

Flask Tutorial을 시작하며

Contents

- 1. Flask 소개 (Intro)
- 2. 맘보기로 만드는 flask app
- 3. Flask 기본 구조 소개
- 4. Appendix List

이 책은 대표적인 Python 웹 프레임워크 중 하나인 Flask를 설명하고 이용할 수 있도록 하기 위해 작성되었습니다. 웹 프레임워크에 대해 전혀 모르더라도 튜토리얼을 따라오면 누구든지 사용할 수 있게 됩니다.

컴퓨터 언어, 그 중에서도 프로그래밍 언어를 배우는 목적이 무엇일까요? 아마도 자신이 원하는 기능을 구현(코딩)하여 컴퓨터에게 일을 시키고 싶기 때문입니다. 그렇다면 이런 질문을 하게 됩니다.

❶ 코딩에 대한 기본적 질문

“어떻게 내가 만든 기능이나 서비스를 사람들이 편리하게 사용하게 할 수 있을까?”

위 질문에 대한 답은 “사람들이 가장 많이 사용하는 방법을 이용해 제공하면 된다”입니다. 현재 사람들이 가장 많이 사용하는 방법은 웹(web)입니다. 거의 모든 사람들은 PC, 스마트폰과 같은 디바이스를 사용하고 있으며, 이런 디바이스는 웹 환경에서 정보를 검색하고, 글/사진/동영상을 공유하고, 자신이 가지고 있는 정보를 업로드합니다.

한가지 예를 들어 보겠습니다.

여러분은 동물 사진 한장을 입력하면 인공지능 기술을 활용해서 그 사진 속 동물이 무엇인지 알려주는 서비스를 구현(코딩)했습니다. 현재 여러분의 프로그램은 소스코드 상에서 이미지를 입력 받고 그 결과를 터미널(콘솔창, 도스창 등 까만 화면)에 출력합니다.

그렇다면 이런 기능을 사용하고 싶은 사람(예를 들면 여러분의 부모님, 친구 등)에게 소스코드를 전달(카톡, 이메일 등으로 전송)하면 받은 사람이 쉽게 사용할 수 있을까요? 아마도 답은 아니오일 것입니다. 여러분이 만든 소스코드는 개발자(프로그래머)가 컴퓨터에게 일을 시키기 위한 언어로 작성되어 있기 때문에 컴퓨터 언어(programming language)를 모르는 사람들은 그 내용이 뭔지 관심도 없고, 설령 소스코드를 열어본다고 한들 웹 외계어만 잔뜩 들어 있다고 생각할 것입니다.

여러분이 어렵게 만든 세부적인 작동 절차나 원리는 서버 속에 숨기고 사용자(부모님, 친구, 미지의 접속자 등)에게는 웹 브라우저(크롬, 엣지, 사파리 등)로 접속해서 클릭, 파일 업로드 등)의 단순 작업만으로 결과를 제공하는 것이 서로에게 좋습니다.

사용자는 친숙한 웹/앱 환경(브라우저)으로 접속하여 정보를 조회하거나 요청(request)하면, 그 서비스를 담당하는 서버(server)에서 여러분이 만든 소스코드를 작동시켜서 나온 결과를 사용자가 현재 보고 있는 브라우저에 뿌려주면(response)하면 됩니다. 이런 식으로 작동하는 것을 client-server 아키텍처라고 합니다. 웹 기반으로 서비스를 제공하는 client-server 아키텍처를 쉽게 개발할 수 있도록 도와주는 것이 웹 개발 프레임워크입니다.

과거에는 웹 개발을 하려면 네트워크, 데이터베이스, 프런트 기술(HTML, CSS, JavaScript), 백엔드 기술(각종 programming language), 보안 기술, 미들웨어, 웹서버 기술 등 모든 지식을 모두 학습해야 하는 지옥과 같은 상황이었습니다.

그러나 최근에는 웹 개발에 필요하지만 공통적으로 적용되는 기술은 미리 구현해 놓고, 개발자는 새로운 서비스 개발에 집중하도록 도와주는 훌륭한 프레임워크(framework)가 많이 등장했습니다. 과거에 오랜시간 동안 학습/실습을 통해 성장한 전문 개발자들이 하던 일을 최근에는 간단한 학습만으로 구현할 수 있는 상황이 되었습니다.

웹 개발 프레임워크의 종류는 많습니다. 그 중에 Python을 이용한 웹 개발 프레임워크도 있습니다. 물론 다른 언어를 기반으로 하는 웹 개발 프레임워크도 많습니다. Python의 편리함과 방대한 라이브러리를 강점으로 Python 사용자가 급격히 증가하였습니다. 우리는 웹 개발 프레임워크 중 Python을 기반으로 제작되는 웹 개발 프레임워크를 공부할 것입니다.

Python 웹 개발 프레임워크는 Flask와 Django가 대표적입니다. 우리는 간단히 배워서 쉽게 적용할 수 있는 Flask에 대하여 학습할 것입니다. 관심있는 독자들은 Django도 같이 학습할 것을 추천합니다.



Fig. 1 Flask 로고

이 책의 저자는 파이썬을 좋아하고 즐겨 사용하는 청주대학교 인공지능소프트웨어전공 교수입니다.

① 저자소개

청주대학교 소프트웨어융합학부 인공지능소프트웨어전공

노기섭 교수

Contact

- E-mail: kafa46@cju.ac.kr
- Phone: 043-229-8496 (유선)
- Mobile: Not open to public (private, 비공개)



Fig. 2 청주대 노기섭 교수

1. Flask 소개 (Intro)

1.1. Flask 초간단 소개

플라스크 둘러보기를 통해 개략적으로 Flask를 이해하는 시간을 갖도록 하겠습니다.

플라스크는 기본적으로 Python 언어를 사용한 웹 프레임워크입니다. Python 웹 프레임워크는 Django(장고)와 Flask(플라스크)가 있습니다. Flask는 마이크로 프레임워크를 지향하고 있습니다.

② 마이크로 프레임워크란?

마이크로 프레임워크 (Micro Framework)는 소규모 웹 애플리케이션 프레임워크를 가리키기 위한 용어입니다. 개발에 필요한 대부분의 기능을 미리 제공하는 풀 스택 프레임워크 (Full-stack Framework)와 대조되는 용어입니다. 마이크로 프레임워크는 개발자에게 핵심적으로 필요한 기능만 제공하고 나머지 부분들은 개발자의 몫으로 남겨둡니다. 다음과 같은 내용들은 제공되지 않기 때문에 개발자가 책임을 지고 구현해야 합니다.

- 계정, 인증, 인가, 역할 등
- 객체 지향 매핑을 통한 데이터베이스 추상화
- 입력 확인 및 입력 정제
- 웹 템플릿 엔진

위와 같은 사항을 프레임워크가 지원하지 않으므로 개발자는 구현해도 되고 안해도 됩니다. 그만큼 구현해야 할 범위가 작아져서 빠르게 개발할 수 있습니다. 하지만 위 기능이 필요한 시스템이라면 개발자가 스스로 구현해야 합니다.

Flask와 Django는 다음과 같은 차이점이 있습니다. 우리는 Flask에 한정해서 공부해 볼 것입니다.

Framework	Flask	Django
출시년도	2010	2005
ORM 지원	X	O
아키텍처	MSA(Micro Service Architecture)	Monolithic
지원기능	적음	많음
초기 학습량	적은 학습으로 사용 가능	상대적으로 배워야 할 것이 많음
소스코드 양	코딩량이 상대적으로 적음	Flask보다 소스코드 크기가 큼(무거움)
개발 자유도	높음	낮음
개발자 책임	높음	낮음(프레임워크)에서 대부분 지원

ORM(Object Relational Mapping)은 프레임워크에서 데이터베이스를 객체로 관리할 수 있도록 지원하는 기능입니다. ORM을 사용하면 별도로 공부하지 않아도 편리하게 데이터베이스를 조작할 수 있습니다.

MSA와 Monolithic에 대한 구체적 설명은 여기를 참고하기 바랍니다.

1.2. 어떤 프레임워크를 더 많이 쓸까요?

Django와 Flask의 인기는 github.com에서 좋아요(star) 숫자를 보면 대충 짐작할 수 있습니다. 아래 바로가기 이미지를 클릭하여 [Django Github](#) 또는 [Flask Github](#) 페이지로 이동하여 'Star' 숫자를 확인해 보세요.

2022년 1월 기준으로 Github의 좋아요(stars)와 퍼가기(Fork) 숫자는 다음과 같습니다.

Framework	Flask	Django
좋아요(Stars)	57,600개	61,600개
복제횟수(Forks)	14,800회	26,300회

위 표만 보면 Django가 보다 많은 좋아요/복제가 발생한 것으로 보아 좀 더 많이 쓰는 것처럼 보입니다. Django의 나이가 더 많으니 어찌보면 당연해 보이기도 합니다. 하지만 짧은 기간에도 불구하고 많은 사람들이 Flask를 사용하고 있습니다. 하지만 단순히 숫자만 가지고 어떤 프레임워크가 좋다고 말하는 것은 바보같은 짓입니다. 각각의 프레임워크는 나름대로의 장단점이 있기 때문입니다.

❶ Git의 기능: fork, clone

- **fork**는 다른 사람이 만든 Git 저장소(repository, 이하 repo)를 내 repo로 복제하는 기능입니다. 원본 repo (내가 복사해온 repo 주인)와 나의 repo는 서로 연결되어 있습니다.
 - 원본 repo가 변경 \(\to\) 내 repo로 반영: `fetch`, `pull` 사용
 - 내 repo가 변경 \(\to\) 원본 repo로 반영: `push` 사용, 원본 repo 주인이 승인해 주면 반영
- **clone**은 내가 만든 원격 저장소(repo) 또는 **fork**를 통해 내 원격 repo로 복제한 것을 로컬 컴퓨터로 복사하는 것입니다.
 - 내 컴퓨터에서 작업 \(\to\) 원격 repo로 전송: `add`, `commit`, `push` 사용
 - **fork**한 경우 `push` 내 원격 repo만 업데이트 됩니다.
 - **fork** 해온 원본 repo에 내 원격 repo의 변경사항을 반영하기 위해서는 별도의 과정을 거쳐야 합니다.
 - 내 원격 repo 변경을 원본 repo에 반영하는 절차는 [이 블로그](#)를 참고하세요.

Flask는 지원 기능이 적기 때문에 필요한 기능을 구현할 때마다 별도의 라이브러리를 설치해야 합니다. 그리고 라이브러리를 설치하면 그 때마다 Flask와 연결하는 **binding** 작업을 해주어야 합니다. Flask는 빠르고 가볍게 구현할 수 있지만 기능이 늘어날 수록 개발에 필요한 **cost (비용)**도 같이 증가합니다. 간단한 시스템을 구현할 때는 flask를 사용하는 것이 답일 수 있지만 복잡하거나 **커스터마이징(customizing)**이 많이 필요한 시스템에는 적당하지 않습니다.

다음은 Python과 Web Framework에 대한 2021년도 Survey 결과입니다. 구체적인 내용은 [Jetbrain](#) 홈페이지를 참고하기 바랍니다.

What do you use Python for?

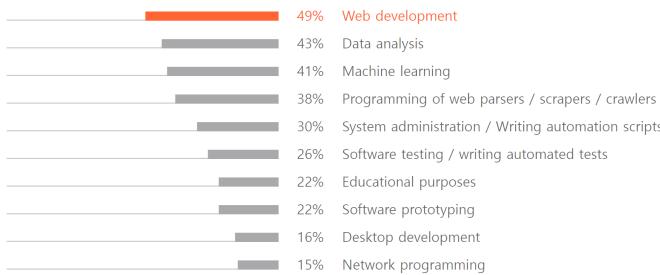


Fig. 1.1 Python 사용의 목적

What web frameworks or libraries do you use in addition to Python?

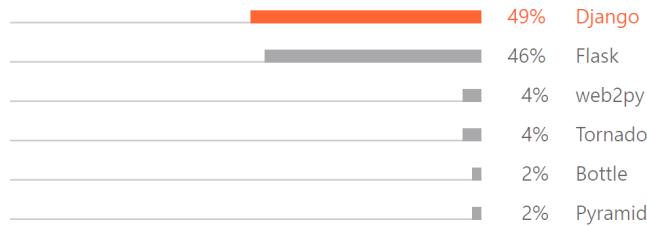


Fig. 1.2 Python 개발자의 웹 프레임워크 사용 비율

1.3. Flask를 위한 사전 준비 목록

Flask를 실행하기 위해서는 몇 가지 사전 준비사항이 필요합니다. 몇 가지 해주어야 할 일들 (To Do List)는 다음과 같습니다.

- VS Code 소개 및 설치
- 가상환경 소개 및 `anaconda`를 이용한 가상환경
- `flask` 설치
- `.gitignore` 세팅
- 의존성 파일 `requirement.txt` 생성

위 사항을 이미 알고 있는 사람들은 생략하고 곧바로 `flask` 관련 Chapter로 이동해도 무방합니다.

4가지 사전 준비 작업에 대하여 차근차근 설명하도록 하겠습니다.

준비 되셨나요?

1.4. 준비 1. VS Code 소개 및 설치

1.4.1. VS Code 소개

Python 언어를 이용해 시스템이나 서비스를 개발하기 위해서는 다양한 기능을 제공하는 **편집기(editor)**를 사용하는 것이 여러모로 편리합니다. 불필요한 시간낭비(프로그래머들은 종종 '삽질'이라고 부르기도 함)를 줄여줄 수 있습니다. 개발자들이 주로 사용하는 편집기(editor) 목록은 다음과 같습니다. 아래 목록은 [Stack Overflow](#)에서 2021년에 실시한 [Survey 결과](#)를 참조하였습니다.

위에서 열거한 편집기 중 Python을 지원하는 편집기라면 어떤 것을 선택하더라도 상관 없습니다. 본인이 평소에 자주 사용하여 친숙한 편집기가 있다면 그대로 사용해도 무방합니다. 각 편집기별 특징을 정리하면 아래 표와 같습니다.

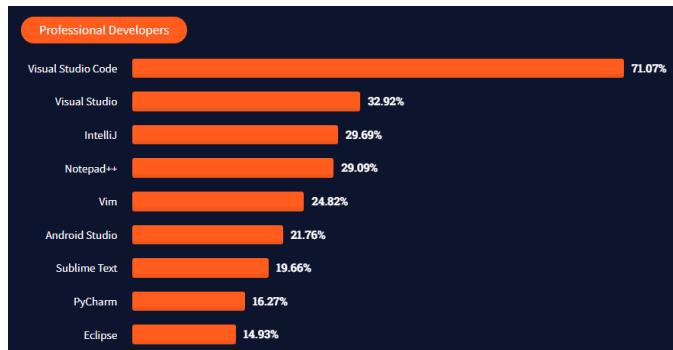


Fig. 1.3 전세계 개발자들이 선호하는 에디터(Stack Overflow 2021 조사)

이전에 본인이 자주 사용하여 익숙한 편집기가 있다면 VS code를 사용하지 않아도 됩니다. 하지만 새로운 편집기를 사용해 보고 싶거나, 프로그래밍 입문 단계에서 편집기를 하나 선택해야 한다면 VS Code를 추천합니다. VS Code는 무료이지만 거의 모든 언어를 지원하고, 개발자들의 필수품인 Git을 지원할 뿐만 아니라, 확장(extension) 프로그램이 매우 다양합니다. 여러분들이 나중에 취업하게 되면 회사에서 VS Code를 사용하고 있을 확률도 높겠죠? 대부분의 프로그래머가 사용하는 도구는 나름대로 이유가 있기 때문입니다.

본 tutorial에서도 VS Code를 사용할 것입니다.

1.4.2. VS Code 설치

VS Code 설치를 안내하는 다양한 블로그가 있습니다. 몇 개 블로그를 소개합니다. 아래 목록 중 본인이 보기 편한 블로그를 선택해서 VS Code를 설치하가 바랍니다.

- 한글 블로그1: [Visual Studio Code 설치하는 방법\(Windows / Ubuntu\)](#)
- 한글 블로그2: [\[Windows 10\] Visual Studio Code 설치하는 방법](#)
- 영어 블로그: [Visual Studio 공식 문서](#)

아마도 대부분의 독자들이 가지고 있는 PC나 노트북은 윈도우 운영체제를 쓰고 있을 것입니다. 여러분들은 윈도우 운영체제에 VS Code를 설치해도 무방합니다.

윈도우에 설치된 VS Code를 실행시키고 리눅스 서버에 접속하여 flask 학습을 계속 진행할 것입니다. 윈도우에 VS Code를 설치할 경우 위 블로그를 참고하여 설치합니다. 설치 완료 후, VS Code를 설치된 폴더에 들어가서 실행파일을 더블클릭하면 VS Code가 실행됩니다.

본 튜토리얼은 리눅스(Ubuntu) 환경을 기본으로 작성되었습니다. 대부분의 명령어나 기능이 윈도우 환경에서도 작동하겠지만, 혹시 안되는 경우가 있을 수 있습니다. 가능하면 가상머신(Virtual Machine), 아마존 AWS 서버 등을 이용해 리눅스 환경에서 실습하기를 추천합니다.

Note

리눅스 환경을 기본으로 실습하는 이유는 대부분의 백엔드(서버 쪽) 개발이나 구현은 리눅스를 사용하기 때문입니다. 다양한 리눅스 종류 중에서 우분투(Ubuntu)를 사용하는 이유는 Ubuntu 리눅스의 사용자가 전세계적으로 가장 많기 때문입니다.

여러분들이 만약 리눅스 운영체제를 사용하는 독자라면 아래와 같이 VS Code를 설치합니다.

리눅스 중에서 우분투(Ubuntu) 리눅스를 사용자가 가장 많으므로, 리눅스의 경우 우분투를 기준으로 설명하겠습니다. 페도라(Fedora) 또는 센토스(CentOS) 리눅스를 사용하는 독자들은 개별적으로 구글링이나 블로그 검색 등을 통해 확인하고 설치하기 바랍니다.

우분투를 환경에서 VS Code 설치는 매우 간단합니다.

- Terminal 창을 하나 엽니다. (GUI 환경에서 단축키: **CTL + ALT + T**)
- 아래와 같이 명령어를 순서대로 입력합니다. (코드를 복사하여 붙여넣기 하면 편합니다.)
 - Ubuntu에 새로운 프로그램을 설치할 경우 관리자 권한(sudo)이 있어야 합니다.

방법 1. Ubuntu에 기본 프로그램으로 설치된 snap 패키지 이용

```
sudo snap install --classic code
```

방법 2. Ubuntu의 소프트웨어 관리자(apt) 이용

- 설치에 필요한 추가 curl 패키지를 설치합니다.

```
sudo apt-get install curl
```

- 마이크로소프트 GPG 키를 다운로드하여 /etc/apt/trusted.gpg.d/ 경로에 복사합니다.
 - 설치 방법은 [\[Ubuntu\] VS Code 설치](#) 블로그를 참고하여 작성하였습니다.
 - 아래 코드는 curl을 이용해 받아온 공개키를 pipe 연산자 (`|`)와 redirection 연산자 (`>`)를 활용해 공개키를 등록하는 코드입니다.
 - [curl, pipe, redirection](#)에 대하여 추가로 공부하는게 귀찮다면 코드를 그대로 복사/붙여넣기 하면 됩니다.

```
sudo sh -c 'curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > /etc/apt/trusted.gpg.d/microsoft.gpg'
```

- Visual Studio Code를 다운로드 받기 위한 저장소를 추가합니다.

```
sudo sh -c 'echo "deb [arch=amd64] https://packages.microsoft.com/repos/vscode stable main" > /etc/apt/sources.list.d/vscode.list'
```

- 추가한 저장소를 최신화 합니다.

```
sudo apt update
```

- VS Code를 설치합니다.

```
sudo apt install code
```

- VS Code를 실행합니다.

```
code .
```

1.4.3. VS Code를 이용해 작업할 공간으로 이동

먼저 VS Code를 실행합니다. VS Code를 처음 실행한 화면입니다.

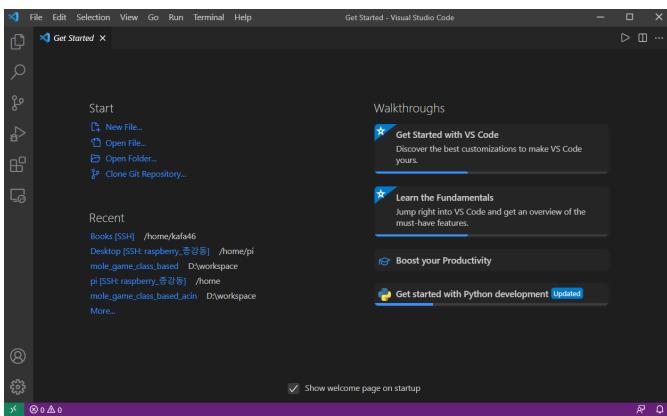


Fig. 1.4 VS Code 시작 페이지

리눅스가 설치된 서버에 원격 접속 합니다.

Note

본 튜토리얼은 여러분이 원격으로 접속할 서버를 가지고 있다고 가정합니다. 다음과 같은 이유로 서버 기반을 가정으로 튜토리얼을 진행하게 되었습니다.

- 서버가 있어야 `flask`를 통해 웹 서비스를 365일/24시간 서비스할 수 있습니다.
- 개인 컴퓨터(로컬 PC)에서 웹 시스템을 구축하더라도 최종적으로 서버를 통해 배포해야 합니다.
- 여러분들이 서버를 활용한 원격 작업에 친숙해지도록 하는 목적으로 있습니다.

원격 접속 세팅은 [여기](#) 블로그를 참고하여 설정할 수 있습니다. 간단한 세팅으로 구현할 수 있으니 개인적으로 실습하고 이 책에서는 구체적인 세팅 방법을 생략하겠습니다.

만약 원격 접속할 리눅스 서버가 없다면 본인의 로컬 서버에서 바록 작업해도 괜찮습니다. 개인 컴퓨터(로컬 PC)에서 작업한 파일들을 나중에 서버로 올리는 작업을 같이 해보도록 할 것입니다.

리눅스 서버에 `ssh` 원격 접속하기 위하여 \(\textcircled{1}\) `Remote Explorer` 아이콘을 누릅니다.

여러분이 등록해 놓은 원격 서버 목록이 \(\textcircled{2}\) 와 같이 보입니다. 위 그림에서 보이는 목록은 저자가 임의로 생성한 목록입니다.

여러분의 경우는 각자 등록한 서버 이름이 보일 것입니다(서버 이름을 별도로 지정하지 않았다면 IP 주소가 보일 수도 있습니다.) 원하는 서버 이름에 마우스를 가져가면 모니터 모양에 더하기(+) 표시가 있는 아이콘이 나타납니다. 그 아이콘을 클릭하세요.

아이콘을 클릭하면 새로운 VS Code 창이 뜨면서 \(\textcircled{3}\) 과 같이 우리가 접속하고자 하는 서버에 본인의 비밀번호를 입력하는 창이 나옵니다. 개인별로 자신의 비밀번호를 입력하고 엔터(`enter`)키를 칩니다.

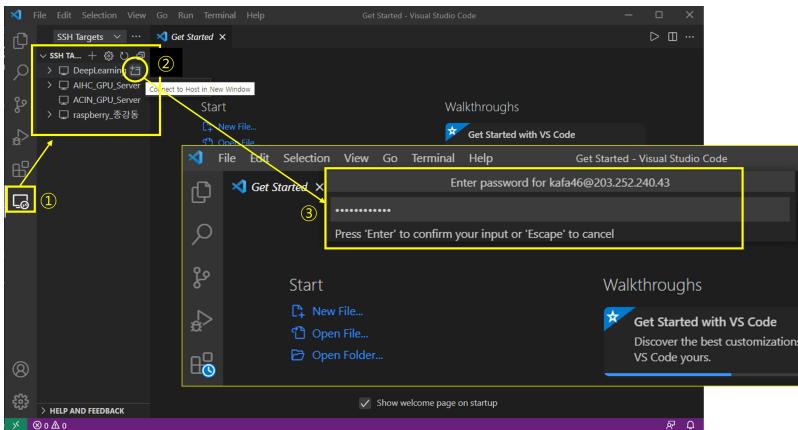


Fig. 1.5 VS Code에서 제공하는 [Remote-SSH](#)를 이용하여 원격 서버에 접속

서버 접속이 성공하면, 본인이 작업(코딩)하고 싶은 디렉토리로 이동합니다.

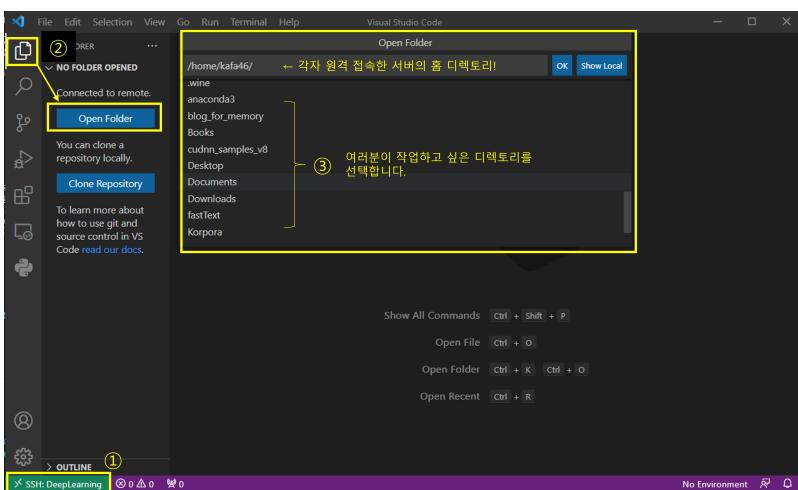


Fig. 1.6 원격 서버 접속 확인 후 본인이 작업(코딩)할 디렉토리 선택

Note

GUI (Graphic User Interface) 환경에서는 [폴더\(folder\)](#)라는 용어를 주로 사용하고, CLI (Command Line Interface) 환경에서는 [디렉토리\(directory\)](#)라는 표현을 사용합니다. 윈도우 10과 같은 운영체제는 GUI 환경이기 때문에 폴더라는 표현을 합니다. 서버기반에서 터미널(terminal)로 작업하는 경우는 디렉토리라는 말을 합니다.

- 폴더와 디렉토리는 같은 개념입니다.
- 실제로 혼용해서 사용하는 경우도 많습니다.

만약 원하는 폴더가 없다면 다음 그림과 같이 VS Code에서 쉽게 폴더를 새로 만들 수 있습니다.

- VS Code 파일 탐색기 위쪽에 현재 작업 디렉토리 이름이 보입니다. 저는 [Lecturers](#)라는 디렉토리 이름이 보입니다. 여러분은 각자 접속한 디렉토리 이름이 보일 것입니다. 이를 위로 마우스를 가져가면 관련된 아이콘이 몇 개 나타납니다.
- 그 중에서 폴더 모양에 더하기(+) 표시가 된 [New Folder](#) 아이콘을 클릭합니다.
- [New Folder](#) 아이콘을 클릭하면 폴더 이름을 입력할 수 있는 작은 입력창이 뜹니다. 거기에 여러분이 원하는 폴더 이름을 입력하고 엔터([enter](#))키를 치세요.
- VS Code 파일 탐색기 (화면 왼쪽)에 여러분이 생성한 폴더가 나타납니다. 클릭하고 들어가세요.
- 저는 [홈 디렉토리](#) 아래에 [~/Lectures/Flask_tutorial/](#)이라는 디렉토리를 작업 폴더로 선택했습니다.

만약 새로운 파일을 만들고 싶다면 해당 VS Code 파일 탐색기 영역으로 이동합니다.

파일을 만들고 싶은 디렉토리 이름에서 마우스 오른쪽 클릭하여 [New File](#) 메뉴를 선택하거나, 파일 모양 아이콘 모양에 더하기(+) 표시가 된 아이콘을 클릭합니다. 폴더를 생성하는 것과 마찬가지로 이름을 입력하는 작은 창에 원하는 파일 이름을 입력하면 원하는 디렉토리에 파일을 생성할 수 있습니다.

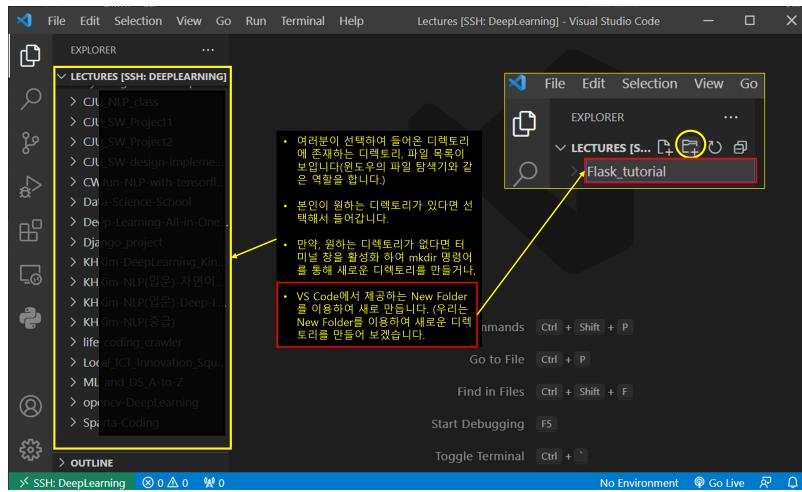


Fig. 1.7 원격 서버 접속 확인 후 본인이 작업(코딩)할 디렉토리 선택

글로 설명하면 복잡해 보이는데요. 한 번만 실습해 보면 너무나 쉽게 해결할 수 있습니다. 각자 서버에 접속해서 원하는 디렉토리나 파일을 만들어 보세요. 필요없을 경우는 마우스 오른쪽 클릭하여 삭제하면 되니 부담갖지 말고 실습해 보기 바랍니다.

1.5. 준비 2. 가상환경 및 anaconda

1.5.1. 가상환경 소개

서비스나 프로그램을 개발할 경우 가상환경(virtual environment)를 사용하는 것은 필수적입니다. Python은 매우 다양한 패키지를 지원한다. 새롭게 Python 프로젝트를 진행할 때 개발에 필요한 패키지가 있다면 자신의 컴퓨터에 추가로 설치해야 합니다. 패키지를 설치할 때마다 기존에 설치된 패키지를 업데이트하거나, 새롭게 설치해야 합니다.

그러나 이 과정에서 과거에 개발한 프로젝트에서 사용하던 패키지를 새로운 패키지 개발을 위해 업그레이드 할 경우 과거 프로젝트에서 더 이상 작동하지 않는 경우가 많이 발생하게 됩니다. 과거 프로젝트가 돌아가게 하려고 업데이트한 패키지를 다운그레이드 하면 새로운 패키지가 작동하지 않고, 업데이트를 하지 않으면 새로운 프로젝트 개발이 어렵습니다.

이런 상황을 **의존성(dependency)** 문제라고 부릅니다.

가상 환경은 개발 환경을 독립적으로 분할하여 개발에 필요한 패키지를 추가로 설치할 경우 기존 환경과의 의존성(dependency) 문제를 완벽하게 해결할 수 있는 방법이다. Python 개발 과정에서 사용할 수 있는 가상환경 도구는 다양합니다. 대표적인 가상환경 도구는 다음과 같습니다. 참고로 우리는 **anaconda**를 이용할 것입니다.

1.5.1.1. Anaconda

Anaconda: 아나콘다는 기본적으로 Python 및 R 언어 패키지 배포판입니다.

■ 위키백과: 아나콘다

아나콘다(Anaconda)는 패키지 관리와 디플로이를 단순화 할 목적으로 과학 계산(데이터 과학, 기계 학습 애플리케이션, 대규모 데이터 처리, 예측 분석 등)을 위한 파이썬과 R 프로그래밍 언어의 자유-오픈 소스[5] 배포판이다. 패키지 버전들은 패키지 관리 시스템 **conda**를 통해 관리된다. 아나콘다 배포판은 1300만 명 이상의 사용자들이 사용하며 윈도우, 리눅스, macOS에 적합한 1,400개 이상의 유명 데이터 과학 패키지가 포함되어 있다. (출처: 온라인 위키 [아나콘다 파이썬 배포판](#))

Anaconda 패키지 배포판에는 수많은 과학, 머신러닝, 데이터처리 등을 쉽게 해주는 패키지들이 포함되어 있습니다. 최신 인공지능, 빅데이터 관련 서비스를 개발에 많이 사용되고 있습니다.

Anaconda에는 수많은 Python 패키지들이 들어 있기 때문에 패키지를 관리해주는 툴(관리자)가 필요합니다. **conda**는 **anaconda**에 포함된 패키지들을 쉽게 설치, 삭제, 업데이트 등 작업을 도와주는 **패키지 관리자**입니다. **Anaconda**의 수많은 패키지들 때문에 의존성 문제가 발생할 수 밖에 없습니다. 이를 해결하기 위해 **conda**는 가상환경을 지원하고 있습니다.

Note

우리는 **anaconda**의 패키지 관리자인 **conda**를 이용하여 가상환경을 생성하고 이용할 것입니다.

물론 여러분들이 지금까지 사용하던 가상환경 관리자가 있다면 그대로 사용해도 무방합니다.

만약, 새로 가상환경에 대한 공부를 시작한다면 가장 많은 사용자를 확보하고 있으며 빠르게 성장하고 있는 **anaconda**로 시작하는 것을 추천합니다.

1.5.1.2. **virtualenv**

virtualenv: **pyenv**는 **pip**(또는 **pip3**)를 이용하여 설치한 이후 사용할 수 있는 가상환경 도구입니다. **virtualenv** 패키지는 다양한 버전의 패키지를 효율적으로 관리하고자 하는 경우 널리 사용되고 있습니다. 파이썬 버전을 쉽게 관리할 수 있는 장점이 있는 것으로 알려져 있습니다. **virtualenv**의 설치 및 사용법에 대한 자세한 내용은 [여기](#)를 참고하세요. 주요 사용법은 다음과 같습니다.

```
# virtualenv 설치  
pip3 install virtualenv  
  
# 가상환경 만들기(생성하기)  
virtualenv [가상환경 이름]  
  
# 가상환경 들어가기(실행하기)  
[가상환경 이름]\Scripts\activate # 윈도우 환경  
source [가상환경 이름]/bin/activate # 리눅스 환경  
  
# 가상환경 빠져나오기(종료하기)  
deactivate
```

1.5.1.3. **venv**

venv: [PEP 405 – Python Virtual Environments](#)으로 제안된 Python 기본 탑재 가상환경 도구입니다. 즉 Python 표준 라이브러리로 제공되기 때문에 별도의 설치과정 없이 언제든 사용 가능합니다. **venv** 설치 및 사용법에 대한 자세한 내용은 [여기](#)를 참고하세요. 주요 사용법은 다음과 같습니다.

```
# 가상환경 만들기(생성하기)  
python3 -m venv [가상환경 이름]  
  
# 가상환경 들어가기(실행하기)  
[가상환경 이름]\Scripts\activate.bat # 윈도의 환경  
source [가상환경 이름]/bin/activate # 리눅스 환경  
  
# 가상환경 빠져나오기(종료하기)  
deactivate
```

1.5.2. **Anaconda**를 이용한 가상환경

1.5.2.1. 가상환경 설치

Anacoda 설치는 운영체제에 따라 약간 다릅니다. 아래 블로그를 이용하여 설치하면 됩니다. 아나콘다 설치는 각자 개인별로 설치하는 것으로 하고 별도의 설명은 아래 링크로 대체합니다.

- **Anaconda** 윈도우 설치(한글): [Anaconda 설치: Windows 10](#)
- **Anaconda** 리눅스 설치(한글): [리눅스/Linux Python 설치하기 Anaconda](#)

Anaconda 설치 위치는 설치 중 자신이 선택한 경로입니다. 기본 설치를 따라 계속 클릭만 했다면,

윈도우의 anaconda 설치 위치는 `C:\Users\사용자아이디\anaconda3\` 입니다.

리눅스의 anaconda 설치 위치는 `~/anaconda3/` 입니다. 참고로 경로명에서 `~` 의미는 현재 접속한 아이디의 홈 디렉토리 `/home/`와 같습니다.

1.5.2.1.1. 기본(base) 가상환경 확인하기

Anaconda 가상환경은 리눅스(우분투)를 기준으로 설명하겠습니다.

Anaconda 설치가 끝나면 anaconda의 기본 가상환경으로 들어갑니다. 아래와 같이 현재 경로에서 `conda activate` 명령어를 치면, 현재 경로 앞에 `(base)`가 붙는 것을 확인할 수 있습니다. `base`는 anaconda가 기본적으로 제공하는 가상환경입니다.

```
current/path$ conda activate  
(base) current/path$
```

`base` 가상환경에서 빠져나오려면 `conda deactivate` 명령어를 치면 원래 본인의 컴퓨터 환경으로 돌아옵니다. 현재 경로 앞에 붙어 있던 `(base)` 표시가 사라지고 현재 경로만 나타나면 가상환경에서 잘 빠져나온 것입니다.

```
(base) current/path$ conda deactivate  
current/path$
```

1.5.2.1.2. 가상환경 새로 만들기

이제 VS Code에서 제공하는 Terminal을 이용하여 가상환경을 만들어 보겠습니다.

- 먼저 VS Code를 실행합니다. VS Code를 처음 실행한 화면입니다.

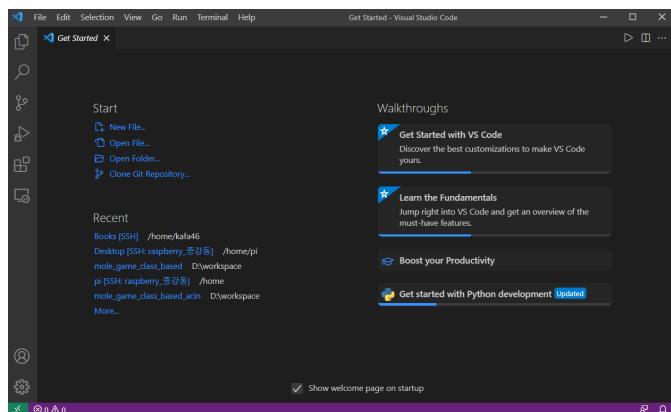


Fig. 1.8 VS Code 시작 페이지

- 리눅스가 설치된 서버에 원격 접속 합니다.

- 원격 접속 세팅은 [여기](#) 블로그를 참고하여 설정할 수 있습니다. 간단한 세팅으로 구현할 수 있으니 개인적으로 실습하고 이 책에서는 구체적인 세팅 방법을 생략하겠습니다.
- 만약 원격 접속할 리눅스 서버가 없다면 본인의 로컬 서버에서 바록 작업해도 괜찮습니다.

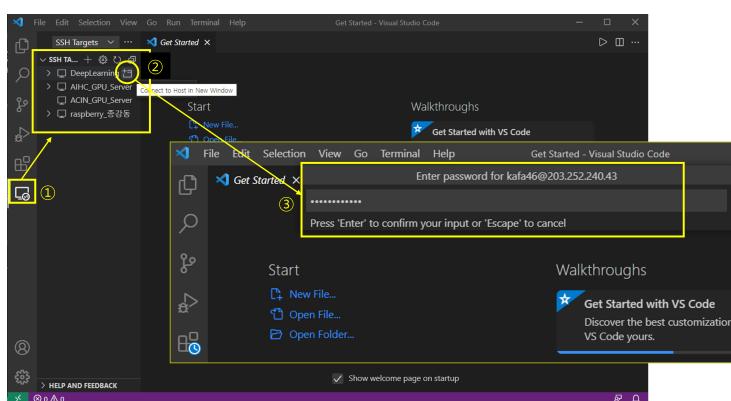


Fig. 1.9 VS Code에서 제공하는 Remote-SSH를 이용하여 원격 서버에 접속

- 리눅스 서버에 ssh 원격 접속하기 위하여 `\(\textcircled{1}\)` `Remote Explorer` 아이콘을 누릅니다.
- 여러분이 등록해 놓은 원격 서버 목록이 `\(\textcircled{2}\)` 와 같이 보입니다. 위 그림에서 보이는 목록은 저자가 임의로 생성한 목록입니다. 여러분의 경우는 각자 등록한 서버 이름이 보일 것입니다(서버 이름을 별도로 지정하지 않았다면 IP 주소가 보일 수도 있습니다.) 원하는 서버 이름에 마우스를 가져가면 모니터 모양에 더하기(+) 표시가 있는 아이콘이 나타납니다. 그 아이콘을 클릭하세요.
- 아이콘을 클릭하면 새로운 VS Code 창이 뜨면서 `\(\textcircled{3}\)` 과 같이 우리가 접속하고자 하는 서버에 본인의 비밀번호를 입력하는 창이 나옵니다. 개인별로 자신의 비밀번호를 입력하고 엔터(`enter`)키를 칩니다.

- 터미널 창을 활성화 시킵니다. VS Code 메뉴 중에서 \(\backslash(\text{to}\)\) 보기(View) \(\backslash(\text{to}\)\) 터미널(Terminal)을 선택하거나, 단축키 **CTL + `**를 동시에 누릅니다. `는 작은 따옴표가 아니라 키보드의 탭(tab)키 위에, 숫자 1 원쪽에 있는 **backtick(`)** 키입니다.

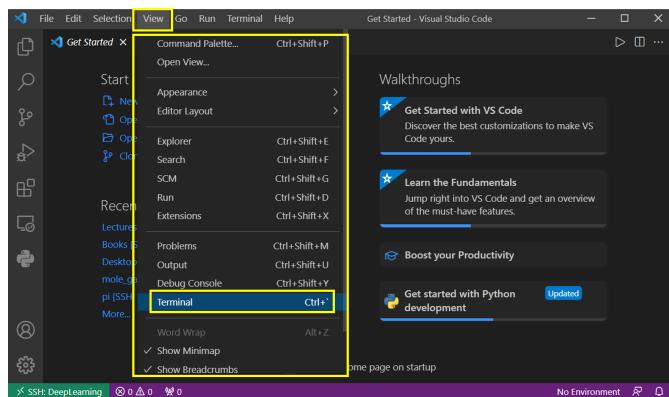


Fig. 1.10 VS Code를 이용해서 터미널을 활성화 시키는 방법

- VS Code에서 터미널 창(윈도우)을 활성화 한 화면은 다음 그림과 같습니다.

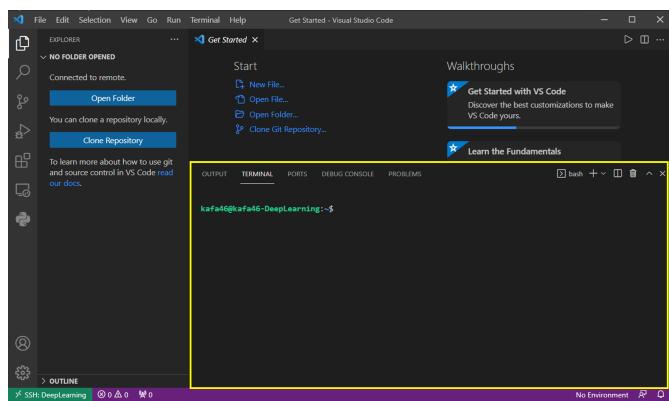


Fig. 1.11 VS Code에서 터미널 창(윈도우)가 활성화된 상태

- conda**를 이용하여 현재 설치된 가상환경 목록을 확인하려면 다음과 같이 명령어를 터미널 창에서 실행시킵니다

```
current/path$ conda create -n 가상환경이름 python=버전
```

- 예를 들어, 가상환경 이름이 **flask_tutorial**이고 **Python 3.8** 버전이 설치된 가상환경을 **conda**를 이용하여 생성하는 명령어는 다음과 같습니다.
 - 가상환경 이름을 임의로 **flask_tutorial**으로 정했습니다. 여러분은 각자 좋아하는 이름을 지정해도 됩니다.
 - 아래 코드에서 **kafa46**은 저자의 아이디, **kafa46-DeepLearning**은 저자가 설정한 서버 이름, **~\$**은 현재 작업 디렉토리 경로입니다. 각자 아이디와 컴퓨터 이름에 따라 표시되는 것은 다를 것입니다.
 - 리눅스 터미널(Bash 쉘) 프롬프트는 **아이디@컴퓨터이름:현재작업경로\$**와 같은 상태로 구성됩니다. 참고로 알고 있으면 됩니다.

```
kafa46@kafa46-DeepLearning:~$ conda create -n flask_tutorial python=3.8
```

- 위 명령어를 입력하고 엔터(**enter**)를 치면 아래와 같은 메시지가 출력되고 마지막에 설치 여부를 묻는 메시지 **Proceed ([y]/n)?**가 뜹니다.

```

Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /home/kafa46/anaconda3/envs/flask_tutorial

added / updated specs:
- python=3.8

The following packages will be downloaded:

  package          |      build
  -----|-----
sqlite-3.37.0     | hc218d9a_0      999 KB
zlib-1.2.11       | h7f8727e_4      108 KB
  -----|-----
                           Total:    1.1 MB

The following NEW packages will be INSTALLED:

_libgcc_mutex      pkgs/main/linux-64::_libgcc_mutex-0.1-main
__openmp_mutex     pkgs/main/linux-64::__openmp_mutex-4.5-1_gnu
ca-certificates    pkgs/main/linux-64::ca-certificates-2021.10.26-h06a4308_2
certifi             pkgs/main/linux-64::certifi-2021.10.8-py38h06a4308_0
ld_impl_linux-64   pkgs/main/linux-64::ld_impl_linux-64-2.35.1-h7274673_9
libffi              pkgs/main/linux-64::libffi-3.3-he6710b0_2
libgcc-ng           pkgs/main/linux-64::libgcc-ng-9.3.0-h5101ec6_17
libgomp             pkgs/main/linux-64::libgomp-9.3.0-h5101ec6_17
libstdcxx-ng        pkgs/main/linux-64::libstdcxx-ng-9.3.0-hd4cf53a_17
ncurses             pkgs/main/linux-64::ncurses-6.3-h7f8727e_2
openssl             pkgs/main/linux-64::openssl-1.1.11-h7f8727e_0
pip                 pkgs/main/linux-64::pip-21.2.4-py38h06a4308_0
python              pkgs/main/linux-64::python-3.8.12-h12debd9_0
readline            pkgs/main/linux-64::readline-8.1-h27cf23_0
setuptools          pkgs/main/linux-64::setuptools-58.0.4-py38h06a4308_0
sqlite              pkgs/main/linux-64::sqlite-3.37.0-hc218d9a_0
tk                  pkgs/main/linux-64::tk-8.6.11-h1ccaba5_0
wheel               pkgs/main/noarch::wheel-0.37.0-pyh3eb1b0_1
xz                  pkgs/main/linux-64::xz-5.2.5-h7b6447c_0
zlib                pkgs/main/linux-64::zlib-1.2.11-h7f8727e_4

Proceed ([y]/n)?

```

- 여기서 **y**를 입력하고 엔터를 치면 설치를 시작하고, **n**을 입력하고 엔터를 치면 가상환경 설치가 취소됩니다. 참고로 **[y]**는 기본값(**default**)이 **y**로 설정되었다는 의미입니다. **[y]**가 있는 경우 귀찮게 **y**를 타이핑하지 않고 바로 엔터키를 치면 바로 설치가 시작됩니다.
- y**를 입력하고 엔터를 치거나, 바로 엔터를 치면 설치 과정을 설명하는 메시지가 출력되면서 가상환경을 설치합니다. 정상적으로 설치되었다면 아래와 같은 메시지가 출력되었을 것입니다. 개인별로 기준에 설치된 가상환경에 따라 메시지는 약간 다를 수 있습니다.

```

Downloading and Extracting Packages
zlib-1.2.11         | 108 KB  | ######| ######| ######| ######| 100%
sqlite-3.37.0       | 999 KB  | ######| ######| ######| ######| 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
# $ conda activate flask_tutorial
#
# To deactivate an active environment, use
#
# $ conda deactivate

```

- 가상환경이 잘 설치되었습니다. 친절하게 새로 생성된 가상환경 **flask_tutorial**를 활성화(activation)하는 방법과 비활성화(deactivate)하는 방법까지 설명해 주고 있습니다.
- 설치된 가상환경 목록을 확인하려면 다음과 같이 명령어를 입력합니다.
 - 우리가 설치한 **flask_tutorial**이라는 가상환경이 잘 설치된 것을 확인할 수 있습니다.

```

kafa46@kafa46-DeepLearning:~$ conda env list
# conda environments:
#
base                  * /home/kafa46/anaconda3
flask_tutorial          /home/kafa46/anaconda3/envs/flask_tutorial

```

- 설치된 가상환경을 진입하는 명령어는 **conda activate** 가상환경이름 입니다. 아래와 같이 명령어를 입력하면 다음과 같이 됩니다. 아래 코드는 저자 컴퓨터 기준입니다. 가상환경으로 정확히 진입하였다면 터미널 프롬프트 앞에 (**가상환경이름**)이 표시될 것입니다. 우리 예제에서는 **flask_tutorial**이라는 가상환경 이름을 지어주었으므로 (**flask_tutorial**)이라는 글자가 프롬프트 맨 앞에 붙어있게 됩니다. 아래 코드를 참고하세요.

```
kafa46@kafa46-DeepLearning:~$ conda activate flask_tutorial  
(flask_tutorial) kafa46@kafa46-DeepLearning:~$
```

- 가상환경에서 빠져나오고 싶을 경우 사용하는 명령어는 `conda deactivate`입니다. 아래와 같이 입력하면 명령 프롬프트 맨 앞에 있던 (가상환경이름)이 없어집니다. 명령 프롬프트 앞에 (가상환경이름)이 사라졌다면 정상적으로 가상환경에서 빠져나온 것입니다.

```
(flask_tutorial) kafa46@kafa46-DeepLearning:~$ conda deactivate  
kafa46@kafa46-DeepLearning:~$
```

1.5.2.1.3. VS Code에 새로 만든 가상환경 등록하기

VS Code는 사용자가 어떤 가상환경 사용할지 모릅니다. 접속할 때마다 여러분들이 만든 가상환경(필자의 경우 `flask_tutorial`)을 알려줘야 합니다. 매번 VS Code를 실행할 때마다 이런 작업을 하기는 번거롭습니다.

작업 디렉토리에서 사용할 가상환경을 VS Code에 등록할 수 있습니다.

VS Code를 실행시키고 `CTL + Shift + P`를 누르면 명령 팔레트 (command palette)가 활성화 됩니다.

명령 팔레트에 `Python: Select Interpreter`를 입력하면 여러분들이 작업하는 디렉토리에서 사용할 파이썬 인터프리터(python 실행파일)을 등록할 수 있습니다.

다음 그림을 참고하세요.

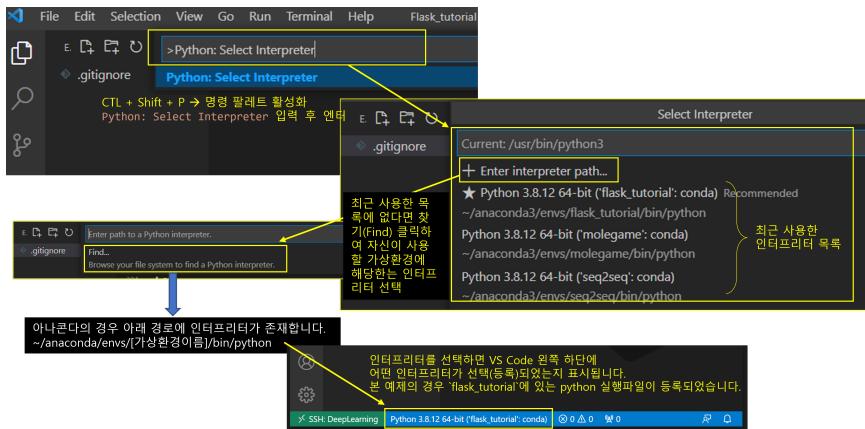


Fig. 1.12 VS Code에 가상환경에서 작동할 파이썬 인터프리터를 등록하는 방법

명령 팔레트에 `Python: Select Interpreter`를 입력하면 최근 사용한 파이썬 인터프리터 목록이 보입니다.

여러분이 만든 가상환경 이름을 가진 인터프리터가 목록에 있다면 그대로 선택하면 됩니다.

만약 없다면 `Enter interpreter path...`를 클릭합니다.

가상환경에서 작동하는 파이썬 실행파일의 경로를 정확히 알고 있다면 경로창에 그대로 타이핑해도 됩니다.

정확히 모를 경우에는 `Find...`을 클릭하여 설치된 가상환경과 파이썬 실행파일을 찾아갑니다.

리눅스에서 `anaconda`를 이용해 가상환경을 만들었다면 파이썬 인터프리터(실행파일) 경로는 `~/anaconda/envs/[가상환경이름]/bin/python`입니다. 제 경우는 `~/anaconda/envs/flask_tutorial/bin/python`이 됩니다.

해당 경로를 찾아서 클릭하면 VS Code 좌측 하단의 파란 창에 여러분이 등록한 파이썬 인터프리터의 버전, 가상환경 이름, 가상환경 생성 도구가 표시됩니다.

제 경우는 `Python 3.8.12 64-bit ('flask_tutorial': conda)`와 같이 표시되었습니다. 여러분들도 각자 이름 지어준 대로 표시가 되었을 겁니다. 이렇게 되면 다음부터 VS Code를 실행할 때마다 자동으로 우리가 등록한 가상환경 인터프리터를 실행하게 됩니다.

1.6. 준비 3. flask 설치

`flask`를 설치하기 위해 실습을 위해 준비한 가상환경에 진입한 이후 `pip3(Linux/macOS)` 또는 `pip(Windows)`를 이용하여 손쉽게 설치할 수 있습니다. 아래 코드를 참고하세요.

```
(flask_tutorial) kafa46@kafa46-DeepLearning:~$ pip3 install flask
```

위 명령어를 입력하고 엔터를 치면 아래와 유사한 메시지가 출력됩니다.

```
Collecting flask
  Downloading Flask-2.0.2-py3-none-any.whl (95 kB)
    ██████████| 95 kB 3.1 MB/s
Collecting click>=7.1.2
  Downloading click-8.0.3-py3-none-any.whl (97 kB)
    ██████████| 97 kB 9.4 MB/s
Collecting Jinja2>=3.0
  Downloading Jinja2-3.0.3-py3-none-any.whl (133 kB)
    ██████████| 133 kB 26.5 MB/s
Collecting Werkzeug>=2.0
Using cached Werkzeug-2.0.2-py3-none-any.whl (288 kB)
Collecting itsdangerous>=2.0
  Downloading itsdangerous-2.0.1-py3-none-any.whl (18 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.0.1-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (30 kB)
Installing collected packages: MarkupSafe, Werkzeug, Jinja2, itsdangerous, click, flask
Attempting uninstall: Jinja2
  Found existing installation: Jinja2 2.11.2
  Uninstalling Jinja2-2.11.2:
    Successfully uninstalled Jinja2-2.11.2
Successfully installed Jinja2-3.0.3 MarkupSafe-2.0.1 Werkzeug-2.0.2 click-8.0.3 flask-2.0.2 itsdangerous-2.0.1
```

만약 에러 메시지가 나온다면 일단 무시하고 넘어갑니다. 짐작한다면 똑같은 명령어를 다시 한번 실행합니다. 만약에 여러분의 가상환경에 이미 `flask`가 설치되어 있다면 `Requirement already satisfied: 패키지명 in 경로명`과 유사한 메시지가 주루룩 뜹니다. 이미 설치된 것으로 더 이상 설치를 시도하지 말고 넘어 가도록 합니다.

