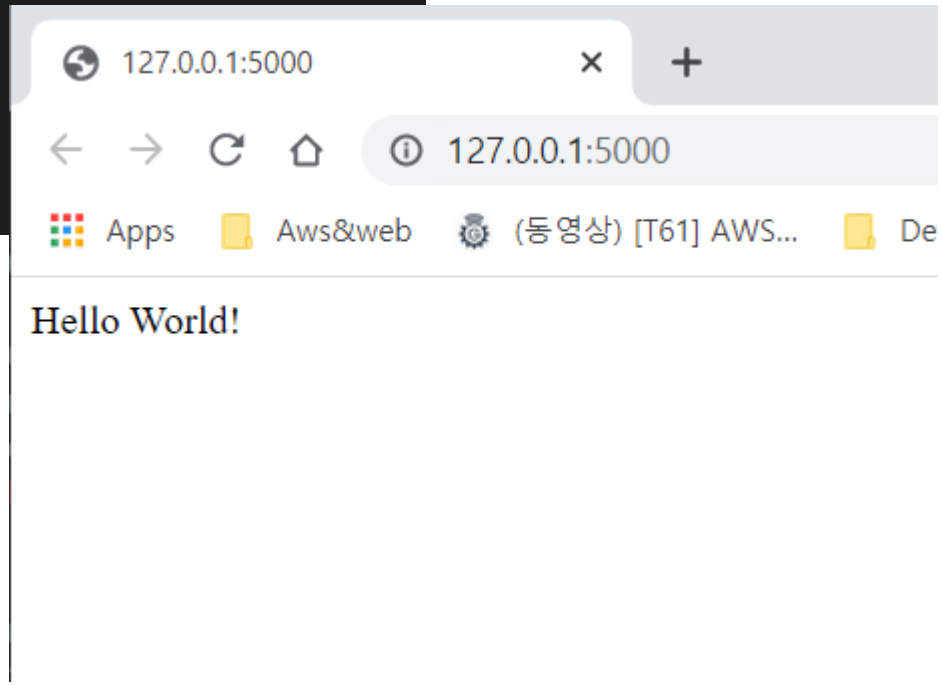
HelloModule.py

```
def func():
    print("function working")

if __name__ == "__main__":
    print("직접 실행")
    print(__name__)
else:
    print("임포트되어 사용됨")
    print(__name__)
```

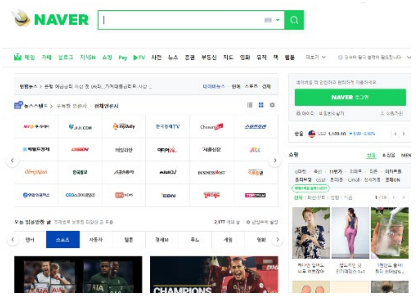
```
(flask) C:₩User₩park₩flaskEx₩> python helloModule.py
직접 실행
__main__
```

