



Django

Python 기반의 웹 프레임워크

Contents

1. Web 개요 구조
 - HTTP
 - server & client 구조
2. Django 개요
 - Django 소개 및 설치
 - Hello Project
 - Pattern
3. Todo 일정관리
 - 프로젝트 구성
 - 프로젝트, app 생성
 - MVC
 - 일정 등록
 - 일정 목록
 - 일정 삭제
4. Todo Refactoring
 - Templates Composite View Pattern
 - Django Form
 - Static 파일 관리



01

Web 개요 및 구조

- HTTP
- Server & Client 구조

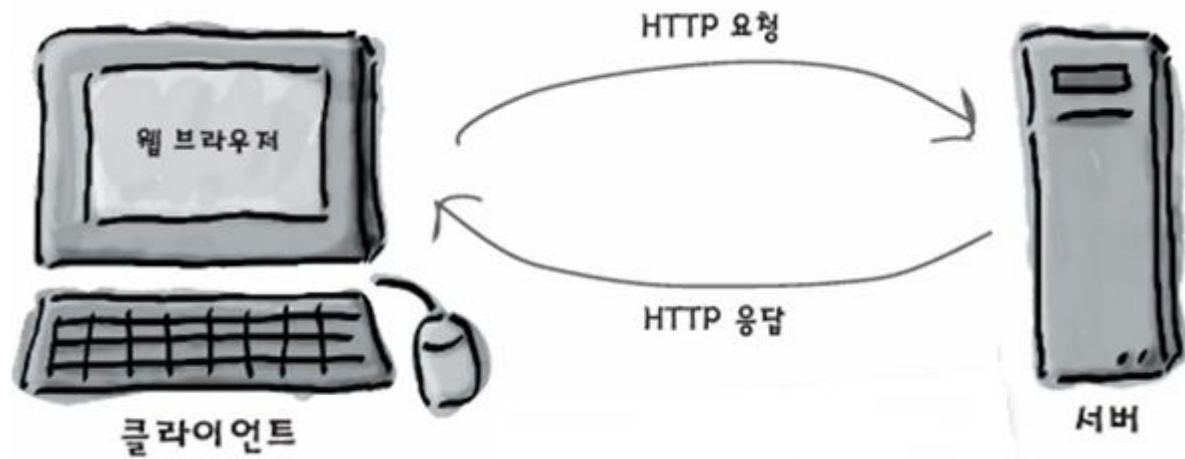
HTTP - 인터넷과 WWW

- 인터넷은 TCP/IP 기반의 네트워크가 전세계적으로 확대되어 하나로 연결된 'network of network'
- 인터넷 = www가 아님
- www는 인터넷 기반의 서비스 중 하나

이름	프로토콜	포트	기능
www	http	80	웹 서비스
Email	SMTP/POP3/IMAP	25/110/114	이메일 서비스
FTP	ftp	21	파일 전송 서비스
telnet	telnet	23	원격 로그인
DNS	DNS	83	도메인 이름 변환 서비스
News	NNTP	119	인터넷 뉴스 서비스

HTTP - HTTP 프로토콜

- HTTP(HyperText Transfer Protocol)
 - ✓ 프로토콜: 네트워크에 연결된 컴퓨터가 서로 통신(대화)하기 위한 규약
 - ✓ HTTP는 www 서비스를 위한 통신 규약
 - ✓ 웹 서버와 클라이언트는 HTTP를 이용해 통신
- TCP/IP 를 기반으로 하여 웹에서 사용하는 프로토콜로서 요청(Request) 과 응답(Response) 데이터를 전송하는 방식