확실히 `flask`가 설치된 것을 확인하고 싶다면 `python`을 실행시켜서 `flask`를 임포트 해보면 됩니다. 임포트 에러가 없다면 완벽히 설치된 것으로 생각해도 됩니다. 아래 코드를 참고하세요.

```
(flask_tutorial) kafa46@kafa46-DeepLearning:~$ python
Python 3.8.12 (default, Oct 12 2021, 13:49:34)
[GCC 7.5.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> import flask
>>> flask.__version__
'2.0.2'
```

이것으로 우리가 설치한 가상환경에 `flask`를 설치하는 것을 실습해 보았습니다. 모두들 성공하셨을 것을 생각합니다. `flask`의 세계에 들어온 것을 환영합니다.

1.7. 준비 4. `.gitignore` 세팅

내 컴퓨터에서 작업한 것들을 협업 또는 백업을 위해 원격 저장소에 업로드해야 하는 경우가 항상 발생합니다. 이 경우 사용하는 것이 `git`이라는 도구입니다. 본 책자는 `git`에 대한 튜토리얼이 아니므로 `git`에 대한 세부적인 설명은 생략합니다. `git`에 대해 좀 더 공부하고자 하는 여러분은 `git` 튜토리얼이 블로그, 유튜브 강의, 전문서적 구매 등을 통하여 학습할 것을 추천합니다.

`.gitignore`는 `git`을 이용하여 내 컴퓨터에서 작업한 소스코드를 원격 저장소(repository)로 업로드 할 때 불필요한 파일까지 업로드 되는 것을 방지하기 위해 작성하는 텍스트 파일입니다. 예를 들면 VS Code 설정파일(`.vscode`)은 내 컴퓨터에서는 필요하지만, 원격저장소에 올릴 필요는 없습니다. Jupyter Notebook에 필요한 `.ipynb_checkpoint` 폴더, 백업 파일(`.bak`), 데이터베이스 파일(`.db`), 개인 정보나 비밀번호 등이 담긴 파일(`*.secret`) 등은 원격 저장소에 올리면 안됩니다.

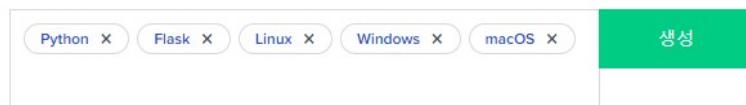
작업 디렉토리에 `.gitignore`라는 빈 파일 하나를 만들고 그 안에 내가 원격저장소로 업로드 할때 자동으로 떨려 올라가지 않도록 할 파일들을 쭉 적어주면 됩니다. `.gitignore` 파일 작성법은 [참고 블로그](#)를 방문해서 관련 내용을 참고하기 바랍니다.

1.7.1. 유용한 사이트 `gitignore.io`

자주 사용하는 운영체제, 개발환경(IDE), 프로그래밍 언어에 대한 `.gitignore` 내용을 자동으로 작성해 주는 유용한 사이트가 있습니다. 해당 사이트는 [gitignore.io](#)입니다. [gitignore.io](#)에서 `python`, `flask`, `linux`, `windows`, `macOS`, `VisualStudioCode`와 관련된 `gitignore` 내용을 생성해 보겠습니다.



자신의 프로젝트에 꼭 맞는 .gitignore 파일을 만드세요



소스 코드 | 커맨드라인 문서

Fig. 1.13 gitignore.io를 이용해서 .gitignore 내용을 자동으로 생성하기

gitignore.io에서 생성 아이콘을 클릭하면 아래와 같이 자동으로 원격 저장소 업로드 시 제외할 파일/디렉토리 목록을 생성해 줍니다.

i Note

gitignore.io에서 모든 것을 해결해 주지는 않습니다. 자주 사용하는 것들만 모아서 생성해 주기 때문에 개인적으로 제외할 파일을 별도로 지정해 주어야 합니다.

```
# Created by https://www.toptal.com/developers/gitignore/api/python,flask,linux,windows,macos,visualstudiocode
# Edit at https://www.toptal.com/developers/gitignore?templates=python,flask,linux,windows,macos,visualstudiocode

### Flask ###
instance/*
!instance/.gitignore
.webassets-cache
.env

### Flask.Python Stack ###
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST
:
:
```

VS Code에서 .gitingnore 파일을 만들고 위에서 생성한 목록을 복사하여 붙여넣기 합니다. 다음 그림을 참고하세요.

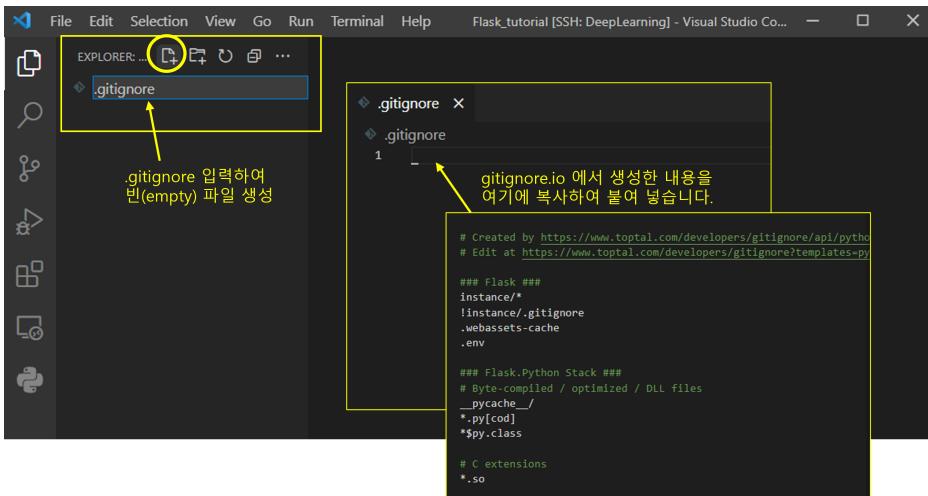


Fig. 1.14 VS Code에서 `.gitignore` 파일 만들기

1.7.2. `.gitignore` 추가 설정

`flask`를 사용하려면 데이터베이스가 필요합니다. 나중에 `sqlite`라는 데이터베이스를 사용할 예정입니다. `.gitignore` 파일의 맨 마지막에 아래 내용을 추가로 복사하여 붙여넣기 한 다음 파일을 저장합니다.

```
# sqlite
*.db
```

파일 이름에서 .과 * 의미

파일이름에서 파일 이름이 없고 점(.) 다음에 확장자 이름 형태로 되어 있는 것은 대부분 설정(configuration) 파일(또는 디렉토리)을 의미합니다.

- `.gitignore` \(\to\) `gitignore` 설정파일
- `.bash` \(\to\) `bash` 설정파일
- `.vscode` \(\to\) `VS Code` 설정 파일

파일 이름에 와일드 문자 *가 있다면 모든(all)을 의미합니다.

- `*.txt` \(\to\) `.txt`를 확장자로 갖는 모든 파일
- `.gitignore` 파일 안에 `*.txt`가 기록되어 있다면, `.txt`를 확장자로 갖는 모든 파일을 원격 저장소 업로드 시 제외하겠다는 의미입니다.

1.8. 준비 5. 의존성 파일 `requirement.txt` 생성

1.8.1. 의존성의 개념

의존성 파일이라는 것이 왜 필요할까요? 개발자들은 개발(코딩)을 안정적으로 진행하고 테스트하기 위하여 독립된 환경, 즉 가상환경(virtual environment)를 만들어서 코딩을 합니다.

이렇게 완성된 코드는 `git`을 이용해 원격 저장소에 업로드하여 공유하거나 협업하게 됩니다. 때로는 이메일이나 카톡으로 소스코드를 주고 받을 수도 있습니다. 문제는 나에게 코드를 공유한 사람이 어떤 환경에서 개발했는지 알 수 없다는 겁니다.

따라서, 코드를 공유할 사람(다른 사람에게 전달하는 개발자)은

본인의 가상환경에서 어떤 패키지나 라이브러리를 설치하여 개발했는지도 같이 알려줘야 합니다. 이렇게 코드를 전달하는 사람의 가상환경을 정리한 파일을 **의존성 파일**이라고 합니다.

의존성 파일은 파이썬에서 지원하는 패키지 관리자 `pip`를 이용해 손쉽게 작성할 수 있습니다.

그런데 한 가지 문제는 **의존성 파일** 이름은 누구나 마음대로 지을 수 있다는 것입니다. 어떤 개발자는 **dependancy.txt**라고 지을 수 있고, 또 다른 개발자는 **virtual_env.txt**라고 지을 수도 있겠죠? 그래서 **의존성 파일**의 이름을 누가 보더라도 쉽게 알도록 통일하기로 했습니다. 그 통일된 이름이 **requirements.txt**입니다.

의존성 파일 **requirements.txt**를 만들어 보겠습니다. 먼저 본인이 개발한 가상환경에 진입합니다. 앞에서 배운대로 **conda activate** 가상환경이름을 통해서 가상환경을 활성화 하겠죠?

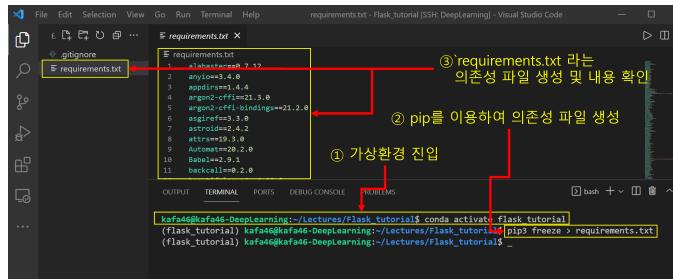


Fig. 1.15 VS Code에서 의존성 파일 **requirements.txt** 만들기

먼저 작업한 가상환경으로 들어갑니다.

requirements.txt 파일을 만들기 위해 **pip3 freeze > requirements.txt**라는 명령어를 입력하고 엔터를 칩니다.

- 원도우즈 운영체제일 경우 **pip** 명령어를 사용하고, 리눅스/macOS일 경우 **pip3** 명령어를 사용합니다.
- **freeze**는 현재 환경에서 사용하고 있는 모든 패키지 목록을 모니터(콘솔, 터미널)에 출력하라는 명령어입니다.
- **>**는 모니터(터미널, 콘솔)에 출력되는 내용의 방향을 바꾸라는 말입니다.
 - 방향을 바꾼다는 것을 **재지향(redirection)**한다고 표현합니다.
 - 모니터에 출력되는 방향을 바꿔서 파일로 출력하도록 합니다.
- **requirements.txt**는 **>**에 의해 출력의 방향이 모니터에서 파일로 바꾼 경우, 내용을 저장할 파일 이름입니다.
 - 파일 이름은 아무거나 사용해도 됩니다.
 - 의존성 파일 이름을 **requirements.txt**로 정했기 때문에 그대로 사용한 것입니다.

pip3 freeze > requirements.txt가 실행되면 VS Code 우측의 파일 탐색기에 **requirements.txt** 파일이 생성된 것을 확인할 수 있습니다.

requirements.txt 파일을 클릭하면 내용을 확인할 수 있습니다. **requirements.txt**에 기록된 내용은 현재 환경에서 사용하고 있는 모든 패키지를 이름과 버전 정보가 포함되어 있습니다. 소스코드를 전달받은 개발자는 **requirements.txt**를 확인하고 어떤 개발 환경에서 개발했는지 쉽게 알 수 있습니다.

requirements.txt 파일을 이용하면 소스코드를 나에게 전달한 사람과 동일한 가상환경을 다음과 같이 손쉽게 생성할 수 있습니다.

- 가상환경을 하나 만듭니다.
- 새로 만든 가상환경에 들어갑니다.
- 아래와 같이 명령어를 주면 **requirements.txt**에 있는 모든 패키지가 자동으로 설치됩니다.

```
pip3 install -r requirements.txt
```

2. 맛보기로 만드는 flask app

처음 웹 개발 프레임워크를 공부할 때 힘든 부분 중 하나가 개발 환경을 구축하는 것입니다. 지금까지 개발 환경 전반에 대하여 살펴 보았습니다. 고생하셨습니다.

이제부터는 flask를 이용해 자신만의 앱을 작성할 수 있게 되었습니다.

우리는 간단한 코딩을 통해 **hello_cju.py**라는 python 스크립트(script)를 만들겠습니다. Flask 서버를 가동시키고 **hello_cju.py**를 실행하면 화면에 **Hello world! Welcome to CJU.**라는 문자열을 출력하는 간단한 웹 시스템을 만들어 보겠습니다.

우리는 다음과 같은 5 단계를 거쳐 위에서 언급한 웹 시스템을 구축할 예정입니다.

❶ 간단 웹 시스템 구축 순서

- Step 1. Flask app 코딩 ([바로가기](#))
- Step 2. Flask 서버 실행 ([바로가기](#))
- Step 3. FLASK_APP 설정 ([바로가기](#))
- Step 4. 개발자 모드 설정 ([바로가기](#))

여러분, 준비 되셨나요?

그러면 차근차근 스텝을 밟아 보도록 하겠습니다.

2.1. Flask app 코딩

파이썬을 이용해서 첫 번째 앱을 만들어 보겠습니다.

우리가 만들 웹 시스템 **Hello CJU** (`hello_cju.py`)는 5줄짜리 프로그램입니다.

아주 간단한 형태이지만 flask 기본 구조를 이해하는데 중요한 과정입니다.

차근차근 작성해 보도록 하겠습니다.

먼저 VS code에서 새로운 프로젝트를 하나 생성합니다. 새로 만들 프로젝트 디렉토리(폴더) 경로는 여러분이 원하는 곳에 자유롭게 만들어도 상관 없습니다.

저는 **Source_code**라는 폴더를 만들었습니다. VS code 실행시키고 `File \(\to\)` `Open Folder` 메뉴를 선택하여 여러분이 만든 폴더를 VS code에서 엽니다. 디렉토리(폴더) 이름에 마우스를 위치시키고 오른쪽 클릭하거나 더하기 아이콘을 클릭하여 새로운 파일을 만듭니다. 이때 파일 이름은 여러분이 원하는 대로 지어도 되지만 본 교재와 일관성을 유지하면서 실습을 하고 싶다면 `hello_cju.py`로 파일명을 지어 주는 것도 좋습니다.

저는 아래 그림 [Fig. 2.1](#)과 같이 파일을 만들었습니다.

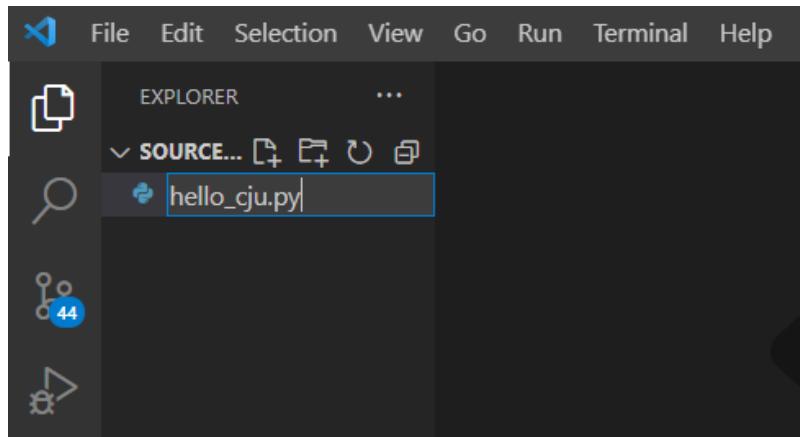


Fig. 2.1 VS code에서 `hello_cju.py` 파일 만들기

이제는 `hello_cju.py` 내용을 작성합니다.

아래 코드와 같이 작성해 봅니다.

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello world! Welcome to CJU.'
```

코드를 살펴볼까요?

1. `python from flask import Flask`

flask에서 제공하는 `Flask` 클래스를 불러옵니다.

2. `python app = Flask(__name__)`

`Flask` 클래스를 이용하여 `app`이라는 객체를 만듭니다. 객체 생성자로 `__name__`이라는 변수가 전달되었습니다.

`__name__`이라는 변수에는 flask `app` 객체가 실행시킬 파일이션 스크립트(즉, 파일 모듈) 이름이 담기게 됩니다.

우리가 만든 것은 `hello_cju.py`라는 모듈을 만들었으니, `__name__` 변수에는 `hello_cju`가 담기게 됩니다.

3. `python @app.route('/')`

`@app.route()`는 괄호() 안에 있는 주소로 접속하면 바로 아랫줄에 있는 함수를 호출하라는 뜻입니다. Flask에서 지원하는 데코레이터(decorator) 기능입니다.

우리 예제에서는 괄호() 안에 /를 써 주었습니다. 접속한 주소가 `root`일 경우 바로 다음 줄에 있는 함수를 호출하라는 의미입니다.

감이 잘 안오시죠? 예를 들어 우리가 만든 `Hello CJU (hello_cju.py)` 시스템을 서비스 하는 컴퓨터(서버)의 주소가 `203.252.240.123`이라고 해볼까요. 우리 시스템을 서비스 해야 겠죠? 서비스를 하려면 웹 서버가 1년 365일 24시간 내내 돌아가면서 인터넷 사용자를 기다리다가 요청이 들어오면 결과를 보내줘야 합니다.

만약 어떤 인터넷 사용자가 크롬 브라우저 주소창에 `203.252.240.123` 또는 `203.252.240.123/` 요렇게 치면, flask는 루트 디렉토리로 인식하고 바로 다음줄에 있는 `python def hello_world():` 함수를 실행한다는 의미입니다.

어떤 사용자가 `203.252.240.123/class`라는 주소를 크롬 주소창에 쳤다면 `@app.route(/)`와 맞지 않으니 `python def hello_world():` 함수는 작동하지 않을 겁니다. 만약, `@app.route(/class)`라는 데코레이터가 있는 함수가 있다면 작동할 것입니다.

일단 경로를 잡아주는 (라우팅 해주는) 데코레이터라고 쉽게 생각하고 넘어가도록 합니다.

4. `python def hello_world():`

app 객체에 할당된 주소(경로)로 요청이 들어왔을 때 어떤 일을 할 것인지를 정의한 함수입니다. 우리는 “python ‘Hello world! Welcome to CJU.’”라는 문자열을 만들어서 서비스를 요청한 인터넷 사용자에게 되돌려 주는 일을 하도록 코딩했습니다.

Flask 데코레이터

Flask 데코레이터는 Python에서 지원하는 데코레이터와 동일합니다.

자세한 내용은 이 [점프 투 플라스크 블로그](#)를 참고하세요

이제 간단한 코드 작성이 끝났습니다.

참고로 VS code를 이용해 작성한 모습은 그림 Fig. 2.2와 같습니다.

```
File Edit Selection View Go Run Terminal Help • hello_cju.py - Source_codes
EXPLORER ... hello_cju.py U ●
SOURCE_CODES ...
hello_cju.py U
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def hello_world():
7     return 'Hello world! Welcome to CJU.'
```

Fig. 2.2 VS code에서 `hello_cju.py` 코드를 작성한 모습

이제 작업할 내용은 flask 서버를 실행하는 것입니다.

[Next](#)를 클릭하여 이동해 보세요.

2.2. Flask 서버 실행

우리는 목표 웹 시스템 `Hello Cju (hello_cju.py)` 코딩을 마쳤습니다.

그 다음 할 일은 flask 서버를 작동시키는 것입니다.

Flask 서버를 작동은 명령창에 `flask run` 이라는 명령어를 입력하면 됩니다.

명령을 실행하기 위해 윈도우의 경우 `cmd, powershell`에서 입력하거나, 리눅스/MacOS 경우 `Terminal` 창을 실행하여 입력하면 됩니다.

VS code, Pycharm 등과 같은 통합개발환경(IDE) 에디터에서는 별도로 명령창 실행 없이 곧바로 활용할 수 있도록 명령창을 함께 제공하고 있습니다.

우리는 VS code를 활용하고 있으므로 그림 Fig.2.3과 같이 `Window \(\to\)\ Terminal` 메뉴를 선택하거나, VS code 단축키로 `CTL + `` 을 사용하여(백틱 키 사용) 명령창을 실행합니다.

Note

Backtick 백틱(backtick)은 작은 따옴표와 비슷하게 출력되지만 작은 따옴표와 다른 키(key)입니다. 작은 따옴표는 키보드의 엔터(enter) 키 왼쪽에 있지만, 백틱 키는 숫자 1의 왼쪽에 있는 키입니다. 백틱은 Markdown과 같은 에디터에서 특수 기능을 활성화 할 때 사용됩니다. 백틱 `은 작은 따옴표 '의 방향과 반대 모양으로 생겨서 back(반대) tick(점)이라고 부릅니다.

백틱은 `grave accent` 또는 `backquote`라고 부르기도 합니다.

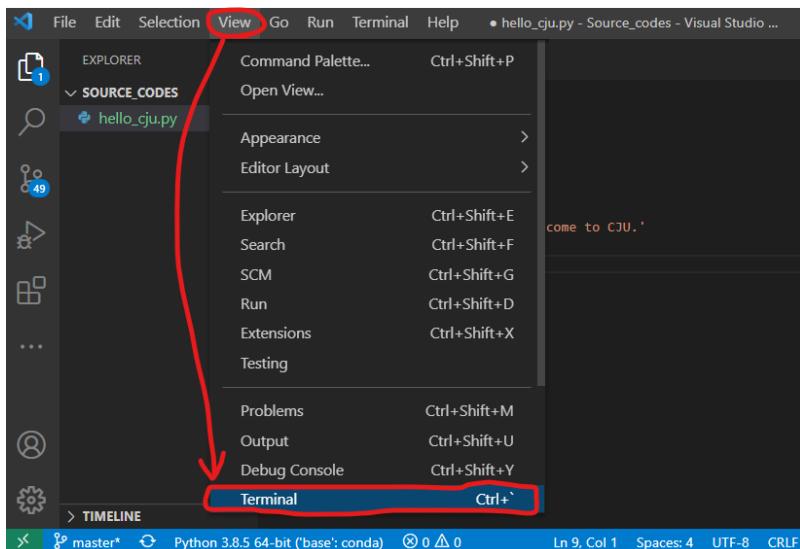


Fig. 2.3 VS code에서 `hello_cju.py` 파일 만들기

명령창이 열리면 여러분이 만든 프로젝트 디렉토리(저자의 경우 `Source_codes`라는 디렉토리)에서 `flask run` 이라고 입력하고 엔터키를 칩니다.

명령을 실행하면 그림 Fig.2.4 같은 메시가 출력됩니다.

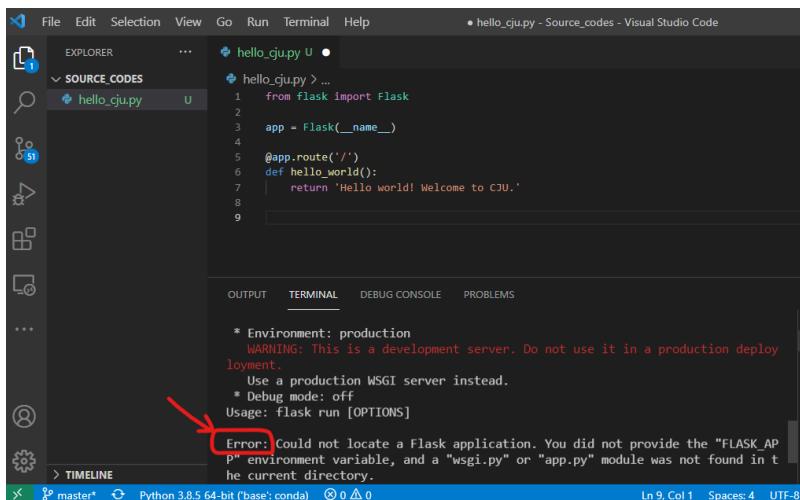


Fig. 2.4 flask run 결과

Fig. 2.4 VS code에서 `hello_cju.py` 파일 만들기

그림 Fig.2.4에서 출력된 에러를 깔끔하게 다시 보면 아래와 같은 터미널 에러 메시지를 확인할 수 있습니다.

```
(가상환경이름) C:\여러분의 컴퓨터 경로\Source_codes>flask run
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
Usage: flask run [OPTIONS]

Error: Could not locate a Flask application.
You did not provide the "FLASK_APP" environment variable,
and a "wsgi.py" or "app.py" module was not found in the current directory.
```

위 메시지를 차근차근 살펴보도록 하겠습니다.

1. **Environment: production:** 현재 여러분이 사용하고 있는 서버 환경은 제품(즉, 서비스 배포) 단계라는 것입니다. 실제로 여러분이 만든 시스템을 배포하는 경우에는 현재 환경 대신 `WSGI` 서버를 사용하라고 경고합니다.
2. **Debug mode: off:** 현재는 디버그 모드가 꺼져 있다고 알려줍니다.
3. **Usage: flask run [OPTIONS]:** 서버를 가동할 때 다양한 옵션이 제공되니 필요하면 `flask run [옵션 이름]`과 같은 형태로 명령어를 사용하라고 알려줍니다.
4. **Error:** Flask가 볼 때 우리가 만든 앱 `hello_cju.py`를 어떻게 위치시켜야 하는지 잘 모르겠다는 의미입니다. 이것을 해결하기 위해서는 `FLASK_APP`이라는 환경 변수가 있는데 여기에 우리가 만든 앱이 무엇인지 알려달라고 합니다. 그리고 현재 디렉토리(폴더)에 `wsgi.py` 또는 `app.py` 모듈을 찾을 수 없다고 합니다.

에러를 보니 답답합니다.

그러나 flask 입장에서도 답답한 것은 우리와 비슷할 겁니다.

우리는 코딩 잘 끝내고 `flask run` 명령어까지 정확히 줬는데 에러라니!!

하지만, Flask 입장은 이렇습니다.

❶ Flask 서버 실행 에러를 내보낸 이유

- 개발자들이 웹 시스템 이름을 어떻게 지을지 어떻게 미리 알지?
- 그러면 기본적으로 `app.py` 이라는 이름을 갖는 웹 시스템을 개발한다고 가정하자.
- 만약에 개발자들이 `app.py`라는 이름으로 모듈을 코딩 했으면 정상적으로 서버를 가동시키자.
- 혹시 다른 이름을 가진 모듈을 만들면 그때는 어쩌나?
 - `FLASK_APP`이라는 환경변수를 만들어 놓고,
 - 개발자가 이 환경변수에 모듈 이름을 등록하게 하면 되겠군.
 - 이 환경변수에 담긴 모듈 이름을 작동시키면 모든게 해결겠군. ㅎ
- 그런데 `app.py` 모듈도 없고, `FLASK_APP` 변수에 다른 이름도 등록 안되어 있다면?
 - 어라! 어떤 모듈을 돌리라는 거지???
 - 개발자가 깜빡한 모양이군! 에러 메시지를 보여주자.
 - `FLASK_APP`에 개발자가 코딩한 모듈 이름을 적어주면 그때 실행하자!

이런 스토리가 숨어 있는 메시지입니다. 참고로 `wsgi.py` 관련 사항은 나중에 따로 설명합니다.

이제 에러 내용을 확이했고 해석할 수 있게 되었습니다.

다음 작업할 내용은 flask 서버에 우리가 만든 Python 모듈을 등록하는 것입니다.

[Next](#)를 클릭하여 이동해 보세요.

2.3. `FLASK_APP` 설정

앞 절 [Flask 서버 실행](#)에서 `flask run` 명령어를 전달하면 에러가 발생하는 것을 살펴보았습니다.

에러의 원인은 우리가 만든 `hello_cju.py`라는 앱이 flask가 기본으로 인식하는 `app.py`가 아니라는 것이었습니다.

Flask는 이것을 해결하기 위해 `FLASK_APP`이라는 환경변수에 우리가 만든 앱 `hello_cju.py`를 등록해 달라고 요청하고 있습니다.

Flask 앱 이름은 개발자가 코딩한 `xxx.py`라는 모듈에서 확장자 `.py`를 제거한 것입니다. 만약 개발자가 `abc.py`라는 모듈을 코딩했다면 앱 이름은 `abc`입니다. 우리는 `hello_cju.py`를 만들었으므로 앱 이름은 `hello_cju`가 됩니다.

우리 앱 `hello_cju`를 등록하려면 VS code 명령창에 `set FLASK_APP=hello_cju`라고 입력하고 엔터를 치면 됩니다. 터미널 입력화면은 다음과 같습니다.

```
(가상환경이름) C:\ 여러분의 컴퓨터 경로\ Source_codes>set FLASK_APP=hello_cju
```

위 명령어를 실행하고 다시 `flask run` 명령어를 입력하면 아래와 같은 메시지를 출력하면서 flask 서버가 작동합니다.

```
(가상환경이름) C:\ 여러분의 컴퓨터 경로\ Source_codes>flask run
 * Serving Flask app 'hello_cju' (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: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

`Serving Flask app 'hello_cju' (lazy loading)`라는 메시지는 우리가 만든 `hello_cju`라는 앱(Python 모듈)을 서버에서 잘 돌리고 있다는 의미입니다.

메시지 중에 `lazy loading`이라는 것이 보입니다. 무엇일까요? 무척 궁금해집니다. 알아두면 나쁠 것 없겠죠?

어떤 인터넷 사용자가 여러분이 만든 웹 시스템에 재접속 기능을 (`reloader`) 사용하는 `flask run` 명령어를 작동시키는 경우에 발생하는 에러 처리 방법이 있습니다. 주로 개발자 모드에서 사용하게 됩니다. 개발자 모드 설정은 다음 절([개발자 모드 설정](#))에서 구체적으로 설명합니다.

`reloader`는 개발자가 코드를 수정하고 서버를 다시 실행시키는 flask 서버의 재시작 도구입니다. 개발자가 코드를 잘 수정했다면 문제가 없지만, 문법 에러나 초기화 관련 코드에 에러가 있는 경우에 시작하는 방법이 2가지 가능합니다.

만약 처음 `flask run` 명령어를 사용할 당시에 이미 문법 에러 (syntax error)가 있다면 웹 시스템에서 제공하는 사이트에 접속할 때까지 기다리지 않고 즉시 서버 작동을 중단하고 에러 추적 결과 (`traceback`)를 출력합니다.

서버를 시작한 이후 코드를 수정하고 재시작 하려고 할 때 에러가 있는 경우 서버를 즉시 중단시키는 대신에 개발자에게 메시지를 주고 받을 수 있는 디버거(`interactive debugger`)를 보여주는 방법이 가능합니다.

이 방법은 서버 중지(`crash`)를 최대한 지연하고 항상 살아있을 수 있도록 합니다. 이렇게 작동하는 방식을 `lazy loading`이라고 합니다.

반면에 flask 서버가 처음 작동을 시작하는 상황이든, 코드를 수정하고 `reload` 하는 상황이든 관계없이 에러가 있다면 항상 서버 작동을 중지시키는 방식을 `eager loading`이라고 합니다.

서버 실행 단계에서 `eager loading` 또는 `lazy loading`을 선택할 수 있습니다. `flask run` 명령어 다음에 옵션을 붙여서 실행시키면 됩니다.

- `eager loading`을 원할 경우: `flask run --eager-loading`
- `lazy loading`을 원할 경우: `flask run --lazy-loading`

Note

개발자(디버그) 모드가 선택되면 `reloader` 와 `debugger`가 자동으로 작동(활성화) 됩니다. `reloader` 와 `debugger`는 개별적으로 선택 가능합니다.

- `reloader`는 `--reload` 또는 `--no-reload` 옵션을 이용해서 활성/비활성 가능합니다.
- `debugger`는 `--debugger` 또는 `--no-debugger` 옵션을 이용해 활성/비활성 가능합니다.

만약 `reloader`가 비활성화 되어 있다면 `reload` 옵션은 기본적으로 `eager loading`이 적용됩니다.

서버에서 잘 돌아간다는 의미는 어떤 인터넷 사용자가 앱 서비스 요청을 하게 되면 언제든지 결과(response)를 알려줄 수 있다는 의미입니다. 그럼 인터넷 사용자는 어디로 앱 서비스를 요청할까요? 그게 좀 애매하니 flask가 알려주는 메시지가 맨 마지막 줄에 있는 메시지 `Running on http://127.0.0.1:5000 (Press CTRL+C to quit)`입니다.

다음과 같은 의미입니다.

- `hello_cju`에 서비스를 요청하려면 크롬 같은 웹 브라우저 창에 `http://127.0.0.1:5000`라고 입력하세요.
- 여기서 `http`는 프로토콜 이름, `127.0.0.1`는 서버의 IP 주소, `5000`은 포트 번호입니다.
 - 서버의 아이피 주소가 `127.0.0.1`와 같은 형태를 `localhost`(로컬호스트)라고 부릅니다.
 - 현재는 전용 서버가 없고, 여러분의 컴퓨터가 서버가 되어서 여러분의 컴퓨터에 서비스하는 상황을 뜻합니다.

- (Press **CTRL+C** to quit)는 Flask 서버를 중단하려면 **CTL + C** 키를 누르라는 뜻입니다.

이제 flask 서버를 작동시키고 서버 관련 내용을 해석할 수 있게 되었습니다.

다음 작업할 내용은 flask 서버를 개발환경으로 실행하는 것입니다.

[Next](#)를 클릭하여 이동해 보세요.

2.4. 개발자 모드 설정

개발자 모드는 flask를 개발하면서 필요한 다양한 디버깅(debugging) 기능을 제공하는 모드입니다.

개발자 모드를 활성화하는 방법은 명령창에서 flask 환경 변수 **FLASK_ENV**을 **development**로 변경해 주면 됩니다.

변경하는 명령어는 `set FLASK_ENV=development`이다. 명령창에 입력한 모양은 다음과 같다.

```
(가상환경이름) C:\여러분의 컴퓨터 경로\Source_codes>set FLASK_ENV=development
```

Flask 환경 변수 **FLASK_ENV**을 설정하고 서버를 다시 실행하면 다음과 같은 메시지가 출력된다.

```
(가상환경이름) C:\여러분의 컴퓨터 경로\Source_codes>flask run
 * Serving Flask app 'hello_cju' (lazy loading)
 * Environment: development
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 262-914-443
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

위 메시지를 보면 서버 환경은 개발자 모드 **Environment: development**로 잘 설정되어 있습니다. 개발자 모드를 선택하니 디버그 모드 **Debug mode: on** 가 자동으로 설정되고, **Debugger**가 작동되었다는 **Debugger is active!** 메시지도 출력되었습니다. 서버를 재시작하는 **reloader**를 **stat**를 이용하는 **Restarting with stat** 메시지를 확인할 수 있습니다.

i Debug mode: on

개발자 모드 (**Environment: development**)가 실행되면 자동으로 디버그 모드 (**Debug mode**)가 실행(**on**) 됩니다.

디버그 모드는 2가지 편리한 기능을 개발자에게 제공합니다.

1. 코드를 수정하면 서브를 자동으로 재시작하여 변경된 내용을 적용해 줍니다.

2. 에러가 발생할 경우 에러 추적 (**traceback**) 결과를 알려 줍니다.

디버그 모드가 있다면 개발자는 매우 편리합니다.

특이한 것은 **Debugger PIN**이라는 정보가 표시된다는 점입니다. **Debugger PIN**은 **Debug mode** 가 **on** 상태일 경우에 생성됩니다. 에러가 발생한 경우에 대화형 디버거(**interactive debugger**)를 실행할 수 있는데 이 때 사용하는 것이 **Debugger PIN**입니다. 대화형 디버거(**interactive debugger**)를 사용하면 시스템의 여러가지 기능이나 설정을 마음대로 변경할 수 있기 때문에 시스템 공격자로부터 방어하기 위한 보안(**security**) 기능으로 제공되는 것이 **Debugger PIN**입니다.

개발자 모드 역시 **localhost**로 서비스 됩니다. 앞 장에서 이미 설명한 것과 마찬가지로 **http**는 프로토콜 이름, **127.0.0.1**는 서버의 IP 주소, **5000**은 포트 번호입니다.

- 서버의 아이피 주소가 **127.0.0.1**와 같은 형태를 **localhost**(로컬호스트)라고 부릅니다.
- 현재는 전용 서버가 없고, 여러분의 컴퓨터가 서버가 되서 여러분의 컴퓨터에 서비스하는 상황을 뜻합니다.
- (Press **CTRL+C** to quit) 는 Flask 서버를 중단하려면 **CTL + C** 키를 누르라는 뜻입니다.

인터넷 브라우저를 열고 flask 서버의 IP 주소를 입력하면 우리가 만든 웹 시스템 **Hello Cju** 이 정상적으로 작동한 것을 확인할 수 있습니다.

그림 [Fig. 2.5](#) 은 크롬 브라우저를 사용해 작동한 결과입니다.

브라우저 주소창에 **127.0.0.1** 대신에 **localhost**를 입력해도 동일한 결과를 얻을 수 있습니다.

그림 [Fig. 2.6](#) 은 크롬 브라우저 주소창에 숫자로 된 IP 주소 대신에 **localhost** 입력하여 flask 서버로부터 응답을 받은 결과를 보여주고 있습니다.

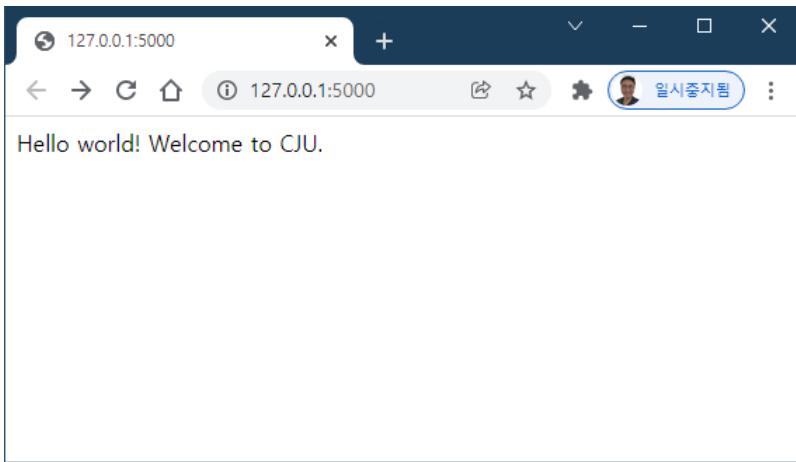


Fig. 2.5 크롬 브라우저에서 flask 서버에 요청한 결과

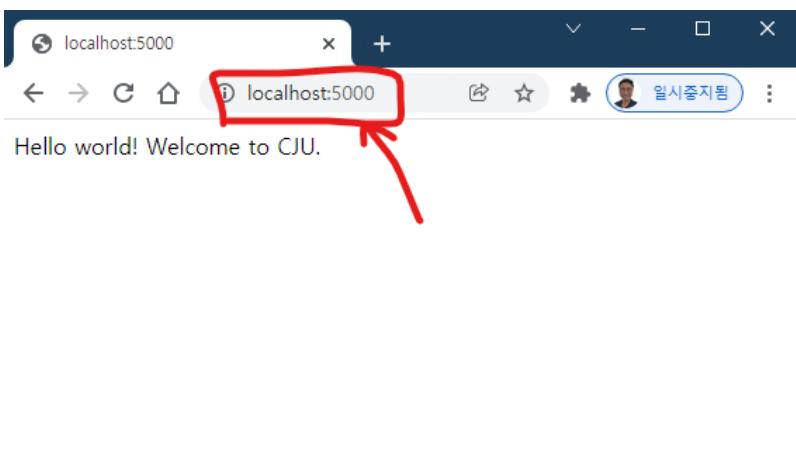


Fig. 2.6 크롬 브라우저에서 IP 대신 localhost를 입력한 결과

이제 flask 서버를 작동시키고 개발환경(디버그 모드)에서 실행할 수 있게 되었습니다.

지금까지 우리는 목표 웹 시스템 [Hello CJU \(hello_cju.py\)](#)를 코딩하고, flask 서버에 실행시키고, 웹 브라우저를 이용해 서버에 요청을 보내고, 그 결과를 웹 브라우저 화면에서 확인하였습니다.

우리가 구축한 웹 시스템은 5줄짜리 파이썬 프로그램(스크립트)입니다.

어떤 인터넷 사용자가 flask 서버로 서비스나 웹사이트 접속 요청(request)을 하는 경우 `Hello world! Welcome to CJU`라는 문자열을 보내주는(response) 아주 간단한 시스템입니다.

지금은 보잘것 없어 보이는 시스템입니다.

하지만 여러분이 여기까지 성공했다면 다음과 같은 것을 이해한 것입니다.

1. 웹 시스템에서 앱(app) 개념
2. Flask 서버 개념
3. 웹 서버 라우팅(routing) 개념
4. 서버(Flask) - 클라이언트(웹 브라우저) 개념
5. Flask 환경 설정 개념

작지만 핵심적인 경험을 바탕으로 우리는 지금까지 만든 `hello_cju` 시스템을 조금씩 업그레이드 할 것입니다.

[Next](#) 를 클릭하여 이동해 보세요.

3. Flask 기본 구조 소개

우리는 [만보기로 만드는 flask app](#)에서 실습해 본 것은 프로젝트 디렉토리에 하나의 파일 `hello_cju.py` 만을 생성해서 flask 서버를 돌렸습니다.

하지만, 직접 개발을 하기 위해서는 추가적인 사항들이 필요합니다.

예를 들면 시스템을 효율적으로 개발하기 위한 디렉토리 구조를 설계하고, 웹 시스템에서 사용할 데이터를 관리할 데이터베이스를 코딩해야 합니다. 웹 화면을 예쁘게 만들기도 해야 합니다. 사용자로부터 데이터를 받아오는 기능도 필요합니다. 아울러 데이터 보안에 대한 설정도 해주어야 합니다.

이번 장에서는 상용화 수준의 웹 시스템을 코딩하기 전에 기본적으로 갖추어야 할 내용들을 차근차근 공부해 볼 것입니다.

우리는 앞서 구현한 [만보기로 만드는 flask app](#)를 보다 구체적이고 현실적으로 만드는 기본기를 다지는 작업을 지속적으로 배울 것입니다.

이번 장에서 다룰 내용들은 모든 Flask 기반 웹 시스템에서 공통적으로 적용되는 내용이기도 합니다.

이번 장에서 학습할 내용만 이해하고 실습하더라도 기본적인 웹 시스템은 구현할 충분히 구현할 수 있습니다. 우리는 기본기를 이해하는 이번 장을 마스터한 이후에 상용화 수준의 웹에서 필요한 내용을 추가적으로 학습할 예정입니다.

이번 장에서 우리가 공부할 10가지 기본기는 다음과 같습니다.

❶ Flask 웹 시스템 구축을 위한 10가지 기본기

1. [프로젝트 기본 구조](#)
2. [Application Factory 패턴](#)
3. [Blueprint 클래스 활용](#)
4. [ORM 모델 외벽 이해](#)
5. [질문 게시판 만들기](#)
6. [게시판 댓글 구현](#)
7. [질문 & 댓글 등록 - Flask Form 활용](#)
8. [예쁘게 - CSS 적용](#)
9. [더 예쁘게 - Bootstrap 활용](#)
10. [HTML 구조와 Template 상속](#)

여러분, 준비 되셨나요?

그려면 차근차근 스텝을 밟아 보도록 하겠습니다.

Next 아이콘을 클릭하여 시작해 보세요.

3.1. 프로젝트 기본 구조

Flask는 개발자에게 높은 자유도를 부여합니다. 자유도가 높다는 것은 개발자의 입맛이나 취향에 맞게 웹 시스템을 개발할 수 있다는 뜻입니다.

웹 시스템에서 필요한 데이터베이스, HTML 페이지, 화면 처리 (view)와 같은 다양한 것들을 개발자가 자유롭게 구성할 수 있습니다.

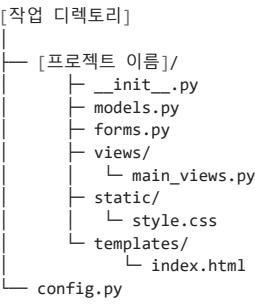
우리가 개발할 웹 시스템이 아주 간단하다면 높은 자유도는 큰 장점이 될 수 있습니다. 하지만 웹 시스템이 복잡해지면 개발을 효율적으로 진행할 수 있는 프로젝트 구조가 필요합니다.

어느 정도 이상의 규모가 되는 프로젝트에서는 [잘 만들어진 프로젝트 구조](#) 가 매우 중요합니다. 하지만 flask는 프로젝트를 구조화하는 어떠한 규칙이나 추천 사항도 없습니다. [Django](#)와 같은 프레임워크가 대부분의 개발 구조를 제공하는 반면에 Flask는 어떤 것도 제시하지 않습니다. 따라서 프로젝트 구조를 만들 때 많은 고민과 생각이 필요합니다. 물론 경험과 노하우도 필요합니다.

그렇다면, 초보자인 우리가 접근할 수 있는 방법은 무엇일까요?

네, 맞습니다. 실력있는 개발자들이 자주 사용하는 프로젝트 구조를 흉내내어 개발하는 것입니다. [모방은 창조의 어머니다.](#) 라는 말이 있지요? 처음에는 잘 정리된 프로젝트 구조를 따라할 수 밖에 없습니다. 자주 개발을 하다보면 자신만의 노하우가 생깁니다. 그때는 여러분 나름대로의 프로젝트 구조를 만들어서 사용하면 됩니다.

우리는 아래와 같은 구조를 사용할 예정입니다.



파이썬 기반 웹 개발 프레임워크인 **Django**는 새로운 프로젝트를 생성하면 위와 같은 프로젝트 구조를 자동으로 생성해 줍니다. Flask는 그렇지 못하기 때문에 개발자가 일일히 만들어 주어야 합니다.

참고로, 어떤 이름 끝에 `/`가 붙어 있다면 폴더를 의미합니다. `/`가 붙어있지 않다면 파일입니다. 예를 들어 위 프로젝트 구조에서 `views/`는 끝에 `/`가 붙어 있으므로 폴더를 의미합니다.

위에서 언급한 프로젝트 구조에 대하여 하나씩 설명하겠습니다. 당장 모든 내용을 암기할 필요는 없습니다. 우리의 공부가 진행되면서 하나하나 완성해 가므로 자연스럽게 이해할 수 있습니다.

- **[작업 디렉토리]**: 현재 여러분이 flask 웹 시스템을 코딩하기 위해 들어와 있는 디렉토리 이름입니다. 저는 `Source_codes`라는 이름을 가진 디렉토리에서 작업하고 있습니다. 여러분은 여러분이 만든 이름을 가진 디렉토리에 있을 겁니다. 그 디렉토리 이름을 의미합니다.
- **[프로젝트 이름]**: 우리가 만들 웹 시스템의 이름입니다. 디렉토리(폴더) 이름이기도 합니다. 우리는 지금까지 `hello_cju`라는 이름을 사용하였습니다. 여러분 나름대로 원하는 이름이 있다면 바꿔줘도 상관 없습니다. 다만 이름을 바꿨다면 환경 변수 설정을 할 때 바뀐 이름을 사용해야 합니다.
- **`__init__.py`**: **[프로젝트 이름]** 디렉토리 안에 있는 파일들을 패키지처럼 사용할 수 있게 해주는 파일입니다. **[프로젝트 이름]**에 속한 파일들이 공통으로 초기화 해야 하는 내용이 담길 수도 있으며, **[프로젝트 이름]**의 원래 내용을 담을 수도 있습니다. 우리는 이전에 만들었던 `hello_cju.py`의 내용을 `__init__.py`에 옮겨 담아서 `__init__.py`가 `hello_cju.py` 역할을 하도록 해 줄 예정입니다.
- **`models.py`**: 우리는 웹 시스템에서 사용할 데이터베이스의 query 언어를 이해하지 못해도 데이터베이스를 쉽게 사용할 수 있는 ORM(Object Relational Mapping) 기술을 사용할 것입니다. ORM을 편리하게 사용할 수 있도록 지원하는 `SQLAlchemy`라는 패키지를 사용할 것입니다. 우리가 개발할 프로젝트에서 사용할 데이터베이스 모델을 정의해 줄 파일이 필요합니다. `models.py` 파일은 데이터베이스 모델을 정의한 내용이 담길 파일입니다.
- **`forms.py`**: 웹 시스템을 구현하다 보면 인터넷 사용자가 크롬과 같은 웹 브라우저에 입력한 데이터를 flask 서버 쪽으로 가져와야 하는 작업을 해야 합니다. 이때 사용하는 것이 `form`이라는 것입니다. 우리는 `WTForms`라는 라이브러리를 활용해 데이터를 받아올 것입니다. `WTForms`에서 제공하는 `Form` 클래스를 어떻게 정의할지에 대한 내용이 담길 파일입니다.
- **`views/`**: 이름 끝에 `/` 붙어 있으니 디렉토리(폴더)입니다. 인터넷 사용자가 접속 했을 때 flask 서버는 어떤 작업을 하고 그 결과를 사용자의 웹 브라우저에 뿌려줘야 합니다. 서버에서 어떤 작업을 처리하는 것을 `view`에서 처리한다고 표현합니다. `view` 작업을 정의한 파일들을 모아 놓은 디렉토리(폴더)입니다. 위 예제에서는 `main_view.py`라는 모듈이 하나 포함되어 있는 경우입니다. 우리는 이 폴더에 여러 개의 `view` 파일을 추가해 나갈 것입니다.
- **`static/`**: 이름 끝에 `/` 붙어 있으니 디렉토리(폴더)입니다. `static`이라는 의미는 정적, 고정된이라는 의미를 가지고 있죠? 말 그대로 정적인 파일들을 모아 놓을 디렉토리(폴더)입니다. 정적인 파일들은 HTML 내용에 효과를 입히는 CSS(Cascade Style Sheet), Java Script (.js 파일), 이미지 파일 (.png, .jpg 등) 등이 있습니다. 우리는 이 폴더에 정적인 파일들을 차곡차곡 모아 놓을 것입니다. 위 예제에는 `style.css`라는 파일 1개만 들어 있는 상황입니다.
- **`templates/`**: 이름 끝에 `/` 붙어 있으니 디렉토리(폴더)입니다. 웹 시스템은 기본적으로 웹 브라우저를 이용합니다. 웹 브라우저의 내용은 HTML 언어로 작성된 문서로 제공됩니다. HTML 언어로 작성된 문서는 `.html`이라는 확장자가 붙게 됩니다. 우리는 다양한 웹페이지 파일 `.html` 파일들을 이 디렉토리에 모아 놓을 것입니다. 위 예제에는 `index.html` 이름을 가진 HTML 파일이 하나 담겨있는 상황입니다.
- **`config.py`**: 웹 시스템을 개발하다 보면 개발 프로젝트에 적용할 환경 설정을 이것 저것 하게 됩니다. `config`는 `configuration`의 약자입니다. `configuration`은 컴퓨터 시스템을 작동시키기 위한 설정을 의미합니다. 개발 과정에서 여기 저기에 환경 설정에 관련한 내용이 흩어져 있다면 관리하기 참 힘들겠죠? 개발에 필요한 환경 설정 내용을 모아 놓은 파일입니다.

여러분이 필요하다면 추가적으로 디렉토리(폴더)나 파일들을 얼마든지 추가할 수 있습니다. 자유롭게 추가하는 것은 여러분이 많은 개발 경험과 노하우를 쌓은 후에 해도 늦지 않습니다. 일단 우리는 위에서 제시한 프로젝트 구조를 가지고 학습을 진행해 나가겠습니다.

VS code의 File Explore 영역에서 마우스 오른쪽 클릭을 하거나, `File` 메뉴를 클릭하여 위에서 제시한 것과 같이 프로젝트 구조를 만들어 봅니다.

제가 만든 VS code에서의 프로젝트 구조는 그림 [Fig.3.1](#)와 같습니다.

그림 [Fig.3.1](#)을 자세히 살펴보면 `__pychache__`라는 디렉토리(폴더)가 자동으로 생긴 것을 볼 수 있습니다. Python은 실행하기 전에 소스코드를 바이트코드로 먼저 바꾸게 됩니다. 이렇게 바뀐 파일들은 바이트코드는 `.pyc`, 최적화된 바이트코드는 `.pyo`라는 확장자를 갖게 됩니다. Python을 실행하기 위해 바이트코드로 변경된 파일들을 모아 두는 디렉토리(폴더)가 바로 `__pychache__`입니다. 프로그래머는 신경쓰지 않아도 됩니다. 삭제하게 되면 다시 생깁니다.

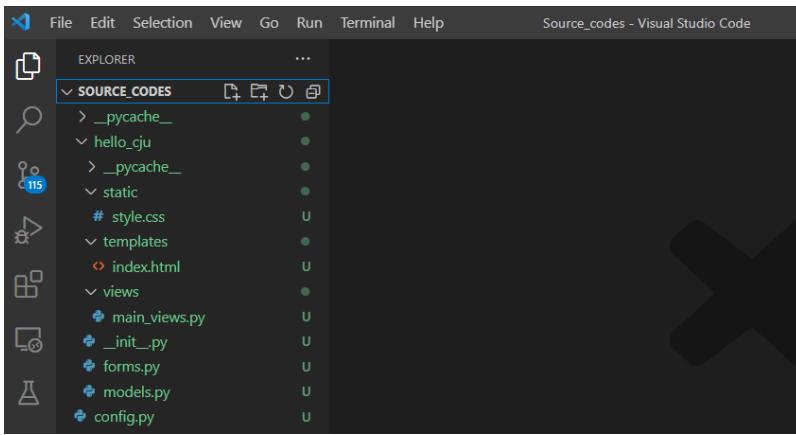


Fig. 3.1 VS code에서 프로젝트 구조 생성

이제 flask 프로젝트 구조에 대하여 구체적으로 살펴봤습니다.

우리가 앞으로 작업할 내용들에 대한 밑그림이 그려지나요?

지금 당장 모든 것을 이해할 수 없어도 상관 없습니다. 앞으로 계속해서 반복적으로 우리가 다루게 될 구조이므로 차차 알아가면 됩니다.

"아! flask를 개발할때 대략 이런 프로젝트 구조를 사용하는구나!" 정도면 충분합니다.

여러분, 이제 다음 기본기를 배울 준비가 되셨나요?

그러면 차근차근 스텝을 밟아 보도록 하겠습니다.

Next 아이콘을 클릭하여 시작해 보세요.

3.2. Application Factory 패턴

우리는 [만보기로 만드는 flask_app](#)에서 목표 시스템 **Hello CJU (hello_cju.py)**를 만들어 보았습니다.

웹 시스템에서 제공하는 서비스를 하기 위해서는 **Flask** 클래스로부터 **app** 객체를 생성하였습니다.

app 객체를 생성하는 명령은 다음과 같았습니다.