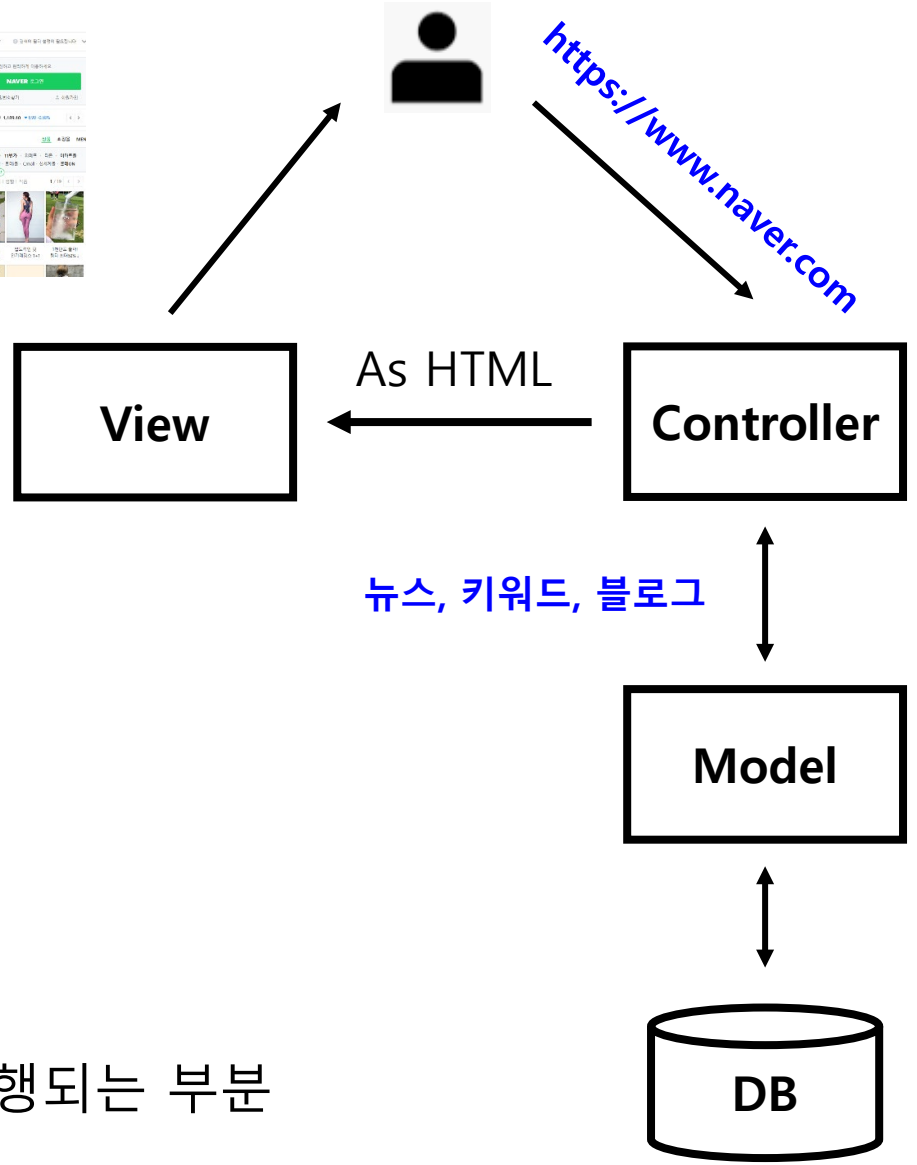
Flask First Example 실행 결과





MVC란?





Model : 데이터베이스와 연결되는 부분
View : 클라이언트가 보는 부분
Controller : 접근 URL에 따라 비즈니스 로직이 실행되는 부분

SQL 데이터베이스

- 데이터 베이스는 구조적인 방법으로 어플리케이션 데이터를 저장한다.
- 어플리케이션은 필요한 특정 데이터를 추출하기 위해 쿼리(query)를 실행한다.
- SQL이라는 이름도 구조화된 쿼리 언어(Structured Query Language)이다.

- 최근에는 문서기반(document-oriented)과 키-값(key-value)데이터베이스를 사용하기 시작했다.
- 이러한 데이터베이스는 NoSQL데이터베이스로 대중화되고 있다.

파이썬 데이터베이스 프레임웍

- 파이썬은 오픈 소스와 상용 데이터베이스에 대한 대부분의 데이터베이스 엔진을 위한 패키지가 있다.
- 이는 사용자가 어떤 DB를 사용하든 제한이 없다는 것이다.
- 원한다면 MySQL, Postgre, SQLite, Redis, MongoDB, CouchDB등 편한 것을 사용하면 된다.

- 선택하기 쉽지 않다면 **SQLAlchemy**나 **MongoEngine**같은 데이터베이스 추상화 레이어를 사용하자.
- 이는 쿼리언어같은 기본 데이터베이스 엔터티 대신 일반적인 파이썬 오브젝트보다 더 상위 레벨에서 동작 할 수 있도록 한다.