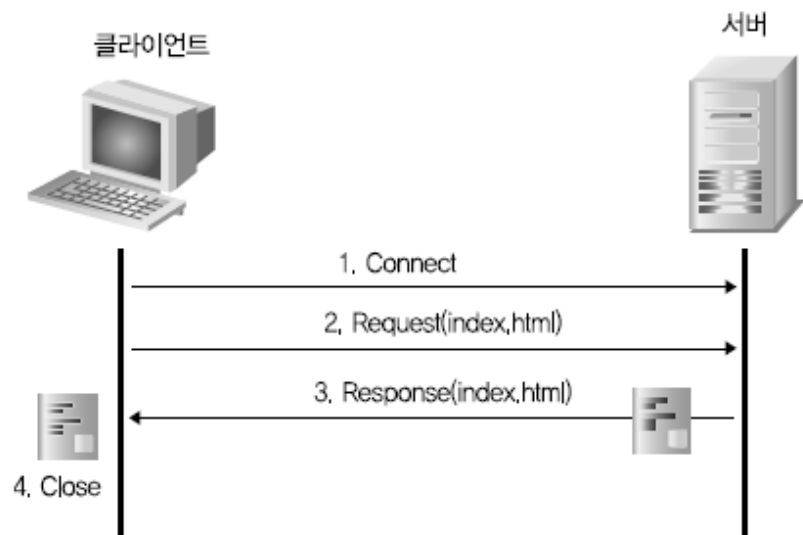
출처:Head First

HTTP - HTTP 프로토콜

- HTTP
- 인터넷에서는 FTP, Telnet, HTTP, SMTP, POP 등 여러 종류의 프로토콜이 사용됨
- 웹에서 가장 많이 사용하는 프로토콜인 HTTP(Hypertext Transfer Protocol)는 문서들 간의 상호 연결을 통해 다양한 텍스트, 그래픽, 애니메이션을 화면에 보여주고, 사운드를 재생할 수 있게 해줌
- HTTP의 기본 개념
- 서울에서 부산까지 차로 이동할 때 반드시 서울 톨게이트를 거쳐 고속도로로 나가듯이 웹을 통해 전 세계의 수많은 정보를 탐색하려면 HTTP를 이용해야 함
- HTTP에 관한 RFC 문서
- HTTP 1.0: RFC <ftp://ftp.ietf.org/rfc/rfc1945.txt>
- HTTP 1.1: RFC <ftp://ftp.ietf.org/rfc/rfc2616.txt>

HTTP - HTTP 프로토콜

- HTTP는 버전 0.9부터 사용되었으며, 0.9 버전의 HTTP는 서버에서 단순히 읽기 기능만 지원



1 먼저 클라이언트가 웹 브라우저를 이용하여 서버에 연결 요청

연결 요청을 받은 서버는 요청한 클라이언트에 대해 서비스 준비

2 클라이언트는 읽고자 하는 문서를 서버에 요청

3 서버는 웹 문서 중 클라이언트가 요청한 문서를 전달해 주고 연결 종료

[그림] HTTP 0.9를 이용하여 서버에 연결하기

- [그림]에서 설명한 기본 연결 기능은 HTTP의 버전에 관계없이 동일하지만, 기능이 많이 부족했던 HTTP 0.9는 그리 오래 사용되지 못했음. 현재 웹에서 사용하는 HTTP는 1.0과 1.1 버전
- HTTP 1.0 : 인터넷이 활성화된 시점인 1996년 5월 완성(메소드는 GET, HEAD, POST 방식 지원)
- HTTP 1.1 : 2001년 공식 발표(메소드는 OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT 방식 지원)

HTTP – Request

- Request 패킷에는 GET, POST, PUT, DELETE, OPTIONS, HEAD 등의 메서드가 있음

```
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Encoding: gzip, deflate
Cookie: HSID=AaxlkKoV2snlEi6UQ; SSID=AAYVu_evC0Tiu3aVc;
APISID=JEWp0eojRTLftYKJ/ACgEWh_0mL8Li_-fl;
SAPISID=kabRBO-uT0ebDcfc/A2byDX-FwN649tHw
Host: www.google.com
Connection: Keep-Alive
Accept-Language: ko-KR
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)
```