```
app = Flask(__name__)
```

기억 나시나요 .^.^. ?

이러한 구조는 웹 시스템에서 서비스 할 애플리케이션(앱)이 몇 개 안될 경우에는 큰 문제가 되지 않습니다. 하지만 다양한 기능을 애플리케이션(앱) 단위로 제공하고 싶다면 문제가 발생하게 됩니다.

현재 **Hello CJU** 시스템에서는 **Hello world! Welcome to CJU.**라는 문자열만 짠다주는 기능(앱) 밖에 없지만, 필요에 따라서 **Hello CJU** 시스템에 정보조회 기능(앱), 회원가입 기능(앱), 게시판 기능(앱), 사진등록 기능(앱) 등 다양한 기능, 즉 다양한 앱을 추가할 수 있습니다. 이럴 경우 문제가 생깁니다.

어떤 문제가 생기나요?

우리가 코딩한 방식은 객체를 전역(global) 객체를 생성했다는 점이 문제입니다. 전역 객체는 어떤 객체, 함수, 변수 할당 등에서 자유롭게 접근할 수 있습니다. 누가 어떻게 앱을 건드렸는지 알 수 없게 되고, 심각한 오류가 발생할 수 있습니다.

다양한 기능을 코딩하다 보면 **B**라는 모듈을 구현하기 위해 **A**라는 앱이 가지고 있는 기능을 **import**하여 구현할 수 있습니다. 그렇게 **C, D, ...** 와 같은 모듈들을 계속해서 코딩해 갑니다. 그러다가 **A**라는 모듈에 추가 기능을 구현하다 보니 **B**라는 모듈의 기능이 필요하여 **B**를 **import** 할 수 있습니다.

위 상황을 정리하면,

A는 **B**를 임포트하고,

B는 **A**를 임포트하는 상황입니다.

많이 보던 상황이죠? 그렇습니다. 이런 경우를 **순환 참조 (circular import)**라고 부릅니다.

2개의 모듈이 순환 참조를 할 수도 있고, 3개 이상의 모듈이 순환 참조 구조를 만들 수도 있습니다.

순환 참조 상황이 발생하면 Python은 헷갈리겠죠? 결국 Python은 다음과 같이 할 수밖에 없습니다.

어떻게 하라는 거지? 예라 모르겠다. 순환 참조 오류를 발생시켜서 프로그래머가 수정하도록 안내하자!

Django나 Flask와 같은 웹 개발 프레임워크에서 자주 발생합니다.

지금이야 예를 든 모듈의 개수가 몇 개 안되니 관리가 되지만 서비스나 기능이 복잡해지다 보면 항상 생길 수 있는 문제입니다.

이를 방지하기 위한 접근법으로 flask는 **Application Factory** 패턴을 사용하라고 권고하고 있습니다. 자세한 내용은 flask 공식 문서 Application Factories ([클릭](#))를 참고하기 바랍니다.

Application Factory 패턴은 `app` 객체를 전역으로 생성하지 않고, 별도의 생성 함수를 만들고 그 함수에서 `app` 객체를 생성해서 리턴하는 형태의 코딩 방법입니다. 이렇게 하면 순환 참조 에러를 대부분 방지할 수 있습니다.

개념을 잡았으니 실습해 볼까요?

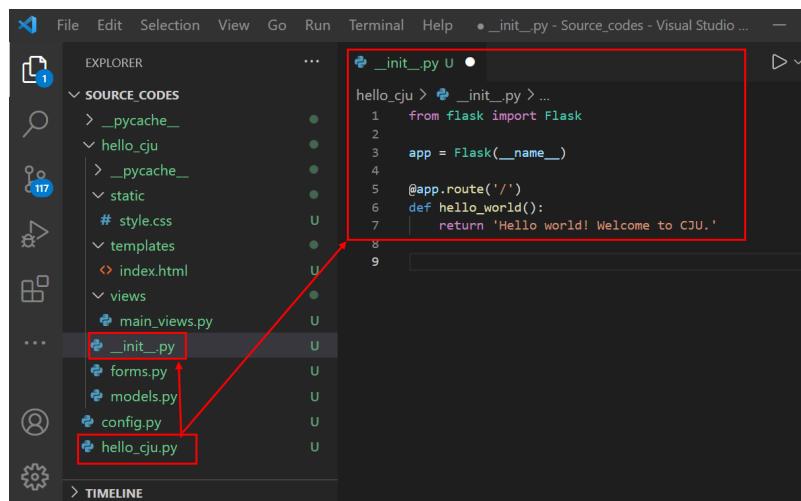


Fig. 3.2 VS code에서 `_init__.py`를 이용한 설정

그림 Fig. 3.2 확인해 보세요.

제가 만들었던 `hello_cju.py`에 있는 내용을 그대로 복사하여 `hello_cju/` 디렉토리 밑에 있는 `_init_.py`에 붙여 넣었습니다.

그런 다음 `hello_cju.py` 파일을 삭제합니다.

`CTL + C`를 눌러서 flask 서버를 중지하고 `flask run` 명령어를 입력하여 flask 서버를 시작하면 정상적으로 작동하는 것을 확인할 수 있습니다.

① 서버 재실행과 flask 환경 설정

혹시라도 서버가 작동하지 않는다면 `set FLASK_APP=[앱 이름]` (우리가 진행했던 실습의 경우 `set FLASK_APP=hello_cju`)를 다시 한번 입력하고 실행하면 정상적으로 작동할 것입니다. 우리가 지금까지 활용한 환경변수 설정은 flask 서버가 작동하는 동안에 유효합니다. 서버를 다시 시작하면 우리가 설정했던 `set FLASK_APP=[앱 이름]` 설정이 초기화되어 더 이상 작동하지 않습니다.

만약 여러분의 앱 이름을 `app`이라고 지어 주었다면 별도의 `set FLASK_APP=[앱 이름]` 실행을 하지 않더라도 정상적으로 서버가 실행됩니다. 이는 flask의 기본 세팅이 `set FLASK_APP=app`으로 되어 있기 때문입니다.

`set FLASK_ENV=development` 설정도 마찬가지입니다. 서버를 재실행 하게 되면 환경변수를 다시 입력해 주어야 합니다.

① 반복적인 입력이 귀찮을 때 - 배치(batch) 파일

윈도우 환경에서 .cmd, 리눅스 환경에서 쉘 크립트 (batch file) .bat 를 생성해서 간단히 실행할 수도 있습니다. 윈도우의 경우 환경변수를 세팅해 주어야 합니다. 배치 파일을 만들면 편리하지만 프로젝트가 여러 개 일 경우 배치 파일이 헷갈립니다. 우리는 번거롭더라도 서버를 재가동 할때마다 직접 입력하여 속달하는 방식으로 학습을 진행하겠습니다.

서버 재가동 시 2줄 명령어 입력이 너무 귀찮게 느껴지거나 배치파일에 대해 추가로 공부하고 싶은 사람은 다음 링크를 참고하세요.

- 윈도우 계열 배치 파일: 점프 투 플라스크 - 배치 파일 ([click](#))
- 리눅스 계열 배치 파일: [Linux] 쉘 스크립트(Shell script) 기초 ([click](#))

작동 결과는 그림 Fig.3.3과 같습니다.

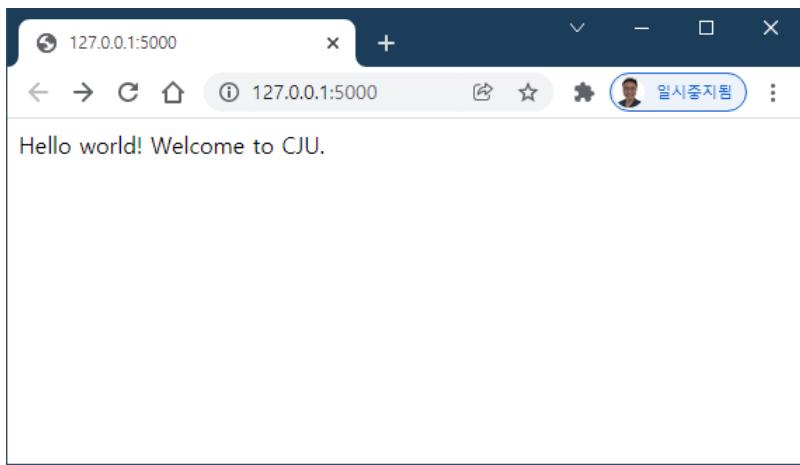


Fig. 3.3 __init__.py 설정 후 브라우저 접속 결과

그림 Fig.3.3를 통해 flask 서버가 참조하던 hello_cju.py 모듈을 hello_cju/__init__.py 모듈로 변경된 것을 알 수 있습니다. 이를 통해 별다른 코드 변경 없이 __init__.py 모듈을 통해 서버를 정상적으로 작동할 수 있음을 확인하였습니다.

이제 __init__.py의 내용에 application factory 패턴을 적용해 보겠습니다.

다음과 같이 __init__.py의 코드를 변경합니다.

```
from flask import Flask

def create_app(): # 함수 생성
    # 기존에 hello_cju.py에 있던 영역
    app = Flask(__name__)
    @app.route('/')
    def hello_world():
        return 'Hello world! Welcome to CJU.'

    # create_app 함수가 생성한 app을 리턴
    return app
```

위 코드는 create_app() 함수를 이용하여 앱(app)을 생성하고, 생성된 앱(app)을 리턴합니다.

수정한 __init__.py 코드를 저장하고 flask 서버를 재실행하면 정상적으로 잘 돌아가는 것을 확인할 수 있습니다.

② Note

__init__.py 의 내용 중 create_app 함수를 사용하여 application factory 패턴을 구현하였습니다. 주의할 것은 create_app 함수 이름을 다른 것으로 사용하면 안된다는 것입니다.

Flask 내부적으로 application factory 패턴을 구현하기 위한 함수 이름은 create_app 으로 사전에 내부적으로 정의되어 있기 때문입니다. 이렇게 사전에 정의된 내용을 준수해야 하는 것이 프레임워크(framework)의 특성 중 하나이기도 합니다.

혹시라도 create_app 함수 이름이 마음에 들지 않더라도 준수하기 바랍니다.

우리는 application factory 패턴을 적용하여 우리의 앱 시스템을 작동하는 것까지 살펴 보았습니다. 다음으로 살펴볼 기본기는 웹페이지의 경로를 컨트롤하는 라우팅 방법 중에서 블루프린트 클래스를 활용하는 방법에 대하여 살펴볼 것입니다.

여러분, 이제 다음 기본기를 배울 준비가 되셨나요?

그리면 차근차근 스텝을 밟아 보도록 하겠습니다.

[Next](#) 아이콘을 클릭하여 시작해 보세요.

3.3. Blueprint 클래스 활용

우리는 다음과 같이 기본기 2가지를 공부하였습니다.

❶ Flask 웹 시스템 구축을 위한 10가지 기본기

1. [프로젝트 기본 구조](#) \(\backslash\to\backslash\) **Clear!**
2. [Application Factory 패턴](#) \(\backslash\to\backslash\) **Clear!**
3. [Blueprint 클래스 활용](#) \(\backslash\to\backslash\) 지금 도전!
4. [ORM 모델 완벽 이해](#)
5. [질문 게시판 만들기](#)
6. [게시판 댓글 구현](#)
7. [질문 & 댓글 등록 - Flask Form 활용](#)
8. [예쁘게 - CSS 적용](#)
9. [더 예쁘게 - Bootstrap 활용](#)
10. [HTML 구조와 Template 상속](#)

이번에 살펴볼 기본기는 블루프린트 (Blueprint) 클래스를 활용해서 웹사이트의 경로를 효율적으로 관리하는 방법입니다.

그렇다면 왜 Blueprint 클래스를 사용하는 것일까요?

그 이유를 찾기 위해 Application factory 패턴을 다시 한번 살펴 볼까요?

`hello_cju/__init__.py`에 있는 `create_app` 함수 내부에 우리가 서비스하고자 하는 웹페이지에 해당하는 URL 경로를 /에 매칭시켰습니다. 참고로 우리가 처음 구현했던 `hello_cju.py`는 사용자가 접속하면 웹페이지에 `Hello world! Welcome to CJU.`라는 문자열을 출력하는 간단한 서비스였습니다.

`hello_cju/__init__.py`에서는 해당 서비스의 경로를 `@app.route('/')`라는 데코레이터를 이용하여 설정하였습니다. `hello_cju.py` 도 마찬가지입니다.

❷ Note

Flask에서 `@app.route` 데코레이터를 이용하여 웹사이트의 경로를 매핑(설정)하는 함수를 **라우트(route) 함수**라고 부릅니다.

그런데, 웹 시스템을 개발하다 보니 기능을 추가하여 해당 기능을 다른 페이지에, 즉 웹사이트 경로를 이용하여 서비스를 해야할 필요가 생겼습니다.

우리가 배운대로라면 새로운 웹사이트 경로가 필요할 때마다 `create_app` 함수 내부에서 라우트 함수 `app.route`를 이용해 계속해서 추가해야 합니다.

웹사이트 구조가 간단한 경우라면 별로 문제가 되지 않습니다. 하지만 웹사이트 기능과 서비스가 다양해지면 지금과 같은 방식으로는 효율적으로 개발하기 어려워집니다. 설령 개발을 끝마쳤다고 하더라도 유지보수 단계에서 많은 번거로운 작업을 해야 되므로 비효율적입니다.

이러한 개발 및 유지보수 효율성을 높일 수 있는 방법이 Blueprint 클래스를 이용하는 것입니다.

❸ 블루프린트(Blueprint)

블루프린트는 우리나라 말로 **청사진**이라는 뜻입니다. 캐드(CAD)와 같은 설계 소프트웨어가 없던 예전에는 설계도를 푸른색 종이에 그렸다고 합니다. 그래서 푸른색 종이, 청사진이 설계도를 의미하는 단어로 쓰이게 되었습니다.

Flask에서 블루프린트는 웹사이트에 접속하는 URL과 그 때 호출되는 함수와의 관계를 나타내는 일종의 **함수 - URL 설계도**로 이해하면 됩니다.

대략적인 개념은 다음과 같습니다.

- 각 웹사이트 페이지별로 수행할 작업들은 `views.py`라는 별도의 모듈에 작성합니다. `views.py` 모듈에서 Blueprint 클래스를 이용하여 각각의 함수와 호출 URL을 매칭시킵니다.
- `hello_cju/__init__.py`에서는 각 웹페이지별 함수 기능과 URL을 전부 기록하지 않고 `views.py` 모듈에서 Blueprint 객체에 매칭된 URL을 등록해 주는 역할만 담당합니다.

감이 잘 안잡히죠?

예제를 통해 Blueprint 클래스의 사용법과 작동 방식에 대하여 살펴보도록 하겠습니다.

먼저 `views` 디렉토리에 만들어 놓았던 `main_views.py` 파일을 열고 아래 코드를 입력합니다. `main_views.py` 파일이 없다면 `views` 디렉토리 밑에 새로 만듭니다.

```
# Blueprint 클래스를 임포트 합니다.
from flask import Blueprint

# Blueprint 객체 bp를 생성합니다.
bp = Blueprint('main', __name__, url_prefix='/')

# Blueprint 객체 bp를 이용하여 함수와 URL을 매칭합니다.
@bp.route('/')
def hello_world():
    return 'Hello world! Welcome to CJU.'
```

위 코드를 간단히 분석해 보면, 먼저 `Blueprint` 클래스를 임포트하고 `bp`라는 이름을 가진 `Blueprint` 객체를 생성하였습니다.

`@bp.route('/')`를 이용하여 `hello_world` 함수의 결과를 웹사이트 경로 `/`에 맵핑하였습니다.

위 코드는 `hello_cju/__init__.py`에 담긴 내용을 `views/main_view.py`로 옮겨 놓은 것입니다. 차이점은 `@app.route` 데코레이터를 사용하는 대신 `@bp.route`를 사용한다는 것입니다.

`Blueprint` 객체를 생성할 때 `bp = Blueprint('main', __name__, url_prefix='/')` 전달된 `main`, `__name__`, `url_prefix='/'`에 대하여 살펴보겠습니다.

`Blueprint` 클래스를 이용하여 객체를 생성할 때 다양한 인자값을 전달하여 생성할 수 있습니다. 자세한 내용은 flask 공식 문서의 [Blueprint \(클릭\)](#)를 참고하기 바랍니다.

```
class flask.Blueprint(
    name,
    import_name,
    static_folder=None,
    static_url_path=None,
    template_folder=None,
    url_prefix=None,
    subdomain=None,
    url_defaults=None,
    root_path=None,
    cli_group=<object object>
)
```

Parameters:

- `name` (str)

The name of the blueprint.
Will be prepended to
each endpoint name.
- `import_name` (str)

The name of the blueprint package,
usually `__name__`.
This helps locate the `root_path`
for the blueprint.
- `url_prefix` (Optional[str])

A path to prepend to all of
the blueprint's URLs,
to make them distinct from
the rest of the app's routes.
- `root_path` (Optional[str])

By default, the blueprint will
automatically set this based on
`import_name`.
In certain situations this automatic
detection can fail, so the path can
be specified manually instead.

Flask 공식 문서([클릭](#))에는 모든 인자에 대한 설명이 제시되어 있습니다. 위 코드는 우리가 사용할 3개 인자 `main`, `import_name```, `url_prefix=`에 대한 설명만 표시하였습니다.

3개 인자 중에서 `main`, `import_name`은 반드시 제공해야 하는 값입니다. `url_prefix=` 인자는 값을 제공해도 되고 생략해도 가능합니다. 생략하게 되면 디폴트(default) 값으로 `None`이 자동 지정됩니다. 각 인자값들이 의미하는 것에 대해 살펴보도록 하겠습니다.

- `name: Blueprint` 이름입니다. 각각의 엔드포인트(endpoint) 앞에 붙게 됩니다. 나중에 `url_for()`를 이용하여 개별 함수가 갖는 경로를 쉽게 추출할 수 있습니다.
만약 `Blueprint` 객체의 `name` 인자로 `main`이 전달되었다면 `main.[함수명]` 형태로 사용하게 됩니다.
- `import_name: Blueprint` 패키지의 하부에 설정되는 모듈 이름입니다. 관용적으로 `__name__`을 사용합니다. `__name__`을 `import_name`의 인자값으로 지정하면, `@bp.route()` 데코레이터(라우팅 함수) 바로 다음에 나오는 함수의 이름이 `Blueprint` 객체에 전달됩니다.
만약 `name`의 인자값으로 `main`이 전달되고, `import_name`의 인자값이 `__name__`으로 전달되고 바로 다음에 나오는 함수 이름이 `hello_cju`라고 한다면 `url_for` 함수를 이용해 웹페이지 주소를 추출하고자 할 경우 `url_for('main.hello_cju')`와 같이 사용합니다.
주로 `Blueprint`가 `root_path`를 위치시키는데 도움을 주는 역할을 합니다.
- `url_prefix=`: 모든 `Blueprint`에 공통적으로 적용되는 URL 패턴입니다. URL 프리픽스(`url_prefix`)는 `main_views.py` 파일에 있는 함수들의 URL 앞에 항상 붙게 되는 프리픽스 URL을 의미합니다.
만약 위에서 `url_prefix='/'` 대신 `url_prefix='/main'`이라고 선언했다면 `hello_pybo` 함수를 호출하기 위해서는 <http://localhost:5000/> 대신 <http://localhost:5000/main>이라고 호출해야 합니다.

❶ 엔드포인트(endpoint)란?

엔드포인트는 인터넷 접속 주소의 마지막에 붙는 문자열을 의미합니다. 예를 들어 `Daum` 포털에서 뉴스 페이지를 검색한다고 가정해 봅니다. `Daum` 뉴스는 `news.daum.net` 주소 아래에 분야별로 분류해 놓게 됩니다. 사회 뉴스는 `society`, 정치 뉴스는 `politics`, 경제 뉴스는 `economic` 등과 같이 분류합니다. 각각의 접근 URL은 다음과 유사한 형태를 갖습니다.

- 사회 뉴스: `news.daum.net/society`
- 정치 뉴스: `news.daum.net/politics`
- 경제 뉴스: `news.daum.net/economic`: 위 예제에서 `society`는 `news.daum.net`의 사회 뉴스 엔드포인트가 됩니다. `politics`는 `news.daum.net`의 정치 뉴스 엔드포인트가 됩니다. `Daum` 뉴스의 경우 각 분야별로 엔드포인트가 존재하겠죠?

이렇게 URL을 이용해서 서버가 제공하는 자원(resource)에 접근하도록 제공하는 것을 엔드포인트(endpoint)라고 부릅니다. 엔드포인트 개념은 REST(Representational State Transfer) API와 관련이 있습니다. 참고할 만한 REST API 블로그 ([클릭](#)).

이제 `hello_cju/__init__.py` 파일을 아래와 같이 수정합니다.

```
# 파일명: hello_cju/__init__.py

from flask import Flask

def create_app(): # 함수 생성
    # 기존에 hello_cju.py에 있던 영역
    app = Flask(__name__)

    # views/main_view.py 모듈을 임포트합니다.
    from .views import main_views

    # Flask 객체 app에 main_view 모듈에 있는
    # Blueprint 객체 bp를 등록합니다.
    app.register_blueprint(main_views.bp)

    # bp 객체를 app에 등록했으므로
    # 기존에 있던 코드는 삭제합니다.
    # @app.route('/')
    # def hello_world():
    #     return 'Hello world! Welcome to CJU.'
    # create_app 함수가 생성한 app을 리턴

    return app
```

파일을 저장하고 flask 서버를 재시작 한 후에 <http://localhost:5000/>로 접속을 시도하면 정상적으로 잘 작동하는 것을 확인할 수 있습니다.

`Blueprint` 클래스 개념을 잡았으니 내친김에 별도 경로를 갖는 웹페이지와 해당 페이지 URL이 호출되었을 경우 처리할 함수를 하나 만들어 보겠습니다.

전공을 소개하는 웹페이지를 하나 더 만들도록 하겠습니다. 전공 소개 웹사이트 URL은 `/major`로 지정하고, 웹페이지에 표시할 내용은 `/views/main_view.py` 모듈에 `intro_major` 함수로 만들어서 추가하겠습니다.

앞에서 작성한 `/views/main_view.py` 모듈에 아래와 같이 내용을 추가해 봅니다.

```
# 파일명: views/main_views.py

# 전공소개 페이지 추가
@bp.route('/major')
def intro_major():
    return '우리 전공은 인공지능소프트웨어입니다.'
```

VS code에서 작업한 화면은 그림 Fig.3.4와 같습니다.

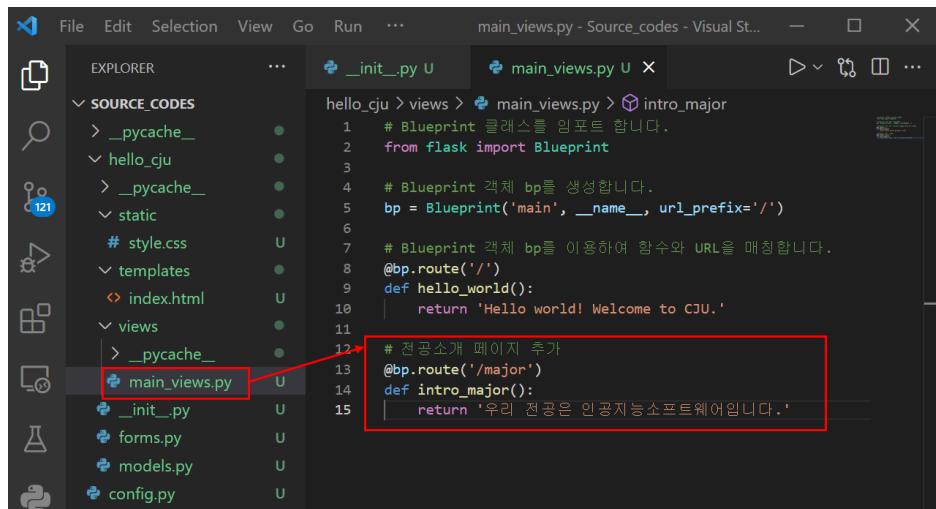


Fig. 3.4 전공 소개 페이지 코드를 추가

크롬과 같은 웹 브라우저 주소창에 `127.0.0.1:5000/major` 또는 `localhost:5000/major`라고 입력하고 엔터를 누릅니다.

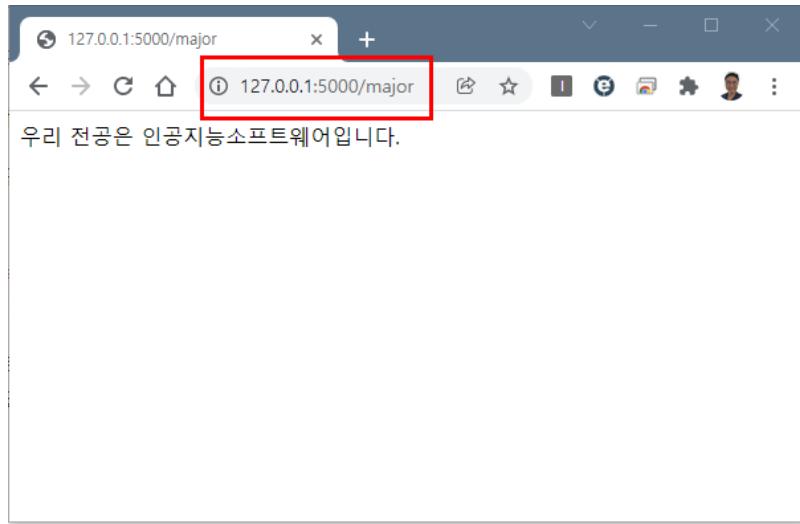


Fig. 3.5 Blueprint를 이용해 추가된 새로운 웹페이지 접속

그림 {numref}`sec03_05_chorme_major_page` 같이 flask 서버의 응답(response) 결과를 확인할 수 있습니다. 브라우저에 표시된 `우리 전공은 인공지능소프트웨어입니다.`라는 문자열은 우리가 `views/main_view.py`에 추가한 `python def intro_major():` 함수에서 처리하여 리턴한 내용입니다. 새로운 웹페이지 주소 `/major`를 추가했지만 `__init__.py` 내용을 수정하지 않고 편리하게 구현할 수 있게 되었습니다.

여기까지 이해한다면 여러분은 다양한 페이지를 만들어서 웹 시스템으로 서비스할 능력을 갖춘 것입니다.

축하드립니다. 짹짜짜!

우리가 지금까지 구현한 시스템은 내부적으로 데이터를 관리하거나 인터넷 사용자로부터 데이터를 가져올 수 없는 구조입니다.

이를 해결하기 위해서는 웹 시스템에서 사용할 데이터베이스가 필요합니다.

여러분, 이제 다음 기본기를 배울 준비가 되셨나요? 다음 기본기는 바로 데이터베이스 관련 내용들입니다.

그러면 차근차근 스텝을 밟아 보도록 하겠습니다.

[Next](#) 아이콘을 클릭하여 시작해 보세요.

3.4. ORM 모델 완벽 이해

우리는 다음과 같이 기본기 3가지를 공부하였습니다.

❶ Flask 웹 시스템 구축을 위한 10가지 기본기

1. [프로젝트 기본 구조](#) \(\backslash to\backslash Clear!
2. [Application Factory 패턴](#) \(\backslash to\backslash Clear!
3. [Blueprint 클래스 활용](#) \(\backslash to\backslash Clear!
4. [ORM 모델 완벽 이해](#) \(\backslash to\backslash\) 지금 도전!
5. [질문 게시판 만들기](#)
6. [게시판 댓글 구현](#)
7. [질문 & 댓글 등록 - Flask Form 활용](#)
8. [예쁘게 - CSS 적용](#)
9. [더 예쁘게 - Bootstrap 활용](#)
10. [HTML 구조와 Template 상속](#)

이번에 살펴볼 기본기는 ORM (Object Relational Mapping) 모델입니다.

ORM 이야기하기 전에 먼저 데이터베이스 이야기를 해볼까 합니다.

웹 시스템을 구현하기 위해서는 데이터를 생성하고, 읽어오고, 수정하고(업데이트), 삭제해야 하는 일이 반드시 필요합니다.

웹 시스템에 구현된 게시판의 예를 들어 볼까요? 인터넷 사용자는 게시판에 글을 쓰고(Create), 자신이 쓴 글이나 다른 사람 글을 조회하여 읽어보고(Read), 자신이 쓴 내용을 수정하고(Update), 더 이상 필요하지 않거나 원하지 않는 글을 삭제(Delete) 해야 합니다.

데이터와 관련된 작업들을 CRUD(Create, Read, Update, Delete)라고 줄여서 부릅니다.

온라인샵에서 상품 관리, 회원관리, 게시물 관리, 등등 데이터를 이용할 CRUD 작업을 해야 할 상황은 항상 발생한다고 해도 과언이 아닙니다.

데이터를 잘 정리해 놓은 것을 [데이터베이스](#)라고 부르고 영어로는 DB(Database)라고 부릅니다.

데이터베이스를 컴퓨터로 관리해 주는 시스템을 [데이터베이스 관리 시스템](#)이라고 부르고 영어로는 DBMS(Database Management Systems)라고 부릅니다. DBMS의 종류로는 MySQL, Oracle, SQLite 등 다양한 종류가 존재합니다. 프로그래머는 개발하려는 웹 시스템의 특징에 맞는 DBMS를 골라 쓰면 됩니다.

컴퓨터를 이용해 DB의 CRUD 작업을 하기 위한 언어가 필요합니다. 우리가 웹 시스템을 개발하기 위해 Python 언어가 필요한 것처럼 DB를 개발하기 위한 언어를 [Query Language \(쿼리 언어\)](#)라고 합니다.

Query는 사전적 의미로 [질문하다, 문의하다](#)의 뜻을 가지고 있습니다. DB에서는 쿼리를 [질의](#)라는 한국어를 더 자주 사용합니다. 프로그래머가 컴퓨터가 가지고 있는 DB에 질의를 한다는 의미로 이해하면 좋습니다. 컴퓨터에 있는 DB에게 다음과 같은 [질의](#)를 하는 것입니다. 아마도 다음과 같은 [질의](#)가 가능할 것 같습니다.

- 생성(Create): DB야, [홍길동](#)이 작성한 글을 생성해 줄래?
- 조회(Read): DB야, [홍길동](#)이라는 사람이 작성한 글을 보여줄 수 있니?
- 수정(Update): DB야, 지난번에 [홍길동](#)이 작성한 글을 수정해 줄래?
- 삭제(Delete): DB야, [홍길동](#)이 어제 작성한 글을 삭제해 줄수 있어?

위와 같이 DB에게 [질의](#) 하면, DB는 해당되는 일을 수행하고 결과를 알려주겠죠? 그렇다고 무턱대고 DB에게 물어볼 수 없으니, DB에서 사용할 수 있도록 언어를 만들게 된 것입니다. 쿼리와 비슷한 의미로 SQL(Structured Query Language)라는 단어를 사용하기도 합니다. 사실 아래 단어는 모두 정확히 같은 말입니다.

- Query
- SQL
- 쿼리
- 질의문

Note

이 글을 읽는 독자들은 SQL과 DB에 대한 기본 지식을 알고 있다고 가정합니다. 추가 학습이 필요한 사람은 개별적으로 블로그나 교재를 활용하여 공부하면 좋습니다.

DB는 일반적으로 표 형태로 구성됩니다. 쉽게 말하면 MS 엑셀로 작성한 자료 파일과 비슷한 형태입니다.

id	title	contents
1	ORM 모델	ORM 모델에 대하여 설명합니다.
2	딥러닝	인공지능 딥러닝의 원리
:	:	:

위 DB에서 DB 이름이 **information**이라고 할 때(엑셀 파일명과 비슷한 개념), 제목(title)이 **Flask 소개**, 내용(contents)에 **Flask 기초에 대하여 설명합니다.**라는 데이터를 SQL을 이용해 생성하려면 다음과 같은 형태가 됩니다.

```
insert into information (title, contents) values ('Flask 소개', 'Flask 기초에 대하여 설명합니다.')
```

위 예시 이외에도 CRUD를 하기 위한 다양한 SQL 문법이 존재합니다. 문제는 웹 시스템 개발자들이 SQL 문법을 추가로 학습해야 한다는 것이죠... 시스템 개발하기도 빠듯한데 DB 언어인 SQL까지 배워야 한다면 부담이 많이 됩니다.

그래서 등장한 것이 ORM입니다. ORM은 DB를 하나의 객체로 보고 DB 객체에 데이터와 관련된 연산을 할 수 있도록 해 줍니다. 하나의 DB 테이블 (하나의 엑셀 sheet과 비슷한 개념)을 **ORM 모델**이라고 부릅니다. Python의 클래스 개념만 알고 있다면 추가적인 SQL 공부 없이도 충분히 시스템을 개발할 수 있습니다. DB나 SQL을 모르는 사람들에게는 환상적이겠죠?

ORM의 핵심 정리

DB 테이블을 클래스로 만들고, 클래스와 객체 활용 개념을 활용하여 CRUD를 지원하는 기술입니다.

ORM은 클래스 개념을 적용한다고 했죠? 그러면 ORM 객체를 만들고 객체가 갖는 메서드를 이용해 CRUD를 수행하면 그만입니다. 복잡한 SQL이 필요없어집니다. 위에서 제시한 SQL 예제를 ORM 스타일로 코딩하면 다음과 같습니다.

```
# 객체 생성
information = Information(
    title='Flask 소개',
    contents='Flask 기초에 대하여 설명합니다.',
)

# 데이터 객체를 DB에 삽입
db.session.add(information)
```

ORM 모델을 사용하면 모델 내부에서 SQL 사용에 필요한 쿼리를 자동으로 처리해 주기 때문에 개발자는 쿼리에 대하여 신경쓰지 않아도 됩니다. 게다가 ORM은 DBMS가 다르더라도 동일한 메서드 형태를 제공하므로 코드의 일관성을 유지할 수 있습니다. 코드의 일관성을 유지할 수 있다는 것은 유지보수에서 큰 장점입니다.

Note

우리는 DB, SQL을 몰라도 웹 시스템을 개발할 수 있는 ORM을 사용합니다. 하지만 개발자라면 DB와 SQL에 대한 지식을 갖고 있어야 합니다.

ORM 관련 내용을 아래와 같은 순서로 공부해 나갈 예정입니다.

① ORM 완벽이해 5단계

- ORM 1. [Flask ORM 설치](#)
- ORM 2. [모델 설계 및 만들기](#)
- ORM 3. [DB 테이블 만들기](#)
- ORM 4. [시각화 도구로 모델 확인](#)
- ORM 5. [모델에 CRUD 해보기](#)

3.4.1. Flask ORM 설치

이 세상에는 다양한 개발 언어 programming language가 있습니다. 많은 사람들이 ORM 사용해 보니 편리했습니다. 그래서 개발 언어별로 ORM을 편리하게 사용할 수 있도록 라이브러리(library)가 개발되었습니다.

우리는 Flask를 이용해서 웹 시스템을 개발하고 있습니다. Flask는 어떤 개발 언어를 사용하나요?

네, 맞습니다. Python입니다.

당연히 Python 언어를 지원하는 ORM 라이브러리가 있습니다. 대표적인 Python ORM 라이브러리는 다음과 같습니다.

- Django ORM: Django 프레임워크에 기본 탑재되어 제공하는 ORM 모델. 공식문서 ([click](#))
- SQLAlchemy: Python 코드에서 DB와 연결하고 CRUD 작업을 위해 사용하는 ORM 모델. 공식문서 ([click](#))

우리는 SQLAlchemy를 사용하겠습니다. SQLAlchemy는 에스큐엘 알케미 또는 씨퀄알케미와 같이 발음합니다. 하지만 정해진 한국어 발음 규칙은 없으니 여러분들 편할대로 발음해도 됩니다. 원문과 비슷한 발음은 씨퀄알케미입니다.

Flask에서 제공하는 [Flask-Migrate](#)라는 확장 프로그램을 설치하면 SQLAlchemy와 Alembic을 동시에 설치해 줍니다.

- Alembic 프로그램 (공식문서 ([click](#)) 단독으로 사용해도 Python 코딩에서 migration 할 수 있습니다.
- 하지만, Flask 공식문서는 ([click](#)) Flask에서 Alembic과 SQLAlchemy의 migration 연동이 최적화 되어 있는 [Flask-Migrate](#) 프로그램 설치하여 사용할 것을 권장하고 있습니다.

② DB Migrations 이란?

파이썬 코드로 작성한 ORM 클래스를 실제 DB 테이블로 옮기는 과정을 뜻합니다. Migration은 다음과 같은 순서로 진행됩니다.

- ORM (클래스) 모델: 개발자가 코딩해 줍니다.
- Migration: [Flask-Migrate](#)가 대신 처리해 줍니다.
- DB Table이 생성됩니다.

우리는 [Flask-Migrate](#)를 설치하여 활용하도록 하겠습니다.

3.4.1.1. Flask-Migrate 설치

가상환경에 진입하여 명령창에 `pip install Flask-Migrate`를 입력하고 엔터키를 치면 다음과 같은 설치 과정이 진행됩니다. 개인 컴퓨터마다 설치 메시지는 약간 다를 수 있습니다.

```
(가상환경이름) C:\여러분의 컴퓨터 경로> pip install Flask-Migrate
Collecting Flask-Migrate
  Downloading Flask_Migrate-3.1.0-py3-none-any.whl (20 kB)
Requirement already satisfied: Flask>=0.9 in c:/users/cju/anaconda3\envs\book_writing\lib\site-packages (from Flask-Migrate) (2.0.2)
Collecting alembic>=0.7
  Downloading alembic-1.7.6-py3-none-any.whl (210 kB)
|██████████| 210 kB 6.4 MB/s
Collecting Flask-SQLAlchemy>=1.0
  Downloading Flask_SQLAlchemy-2.5.1-py2.py3-none-any.whl (17 kB)

:
(중간 생략)
:

Successfully installed Flask-Migrate-3.1.0 Flask-SQLAlchemy-2.5.1 Mako-1.1.6 alembic-1.7.6
```

맨 마지막 줄에 `Successfully installed ...` 다음에 `Flask-Migrate-X.X.X, Flask-SQLAlchemy-X.X.X, alembic-X.X.X`와 같은 형태로 우리가 필요로하는 패키지들이 성공적으로 설치되었다는 메시지만 확인하면 됩니다.

3.4.1.2. config.py 설정

SQLAlchemy를 사용하기 위해서는 설정 파일 `config.py`에 일부 설정을 추가해야 합니다. 아래와 같이 `config.py` 파일에 코드를 입력합니다.

```
import os

# __file__ 이름을 갖는 파일의
# 디렉토리를 기본 디렉토리로 설정
# (프로젝트 루트 디렉토리와 동일)
BASE_DIR = os.path.dirname(__file__)

# SQLALCHEMY_DATABASE_URI -> DB 접속 주소
# 프로젝트 루트 디렉토리와 우리가 생성할 DB (hello_cju.db) 연결
# 'sqlite:///` -> 사용할 DB는 SQLite
SQLALCHEMY_DATABASE_URI = 'sqlite:///{}'.format(
    os.path.join(BASE_DIR, 'hello_cju.db')
)

# 만약 True 설정된 경우 ORM 객체의 변경사항을
# 지속적으로 추적하고 변동 이벤트에 대한 메시지를 출력함
# True 일 경우 추가 메모리를 사용하므로
# 불필요한 경우 False로 꺼놓는 것을 추천함
SQLALCHEMY_TRACK_MODIFICATIONS = False
```

Python 코드를 보다보면 `__file__`이라고 적힌 부분이 있는데, 이는 현재 수행중인 코드를 담고 있는 파일의 위치한 Path를 알려줍니다. `pytest.py`가 `C:/Users/test`에 위치해 있고, 아래와 같은 코드를 수행한다고 가정하겠습니다.

```
import os
print(os.path.dirname(__file__))
```

해당 코드를 실행하면 해당 파일이 위치한 Path를 아래와 같이 출력합니다.

```
C:/Users/test
```

'`sqlite://``'는 DB를 `SQLite`를 사용하겠다는 의미입니다. `SQLite`는 작은 크기의 프로젝트 개발에 자주 사용하는 DB입니다. 협업에서는 `SQLite`를 이용해 시스템을 빠르게 개발하고, 배포 및 상용화 단계에서 규모가 큰 DB로 변경하여 개발하는 방식을 취합니다.

하지만 웬만한 웹 시스템은 `SQLite`로 충분히 운영 가능합니다. 우리는 `SQLite`를 이용해서 개발을 계속해 나갈 것입니다.

`SQLALCHEMY_DATABASE_URI` 변수는 Flask에게 우리가 사용할 DB 위치를 알려줍니다. 우리가 현재 작업하고 있는 프로젝트 디렉토리 아래에 `hello_cju.db` 파일을 우리의 DB 파일로 등록하겠다는 의미입니다.

`SQLALCHEMY_TRACK_MODIFICATIONS` 변수는 ORM 모델 객체에 변동이 생기는 것을 추적하면서 이벤트를 알려줄 것인지를 결정하는 변수입니다. DB 이벤트를 추적할 일이 없다면 `False`를 지정하여 꺼놓는 것을 추천합니다.

VS code에 입력한 모습은 그림 [Fig. 3.6](#)과 같습니다.

```

File Edit Selection View Go Run Terminal Help
• config.py - Source_codes - Visual Studio Code

EXPLORER ... config.py > ...
SOURCE_CODES _init_.py U main_views.py U config.py U
> __pycache__ ...
hello_cju > __pycache__ ...
static # style.css U
templates < index.html U
views > __pycache__ ...
main_views.py U
< config.py U
15 import os
16
17 # __file__ 이름을 갖는 파일의
18 # 디렉토리를 기본 디렉토리로 설정
19 # (프로젝트 루트 디렉토리와 동일)
20 BASE_DIR = os.path.dirname(__file__)

21 # SQLALCHEMY_DATABASE_URI -> DB 접속 주소
22 # 프로젝트 루트 디렉토리와 우리가 생성할 DB (hello_cju.db) 연결
23 # 'sqlite:///` -> 사용할 DB는 SQLite
24 SQLALCHEMY_DATABASE_URI = 'sqlite:///{}'.format(
25     os.path.join(BASE_DIR, 'hello_cju.db')
26 )
27
28 # 만약 True 설정된 경우 ORM 객체의 변경사항을
29 # 지속적으로 추적하고 변동 이벤트에 대한 메시지를 출력함
30 # True일 경우 추가 메모리를 사용하므로
31 # 불필요한 경우 False로 깨놓는 것을 추천함
32 SQLALCHEMY_TRACK_MODIFICATIONS = False

```

Fig. 3.6 config.py에 DB 설정 내용을 입력

3.4.1.3. SQLAlchemy 적용

설정 파일을 완성하였으니 이번에는 SQLAlchemy를 우리 시스템에 적용해 보겠습니다. 우리 시스템에 사용할 App(앱)은 `hello_cju/_init_.py` 모듈에 코딩되어 있습니다.

`hello_cju/_init_.py`에서 필요한 클래스를(`Migrate, SQLAlchemy`) `import`하고 필요한 작업을 코딩해 줍니다.

```

from flask import Flask

# DB 처리에 필요한 클래스 import
from flask_migrate import Migrate
from flask_sqlalchemy import SQLAlchemy

# 설정 파일을 import
import config

# DB 및 migrate 객체 생성
db = SQLAlchemy()
migrate = Migrate()

def create_app(): # 함수 생성
    app = Flask(__name__)

    # app 객체에 설정 내용 적용
    app.config.from_object(config)

    # ORM 관련사항 코드
    db.init_app(app)
    migrate.init_app(app, db)

    from .views import main_views
    app.register_blueprint(main_views.bp)

    return app

```

DB 설정에서 주의해야 할 점

db 객체는 `create_app` 함수 밖에서 생성하고, `config.py` 모듈을 불러와서 초기화하는 것은 `create_app` 함수 내부에서 수행합니다. 왜냐하면 Blueprint 객체에서 db를 사용해야 할 경우 함수 내부에서 생성하면 객체가 지역변수 성격을 갖게 되어 db를 불러올 수 없기 때문입니다.

VS code에 코딩한 화면은 그림 Fig.3.7과 같습니다.

Flask 서버를 중지하고 아래 명령어를 순서대로 입력하여 다시 시작합니다.

```

File Edit Selection View Go Run Terminal Help
__init__.py - Source_codes - Visual Studio Code

EXPLORER ... _init_.py 2.U config.py U
SOURCE_CODES
> __pycache__ ...
hello_cju > __init__.py > ...
1   from flask import Flask
2
3   # DB 처리에 필요한 클래스 import
4   from flask_migrate import Migrate
5   from flask_sqlalchemy import SQLAlchemy
6
7   # 설정 파일을 import
8   import config
9
10  # DB 및 migrate 객체 생성
11  db = SQLAlchemy()
12  migrate = Migrate()
13
14  def create_app(): # 함수 생성
15
16      app = Flask(__name__)
17
18      # app 객체에 설정 내용 적용
19      app.config.from_object(config)
20
21      # ORM 관련사항 코드
22      db.init_app(app)
23      migrate.init_app(app, db)
24
25      from .views import main_views
26      app.register_blueprint(main_views.bp)
27
28
29
30

```

Fig. 3.7 hello_cju/__init__.py에 DB 설정 적용 및 DB 생성

```

set FLASK_APP=[여러분의 앱 이름]
# 우리 교재의 경우 set FLASK_APP=hello_cju

set FLASK_ENV=development

flask run

```

이전과 같이 잘 돌아간다면 정상적으로 SQLAlchemy를 __init__.py에 잘 등록한 것입니다.

혹시 다음과 같은 Warning이 뜰 경우가 있습니다. 이 경고는 무시하고 넘어가도록 합니다. `SQLALCHEMY_TRACK_MODIFICATIONS`를 `True`로 설정할 경우 리소스를 많이 잡아먹어서 미래 버전에서는 기본 사항으로 비활성화 시키겠다는 경고입니다. 우리는 `False`로 세팅 했으니 상관 없습니다.

```

FSDeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant
overhead and will be disabled by default in the future. Set it to True or False to suppress this warning.

```

3.4.1.4. DB 초기화

`CTL + C`를 눌러서 Flask 서버를 중지시키고 DB를 초기화 하도록 하겠습니다.

DB 초기화 하는 명령어는 `flask db init` 입니다. 이 명령어를 입력하고 엔터키를 치면 다음과 같은 메시지가 출력되면서 SQLite DB를 초기화 합니다.

```

(가상환경 이름) C:\여러분의 프로젝트 디렉토리 경로> flask db init
Creating directory C:\여러분의 프로젝트 디렉토리 경로\migrations ... done
Creating directory C:\여러분의 프로젝트 디렉토리 경로\migrations\versions ... done
Generating C:\여러분의 프로젝트 디렉토리 경로\migrations\alembic.ini ... done
Generating C:\여러분의 프로젝트 디렉토리 경로\migrations\env.py ... done
Generating C:\여러분의 프로젝트 디렉토리 경로\migrations\README ... done
Generating C:\여러분의 프로젝트 디렉토리 경로\migrations\script.py.mako ... done
Please edit configuration/connection/logging settings in 'C:\여러분의 프로젝트 디렉토리 경로\migrations\alembic.ini' before proceeding.

```

출력되는 메시지를 보면 현재 작업(프로젝트) 디렉토리 아래에 `migrations`라는 디렉토리를 만들고 그 안에 여러가지 필요한 파일과 디렉토리를 생성한다는 것입니다.

VS code의 File 탐색 영역을 확인하면 실제로 `migration` 폴더 내부에 여러가지 파일들이 생성된 것을 확인할 수 있습니다. 그림 Fig. 3.8에서 화살표가 가리키는 빨간색 사각형 영역을 참고하세요.

Flask로 웹 시스템을 개발하면서 DB 관련 명령어는 2가지만 기억하면 됩니다.

```

File Edit Selection View Go Run Terminal Help _init_.py - Source_codes - Visual Studio Code

EXPLORER ... _init_.py 2. U config.py U
SOURCE_CODES hello_cju > _init_.py > create_app
> __pycache__ . ●
hello_cju . ●
< 154 > __pycache__ . ●
< _init_.cpython-38.pyc U
static . ●
< style.css U
templates > index.html U
views . ●
< __pycache__ . ●
main_views.py U
< _init_.py 2. U
forms.py U
models.py U
migrations . ●
< versions . ●
< alembic.ini U
< env.py U
< README U
< script.py.mako U
config.py U

hello_cju > _init_.py > create_app
1   from flask import Flask
2
3   # DB 처리에 필요한 클래스 import
4   from flask_migrate import Migrate
5   from flask_sqlalchemy import SQLAlchemy
6
7   # 설정 파일을 import
8   import config
9
10  # DB 및 migrate 객체 생성
11  db = SQLAlchemy()
12  migrate = Migrate()
13
14  def create_app(): # 함수 생성
15
16      # 기존에 hello_cju.py에 있던 영역
17      app = Flask(__name__)
18
19      # app 객체에 설정 내용 적용
20      app.config.from_object(config)
21
22      # ORM 관련사항 코드
23      db.init_app(app)
24      migrate.init_app(app, db)
25
26      # views/main_view.py 모듈을 임포트합니다.
27      from .views import main_views
28
29      # Flask 객체 app에 main_view 모듈에 있는
30      # Blueprint 객체 bp를 등록합니다.
31      app.register_blueprint(main_views.bp)
32
33      # bp 객체를 app에 등록했으므로
34      # 기존에 있던 코드는 삭제합니다.
35      # @app.route('/')
36      # def hello_world():
37      #     return 'Hello world! Welcome to CJU.'
38      # create_app 함수가 생성한 app을 리턴
39
40      return app
41

```

Fig. 3.8 flask db init 실행 결과로 생성된 디렉토리와 파일

- **flask db migrate:** ORM 모델을 새로 생성하거나 변경
- **flask db upgrade:** ORM 모델의 변경 내용을 실제 DB에 적용

이제 Flask 모델을 설치하고 DB를 초기화 하는 것까지 완성했습니다.

다음에 다룰 내용은 우리가 실제로 사용할 DB를 설계하고 만들어볼 차례입니다.

준비 되셨나요?

Next 아이콘을 눌러서 이동해 주세요.

3.4.2. 모델 설계 및 만들기

모델을 정의하고 등록하였다면 이제부터는 실제로 우리가 사용할 DB를 설계해야 합니다.

DB를 설계하는 일은 어려울 수도 있고 쉬울 수도 있습니다. 우리가 개발할 웹 시스템이 다양한 종류의 데이터를 다루어야 하고 데이터가 서로 연결되어 있다면 매우 복잡한 DB 구조를 갖게 됩니다. 반면에 간단한 웹 시스템은 DB 구조도 간단하게 설계할 수 있습니다. 복잡한 DB를 설계하는 것은 고수준을 요구하는 작업이기도 합니다. 많은 경험과 노하우가 필요합니다.

우리가 처음부터 복잡한 DB를 설계할 수는 없겠죠? 그래서 가장 기초가 되는 기능부터 실습해 보도록 하겠습니다.

DB 구현에서 가장 기초가 되는 것은 게시판입니다. 게시판 구현을 통해 CRUD 기능을 모두 학습할 수 있고, 조금만 응용하면 다른 기능도 충분히 구현 가능합니다.

우리는 Q&A 게시판을 만들어 보겠습니다. Q&A 게시판은 다음과 같은 특성이 있다고 가정하겠습니다.

❶ Q&A 게시판 특성

- Q&A 게시판은 질문과 답변으로 구성됩니다.
- 각각의 질문은 다음과 같은 특성을 가집니다.
 - 각각의 질문은 고유 아이디(id)를 가집니다.
 - 각각의 질문은 제목(title), 내용(contents), 생성일시(create_date)를 갖습니다.
- 각각의 답변은 다음과 같은 특성을 가집니다.
 - 각각의 답변은 고유 아이디(id)를 가집니다.
 - 각각의 답변은 어떤 질문에 대한 답변인지를 알아내기 위한 질문 고유번호(question_id)를 가집니다.
 - 각각의 답변은 답변 내용(contents), 생성일시(create_date)를 갖습니다.

위와 같은 Q&A 게시판 특성에 대한 분석이 끝나면 이를 구성하기 위한 모델을 코딩합니다.

먼저 아래와 같이 코딩합니다.

```
# file name: hello_cju/models.py

# hello_cju/__init__.py에서 생성한
# db 객체를 임포트
from hello_cju import db

# db 객체의 Model 클래스를 상속받아
# 질문(Question) 클래스를 정의함
class Question(db.Model):
    # 고유 아이디는 정수형, 기본키로 등록
    id = db.Column(db.Integer, primary_key=True)

    # 질문 제목은 최대 200글자, 공백허용 안됨
    title = db.Column(db.String(200), nullable=False)

    # 질문 내용은 문자열 글자수 제한 없음, 공백허용 안됨
    contents = db.Column(db.Text(), nullable=False)

    # 생성일시는 날짜 및 시간, 공백허용 안됨
    create_date = db.Column(db.DateTime(), nullable=False)
```

❷ Note

DB에서 유일한 값을 가지는 속성을 기본키(primary key)라고 부릅니다. Primary key로 정수형이 지정되면 데이터가 추가될 때마다 자동으로 값이 1 만큼 증가합니다.

질문(Question) 클래스를 만들었으니 이번에는 답변(Answer) 클래스도 만듭니다. 위에서 코딩한 `Question` 클래스에 이어서 아래와 같이 `Answer` 클래스도 코딩해 줍니다.

```

# Question 클래스 코드 이후에 Answer 클래스 추가

# db 객체의 Model 클래스를 상속받아
# 답변(Answer) 클래스를 정의함
class Answer(db.Model):

    # 고유 아이디는 정수형, 기본키로 등록
    id = db.Column(db.Integer, primary_key=True)

    # 질문 고유번호는 정수형,
    # 외래키(ForeignKey)를 Question 클래스의 고유번호(id)로 지정
    # 질문이 삭제되면 답변도 같이 삭제 (CASCADE)
    question_id = db.Column(
        db.Integer,
        db.ForeignKey('question.id', ondelete='CASCADE'),
    )

    # 답변에서 Question 클래스의 내용을 참조할 수 있도록 설정
    # Question 클래스에서 답변 내용을 참조하도록 설정
    # 하나의 질문에 여러개 답변이 달린 경우,
    # 질문에 달린 여러 답변을 확인
    question = db.relationship(
        'Question',
        backref=db.backref('answer_set'),
    )

    # 질문 내용은 문자열 글자수 제한 없음, 공백허용 안됨
    contents = db.Column(db.Text(), nullable=False)

    # 생성일시는 날짜 및 시간, 공백허용 안됨
    create_date = db.Column(db.DateTime(), nullable=False)

```

Answer 클래스에서 `db.relationship`은 다른 모델의 값을 참조하기 위한 것입니다. 만약 참조할 모델이 Question 클래스라면 `db.relationship`의 첫번째 인자로 `Question`을 전달합니다. 해당 답변에 대한 질문의 제목을 알고 싶다면 `answer.question.title`과 같이 참조할 수 있습니다.

하나의 질문에 여러 개의 답변이 달릴 수 있다. 어떤 질문 aaa라는 객체가 있다면 `aaa.answer_set`과 같은 형태로 해당 질문에 대한 답변들을 얻어낼 수 있습니다. 이런 기능 활용을 위해서 `db.relationship`의 두번째 인자값으로 `backref=db.backref('answer_set')`을 전달합니다.

VS Code에서 작성한 내용은 그림 Fig. 3.9와 같습니다.

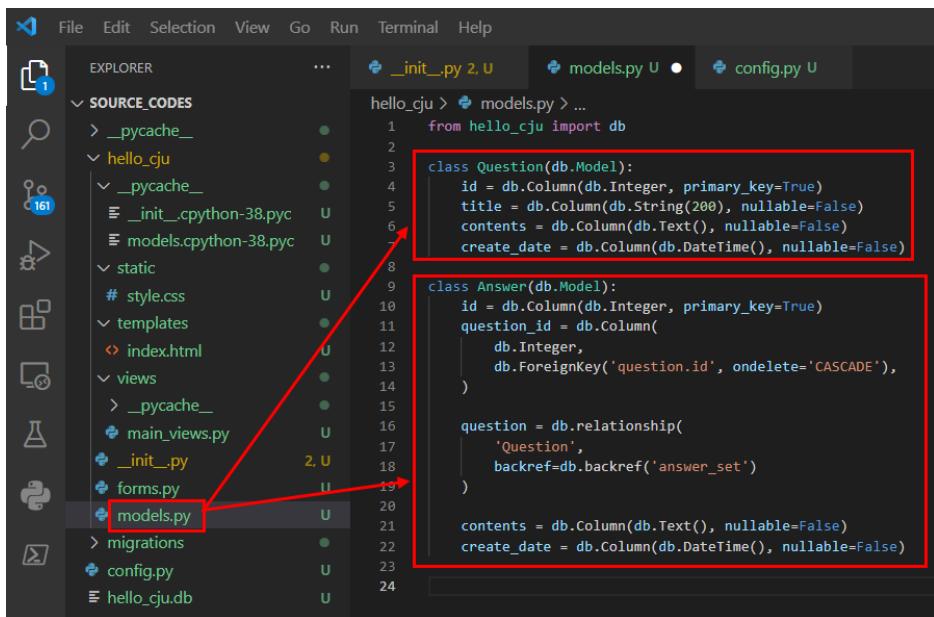


Fig. 3.9 VS code에서 Question, Answer 클래스를 코딩한 모습

SQLAlchemy에서 지원하는 Column과 Data 타입

SQLAlchemy에서 지원하는 자료와 속성들은 매우 다양합니다. 개발자가 필요에 의해 참고하고 가져다 쓰면 됩니다.

- Flask 공식 문서([click](#))를 참고하세요.

Note

ForeignKey 옵션을 `CASCADE`로 설정할 경우 참조하는 객체의 데이터가 삭제되면 관련된 모든 데이터가 완전 삭제되는 것이 아니라 primary_key 값만 빈 값으로 변경됩니다. 사용자들은 삭제된 것처럼 보이지만, 실제로 DB에 데이터는 남아있게 됩니다. 완전삭제를 위해서는 `db.backref` 옵션을 아래와 같이 변경해야 합니다.