  Python의 객체에 데이터 모델을 정의하고, 이를 데이터베이스와 매핑해주는 것을 ORM(Object Relational Mapping)이라고 한다. 덕분에 코드는 특정 데이터베이스에 종속되지 않고, 기본 객체 만으로 데이터를 기술할 수 있기 때문에 조금 더 OOP 스러운 코드를 작성할 수 있다.

  SQLAlchemy는 이런 **ORM**이라고 이해하면 된다. 그리고 Flask에서 플러그인 처럼 사용하기 쉽게 만들어져 있다.

flask sql-alchemy 패키지 설치

Flask는 데이터베이스 기능을 내장하고 있지 않다. 따라서 우리는 적당한 데이터베이스를 선택해서 사용하는데, 각 데이터베이스마다 사용되는 API도 다르고, SQL 문법도 다르다. Python의 객체에 데이터 모델을 정의하고, 이를 데이터베이스와 매핑해주는 것을 ORM(Object Relational Mapping)이라고 한다. 덕분에 코드는 특정 데이터베이스에 종속되지 않고, 기본 객체 만으로 데이터를 기술할 수 있기 때문에 조금 더 OOP 스러운 코드를 작성할 수 있다.

Python에서 ORM으로 많이 쓰이는 것 중 SQLAlchemy가 있는데, 이를 Flask에서 플러그인 처럼 사용하기 쉽게 만들어진 Flask-SQLALchemy가 있습니다.

```
pip install flask-sqlalchemy
```

flask sql-alchemy 기본 동작 확인

app.py

```
import os
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

basedir = os.path.abspath(os.path.dirname(__file__))
dbfile = os.path.join(basedir, 'db.sqlite')

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///' + dbfile
app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN'] = True
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

class Test(db.Model):
    __tablename__ = 'test_table'
    id = db.Column(db.Integer, primary_key = True)
    name = db.Column(db.String(32), unique=True)

db.create_all()

@app.route('/')
def hello():
    return 'Hello World!'
```

flask sql-alchemy 패키지 설치

터미널에서 아래와 같이 코드를 실행한다.

```
python app.py
```

실행을 하면 dq.sqlite파일이 생성된다.
파일내용을 살펴보려면 아래 내용을 실행한다.

```
sqlite3 db.sqlite
```

```
(flask) C:\Users\park0\flaskEx\project>sqlite3 db.sqlite
SQLite version 3.32.3 2020-06-18 14:00:33
Enter ".help" for usage hints.
sqlite> .tables
test_table
sqlite> .schema
CREATE TABLE test_table (
        id INTEGER NOT NULL,
        name VARCHAR(32),
        PRIMARY KEY (id),
        UNIQUE (name)
);
sqlite>
```

flask sql-alchemy 기본 동작 확인 2

app.py

```
import os
from flask import Flask
from flask import render_template
from flask_sqlalchemy import SQLAlchemy

basedir = os.path.abspath(os.path.dirname(__file__))
dbfile = os.path.join(basedir, 'db.sqlite')

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///' + dbfile
app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN'] = True
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

# html을 만들어서 전달하면 Controller(현재파일)과
# View를 분리하게 된다.
@app.route('/')
def hello():
    return render_template('hello.html')
```