- 첫 번째 줄: GET / HTTP/1.1
 - ✓ HTTP 전송 방법 : 웹 서버로부터 자료를 가져오는 기능을 하는 GET 또는 POST를 많이 사용
GET 메소드는 별도의 메시지 바디를 필요로 하지 않음
 - ✓ 요청된 URL : 웹 서버에 있는 자료를 요청할 때 사용되는 경로
 - ✓ HTTP 버전 : 인터넷에서 가장 일반적으로 사용되는 HTTP 버전은 1.0과 1.1임
대부분의 브라우저는 초기값으로 1.1을 사용 → 1.1은 1.0과 달리 요청이 강제적

HTTP – Request

- Request 패킷 – POST 방식
 - ✓ URL에 요청 데이터를 전달하지 않고, HTTP의 헤더 영역이 아닌 바디 영역에 소켓을 이용하여 데이터 전송
 - ✓ 위 예의 '?hi_id=363&name=hong'과 같은 부분이 URL에 나타나지 않고 POST Body에 포함 됨
 - ✓ 인수 값을 URL을 통하여 전송하지 않기 때문에 다른 사람이 링크를 통해 해당 페이지를 볼 수 없음.
 - ✓ POST 방식은 보내려는 인자값이 URL을 통해 노출되지 않기 때문에 보안 측면에서 GET 방식 보다 안전한 편임

```
POST / HTTP/1.0
Accept: */*X-CI: 126323033
X-AT: OVERNET
X-GO: 1;KR;842;9530
X-DM: www.google.co.kr
X-SP: 762
Host: dr1.webhancer.com
Content-Length: 0
Pragma: no-cache
Connection: Close
```

HTTP – Request

- Request 패킷 – 기타
 - ✓ HEAD : 서버 쪽 데이터를 검색하고 요청하는 데 사용
 - ✓ OPTIONS : 자원에 대한 요구/응답 관계에서 관련된 선택 사항에 대한 정보를 요청할 때 사용
이를 통해 클라이언트는 어느 것을 선택할지 결정할 수 있으며, 자원과 관련된 필요 사항을 결정할 수 있음
아울러 서버의 수행 능력에 대해서도 알아볼 수 있음
 - ✓ PUT : 메시지에 포함되어 있는 데이터를 지정한 URI 장소에 지정된 이름으로 저장
 - ✓ DELETE : URI에 지정되어 있는 자원을 서버에서 지울 수 있게 함
 - ✓ TRACE : 요구 메시지의 최종 수신처까지 루프백 검사용으로 사용
즉 클라이언트가 보내는 요구 메시지가 거쳐가는 프록시나 게이트웨이의 중간 경로와 최종 수신 서버에 이르는 경로를 알아내는 데 사용