```
question = db.relationship(
    'Question',
    backref=db.backref(
        'answer_set',
        cascade='all, delete-orphan'
    ),
)
```

3.4.3. DB 테이블 만들기

우리는 ORM DB를 설계했습니다. 비록 간단하기는 하지만 우리가 코딩한 `Question`과 `Answer` 클래스가 바로 ORM DB 모델입니다.

이제 클래스로 코딩된 모델을 실제 SQLite DB로 바꿔주는 작업을 해야 합니다. 이 작업을 `migrate`한다고 표현 합니다. `migrate`이 이사간다는 뜻이니까 클래스 형태가 DB 구조로 이사간다고 생각하면 이해가 좀 더 쉬울 것 같습니다.

`migrate`을 수행하기 위해서는 `hello_cju/__init__.py` 내용 중에서 `migrate` 객체가 `models.py` 내용을 참조할 수 있도록 코드 한줄을 넣어줘야 합니다. 아래와 같은 코드를 추가합니다.

```
def create_app():
    # ... 초기 코드 생략

    # ORM 관련사항 코딩
    db.init_app(app)
    migrate.init_app(app, db)

    # 추가할 코드
    from . import models

    # 이후 코드 생략...
```

그림 [Fig. 3.10](#) 으로 보면 다음과 같은 위치입니다.

이제 진짜로 `migrate`을 통해 DB 테이블을 생성할 순서입니다. `flask db migrate` 명령어를 실행하면 다음과 같은 메시지가 출력되면서 `migrate`이 진행됩니다.

```
(가상환경이름) C:\여러분의 컴퓨터 경로>flask db migrate
INFO [alembic.runtime.migration] Context impl SQLiteImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
INFO [alembic.autogenerate.compare] Detected added table 'question'
INFO [alembic.autogenerate.compare] Detected added table 'answer'
Generating (가상환경이름) C:\여러분의 컴퓨터 경로\migrations\versions\3277f1bc750b_.py ... done
```

위 실행 결과에서 마지막 줄을 보면 작업 프로젝트 디렉토리에 `\migrations\versions\` 아래에 `3277f1bc750b_.py` 파일이 생성했다는 메시지를 볼 수 있습니다.

실제로 확인해 보면 그림 [Fig. 3.11](#) 같이 `3277f1bc750b_.py` 파일이 생성된 것을 확인할 수 있습니다.

여기서 `3277f1bc750b_.py`와 같이 생성된 파일을 `리비전(revision)` 파일이라고 부릅니다. DB의 변경사항들을 처리하는 파일입니다. 리비전 파일 이름은 랜덤하게 생성되기 때문에 여러분과 제 파일명은 다를 것입니다. 대부분 숫자/문자 조합 + `.py`와 같은 형태의 이름을 갖게 됩니다.

VS code에서 `3277f1bc750b_.py` 파일을 열어보면 Flask-Migrate 프로그램이 우리가 만든 ORM 모델 (`models.py`)를 참고하여 자동으로 DB를 생성한 결과를 확인할 수 있습니다.

그림 [Fig. 3.11](#)에서 작업 프로젝트의 루트 디렉토리에 프로젝트 이름과 동일한 `hello_cju.db` 파일이 생성된 것을 볼 수 있습니다. `hello_cju.db` 파일이 바로 SQLite DB 파일입니다.

현재 상태의 DB 변경사항을 최종적으로 실제 DB에 적용하기 위해서는 `flask db upgrade` 명령을 실행시킨다. 아래와 같은 메시지가 출력되면서 DB를 최종 생성한다.

```

File Edit Selection View Go Run Terminal Help
EXPLORER ... _init_.py 2, U models.py config.py
SOURCE CODES
hello_cju
  __pycache__
  hello_cju
    __pycache__
      __init__.cpython-38.pyc U
      models.cpython-38.pyc U
    static
      # style.css U
    templates
      index.html U
    views
      __pycache__
        main_views.py U
      __init__.py 2, U
      forms.py U
      models.py U
    migrations
    config.py
    hello_cju.db U
  _init_.py 2, U
  forms.py U
  models.py U
  hello_cju.db U
  migrations
  config.py
  hello_cju.db U

def create_app(): # 함수 생성
    # 기존에 hello_cju.py에 있던 영역
    app = Flask(__name__)

    # app 객체에 설정 내용 적용
    app.config.from_object(config)

    # ORM 관련사항 코드
    db.init_app(app)
    migrate.init_app(app, db)

    # 함수 밖에서 생성한 migrate 객체가
    # models.py를 참조하도록 ORM 모델 파일 import
    from . import models

    # views/main_view.py 모듈을 임포트합니다.
    from .views import main_views

    # Flask 객체 app에 main_view 모듈에 있는
    # Blueprint 객체 bp를 등록합니다.
    app.register_blueprint(main_views.bp)

    # bp 객체를 app에 등록했으므로
    # 기존에 있던 코드는 삭제합니다.
    # @app.route('/')
    # def hello_world():
    #     return 'Hello world! Welcome to CJU.'
    # create_app 함수가 생성한 app을 리턴

    return app

```

Fig. 3.10 `__init__.py`의 `models` 임포트

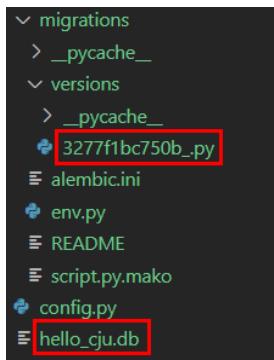


Fig. 3.11 ORM model을 migration 수행한 결과

```
(가상환경이름) C:\여러분의 컴퓨터 경로> flask db upgrade
INFO [alembic.runtime.migration] Context impl SQLiteImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
INFO [alembic.runtime.migration] Running upgrade  -> 3277f1bc750b, empty message
```

여러분! 정말 수고하셨습니다. 여기까지 왔다면 Flask 웹 시스템 구축을 위한 큰 산을 넘어선 것입니다.

앞으로 데이터베이스가 실제로 만들어 졌는지 시각화 도구를 이용해 확인하고, 실제로 만들어진 DB에 데이터를 적용해보면 됩니다.

준비 되셨나요?

Next 아이콘을 눌러 이동해 주세요.

3.4.4. 시각화 도구로 모델 확인

코딩을 통해 완성한 .db 파일은 VS code 또는 다른 편집기로 내용을 볼 수 없다. 이를 확인하기 위해서는 DBMS 별로 제공되는 시각화 도구를 사용하면 편리하다.

우리가 생성한 DB는 SQLite 이므로 [DB Browser for SQLite](#)를 설치하여 손쉽게 확인할 수 있다. [DB Browser for SQLite](#)는 무료이므로 다운로드 페이지 ([click](#))에서 자신의 운영체제와 환경에 맞는 설치 파일을 다운로드 받아서 설치하면 된다.

저는 아래 그림 [Fig. 3.12](#) 같은 버전을 다운로드 했습니다.

Windows

Our latest release (3.12.2) for Windows:

- DB Browser for SQLite - Standard installer for 32-bit Windows
- DB Browser for SQLite - .zip (no installer) for 32-bit Windows
- **DB Browser for SQLite - Standard installer for 64-bit Windows**
- DB Browser for SQLite - .zip (no installer) for 64-bit Windows

Fig. 3.12 DB Browser for SQLite 다운로드 페이지

설치 과정에서 옵션 선택은 그림 [Fig. 3.13](#) 같이 하고 나머지는 [Next](#)를 클릭하여 프로그램을 설치합니다.

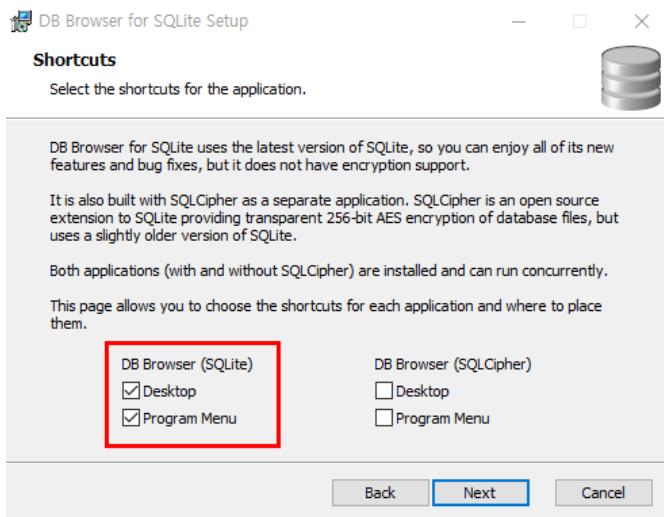


Fig. 3.13 DB Browser for SQLite 설치 옵션 선택

DB Browser for SQLite 설치가 끝나면 프로그램을 실행시키고 그림 [Fig. 3.14](#) 같이 우리가 생성한 SQLite DB 파일을 엽니다. 제 기준으로는 `hello_cju.db` 파일입니다.

.db 파일을 열면 그림 [Fig. 3.15](#) 같이 SQLite DB가 정상적으로 생성되어 있는 것을 확인할 수 있습니다.

이것으로 우리가 `flask db migrate`, `flask db upgrade` 명령을 통해 `migration`을 수행하여 생성한 SQLite DB 파일을 확인하였습니다.

원하는 DB를 생성하였습니다. 그 다음 할 일은 무엇일까요?

네, 맞습니다.

우리가 만든 DB에 실제로 CRUD 되는지 확인해 봐야 겠죠?

준비 되었으면 [Next](#) 아이콘을 눌러 확인하러 가겠습니다.

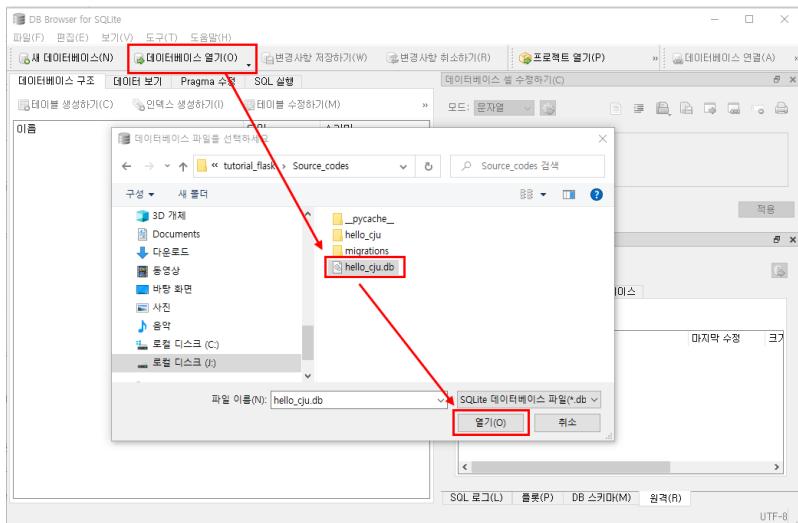


Fig. 3.14 DB Browser for SQLite로 .db 파일 열기

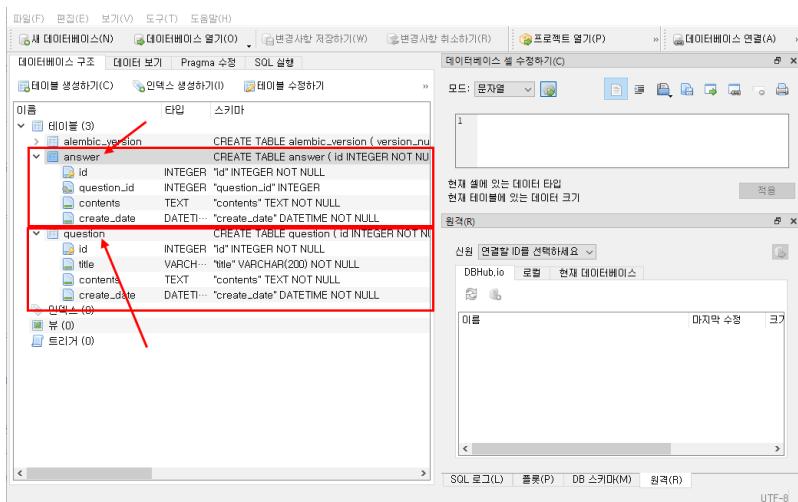


Fig. 3.15 SQLite 생성 확인하기

3.4.5. 모델에 CRUD 해보기

드디어 모델을 생성하고 시각화 도구인 **DB Browser**를 이용해 SQLite DB도 잘 있는지 확인했습니다.

DB의 핵심 기능은 CRUD라고 말했습니다.

이번에는 CRUD 각각의 기능을 직접 실습해 보도록 하겠습니다.

Flask는 `flask shell` 명령을 통해 CLI 환경에서 CRUD를 수행할 수 있도록 해줍니다. 나중에는 CRUD 기능 모두 VS code에서 직접 코딩해 볼 것입니다. 그 전에 CLI 환경에서 간단히 CRUD 기능을 경험하여 flask ORM 작동 원리를 이해하도록 하겠습니다.

3.4.5.1. `flask shell` 실행

명령창에 `flask shell`을 실행하면 됩니다. 실행 화면은 다음과 같습니다.

```
(가상환경 이름) C:\여러분의 컴퓨터 경로> flask shell
Python 3.8.12 (default, Oct 12 2021, 03:01:40) [MSC v.1916 64 bit (AMD64)] on win32
App: hello_cju [development]
Instance: C:\여러분의 컴퓨터 경로\instance
>>>
```

3.4.5.2. Create 실습 - add 데이터 저장

데이터를 저장하기 위해서 필요한 클래스를 import하고 객체를 생성합니다. 먼저 Question 객체를 생성하고 DB를 불러와서 `add()` 메서드를 이용하여 저장합니다.

```
>>> from hello_cju.models import Question, Answer
>>> from datetime import datetime
>>> q = Question(title='청주대 위치 문의', contents='충북 청주시 청원구에 있어요', create_date=datetime.now())
```

위 코드에서는 질문 객체 1개를 생성했습니다. `datetime.now()`는 현재 시간을 알려주는 함수입니다.

Question 객체 `q`의 `id`는 별도로 입력하지 않았지만 `primary key`로 설정되어 있기 때문에 자동으로 부여됩니다. 아래 코드를 통해 확인할 수 있습니다.

```
>>> q.id
1
```

Question 객체를 만들었으므로 `db`를 불러와서 저장합니다.

```
>>> from hello_cju import db
>>> db.session.add(q)
>>> db.session.commit()
```

위 코드에서 `db.session`은 DB와 연결한다는 의미입니다.

`db.session.add(q)`는 연결된 DB에 객체를 추가한다는 의미입니다.

`session`을 통해서 DB에 CRUD를 수행합니다.

모든 작업이 완료되면 `commit()`을 이용하여 DB에 데이터를 최종 반영합니다. 참고로 `commit()`은 취소할 수 없습니다. 수행한 작업을 취소하고자 하는 경우에는 `db.session.rollback()`을 이용하여 되돌리기 후 다시 작업을 수행해야 합니다.

내친 김에 데이터를 하나 더 추가하고 `primary key`가 자동으로 증가하는지 확인해 보겠습니다.

```
>>> q = Question(title='SW융합학부 세부전공', contents='SW융합학부에는 인공
지능소프트웨어전공, 빅데이터통계학전공, 디지털보안전공이 있습니다.', create_date=datetime.now())
>>> db.session.add(q)
>>> db.session.commit()
>>> q.id
2
```

`q.id`를 확인해 보니 2가 저장되어 있습니다. 우리가 살펴봤던 것과 동일하게 `primary key`가 1 만큼 자동 증가하였습니다.

3.4.5.3. Read 실습 - all, filter, get, like

`add()`를 이용해서 DB에 데이터를 업로드 해봤습니다. SQLAlchemy를 이용하여 데이터를 읽어들이는 방법은 `all, filter, get` 메서드를 이용하는 것이 가능합니다.

`all`은 DB에 저장된 모든 데이터를 읽어 옵니다.

```
>>> Question.query.all()
[<Question 1>, <Question 2>]
```

`Question.query.all()` 메서드를 이용하면 `Question` 전체 객체가 리스트(list)로 반환됩니다. 반환된 객체 `<Question 1>, <Question 2>`에서 숫자 1과 2는 primary key `id`의 값입니다.

`filter` 속성을 이용하면 조건에 맞는 데이터만 뽑아낼 수 있습니다.

```
>>> Question.query.filter(Question.id==1).all()
[<Question 1>]
```

위 코드는 `Question` 객체 중 `id`가 1인 모든 객체를 뽑아내는 명령입니다. Primary key는 고유한 값이므로 객체 1개만 리턴 되었습니다. 동일한 날짜, 동일한 제목(title)을 가진 데이터가 여러 개라면 다수의 데이터가 리턴됩니다.

`get` 메서드는 조건에 맞는 데이터가 여러 개일 경우 1개만 리턴합니다. Primary key는 고유한 값이므로 `get` 메서드를 이용해 뽑아낼 수 있습니다. 다음 코드는 `Question` 객체 중 `id`가 1인 객체만 뽑아오는 코드입니다.

```
>>> Question.query.get(1)
<Question 1>
```

`like`는 필드 속성에 특정 문자열이 포함된 데이터를 뽑아낼 경우 사용합니다. ```Question.title.like('%청주대%')```는 `Question` 객체들 중에서 `title`에 '청주대'라는 단어가 포함된 객체가 `True`가 됩니다. 종종 `filter`와 혼합하여 사용합니다.

```
>>> Question.query.filter(Question.title.like('%청주대%')).all()
[<Question 1>]
```

위 코드는 `Question` 객체 중에서 `title`에 '청주대'라는 문자열이 포함된 모든 데이터를 뽑아 옵니다.

`like`를 활용하여 특정 문자열을 조건으로 검색할 경우 다음과 같은 옵션이 가능합니다.

- 찾을문자열%: '찾을문자열'로 시작하는 모든 데이터
- %찾을문자열%: '찾을문자열'이 포함된 모든 데이터
- 찾을문자열%: '찾을문자열'로 끝나는 모든 데이터

3.4.5.4. `Update` 실습 - = 대입 연산자

데이터 수정은 수정할 내용을 담고 있는 객체를 불러와서 대입 연산자 `=`를 사용하여 업데이트 합니다. `Question` 객체 중 `id`가 2인 데이터의 `contents`를 'SW 학부는 3개 전공이 있습니다.'로 변경하고 업데이트 시간도 변경해 보겠습니다.

```
>>> q = Question.query.get(2)
>>> q
<Question 2>
>>> q.contents
'SW융합학부에는 인공지능소프트웨어전공, 빅데이터통계학전공, 디지털보안전공이
있습니다.'
>>> q.contents = 'SW학부는 3개 전공이 있습니다.'
>>> q.contents
'SW학부는 3개 전공이 있습니다.'
>>>
>>> q.create_date
datetime.datetime(2022, 2, 3, 15, 31, 10, 216682)
>>> q.create_date = datetime.now()
>>> q.create_date
datetime.datetime(2022, 2, 3, 16, 13, 40, 243688)
```

3.4.5.5. `Delete` 실습 - `delete`

데이터를 삭제할 경우에는 `delete` 메서드를 이용합니다. 먼저 삭제할 객체를 찾아내고, 삭제할 객체를 `delete()` 메서드의 인자로 전달합니다.

`Question` 객체 중 `id`가 1인 데이터를 찾아서 삭제하는 경우 다음과 같습니다.

```
>>> q = Question.query.get(1)
>>> q
<Question 1>
>>> db.session.delete(q)
>>> db.session.delete(q)
>>> Question.query.all()
[<Question 2>]
```

`Question` 객체 중 `id`가 1인 데이터는 삭제된 것을 확인할 수 있습니다.

3.4.5.6. `Answer Create - add`

이번에는 특정 질문(`Question`)에 대한 답변(`Answer`)을 등록해 보겠습니다. 답변을 등록하려면 질문이 어떤 것인지 먼저 찾아야 합니다.

```

>>> from hello_cju.models import Question, Answer
>>> from datetime import datetime
>>> from hello_cju import db
>>> q = Question.query.get(2)
>>> q
<Question 2>
>>> a = Answer(question=q, contents='자세한 내용은 이메일로 보내주세요', create_date=datetime.now())
>>> a
<Answer (transient 2364111491856)>
>>> db.session.add(a)
>>> db.session.commit()
>>> a.id
1
>>> a.contents
'자세한 내용은 이메일로 보내주세요'
>>> a.create_date
datetime.datetime(2022, 2, 3, 16, 42, 18, 864192)

```

`Answer` 클래스는 다음과 같이 설정되어 있습니다.

```

class Answer(db.Model):
    id = db.Column(db.Integer, primary_key=True)

    question_id = db.Column(
        db.Integer,
        db.ForeignKey('question.id', ondelete='CASCADE'),
    )

    question = db.relationship(
        'Question',
        backref=db.backref('answer_set')
    )

    contents = db.Column(db.Text(), nullable=False)
    create_date = db.Column(db.DateTime(), nullable=False)

```

`Answer` 객체 `a`를 생성할 때 `question` 인자를 `Question` 객체인 `q`로 전달하였습니다. 위와 같은 코드 대신에 `Question` 객체를 대입해도 동일하게 작동하게 됩니다.

`Answer` 객체를 생성할 때 `id`는 primary key 이므로 자동 부여됩니다.

`question_id`는 `question` 인자를 `Question` 객체인 `q`로 전달하면 자동으로 배정됩니다.

3.4.5.7. Answer에 연결된 Question 검색

앞에서 살펴 보았던 `Answer` 클래스에서 `question` 인자를 활용하면 해당 답변(`Answer`)에 해당하는 질문 객체를 뽑아낼 수 있습니다.

```

>>> a.question
<Question 2>

```

3.4.5.8. Question에 연결된 Answer 검색

반대의 경우는 `Answer` 모델의 `question` 속성에 역참조 설정 `backref=db.backref('answer_set')`을 활용하면 가능합니다.

다음과 같은 코딩을 통해 질문에 달린 모든 답변을 뽑아올 수 있습니다.

```

>>> q.answer_set
[<Answer 1>]

```

SQLAlchemy 질의

SQLAlchemy에서 지원하는 다양한 CRUD 관련 옵션은 아래 링크를 참고하세요. SQLAlchemy CRUD

3.5. 질문 게시판 만들기

우리는 다음과 같이 기본기 4가지를 공부하였습니다.

① Flask 웹 시스템 구축을 위한 10가지 기본기

1. [프로젝트 기본 구조](#) \(\backslash\backslash to\backslash Clear!
2. [Application Factory 패턴](#) \(\backslash\backslash to\backslash Clear!
3. [Blueprint 클래스 활용](#) \(\backslash\backslash to\backslash Clear!
4. [ORM 모델 완벽 이해](#) \(\backslash\backslash to\backslash Clear!
5. [질문 게시판 만들기](#) \(\backslash\backslash to\backslash\) 지금 도전!
6. [게시판 댓글 구현](#)
7. [질문 & 댓글 등록 - Flask Form 활용](#)
8. [예쁘게 - CSS 적용](#)
9. [더 예쁘게 - Bootstrap 활용](#)
10. [HTML 구조와 Template 상속](#)

기본기 4. [ORM 모델 완벽 이해](#) 경우는 DB와 ORM 관련된 사항이 많았습니다. 우리는 5단계를 거치면서 ORM에 대한 세부 기본기를 공부했습니다.

② ORM 완벽 이해 5단계

1. [Flask ORM 설치](#) \(\backslash\backslash to\backslash Clear!
2. [모델 설계 및 만들기](#) \(\backslash\backslash to\backslash Clear!
3. [DB 테이블 만들기](#) \(\backslash\backslash to\backslash Clear!
4. [시각화 도구로 모델 확인](#) \(\backslash\backslash to\backslash Clear!
5. [모델에 CRUD 해보기](#) \(\backslash\backslash to\backslash Clear!

이제는 5번째 기본기 [질문 게시판 만들기](#) 통해 지금까지 배운 것을 총동원해서 게시판을 구현해 볼 것입니다. 질문 게시판 만들기는 3단계를 거치면서 만들어 갈 것입니다. 배울 순서는 다음과 같습니다.

③ 질문 게시판 구현 3단계

1. [Question List 조회](#)
2. [Question 상세 조회](#)
3. [존재하지 않는 페이지 처리 - 404 오류](#)
4. [기능 분리 - Blueprint 활용](#)

3.5.1. Question List 조회

`views/main_view.py`에서 작성한 코드를 다시 한번 확인해 봅니다.

The screenshot shows the Visual Studio Code interface with the main_views.py file open in the editor. The code defines a Blueprint named 'main' with a root URL prefix. It contains a single route '/' that returns the string 'Hello world! Welcome to CJU.' A red box highlights the line `@bp.route('/')`. To the right, a browser window titled '127.0.0.1:5000' shows the same message: 'Hello world! Welcome to CJU.'

```
File Edit Selection View Go Run Terminal Help
EXPLORER SOURCE CODES ...
SOURCE CODES
> _pycache_ ...
> hello_cju ...
> _pycache_ ...
> static ...
> templates ...
> index.html ...
views ...
> _pycache_ ...
main_views.py ...
main_views.py ...
main_cju > views > main_views.py ...
1 # 파일 경: views/main_views.py
2
3 # Blueprint 클래스를 임포트 합니다.
4 from flask import Blueprint
5
6 # Blueprint 객체 bp를 생성합니다.
7 bp = Blueprint('main', __name__, url_prefix='/')
8
9 # Blueprint 객체 bp를 이용하여
10 # 함수와 URL을 매핑합니다.
11 @bp.route('/')
12 def hello_world():
13     return 'Hello world! Welcome to CJU.'
14
15 # 전공 소개 페이지 추가
16 @bp.route('/major')
17 def intro_major():
18     return '우리 전공은 인공지능소프트웨어입니다.'
19
```

Fig. 3.16 루트 주소(127.0.0.1:5000)로 접속한 경우 화면

그림 Fig.3.16에서 확인할 수 있는 것처럼 `main_view.py` 내부에서 루트 / 주소로(127.0.0.1:5000 또는 localhost:5000) 접속하면 `hello_world` 함수가 실행되고 사용자는 브라우저 화면에서 Flask가 보내준 정보를 확인하게 됩니다. 우리 예제에서는 'Hello world! Welcome to CJU.'라는 문자열이 보입니다.

우리는 ORM을 활용해서 DB를 다룰 수 있게 되었습니다. DB에 질문도 등록하고 답변도 등록할 수 있습니다.

이제는 단순히 문자열만 리턴하던 `hello_world` 함수를 변경하여 질문 목록을 리턴하는 함수로 변경하겠습니다. 이렇게 하면 루트 / 경로로 접속하면 ORM DB에 저장되어 있는 질문 데이터를 보여줄 수 있습니다.

질문 목록을 출력하기 위해 할 일은 2가지입니다.

➊ To-Do List: 질문 목록 출력

1. views 파일 (`/views/main_views.py`) 수정
2. 질문 목록을 보여줄 템플릿(`.html`) 파일 작성

3.5.1.1. view 파일 수정

views 파일 (`/views/main_views.py`) 수정 작업을 하겠습니다.

- 필요한 모듈 import
 - `render_template`: 템플릿 파일 (`.html`)을 모니터 화면에 그려 줍니다.
 - `Question` 클래스: ORM 모델로 만들어 놓은 `Question` 클래스

아래와 같이 `import` 합니다.

```
from flask import render_template
from hello_cju.models import Question
```

3.5.1.1.1. render_template 깊은 이해

좀 더 깊은 이해를 위해 아래 내용을 참고하세요.

❶ render_template 함수의 깊은 이해

Flask 공식 문서에서 제공하는 `render_template` 설명 ([click](#))

```
```{code} bash
flask.render_template(template_name_or_list, **context)
 Renders a template from the template folder
 with the given context.

Parameters:
 template_name_or_list (Union[str, List[str]])
 the name of the template to be rendered,
 or an iterable with template names
 the first one existing will be rendered

 context (Any)
 the variables that should be available
 in the context of the template.

Return type: str
````
```

`render_template` 기능: 전달된 `context` 정보를 사용하여 `template` 폴더에 있는 템플릿을 그려(render) 줍니다. 템플릿을 그린 다음에 사용자의 브라우저로 보내는 역할을 합니다.

`render_template` 함수는 2개의 인자가 전달됩니다.

- 첫 번째 인자: `template_name_or_list`
 - 반드시 값이 전달되어야 합니다.
 - 전달되는 인자는 문자열(str) 또는 문자열(str)을 담고 있는 이터러블(iterable)이어야 합니다 (list, tuple 등).
 - 인자값은 화면에 그려야(render) 할 템플릿 이름입니다. 리스트로 전달될 경우 첫 번째 템플릿 이름을 적용합니다.
- 두 번째 인자: `**context`
 - 템플릿에서 사용할 데이터를 담고 있는 변수입니다.
 - 어떤(any) 형태라도 가능합니다.
 - 일반적으로 반복 가능한 iterable 객체를 사용하여 전달합니다. dictionary, list, tuple 자료형 모두 가능합니다.

`hello_world` 함수를 수정하여 기존에 출력하던 '`Hello world! Welcome to CJU.`' 문자열 대신에 질문 목록을 출력하도록 수정하겠습니다.

❷ 기존 hello_cju 소스 코드

```
@bp.route('/')
def hello_world():
    return 'Hello world! Welcome to CJU.'
```

변경된 소스코드는 다음과 같습니다.

❸ 변경된 hello_cju 소스 코드

```
@bp.route('/')
def hello_world():
    # return 'Hello world! Welcome to CJU.'
    question_list = Question.query.order_by(Question.create_date.desc())
    return render_template(
        'question/question_list.html',
        question_list=question_list
    )
```

변경된 소스 코드에서 `Question.query.order_by(Question.create_date.desc())`의 의미는 `Question` 객체 중에서 생성 시간 `.create_date` 기준으로 역정렬 `.desc()` 하라는 뜻입니다.

`main_views.py`의 `hello_cju` 함수에서 템플릿 파일 `question/question_list.html`로 데이터 `question_list`를 전달하면, 템플릿 파일에서 정의한 대로 화면을 그려서(rendering) 브라우저로 전달하게 됩니다.

참고로 우리는 아직까지 `question/question_list.html` 만들지 않았습니다.

현재 상태에서 flask 서버를 실행시킨다면 그림 [Fig. 3.17](#) 같은 에러 메시지가 뜨게 됩니다.

① TemplateNotFound Exception

템플릿 파일이 없어서 생기는 에러(예외)

jinja2.exceptions.TemplateNotFound

jinja2.exceptions.TemplateNotFound: question/question_list.html

Traceback (most recent call last)

Fig. 3.17 TemplateNotFound 에러(Exception)

위 예외 메시지 아래로 에러가 발생한 추적 경로 (Traceback) 정보가 주루록 표시됩니다. 경우에 따라서는 수십 라인이 될 수도 있습니다. 개발자는 Traceback 경로를 차근차근 읽어보면 어디에서 왜, 어떤 것이 잘못되었는지 파악할 수 있습니다.

그림 Fig.3.17 화살표가 가리키는 빨간색 사각형 안에 `question/question_list.html` 파일이 없다는 의미입니다.

Flask 입장에서는 `main_views.py`에서 `question/question_list.html`로 context를 `question_list`라는 이름을 붙여서 보냈는데, 막상 찾아보니 데이터를 받을 `question/question_list.html` 자체가 없는 겁니다. 대략 난감하겠죠?

Flask는 고민하다가 이렇게 결정했을 겁니다.

② Flask의 고민

“데이터를 받아 처리할 템플릿 `.html` 자체가 없네...ㅠ.”

프로그래머가 깜빡한 모양이다.

에러(Exception)를 출력해서 알려줘야겠다!”

그래서 나오는 에러가 바로 그림 Fig.3.17 입니다.

3.5.1.2. 템플릿(`.html`) 파일 작성

이 문제를 어떻게 해결하면 될까요?

네, 맞습니다. `question/question_list.html` 파일을 만들면 됩니다.

어라? 그런데 어디에 만들라는 거지? 약간 당황스럽습니다. 하지만, 당황하지 마세요. Flask는 기본적으로 우리가 만든 프로젝트 폴더에 있는 `templates/` 디렉토리에서 템플릿을 찾도록 약속해 놨습니다. 여러분이 제 실습 코드를 그대로 따라왔다면 `hello_cju/templates/` 디렉토리가 됩니다.

어디에 템플릿 파일을 만들지 알았습니다. 그러면 만들면 되겠죠?

그림 Fig.3.18 참고하여 `hello_cju/templates/` 디렉토리 아래에 `question` 디렉토리를 만들고 그 안에 `question_list.html` 파일을 만들면 됩니다.

템플릿 파일 `question_list.html`을 만들었으니 내용을 채워 넣으면 됩니다. 우선 아래와 같이 코딩해 줍니다.

```
{% if question_list %}  
  <ul>  
    {% for question in question_list %}  
      <li>  
        <a href="/detail/{{ question.id }}"/>{{ question.title }}</a>  
      </li>  
    {% endfor %}  
  </ul>  
  
  {% else %}  
    <p>질문이 없습니다.</p>  
  {% endif %}
```

위 코드에서 `{%와 %}`로 둘러싸인 부분에 파이썬 명령어와 비슷한 것이 들어있는 표현이 있습니다. 이것을 템플릿 태그라고 합니다. HTML 문서에서는 다양한 태그(tag)를 이용하여 문서를 작성합니다.

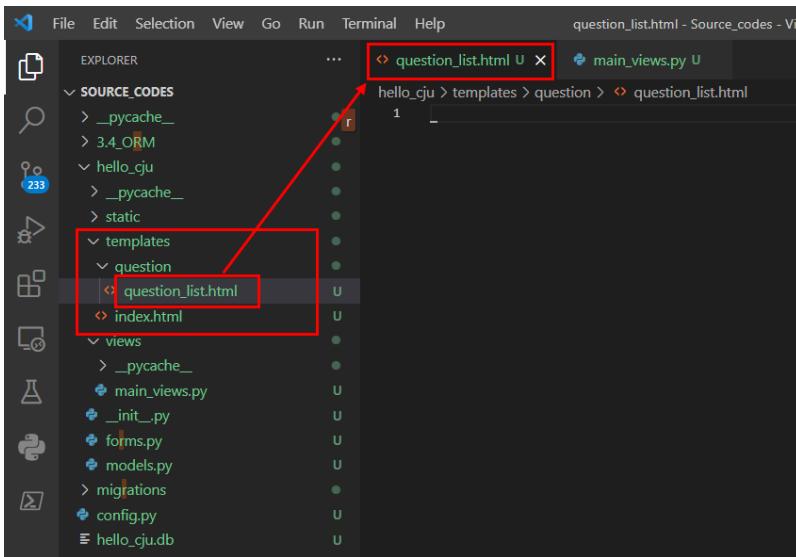


Fig. 3.18 템플릿 파일 `hello_cju/templates/question/question_list.html` 만들기

원래 `html` 문서는 말 그대로 한글이나 파워포인트와 같은 문서 그 자체이므로 프로그래밍 언어에서 사용하는 조건문, 반복문 등을 사용할 수 없습니다.

하지만 Flask에서 사용하는 `Jinja2` 템플릿 엔진은 Python 명령어를 이용하여 데이터를 처리하여 최종 문서를 만들 수 있도록 지원합니다.

Python 명령어를 사용하기 위해서는 `{% 파이썬 명령어 %}` 와 같은 형태로 사용합니다.

특정 데이터 값을 참조하기 위해서는 `{{ 참조할 자료 }}` 와 같은 형태를 사용합니다.

- {}내부에 % 로 둘러싸면 파이썬 명령어,

- {}내부에 {} 로 둘러싸면 데이터라고 생각하면 편합니다.

위 코드는 만약 `view` 파일에서 전달받은 `context`에(우리의 경우 `question_list`) 자료가 있을 경우는 `for`문을 돌면서 데이터를 `question`이라는 변수에 담은 다음 차례대로 `li` 태그와 `a` 태그를 이용해 출력하는 템플릿입니다. 만약 자료가 없을 경우에는 `p` 태그를 이용해 질문이 없습니다.라는 문자열만 출력합니다.

`a` 태그는 해당 문자열에 링크를 걸어주는 역할을 합니다. 우리는 질문 제목(`question.title`)에 링크를 걸어주고, 그 질문 제목을 클릭하면 `/detail/질문ID`로 연결하도록 코딩했습니다. 링크만 걸려 있지 아직은 작동하지 않습니다. `localhost:5000/detail/2` 요런 식으로 주소창에 입력해도 `Not Found`라는 메시지만 출력될 것입니다. 우리가 `view` 파일도, 템플릿 `.html` 파일도 작성하지 않았기 때문입니다.

Note

독특한 것은 `{% endfor %}, {% endif %}`와 같이 Python 명령어의 끝을 표시해야 한다는 것입니다. 파이썬은 줄바꿈과 들여쓰기로 명령어 실행 구역을 자동으로 감지하지만 `.html` 파일은 줄바꿈, 들여쓰기를 인식하지 못하기 때문에 별도로 지정을 해주어야 합니다.

Flask 서버를 재시작하고 `localhost:5000`으로 접속하면 그림 Fig. 3.19 같은 응답 화면을 볼 수 있습니다.

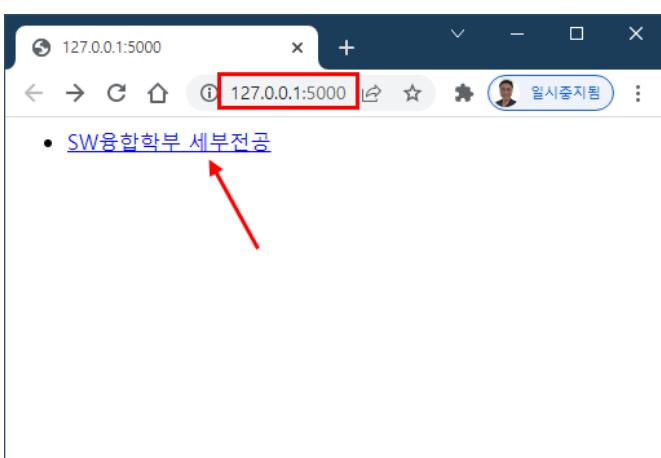


Fig. 3.19 SQLite DB에 저장된 질문 목록을 출력한

우리가 만든 SQLite DB에 질문이 1개 밖에 없으므로 하나의 질문 제목만 출력될 것입니다.

● Note

Flask에서 자주 활용하는 템플릿 태그 조건문과 반복문입니다.

- 조건문: `if, elif, else - endif` 구조
- 반복문: `for - endfor` 구조
 - `loop`: 반복문을 돌 때 반복순서(순번) 출력을 편리하게 해주는 태그
 - `loop.index`: 반복순서를 표시, 1부터 1씩 증가
 - `loop.index0`: 반복순서를 표시, 0부터 1씩 증가
 - `loop.first`: 반복순서가 처음이면 `True`, 아니면 `False`
 - `loop.last`: 반복순서가 마지막이면 `True`, 아니면 `False`

좀 더 다양한 템플릿 태그 정보는 [jinja](#) 엔진 공식 문서 ([click](#)) 참고하기 바랍니다.

3.5.2. Question 상세 조회

바로 이전 장 [Question List 조회](#)에서 질문 제목(title)을 목록으로 출력하였습니다. 구현할 때 `a` 태그를 활용해서 질문 제목에 링크를 걸어 주었습니다.

아래와 같이 코딩 했습니다. 기억 나시죠?

```
<a href="/detail/{{ question.id }}">{{ question.title }}</a>
```

위 코드는 질문 제목을 클릭하면 `localhost:5000/detail/질문아이디`라는 경로로 이동하고 거기에 있는 템플릿을 작동시켜 결과를 확인하라는 의미입니다. 다시 말하면 `Question` 모델에 저장된 데이터 중에서 `id`가 2인 데이터를 조회해 주세요라는 의미와 동일합니다.

하지만 질문을 클릭하거나 브라우저 주소창에 `localhost:5000/detail/질문아이디`를 입력하면 `Not Found` 에러가 발생합니다.

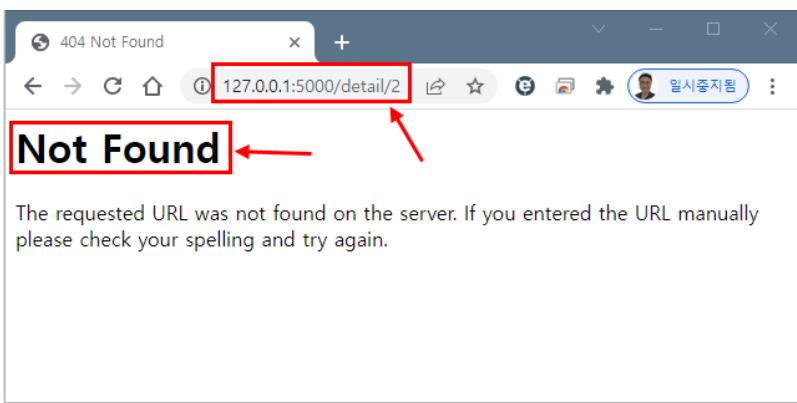


Fig. 3.20 질문 상세 조회 `Not Found` 에러

그림 [Fig. 3.20](#) 에러 메시지를 해석하면 다음과 같습니다.

Flask 서버로 요청한 URL이 존재하지 않습니다. 브라우저 직접 주소를 쳐 넣거나 주소에 오타가 있는지 확인하세요.

이걸 어떻게 해결하면 될까요?

네, 맞습니다. 우리가 이전 장에서 했던 작업을 동일하게 반복하면 됩니다.

- 먼저 `main_view.py`에서 `Blueprint` 객체를 이용해 경로를 정해주고,
- 질문에 대한 상세 내용을 보여줄 함수를 만들면 됩니다.
- 마지막으로 view 파일에서 전달한 `context` 정보를 처리할 템플릿 `.html` 파일을 만듭니다.

순서대로 진행해 보겠습니다.

3.5.2.1. view 파일 수정

hello_cju/views/vmain_views.py 파일을 편집합니다.

```
`localhost:5000/detail/질문아이디`
```

먼저 위와 같은 경로로 요청이 있을 경우 대응하는 URL 패턴을 지정합니다. Blueprint 객체를 활용하여 다음과 같이 라우팅 함수를 코딩해 줍니다.

```
@bp.route('/detail/<int:question_id>')
def detail(question_id):
    question = Question.query.get(question_id)
    return render_template(
        template_name_or_list='question/question_detail.html',
        context=question,
    )
```

위 코드에서 라우팅 함수 `@bp.route('/detail/<int:question_id>')`는 해당 경로로 서버 요청이 들어오면 URL 경로에서 실행할 함수(데코레이터 `bp.route` 바로 다음에 등장하는 함수, 여기서는 `detail` 함수)를 실행하라는 의미입니다.

`def detail(question_id):`에서 매개변수 `question_id`라는 인자가 등장합니다. `question_id` 값은 라우팅 함수에서 지정한 `<int:question_id>` 값이 전달됩니다.

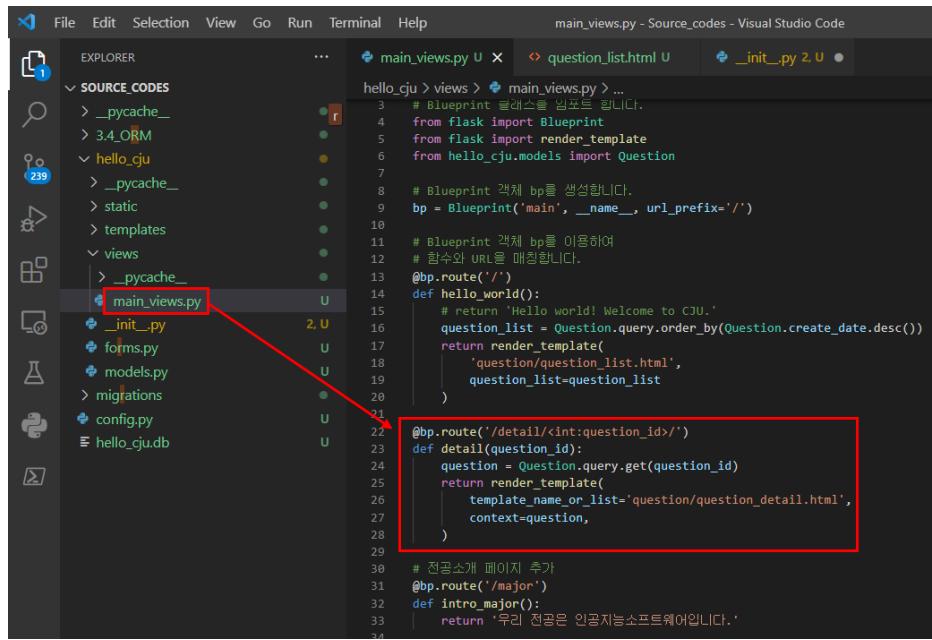
예를 들어 `localhost:5000/detail/2`라는 주소(URL)로 서비스 요청이 들어오면 `detail(2)`와 같이 view 함수가 작동하게 됩니다.

`detail` 함수는 전달받은 `question_id`를 활용하여 `Question` ORM DB에 저장된 데이터 중에서 `id` 값이 `question_id`와 동일한 데이터를 찾고 `Question` 객체를 리턴합니다. 우리 코드에서는 리턴된 객체를 `question`이라는 이름으로 받았습니다.

`render_template` 함수에서는 [render template 깊은 이해](#)에서 설명한 대로 명시적으로 `template_name_or_list`에 템플릿 파일 경로를 지정하고, `question` 객체를 `context`라는 이름을 붙여서 전달하도록 코딩하였습니다.

이제 경로 설정을 마치고 해당 경로로 요청이 들어왔을 때 서버에서 처리할 함수 `detail`까지 만들었습니다.

VS code 화면으로 보면 그림 [Fig. 3.21](#) 같은 형태입니다.



```
File Edit Selection View Go Run Terminal Help
main_views.py - Source Codes - Visual Studio Code
EXPLORER ... main_views.py U question_list.html _init_.py 2.U
hello_cju > views > main_views.py ...
3 # Blueprint 클래스를 import합니다.
4 from flask import Blueprint
5 from flask import render_template
6 from hello_cju.models import Question
7
8 # Blueprint 객체 bp를 생성합니다.
9 bp = Blueprint('main', __name__, url_prefix='/')
10
11 # Blueprint 객체 bp를 이용하여
12 # 험수의 URL을 매칭합니다.
13 @bp.route('/')
14 def hello_world():
15     # return 'Hello world! Welcome to CJU.'
16     question_list = Question.query.order_by(Question.create_date.desc())
17     return render_template(
18         'question/question_list.html',
19         question_list=question_list
20     )
21
22 @bp.route('/detail/<int:question_id>')
23 def detail(question_id):
24     question = Question.query.get(question_id)
25     return render_template(
26         template_name_or_list='question/question_detail.html',
27         context=question,
28     )
29
30 # 전공소개 페이지 추가
31 @bp.route('/major')
32 def intro_major():
33     return '우리 전공은 인공지능소프트웨어입니다.'
```

[Fig. 3.21](#) main_view.py에 라우트와 detail 함수를 추가

3.5.2.2. 템플릿 파일 작성

[view 파일 수정](#)을 끝내고 해당 주소로 요청을 보내면 여전히 `TemplateNotFound` 예외가 발생할 것입니다. 그림 [Fig. 3.17](#) 유사한 예러가 발생하니 참고하기 바랍니다.

당연하겠죠? `main_view.py`에 있는 `detail` 함수에서는 템플릿 파일 `question/question_detail.html` 위치로 `context`라는 이름을 붙여서 데이터를 보낸다고 했는데 정작 해당 템플릿 파일이 없으니 Flask가 할 수 있는 유일한 방법은 에러를 내보내는 것 밖에 없으니까요.

우리는 `question` 경로에 템플릿 파일 `question_detail.html` 만들어 주면 됩니다. 파일에 입력할 내용은 다음과 같습니다.

```
<h1>{{ context.title }} </h1>  
  
<div>  
    {{ context.contents }}  
</div>
```

위 코드에서 `main_view.py`에 있는 `detail` 함수에서 데이터를 `context`라는 이름으로 템플릿 파일 `question_detail.html`에 보냈으므로 값을 참조할 때 `context`를 사용하였습니다.

먼저 템플릿을 코딩하기 위하여 해당 경로에 `question_detail.html` 파일을 만들어 줍니다. VS code의 경우 그림 Fig.3.22 같습니다.

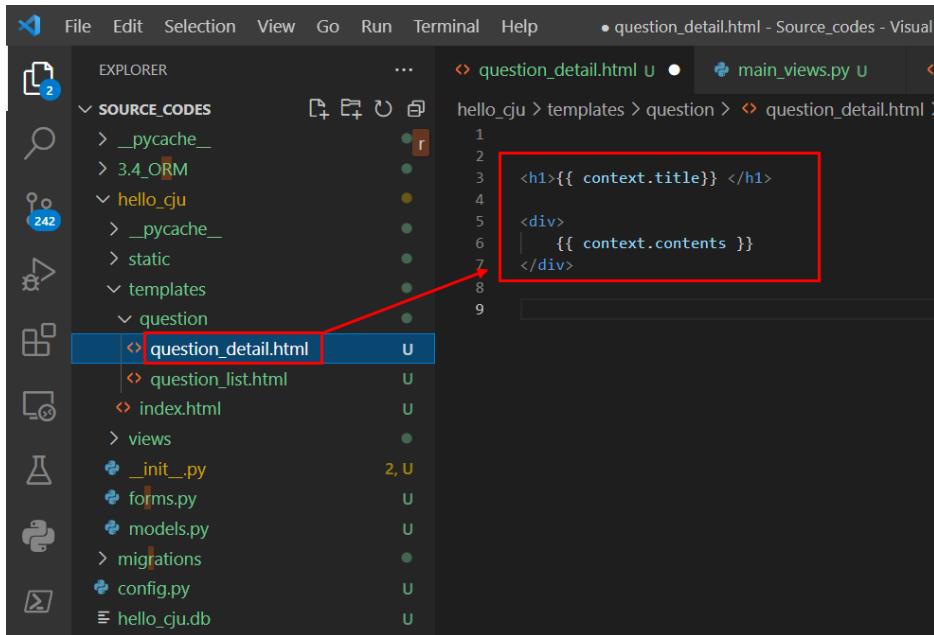


Fig. 3.22 `main_view.py`에 라우트와 `detail` 함수를 추가

Flask 서버를 실행하고 `127.0.0.1:5000` 또는 `localhost:5000`으로 접속하고 질문 리스트 중 하나를 클릭하면 브라우저 주소창에 URL이 변경하여 Flask 서버에게 서비스를 요청합니다. 요청한 서비스 응답으로 `main_view.py`의 `detail` 함수에서 처리한 결과를 받아 템플릿 `question_detail.html`에서 렌더링한 결과를 확인할 수 있습니다.

그림 Fig.3.23 참고하세요.

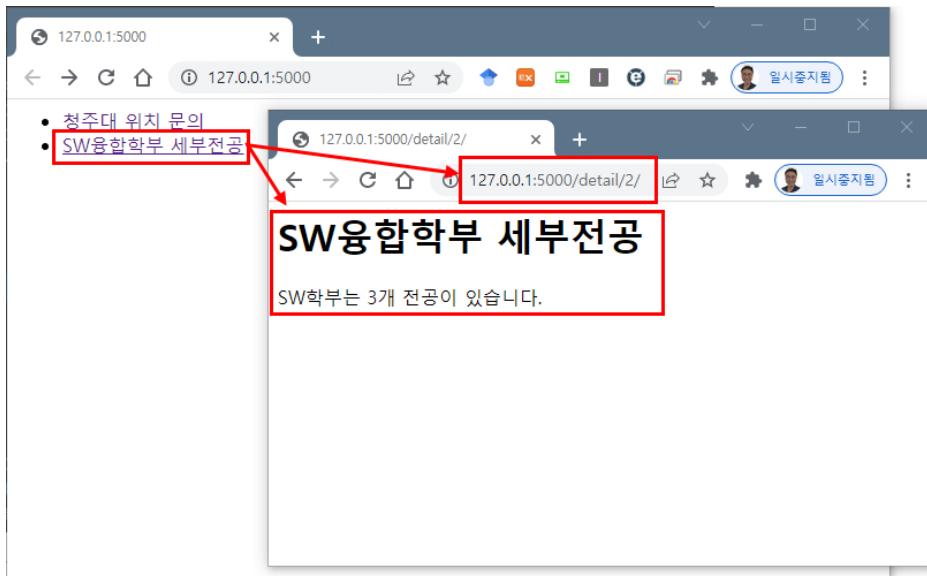


Fig. 3.23 질문 세부 정보 확인

i Note

질문 목록과 질문의 세부 내용은 [모델에 CRUD 해보기](#)에서 Flask ORM CRUD 실습 과정에서 개인별로 입력한 데이터에 따라 화면이 다를 수 있습니다.

우리는 질문 목록을 만들고, 각 질문에 대한 세부 정보를 조회할 수 있게 되었습니다.

다음은 존재하지 않는 페이지를 어떻게 처리할 것인지 살펴보겠습니다.

준비 되었나요?

[Next](#) 아이콘을 클릭하여 이동하세요.

3.5.3. 존재하지 않는 페이지 처리 - 404 오류

앞 절 [Question 상세 조회](#)에서 `Question` ORM DB에 저장된 `id`를 이용하여 질문의 세부 정보를 보여주는 것을 실습했습니다.

그런데 한 가지 문제점이 있습니다.

만약에 사용자가 세부 정보를 조회하기 위해서 브라우저 주소창에 `localhost:5000/detail/300`이라고 입력하여 서비스를 요청할 수 있습니다.

그런데 Flask ORM 모델에 저장된 데이터가 2개 뿐이어서 `id`가 300인 데이터가 없다면 어떻게 될까요? 아마도 그림 Fig. 3.24 같은 결과가 나올 겁니다.

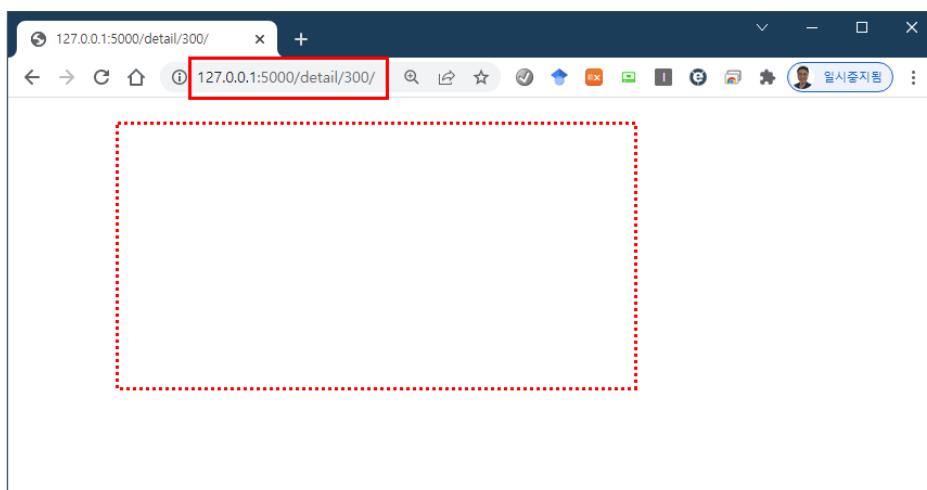


Fig. 3.24 보여줄 데이터가 없어서 빈 화면으로 렌더링된 결과 (점선 사각형: 아무런 정보도 없음)

도대체 그림 Fig. 3.24에 어떤 문제가 있는 것일까요?

빈 화면이 의미하는 것이 **id**가 300인 데이터는 존재하지만 보여줄 데이터가 없는 것인지, 아니면 데이터 자체가 존재하지 않는 것인지 구분하기 어렵습니다. 일반적으로 인터넷 사용자가 주소창에 원하는 URL을 입력했는데 에러가 없다면 해당 페이지가 존재하는 것으로 인식할 것입니다.

하지만 이 상황은 잘못된 것입니다. 사용자가 잘못 판단을 하게 됩니다. 데이터가 존재하지 않는다는 메시지를 보여 주는 것이 맞습니다.

이 상황에서 보여주는 에러 메시지는 보통 **Not Found (404)**를 출력하게 됩니다. 여기서 숫자 404는 통신 프로토콜 HTTP 응답 코드 중 하나입니다.

HTTP 응답 코드

HTTP 프로토콜을 사용해서 (쉽게 인터넷 브라우저를 이용해서) 서버에 서비스 요청(request)했을 경우 서버에서 보내오는 응답 코드입니다. 응답 코드는 서버가 요청을 받을 때마다 요청한 측에 보내줍니다. 주저리 주저리 쓰는 것 보다는 숫자로 약속하고, 숫자만 보냅니다. 웹 시스템 개발자는 이런 코드를 참고하여 다양한 처리를 하는데 사용합니다.

가장 자주 사용하는 코드는 다음과 같습니다.

- 200: 성공
- 500: 서버 오류 (Internal Server Error)
- 404: 서버에서 요청한 페이지를 찾을 수 없음 (Not Found)

이 밖에도 다양한 에러 코드가 있습니다.

자세한 내용은 Mozilla 공식 문서 ([click](#)) 참고하세요.

Flask는 이런 상황에 편리하게 대응할 수 있도록 **get_or_404** 함수를 제공합니다. 참 고맙죠?

get_or_404() 함수는 찾고자 하는 데이터가 없는 경우에 404 페이지를 자동으로 출력해주는 함수입니다. **get()** 함수가 찾을 값이 없다면 **None**을 리턴하는 것과 약간 차이가 있습니다.

사용은 간단합니다.

우리가 지금까지 해왔던 코드를 사용해서 **get_or_404()**를 적용해 보겠습니다.