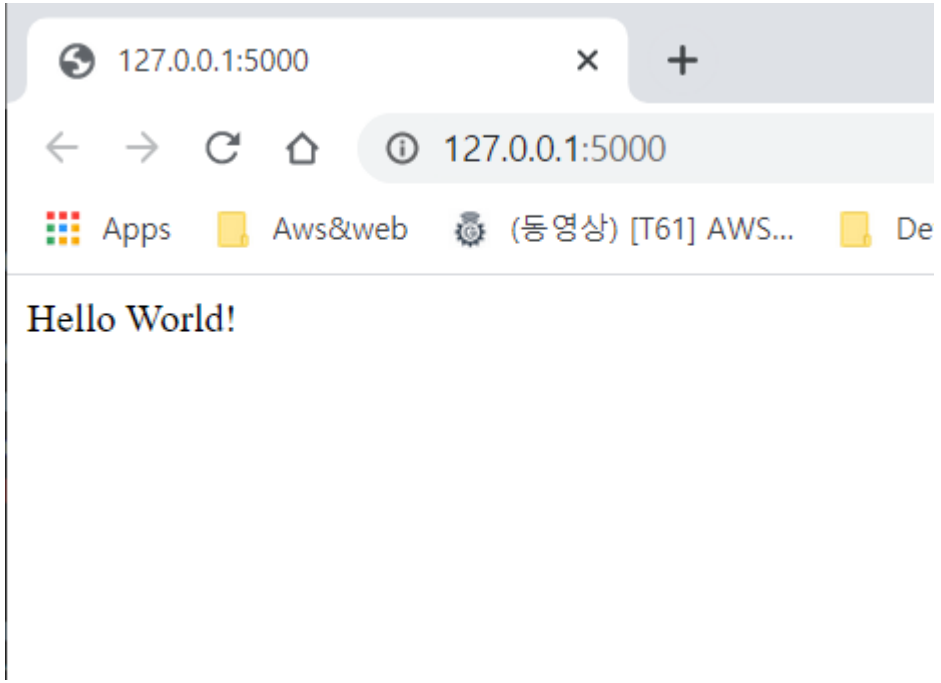
flask sql-alchemy 기본 동작 확인 2

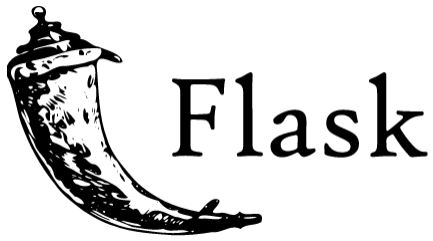
hello.html

```
Hello world!
```

실행 : 터미널 창에 아래의 내용을 입력한다.

```
set FLASK_APP=app
flask run
```





Flask를 활용한 웹서비스 구현

회원 관리 서비스

모델과 컨트롤러로 분리하기 – app.py

```
import os
from flask import Flask
from flask import render_template
from models import db

app = Flask(__name__)

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

if __name__ == "__main__":
    print('hello')
    basedir = os.path.abspath(os.path.dirname(__file__))
    print('basedir:{}'.format(basedir))
    dbfile = os.path.join(basedir, 'db.sqlite')
    print('file:{}'.format(dbfile))

    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///' + dbfile
    app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN'] = True
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

    db.init_app(app)
    db.app = app
    db.create_all()
    app.run(host='127.0.0.1', port = 5000, debug= True)
```

모델과 컨트롤러로 분리하기 – models.py

```python
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Fcuser(db.Model):
    __tablename__ = 'fcuser'
    id = db.Column(db.Integer, primary_key=True)
    password = db.Column(db.String(64))
    userid = db.Column(db.String(32))
    username = db.Column(db.String(8))
```

register 페이지 추가 – models.py

```
import os
from flask import Flask
from flask import render_template
from models import db


app = Flask(__name__)

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

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

if __name__ == "__main__":
    print('hello')
    basedir = os.path.abspath(os.path.dirname(__file__))
    print('basedir:{}'.format(basedir))
    dbfile = os.path.join(basedir, 'db.sqlite')
    print('file:{}'.format(dbfile))
```