HTTP – Request

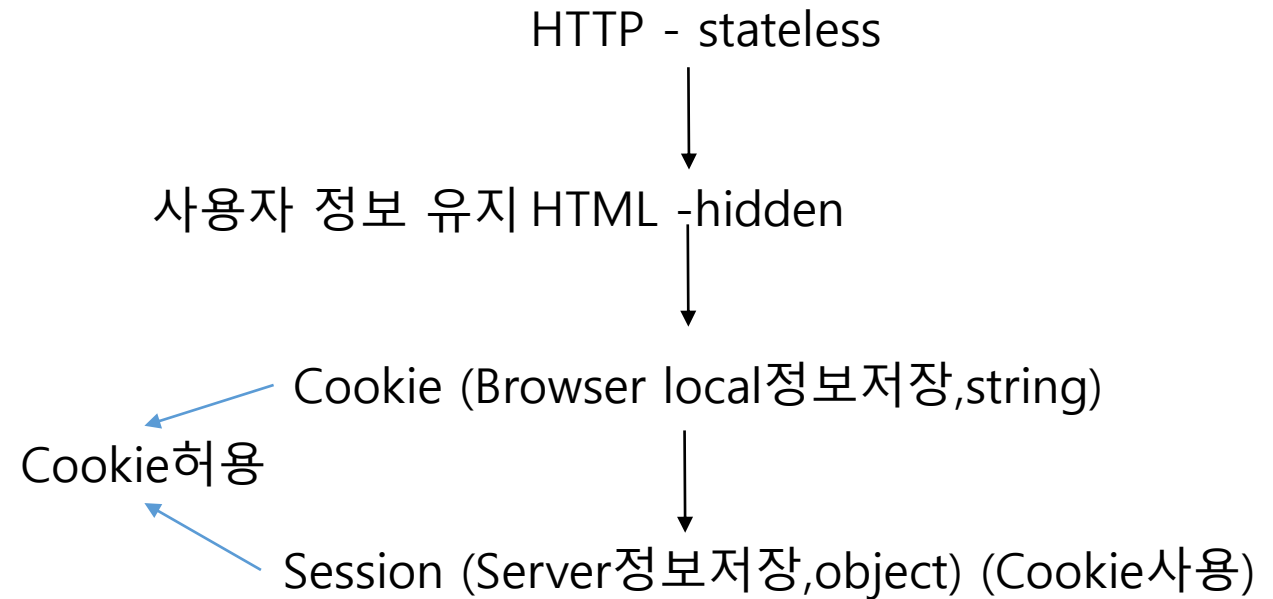
• Session Management

- ✓ Http stateless라 응답속도는 빠르
나 사용자정보를 저장할 수 없음
- ✓ 대안
 - html hidden tag
 - Cookie
 - Session
 - URL Rewriting

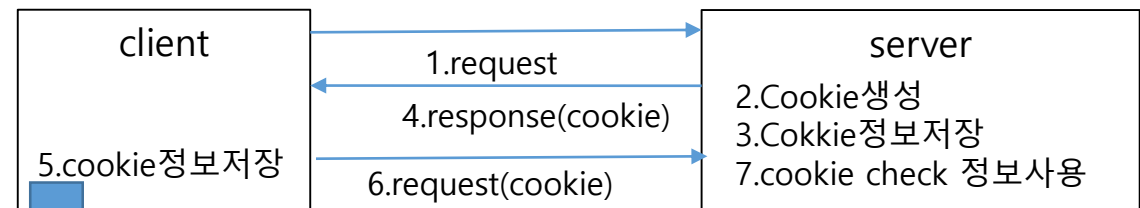
Cookie는 Client에 정보를 저장해서 보안 취약
하고 네트워크 스템으로 문자열정보 저장 단점

-보안 HttpOnly, SecureCookie 사용

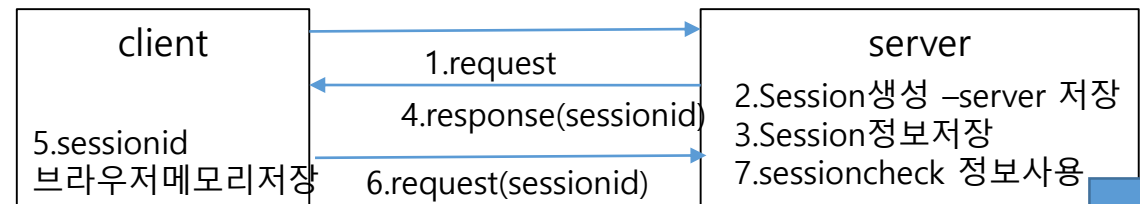
Session 사용 권고



Cookie



Session



HTTP – Request(Cookie)