```
@bp.route('/detail/<int:question_id>')
def detail(question_id):
    # question = Question.query.get(question_id)
    question = Question.query.get_or_404(question_id)
    return render_template(
        template_name_or_list='question/question_detail.html',
        context=question,
    )
```

`views/main_views.py`에 있는 **detail** 함수에서 다음과 같이 수정했습니다.

```
Question.query.get(question_id) \(\to\) Question.query.get_or_404(question_id)
```

이제 존재하지 않는 데이터를 다시 한번 요청해 보겠습니다. 결과는 그림 Fig. 3.25 와 같습니다.

그림 Fig. 3.24에는 아무런 내용이 없었지만, **get_or_404()** 함수를 사용한 결과인 그림 Fig. 3.25에는 404 에러 (**Not Found**)가 정상적으로 나타납니다.

우리는 질문 목록을 조회하고, 각 질문에 대한 세부 내용을 확인할 수 있습니다. 게다가 존재하지 않는 데이터에 대한 결과를 **get_or_404** 함수를 이용하여 **Not Found** 메시지를 출력하는 기능까지 완벽하게 이해 했습니다.

다음은 무엇일까요? 여전히 갈증이 느껴지나요?

다음에 공부할 내용은 **view** 기능을 분리하는 방법입니다.

왜 **view**를 분리하고, 어떻게 하는지 궁금하신가요?

준비가 되었다면 **Next** 아이콘을 클릭해서 이동해 주세요

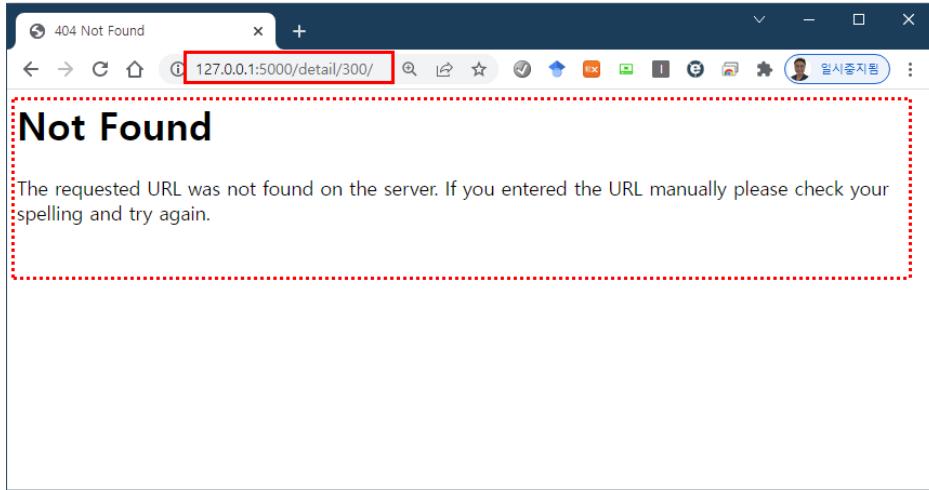


Fig. 3.25 `get_or_404()`를 사용하여 404 에러 메시지 출력

3.5.4. 기능 분리 - Blueprint 활용

우리는 지금까지 `view` 처리를 `views/main_view.py`라는 모듈을 만들고 그 안에 다양한 함수를 만들어서 처리해 왔습니다.

우리가 만들 웹 시스템이 간단하다면 이런 방법으로 하나의 모듈에 다양한 `view` 함수를 모아 놓는 것도 나쁘지는 않습니다. 하지만, 구현할 시스템이 복잡해 진다면 상황이 달라집니다.

하나의 모듈 안에서 특정 기능의 함수를 찾아서 수정하거나 업데이트하는데 불편할 뿐만 아니라 모듈 안에 있는 함수들끼리 의존성이 생겨서 예기치 못한 결과를 얻을 수도 있습니다.

우리는 `main_views.py`에 있는 질문 관련 기능을 분리해 보도록 하겠습니다. 기능을 분리하려면 새로운 `view` 파일을 만들어야겠죠? `views/` 디렉토리 아래에 `question_view.py` 파일을 생성하고 `main_view.py` 기능 중 질문에 관련한 내용을 이동하겠습니다. 복붙(복사하여 붙여넣기)하면 됩니다.

예전에 배웠던 [Blueprint 클래스 활용](#)을 참고하면서 진행해 보겠습니다.

3.5.4.1. `view` 파일 분리 및 코딩

먼저 `question_view.py` 파일을 생성하고 다음과 같이 코드를 입력합니다.

```
# 파일명: views/question_view.py

# Blueprint 클래스를 임포트 합니다.
from flask import Blueprint
from flask import render_template
from hello_cju.models import Question

# Blueprint 객체 bp를 생성합니다.
# 변경사항: 'main' -> 'question'
#           url_prefix='/' -> '/question'
bp = Blueprint('question', __name__, url_prefix='/question')

# Blueprint 객체 bp를 이용하여
# 험수와 URL을 매칭합니다.
@bp.route('/list/') # 변경사항: '/' -> '/List/'
def question_list(): # 변경사항: hello_cju ->
    question_list = Question.query.order_by(Question.create_date.desc())
    return render_template(
        'question/question_list.html',
        question_list=question_list
    )

@bp.route('/detail/<int:question_id>/')
def detail(question_id):
    # question = Question.query.get(question_id)
    question = Question.query.get_or_404(question_id)
    return render_template(
        template_name_or_list='question/question_detail.html',
        context=question,
    )
```

변경사항은 다음과 같습니다.

- Blueprint 객체 생성
 - name 인자값 변경: main \(\to\) question
 - url_prefix 인자값 변경: / \(\to\) /question
- 라우트 함수
 - bp.route() 인자값 변경: / \(\to\) /list/
 - hello_cju() 함수이름 변경: hello_cju \(\to\) question_list

VS code에 입력한 화면은 그림 Fig. 3.26 와 같습니다.

The screenshot shows the VS Code interface with the 'EXPLORER' view on the left and the 'question_views.py' file open in the center. A red box highlights the code area. The code defines a Blueprint 'bp' for the '/question' URL prefix, routes it to the 'question_list' function, and then defines a route for '/list/' to the same function. It also defines a 'detail' route for '/detail/<int:question_id>'.

```
# 파일명: views/question_views.py
1 # Blueprint 클래스를 임포트 합니다.
2 from flask import Blueprint
3 from flask import render_template
4 from hello_cju.models import Question
5
6 # Blueprint 객체 bp를 생성합니다.
7 # 변경사항: 'main' -> 'question'
8 #         url_prefix='/' -> '/question'
9 bp = Blueprint('question', __name__, url_prefix='/question')
10
11 # Blueprint 객체 bp를 이용하여
12 # 함수와 URL을 매칭합니다.
13 @bp.route('/list/') # 변경사항: '/' -> '/list/'
14 def question_list(): # 변경사항: hello_cju ->
15     question_list = Question.query.order_by(Question.create_date.desc())
16     return render_template(
17         'question/question_list.html',
18         question_list=question_list
19     )
20
21 @bp.route('/detail/<int:question_id>/')
22 def detail(question_id):
23     # question = Question.query.get(question_id)
24     question = Question.query.get_or_404(question_id)
25     return render_template(
26         'question/question_detail.html',
27         context=question,
28         template_name_or_list='question/question_detail.html',
29         question_id=question_id
30     )
31
```

Fig. 3.26 question_views.py 생성 및 view 코딩

3.5.4.2. `__init__.py`에 블루프린트 등록

[view 파일 분리 및 코딩](#)에서 생성한 Blueprint 객체를 `__init__.py`에 등록해 주어야 합니다. 그래야 프로그램이 시작될 때 정상적으로 경로 라우팅이 가능하겠죠?

[프로젝트명/_init_.py](#) 파일을 열고 `view` 파일 임포트 하는 위치에 [view 파일 분리 및 코딩](#)에서 만든 `question_views.py` 파일을 임포트 합니다. 모듈을 임포트하는 경우 확장자 `.py`는 생략합니다.

관련 코드는 다음과 같습니다.

```
# question_views 모듈 추가 임포트합니다.
from .views import main_views, question_views

# question_view에서 생성한
# Blueprint 객체 bp를 등록합니다.
app.register_blueprint(main_views.bp)
app.register_blueprint(question_views.bp)
```

`views`를 임포트할 때 `question_views`를 추가하고, `question_view`에서 생성한 Blueprint 객체 `bp`를 `app`에 등록합니다. 관련 명령어는 다음과 같습니다.

```
app.register_blueprint(question_views.bp)
```

3.5.4.3. `main_view` 수정 - `redirect`, `url_for` 적용

`questions_view.py`를 완성했으므로 `main_views.py`에서 불필요한 코드를 삭제합니다. 수정한 내용과 주석을 포함한 코드는 다음과 같습니다.

```

# 파일명: views/main_views.py

# Blueprint 클래스를 임포트 합니다.
from flask import Blueprint

# 추가로 import 되는 모듈
from flask import url_for
from werkzeug.utils import redirect

from flask import render_template
from hello_cju.models import Question

# Blueprint 객체 bp를 생성합니다.
bp = Blueprint('main', __name__, url_prefix='/')

# Blueprint 객체 bp를 이용하여
# 함수와 URL을 매칭합니다.
@bp.route('/')
def index():
    # question이라는 이름으로 등록된
    # 블루프린트 객체(bp)에 연결된 함수 중
    # question_list와 연결된 URL을 추출하여
    # 리다이렉션 수행
    return redirect(url_for('question.question_list'))

# hello_cju 함수는 단순히 인사말 문자열만 출력하도록 수정
# URL 경로를 /hello 로 변경
@bp.route('/hello')
def hello_cju():
    return 'Hello world! Welcome to CJU.'
    # question_list = Question.query.order_by(Question.create_date.desc())
    # return render_template(
    #     'question/question_list.html',
    #     question_list=question_list
    # )

### 아래 코드는 불필요한 내용 -> 삭제함 ####

# @bp.route('/detail/<int:question_id>/')
# def detail(question_id):
#     # question = Question.query.get(question_id)
#     question = Question.query.get_or_404(question_id)
#     return render_template(
#         template_name_or_list='question/question_detail.html',
#         context=question,
#     )

# # 전공소개 페이지 추가
# @bp.route('/major')
# def intro_major():
#     return '우리 전공은 인공지능소프트웨어입니다.'

```

지저분한 코드와 주석을 제거하면 다음과 같습니다.

```

# 파일명: views/main_views.py

from flask import Blueprint
from flask import url_for
from werkzeug.utils import redirect
from flask import render_template
from hello_cju.models import Question

bp = Blueprint('main', __name__, url_prefix='/')

@bp.route('/')
def index():
    return redirect(url_for('question.question_list'))

@bp.route('/hello')
def hello_cju():
    return 'Hello world! Welcome to CJU.'

```

추가로 임포트한 모듈은 `url_for, redirect` 입니다.

ⓘ Note

`redirect` 모듈은 `werkzeug.utils` 패키지에서 임포트 했습니다. Flask는 `werkzeug`를 랩핑하여 `WSGI`를 사용합니다. Flask를 설치하면 `werkzeug`도 같이 설치 됩니다. `werkzeug`에서 지원하는 `redirect` 함수는 특정 URL로 요청이 들어온 경우 지정된 URL로 변경해주는 역할을 합니다.

`WSGI`는 'Web Server Gateway Interface' 약어로 Python 언어로 작성된 웹 시스템이 웹 서버와 통신하도록 만들어진 파이썬 표준 인터페이스입니다. 세부 내용은 PEP 3333 ([click](#)) 참고하세요.

ⓘ url_for() 함수

첫 번째 인자 `endpoint`는 엔드포인트 주소로 반드시 전달되어야 합니다. 두 번째 인자 `values`는 옵션입니다. 값이 전달되면 엔드포인트에 `value` 값이 추가되어 리턴됩니다.

`url_for()` 함수에 대한 자세한 설명은 Flask 공식문서에 ([click](#)) 자세히 설명되어 있으니 참고하기 바랍니다.

```
url_for(endpoint, **values)

Parameters:
    endpoint (str) - the endpoint of the URL (name of the function)
    values (Any) - the variable arguments of the URL rule

Return type:
    str
```

약간 복잡해 보이는 코드 `return redirect(url_for('question.question_list'))`를 살펴보겠습니다.

먼저 `redirect()` 함수의 인자로 전달된 `url_for('question.question_list')`부터 살펴 볼까요?

`url_for()`에 절달되는 인자는 현재 시스템에서 사용할 URL을 만들어서 리턴해 줍니다. 현재 우리가 사용하는 서버 주소는 `127.0.0.1:5000` (또는 `localhost:5000`) 입니다.

블루프린트 `bp` 객체에 붙여진 이름 `question`에 배정된 라우트는 `/question` 였습니다 (`question_views.py` 파일을 참고하세요). `question_list`는 블루프린트 객체 `bp`에 등록된 함수입니다. `question_list` 함수의 라우트는 `/list/` 였습니다.

결국 '`question.question_list`' 반환하는 값은 `/question`과 `/list/`가 결합된 `/question/list/` 입니다.

`url_for('question.question_list')`를 다시 쓰면 `url_for('/question/list/')`가 됩니다. `url_for`는 서버 주소까지 결합하여 최종 라우트(URL) `localhost:5000/question/list/` 를 리턴하게 됩니다.

결국 `redirect(url_for('question.question_list'))`는 `redirect(localhost:5000/question/list/)`와 동일해 집니다.

`@bp.route('/')`에 의해 루트 경로로 접속하면 `redicrect` 함수에 의해 자동으로 URL 경로가 `localhost:5000/question/list/`로 변경되어 요청됩니다.

3.5.4.4. 템플릿에 url_for 적용

이제 템플릿에 적용된 하드 코딩 (Hard Coding)을 `url_for`를 이용해서 제거해 보겠습니다. `하드 코딩`이 낯설다면 다음 정보를 읽어보세요.

ⓘ 하드 코딩 (hard coding) 이란?

프로그램에서 사용할 데이터를 소스코드에 직접 입력하는 코딩 스타일을 의미합니다. 예를 들어 다음과 같은 코딩 스타일입니다. 이 코드는 출력 할 내용을 변경해야 하는 경우 함수 내부까지 찾아가서 일일히 바꿔줘야 합니다.

```
def print_msg():
    print('Hello world')
```

변경에 대처하려면 다음과 같은 소프트 코딩(Soft Coding, 하드 코딩의 반대말) 해야 합니다.

```
def print_msg(info):
    print(info)
```

위 코드는 `print_msg` 함수를 호출하때 전달하는 인자값에 따라 출력하는 내용이 달라집니다. 출력 내용을 변경하고자 할 때 편리합니다. 이런 점에서 소프트 코딩은 유지보수 측면에서 유리합니다. 추가적으로 하드 코딩은 보안에 취약합니다. 소스코드 내부에 모든 정보들이 입력되어 있기 때문에 소스코드가 공개될 경우 보안 취약성이 증가합니다.

우리가 만든 템플릿 `question_list.html`에도 하드 코딩된 부분이 있습니다. 바로 다음 코드가 하드 코딩된 것입니다.

```
<a href="/detail/{{ question.id }}"> {{ question.title }} </a>
```

소프트 코딩으로 변경하면 다음과 같습니다.

```
<a href="{{ url_for('question.detail', question_id=question.id) }}> {{ question.title }} </a>
```

템플릿 `question_list.html`의 전체 코드는 다음과 같습니다.

```
{% if question_list %}
  <ul>
    {% for question in question_list %}
      <li>
        <a href="{{ url_for('question.detail', question_id=question.id) }}>
          {{ question.title }}
        </a>
      </li>
    {% endfor %}
  </ul>

{% else %}
  <p>질문이 없습니다.</p>
<% endif %>
```

위 코드에서 `url_for` 함수는 `question`이라는 이름을 가진 블루프린트 객체 `bp`에 등록된 함수 `detail`의 URL을 찾고, `detail` 함수에서 필요한 `question_id` 값을 전달하여 최정적으로 URL을 생성해서 리턴합니다.

참고 사항으로 예전에 만들었던 블루프린트 객체와 `detail` 함수를 한번 더 표시하였으니 아래 코드를 보면서 차근차근 생각해 보기 바랍니다.

```
bp = Blueprint('question', __name__, url_prefix='/question')

@bp.route('/detail/<int:question_id>')
def detail(question_id):
    # question = Question.query.get(question_id)
    question = Question.query.get_or_404(question_id)
    return render_template(
        template_name_or_list='question/question_detail.html',
        context=question,
    )
```

템플릿에서 왜 `url_for`를 사용할까요? 웹 사이트 구조 변경 (URL) 변경 시 대처하기 편리하기 때문입니다.

```
<a href="/detail/{{ question.id }}"> {{ question.title }} </a>
```

위와 같이 코딩했었는데 URL 구조를 다음과 같이 변경하면 어떻게 될까요?

```
# 원래 URL 구조
localhost:5000/detail/question/2

# URL 구조를 아래와 같이 변경
localhost:5000/detail/2/question
```

`url_for`를 사용했다면 추가로 수정해야 할 일이 거의 없습니다. `question_view.py`에서 블루프린트 라우트 함수만 변경하면 끝입니다. 다음과 같이 변경할 수 있습니다.

```
# @bp.route('/detail/<int:question_id>/')
@bp.route('/<int:question_id>/detail/')
```

하지만 하드코딩을 했다면 모든 템플릿 파일을 확인하고, 해당되는 모든 URL을 변경해 주어야 합니다. 여간 고된 일이 아닐 수 없습니다. 힘든 것도 힘든 거지만, 실수로 잘못 입력하여 에러가 발생할 수도 있습니다. 하드 코딩을 하니 여러 가지로 피곤합니다. 여러분은 가급적 하드 코딩은 피하는 것이 좋습니다.

드디어 하드 코딩을 제거하기 위한 모든 수정이 끝났습니다.

Flask 서버를 작동시키고 접속해 보면 예전과 동일하게 작동하는 것을 확인할 수 있습니다.

각자 확인해 보시기 바랍니다.

❶ Where we are?

우리는 여기까지 오면서 질문 게시판을 만드는 것까지 마쳤습니다. 앞으로 해야 할 일은 질문에 대한 답변을 다는 기능을 추가해야 합니다.

답변 기능 구현에 대해 배워볼까요?

준비가 끝난 여러분은 [Next](#) 아이콘을 눌러 이동해 주세요.

3.6. 게시판 댓글 구현

우리는 다음과 같이 기본기 5가지를 공부하였습니다.

❷ Flask 웹 시스템 구축을 위한 10가지 기본기

1. [프로젝트 기본 구조](#) \(\to\) [Clear!](#)
2. [Application Factory 패턴](#) \(\to\) [Clear!](#)
3. [Blueprint 클래스 활용](#) \(\to\) [Clear!](#)
4. [ORM 모델 완벽 이해](#) \(\to\) [Clear!](#)
5. [질문 게시판 만들기](#) \(\to\) [Clear!](#)
6. [게시판 댓글 구현](#) \(\to\) 지금 도전!
7. [질문 & 댓글 등록 - Flask Form 활용](#)
8. [예쁘게 - CSS 적용](#)
9. [더 예쁘게 - Bootstrap 활용](#)
10. [HTML 구조와 Template 상속](#)

앞서 우리는 질문을 질문을 입력하고 질문에 대한 세부 내용을 조회하는 것을 구현하였습니다. 게시판을 게시판처럼 구현하려면 질문에 대한 답변 기능도 구현해야겠죠?

질문 목록을 구현하는 실습 [Question List 조회](#)에서는 [view](#) 파일을 먼저 코딩하고, 이어서 템플릿 [.html](#) 파일을 작성하였습니다.

댓글의 경우는 반대로 진행하고자 합니다. 왜냐하면 댓글은 질문 상세조회를 상태, 즉 템플릿 [.html](#) 내용을 읽어봐야 댓글을 달 수 있기 때문입니다.

사용자가 댓글을 입력하고 등록 버튼을 누르면 댓글 정보가 Flask 서버에 전달되어야 하고, 서버는 댓글 정보를 받아서 ORM 모델 [Answer](#)에 등록해야 합니다. 사용자 댓글 정보를 DB에 저장해야 한다는 의미와 같습니다.

따라서 사용자가 댓글을 입력하는 템플릿을 먼저 작성하고, 이어서 [view](#) 파일을 코딩하겠습니다.

다음과 같은 순서로 댓글 등록 기능을 구현하도록 합니다.

❸ 댓글 구현 순서

1. 댓글 입력을 위한 템플릿 파일 [question_detail.html](#) 수정
2. [view](#) 파일 [answer_views.py](#) 작성
3. [__init__.py](#)에 블루프린트 등록
4. 댓글 표시를 위한 템플릿 파일 [question_detail.html](#) 수정

❹ Note

우리는 템플릿 파일을 2번 업그레이드 할 예정입니다.

3.6.1. 댓글 입력을 위한 템플릿 파일 [question_detail.html](#) 수정

예전에 만들었던 [templates/question/question_detail.html](#) 파일을 수정하여 댓글 입력을 위한 기능을 추가합니다.

입력 코드는 다음과 같습니다.

```
<h1>{{ context.title }} </h1>  
<div>  
  {{ context.contents }}  
</div>  
  
<!-- 댓글 입력을 위해 추가한 부분 -->  
<form action="{{ url_for('answer.create', question_id=context.id) }}" method="post">  
  <textarea name="contents" id="contents" cols="30" rows="10"></textarea>  
  <input type="button" value="댓글 등록">  
</form>
```

위 코드에서 사용된 HTML 태그는 3가지 `form`, `textarea`, `input` 입니다. 각각에 대한 세부 설명은 아래 정보를 참고하세요.

i form 태그

서버에 정보를 전송하기 위한 태그입니다. 우리가 사용할 속성은 다음과 같습니다.

- `action`: 데이터를 받아서 처리할 서버의 주소
- `method`: 데이터를 전송할 때 사용할 HTTP 전송 방식
 - `post`: 전송할 데이터를 HTTP 메시지 body 부분에 담아서 전송
 - `get`: URL 주소 끝 부분에 ?를 붙이고 `name1=value1&name2=value2...` 이런 형식으로 전송합니다. 서버는 서버에서는 `name1`과 `name2`라는 파라미터 명으로 각각 `value1`과 `value2`의 파라미터 값을 전달 받게 됩니다.

`form` 태그: 모질라 공식 문서 ([click](#))

i textarea 태그

여러줄에 걸친 텍스트를 입력하는 편집창을 제공합니다.

- `name`: `textarea`를 제어하기 위한 이름
- `id`: HTML 문서 내 전체에서 사용할 고유 식별자, 주로 CSS에서 많이 활용함
- `'cols'`: 텍스트 입력창의 폭 (양의 정수값 사용)
- `'rows'`: 텍스트 입력창에 볼 수 있는 라인 수

`textarea` 태그: 모질라 공식 문서 ([click](#))

i input 태그

마우스 클릭이 가능한 버튼을 제공합니다.

- `type`: 버튼을 클릭했을 때 수행되는 동작
 - `submit`: `form` 태그로 둘러싸인 영역의 데이터를 서버로 전송
 - `reset`: 모든 값을 초기화
 - `button`: 클릭했을 때 아무 것도 하지 않음. 클라이언트 쪽 스크립트와 연결할 경우에 사용

`input` 태그: 모질라 공식 문서 ([click](#))

그리면 위 코드를 분석해 볼까요? 어떻게 돌아가는지 이해되면 훨씬 좋겠죠?

`form` 태그 내부에 정의된 `input` 태그의 `댓글등록` 버튼을 클릭하면 `textarea`에 입력된 데이터를 `{{ url_for('answer.create', question_id=question.id) }}`에서 정의된 위치로 `post` 방식으로 서비스를 요청합니다.

다시 말하면 브라우저에 `url_for('answer.create', question_id=question.id)` 주소를 입력하여 서버 요청을 하는데 `textarea` 데이터를 담아서 요청한다는 의미입니다.

Flask 서버를 실행하고 질문 목록에서 세부 내용을 조회하기 위해 하난의 질문 제목을 클릭하면 우리가 코딩한 대로 서버로부터 업데이트 된 웹페이지를 제공 받을 수 있을까요?

실행해 보도록 하겠습니다. 실행 결과는 그림 [Fig.3.27](#)과 같이 나타납니다.

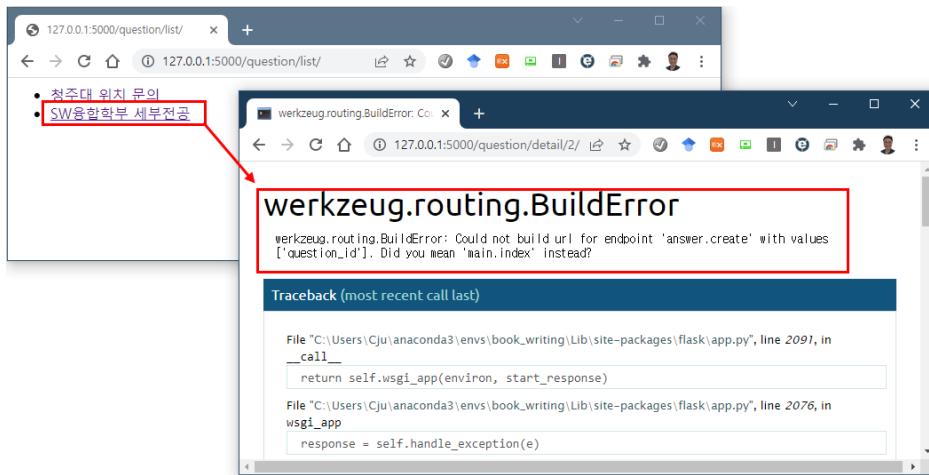


Fig. 3.27 `get_or_404()`를 사용하여 404 에러 메시지 출력

그림 Fig. 3.27에서 질문 목록 중 하나를 클릭했더니 오른쪽과 같은 에러가 나타났습니다.

Flask 서버는 우리가 원한 결과를 보여주는 대신에 에러 `werkzeug.routing.BuildError` 내용을 우리에게 보냈습니다. 그리고 친절하게 어디서 어떻게 에러가 발생했는지 에러 추적 `Traceback` 내용도 같이 보냈습니다.

```
werkzeug.routing.BuildError
werkzeug.routing.BuildError: Could not build url
for endpoint 'answer.create' with values ['question_id'].
Did you mean 'main.index' instead?

Traceback (most recent call last)

File "[여러분의 가상환경 경로]\Lib\site-packages\flask\app.py",
line 2091, in __call__
    return self.wsgi_app(environ, start_response)
    :
    (중간 생략)
    :
File "[여러분의 프로젝트 경로]\hello_cju\views\question_views.py",
line 27, in detail
    return render_template(
    :
    (중간 생략)
    :

File "[여러분의 프로젝트 경로]\hello_cju\templates\question\question_detail.html", line 8, in top-level template code
    <form action="{{ url_for('answer.create', question_id=context.id) }}" method="post">
    :
    (중간 생략)
    :
```

에러 내용을 잘 파악할 수 있어야겠죠? `werkzeug.routing.BuildError` 에러는 `werkzeug` 패키지에서 `routing`을 지원하는 동안 발생하였습니다. 사용자가 서버에 입력한 경로, 즉 `url_for('answer.create', question_id=question.id)` 주소로 라우팅 하려고 했지만 실패 `BuildError` 했다는 의미입니다.

Note

`werkzeug`는 파일 `wsgi`를 제어하기 위한 라이브러리입니다. Flask는 `werkzeug`를 랩핑하여 다양한 기능을 제공합니다. `Werkzeug`에 대한 자세한 설명은 공식 문서 ([click](#))를 참고하기 바랍니다.

에러의 원인은 다음과 같습니다.

```
werkzeug.routing.BuildError:
Could not build url for endpoint 'answer.create'
with values ['question_id'].
Did you mean 'main.index' instead?
```

서버에서 앤드 포인트 `'answer.create'` 와 결합된 값 `values ['question_id']` 으로 구성된 URL을 찾을 수 없다는 의미입니다. Flask는 프로그래머가 혹시 `main.index`로 착각한거 아니니? 라고 묻고 있습니다.

Traceback을 찬찬히 살펴보면 이 에러는 다음 경로에 있는 파일의 8번째 라인에서 발생했다는 것을 알 수 있습니다.

```
File "[여러분의 프로젝트 경로]\hello_cju\templates\question\question_detail.html",
line 8, in top-level template code
  <form action="{{ url_for('answer.create', question_id=context.id) }}" method="post">
```

이상하죠? 우리는 코딩을 정확히 했는데 에러라니요...

그런데 곰곰히 생각해보면 Flask의 말이 맞습니다.

우리는 `url_for('answer.create', question_id=question.id)`에 인자로 전달한 `answer.create`를 정의한 적이 없습니다. `answer`라는 이름을 가진 블루프린트 객체를 정의하고 `answer`라는 이름을 가진 블루프린트 객체에 등록된 `create` 함수를 정의한 적이 없으니

Flask 입장에서는 이렇게 생각하게 될 것입니다. "뭐냐? 없는 경로를 찾아서 응답(response)를 달라고 하네? 이건 에러(error)야~~ 대신 친절하게 에러 메시지를 보내서 수정하도록 하자!!"

결국 우리는 브라우저 화면에서 Flask가 보내온 무시무시한 에러를 보게 된 것입니다.

이 오류를 해결하는 방법은 `view` 파일을 작성할 때 블루프린트로 URL 경로를 정의하고 해당 경로로 사용자 요청이 들어올 경우 처리할 함수 `create`를 작성하면 됩니다.

3.6.2. `view` 파일 `answer_views.py` 작성

`view` 파일 작성은 [Question List 조회](#)와 거의 동일한 방법으로 진행합니다.

먼저 `views` 디렉토리에 `answer_views.py` 파일을 만들고 다음과 같이 코딩해 줍니다.

```

from datetime import datetime
import imp
from flask import Blueprint, url_for, request
from werkzeug.utils import redirect
from hello_cju import db
from hello_cju.models import Question, Answer

# 블루프린트 객체 생성
bp = Blueprint(
    name='answer',
    import_name=__name__,
    url_prefix='/answer',
)

# 블루프린트 객체 bp에 라우팅 적용하고
# create 함수 등록
@bp.route(rule='/create/<int:question_id>', methods=['POST'])
def create(question_id):

    # 사용자로부터 제공받은 question_id를
    # 이용해 Question 객체 생성
    question = Question.query.get_or_404(question_id)

    # 사용자가 form에 담아 보낸 데이터 중에서
    # name이 "contents"인 데이터를 뽑아서
    # contents 변수에 저장
    contents = request.form['contents']

    # contents 변수에 저장된 데이터를 이용하여
    # Answer 객체 생성
    answer = Answer(contents=contents, create_date=datetime.now())

    # 다음과 같이 해도 동일하게 저장됩니다.
    # 아래 코드는 위 코드와 정확히 동일한 코드입니다.
    # answer = Answer(
    #     question=question,
    #     contents=contents,
    #     create_date=datetime.now()
    # )

    # Answer 클래스의 db.relationship 속성 중
    # backref를 이용하여 Question 클래스에서 등록한
    # 'answer_set'을 통해 현재 댓글을 해당 질문에 추가
    question.answer_set.append(answer)

    # SQLite DB에 등록
    db.session.commit()

    # 해당 정보를 url_for에서 제공하는
    # URL 경로로 리다이렉트
    return redirect(
        location=url_for(
            # 경로: /answer/create/question_id
            endpoint='question.detail',
            question_id=question_id,
        )
    )

```

위 코드 중 `create` 함수의 매개변수 값 `question_id`는 URL을 통해서 전달됩니다. 만약 `localhost:5000/answer/create/2`라는 URL로 Flask 서버에 서비스를 요청하면 `question_id` 값으로 2가 전달됩니다.

`question_detail.html`의 `form` 태그의 속성 `action="{} url_for('answer.create', question_id=context.id) {}"`에서 생성된 URL이 서버로 전달된 상황입니다.

라우팅 함수 `@bp.route` 인자 `method`에는 `POST`를 지정했습니다. 이렇게 지정한 이유는 `question_detail.html`의 `form` 태그의 속성 `method="post"`로 지정했기 때문입니다. HTTP를 동일하게 지정해 주어야 합니다. 템플릿과 `view`에서 지정한 형식이 다르면 에러가 발생합니다.

`methods=['POST']`를 지정할 때 반드시 '`POST`'를 리스트(`list`)나 튜플(`tuple`) 같은 리터러블 객체에 담아서 값을 전달해야 합니다. 이것은 Flask와 개발자간 약속이니 그냥 지키면 되겠습니다.

`question_detail.html`의 `form` 태그를 통해 Flask 서버로 전달한 데이터는 `request` 객체를 이용해 뽑아낼 수 있습니다. `request.form['contents']`는 `POST` 방식으로 Flask 서버에 전송된 데이터 중에서 이름 `name`이 `contents`인 데이터를 뽑아낼 경우에 사용합니다.

`question.answer_set`는 특정 질문에 달린 댓글 집합을 의미합니다. ORM 모델 `Answer`를 정의할 때 다음과 같은 코드를 사용했던 것을 기억해 보세요.

```
question = db.relationship(
    'Question',
    backref=db.backref('answer_set')
)
```

위 코드는 이미 ORM 모델을 만들때 우리가 작성했던 코드입니다. `Question` 모델에서 `Answer` 모델을 연결하고, `Question` 모델에서 `answer_set`이라는 이름을 가지고 `Answer`에 접근하게 됩니다. 댓글을 생성하고 이동할 경로는 `redirect` 함수를 이용해 처리할 경로를 알려주고 있습니다.

```
return redirect(
    location=url_for(
        # 경로: /answer/create/question_id
        endpoint='question_detail',
        question_id=question_id,
    )
)
```

이 코드는 `localhost:5000/question/detail/question_id` URL로 이동하라는 의미입니다. 처음 템플릿에서 질문 상세조회 페이지로 이동하라는 뜻입니다. 질문 상세 페이지에서 `form`을 이용해 전달받은 댓글 데이터를 저장한 후에 다시 원래의 질문 상세 페이지로 돌아간다는 의미가 됩니다.

❶ request 객체

Flask에서 기본으로 제공하는 `Request` 클래스의 객체입니다. `request` 객체는 [Werkzeug](#)에서 정의한 모든 속성과 Flask에서 추가로 정의한 속성값을 가지고 있습니다.

Flask는 브라우저의 요청(`request`)에서 서버의 응답(`response`) 구간에서 `request` 객체를 활용할 수 있도록 지원합니다. 우리는 `request` 객체를 이용해서 브라우저로부터 요청(`request`)과 관련한 다양한 정보를 뽑아 낼 수 있습니다.

우리가 코딩한 `request.form['contents']`는 `request`가 가지고 있는 수많은 속성값 중에서 `form` 데이터에 관한 것을 뽑아낸 것입니다.

`request` 객체에 대한 정보는 Flask 공식 문서([click](#))를 확인하기 바랍니다.

여기까지 하고 질문 목록 중 하나를 클릭하면 제대로 작동할까요?

네, 맞습니다. 여전히 작동하지 않습니다. 왜냐하면 블루프린트에 설정된 경로와 등록된 라우팅 함수가 `__init__.py`에 등록되지 않았기 때문에 Flask는 여전히 요청된 URL 경로에 따른 서비스를 제공할 수 없습니다.

이어서 `__init__.py`에 블루프린트 객체를 등록하여 이 문제를 해결해 보겠습니다.

3.6.3. `__init__.py`에 블루프린트 등록

우리는 블루프린트를 `__init__.py`에 등록하는 것은 몇 번 해보았습니다. `answer_views.py`에서 생성한 객체를 등록하려면 다음과 같이 코딩합니다.

```
# 앞 부분 생략
# 기존 __init__.py 내용과 동일

def create_app(): # 함수 생성
    # 중간 부분 생략
    # 기존 __init__.py 내용과 동일

    # answer_views 모듈을 추가로 임포트합니다.
    from .views import main_views, question_views, answer_views

    # Blueprint 객체 bp를 등록합니다.
    app.register_blueprint(main_views.bp)
    app.register_blueprint(question_views.bp)
    app.register_blueprint(answer_views) # answer_view 추가 등록

    return app
```

코딩이 끝나면 저장 후 서버를 실행합니다.

질문 리스트에서 특정 질문을 클릭하면 그림 [Fig.3.28](#) 같이 정상적으로 실행되는 것을 확인할 수 있습니다.

그림 [Fig.3.28](#)에서는 두 번째 질문을 클릭했습니다. 클릭하면 우리가 블루프린트에 등록한 URL을 이용해 Flask 서버로 요청을 하게 됩니다.

`form` 태그를 활용하여 코딩한 대로 텍스트 입력을 위한 `textarea`, 데이터 전송을 위한 `input`을 이용한 클릭 단추가 나타납니다.

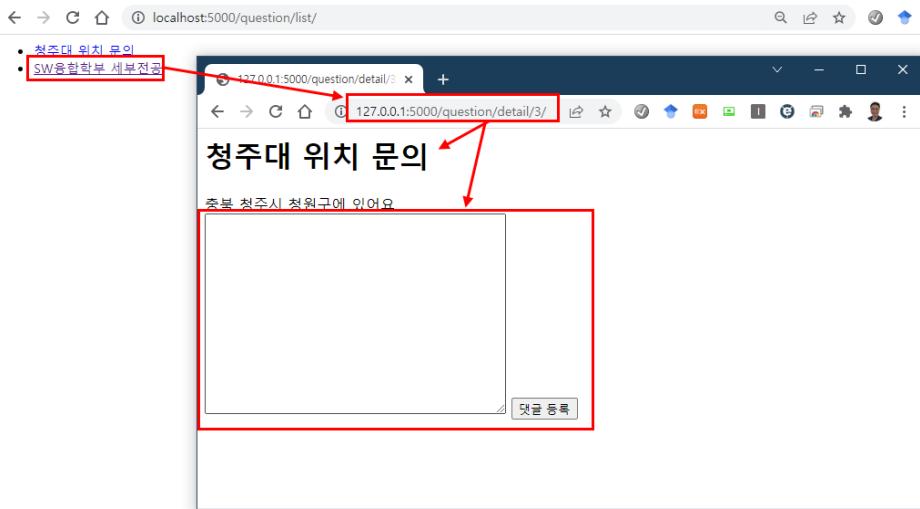


Fig. 3.28 `view` 함수, 블루프린트 등록 이후 정상 실행 화면

모든 것이 정상적으로 잘 실행되었습니다.

이번에는 댓글을 입력하면 입력된 댓글이 질문 밑부분에 표시되도록 해보겠습니다.

3.6.4. 댓글 표시를 위한 템플릿 파일 `question_detail.html` 수정

질문에 대한 댓글을 입력하면 바로 표시되게 하려면 어디를 손봐야 할까요?

네, 맞습니다.

바로 `question_detail.html` 파일을 약간만 손보면 됩니다.

템플릿 파일 `question_detail.html`을 아래와 같이 코딩해 줍니다.

```
<h1>{{ context.title }} </h1>
<div>
  {{ context.contents }}
</div>

<!-- 댓글을 뿌려주기 위해 추가한 부분 -->
<h5>{{ context.answer_set|length }}개의 댓글이 있습니다.</h5>
<div>
  <ul>
    {% for answer in context.answer_set %}
      <li>{{ answer.contents }}</li>
    {% endfor %}
  </ul>
</div>

<!-- 댓글 입력을 위해 이전에 추가했던 부분 -->
<form action="{{ url_for('answer.create', question_id=context.id) }}" method="post">
  <textarea name="contents" id="contents" cols="30" rows="10"></textarea>
  <input type="submit" value="댓글 등록">
</form>
```

위 코드에서 낯선 문법이 하나 있습니다. 바로 `{{ context.answer_set|length }}` 입니다.

| 표시는 필터 기능을 의미합니다. `context.answer_set`에 들어있는 원소의 개수를 반환합니다.

Flask 템플릿 필터

Flask는 `jinja2` 템플릿 엔진을 사용하기 때문에 `jinja2`에서 지원하는 모든 필터를 사용할 수 있습니다. 템플릿 필터는 매우 다양합니다. 일일히 거론하기 어렵습니다.

관심있는 사람은 `jinja2` 템플릿 필터 공식 문서 ([click](#))를 참고하여 필요한 것들을 활용하기 바랍니다.

파일을 저장하고 서버를 실행합니다. `textarea`에 댓글을 입력하고 **댓글 등록** 버튼을 누르면 그림 Fig. 3.29 와 같이 질문 아래 부분에 댓글이 달리는 것을 확인할 수 있습니다.

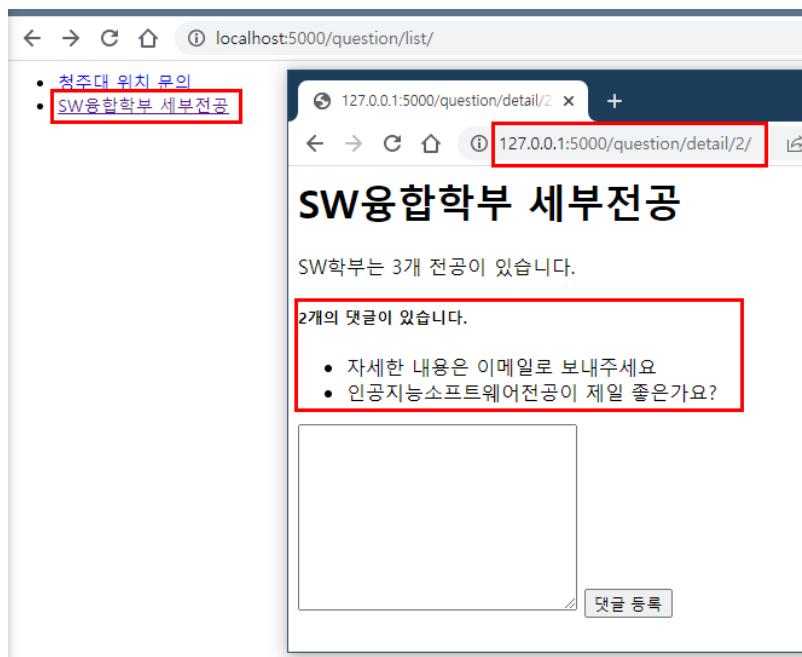


Fig. 3.29 입력한 댓글이 정상적으로 달리는 기능 완성

이제 우리는 `flask shell`을 이용해 등록한 질문 리스트를 조회하고, 개별 질문에 대한 상세 내용을 조회할 수 있게 되었습니다.

추가적으로 질문 상세 내용 페이지에서 댓글을 작성할 수 있도록 구현하였습니다.

이제 만족하시나요?

아마 뭔가 부족하다는 느낌이 들 겁니다.

그렇습니다. 일반적인 게시판이라면 질문 내용을 입력하고 등록할 수 있어야 하겠죠?

다음 절에서는 Flask에서 질문을 등록하는 방법에 대해 살펴보도록 하겠습니다.

3.7. 질문 & 댓글 등록 - Flask Form 활용

우리는 다음과 같이 기본기 6가지를 공부하였습니다.

❶ Flask 웹 시스템 구축을 위한 10가지 기본기

1. [프로젝트 기본 구조](#) \(\backslash\text{to}\) `Clear!`
2. [Application Factory 패턴](#) \(\backslash\text{to}\) `Clear!`
3. [Blueprint 클래스 활용](#) \(\backslash\text{to}\) `Clear!`
4. [ORM 모델 완벽 이해](#) \(\backslash\text{to}\) `Clear!`
5. [질문 게시판 만들기](#) \(\backslash\text{to}\) `Clear!`
6. [게시판 댓글 구현](#) \(\backslash\text{to}\) `Clear!`
7. [질문 & 댓글 등록 - Flask Form 활용](#) \(\backslash\text{to}\) 지금 도전!
8. [예쁘게 - CSS 적용](#)
9. [더 예쁘게 - Bootstrap 활용](#)
10. [HTML 구조와 Template 상속](#)

이제 많은 것을 배웠습니다. 많이 힘들기도 하고 어렵기도 했을 것입니다. 하지만 어느새 10가지 기본기 중에서 6가지를 완성했습니다. 이번 장까지가 핵심 기본기입니다. 조금만 더 집중하면 끝이 보이니, 좀 더 화이팅 하기 바랍니다.

우리는 질문/댓글 게시판을 만들었습니다.

그런데 등록된 질문은 `flask shell`을 이용하여 command 창에서 직접 입력했습니다. 자세한 내용은 우리가 이미 공부했던 [모델에 CRUD 해보기](#) 다시 한번 확인하세요.

하지만, 인터넷 사용자가 `flask shell`을 실행시켜서 검은 CLI 창에서 명령어를 입력하게 하는 것은 너무나도 가혹한 일입니다.

인터넷 사용자가 브라우저 상에서 질문을 입력하고 등록할 수 있도록 프로그래머가 도와줘야 합니다.

인터넷 사용자로부터 데이터를 전달받는 방법은 `form` 태그를 활용하는 것입니다. 우리는 [게시판 댓글 구현](#)에서 공부한 바 있습니다.

이미 공부했던 것처럼 `form`은 사용자에게 데이터 입력 양식을 편리하게 제공하기 위해서 사용하는 HTML 태그 중 하나입니다.

Flask는 `form`을 좀 더 편리하게 사용할 수 있도록 Flask 전용 `form`을 제공하고 있습니다. 참고마운 일입니다. 해당 모듈은 `Flask-WTF`라는 라이브러리를 설치하면 사용할 수 있습니다.

명령창에서 `pip install Flask-WTF`를 입력하여 필요한 라이브러리를 설치합니다.

```
(가상환경 이름) c:\여러분의 작업 경로>pip install Flask-WTF
Collecting Flask-WTF
  Downloading Flask_WTF-1.0.0-py3-none-any.whl (12 kB)
    :
    (중간 생략)
    :
Installing collected packages: WTForms, Flask-WTF
Successfully installed Flask-WTF-1.0.0 WTForms-3.0.1
```

위 코드에서 여러분의 가상환경에 `Flask-WTF`가 이미 설치되어 있다면 `Requirement already satisfied: ~~와 같은 메시지가 출력됩니다. 만약 설치되어 있지 않다면 pip가 자동으로 필요한 파일들을 설치해 줍니다.`

`Flask-WTF-1.0.0 WTForms-3.0.1`에서 보이는 숫자는 라이브러리 버전을 표시하는 것입니다. 여러분들이 설치하는 시기에 따라 숫자는 달라질 수 있습니다.

사용자가 데이터를 입력하고 서버로 전송하는 구간은 인터넷 네트워크입니다. 인터넷에 돌아다니는 패킷은 누구나 훔쳐볼 수 있고 조작할 수도 있습니다. 따라서 보안 이슈가 발생합니다. `form`을 통해 데이터를 전송할 때 생길 수 있는 보안 취약점은 인터넷 사용자의 요청(`request`)을 위조하는 `CSRF(Cross Site Request Forgery)`라는 공격이 있습니다.

CSRF 공격에 방어하기 위해 CSRF 토큰을 사용합니다. CSRF 토큰은 `form`을 이용해서 전송된 데이터가 실제 웹 사이트에서 정상적으로 작성된 데이터인지 판단하는 역할을 합니다. 이 경우 CSRF 토큰을 생성하기 위한 비밀키(`secret key`)가 필요합니다. `SECRET_KEY`는 `config.py` 파일에 등록해서 사용합니다.

VS code 상에서 `SECRET_KEY` 등록은 그림 Fig.3.30 같이 합니다.

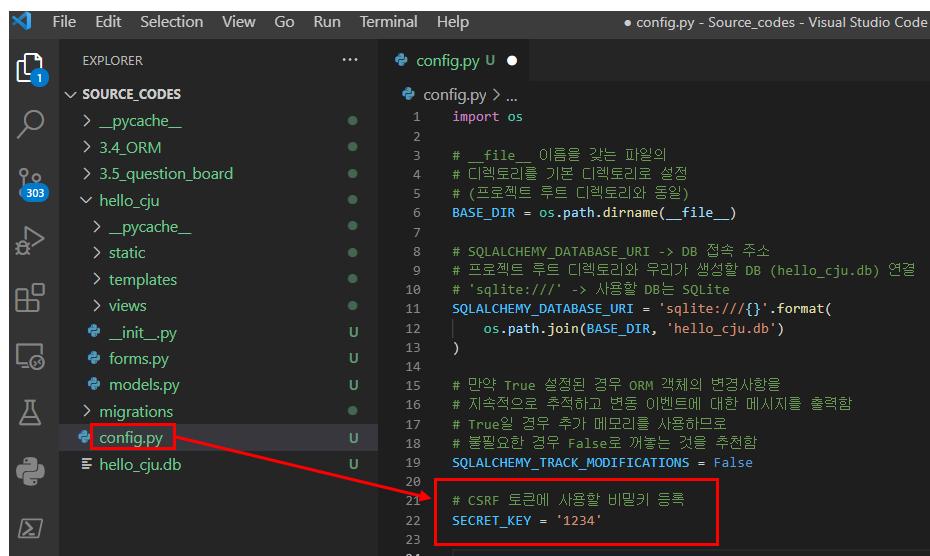


Fig. 3.30 config.py에 SECRET_KEY등록

Note

실제 서비스를 배포하는 단계에서는 절대로 그림 Fig.3.30 같이 `SECRET_KEY`를 공개해서는 안됩니다. 현재는 개발 단계이기 때문에 간단한 숫자 `1234`를 입력했습니다. `SECRET_KEY`를 보안성을 유지하면서 사용하는 방법은 고급 과정에서 배우게 됩니다.

3.7.1. 템플릿 파일 수정

이제 설치가 끝났으니 질문을 입력하기 위한 버튼을 만들어 볼까요?

질문 목록이 있는 페이지 `question_list.html` 아래쪽에 질문 등록하기 링크를 만듭니다.

```
# 파일경로: hello_cju/templates/question_detail.html
:
(앞부분 생략)
:
<!-- 질문 등록하기 위한 링크 추가 -->
<a href="{{ url_for('question.create') }} ">
    질문 등록하기
</a>
```

위 코드는 `질문 등록하기` 링크를 클릭하면 `question`이라는 이름을 가진 블루프린트 객체에 등록된 `create` 함수를 실행하라는 의미입니다.

현재 상태에서 Flask 서버를 작동시키면 `werkzeug.routing.BuildError` 에러가 납니다. View 파일 `views/question_views.py` 파일에 등록된 `create` 함수가 없기 때문입니다.

다음으로 할 작업은 3가지입니다.

- Flask-WTF에서 지원하는 `form`을 활용하여 질문 내용을 입력할 수 있는 질문 클래스를 만들고,
- 우리가 만든 `form` 클래스를 이용해서 `views/question_views.py` 파일에서 라우트 함수를 코딩(수정)해 줍니다.
- `view`에서 지정한 경로에 템플릿 파일 `.html` 코딩해 줍니다.

3.7.2. `form.py` 코딩

먼저 Flask에서 지원하는 `form` 클래스를 이용해서 우리가 사용할 질문 입력을 위한 클래스를 만들겠습니다. `forms.py`에 다음과 같이 코딩해 줍니다. 만약 파일이 없다면 새 파일을 만들고 이름을 `forms.py`로 지어줍니다.

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

class QuestionForm(FlaskForm):
    title = StringField(
        label='제목',
        validators=[DataRequired()],
    )
    contents = TextAreaField(
        label='내용',
        validators=[DataRequired()],
    )
```

위 코드에서 인터넷 사용자가 질문 내용을 입력하기 위한 `form`은 Flask-WTF 모듈의 `FlaskForm` 클래스를 상속 받아서 `QuestionForm` 이름을 갖는 클래스로 만들었습니다.

`QuestionForm` 클래스에는 2개의 속성 `title`, `contents`를 정의하였습니다.

- `title` 속성
 - 글자수 제한이 있는 `StringField` 클래스를 이용하여 객체를 생성하였습니다.
 - `title` 입력 양식에 나타날 이름 `label`은 `제목`으로 지정하였습니다.
 - `validators`는 입력 내용에 특정 조건을 지정하여 만족하는지 검사하는 방법을 지정합니다.
 - `DataRequired`의 경우 입력 내용이 없을 경우 에러를 발생시킵니다.
- `contents` 속성
 - 글자수 제한이 없는 `StringField` 클래스를 이용하여 객체를 생성하였습니다.
 - `title` 입력 양식에 나타날 이름 `label`은 `내용`으로 지정하였습니다.
 - `validators`는 입력 내용에 특정 조건을 지정하여 만족하는지 검사하는 방법을 지정합니다.
 - `DataRequired`의 경우 입력 내용이 없을 경우 에러를 발생시킵니다.

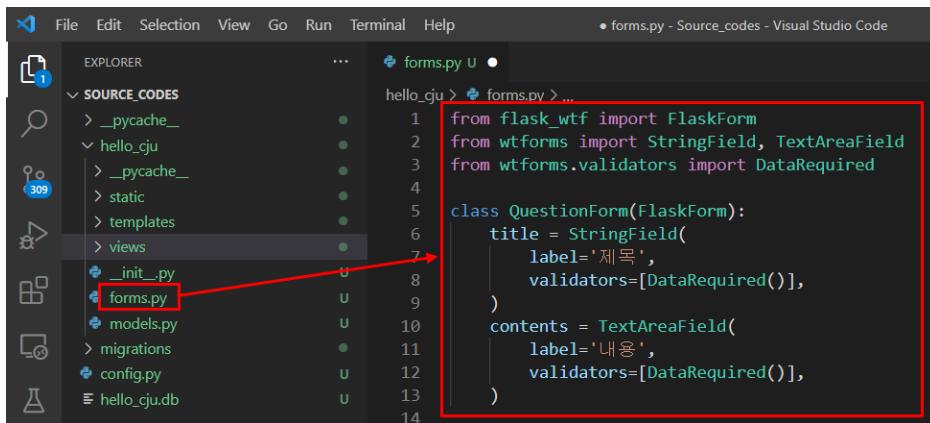
① Flask에서 활용할 수 있는 Field, validator

`form` 클래스 속성으로 지정할 수 있는 `Field`, 그리고 `Field` 인자으로 지정할 수 있는 `validator`는 다양합니다.