추가

register 페이지 추가 – bootstrap 링크 가져오기

https://getbootstrap.com/docs/4.5/getting-started/introduction/

CSS

Copy-paste the stylesheet <link> into your <head> before all other stylesheets to load our CSS.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.1/css/bootstrap.min.css" int
```

Copy

JS

Many of our components require the use of JavaScript to function. Specifically, they require jQuery, Popper.js, and our own JavaScript plugins. Place the following <script>s near the end of your pages, right before the closing </body> tag, to enable them. jQuery must come first, then Popper.js, and then our JavaScript plugins.

We use jQuery's slim build, but the full version is also supported.

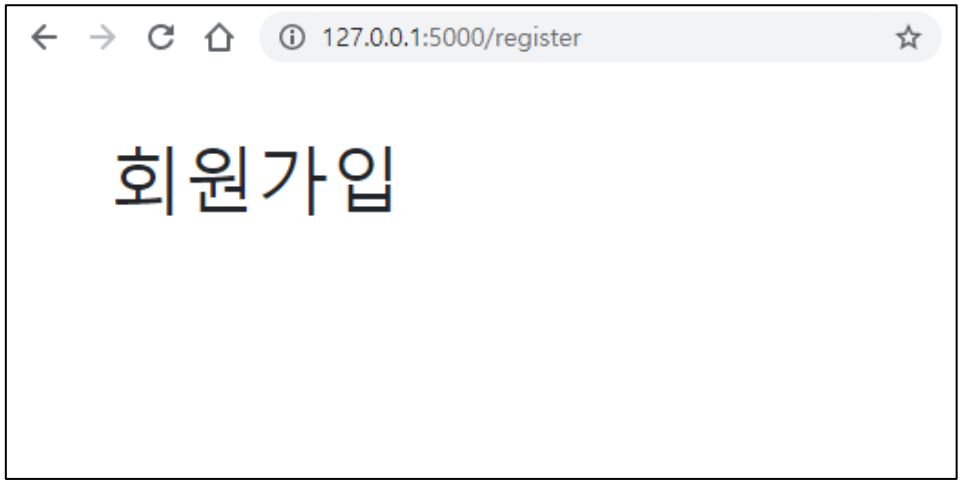
```
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/z
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-9/reF
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.1/js/bootstrap.min.js" integrity="sha384-XEe
```

Copy

register 페이지 추가 – register.html

```
<html>
<head>
    <meta charset='utf-8' />
    <meta name='viewport' content='width=device-width, initial-scale=1, shrink-to-fit=no' />

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" integrity="sha384-
9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk" crossorigin="anonymous">
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-
Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" integrity="sha384-
OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI" crossorigin="anonymous"></script>
</head>

<body>
    <div class="container">
        <div class="row mt-5">
            <h1>회원가입</h1>
        </div>
        <div class="row mt-5">
            <div class "col-12">
                <form method="POST" action=".">

                </form>
            </div>
        </div>
    </div>
</body>

</html>
```

register 페이지 추가 – models.py




```
set FLASK_APP=app
flask run
```

Html에 대해 기초적인 부분을 이해하고 싶다면 아래 페이지를 참조하자.

http://jun.hansung.ac.kr/CWP/htmls/HTML%20intro.html

register 페이지 추가 – models.py

```
<body>
    <div class="container">
        <div class="row mt-5">
            <h1>회원가입</h1>
        </div>
        <div class="row mt-5">
            <div class "col-12">
                <form method="POST" action=".">
```

추가

```
                    <div class="form-group">
                        <label for"userid">아이디</label>
                        <input type="text" class="form-control" id="userid" palceholer="아이디" name="userid" />
                    </div>
                    <div class="form-group">
                        <label for"username">사용자 이름</label>
                        <input type="text" class="form-control" id="username" palceholer="사용자 이름" name="username" />
                    </div>
                    <div class="form-group">
                        <label for"username">비밀번호</label>
                        <input type="text" class="form-control" id="password" palceholer="비밀번호" name="password" />
                    </div>
                    <div class="form-group">
                        <label for"username">비밀번호 확인</label>
                        <input type="text" class="form-control" id="re-password" palceholer="비밀번호 확인" name="re-password" />
                    </div>
                    <button type="submit" class="btn btn-primary">등록</button>
```