자세한 내용은 WTForms의 `validator` 공식 문서를 참고하기 바랍니다.

- `Field` 공식 문서 ([click](#))
- `validator` 공식 문서 ([click](#))

VS code 활용해서 입력한 결과는 그림 Fig. 3.31 와 같습니다.



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

class QuestionForm(FlaskForm):
    title = StringField(
        label='제목',
        validators=[DataRequired()],
    )
    contents = TextAreaField(
        label='내용',
        validators=[DataRequired()],
    )
```

Fig. 3.31 `forms.py` 코딩

3.7.3. 라우트 함수 코딩 - `question_views.py`

우리가 사용할 `QuestionForm`을 작성했으니 이를 이용해서 라우팅 경로를 설정해 주도록 하겠습니다. 다시 한번 말하지만 `view`는 템플릿 렌더링 하기 이전에 서버에서 처리할 작업을 정의하는 것입니다.

질문 목록이 있는 페이지 `question_list.html`에 정의한 코드 `질문 등록하기`에 의해 서버로 요청이 오면 Flask 서버는 블루프린트 이름 `name`이 `question`인 객체를 찾고, 그 블루프린트에 등록된 `create` 함수를 찾게 됩니다.

`create` 함수에 Flask 서버에서 할 일을 처리하고 리턴 값인 템플릿 경로를 찾아가게 됩니다.

아직까지 `question_list.html`에 `create` 함수가 없으므로 다음과 같이 코딩해 줍니다.

```
# 파일명: views/question_views.py

# QuestionForm 클래스를 임포트
from ..forms import QuestionForm

bp = Blueprint('question', __name__, url_prefix='/question')

@bp.route('/list/')
def question_list():
    # (...함수 내용 생략...)

@bp.route('/detail/<int:question_id>/')
def detail(question_id):
    # (...함수 내용 생략...)

# 질문 등록을 위해 추가한 코드입니다.
@bp.route('/create/')
def create():
    form = QuestionForm()
    return render_template(
        template_name_or_list='question/question_form.html',
        form=form
    )
```

위 코드에서 `question`이라는 `name`을 가진 블루프린트 객체 `bp`는 해당되는 객체에 등록된 `create`함수를 작동시킵니다. 그리고 결과는 `question/question_form.html`로 보냅니다. `question/question_form.html`가 렌더링한 결과는 `localhost:5000/question/create` 경로를 이용해 인터넷 사용자에게 전달(`response`)됩니다.

`create` 함수는 우리가 만든 `QuestionForm` 클래스를 이용해 객체 `form` 생성하고, 객체 `form`을 포함해서 템플릿으로 보냅니다. `render_template` 함수에서 `form=form` 의미는 template에 `form` 객체를 전달하게 되는데, `form`이라는 이름으로 참조하라는 의미입니다.

참고로 우리가 만든 `QuestionForm` 클래스를 사용하기 위해서는 임포트 `from ..forms import QuestionForm` 해주어야 에러가 발생하지 않습니다.

Flask 서버를 실행하고 `localhost:5000`으로 접속해 봅니다.

VS code 활용해서 입력한 결과는 그림 Fig. 3.32 와 같습니다.

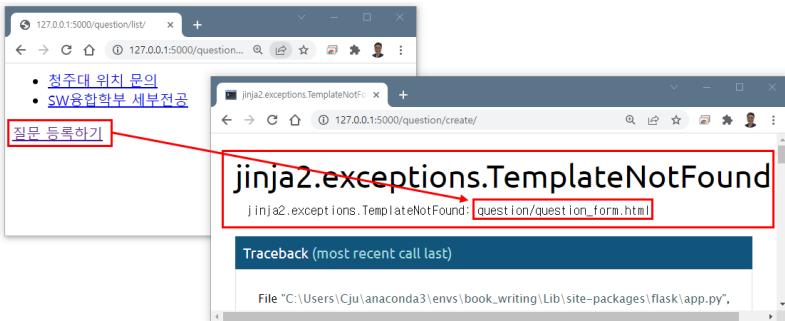


Fig. 3.32 view에서 라우트 함수를 코딩한 이후 화면

그림 Fig. 3.32 왼쪽과 같이 `질문 등록하기`라는 링크가 잘 생성된 것을 확인할 수 있습니다. 하지만 `질문 등록하기` 클릭하면 `TemplateNotFound` 에러가 발생합니다. 에러 내용 `jinja2.exceptions.TemplateNotFound: question/question_form.html` 살펴보니 템플릿 파일 `question/question_form.html` 없어서 Flask는 이러한 저리지도 못하는 상황입니다.

Flask 입장에서는 `create` 함수에서 `question/question_form.html`로 가라고 했는데 막상 가야할 장소가 없는 형국입니다. 우리가 아직은 템플릿 파일 `question/question_form.html` 코딩하지 않았으니 당연한 결과겠죠?

우리는 `create` 함수에서 지정한 위치와 이름으로 템플릿 파일 `.html`을 만들어 주면 됩니다.

3.7.4. 질문 입력받기 위한 템플릿 코딩 - `question_form.html`

`templates/question/` 위치에 `question_form.html`이라는 이름으로 파일을 하나 생성합니다. 그리고 아래와 같이 코딩해 줍니다.

```
<div>
  <h5>질문 등록</h5>
  <form action="{{ url_for('question.create') }}" method="post">
    {{ form.csrf_token }}
    {{ form.title.label()}}
    <br>
    {{ form.title()}}
    <br>
    {{ form.contents.label }}
    <br>
    {{ form.contents() }}
    <br>
    <button type="submit">저장하기</button>
  </form>
</div>
```

위 코드는 `view`로부터 전달받은 `form`이라는 이름을 갖는 context 객체를 활용하여 브라우저에 그려줄 내용을 렌더링 하는 `.html` 코드입니다. `저장하기` 버튼을 누르면 `question`이라는 이름을 가진 블루프린트 객체에 등록된 `create` 함수를 실행하도록 코딩하였습니다.

데이터를 전송할 `form`은 HTML의 `POST` 방식을 이용해 처리할 수 있도록 `method="post"`로 코딩해 주었습니다.

`{{ form.csrf_token }}` CSRF 공격에 대응하기 위한 토큰을 자동으로 생성해주는 코드입니다. 만약 이 코드가 없다면 `view`에서 `validate_on_submit()` 함수를 이용해 전달받은 데이터를 검사할 때 `False` 값을 얻게 됩니다.

참고로 `div` 태그는 공간영역을 구분할 때, `br` 태그는 줄바꿈, `h5`태그는 제목 수준 5를 표현, `button` 태그는 입력 버튼을 만들어주는 HTML 태그입니다. 자세한 내용은 다음 모질라 공식 문서를 참고하기 바랍니다.

❶ HTML 요소(태그)

위 코드에서 사용한 요소(태그)에 대한 설명은 다음 공식 문서를 참고하기 바랍니다.

- `h` 태그 ([click](#))
- `div` 태그 ([click](#))
- `button` 태그 ([click](#))
- `br` 태그 ([click](#))

HTML 요소 전체 목록 참고서

- W3C Schools HTML Element Reference ([click](#))
- 모질라 HTML 요소 참고서 ([click](#))

❷ Note

HTML 요소(태그) 중에서 `input`과 `button`은 비슷한 것 같으면서도 차이가 있습니다. 우리는 [개시판 댓글 구현](#)에서 템플릿 파일 `question_detail.html`을 코딩할 때 `input` 요소(태그)를 사용했지만, 위에서 작성한 `question_form.html` 코드는 `button` 요소(태그)를 사용했습니다. 유사점 및 차이점을 설명한 블로그를 참고하기 바랍니다.

- 참고블로그 1. 기분따라 코딩 ([click](#))
- 참고블로그 2. 클로시션 작은 공간 ([click](#))

서버를 실행하고 `localhost:5000`에 접속해 봅니다.

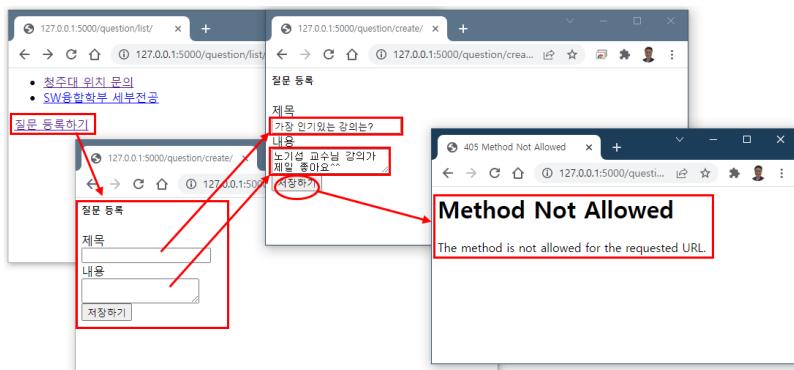


Fig. 3.33 질문 등록을 위한 `form`을 코딩한 이후 화면 변화

그림 Fig. 3.33에서 `질문 등록하기`를 클릭해 봅니다. `question_form.html`에서 코딩한 대로 입력 화면이 나타납니다. 입력창에 텍스트(글자)를 입력하고 `저장하기` 버튼을 누릅니다.

앗! 정상적으로 작동할 줄 알았는데... `Method Not Allowed` 에러가 떽습니다.

원인이 무엇일까요?

템플릿에서 지정한 HTML 통신 방식과 블루프린트 객체에 등록한 라우트 함수의 통신 방법이 맞지 않았기 때문입니다.

템플릿 파일 `question_form.html`에는 다음과 같이 코딩해 줬습니다.

```
<form action="{{ url_for('question.create') }}" method="post">
```

우리는 위 코딩을 통해 서버로 `form` 영역의 데이터를 전송할 때 `post` 방식을 사용하라고 지정해 했습니다.

하지만 `view` 파일 `question_views.py`에는 다음과 같이 코딩 했습니다.

```

@bp.route('/create/')
def create():
    form = QuestionForm()
    return render_template(
        template_name_or_list='question/question_form.html',
        form=form
    )

```

`@bp.route('/create/')` 코드에는 별도로 전송 방식을 지정하지 않았습니다. 별도 지정이 없을 경우 블루프린트 객체는 `GET` 방식을 사용하도록 설정되어 있습니다.

Flask는 이렇게 생각 했을 겁니다.

"음... 인터넷 사용자가 `post` 전송방식으로 나한테 데이터를 보냈군. 하지만 나는 별도 설정이 없으니 `GET` 방식을 사용하라고 코딩되었네? 엉! 그러면 통신 방식이 다르잖아! 그렇다면 사용자나 개발자에게 에러를 내보내서 수정하도록 하는게 좋겠군."

그래서 나온 에러가 바로 `Method Not Allowed` 입니다.

이 에러는 블루프린트 객체에 통식 방식을 결정하는 인자 `methods`를 `POST`로 지정해 주면 쉽게 해결됩니다.

위 코드 중에서 블루프린트 라우트함수에 대한 내용을 다음과 같이 수정합니다.

```

@bp.route('/create/') \(\to\) @bp.route('/create/', methods=('POST', 'GET'))

```

코드를 수정해주고 다시 `저장하기` 버튼을 클릭하면 더 이상 에러가 발생하지 않습니다.

그런데 좀 이상합니다. `Method Not Allowed` 에러는 해결했는데, 막상 아무일도 일어나지 않습니다. 네... 맞습니다. 아무일도 일어나지 않는 것이 정상입니다. 왜냐하면 `form` 데이터를 전송했지만, 데이터를 전송받은 Flask 서버에서 할 일을 정해주지 않았기 때문입니다.

템플릿을 실행하기 전에 서버에서 할 일을 정의하는 모듈... 기억 나시죠? 바로 `view`에서 처리해 주어야 합니다. 이를 처리하기 위해 `form` 데이터를 받는 `question_views.py`를 업그레이드 하겠습니다.

`question_views.py` 파일을 열어서 아래와 같이 코딩해 줍니다.

```

# form으로부터 받은 데이터를 처리하기 위한 모듈 추가 임포트
from datetime import datetime
from flask import request, url_for
from werkzeug.utils import redirect
from .. import db

# ...중간 생략...

@bp.route('/create/', methods=('GET', 'POST'))
def create():
    form = QuestionForm()

    # form으로부터 받은 데이터를 처리하는 코드
    if request.method == 'POST' and form.validate_on_submit():
        # if request.method == 'POST':
        print('POST')
        question = Question(
            title=form.title.data,
            contents=form.contents.data,
            create_date=datetime.now(),
        )

        db.session.add(question)
        db.session.commit()

        return redirect(url_for('question.question_list'))

    return render_template(
        template_name_or_list='question/question_form.html',
        form=form
    )

```

VS code를 사용한다면 그림 [Fig. 3.34](#) 같은 형태로 코딩하면 됩니다.

현재 `/question/create/` 경로로 요청이 들어올 수 있는 경우는 2가지입니다.

그림 [Fig. 3.35](#)를 살펴 볼까요?

그림 [Fig. 3.35](#)의 왼쪽 화면에서 `질문 등록하기` 버튼을 누르면 `/question/create/`로 요청이 전송됩니다. 왜냐하면 질문 등록 링크를 아래와 같이 코딩했었기

```

hello_cju > views > question_views.py > ...
1 # 파일명: views/question_views.py
2
3 # Blueprint 클래스를 임포트 합니다.
4 import flask
5 from flask import Blueprint
6 from flask import render_template
7 from hello_cju.models import Question
8 from ..forms import QuestionForm
9
10 # form으로부터 받은 데이터를 처리하기 위한 모듈 추가 임포트
11 from datetime import datetime
12 from flask import request, url_for
13 from werkzeug.utils import redirect
14 from .. import db
15
16 bp = Blueprint('question', __name__, url_prefix='/question')
17
18 # Blueprint 객체 bp를 이용하여 함수와 URL을 매칭합니다.
19 @bp.route('/list/')
20 def question_list():
21     return render_template(
22         'question/question_list.html',
23         questions=Question.query.all()
24     )
25
26 @bp.route('/detail//')
27 def detail(question_id):
28     question = Question.query.get_or_404(question_id)
29
30     return render_template(
31         'question/question_detail.html',
32         question=question
33     )
34
35 @bp.route('/create/', methods=['GET', 'POST'])
36 def create():
37     form = QuestionForm()
38
39     # form으로부터 받은 데이터를 처리하는 코드
40     if request.method == 'POST' and form.validate_on_submit():
41         # if request.method == 'POST':
42             print('POST')
43             question = Question(
44                 title=form.title.data,
45                 contents=form.contents.data,
46                 create_date=datetime.now(),
47             )
48
49             db.session.add(question)
50             db.session.commit()
51
52             return redirect(url_for('question.question_list'))
53
54     return render_template(
55         'question/question_form.html',
56         form=form
57     )
58

```

Fig. 3.34 동일한 /question/create/ 경로로 서버 요청(request)

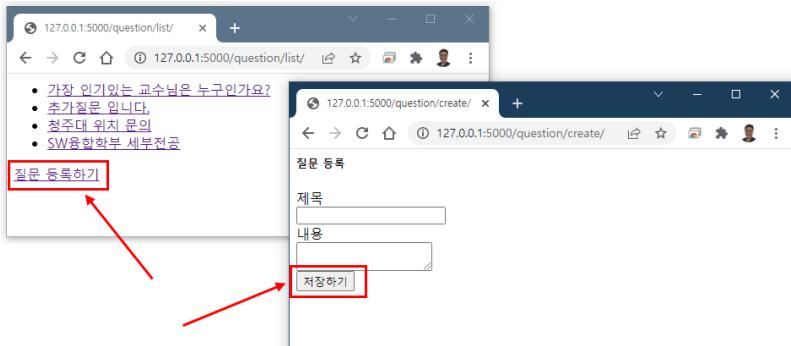


Fig. 3.35 동일한 /question/create/ 경로로 서버 요청(request)

때문입니다.

```

<!-- 질문 등록하기 위한 링크 추가 -->
<a href="{{ url_for('question.create') }}">
    질문 등록하기
</a>

```

그림 Fig. 3.35의 오른쪽 화면에서 **저장하기** 버튼을 누르면 **질문 등록하기**와 마찬가지로 /question/create/로 요청이 전송됩니다. 왜냐하면 **form**을 아래와 같이 정의했기 때문입니다.

```

<!-- 질문 저장하기에 필요한 데이터를 전송하기 위한 form -->
<form action="{{ url_for('question.create') }}" method="post">

```

하나의 경로 /question/create/에 서로 다른 기능을 요청하고 있습니다. 이 경우 기능에 따라 구분을 해줘야 겠죠?

질문 등록하기 링크 속성 **href**=는 별다른 HTML 전송 방식을 지정하지 않았기 때문에 **GET** 방식으로 전송됩니다.

저장하기는 명시적으로 데이터 전송 방식을 **method="post"**으로 지정했기 때문에 **POST** 방식으로 전송됩니다. 이 차이점을 구분하여 구현하기 위하여 **if request.method == 'POST'** 코드를 작성하였습니다.

위 코드에서 **POST** 방식으로 데이터를 받았다면 **Question** 객체를 생성하여 DB에 저장한 후에 질문 리스트를 업데이트 하여 보여주도록 **return redirect(url_for('question.question_list'))** 코딩하였습니다.

`GET` 방식으로 데이터를 전송 받았다면 [질문 등록하기](#) 버튼을 클릭한 것이므로 `question_form.html`을 렌더링 하도록 아래와 같이 기존 코드를 유지하였습니다.

```
return render_template(  
    template_name_or_list='question/question_form.html',  
    form=form  
)
```

`form.validate_on_submit()`은 인터넷 사용자가 `form` 지정한 전송방식이 `POST`를 사용하였는지 검사합니다. 또한 `form`에서 지정한 `validators` 요구사항을 정확히 지켰는지를 검사합니다. 전송받은 `form` 데이터가 정상이면 `True`, 이상이 있는 경우 `False`를 리턴합니다.

3.7.5. 수작업으로 `form` 작성해 보기

우리는 Flask가 지원하는 `form` 객체를 통해 `form.title()`, `form.contents()`와 같이 편리한 방식으로 데이터 입력을 위한 입력창을 생성하였습니다.

하지만 이 방식은 빠르게 `form`을 완성할 수 있지만 HTML 요소(태그) 또는 요소의 속성을 추가하기 어렵습니다. 이는 우리가 원하는 웹사이트 디자인을 구현하기 어렵게 하고, 부가적인 기능을 구현하기 어렵게 만듭니다.

이미 만들었던 `question_form.html` 파일을 수정하여 동일한 기능을 갖는 `form`을 만들어 보겠습니다.

이전 코드를 주석(또는 삭제)처리하고 아래와 같이 새로운 코드를 작성합니다.

```
<!-- form 지원 기능 활용 -->  
<!-- <div>  
    <h5>질문 등록</h5>  
    <form action="{{ url_for('question.create') }}" method="post">  
        {{ form.csrf_token }}  
        {{ form.title.label()}}  
        <br>  
        {{ form.title()}}  
        <br>  
        {{ form.contents.label }}  
        <br>  
        {{ form.contents() }}  
        <br>  
        <button type="submit">저장하기</button>  
    </form>  
</div> -->  
  
<!-- 수기로 만든 form -->  
<div>  
    <h5>질문 등록</h5>  
    <form action="{{ url_for('question.create') }}" method="post">  
        <div>  
            <label for="title">제목</label>  
            <input type="text" name="title" id="title">  
        </div>  
  
        <div>  
            <label for="contents">내용</label>  
            <textarea name="contents" id="contents" cols="30" rows="10"></textarea>  
        </div>  
  
        <button type="submit">저장하기</button>  
    </form>  
</div>
```

위 코드 중 상단에 주석 처리된 영역은 우리가 이전에 `form` 지원 기능을 활용하여 빠르게 개발했던 코드입니다. 아래쪽은 지원기능을 사용하지 않고 직접 HTML 요소(태그) `input`, `textarea`를 활용하여 작성한 코드입니다. 이전 영역의 코드는 삭제해도 무방합니다.

그림 [Fig. 3.36](#)에서 위쪽 그림은 이전에 우리가 처음에 코딩한 것을 렌더링한 화면이고 아래쪽 그림은 수기로 작성한 코드와 `form`을 적용한 화면입니다. 약간 차이는 있지만 데이터를 입력하고 전송하는 역할은 동일합니다.

어떤 코딩 스타일을 선택할 것인지는 개발자의 성향이나 개발하고 있는 프로젝트 특성에 맞게 선택하면 됩니다.

모든 기능이 완성되었습니다.

이제 질문을 올려 보겠습니다.

그림 [Fig. 3.37](#)에서 볼 수 있듯 질문 목록 페이지에서 [질문 등록하기](#)를 클릭하면 우리가 `form`을 이용하여 작성한 입력화면이 제공됩니다. 제목, 내용 입력창에 원하는 글을 입력하고 [등록하기](#) 클릭합니다.

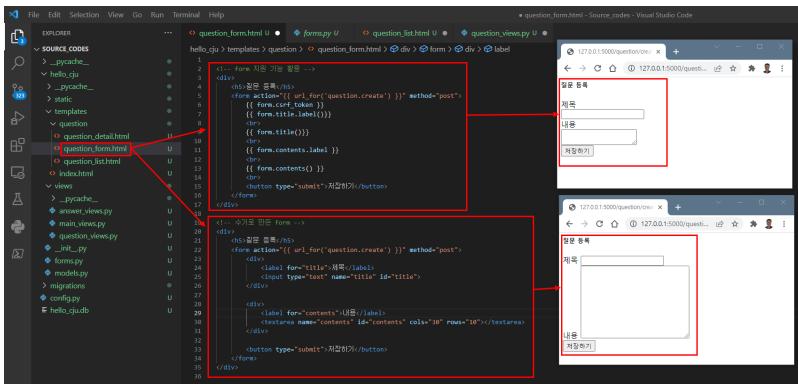


Fig. 3.36 수기로 작성한 question_form.html 비교

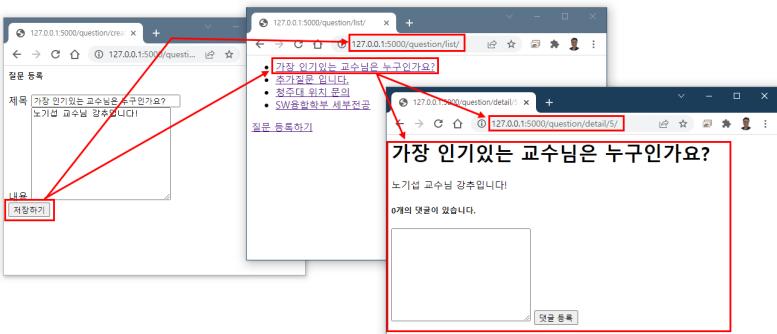


Fig. 3.37 수기로 작성한 form을 활용하여 댓글 올리기

등록하기를 클릭하는 순간 `form`에 저장된 데이터는 Flask 서버로 전송됩니다. 이때 전송 받은 데이터는 `question_form.html` 파일의 `form` 태그 속성 `action={{ url_for('question.create') }}`에서 지정한 곳에서 처리합니다. 우리 코드의 경우 `<form action="{{ url_for('question.create') }}" method="post">`으로 설정하였습니다.

`view` 파일 중에서 블루프린트 객체의 이름이 `question`이고, 그 객체에 등록된 `create` 이름을 가진 함수에서 데이터를 처리합니다. 우리 경우는 `views/question_views.py`에 있는 `def create():` 함수에서 사용자 데이터를 처리합니다.

`def create():` 함수는 `POST` 방식으로 데이터를 수신한 경우 데이터를 SQLite DB에 저장한 후에 다음 페이지를 질문 목록 페이지로 지정하여 웹페이지를 렌더링합니다. 우리의 경우 질문 목록 페이지를 `redirect(url_for('question.question_list'))`로 코딩해 줬습니다.

질문 목록을 렌더링하는 템플릿 `question_list.html` 파일은 DB에 저장된 질문 목록을 전달받아 브라우저 화면에 뿐려줍니다. 그 결과 우리가 조금 전입력한 질문을 포함한 목록이 브라우저 화면에 렌더링 됩니다. 이 과정의 결과는 그림 Fig.3.37 중간에 질문 목록을 보여주는 브라우저 입니다.

특정 질문 목록을 클릭하면 해당 질문에 대한 세부 내용을 조회할 수 있습니다. 템플릿 `question_list.html` 파일에서 각 질문에 대한 링크를 ``와 같이 설정하였으므로 특정 질문을 클릭하면, Flask 서버에 요청(`request`)을 보내게 됩니다.

Flask 서버에서는 요청을 받아서 블루프린트 객체 중 `question`이라는 이름을 갖는 객체를 찾고 그 객체에 등록된 `detail`이라는 함수에서 서버가 할 작업을 처리합니다. 세부 내용은 `question_views.py`에 코딩된 `def detail(question_id):...` 함수 내용을 참고하세요. 그림 Fig.3.37 오른쪽에 결과 화면을 볼 수 있습니다.

3.7.6. 오류 확인 및 표시

`form`을 이용해 데이터를 전송할 때 다양한 형태의 오류가 발생할 수 있습니다. 예를 들어 `validators` 중에서 `DataRequired()`를 적용했다면 해당하는 입력창에 반드시 데이터가 입력되어야 합니다. 그 밖에 이메일인지 점검하는 `Email()`, 입력 문장의 길이를 제한하는 `Length()` 등이 가능합니다.

그런데 `저장하기` 버튼을 눌렀는데 아무런 일도 발생하지 않는다면 무엇이 잘못되었는지 파악하기 어렵습니다. 이 경우 유용하게 사용할 수 있는 것은 `form`에 내장된 기능인 `.errors`를 활용하는 것입니다.

만약 `view` 함수 `create`에 있는 `form.validate_on_submit()`에서 실패(`False`)를 반환하면 템플릿 `question_form.html`에 발생된 오류를 전달하게 됩니다. 이 기능을 활용하기 위해 `question_form.html`에 다음과 같이 추가 코딩을 해줍니다.

```

<!-- 수기로 만든 form -->
<div>
    <!-- 오류 내용을 표시하는 코드 추가-->
    <form method="post">
        {{ form.csrf_token }}
        {% for field, errors in form.errors.items() %}
            <div role="alert">
                {{ form[field].label }}: {{ ', '.join(errors) }}
            </div>
        {% endfor %}
    </form>

    <h5>질문 등록</h5>
    <form action="{{ url_for('question.create') }}" method="post">
        {{ form.csrf_token }}
        <div>
            <label for="title">제목</label>
            <input type="text" name="title" id="title" size="30">
        </div>

        <div>
            <label for="contents">내용</label>
            <textarea name="contents" id="contents" cols="30" rows="10"></textarea>
        </div>

        <button type="submit">저장하기</button>
    </form>
</div>

```

위 코드는 수기로 만든 코드에서 `form` 작성에 오류가 있을 경우 오류 항목을 표시하도록 한 코드입니다.

VS code에서 작성한 결과는 그림 Fig. 3.38 을 참고하기 바랍니다.

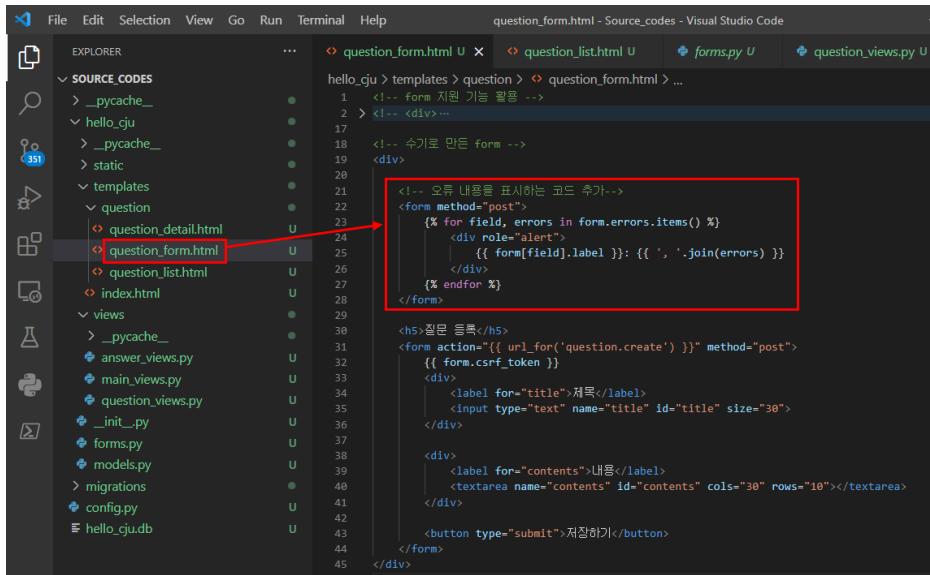


Fig. 3.38 템플릿 파일 `question_form.html`에 에러 처리 코드를 추가

`form` 입력창에 적절한 데이터가 `validators` 검증 조건을 충족하지 못하거나 `csrf_token` 오류가 있는 경우 오류를 브라우저에 뿐려줍니다.

그림 Fig. 3.39는 내용을 입력하지 않았기 때문에 데이터가 반드시 입력되어야 한다는 `DataRequired()` 조건을 충족하지 못하고 에러가 발생한 경우입니다. 아래 그림은 제목과 내용을 모두 입력하지 않았기 때문에 에러가 2개 표시된 상황입니다.

에러 메시지는 기본적으로 영어로 설정되어 있습니다. 영어에 친숙하지 않은 인터넷 사용자를 위해 한글로 표현해 주면 더욱 좋을 것 같습니다.

`form`을 정의했던 `forms.py`에서 설정한 `validators` 여러분이 보여줄 메시지를 입력해 주면 간단하게 해결됩니다. `forms.py` 파일을 아래와 같이 수정합니다.

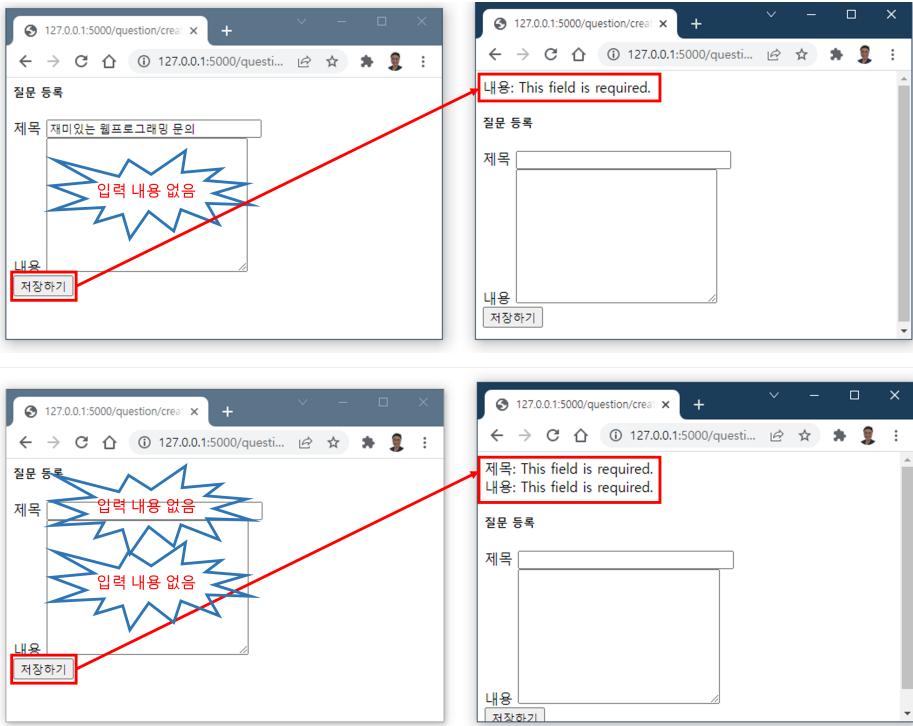


Fig. 3.39 템플릿 파일 question_form.html에 에러 처리 코드를 추가

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

class QuestionForm(FlaskForm):
    title = StringField(
        label='제목',
        # form 검증 실패 시 보여줄 메시지 포함
        validators=[DataRequired('제목은 필수 입력 항목입니다.')]
    )
    contents = TextAreaField(
        label='내용',
        # form 검증 실패 시 보여줄 메시지 포함
        validators=[DataRequired('내용은 필수 입력 항목입니다.')]
    )
```

코드를 위와 같이 수정하고 다시 입력 에러를 발생시키면 그림 Fig. 3.40와 같이 한글로 에러가 표시되는 것을 확인할 수 있습니다.

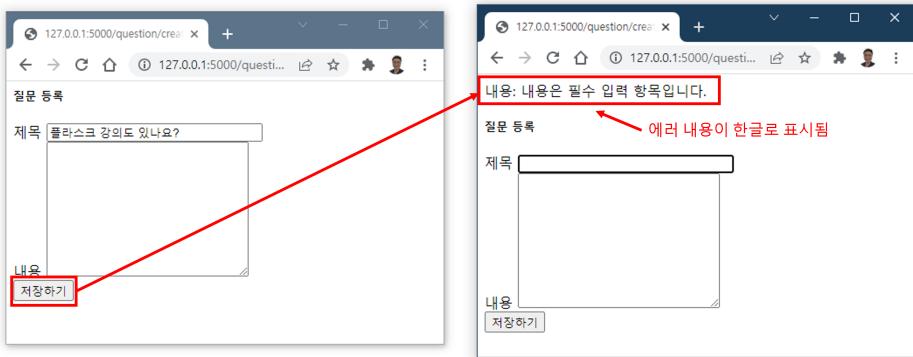


Fig. 3.40 form 에러를 한글로 출력

3.7.7. 에러 발생 시 form 내용 유지

온라인에서 회원가입이나 물건을 주문할 때 다양한 정보를 입력해야 하는 경우가 있습니다. 이런 저런 데이터를 `form` 양식에 입력하고 제출하기를 눌렀는데 특정 필드값 입력이 잘못되어 다시 입력해야 하는 경우가 생깁니다.

사용자가 입력한 데이터가 정확한지 확인하는 검증(validation) 과정을 통과하지 못했기 때문입니다. 우리가 구현한 시스템에서도 동일한 경우가 발생합니다. 그림 Fig. 3.41을 살펴 볼까요?

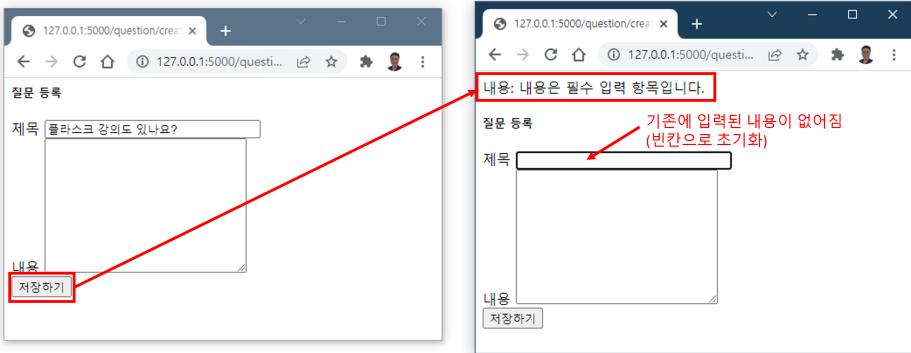


Fig. 3.41 `form` 에러를 한글로 출력

그림 Fig. 3.41에서 제목에 "재미있는 웹 프로그래밍 문의"라는 글자를 입력했지만 내용에 글자를 입력하지 않고 저장하기를 클릭했습니다. 내용을 작성해야 한다는 에러 메시지가 표시되고 나머지 모든 값들은 빈칸으로 초기화 되었습니다.

입력해야 하는 내용이 짧고 간단하다면 별 문제가 없습니다. 하지만 힘들게 이것 저것 입력한 내용이 많았는데 모두 날아가 버리면 참 허무하겠죠? 사용자 친화적이지 않습니다.

이런 현상은 `form` 입력값이 없을 경우 `None`으로 인식하게 됩니다. Flask 서버는 `form`에 입력된 필드 값 중 하나라도 없으면 에러를 발생시키고 에러 내용을 `form`에 내부적으로 담아서 다시 템플릿 파일 `question_form.html`로 보내주게 됩니다.

하지만 아래와 같이 코딩해 주면 기존에 입력한 데이터는 그대로 유지하면서, 데이터를 입력하지 않은 필드에 대해서만 에러 메시지를 출력할 수 있습니다.

```
<div>
    <!-- 오류 내용을 표시하는 코드 추가-->
    <form method="post">
        {% for field, errors in form.errors.items() %}
            <div role="alert">
                {{ form[field].label }}: {{ ', '.join(errors) }}
            </div>
        {% endfor %}
    </form>

    <h5>질문 등록</h5>
    <form action="{{ url_for('question.create') }}" method="post">
        {{ form.csrf_token }}
        <div>
            <label for="title">제목</label>
            <!-- 에러가 발생해도 기존 내용 유지하도록 value 속성 추가 -->
            <input type="text" name="title" id="title" size="30" value="{{ form.title.data or '' }}>
        </div>

        <div>
            <label for="contents">내용</label>
            <!-- 에러가 발생해도 기존 내용 유지하도록 value 속성 추가 -->
            <textarea name="contents" id="contents" cols="30" rows="10" value="{{ form.contents.data or '' }}></textarea>
        </div>

        <button type="submit">저장하기</button>
    </form>
</div>
```

위 코드를 `question_form.html` 파일에 적용하여 업데이트하고 다시 내용을 입력해 보면 그림 Fig. 3.42와 같이 기존 내용이 잘 유지되는 것을 확인할 수 있습니다.

`input` 요소(태그)에 속성값 `value="{{ form.title.data or '' }}"`을 약간 변경했을 뿐인데 왜 이런 현상이 발생할까요?

개발자라면 끝까지 원인을 알아내야 직성이 풀리겠죠?

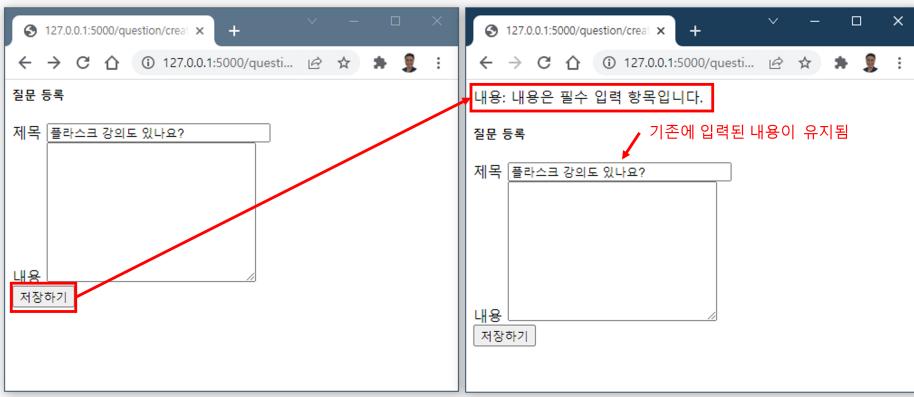


Fig. 3.42 form에 입력한 데이터는 그대로 유지

가장 빠르고 정확하게 답을 찾는 방법은 공식 문서를 확인하는 방법입니다.

내친김에 공식문서를 보면서 원인을 파악하는 방법을 설명하겠습니다. 여러분들도 이런 방법을 통해 근본적인 작동 방식을 알아가는 법을 배우시기 바랍니다.

Flask는 사용자로부터 form을 사용하기 위해 `WTForms` 패키지를 사용합니다. 우리가 코딩한 `forms.py` 모듈의 `import` 부분을 잘 살펴보세요. 그리고 `form`에 입력된 데이터를 검증하기 위해 `DataRequired`라는 빌트인(Built-in, 사전에 만들어서 제공하는) 클래스를 임포트 해서 사용했습니다.

구글링을 해보면 WTForms에서 제공하는 `Built-in validators` 페이지([click](#))를 찾을 수 있습니다.

그림 Fig. 3.43에 `Built-in validators` 페이지([click](#))를 가져왔습니다. 참고로 보세요.

Built-in validators

`class wtforms.validators.DataRequired(message=None)`

Checks the field's data is 'truthy' otherwise stops the validation chain.

This validator checks that the `data` attribute on the field is a 'true' value (effectively, it does `if field.data`). Furthermore, if the data is a string type, a string containing only whitespace characters is considered false.

If the data is empty, also removes prior errors (such as processing errors) from the field.

NOTE this validator used to be called `Required` but the way it behaved (requiring coerced data, not input data) meant it functioned in a way which was not symmetric to the `Optional` validator and furthermore caused confusion with certain fields which coerced data to 'falsy' values like `0`, `Decimal(0)`, `time(0)` etc. Unless a very specific reason exists, we recommend using the `InputRequired` instead.

Parameters: `message` – Error message to raise in case of a validation error.

Sets the `required` attribute on widgets.

This also sets the `required` flag on fields it is used on. This flag causes the `required` attribute to be rendered in the tag, which prevents a request/response cycle for validation. This behavior can be overridden in the following ways:

- Specifying `required=False` when rendering in the template.
- Making a custom widget that doesn't set it.
- Rendering the `novalidate` attribute on the `form` tag, or the `formnovalidate` attribute on a submit button.

The required flag behavior also applies to the `InputRequired` class.

:

:

Fig. 3.43 Built-in validators 공식 문서

그림 Fig. 3.43에 `Built-in validators` 페이지 주소 <https://wtforms.readthedocs.io/en/2.3.x/validators/#built-inValidators>로 접속한 화면입니다.

`DataRequired` 클래스의 구조와 기능에 대하여 자세히 설명되어 있습니다. 그림의 우측 상단에 `[source]`라는 링크 ([click](#))를 클릭해서 들어가면 `DataRequired` 클래스가 어떻게 코딩(구현)되어 있는지 확인할 수 있습니다.

공식 문서에서 제공하는 `DataRequired` 클래스 소스코드는 다음과 같습니다.

```

class DataRequired:
    """
    Checks the field's data is 'truthy' otherwise stops the validation chain.

    This validator checks that the ``data`` attribute on the field is a 'true'
    value (effectively, it does ``if field.data``.) Furthermore, if the data
    is a string type, a string containing only whitespace characters is
    considered false.

    If the data is empty, also removes prior errors (such as processing errors)
    from the field.

    **NOTE** this validator used to be called `Required` but the way it behaved
    (requiring coerced data, not input data) meant it functioned in a way
    which was not symmetric to the `Optional` validator and furthermore caused
    confusion with certain fields which coerced data to 'falsy' values like
    ``0``, ``Decimal(0)``, ``time(0)`` etc. Unless a very specific reason
    exists, we recommend using the :class:`InputRequired` instead.

    :param message:
        Error message to raise in case of a validation error.

    Sets the `required` attribute on widgets.
    """
    def __init__(self, message=None):
        self.message = message
        self.field_flags = {"required": True}

    def __call__(self, form, field):
        if field.data and (not isinstance(field.data, str) or field.data.strip()):
            return

        if self.message is None:
            message = field.gettext("This field is required.")
        else:
            message = self.message

        field.errors[:] = []
        raise StopValidation(message)

```

조금 복잡한 이야기지만...

그래도 차근차근 읽다 보면 이해가 될 것입니다.

만약, 아무것도 입력하지 않았다면 해당 필드값으로 `None`이 채워지게 됩니다. 그리고 당연히 어떤 내용이 입력되어야 한다는 `validation` 조건을 통과하지 못했기 때문에 `form` 데이터를 전달받는 `question_views.py`의 `create` 함수 내부에 있는 `form.validate_on_submit()`은 `False`를 리턴하게 됩니다. 유용한 블로그가 ([click](#)) 있으니 참고하기 바랍니다.

`question_views.py`에서 `form` 데이터를 제대로 처리할 수 없기 때문에 아래 코드를 실행하게 되겠죠?

```

return render_template(
    template_name_or_list='question/question_form.html',
    form=form
)

```

그렇게 되면 `form`에 있는 모든 필드가 `None`으로 다시 채워지게 됩니다. 원래 코드였던 아래 `question_form.html`에서는 각 필드를 `None`으로 다시 채울 겁니다.

```

<!-- 수기로 만든 form -->
<div>

    <!-- 오류 내용을 표시하는 코드 추가-->
    <form method="post">
        {% for field, errors in form.errors.items() %}
            <div role="alert">
                {{ form[field].label }}: {{ ', '.join(errors) }}
            </div>
        {% endfor %}
    </form>

    <h5>질문 등록</h5>
    <form action="{{ url_for('question.create') }}" method="post">
        {{ form.csrf_token }}
        <div>
            <label for="title">제목</label>
            <input type="text" name="title" id="title" size="30" >
        </div>

        <div>
            <label for="contents">내용</label>
            <textarea name="contents" id="contents" cols="30" rows="10" ></textarea>
        </div>

        <button type="submit">저장하기</button>
    </form>
</div>

```

위 코드에 의하면 각 필드값에 `None`을 출력하니 아무것도 없는 빈칸이 될 것입니다.

하지만 우리가 `input` 태그와 `textarea` 태그에 `value="{{ form.contents.data or '' }}`" 속성을 추가한다면 어떻게 될까요?

공식 문서에서 제공하는 `DataRequired` 클래스 소스코드 코드에서 `__call__(self, form, field)` 함수에서 다음과 같은 코드를 볼 수 있습니다. 참고로 `call` 함수는 객체가 호출되었을 때 자동으로 실행되는 함수입니다.

```

if field.data and (not isinstance(field.data, str) or field.data.strip()):
    return

```

`form` 데이터로 빈 문자열 ''이 채워지고 `validator`인 `DataRequired()`를 통과하게 되어 Flask 서버로 전송될 것입니다.

그렇다면 왜 `DataRequired()` 통과할까요?

빈 문자열 ''이 담겨서 `DataRequired()`로 전달 됩니다. 그러면 `__call__` 함수가 실행되겠죠?

- 빈 문자열 ''이 전달되었으므로 `field.data`는 `False`입니다.
- 빈 문자열도 문자열 객체이므로 `isinstance(field.data, str)`는 `True`입니다.
- 빈 문자열은 `field.data.strip()`를 통과하면 `False`입니다.

따라서 `if True and (not True or False)`가 됩니다.

아무것도 없으므로 `None`입니다. `if`문에서 `None`은 거짓(`False`)로 평가되므로 자연스럽게 다음과 같은 구조를 갖습니다.

```

if field.data and (not isinstance(field.data, str) or field.data.strip())
    \(\downarrow\)
if False and (not True or False)

```

`and`는 하나라도 거짓이면 거짓이므로 뒤에 있는 `not isinstance(field.data, str) or field.data.strip()`와 상관없이 거짓이 됩니다. 그러므로 위 `if`문은 실행되지 않습니다.

그 이후에 `DataRequired`는 아래 코드를 실행하게 됩니다.

```

if self.message is None:
    message = fieldgettext("This field is required.")
else:
    message = self.message

field.errors[:] = []
raise StopValidation(message)

```

사용자가 별도로 메시지를 지정하지 않았다면 `This field is required.`가 표시되고, 별도로 지정했다면 해당 메시지를 출력합니다.

우리는 `forms.py`에서 `validators=[DataRequired('제목은 필수 입력 항목입니다.')]와 같이 지정했기 때문에 '제목은 필수 입력 항목입니다.'라는 메시지가 표시됩니다. 위 코드를 보면 모든 에러 메시지를 빈 리스트로 만들고 검증(validation) 과정을 중단 StopValidation(message)하게 됩니다.`

3.7.8. 댓글 등록도 form으로 바꾸기

질문 등록을 위한 `form` 클래스를 `forms.py`에 추가한 이후에 예전에 만들어 두었던 `question_detail.html`을 수정하고 `view`를 마저 바꿔줘야 겠죠?

`forms.py`에 `AnswerForm` 클래스를 다음과 같이 추가합니다.

```
# 파일 이름: /forms.py

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

class QuestionForm(FlaskForm):
    title = StringField(
        label='제목',
        # form 검증 실패 시 보여줄 메시지 포함
        validators=[DataRequired('제목은 필수 입력 항목입니다.')]
    )
    contents = TextAreaField(
        label='내용',
        # form 검증 실패 시 보여줄 메시지 포함
        validators=[DataRequired('내용은 필수 입력 항목입니다.')]
    )

# 추가된 부분
class AnswerForm(FlaskForm):
    contents = TextAreaField(
        label='내용',
        validators=[
            DataRequired(message='내용은 필수 입력 항목입니다.')
        ],
    )
```

`view` 파일 `answer_views.py`에서 `AnswerForm` 클래스를 사용할 수 있도록 수정합니다.

```

# 파일 이름: views/answer_views.py

from datetime import datetime
import imp
from flask import Blueprint, render_template, url_for, request
from werkzeug.utils import redirect
from hello_cju import db
from hello_cju.models import Question, Answer

# forms.py에서 만든 AnswerForm 클래스 임포트
from ..forms import AnswerForm

bp = Blueprint(
    name='answer',
    import_name=__name__,
    url_prefix='/answer',
)

@bp.route(rule='/create/<int:question_id>', methods=['POST'])
def create(question_id):

    # AnswerForm 객체 생성
    form = AnswerForm()

    question = Question.query.get_or_404(question_id)

    # AnswerForm 객체를 이용한 form 데이터 처리
    if form.validate_on_submit():
        contents = request.form['contents']
        answer = Answer(contents=contents, create_date=datetime.now())
        question.answer_set.append(answer)
        db.session.commit()
        return redirect(
            location=url_for(
                endpoint='question.detail',
                question_id=question_id,
            )
        )
    # form 데이터가 전달되지 않는 경우
    else:
        return render_template(
            template_name_or_list='question/question_detail.html',
            context=question,
            form=form,
        )

```

템플릿 파일 `question_detail.html`에서는 `form` 데이터에 에러가 있을 경우 표시하는 코드를 추가합니다. 기본적으로 [질문등록](#) 기능과 동일하므로 구체적인 설명은 생략합니다. 다음 코드에 포함된 주석을 참고하면서 보세요.

```

<!-- 파일 이름: templates/question/question_detail.html -->

<h1>{{ context.title }} </h1>

<div>
    {{ context.contents }}
</div>

<!-- 댓글을 뿌려주기 위해 추가한 부분 -->
<h5>{{ context.answer_set|length }}개의 댓글이 있습니다.</h5>
<div>
    <ul>
        {% for answer in context.answer_set %}
            <li>{{ answer.contents }}</li>
        {% endfor %}
    </ul>
</div>

<!-- 댓글 입력을 위해 추가한 부분 -->
<div>
    <form action="{{ url_for('answer.create', question_id=context.id) }}" method="post">
        {{ form.csrf_token }}

        <!-- form 에러 처리 -->
        {% for field, errors in form.errors.items() %}
            <div>
                <strong>{{ form[field].label }}</strong>: {{ ', '.join(errors) }}
            </div>
        {% endfor %}

        <!-- 댓글 입력 필드 -->
        <div>
            <textarea name="contents" id="contents" cols="30" rows="10"></textarea>
        </div>

        <!-- 댓글 등록 버튼 -->
        <input type="submit" value="댓글 등록">
    </form>
</div>

```

위 코드에서 댓글 조회 템플릿 파일 `question_detail.html`에서 `form`을 사용하여 에러를 표시하도록 코딩하였습니다. 템플릿에서 `form`을 사용할 수 있도록 `question_view.py`에 등록된 `detail` 험수에도 `AnswerForm`을 추가해 주어야 합니다. 아래 코드와 같이 `question_view.py` 모듈을 업그레이드 합니다.

```

# 파일 이름: views/question_views.py

# (... 이전 임포트 부분 생략 ...)

from ..forms import AnswerForm

# (... 중간 생략 ...)

@bp.route('/detail/<int:question_id>/')
def detail(question_id):
    form = AnswerForm()
    question = Question.query.get_or_404(question_id)
    return render_template(
        template_name_or_list='question/question_detail.html',
        context=question,
        form=form,
    )

# (... 이하 생략 ...)

```

이제 이번 장에서 목표로한 모든 기능 구현이 끝났습니다.

질문도 등록하고, 댓글도 등록해 보기 바랍니다.

현재 우리의 웹 시스템은 **CRUD** 기능 중에서 입력(**Create**)과 조회(**Read**) 기능만 적용되었습니다. 우리 시스템은 아직 수정(**Update**)과 삭제(**Delete**) 기능은 없습니다.

하지만 기능 구현은 여기서 멈추도록 하겠습니다. 수정(**Update**)과 삭제(**Delete**) 기능은 권한관리와 관련이 깊기 때문입니다. 아무나 들어와서 글을 수정하거나 삭제하면 큰일 나겠죠?

수정(**Update**)과 삭제(**Delete**) 기능은 회원관리 기능을 구현한 이후에 추가하도록 하겠습니다.

이제부터는 지금껏 만들어 놓은 시스템에 디자인을 입혀서 좀더 이쁘게 만드는 방법을 살펴보도록 하겠습니다.

3.8. 예쁘게 - CSS 적용

우리는 다음과 같이 기본기 7가지를 공부하였습니다.

❶ Flask 웹 시스템 구축을 위한 10가지 기본기

1. [프로젝트 기본 구조](#) \(\backslash\to\backslash\) **clear!**
2. [Application Factory 패턴](#) \(\backslash\to\backslash\) **Clear!**
3. [Blueprint 클래스 활용](#) \(\backslash\to\backslash\) **Clear!**
4. [ORM 모델 완벽 이해](#) \(\backslash\to\backslash\) **Clear!**
5. [질문 게시판 만들기](#) \(\backslash\to\backslash\) **Clear!**
6. [게시판 댓글 구현](#) \(\backslash\to\backslash\) **Clear!**
7. [질문 & 댓글 등록 - Flask Form 활용](#) \(\backslash\to\backslash\) **Clear!**
8. [예쁘게 - CSS 적용](#) \(\backslash\to\backslash\) **Clear!**
9. [더 예쁘게 - Bootstrap 활용](#)
10. [HTML 구조와 Template 상속](#)

이제 3가지 기본기만 더 배우면 웹 시스템 구현 전문가가 됩니다.

이번 장에서 배우는 [예쁘게 - CSS 적용](#)과 다음 장에서 배울 [더 예쁘게 - Bootstrap 활용](#)은 서로 연관되어 있는 내용입니다. HTML에 디자인을 입혀서 보다 예쁘게 만드는 기술입니다.

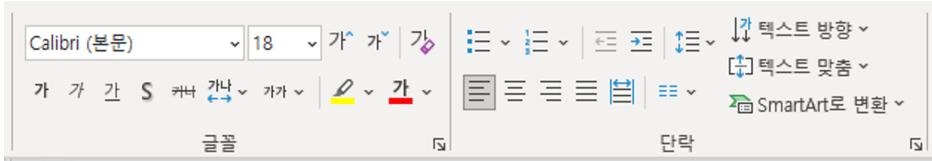
조금만 더 힘내세요 ^^.

3.8.1. 스타일(Style)의 필요성

지금까지 질문과 댓글을 등록하는 구현하였습니다. 그런데 아쉬움이 조금 남습니다. 기능은 완벽한데 보기에는 별로입니다. "보기 좋은 떡이 먹기도 좋다."라는 속담이 있듯 가급적이면 웹페이지를 보기 좋게 만들면 인터넷 사용자들이 보다 편리하게 사용할 수 있습니다.