```
                </form>
            </div>
        </div>
    </div>
</body>
```

회원가입 입력 페이지 동작 구현

```
import os
from flask import Flask
from flask import request
from flask import redirect
from flask import render_template
from models import db

from models import Fcuser

app = Flask(__name__)
```

```
@app.route('/register', methods=['GET','POST'])
def register():
    if request.method=='POST':
        print(request.method)
        userid = request.form.get('userid')
        username = request.form.get('username')
        password = request.form.get('password')
        re_password = request.form.get('re-password')

        if (userid and username and password and re_password) and (password==re_password):
            fcuser = Fcuser()
            fcuser.userid = userid
            fcuser.username = username
            fcuser.password = password

            db.session.add(fcuser)
            db.session.commit()

            return redirect('/')

    return render_template('register.html')
```

데이터베이스 생성 및 호출

```
if __name__ == "__main__":
    basedir = os.path.abspath(os.path.dirname(__file__))
    dbfile = os.path.join(basedir, 'db.sqlite')

    # SQLITE Database uri address
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///' + dbfile

    # True로 설정하면 각 리퀘스트의 끝에 데이터베이스 변동사항을 자동 커밋한다.
    app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN']=True

    # 개체 수정을 추적하고 신호를 방출한다. 일반적으로 잘 사용하지 않는 기능이다.
    # 또한 추가 메모리를 필요로 하므로 False
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS']=False

    db.init_app(app)
    db.app = app
    db.create_all()
    app.run(host='127.0.0.1', port = 5000, debug= True)
```

CSRF 공격이란?

사이트 간 요청 위조(**C**ross-**S**ite **R**equest **F**orgery, **CSRF**, **XSRF**)는 웹사이트 취약점 공격의 하나로, 사용자가 자신의 의지와는 무관하게 공격자가 의도한 행위(수정, 삭제, 등록 등)를 특정 웹사이트에 요청하게 하는 공격을 말한다.

사이트 간 스크립팅(XSS)을 이용한 공격이 사용자가 특정 웹사이트를 신용하는 점을 노린 것이라면, **사이트간 요청 위조(CSRF)**는 특정 웹사이트가 사용자의 웹 브라우저를 신용하는 상태를 노린 것이다. 일단 사용자가 웹사이트에 로그인한 상태에서 사이트간 요청 위조 공격 코드가 삽입된 페이지를 열면, 공격 대상이 되는 웹사이트는 위조된 공격 명령이 믿을 수 있는 사용자로부터 발송된 것으로 판단하게 되어 공격에 노출된다.

대표적인 예) 유명 경매 사이트인 옥션에서 발생한 개인정보 유출 사건

관리자가 로그인을 한 상태이고, 자주 가는 사이트가 있었다. 공격자가 사이트에 악성스크립트를 넣은 글을 올렸고, 관리자가 그 글을 본 순간 아이디와 비밀번호가 노출되며, 해커는 관리자로 로그인을 하여 개인정보 데이터를 빼낸 것이다.

csrfProtect 추가와 form변경

```
import os
from flask import Flask
from flask import request
from flask import redirect
from flask import render_template
from models import db

from flask_wtf.csrf import CSRFProtect
from forms import RegisterForm
```

추가

```
@app.route('/register', methods=['GET','POST'])
def register():
    form = RegisterForm()
    if request.method=='POST':
        print(request.method)
        userid = request.form.get('userid')
        username = request.form.get('username')
        password = request.form.get('password')
        re_password = request.form.get('re-password')

        if (userid and username and password and re_password) and
(password==re_password):
            fcuser = Fcuser()
            fcuser.userid = userid
            fcuser.username = username
            fcuser.password = password

            db.session.add(fcuser)
            db.session.commit()

            return redirect('/')


    return render_template('register.html', form=form)
```

추가

csrfProtect추가와 form변경

```
if __name__ == "__main__":
    basedir = os.path.abspath(os.path.dirname(__file__))
    dbfile = os.path.join(basedir, 'db.sqlite')