우리가 문서를 작성할 때 보기 좋게 하려면 어떻게 할까요?

파워포인트 서식 편집



한글 서식 편집

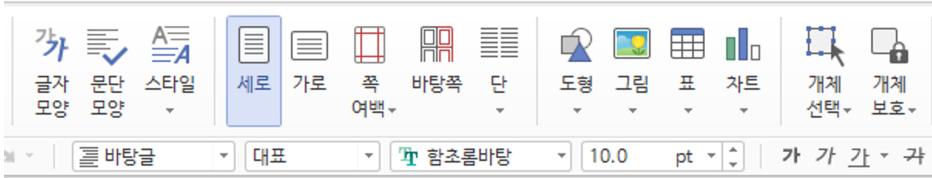


Fig. 3.44 파워포인트(위 그림) 및 한글(아래)에서 문서 서식 편집

그림 Fig. 3.44에서와 대부분의 문서 편집기에는 서식을 지정할 수 있습니다. 그림 Fig. 3.44는 파워포인트와 한글의 서식 편집 메뉴(아이콘)을 캡처한 그림입니다.

문서에 포함되는 객체에 따라 다양한 디자인 스타일을 적용할 수 있을 것입니다.

- 제목은 큰 글자로 가운데 정렬하고, 중요한 내용은 밑줄을 긋거나 빨간색으로 표시할 수 있을 것입니다.
- 표를 추가한 경우 선 색깔, 제목셀의 배경색 등을 지정할 필요도 있습니다.
- 그림을 추가할 경우 테두리를 넣을지, 캡션을 추가해서 그림 제목을 추가할 수도 있습니다.

물론 [메모장\(Notepad\)](#)이라는 간단한 문서편집기를 사용한다면 디자인 스타일을 적용할 필요가 없습니다. 메모장에는 서식을 사전에 편집하여 스타일로 등록하고 이것을 적용하는 기능이 지원되지 않기 때문입니다.

우리가 많이 사용하는 문서 편집기 **한글 hwp**은 디자인 스타일(style)을 편집하고 적용하는 기능을 지원하고 있습니다. 마이크로소프트사의 MS Word .docx 소프트웨어에도 물론 스타일 기능이 있습니다.

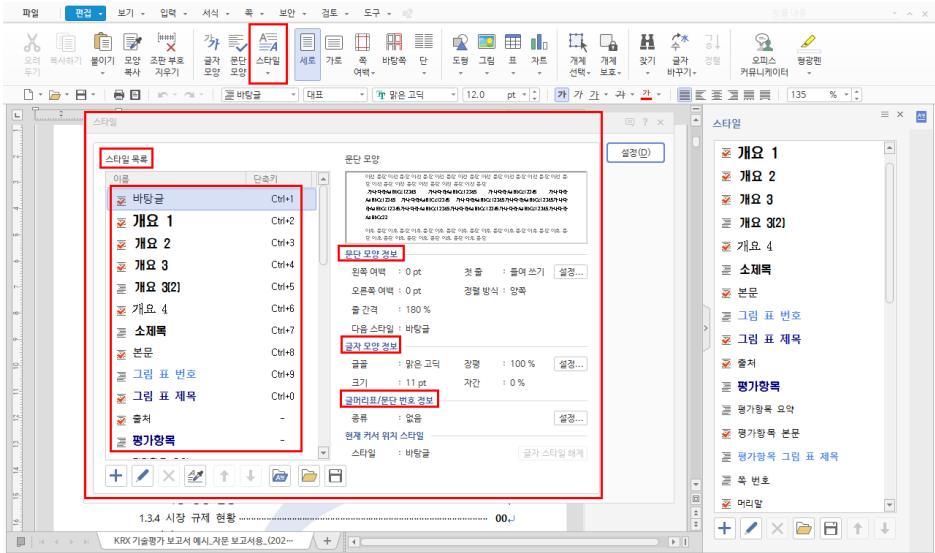


Fig. 3.45 한글에서 지원하는 스타일 편집 기능

그림 Fig. 3.45는 한글 hwp 에서 지원하는 스타일 편집창입니다. 우리가 편집할 문서의 내용이 많지 않다면 굳이 스타일을 사용할 이유가 없습니다.

하지만 문서 분량이 많아서 장/절을 편성하고, 각각의 장/절 수준, 표안의 내용, 각주, 참고문헌 등에 따라 일정한 디자인을 적용하고 싶을 경우라면 어떨까요? 스타일을 사전에 정의하고 필요한 부분에 단축키를 활용해서 편리하게 적용하는 것이 필수입니다.

스타일이 기능이 없다면 각각의 장/절 제목이나 문단 수준에 따라 글자크기, 줄간격, 굵게, 밑줄, 글자색깔 등 다양한 디자인 스타일을 일일히 지정해 주어야 할 것입니다. 여간 번거로운 일이 아닐 수 없습니다. 문서 편집기에서 스타일 기능은 문서 작성자에게 많은 수고를 덜어 주는 고마운 기능입니다.

HTML 문서는 어떨까요?

우리가 코딩한 템플릿 파일 **.html**은 화면을 렌더링한 다음에 그 결과를 크롬 같은 브라우저에 뿐려주게 됩니다. 템플릿 파일이 렌더링을 하는 과정에서 HTML 태그를 인식하고 그 효과를 적용하여 브라우저에 결과를 보여주게 됩니다. **<a>** 태그를 이용해서 링크를 걸어줄 수도 있고 **** 태그를 이용하면 굵은 글자로 표현할 수도 있습니다.

.html도 어차피 문서입니다. 렌더링 과정을 거쳐 최종 문서를 브라우저에 보여주게 됩니다. 브라우저 화면에 보이는 문서도 다양한 디자인 스타일이 필요할 것입니다. 브라우저에 보이는 문서에도 제목은 글자 크기를 크게 하고 밑줄을 짓는 등 디자인이 필요합니다. HTML 문서도 분량이 많아지면 다양한 디자인이 필요하게 될 것입니다. 디자인이 필요할 때마다 그때 그때 스타일을 매번 반복적으로 코딩해야 할까요?

우리가 이쁜 인터넷 문서를 볼 수 있는 것은 **.html**에 포함된 다양한 태그와 각각의 태그에 적용한 속성에 따라 렌더링을 해주기 때문입니다.

결국 **.html** 파일도 최종 문서를 보여주기 위한 템플릿 문서라고 볼 수 있습니다. 한글이나 파워포인트는 다양한 서식을 지정하면 프로그램 자체에서 내부적으로 처리하고 그 결과를 문서 편집창에 띄워줄 뿐입니다. 한글이나 MS Word 처럼 스타일 기능이 있다면 당연히 좋겠죠?

브라우저 화면에서 볼 수 있는 이쁜 문서들은 우리가 한글이나 파워포인트가 제공하는 서식 아이콘을 HTML 태그를 통해 작성한 것입니다.

그림 Fig. 3.46을 볼까요?

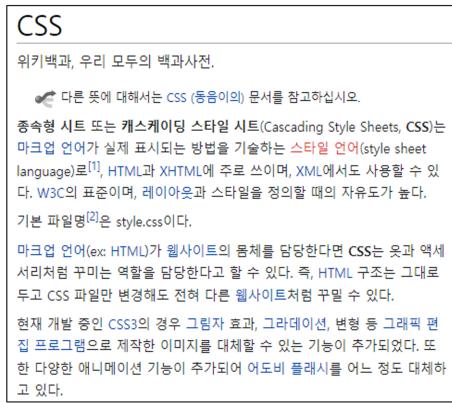
그림 Fig. 3.46의 왼쪽 그림은 구글 검색창에서 CSS (**click**)를 검색한 결과에 대한 일부 내용을 캡처한 것입니다. 오른쪽 그림은 동일한 내용을 메모장을 이용하여 입력한 그림입니다.

어떤 그림이 더 보기 좋은가요?

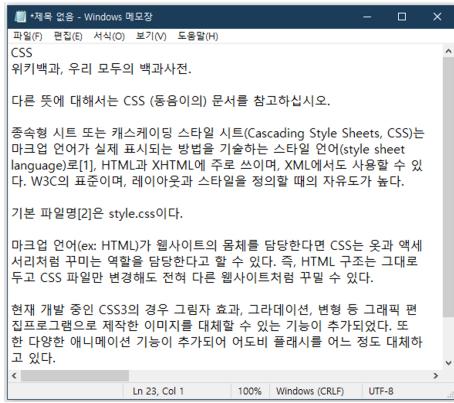
대부분은 왼쪽 그림이 더 좋다고 할 것입니다. 링크가 걸린 문자는 파란색, 강조하는 단어는 굵게, 중요한 단어는 빨간색으로 표시한 것을 볼 수 있습니다. 그림 아이콘도 추가되었군요.

이왕이면 사용자에게 보기 편하고 직관적으로 보다 많은 정보를 제공할 수 있는 디자인 스타일을 적용하는게 당연히 좋겠죠?

이렇게 하면 다양한 디자인 효과를 줄 수 있습니다.



웹 브라우저(크롬)에서 보이는 내용



메모장을 이용해 내용 편집

Fig. 3.46 브라우저에서 CSS를 검색한 화면(일부)과 동일한 내용을 메모장으로 편집한 화면

하지만, 문제가 하나 있습니다.

예를 들어 `<div>` 태그에 속성값을 부여하여 디자인을 적용하고 싶은 경우가 있다고 가정해 봅니다. `<div>` 태그는 웹페이지 렌더링할 때 구역을 구분하는 태그로 많이 쓰입니다. 동일한 디자인을 적용해야 할 `<div>` 태그가 1개 문서에 100번 등장한다고 가정하면 100번에 걸쳐 동일한 코드를 입력해야 합니다. 여간 번거로운 일이 아닐 수 없습니다.

어찌어찌 해서 100개의 `<div>` 태그에 속성을 코딩해 줬다고 해도 문제는 여전히 남습니다. 디자인을 변경하고 싶은 경우에는 100군데를 일일히 찾아서 수정해 줘야 합니다.

게다가 이런 웹 페이지가 1개가 아니라 수십 개인 경우는 문제가 더욱 심각해 집니다. 상상하기도 어려운 삽질이 되는 겁니다. 설령 삽질을 감수한다고 하더라도 중간에 오타가 발생할 확률은 언제든지 있습니다.

어떻게든 이런 문제점을 해결해야 겠죠?

이래서 등장한 것이 바로 CSS(Cascading Style Sheet)입니다.

Cascading은 '위에서 아래로 흐르는' 또는 '상속, 종속하는'이라는 의미를 가지고 있습니다. HTML 문서는 다양한 태그들이 구조를 가지고 있습니다. `<body>` 태그 내부에 `<div>` 태그가 있고, 그 안에 `<p>` 태그가 있는 것과 같은 구조입니다. CSS에서 Cascading이라는 단어는 구조와 상속 관계에서 우선순위를 결정해서 디자인을 적용한다는 의미가 있습니다.

`style`은 우리가 적용하고 싶은 디자인을 의미합니다.

`Sheet`은 데이터를 담고 있는 파일을 의미합니다. 엑셀에도 데이터 시트(Sheet)가 있는 것처럼 CSS에도 디자인 `style` 데이터를 담고 있는 `Sheet`가 필요하겠죠?

요렇게 3개 단어가 합쳐져서 만들어진 것이 CSS입니다.

CSS는 하나의 파일로 만들게 됩니다. 거기에서 스타일 적용이 필요한 태그의 디자인을 정의합니다. CSS파일은 `.css`라는 확장자를 사용합니다. `.css` 파일에서 특정 태그에 대한 속성을 지정하면 해당되는 태그가 1개이든 100개이든 상관없이 공통적으로 스타일을 적용할 수 있습니다.

스타일을 대상을 지정하는 것을 [선택자\(selector\)](#)라고 부릅니다. 태그를 지정하는 것을 태그 선택자, 특정한 부분들만 골라서 해당되는 요소들에게 공통적인 스타일을 지정하는 것을 [클래스\(class\)](#) 선택자라고 합니다. 문서 내에서 유일한 CSS는 태그를 선택해서 스타일을 지정하는 것을 [아이디\(id\)](#) 선택자라고 부릅니다. 보다 자세한 내용은 참고 블로그 [CSS 기초\(1\)](#) [click](#)을 참고하기 바랍니다.

W3CSchools ([click](#))에도 훌륭한 Tutorial이 있으니 추가적인 학습을 원하는 사람은 참고하면 좋을 것 같습니다.

CSS는 여러분에게 참 편리한 기능입니다.

CSS도 컴퓨터가 이해하고 해석해야 하기 때문에 일종의 프로그래밍 언어라고 볼 수도 있습니다. 프로그래밍 언어는 계속해서 발전하고 있습니다. CSS도 마찬가지겠죠? 현재 CSS 버전은 3 입니다. 현재는 W3C (World Wide Web Consortium) ([click](#))이라는 단체에서 표준을 만들고 지속적으로 업그레이드를 진행하고 있습니다. 참고로 CSS3 로고는 그림 [Fig. 3.47](#)와 같습니다.

그림 [Fig. 3.47](#)에서 CSS 아래에 숫자 3을 볼 수 있죠? CSS 버전을 의미합니다.

CSS 파일은 템플릿에서 렌더링 하는 과정에서 값이 변하지 않습니다. 웹페이지를 생성할때마다 스타일이 바뀌면 안되겠죠? 이런 점 때문에 정적 `static` 파



Fig. 3.47 CSS3 로고

일이라고 부릅니다.

템플릿에서 디자인 스타일을 적용할 때는 `static/` 이라는 디렉토리를 만들고 그 안에 `.css` 파일을 모아놓게 됩니다.

우리는 Flask를 이용한 웹 시스템의 기능 구현에 보다 초점을 맞추고 있기 때문에 자세한 설명은 생략하도록 하겠습니다. CSS에 대하여 좀 더 깊은 이해를 원하는 사람들은 전문 교재를 구입하거나 구글 검색을 통해 블로그나 유튜브 강의를 참고하시기 바랍니다.

3.8.2. CSS 파일을 만들고 적용해 보기

CSS 파일을 모아두기 위한 디렉토리를 하나 만들겠습니다. VS code에서 작업하는 경우라면 그림 Fig. 3.48와 같이 `static/` 디렉토리를 만들고 그 안에 `style.css`라는 파일을 하나 만듭니다.

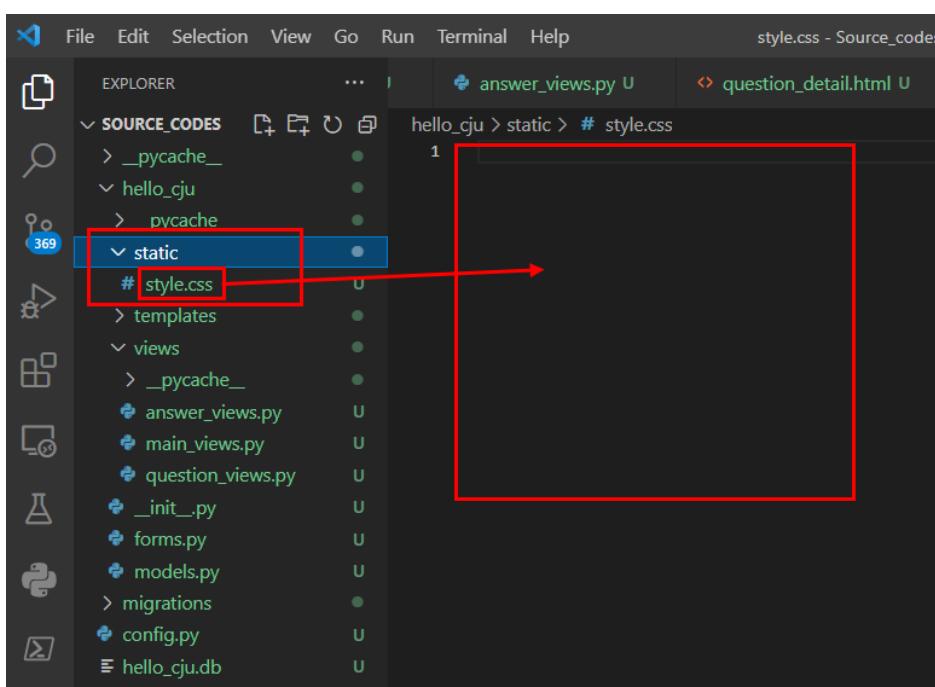


Fig. 3.48 VS code 파일 탐색창에 `static` 디렉토리를 만들고 `style.css` 파일 생성

우리가 만들었던 템플릿 파일 `question_detail.html`에 있는 `<textare>` 태그와 `<input>` 태그에 스타일을 적용하기 위한 내용을 `style.css` 파일에 아래와 같이 코딩해 줍니다.

```

/* 파일 이름: /static/style.css */

textarea {
    width: 100%;
}

input[type=submit] {
    margin-top: 10px;
}

```

위 코드는 `<textarea>` 태그에 입력창 너비를 화면의 100%로 늘리는 스타일을 적용하고, `<input>` 태그에는 위쪽 마진(여유)를 10 px (픽셀, pixel) 주겠다는 의미입니다. 그림 Fig.3.48의 오른쪽 사각형 영역에 위 코드를 입력하면 됩니다.

이제 `style.css`를 템플릿 파일 `question_detail.html`에 적용해 보도록 하겠습니다.

`question_detail.html` 파일을 열고 맨 윗부분에 아래 코드를 입력합니다.

```
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}>
```

VS code에서 입력한다면 그림 Fig.3.49를 참고하면 됩니다.

```

File Edit Selection View Go Run Terminal Help
question_detail.html - Source Codes - Visual Studio Code

EXPLORER
  SOURCE CODES
    > .pycache_
    > hello_cju
      > .pycache_
        static
          # style.css
        templates
          > question
            < question_detail.html > U
              <!-- 파일 이름: templates/question/question_detail.html -->
              1 <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}>
              2
              3
              4
              5 <h1>{{ context.title }}</h1>
              6
              7 <div>
              8   | {{ context.contents }}
              9 </div>
              10
              11 <!-- 댓글을 뿌려주기 위해 추가한 부분 -->
              12 <h5>{{ context.answer_set|length }}개의 댓글이 있습니다.</h5>
              13 <div>
              14   <ul>
              15     {% for answer in context.answer_set %}
              16       | <li>{{ answer.contents }}</li>
              17     {% endfor %}
              18   </ul>
              19 </div>
              20
              21 <!-- 댓글 입력을 위해 추가한 부분 -->
              22 <div>
              23   <form action="{{ url_for('answer.create', question_id=context.id) }}" method="post">
              24     {{ form.csrf_token }}
              25
              26     <!-- form 배려 처리 -->
              27     {% for field, errors in form.errors.items() %}
              28       <div>
              29         | <strong>{{ form[field].label }}</strong>: {{ ', '.join(errors) }}
              30       </div>
              31     {% endfor %}
              32
              33     <!-- 댓글 입력 필드 -->
              34     <div>
              35       | <textarea name="contents" id="contents" cols="30" rows="10"></textarea>
              36     </div>
              37
              38     <!-- 댓글 등록 버튼 -->
              39     <input type="submit" value="댓글 등록" />
              40   </form>
              41 </div>

```

Fig. 3.49 VS code에서 css 적용 코드 입력

위 코드를 입력하고 입력된 질문 중 하나를 클릭해서 들어가면 우리가 `style.css` 파일에 설정한 스타일이 `<textarea>` 태그와 `<input>` 태그에 적용된 것을 확인할 수 있습니다.

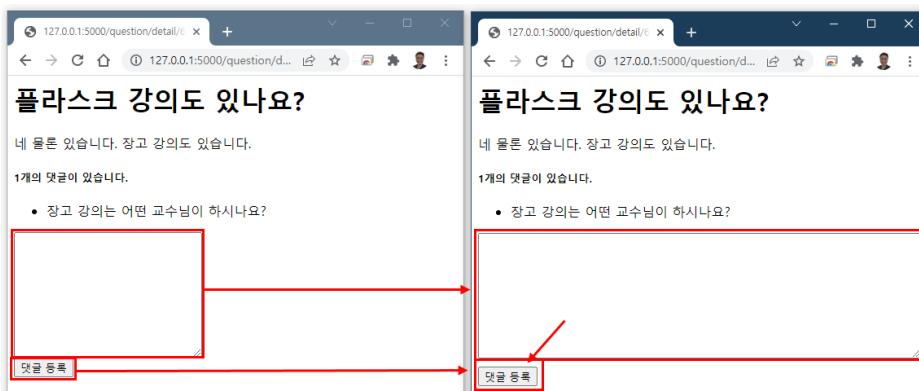


Fig. 3.50 CSS 적용 전,후 비교

그림 Fig.3.50의 왼쪽 그림은 CSS 적용 전 화면이고, 오른쪽은 적용 후 화면입니다. 우리가 style.css 파일에서 지정한대로 잘 반영이 되었습니다. <textarea> 태그는 입력창 폭이 화면의 100% 차지하도록 넓어졌습니다. <input> 태그 적용된 댓글등록 버튼은 위쪽으로 10픽셀 마진(여유공간)이 추가되었습니다. 만약 <textarea> 태그와 <input> 태그가 여러 개 있다면 모두 공통적으로 적용됩니다.

다양한 CSS 문법을 더 공부한다면 여러분들의 웹페이지를 이쁘고 아름답게 꾸밀 수 있겠죠?

그런데 CSS는 HTML에 디자인 스타일을 적용하기 위한 기술입니다. CSS 역시 컴퓨터에게 명령을 하는 것이니 또 다른 프로그래밍 언어라고 볼 수 있습니다.

프로그래밍 언어를 새로 배워야 하는 것은 큰 부담이겠죠?

게다가 우리는 주로 백엔드(Back-end) 관련 기능을 구현하는데 집중하고 있습니다. 백엔드 구현에 쏟을 시간도 부족한데 프런트에 해당하는 스타일 작업을 위해 CSS 문법을 공부해야 한다는 것은 부담일수 밖에 없습니다.

이런 문제점 때문에 자주 사용하는 기능들을 미리 CSS로 이쁘게 구현해 놓고 가져다 쓰기만 하도록 할 수 있는 방법이 있습니다. 우리가 프로그래밍 언어에서 잘 만들어지고 검증된 라이브러리를 가져다 쓰는 것과 동일한 개념입니다.

우리가 살펴볼 것은 Bootstrap이라는 프레임워크입니다.

Bootstrap을 이용하면 아주 쉽게 전문가 수준의 스타일을 템플릿 파일에 적용하여 웹페이지를 렌더링 할 수 있습니다.

Bootstrap이 궁금하신 분들은 Next를 클릭해서 다음 절로 이동해 주세요.

3.9. 더 예쁘게 - Bootstrap 활용

우리는 다음과 같이 기본기 8가지를 공부하였습니다.

❶ Flask 웹 시스템 구축을 위한 10가지 기본기

1. [프로젝트 기본 구조](#) \(\to\) Clear!
2. [Application Factory 패턴](#) \(\to\) Clear!
3. [Blueprint 클래스 활용](#) \(\to\) Clear!
4. [ORM 모델 완벽 이해](#) \(\to\) Clear!
5. [질문 게시판 만들기](#) \(\to\) Clear!
6. [게시판 댓글 구현](#) \(\to\) Clear!
7. [질문 & 댓글 등록 - Flask Form 활용](#) \(\to\) Clear!
8. [더 예쁘게 - CSS 적용](#) \(\to\) Clear!
9. [더 예쁘게 - Bootstrap 활용](#) \(\to\) 지금 도전!
10. [HTML 구조와 Template 상속](#)

이제 우리는 8가지 기본기를 살펴보았습니다. 이제 끝이 보입니다. 조금만 더 힘내세요.

이번 절은 앞에서 [더 예쁘게 - CSS 적용](#) 배운 CSS를 보다 빠르고 편리하면서도 고품질의 프런트를 개발하기 위한 기술인 Bootstrap에 대하여 공부할 것입니다.

기능 개발자들이 디자인에 신경쓰고 이쁘게 만드는 작업까지 한다는 것은 엄청난 부담입니다. 사실 디자인 작업은 엄청난 고민과 시간이 필요합니다. 개발자인 우리들이 디자인을 직접 할 수도 없고, 돈을 들여 디자이너를 고용하기도 힘든 경우가 많습니다.

이때 활용할 수 있는 것이 Bootstrap입니다.

Bootstrap은 트위터(Twitter)를 개발하면서 만들어졌습니다. 오픈 소스로 공개되어 있고 해당 코드는 Github에 ([click](#)) 공개되어 있습니다. 세계적으로 가장 많이 사용하는 툴킷(Toolkit)으로 알려져 있습니다. 우리도 배워놓으면 좋겠죠?

3.9.1. Bootstrap 활용 방법

Bootstrap을 활용할 수 있는 방법은 2가지입니다.

1. 소스 파일을 다운로드 받는 방법
2. 온라인(CDN)으로 연결하여 사용하는 방법

i CDN 이란?

CDN(Content Delivery Network)은 필요한 데이터를 여러 개의 서버에 복사해 두고 인터넷 사용자의 요구(request)가 있을 경우에 사용자에게 네트워크(인터넷)를 통해 제공하는 네트워크입니다. 서비스나 데이터 제공자의 입장에서는 하나의 서버에 모든 데이터를 한꺼번에 저장하지 않아서 속도 측면에서 유리합니다. 자세한 내용은 온라인 위키 [콘텐츠 전송 네트워크](#) ([click](#)) 참고하세요.

우리는 2가지 경우를 모두 살펴보도록 하겠습니다.

3.9.2. 소스 파일 활용

소스 파일을 활용하는 방법은 Bootstrap에서 공개한 [.css](#) 파일을 다운로드 받은 다음 [static/](#) 폴더에 복사하여 사용하는 방법입니다.

이 방법의 장점은 [.css](#) 파일을 내 컴퓨터(혹은 서버)에 저장하고 있기 때문에 안정적으로 서비스를 할 수 있다는 것입니다. 네트워크가 불안정해도, Bootstrap 버전이 업데이트 되더라도 내 컴퓨터나 서버에 저장한 파일을 이용할 수 있습니다.

소스 파일을 이용하는 방법의 단점도 있습니다. 웹 시스템의 프론트엔드(Front-end) 기술은 매우 빠르게 발전하고 있습니다. Bootstrap도 프론트엔드를 지원하는 기술이므로 업데이트 되고 있습니다. 새로 개발되거나 보완되어 배포(Release)된 디자인 스타일을 적용하려면 다시 소스 파일 [.css](#)을 다운로드하고 테스트 해야 하는 번거로움이 있습니다.

최신 버전은 Bootstrap의 [Versions](#) ([click](#))에서 확인할 수 있습니다. [Latest](#)라고 표시된 버전이 최신 버전입니다.

3.9.2.1. 소스 파일 다운로드

Bootstrap 다운로드 페이지 ([click](#)) 방문하여 [Compiled CSS and JS](#)에서 [Download](#) 아이콘을 클릭하여 다운로드 받습니다.

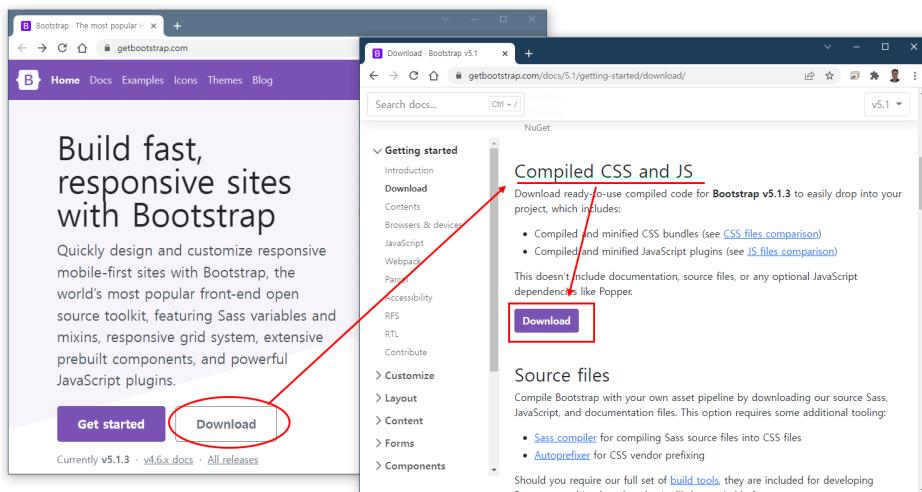


Fig. 3.51 Bootstrap 소스 파일 다운로드 받기

그림 [Fig. 3.51](#)를 참고하여 소스 파일을 다운로드하면 압축 파일 [bootstrap-x.x.x-dist.zip](#) 형태로 다운로드 됩니다. 파일 이름에서 [x.x.x](#)는 숫자로 표기되며 다운로드하는 Bootstrap의 버전을 알려줍니다. [dist](#)는 배포판이라는 의미이고, [zip](#)은 압축파일이라는 뜻입니다.

다운로드 받은 파일을 압축해제하면 다음과 같은 디렉토리와 파일 구조를 가집니다.

```

bootstrap/
├── css/
│   ├── bootstrap-grid.css
│   ├── bootstrap-grid.css.map
│   ├── bootstrap-grid.min.css
│   ├── bootstrap-grid.min.css.map
│   ├── bootstrap-grid rtl.css
│   ├── bootstrap-grid rtl.css.map
│   ├── bootstrap-grid rtl.min.css
│   ├── bootstrap-grid rtl.min.css.map
│   ├── bootstrap-reboot.css
│   ├── bootstrap-reboot.css.map
│   ├── bootstrap-reboot.min.css
│   ├── bootstrap-reboot.min.css.map
│   ├── bootstrap-reboot rtl.css
│   ├── bootstrap-reboot rtl.css.map
│   ├── bootstrap-reboot rtl.min.css
│   ├── bootstrap-reboot rtl.min.css.map
│   ├── bootstrap-reboot utilities.css
│   ├── bootstrap-reboot utilities.css.map
│   ├── bootstrap-reboot utilities.min.css
│   ├── bootstrap-reboot utilities.min.css.map
│   ├── bootstrap-reboot utilities.rtl.css
│   ├── bootstrap-reboot utilities.rtl.css.map
│   ├── bootstrap-reboot utilities.rtl.min.css
│   ├── bootstrap-reboot utilities.rtl.min.css.map
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap.rtl.css
│   ├── bootstrap.rtl.css.map
│   ├── bootstrap.rtl.min.css
│   ├── bootstrap.rtl.min.css.map
└── js/
    ├── bootstrap.bundle.js
    ├── bootstrap.bundle.js.map
    ├── bootstrap.bundle.min.js
    ├── bootstrap.bundle.min.js.map
    ├── bootstrap.esm.js
    ├── bootstrap.esm.js.map
    ├── bootstrap.esm.min.js
    ├── bootstrap.esm.min.js.map
    ├── bootstrap.js
    ├── bootstrap.js.map
    ├── bootstrap.min.js
    └── bootstrap.min.js.map

```

파일 구조가 무시무시하죠?

하지만 크게 2개의 디렉토리로 구성되어 있습니다. 하나는 CSS에 관련된 파일을 모아놓은 `css` 디렉토리이고 다른 하나는 JavaScript 관련 파일을 모아놓은 `js` 디렉토리입니다. 해당 디렉토리에 다양한 파일들이 속해 있는 구조입니다.

Bootstrap에서 제공하는 CSS 기능만 활용하려면 `bootstrap/css/bootstrap.min.css` 파일만 사용해도 큰 무리가 없습니다. `bootstrap.min.css` 파일을 복사하여 우리가 기존에 만들어 놓았던 `static/` 디렉토리에 붙여넣게 합니다.

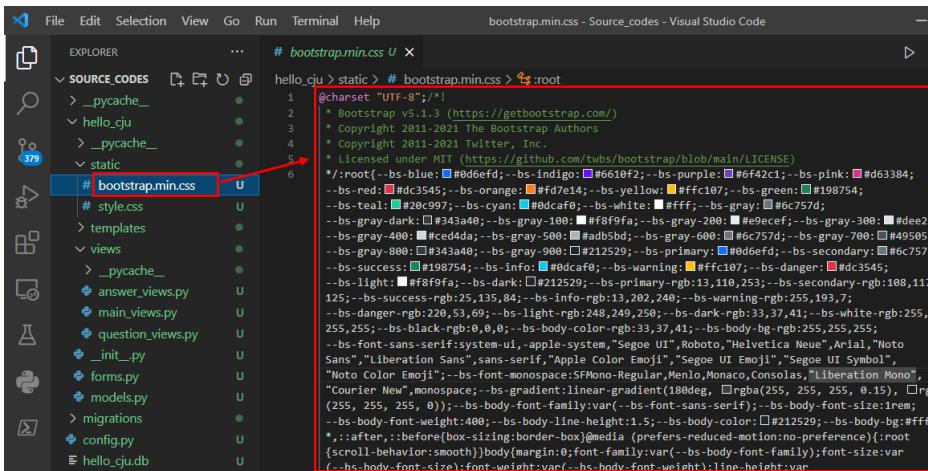


Fig. 3.52 다음 받은 `bootstrap.min.css` 파일을 VS code에서 확인해 보기

그림 Fig.3.52와 같이 static/bootstrap.min.css를 복사하여 붙여넣기 이후에 파일을 클릭해보면 무시무시한 코드가 보입니다. 하지만 걱정하지 마세요.

bootstrap.min.css에 있는 코드는 Bootstrap에서 디자인 전문가들과 개발자들이 미리 만들어 놓은 코드이므로 우리가 건드리거나 이해할 필요는 없습니다. bootstrap.min.css 파일을 잘 활용할 수 있으면 되는 겁니다.

이제 Bootstrap 소스 파일을 이용하여 디자인을 적용할 준비가 끝났습니다.

3.9.3. 우리가 구현한 템플릿에 적용해 보기

3.9.3.1. 템플릿 question_list.html에 적용하기

템플릿 파일 templates/question/question_list.html에 자세한 설명을 하면서 CSS 스타일을 적용해 보겠습니다.

먼저 아래 코드와 같이 question_list.html 파일을 코딩해 줍니다. 코드를 먼저 제시하고, 코드를 참조하면서 CSS를 설명하도록 하겠습니다.

```
<!-- bootstrap.min.css 파일을 사용하기 위한 임포트 -->
<link rel="stylesheet" href="{{ url_for('static', filename='bootstrap.min.css') }}>

<!-- 기존 질문 리스트를 div 태그로 둘러싸고
    클래스 이름이 'container my-3'인 디자인을 적용 -->
<div class="container my-3">

    <!-- 질문 목록을 표로 만들기 위한 table 태그
        추가하고 CSS 디자인은 'table'을 적용 -->
    <table class="table">

        <!-- 표 헤더(컬럼명)를 지정, 디자인은 'table-dark' 스타일 지정 -->
        <thead>
            <tr class="table-dark">
                <th>번호</th>
                <th>제목</th>
                <th>작성일시</th>
            </tr>
        </thead>

        <!-- 표 내용(tbody: table body)에 질문 목록을 렌더링 -->
        <tbody>
            {% if question_list %}
            {% for question in question_list %}
            <tr>
                <td>{{ loop.index }}</td>
                <td>
                    <a href="{{ url_for('question.detail', question_id=question.id) }}">
                        {{ question.title }}</a>
                </td>
                <td>{{question.create_date}}</td>
            </tr>
            {% endfor %}

            {% else %}
            <p>질문이 없습니다.</p>
            {% endif %}
        </tbody>
    </table>

    <!-- 질문 등록하기 버튼에 CSS 'btn btn-primary' 스타일 지정 -->
    <a href="{{ url_for('question.create') }}" class="btn btn-primary">
        질문 등록하기
    </a>
</div>
```

위 코드에서 특징적인 것은 이전 코드는 `` 태그를 이용해서 목록만 간단히 렌더링 했지만 업그레이드 된 코드는 `<table>` 태그를 적용하였습니다. `<table>` 태그에서는 표(테이블)의 컬럼명을 지정하는 `<thead>`와 표 내용을 표시하는 `<tbody>` 태그를 사용하였습니다.

그림 Fig.3.53은 question_list.html 파일에 CSS 디자인을 적용한 결과입니다.

위 코드를 잘 살펴보면 `class="클래스 이름"`이라는 형태의 속성을 추가한 태그들이 있습니다. `class` 속성을 추가한 태그 목록은 다음과 같습니다.

- `<div class="container my-3">` \(\rightarrow\) Bootstrap 참고 페이지 ([click](#))
- `<table class="table">` \(\rightarrow\) Bootstrap 참고 페이지 ([click](#))
- `<tr class="table-dark">` \(\rightarrow\) Bootstrap 참고 페이지 ([click](#))
- `` \(\rightarrow\) Bootstrap 참고 페이지 ([click](#))

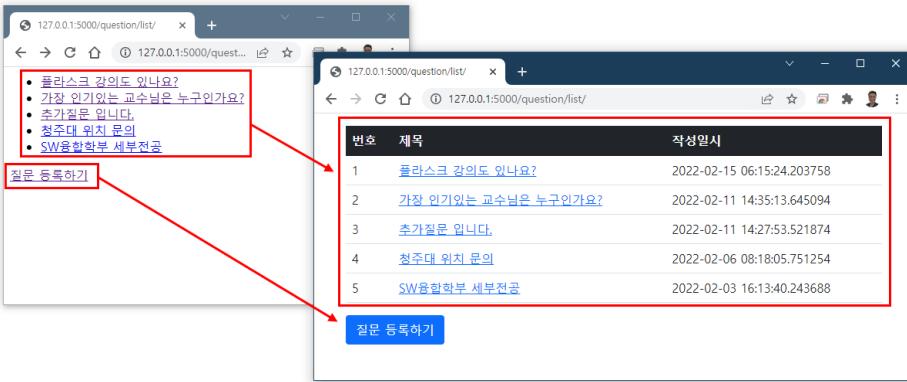


Fig. 3.53 question_list.html에 CSS 디자인을 적용한 결과

본 튜토리얼의 주제가 Bootstrap은 아니므로 모든 속성에 대하여 설명하지는 않습니다. 다만, 여러분들이 어떻게 사용할 수 있을지, 즉 물고기를 잡아서 건네주는 대신에 물고기 잡는 방법을 알려드리겠습니다. 설명을 위해 Bootstrap이 적용된 `<div class="container my-3">` 사용하겠습니다. 나머지 내용은 위에서 설명한 목록에 **Bootstrap 참고 페이지**를 같이 제공하였으니 직접 방문해서 활용해 보시기 바랍니다.

먼저 태그에 사용된 속성 중 `class`에 대하여 설명하겠습니다. `class`는 CSS에서 주로 사용하는 문법입니다. 템플릿 파일 `.html`과 CSS 파일 `.css`에서 `class` 속성이 작동하는 원리는 다음과 같습니다.

- CSS 파일을 열어보면 `.클래스이름 {적용할 디자인 코드}` 형태로 코딩된 부분을 볼 수 있습니다.
- CSS 파일에서 점 `.`은 클래스를 의미합니다.
- 템플릿 파일 `.html`에서 속성이 `class`로 지정된 모든 태그를 찾습니다.
- `class` 속성을 갖는 태그들 중에서 `클래스이름`이 동일한 태그를 추려냅니다.
- 이렇게 추려낸 태그들에 `{적용할 디자인 코드}`를 적용하여 스타일을 완성합니다.

`<div class="container my-3">`를 하나의 예로 활용하겠습니다.

먼저 `div` 태그에 적용할 스타일은 `class` 이름이 `container my-3`이 될 것입니다. 그림 Fig. 3.52에서 내용을 잘 찾아보면 분명히 `.contatiner`라는 스타일이 있을 것입니다. VS code에서 찾기 기능(단축키 `CTL+F```)을 활용해 `.container`를 키워드로 검색하면 여러 개를 찾을 수 있을 것입니다.

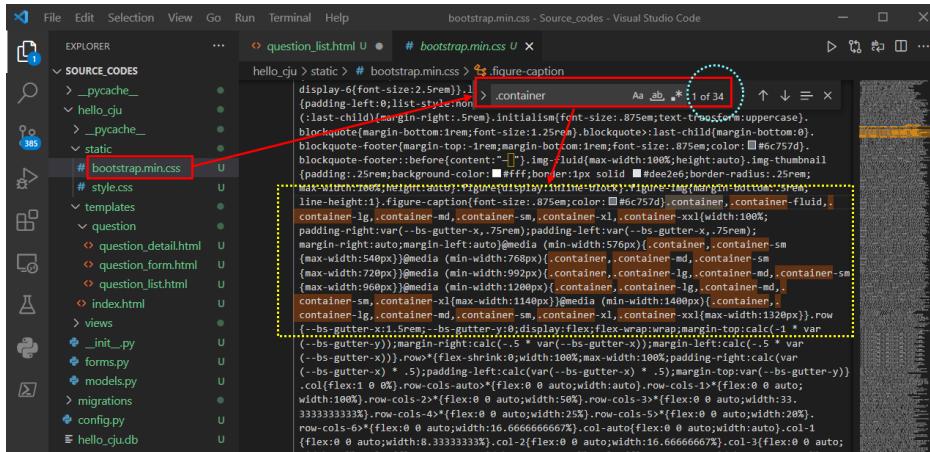


Fig. 3.54 VS code를 이용해 bootstrap.min.css에서 `.container`를 검색한 결과

제가 검색해보니 그림 Fig. 3.54와 같이 보입니다. 34개가 검색되었네요. 그리고 복잡하게 뭔가 코딩되어 있습니다. 우리가 신경쓸 것은 없습니다. "아! 우리가 다운받은 bootstrap.min.css 파일 어딘가에 container 클래스가 정의되어 있구나!!" 정도로 개념을 잡으면 됩니다.

그러면 `contatiner` 클래스의 모양(스타일)은 어떻게 생겼을까요?

구체적인 스타일과 사용법은 Bootstrap 공식 문서를 참고하면 됩니다. Bootstrap의 [Docs](#) 페이지에서 [Layout \\$\to\\$ Containers](#) 메뉴를 찾아서 들어가거나 검색창에 `Containers` 입력해도 쉽게 찾을 수 있습니다.

이 참고 링크 ([click](#))를 클릭해서 방문해 봅니다. 링크를 클릭해서 방문하면 다음과 같은 화면이 나타납니다.

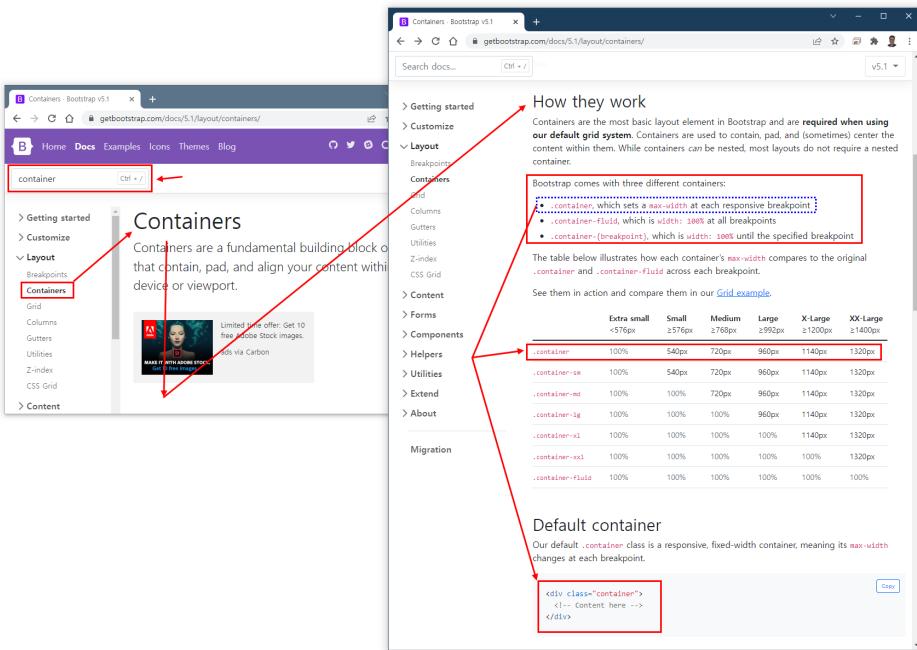


Fig. 3.55 Container 클래스를 설명하는 Bootstrap 공식 문서

그림 Fig. 3.55가 잘 안보이면 이미지를 클릭해서 확대해 보거나 해당 링크를 따라서 방문해 보기 바랍니다. 일단 그림 Fig. 3.55를 이용해서 설명하겠습니다.

그림 Fig. 3.55 왼쪽 화면에서 **Containers** 메뉴를 클릭하면 해당 페이지가 나타납니다. 아래쪽으로 약간 내려오면 오른쪽 화면과 같이 **How they work**라는 설명을 만날 수 있습니다.

첫 번째 빨간색 네모칸 - 파란 점선 네모칸 안에 우리가 사용한 `.container`가 보이죠? 설명을 읽어보니 해당하는 중단점까지 최대한 확대하는 효과를 준다 `which sets a max-width at each responsive breakpoint`라고 써있네요. 이걸 어떻게 해석하나고요? 그러면 중간에 있는 빨간 네모칸을 참고하면 됩니다. 표 컬럼명은 브라우저의 가로 픽셀수입니다.

컬럼명이 `Small`인 경우를 예로 들어볼까요? 브라우저(크롬 등)의 가로 길이가 `576px` 이상이면 `.container`를 클래스의 스타일을 사용하는 태그는 최대 540 픽셀 `max-540`까지 늘인다는 의미입니다. 만약 브라우저의 가로 길이가 `Medium`으로 768 픽셀 이상으로 커지면 `.container`를 적용한 태그의 가로 길이는 최대 720 픽셀 만큼 늘이게 됩니다. 100 퍼센트(`100%`)로 표시된 것은 최대 가로 픽셀수를 모두 활용해서 해당 태그를 늘이겠다는 의미입니다.

이렇게 하면 뭐가 좋을까요?

알송달송 하시죠?

이런 스타일의 장점은 디스플레이(모니터, 액정, 스크린) 또는 활성화된 브라우저 크기에 따라 자동으로 크기가 태그의 크기를 적절하게 조절할 수 있습니다. 모니터가 작은데 태그의 크기가 커서 화면에서 잘려 보이거나 좌우 스크롤을 하지 않아도 되는 장점이 있습니다.

이런 식으로 상황에 따라 적절히 변하는 방식으로 작성한 애플리케이션을 **반응형 앱**이라고 합니다.

그러면 `container` 다음에 공백 한칸 다음에 있는 `my-3`는 도대체 뭘까요? 클래스 속성 다음에 한칸 공백 후 나오는 옵션은 **여백(Spacing)** 설정에 관한 사항입니다.

`<div class="container my-3">`의 경우 태그는 `div`, 클래스 이름은 `container`, 그리고 여백은 `my-3`이라는 의미입니다.

그럼 도대체 여백 옵션 `my-3`의 정체는 무엇일까요?

다시 한번 알송달송 합니다.

이에 대한 답은 Bootstrap 공식문서 **Spacing** 페이지 ([click](#))에 상세하게 나와 있습니다.

공식문서 **Spacing** 페이지 ([click](#))를 클릭해서 들어가거나 **Docs** 페이지에서 **Utilities \$\to\$ Spacing** 메뉴로 들어가면 아래 그림 Fig. 3.56이 나옵니다.

그림 Fig. 3.56의 내용을 살펴보면 **Spacing**은 스타일을 적용한 태그의 여백을 만들어 주는 클래스라고 설명하고 있습니다. `container`라는 클래스 다음에 `my-3`이라는 클래스가 또 나온 경우가 되겠습니다.

The screenshot shows the Bootstrap v5.1 documentation for the Spacing module. The 'Spacing' section is highlighted with a red box. Three arrows point from the 'property', 'sides', and 'size' definitions back to their respective sections in the sidebar.

- Where `property` is one of:**
 - `m` - for classes that set `margin`
 - `p` - for classes that set `padding`
- Where `sides` is one of:**
 - `t` - for classes that set `margin-top` or `padding-top`
 - `b` - for classes that set `margin-bottom` or `padding-bottom`
 - `s` - (start) for classes that set `margin-left` or `padding-left` in LTR, `margin-right` or `padding-right` in RTL
 - `e` - (end) for classes that set `margin-right` or `padding-right` in LTR, `margin-left` or `padding-left` in RTL
 - `x` - for classes that set both `*-left` and `*-right`
 - `y` - for classes that set both `*-top` and `*-bottom`
 - blank - for classes that set a `margin` or `padding` on all 4 sides of the element
- Where `size` is one of:**
 - `0` - for classes that eliminate the `margin` or `padding` by setting it to `0`
 - `1` - (by default) for classes that set the `margin` or `padding` to `$spacer * .25`
 - `2` - (by default) for classes that set the `margin` or `padding` to `$spacer * .5`
 - `3` - (by default) for classes that set the `margin` or `padding` to `$spacer`
 - `4` - (by default) for classes that set the `margin` or `padding` to `$spacer * 1.5`
 - `5` - (by default) for classes that set the `margin` or `padding` to `$spacer * 3`
 - `auto` - for classes that set the `margin` to `auto`

Fig. 3.56 클래스 여백(Spacing)을 설명하는 Bootstrap 공식 문서

클래스가 공백(space)을 사이에 두고 여러 개가 나올 경우 첫 클래스에 해당하는 스타일을 적용하고, 적용된 스타일에 두 번째 클래스에 해당하는 스타일을 적용합니다.

클래스가 여러 개일 경우 순차적으로 스타일을 적용합니다. 연결되는 것이죠. 바로 Cascading 된다고 표현합니다. 부모 요소를 자식이 상속(Inheritance)한다는 표현을 쓰기도 합니다.

Spacing 옵션은 2가지로 사용할 수 있습니다.

- `{property}{sides}-{size}`
- `{property}{sides}-{breakpoint}-{size}`

그림 Fig. 3.56에서 처음 나오는 `property`는 반드시 다음 둘 중 하나를 선택해야 합니다.

- `m`: 클래스의 외부여백(margin) 설정
- `p`: 클래스의 내부여백(padding) 설정

`sides`는 반드시 다음 옵션 중 하나를 선택해야 합니다.

- `t`: 클래스의 위쪽 마진 또는 위쪽 패딩
- `b`: 클래스의 아래쪽 마진 또는 아래쪽 패딩
- `s`: 스타일을 적용받는 맨 처음 클래스의 왼쪽 마진 `margin-left` 또는 왼쪽 패딩 `margin-left`
- `e`: 스타일을 적용받는 맨 마지막 클래스의 오른쪽 마진 `margin-right` 또는 왼쪽 패딩 `margin-right`
- `x`: 클래스의 왼쪽과 오른쪽 공백
- `y`: 클래스의 위쪽과 아래쪽 공백
- blank: 위, 아래, 왼쪽, 오른쪽에 공백

마지막으로 `size`는 크기를 나타냅니다. 반드시 다음 중 하나의 옵션을 사용해야 합니다.

- **0**: 마진이나 패딩 크기가 0
- **1**: 마진 또는 패딩 크기가 `$spacer * 0.25`
- **2**: 마진 또는 패딩 크기가 `$spacer * 0.5`
- **3**: 마진 또는 패딩 크기가 `$spacer * 1.0`
- **4**: 마진 또는 패딩 크기가 `$spacer * 1.5`
- **5**: 마진 또는 패딩 크기가 `$spacer * 3.0`
- **auto** - Bootstrap이 자동으로 처리

`$spacer`의 단위는 기본값으로 **1 rem**입니다. 컴퓨터가 `rem`은 브라우저에서 사용하는 폰트 크기 `font-size` 속성에 따라서 상대적인 길이를 계산해 주는 값입니다. 일반적으로 **1 rem**은 10픽셀을 의미합니다.

`<div class="container my-3">`의 의미는 `div` 태그에 `container` 스타일을 적용하고, 그 다음 외부여백 마진(**m**) 위치를 위쪽과 아래쪽에 `y $spacer` (약 10 픽셀) 크기만큼 적용하라는 뜻입니다. 그림 [Fig. 3.56](#)을 자세히 살펴보면 표 외곽에 `container` 클래스 스타일이 적용되고 위, 아래 여백으로 10픽셀이 추가된 것을 알 수 있습니다.

나머지 태그 `table, tr, a` 적용된 스타일도 직접 클릭해서 특징을 살펴보기 바랍니다.

- `<table class="table">` \(\backslash(\to\backslash)\) Bootstrap 참고 페이지 ([click](#))
- `<tr class="table-dark">` \(\backslash(\to\backslash)\) Bootstrap 참고 페이지 ([click](#))
- `` \(\backslash(\to\backslash)\) Bootstrap 참고 페이지 ([click](#))

3.9.3.2. 템플릿 `question_detail.html`에 적용하기

이번에는 템플릿 파일 `question_detail.html`에 Bootstrap을 적용해 보겠습니다.

본 Tutorial의 주제가 Bootstrap이 아니므로 구체적인 설명은 생략합니다. 아래 코드에 적용된 구체적인 내용은 Bootstrap 공식 문서 ([click](#)) 참고하세요

```

<!-- 파일 이름: templates/question/question_detail.html -->

<!-- bootstrap.min.css 파일을 사용하기 위한 임포트 -->
<link rel="stylesheet" href="{{ url_for('static', filename='bootstrap.min.css') }}">

<!-- container 클래스로 답변 전체를 감싸기 -->
<div class="container my-3">
    <h2 class="border-bottom py-2">{{ context.title }}</h2>
    <div class="card my-3">
        <div class="card-body">
            <div class="card-text" style="white-space: pre-line;">{{ context.contents }}</div>
            <div class="d-flex justify-content-end">
                <div class="d-flex p-2 bd-highlight">
                    작성시간: {{ context.create_date }}
                </div>
            </div>
        </div>
    </div>
</div>

<h5 class="border-bottom my-3 py-2">{{ context.answer_set|length }}개의 답변이 있습니다.</h5>

{% for answer in context.answer_set %}
<div class="card my-3">
    <div class="card-body">
        <div class="card-text" style="white-space: pre-line;">{{ answer.contents }}</div>
        <div class="d-flex justify-content-end">
            <div class="d-flex p-2 bd-highlight">
                작성시간: {{ context.create_date }}
            </div>
        </div>
    </div>
</div>
{% endfor %}

<!-- form 에러 처리 -->
{% for field, errors in form.errors.items() %}
<div class="alert alert-danger" role="alert">
    <strong>{{ form[field].label }}</strong>: {{ ', '.join(errors) }}
</div>
{% endfor %}

<!-- 댓글 입력 필드 -->
<form action="{{ url_for('answer.create', question_id=context.id) }}" method="post" class="my-3">
    {{ form.csrf_token }}
    <div class="form-group">
        <textarea name="content" id="content" class="form-control" rows="10"></textarea>
    </div>

    <div class="form-group my-3">
        <input type="submit" value="댓글등록" class="btn btn-primary" />
    </div>
</form>
</div>

```

3.9.3.3. 템플릿 question_form.html에 적용하기

이번에는 마지막으로 남은 템플릿 파일 `question_form.html`에 Bootstrap을 적용해 보겠습니다.