    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///' + dbfile
   app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN'] = False
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

    app.config['SECRET_KEY'] = 'sdfsfsdretgf'
```

추가 임의의 문자열을 넣는다

```
    csrf = CSRFProtect()
    csrf.init_app(app)
```

추가

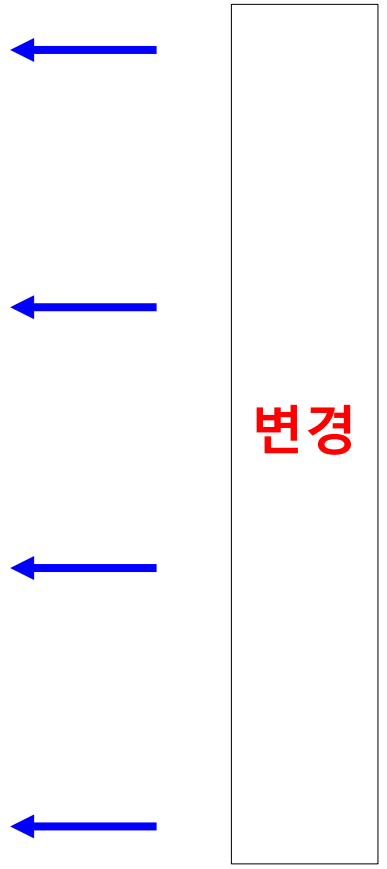
```
    db.init_app(app)
    db.app = app
    db.create_all()
    app.run(host='127.0.0.1', port = 5000, debug= True)
```

csrfProtect추가와 form변경

```
<form method="POST">
 {{ form.csrf_token }}
 <div class="form-group">
```

```
<label for="userid">아이디</label>
  <input type="text" class="form-control" id="userid" placeholder="아이디" name="userid" />
```

```
 {{ form.userid.label("아이디") }}
 {{ form.userid(class="form-control", placeholder="아이디")}}
</div>
<div class="form-group">
 {{ form.username.label("사용자 이름") }}
 {{ form.username(class="form-control", placeholder="사용자 이름")}}
</div>
<div class="form-group">
 {{ form.password.label("비밀번호") }}
 {{ form.password(class="form-control", placeholder="비밀번호")}}
</div>
<div class="form-group">
 {{ form.repassword.label("비밀번호 확인") }}
 {{ form.repassword(class="form-control", placeholder="비밀번호 확인")}}
</div>
```



forms.py 파일 추가 생성

```
from flask_wtf import FlaskForm
from wtforms import StringField
from wtforms import PasswordField
from wtforms.validators import DataRequired

class RegisterForm(FlaskForm):
    userid = StringField('userid', validators=[DataRequired()])
    username = StringField('username', validators=[DataRequired()])
    password = PasswordField('password', validators=[DataRequired()])
    repassword = PasswordField('repassword', validators=[DataRequired()])
```

app.py 파일 수정

```
@app.route('/register', methods=['GET','POST'])
def register():
    form = RegisterForm()

    if form.validate_on_submit():
        fcuser = Fcuser()
        fcuser.userid = form.data.get('userid')
        fcuser.username = form.data.get('username')
        fcuser.password = form.data.get('password')


        db.session.add(fcuser)
        db.session.commit()
        print('Success!')

        return redirect('/')
    else:
        print('retry!!')
    return render_template('register.html', form=form)
```

추가

forms.py 파일 추가 생성

```
from flask_wtf import FlaskForm
from wtforms import StringField
from wtforms import PasswordField
from wtforms.validators import DataRequired, EqualTo

class RegisterForm(FlaskForm):
    userid = StringField('userid', validators=[DataRequired()])
    username = StringField('username', validators=[DataRequired()])
    password = PasswordField('password', validators=[DataRequired(), EqualTo('repassword')])
    repassword = PasswordField('repassword', validators=[DataRequired()])
```