본 Tutorial의 주제가 Bootstrap이 아니므로 구체적인 설명은 생략합니다. 아래 코드에 적용된 구체적인 내용은 Bootstrap 공식 문서 ([click](#)) 참고하세요

```

<!-- 파일 이름: templates/question/question_form.html -->
<!-- bootstrap.min.css 파일을 사용하기 위한 임포트 -->
<link rel="stylesheet" href="{{ url_for('static', filename='bootstrap.min.css') }}" %>

<div class="container">
    <!-- 오류 내용을 표시하는 코드 추가-->
    <form method="post" class="my-3">
        {% for field, errors in form.errors.items() %}
            <div class="alert alert-danger" role="alert">
                {{ form[field].label }}: {{ ', '.join(errors) }}
            </div>
        {% endfor %}
    </form>

    <h5 class="my-3 border-bottom pb-2">질문 등록</h5>

    <form action="{{ url_for('question.create') }}" method="post">
        {{ form.csrf_token }}
        <div class="card my-3">
            <div class="card-body">
                <div class="card-text" style="white-space: pre-line;">제목</div>
                <div class="input-group mb-3">
                    <input type="text" class="form-control" name="title" id="title" size="100" value="{{ form.title.data or '' }}"/>
                </div>
            </div>
        </div>
    </div>

    <div class="card my-3">
        <div class="card-body">
            <div class="card-text" style="white-space: pre-line;">내용</div>
            <div class="form-floating">
                <div class="d-flex p-2 bd-highlight">
                    <textarea class="form-control" placeholder="질문 내용을 입력하세요" id="floatingTextarea2" style="height: 300px" value="{{ form.contents.data or '' }}"/>
                    <!-- <textarea name="contents" id="contents" cols="30" rows="10" value="{{ form.contents.data or '' }}"/></textarea> -->
                </div>
            </div>
        </div>
    </div>

    <div class="form-group my-3">
        <button type="submit" class="btn btn-primary">저장하기</button>
    </div>
    </form>
</div>

```

이제 모든 템플릿 파일에 Bootstrap 적용을 마무리 했습니다.

Flask 서버를 작동시키고 이리저리 입력도 해보고 댓글도 달아 보면서 여러분의 웹 시스템을 즐겨 보세요.

3.9.4. 온라인(CDN)으로 연결하여 활용

앞서 우리는 Bootstrap의 CSS 파일을 다운로드한 뒤에 `static/` 폴더에 복사하여 사용하는 방법을 알아보았습니다. 그리고 Bootstrap을 해석하는 방법을 알아보았습니다.

이번에는 파일을 다운로드하고, 압축 풀고, 복사하는 것 없이 직접 인터넷 **CDN**에 접속하여 활용하는 방법에 대해 살펴보도록 하겠습니다.

Bootstrap 공식 문서 중에서 [Quick start \(click\)](#)에 접속해 봅니다.

그림 Fig.3.57은 Bootstrap 공식 문서에서 CDN으로 CSS에 접속하는 페이지입니다.

Bootstrap은 CDN 링크를 복사할 수 있는 방법을 `CSS`, `JS`, `Bundle` 형태로 제시하고 있습니다. 다양한 선택 옵션이 있으니 개발자들이 필요에 따라 링크를 복사해 가면 되겠죠?

그런데 `CSS`, `JS`, `Bundle` 형태가 각각 어떤 의미인지 모르는 사람도 있을 겁니다. 그런 사람들을 위해 간단히 설명하겠습니다.

- **CSS**: Bootstrap에서 제공하는 CSS 기능에 접속하는 링크입니다. 링크는 `<head>` 태그의에서 다른 CSS를 로딩하기 전 위치에 있어야 합니다.
- **JS**: Bootstrap의 CSS는 구현하기 위해서는 다양한 JavaScript 플러그인과 `Popper`을 사용해야 합니다. Bootstrap에서 지원하는 기능을 모두 사용하기 위해서는 `JS`에서 제공하는 링크입니다. `JS` 기능을 CDN에서 불러오는데 시간이 걸릴 수 있습니다. 굳이 많은 기능이 필요 없다면 `JS`에서 지원하는 링크를 포함하지 않아도 됩니다. 링크 위치는 `.html` 파일의 어디라도 상관 없지만 `</body>` 태그 바로 직전에 있는 것을 추천합니다.
 - **Bundle**: JavaScript 플러그인 링크와 `Popper` 링크를 각각 복사하는 번거로움을 덜어주기 위해, 모든 기능을 하나로 묶어서 제공하는 링크입니다.

Quick start

Looking to quickly add Bootstrap to your project? Use jsDelivr, a free open source CDN. Using a package manager or need to download the source files? [Head to the downloads page.](#)

CSS

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

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
```

JS

Many of our components require the use of JavaScript to function. Specifically, they require our own JavaScript plugins and [Popper](#). Place **one of the following** `<script>`s near the end of your pages, right before the closing `</body>` tag, to enable them.

Bundle

Include every Bootstrap JavaScript plugin and dependency with one of our two bundles. Both `bootstrap.bundle.js` and `bootstrap.bundle.min.js` include [Popper](#) for our tooltips and popovers. For more information about what's included in Bootstrap, please see our [contents](#) section.

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="...>
```

Separate

If you decide to go with the separate scripts solution, Popper must come first (if you're using tooltips or popovers), and then our JavaScript plugins.

```
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js" integrity="...>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js" integrity="...>
```

Fig. 3.57 Bootstrap CDN 접속 링크 복사하기

다.

- **Separate:** JavaScript 플러그인과 [Popper](#) 링크를 분리해서 제공합니다. 2개의 링크를 복사해서 사용할 경우 [Proper](#) 툴킷과 관련된 코드가 먼저 나와야 합니다.

① Popper

먼저 툴팁([tooltip](#))을 이해해야 합니다. 툴팁은 마우스가 어떤 [요소\(element\)](#) 위치에 있을 때 마우스를 클릭하지 않아도 작은 상자가 요소 위에 나타나서 보충 설명을 보여주는 것입니다. 쉽게 [말풍선](#)이라고 생각하며 됩니다.

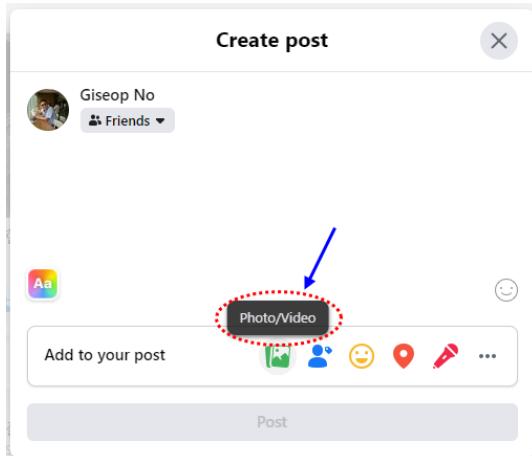


Fig. 3.58 페이스북 글쓰기에서 이미지 위에 마우스를 가져갔을 때 나타는 툴팁

[Popper](#)는 툴팁의 위치를 쉽게 잡아주도록 도와주는 유틸리티입니다.

우리는 그림 Fig. 3.57에서 표시한 [css](#)와 [Bundle](#) 링크를 복사하여 사용하겠습니다. 빨간색 원에 있는 [Copy](#) 아이콘을 누르면 링크 주소가 복사됩니다. 물론 마우스로 드래그 해서 복사해도 됩니다. 링크 주소가 아예 `<script>` 태그 안에 작성되어 있기 때문에 엄청 편리하게 사용할 수 있습니다.

그림 Fig. 3.57에서 복사한 코드는 각각 다음과 같습니다.

```
<!-- Bootstrap CDN의 CSS 접속/사용을 위한 코드 -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
```

```
<!-- Bootstrap CDN의 JS Bundle 접속/사용을 위한 코드 -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
```

우리가 작성한 모든 템플릿 파일 .html의 맨 앞부분에 있는 코드 `<link rel="stylesheet" href="{{ url_for('static', filename='bootstrap.min.css') }}>`를 주석처리하거나 삭제하고 위 코드로 교체합니다.

저는 아래와 같이 주석처리 하였습니다. 여러분은 과감하게 삭제하고 위 코드로 교체해 보기 바랍니다.

```
<!-- bootstrap.min.css 파일을 사용하기 위한 임포트 -->
<!-- <Link rel="stylesheet" href="{{ url_for('static', filename='bootstrap.min.css') }}> -->

<!-- Bootstrap CDN의 CSS/JS Bundle 사용 -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
```

다시 Flask 서버를 작동시키고 지금까지 우리가 만든 웹페이지에 접속하면 정상적으로 잘 작동하는 것을 확인할 수 있습니다.

심지어는 Bootstrap 홈페이지에서 다운로드하고, 압축 풀어서 static 디렉토리에 복사해 놓은 bootstrap.min.css 파일을 삭제해 버려도 아주 잘 작동합니다. 우리가 직접 만든 style.css 파일은 나중에 필요할지도 모르니 삭제하지 않고 그냥 유지하도록 하겠습니다.

저는 앞으로 CDN에 접속해서 Bootstrap 사용하는 방법을 기본으로 적용하겠습니다. 물론 파일을 다운로드 받는 방법으로 계속 사용해도 전혀 문제되지 않습니다. 여러분의 취향대로 선택하면 됩니다.

[예쁘게 - CSS 적용](#)과 [더 예쁘게 - Bootstrap 활용](#)는 우리가 만든 웹 시스템을 더 이쁘게 보이도록 하는 기술에 대해서 배웠습니다.

다음으로 배울 내용은 HTML 문서의 기본 구조를 살펴보고 템플릿 파일을 분할하여 효율적으로 코딩하는 방법입니다. 이 방법은 템플릿 상속이라는 개념을 통해 쉽게 달성할 수 있습니다.

더 배울 준비가 되었나요?

좋습니다!

Next 아이콘을 클릭하여 이동하기 바랍니다.

3.10. HTML 구조와 Template 상속

우리는 어디까지 와 있는 것일까요?

현재 아래와 같이 기본기 9가지를 공부하였습니다.

❶ Flask 웹 시스템 구축을 위한 10가지 기본기

1. [프로젝트 기본 구조](#) \(\to\) Clear!
2. [Application Factory 패턴](#) \(\to\) Clear!
3. [Blueprint 클래스 활용](#) \(\to\) Clear!
4. [ORM 모델 완벽 이해](#) \(\to\) Clear!
5. [질문 게시판 만들기](#) \(\to\) Clear!
6. [게시판 댓글 구현](#) \(\to\) Clear!
7. [질문 & 댓글 등록 - Flask Form 활용](#) \(\to\) Clear!
8. [예쁘게 - CSS 적용](#) \(\to\) Clear!
9. [더 예쁘게 - Bootstrap 활용](#) \(\to\) Clear!
10. [HTML 구조와 Template 상속](#) \(\to\) 지금 도전!

이제 우리는 9가지를 기본기를 살펴보았습니다.

어느덧 끝이 보입니다.

Flask를 이용해 웹 시스템을 구축하는 기본기의 마지막입니다.

끝까지 힘내세요!!!

지금까지 우리는 템플릿 파일 `.html` 만들면서 우리만의 웹 시스템을 구축해 왔습니다. 하지만 템플릿 파일 작성에서 지켜야 할 규칙들을 완전히 무시하면서 진행했습니다.

솔직히 말하면, 지금 우리가 만든 `.html` 파일들은 정상적이지 않습니다.

그럼 `.html`에도 정상과 비정상이 있다는 말인가요? 네! 그렇습니다. 정상적인 경우는 HTML 표준 구조를 준수하는 것이고, 미준수하는 경우는 비정상적인 HTML 파일입니다.

그럼 정상적인 `HTML 표준 구조`는 무엇일까요?



Fig. 3.59 HTML5 로고

3.10.1. HTML 기본 구조 살펴보기

현재 HTML 표준 버전은 5 입니다. 사람들은 `HTML5` 또는 `HTML 파이브`라고 부릅니다. HTML5의 로고는 그림 Fig. 3.59와 같습니다. HTML 로고 아랫쪽 숫자 5가 보이죠? HTML 버전 5라는 의미를 담고 있습니다.

HTML은 브라우저에 보이는 화면을 생성해(렌더링) 내기 위한 문서이지만, 결국 컴퓨터가 `.html` 문서를 해석해야 하므로 일종의 컴퓨터 언어라고 보아야 할 것입니다. 컴퓨터 기술이 발전하면서 HTML도 덩달아 발전 하겠죠? 그래서 표준을 만들고 계속해서 업그레이드를 진행하고 있습니다.

여러분이 현재 사용하는 모든 HTML 관련 사항은 모두 `HTML5`입니다. 이전 버전은 굳이 배우지 않아도 됩니다. HTML의 표준 업그레이드는 새로운 문법이 추가/삭제되거나 HTML 설계 철학이 변경되는 경우에 진행됩니다. 우리는 굳이 세부적인 내용까지 알아야 할 필요는 없습니다. 그건 전세계에서 HTML을 가장 잘 아는 전문가들이 결정한 사항이니까요. 알아서 잘 했겠죠? 우리는 그냥 표준을 믿고 쓰면 됩니다.

사실 HTML 표준은 W3C (World Wide Web Consortium) ([click](#))이라는 단체에서 추천하는 HTML 작성 규칙입니다. 가장 최신 표준은 `HTML Living Standard` ([click](#))입니다. HTML5 표준을 유지하고 관리하는 사람들은 애플, 구글, 모질라, 마이크로소프트가 주축이 되어 만든 WHATWG (Web Hypertext Application Technology Working Group) ([click](#)) 기관(일종의 연구 그룹이라고 생각하면 됩니다)에서 담당하고 있습니다.

정상적인 `HTML 표준 구조` 이야기를 꺼내려다 보니 말이 길어졌습니다. 제가 결국 하고 싶은 말은 모든 템플릿 파일 `.html` 작성할 때 W3C ([click](#))에서 제시한 표준 구조를 잘 따라야 한다는 말씀입니다.

어? 그럼 그 표준 구조는 어디서 확인할까요? `HTML Living Standard` ([click](#))의 내용 중에 `4.1.1 The html element`라는 부분에 ([click](#)) 나와 있습니다.

표준 문서에서 추천하는 HTML 구조는 다음과 같습니다.

```
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8">
    <title>Flask 튜토리얼</title>
  </head>
  <body>
    <h1>Flask 소개</h1>
    <p>CJU Flask 튜토리얼을 통해
       웹 시스템을 구축할 수 있어요.</p>
  </body>
</html>
```

위 코드 내용 중 `<meta charset="UTF-8">`는 `4.2.5 The meta element` ([click](#)) 부분을 참고하여 추가하였습니다.

각각의 코드를 살펴보면 어떤 역할을 하는지 알 수 있습니다. 그리고 위 코드의 구조는 정상적인 HTML의 기본 구조가 됩니다.

① HTML 기본구조

- <!DOCTYPE HTML>: 브라우저는 스스로 HTML 문서의 버전을 판단하기 어렵기 때문에 사용자가 직접 Version을 선언 HTML5를 사용함을 브라우저에 선언하는 역할을 합니다.
- <html>: 전체 HTML 문서를 감싸는 태그입니다. 하나만 존재해야 하고 html 바깥에 DOCTYPE을 제외한 다른 태그가 있으면 안 됩니다. <html> 태그에 속성을 부여해서 문서에서 사용할 언어를 한국어 lang=ko로 지정하였습니다.
- <head>: html 문서에 대한 정보를 나타내는 부분입니다. 하나만 존재해야하고, <html> 태그 바로 아래에 있어야합니다.
 - <title>: <head> 태그 내부에 들어가는 태그로 제목 표시줄의 내용을 나타냅니다. 위의 예시를 보면 제목 표시줄의 내용은 HTML 기본구조입니다.
 - <meta> : <meta> 역시 <head> 태그 안에 들어가는 태그로 문서에 대한 설명을 표시합니다. 사람에게는 보이지 않고 브라우저(컴퓨터)만 볼 수 있습니다. 속성으로 charset="utf-8"이라고 한 것은 브라우저에게 한글 인코딩을 UTF-8로 설정하라고 알려줍니다. 이 부분이 있어야 한글이 깨지지 않습니다.
 - <meta> 태그는 문서 전체에 공통적으로 적용할 속성을 정의할 때 사용합니다.
- <body>: HTML 문서에서 실제적으로 브라우저 화면에 보이는 부분입니다. 하나만 존재해야 하고, <html> 태그 내부에, <head> 태그 다음에 위치해야 합니다.
 - <body> 태그 내부는 문서의 내용과 그 내용을 설정하는 다양한 태그를 사용하여 작성합니다.
 - 예시에서는 <h1>, <p> 태그를 이용하여 간단한 내용을 작성해 봤습니다.

위 코드를 저장하고, 그 .html 파일을 브라우저로 열면 그림 Fig. 3.60 같습니다.

저는 파일명을 index.html로 지어주고 저장해 봤습니다.



Fig. 3.60 .html 파일을 브라우저(크롬)로 열었을 때 화면

그림 Fig. 3.60은 템플릿 파일 index.html을 활용해 랜더링을 해서 브라우저 화면에서 확인한 것입니다.

❶ VS code에서 HTML 기본 구조 쉽게 만들기

모든 템플릿 `.html` 파일을 만들 때마다 HTML 기본 구조를 준수하는 코드를 작성한다는 것은 프로그래머에게 매우 귀찮은 일입니다. 거의 똑같은 코드를 매번 타이핑 해야하기 때문입니다.

그래서 대부분의 IDE는 단축키를 통해 자동으로 HTML 기본구조를 만들어 줍니다.

우리가 사용하고 있는 IDE는 `VS code` 입니다. `VS code`에서 임의의 `.html` 파일을 만들고 내용 편집창에 느낌표 `!`를 입력하면 표준 HTML 문서 구조를 자동완성 해주는 기능이 활성화 됩니다. 자동완성 기능을 영어로는 `auto completion` 이라고 부릅니다.

아래 그림 Fig. 3.61과 같이 자동완성 활성화 창이 나옵니다. 마우스로 원하는 형태를 클릭하거나 `tab` 키를 치면 자동으로 HTML 표준 구조를 만들어 줍니다.

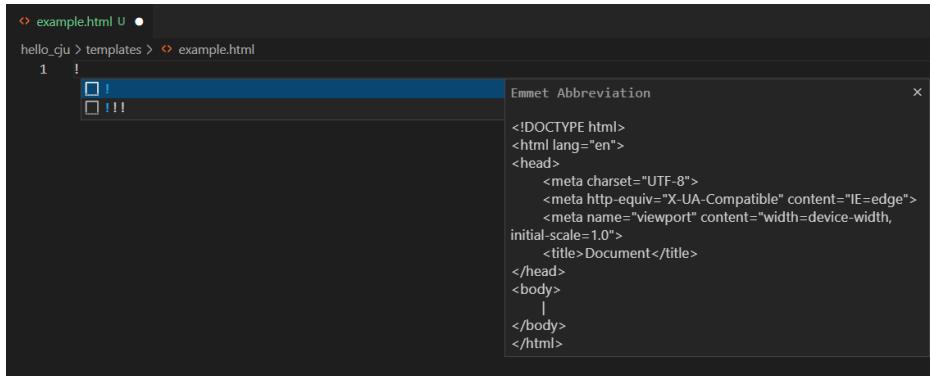


Fig. 3.61 `example.html` 새로 만들고 편집창에서 느낌표 `!` 입력하면 나오는 화면

자동 완성이 되면 그림 Fig. 3.62에서 표시한 예와 같이 본인이 필요한 부분은 수정하고, 삭제할 내용은 삭제하고, 추가할 내용은 각자 선택에 따라 합니다. `<body>` 태그에 문서 내용을 작성하면 됩니다.



Fig. 3.62 `example.html` 새로 만들고 편집창에서 느낌표 `!` 입력하면 나오는 화면

여러분들이 `VS code` 이외의 다른 IDE를 사용한다면 그 편집기에도 자동완성 기능이 있을 것입니다. 각자 사용하는 IDE의 `auto completion` 기능은 구글링 해보세요.

3.10.2. 템플릿 상속의 필요성

우리는 템플릿 `.html` 파일의 기본 구조를 살펴보았습니다. 모든 템플릿이 기본 구조를 따르는 것은 좋지만, 약간 문제가 있습니다. `<head>` 태그에 들어가는 내용은 거의 모든 템플릿 `.html` 파일에 동일하게 적용된다는 점입니다.

물론 `VS code`의 `auto completion` 기능을 활용하면 약간 번거롭기는 하지만 일일히 추가할 수도 있습니다. 그래도 문제가 있습니다.

만약에 `<head>` 태그 내용을 추가하거나, 수정, 삭제할 경우에는 어떻게 될까요?

맞습니다. 모든 템플릿 `.html` 파일을 찾아서 바꿔줘야 합니다. 파일 내용을 바꾸는 과정에서 오타가 발생할 수도 있고, 어떤 파일은 실수로 빼먹을 수도 있습니다. 프로그래머도 사람이다 보니 이런 위험성은 항상 존재한다고 봐야 합니다.

이런 문제를 해결하기 위한 방법은 무엇일까요? 여러분들은 이렇게 생각할 것입니다.

"모든 템플릿 '.html' 파일에 공통적으로 적용되는 부분은 별도의 파일로 만들고, 내용이 변하는 부분만 따로 작성한다면 어떨까?"

네, 정확히 예측하셨습니다.

맞습니다. 공통적으로 적용되는 내용을 별도로 만들고, 나머지 파일들은 그 내용을 기본적으로 가져다 쓰도록 하면 됩니다. 이것이 바로 이번 장에서 배우려고 하는 템플릿 상속의 기본 아이디어와 정확히 똑같은 개념입니다.

템플릿 상속은 반복적인 `<head>` 태그 내용을 단순화 하는 것 이외에 프로그래머에게 많은 편리함을 줍니다.

브라우저에서 화면이 바뀌어도 바뀌지 않는 내용이 있습니다. 아마 여러분들도 많이 보셨을 것입니다.

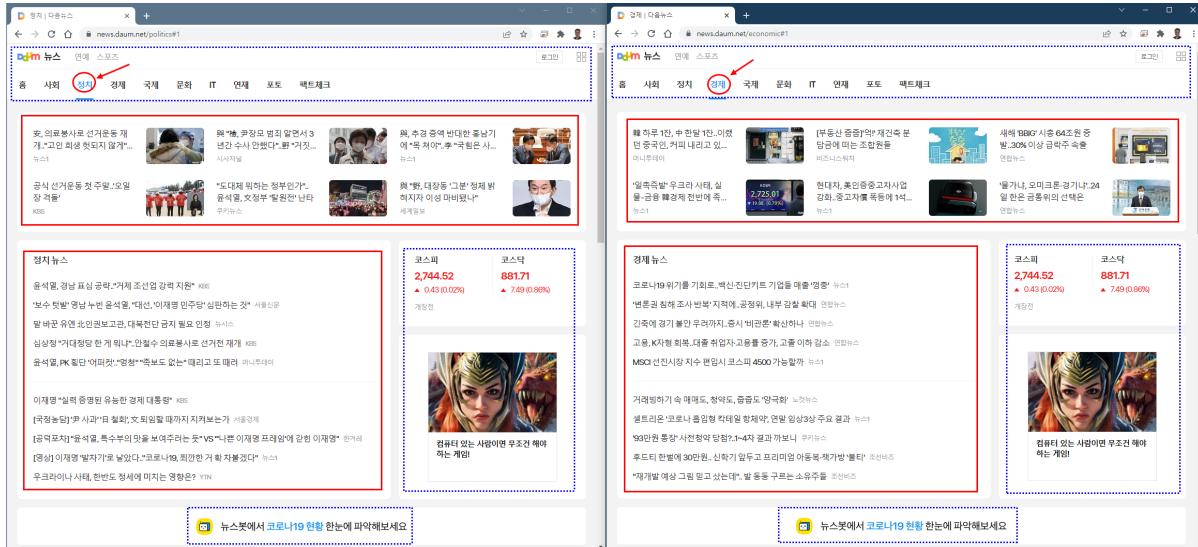


Fig. 3.63 브라우저에서 바뀌는 내용과 그대로인 내용

그림 Fig. 3.63 살펴볼까요? (만약 그림이 작게 보이면, 그림을 클릭해서 확대된 이미지로 보세요)

좌측 브라우저 내용은 빨간 동그라미 정치 클릭해서 나타난 화면입니다. 우측 브라우저는 경제를 클릭해서 나타난 화면입니다.

두 화면을 잘 살펴보면 내용이 바뀌는 부분이 있고, URL 주소가 달라져도 내용이 그대로 유지되는 부분이 있습니다.

서로 다른 정치와 경제 메뉴(URL)를 선택했을 때 내용이 변경된 부분은 빨간색 실선 사각형으로 표시했습니다. 내용이 똑같이 유지되는 부분은 파란색 점선 사각형으로 표시했습니다.

서로 다른 메뉴를 선택했으니 다른 내용이 보이는 것은 당연하겠죠? 정치 메뉴를 클릭하면 정치 관련 내용이 보여야 하고, 경제 메뉴를 선택하면 경제 관련 내용이 나오는게 당연합니다.

그런데 변하지 않는 부분들이 있죠?

위쪽 메뉴 부분은 변하지 않습니다. 동일한 뉴스 카테고리라면 사회, 정치, 경제, 국제, 문화, IT 등 세부 메뉴로 이동하더라도 동일한 구조를 유지하는게 좋습니다. 그림 Fig. 3.63에서 맨 꼭대기에 표시한 파란색 점선 사각형에 해당합니다. 이렇게 앞쪽(머리 부분)에 동일한 내용으로 고정되는 것을 헤더(Header)라고 부릅니다.

맨 아래 부분도 변하지 않습니다. 그림 Fig. 3.63 아래쪽의 파란 점선 사각형을 비교해 보세요. 내용이 동일하죠? 이렇게 아랫쪽에 변하지 않는 내용을 일관되게 보여주는 것을 풋터(Footer)라고 부릅니다.



Fig. 3.64 풋터(Footer) 사용 예 (청주대학교 홈페이지 footer)

그림 Fig. 3.64와 같이 일반적으로 홈페이지 안내, 연락 정보, 이메일 등의 공통 정보를 제공할 때 자주 사용합니다.

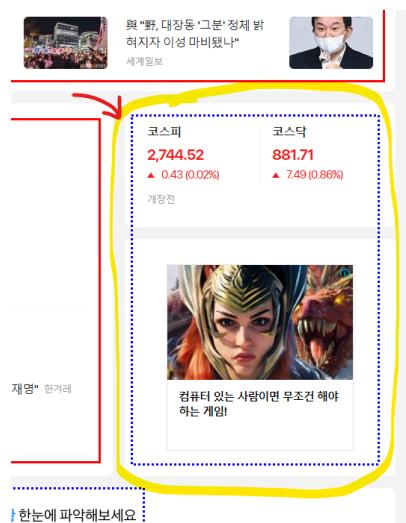


Fig. 3.65 풋터(Footer) 사용 예 (청주대학교 홈페이지 footer)

그림 Fig. 3.65와 같이 중앙 오른쪽에 있는 내용도 변하지 않습니다. 그림 Fig. 3.65은 그림 Fig. 3.63에서 각 브라우저 내용 중 내용이 유지되는 오른쪽 중앙 부분을 다시 잘라낸 그림입니다.

그림 Fig. 3.63 상에서는 주식시장 정보와 광고가 동일하게 보입니다. 이렇게 메인 컨텐츠(내용) 좌측 또는 우측에 배치된 것을 어사이드(Aside)라고 부릅니다.

진짜로 하고 싶은 이야기는 여기서부터입니다.

변하지 않는 영역 Header, Footer, Aside는 어떻게 코딩해 주는게 좋을까요? 물론 모든 페이지에 보이는 내용을 동일하게 하려면 템플릿 .html 파일에 똑같은 코드를 넣어주면 될 겁니다. 하지만 문제점이 바로 보이죠? 반복적인 <head> 태그를 집어 넣을 때와 동일한 단점이 생깁니다.

동일한 코드를 반복적으 복사해서 붙여 넣는 것도 거대한 삽질이지만, 내용을 추가/변경/삭제해야 할 경우에는 그 많은 코드를 일일히 뒤져가며 수정해 줘야 하는 더 큰 삽질이 기다리고 있는 것입니다.

이 경우도 공통되는 Header, Footer, Aside 부분은 하나의 템플릿 파일로 만들어 놓고, 변하는 내용은 공통 템플릿을 상속하여 그대로 보여주고 변하는 부분만 추가로 코딩하면 아주 좋습니다. 공통된 부분을 변경할 경우는 그냥 하나의 파일만 변경하면 그만입니다.

엄청난 삽질의 공포에서 개발자를 해방시켜 주는 것이 바로 템플릿 상속입니다.

3.10.3. 템플릿 상속 구현하기

Flask로 템플릿 상속을 구현하는 문법은 두 가지로 구분할 수 있습니다. Flask는 템플릿 엔진으로 Jinja를 사용합니다. 보다 구체적인 내용을 확인하고 싶다면 Jinja 공식 문서의 템플릿 상속 페이지 ([click](#)) 방문하기 바랍니다.

3.10.3.1. Base(기본) 템플릿 이해하기

첫째, 우선 기본이 되는 부모 템플릿 파일이 있어야 겠죠? 부모가 있어야 상속받을 자식들도 있는 거니까요... 부모 템플릿은 모든 웹페이지에 똑같은 내용을 담아 놓은 페이지를 말합니다.

다른 페이지의 기본(Base)이 되기 때문에 Base(기본) 템플릿이라고 부릅니다. 부모 템플릿을 상속받는 모든 자식(child) 템플릿은 부모의 내용을 동일하게 갖게 됩니다.

우리는 앞으로 부모 템플릿을 Base 템플릿이라는 용어로, Base 템플릿을 상속받은 템플릿은 Child 템플릿이라는 용어로 통일해서 부르도록 하겠습니다.

관습적으로 부모 템플릿 파일 이름은 `base.html`이라고 붙여 줍니다.

Base 템플릿을 작성하는 방법은 간단합니다. 우선 모든 파일에 공통으로 들어갈 내용을 작성합니다. 그리고 base 템플릿을 상속받은 child 템플릿에서 작성해야 할 시작 부분에 `{% block 블록이름 %}`, child 템플릿이 작성해야 할 마지막 부분에 `{% endblock %}`이라고 표시해 주면 그만입니다.

예상했던 것보다는 많이 간단하죠?

몇 가지 예제를 살펴보면 더 쉽게 이해될 것입니다.

우선 간단한 코드를 살펴보겠습니다.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} - My Webpage</title>
</head>
<body>
    <div id="content">{% block content %}{% endblock %}</div>
</body>
</html>
```

```
base.html U X
hello_cju > templates > base.html > ...
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <link rel="stylesheet" href="style.css" />
5       <title>{% block title %}{% endblock %} - My Webpage</title>
6   </head>
7   <body>
8       <div id="content">{% block content %}{% endblock %}</div>
9   </body>
10  </html>
```

Fig. 3.66 2개의 block을 포함하고 있는 base 템플릿 (파일명: base.html)

그림 Fig. 3.66를 보면 `{% block 블록이름 %}` ~ `{% endblock %}`로 둘러싸인 부분이 2군데 있습니다. 그림에서 노란색 사각형으로 표시한 부분입니다.

- `{% block title %}` ~ `{% endblock %}`: Block 이름을 `title`이라고 하고 이 부분은 상속받은 템플릿에서 채워라.
- `{% block content %}` ~ `{% endblock %}`: Block 이름을 `content`이라고 하고 이 부분은 상속받은 템플릿에서 채워라.

조금 더 복잡한 구조를 볼까요?

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} - My Webpage</title>
    {% endblock %}
</head>
<body>
    <div id="content">{% block content %}{% endblock %}</div>
    <div id="footer">
        {% block footer %}
        &copy; Copyright 2008 by
        <a href="http://domain.invalid/">you</a>.
        {% endblock %}
    </div>
</body>
</html>
```

그림 Fig. 3.67에는 다음과 같이 4개의 block이 있습니다. 그림 Fig. 3.66에서 2개의 block이 추가되었습니다.

- `{% block title %}` ~ `{% endblock %}`
- `{% block content %}` ~ `{% endblock %}`
- `{% block head %}` ~ `{% endblock %}`
- `{% block footer %}` ~ `{% endblock %}`

그림 Fig. 3.67에서 `title` block은 `head` block 내부에 있습니다. `<body>` 태그의 마지막 부분에 풋터(footer)를 넣기 위한 `footer` block도 추가된 것을 확인할 수 있습니다.

Block의 범위는 `{% block 블록이름 %}`에서 시작하여 가장 가까운 `{% endblock %}`까지입니다.

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  {% block head %}
5  <link rel="stylesheet" href="style.css" />
6  <title>{% block title %}{% endblock %} - My Webpage</title>
7  {% endblock %}
8  </head>
9  <body>
10 <div id="content">{% block content %}{% endblock %}</div>
11 <div id="footer">
12 <% block footer %>
13 &copy; Copyright 2008 by
14 <a href="http://domain.invalid/">you</a>.
15 <% endblock %>
16 </div>
17 </body>
18 </html>

```

Fig. 3.67 4개의 `block`을 포함하고 있는 base 템플릿

Block 내부에 여러 개의 block이 포함될 수 있고, block 내부에 block이 포함될 수도 있습니다. 모두 가능합니다.

3.10.3.2. Child(자식) 템플릿 이해하기

부모 템플릿을 상속받아 똑같은 내용을 뿌려준 다음, 나름대로 변하는 부분을 작성하는 템플릿이 있어야 겠죠? Base 템플릿을 상속받는 템플릿을 `child` 템플릿이라고 부릅니다.

우선 자식 템플릿 파일을 하나 만듭니다. 저는 빈 템플릿 파일을 만들고 이름을 `child.html`이라고 붙여 주었습니다.

Child 템플릿에 어떤 base 템플릿을 상속받을지 알려주는 문법은 `{% extends "부모템플릿 파일명" %}`입니다. 우리는 `base.html`을 만들었으니 다음과 같이 써주면 됩니다.

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

[Fig. 3.67](#)에서 지정한 4개 block에 대하여 각각 내용을 작성합니다. 내용을 작성할 때는 우리가 base 템플릿에서 지정한 이름을 활용하면 됩니다.

다음에 각각을 작성한 내용을 예시 코드를 보겠습니다.

```

{% extends "base.html" %}

{% block title %}Index{% endblock %}

{% block head %}
  {{ super() }}
  <style type="text/css">
    .important { color: #336699; }
  </style>
{% endblock %}

{% block content %}
  <h1>Index</h1>
  <p class="important">
    Welcome to my awesome homepage.
  </p>
{% endblock %}

```

위 코드에서 4개의 block에 대한 내용을 작성해 주었습니다. `child.html`은 먼저 `base.html`을 만들고 빈 자리(block)는 `child.html`에서 내용을 채웁니다.

그림 [Fig. 3.68](#)는 base 템플릿 `base.html`과 child 템플릿 `child.html`의 관계를 그림으로 표현한 것입니다.

base 템플릿의 각 블록이 child 템플릿에서 어떻게 채워지는지 살펴볼까요? 먼저 `base.html`의 내용이 그대로 렌더링 됩니다. 이후에 `base.html`에서 block으로 지정한 부분들이 채워집니다.

- `head` block: 부모의 내용을 그대로 가져옵니다. `{{ super() }}`를 사용하였습니다. 이후 스타일을 지정해 주었습니다. 스타일 `.css` 파일을 만들지 않고

```

base.html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   {% block head %}
5     <link rel="stylesheet" href="style.css" />
6   <title>{% block title %}{% endblock %} - My Webpage</title>
7   {% endblock %}
8 </head>
9 <body>
10  <div id="content">{% block content %}{% endblock %}</div>
11  <div id="footer">
12    {% block footer %}
13      &copy; Copyright 2008 by
14      <a href="http://domain.invalid/">you</a>.
15    {% endblock %}
16 </div>
17 </body>
18 </html>
19

```

```

child.html
1 {% extends "base.html" %}
2
3 {% block title %}Index{% endblock %}
4
5 {% block head %}
6   {{ super() }}
7   <style type="text/css">
8     .important { color: #336699; }
9   </style>
10  {% endblock %}
11
12  {% block content %}
13    <h1>Index</h1>
14    <p class="important">
15      Welcome to my awesome homepage.
16    </p>
17  {% endblock %}
18
19

```

Fig. 3.68 4개의 **block**을 포함하고 있는 base 템플릿

코드 내부에서 직접 스타일을 지정하는 태그는 `<style>` 입니다.

- **title** block: 내용을 `Index`로 대체합니다.
- **content** block: 아래 코드 내용으로 대체됩니다.

```

<h1>Index</h1>
<p class="important">
  Welcome to my awesome homepage.
</p>

```

- **footer** block: `child.html`에서 별도의 내용을 작성하지 않았기 때문에 `base.html` 내용을 그대로 사용합니다. 만약 `child.html`에서 `{% block footer %}` 내용을 작성했다면 `child` 템플릿에서 작성한 내용으로 대체됩니다.

3.10.4. 우리가 만든 코드에 적용하기

템플릿 상속에서 [Base\(기본\) 템플릿 이해하기](#)와 [Child\(자식\) 템플릿 이해하기](#)의 개념 잡기가 끝났습니다.

이제 지금까지 만들었던 템플릿 `.html` 파일들에 템플릿 상속의 개념을 적용해 보겠습니다.

먼저 `base.html`을 만들어야겠죠?

3.10.4.1. `base.html` 만들기

`templates/` 디렉토리에 빈 파일을 하나 만들고 이름을 `base.html`이라고 붙여 주겠습니다. 그리고 아래와 같이 코드를 작성합니다.

```

<!-- 파일 이름: /templates/base.html --&gt;

&lt;!DOCTYPE html&gt;
&lt;html lang="ko"&gt;

&lt;head&gt;
  &lt;meta charset="UTF-8"&gt;
  &lt;meta name="viewport" content="width=device-width, initial-scale=1.0"&gt;

  &lt;!-- title 블록: 원도우 창 이름은 child 템플릿에 맞게 지정 --&gt;
  &lt;title&gt;{% block title %}{% endblock %}&lt;/title&gt;

  &lt;!-- Bootstrap CDN의 CSS 접속/사용을 위한 코드 --&gt;
  &lt;link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jiW3" crossorigin="anonymous"&gt;
  &lt;script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka75k0gMtzz2M1QnikT1wXgYsOg+OMhp+I1RH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"&gt;&lt;/script&gt;
&lt;/head&gt;

&lt;body&gt;
  &lt;!-- content 블록: 각 원도우 내용은 해당 템플릿에서 작성 --&gt;
  {% block content %}{% endblock %}
&lt;/body&gt;

&lt;/html&gt;
</pre>

```

위 코드를 상속받는 child 템플릿에서 작성할 부분은 각 원도우 창 제목을 지정하는 `title` 블록과 `<body>` 태그의 내용을 채워줄 `content` 블록입니다.

지금까지 우리가 만든 템플릿 파일은 3개이며 `/templates/question/` 디렉토리에 저장되어 있습니다. 아래 그림 Fig. 3.69 참고하면 금방 기억이 날 것입니다.

```
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-s
      <!-- title 블록: 원도우 창 이름은 child 템플릿에 맞게 지정 -->
      <title>{% block title %}{% endblock %}</title>
      <!-- Bootstrap CDN의 CSS 접속/사용을 위한 코드 -->
      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/di
      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/di
    </head>
    <body>
      <!-- content 블록: 각 원도우 내용은 해당 템플릿에서 작성 -->
      {% block content %}{% endblock %}
    </body>
</html>
```

Fig. 3.69 `base.html`을 상속할 우리의 템플릿 파일 구조

그림 Fig. 3.69 우측의 파일 탐색기(File Explorer) 창을 살펴보면 우리가 만들었던 템플릿 `.html` 파일들은 `/templates/question/` 디렉토리에 잘 담겨져 있습니다. 그리고 우리가 조금전에 만든 템플릿 `base.html`도 `templates/` 디렉토리에 무사히 보관되어 있습니다.

이제는 차례대로 템플릿을 고쳐 보겠습니다.

3.10.4.2. `question_list.html` 업그레이드

`/templates/question/question_list.html`을 다음과 같이 수정합니다.

기존 `question_list.html`의 내용 대부분은 정확히 그대로입니다.

다만, 이전에 Bootstrap을 사용하게 위해 포함했던 코드는 `base.html`에서 공통적으로 지정했기 때문에 모두 삭제하였습니다.

원도우 창 이름을 지정하는 `title` 블록을 이용해 `질문 목록`으로 지정하고, `content` 블록을 이용해 예전에 있던 내용을 그대로 유지했습니다.

```

{% extends 'base.html' %}

{% block title %}질문 목록{% endblock %}

{% block content %}
<!-- 기존 질문 리스트를 div 태그로 둘러싸고
클래스 이름이 'container my-3'의 디자인을 적용 -->


<!-- 질문 목록을 표로 만들기 위한 table 태그
    추가하고 CSS 디자인은 'table-dark' 스타일 지정 -->
    <table class="table">
        <!-- 표 헤더(컬럼명)를 지정, 디자인은 'table-dark' 스타일 지정 -->
        <thead>
            <tr class="table-dark">
                <th>번호</th>
                <th>제목</th>
                <th>작성일시</th>
            </tr>
        </thead>
        <!-- 표 내용(tbody: table body) // 질문 목록을 렌더링 -->
        <tbody>
            {% if question_list %}
                {% for question in question_list %}
                    <tr>
                        <td>{{ loop.index }}</td>
                        <td>
                            <a href="{{ url_for('question.detail', question_id=question.id) }}">
                                {{ question.title }}</a>
                        </td>
                        <td>{{ question.create_date }}</td>
                    </tr>
                {% endfor %}
            {% else %}
                <p>질문이 없습니다.</p>
            {% endif %}
        </tbody>
    </table>
    <!-- 질문 등록하기 버튼에 CSS 'btn btn-primary' 스타일 지정 -->
    <a href="{{ url_for('question.create') }}" class="btn btn-primary">
        질문 등록하기
    </a>


{% endblock %}

```

3.10.4.3. `question_detail.html` 업그레이드

`/templates/question/question_detail.html`을 다음과 같이 수정합니다. 기존 `question_detail.html`의 내용 대부분은 정확히 그대로입니다.

만약, 이전에 Bootstrap을 사용하게 위해 포함했던 코드는 `base.html`에서 공통적으로 지정했기 때문에 모두 삭제하였습니다.

윈도우 창 이름을 지정하는 `title` 블록을 이용해 질문 상세내용으로 지정하고, `content` 블록을 이용해 예전에 있던 내용을 그대로 유지했습니다.

```

{% extends 'base.html' %}

{% block title %}질문 상세내용{% endblock %}

{% block content %}
<!-- container 클래스로 답변 전체를 감싸기 -->


<h2 class="border-bottom py-2">{{ context.title }}</h2>
    <div class="card my-3">
        <div class="card-body">
            <div class="card-text" style="white-space: pre-line;">{{ context.contents }}</div>
            <div class="d-flex justify-content-end">
                <div class="d-flex p-2 bd-highlight">
                    작성시간: {{ context.create_date }}
                </div>
            </div>
        </div>
    </div>
</div>

<h5 class="border-bottom my-3 py-2">{{ context.answer_set|length }}개의 답변이 있습니다.</h5>

{% for answer in context.answer_set %}
<div class="card my-3">
    <div class="card-body">
        <div class="card-text" style="white-space: pre-line;">{{ answer.contents }}</div>
        <div class="d-flex justify-content-end">
            <div class="d-flex p-2 bd-highlight">
                작성시간: {{ context.create_date }}
            </div>
        </div>
    </div>
</div>
{% endfor %}

<!-- form 에러 처리 -->
{% for field, errors in form.errors.items() %}
<div class="alert alert-danger" role="alert">
    <strong>{{ form[field].label }}</strong>: {{ ', '.join(errors) }}
</div>
{% endfor %}

<!-- 댓글 입력 필드 -->
<form action="{{ url_for('answer.create', question_id=context.id) }}" method="post" class="my-3">
    {{ form.csrf_token }}
    <div class="form-group">
        <textarea name="content" id="content" class="form-control" rows="10"></textarea>
    </div>

    <div class="form-group my-3">
        <input type="submit" value="댓글등록" class="btn btn-primary" />
    </div>
</form>
</div>
{% endblock %}


```

3.10.4.4. `question_form.html` 업그레이드

이제 마지막 템플릿 파일을 업그레이드 해겠습니다.

`/templates/question/question_form.html`을 다음과 같이 수정합니다. 기존 `question_form.html`의 내용 대부분은 정확히 그대로입니다.

다만, 이전에 Bootstrap을 사용하게 위해 포함했던 코드는 `base.html`에서 공통적으로 지정했기 때문에 모두 삭제하였습니다.

윈도우 창 이름을 지정하는 `title` 블록을 이용해 `질문` 입력으로 지정하고, `content` 블록을 이용해 예전에 있던 내용을 그대로 유지했습니다.

```

{% extends 'base.html' %}

{% block title %}질문 입력{% endblock %}

{% block content %}
<div class="container">

    <!-- 오류 내용을 표시하는 코드 추가-->
    <form method="post" class="my-3">
        {% for field, errors in form.errors.items() %}
            <div class="alert alert-danger" role="alert">
                {{ form[field].label }}: {{ ', '.join(errors) }}
            </div>
        {% endfor %}
    </form>

    <h5 class="my-3 border-bottom pb-2">질문 등록</h5>

    <form action="{{ url_for('question.create') }}" method="post">
        {{ form.csrf_token }}
        <div class="card my-3">
            <div class="card-body">
                <div class="card-text" style="white-space: pre-line;">제목</div>
                <div class="input-group mb-3">
                    <input type="text" class="form-control" name="title" id="title" size="100" value="{{ form.title.data or '' }}"/>
                </div>
            </div>
        </div>

        <div class="card my-3">
            <div class="card-body">
                <div class="card-text" style="white-space: pre-line;">내용</div>
                <div class="form-floating">
                    <div class="d-flex p-2 bd-highlight">
                        <textarea class="form-control" placeholder="질문 내용을 입력하세요" id="floatingTextarea2" style="height: 300px" value="{{ form.contents.data or '' }}"/>
                    <!-- <textarea name="contents" id="contents" cols="30" rows="10" value="{{ form.contents.data or '' }}"/></textarea> -->
                </div>
            </div>
        </div>
    </div>
    <div class="form-group my-3">
        <button type="submit" class="btn btn-primary">저장하기</button>
    </div>
    </form>
</div>
{% endblock %}

```

이상으로 모든 템플릿 상속에 대한 공부를 마치도록 하겠습니다.

3.11. Flask 기본기 마치며...

우리는 어디까지 와 있는 것일까요?

우리 모두 열심히 달려왔죠?

여러분들은 아래와 같이 기본기 10가지를 모두 정복하였습니다.

❶ Flask 웹 시스템 구축을 위한 10가지 기본기

1. [프로젝트 기본 구조](#) \(\to\) Clear!
2. [Application Factory 패턴](#) \(\to\) Clear!
3. [Blueprint 클래스 활용](#) \(\to\) Clear!
4. [ORM 모델 완벽 이해](#) \(\to\) Clear!
5. [질문 게시판 만들기](#) \(\to\) Clear!
6. [게시판 댓글 구현](#) \(\to\) Clear!
7. [질문 & 댓글 등록 - Flask Form 활용](#) \(\to\) Clear!
8. [예쁘게 - CSS 적용](#) \(\to\) Clear!
9. [더 예쁘게 - Bootstrap 활용](#) \(\to\) Clear!
10. [HTML 구조와 Template 상속](#) \(\to\) Clear!

끌이 안보이던 **Flask** 기본기를 마스터 했습니다. 이제 여러분들은 기본적인 웹 시스템을 구축할 수 있는 충분한 능력을 가졌습니다.

이 페이지를 읽는 사람들은 수많은 성취감을 느꼈을 것입니다.

축하드립니다. 짹짜짜!!!



Fig. 3.70 Flask 기본기를 마스터한 당신에게 보내는 박수 ^^

Image source: <https://media.istockphoto.com/photos/what-a-great-achievement-picture-id641853864>

어떤 시스템이든, 알고리즘이든, 인공지능이든... 최종적으로는 사용자에게 적절한 서비스를 제공해야 합니다. 그 방법 중 가장 일반적인 것은 웹 기반으로 서비스를 구축하는 것입니다.

이 점에서 여러분들은 개발자로서 또 하나의 강력한 무기를 갖게 된 셈입니다.

하지만, 기본기는 기본기입니다. 웹 시스템은 다양한 기술과 복잡한 메커니즘이 총 집합한 시스템입니다. 더 많은 것들을 배워야 합니다.

중급, 고급 기술은 다른 장에서 다룰 예정입니다.

여러분들의 아름다운 도전이 계속 이어지기를 바랍니다.

4. Appendix List

Flask를 공부하면서 추가로 알아두면 좋은 내용을 별도로 구분하여 정리하였습니다.

① Flask 첨부 목록

1. [Monolithic vs. MSA](#)

4.1. Monolithic vs. MSA

Flask를 배우다 보면 선배 개발자들이 “플라스크는 MSA라는 것은 알고 있지?”라는 말을 자주 해줍니다. 그런데 처음 웹 프레임워크를 배우는 학생들은 **MSA**라는 용어 자체가 생소한 경우가 많습니다.

MSA라는 말을 처음 들어보는 것도 멘붕인데 선배 개발자들이 이런 말까지 합니다. “Django는 Monolithic 아키텍처야. 그래서 Django와 Flask는 근본적으로 달라. 알지?”

이런 말을 처음 접하게 된다면 적지 않게 당황하게 됩니다. 하지만 알고 보면 별로 복잡하지 않은 개념입니다. Flask를 배우는 김에 MSA와 Monolithic(모놀리식)의 개념에 대해 간단히 살펴보도록 하겠습니다.

Monolithic과 MSA에 대한 설명은 [\[MSA\] Monolithic Architecture VS Micro Service Architecture](#) 참조하여 재정리 하였으니 관심있는 독자는 해당 블로그를 방문하기 바랍니다.

4.1.1. Monolithic Architecture

Monolithic(모놀리식)을 네이버 영어사전에서 찾아보면 '(조직, 단결 등이) 단일체인, 한 덩어리로 뭉친'이라고 나옵니다. 사전적 의미 그대로 모놀리식 아키텍처는 어떤 서비스(또는 프로젝트)를 하나의 큰 덩어리로 된 어플리케이션(Application, 이하 앱)으로 만든다는 뜻입니다. 우리가 일반적으로 생각하는 방식과 비슷합니다.

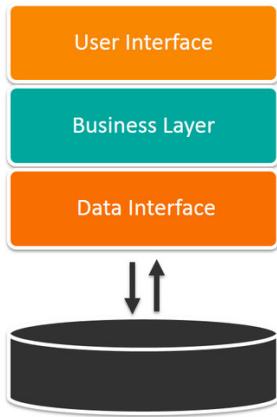


Fig. 4.1 모놀리식 아키텍처 (이미지 출처: [sangmin7476](#))

모놀리식 아키텍처는 장단점이 존재합니다. 장점과 단점에 대해 간단히 알아보겠습니다.

- 장점
 - 로컬 머신(예: 개인 컴퓨터)에서 편리하게 개발 \(\backslash\backslash to\backslash\) 한 덩어리로 만들기 때문에 분산 개발 환경이 불필요합니다.
 - 통합된 시나리오를 활용해서 전체 기능을 쉽게 테스트할 수 있습니다.
 - 배포가 간단합니다(하나의 앱만 배포하면 끝)
- 단점
 - 큰 덩어리로 되어 있기 때문에 유지보수(코드 수정 및 추가)가 어렵습니다.
 - 필요에 따라 자원(CPU, Memory, Storage 등)을 적절히 분배하기 어렵습니다.
 - 신속한 업데이트가 어렵습니다.
 - 새로운 기술이 개발되었을 경우 적용하기 어렵습니다(유지보수가 어려운 것과 같은 이치)
 - 부분적 장애가 발생해도 전체 시스템 작동이 안될 수 있습니다.
 - Scale out이 어렵습니다.

❶ 스케일 아웃(scale out)에 대한 설명

갑자기 이용자(접속자)가 늘어나면 서버를 몇 대 늘여서 서비스 능력을 높여줘야 합니다. 이렇게 서버를 증설하는 것을 **scale out**한다고 말합니다. 앱이 기능별로 분리가 되어 있다면 증설이 쉽습니다. 청주대 온라인 강의 시스템에 학생들이 몰리면 온라인 강좌만 담당하는 서버만 추가하면 됩니다. 만약 청주대의 모든 시스템이 하나의 큰 덩어리로 코딩되어 있으면 scale out 하기 매우 어려워집니다.

모놀리식 구조는 한 덩어리만 생각하면 되므로 단순합니다. 그래서 개발도, 테스트도, 배포도 편리합니다. 하지만 개발 규모가 커지면 이런 단순함이 독이 됩니다. 왜냐하면 모듈간 의존성이 높아지게 되고, 개발자들은 코드 이해가 어렵고, 하나의 프로젝트에 많은 개발자가 투입되어 의사소통도 어려워집니다. 그래서 대안으로 등장한 것이 **MSA**입니다. 이제 MSA Architecture에 대해 간단히 살펴보도록 하겠습니다.

4.1.2. MSA Architecture

MSA는 'Micro Service Architecture'의 약어입니다. 단어 그대로 아주 작은 서비스를 위한 아키텍처라는 말입니다. 기술적(Technical)으로 말하면 하나의 큰 앱을 작은 앱으로 나누어 만드는 방법입니다. 우리가 직관적으로 생각하는 앱은 전체가 한 덩어리로 된 하나의 앱으로 만드는 것입니다.

물론 MSA도 장단점이 있습니다.

- 장점
 - 빌드 및 테스트 시간이 줄어듭니다. 이미 만들어진 서비스는 다시 빌드하지 않고 새롭게 추가되거나 수정된 부분만 처리하기 때문에 시간을 아낄 수 있습니다.
 - 서비스를 분리해서 개발하므로 유연하게 기술을 적용할 수 있습니다. 인증 기능은 django auth로 구현하고, 채팅 기능은 node를 이용해서 구현하는 예를 들 수 있습니다.
 - 손쉽게 scale out 가능합니다.
 - 서비스 사이에 결합도가 낮아져 유지보수가 편리하고 시스템을 안정적으로 운영할 수 있습니다.
- 단점

Microservices Architecture

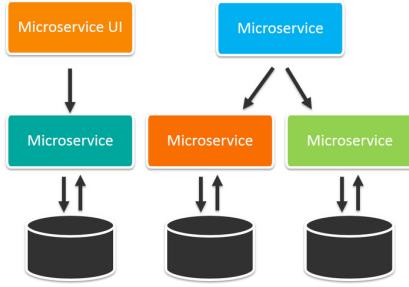


Fig. 4.2 MSA 아키텍처 (이미지 출처: [sangmin7476](#))

- 하나의 서비스가 다른 서비스를 호출할 경우 네트워크 통신이 추가적으로 발생하므로 속도가 느려질 수 있습니다.
- 서버를 나누고 각각의 서버가 담당하는 앱이 있기 때문에 서버간 데이터베이스 처리를 해야하는 경우 트랜잭션 처리하는 알고리즘을 추가로 작성해야 합니다.
- 서버가 증가하기 때문에 로깅, 모니터링, 배포, 테스트, 데이터베이스 관리 등 추가적인 작업이 필요합니다.

4.1.3. 모놀리식과 MSA, 어떤 것이 좋은 건가요?

앞에서 살펴본 것과 마찬가지로 어떤 아키텍처든 장단점이 있기 마련입니다. 모놀리식과 MSA도 각각 장단점이 있습니다. 따라서 어떤 것이 좋은 아키텍처라고 쭉 짚어서 말할 수 없습니다. MSA가 더 좋을 것 같지만 아직도 많은 서비스가 모놀리식 아키텍처를 적용하고 있는 것으로 알려져 있습니다.

아키텍처는 것은 다양한 개발 경험과 운영 노하우를 바탕으로 선택해야 합니다. 아키텍처를 선택하거나 새롭게 고안하는 것을 아키텍처 설계(Architecture Design)라고 합니다. 소프트웨어공학 측면에서 볼 때 '상위 설계'에 해당합니다. 이런 아키텍처 설계를 담당하는 사람을 아키텍처 설계자 (Architecture Designer) 또는 소프트웨어 시스템 설계자 (Software System Designer)라고 부릅니다.

By Giseop Noh

© Copyright Prof. Giseop Noh in Cheongju University.

본 책자에 적용되는 라이센스 [MIT License](#).