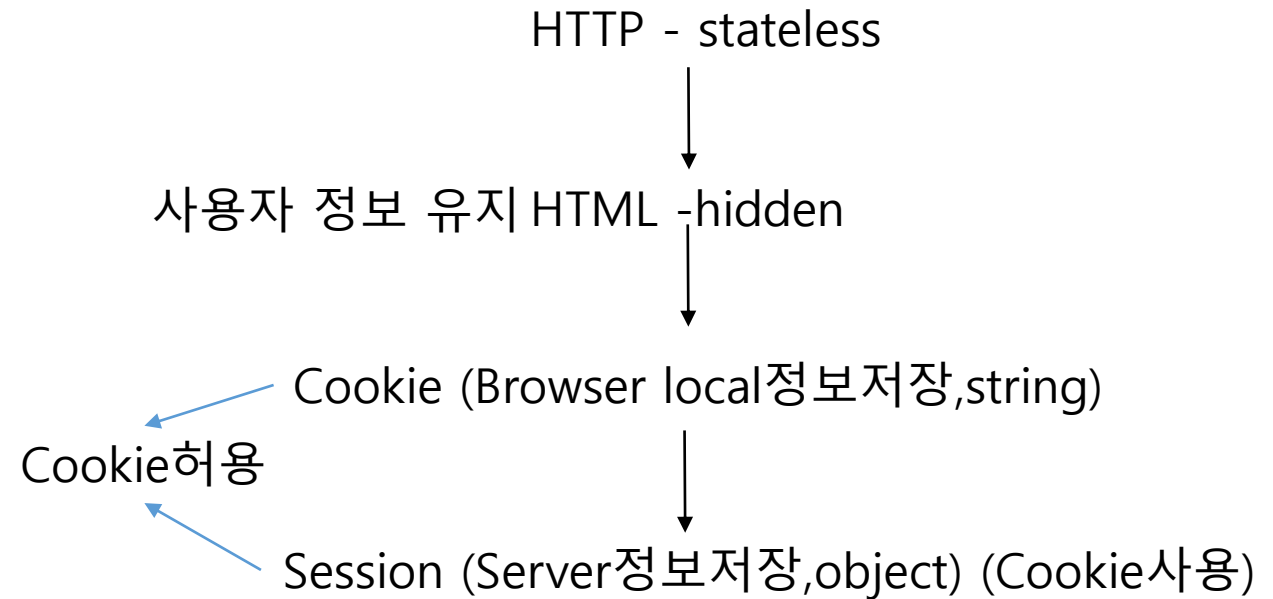
• Session Management

- ✓ Http stateless라 응답속도는 빠르
나 사용자정보를 저장할 수 없음
- ✓ 대안
 - html hidden tag
 - Cookie
 - Session
 - URL Rewriting

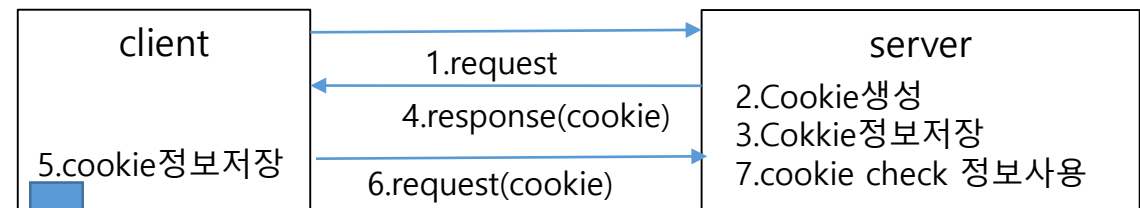
Cookie는 Client에 정보를 저장해서 보안 취약
하고 네트워크 스템으로 문자열정보 저장 단점

-보안 HttpOnly, SecureCookie 사용

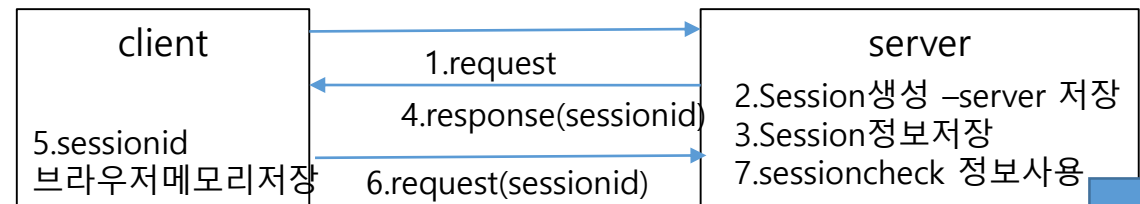
Session 사용 권고



Cookie

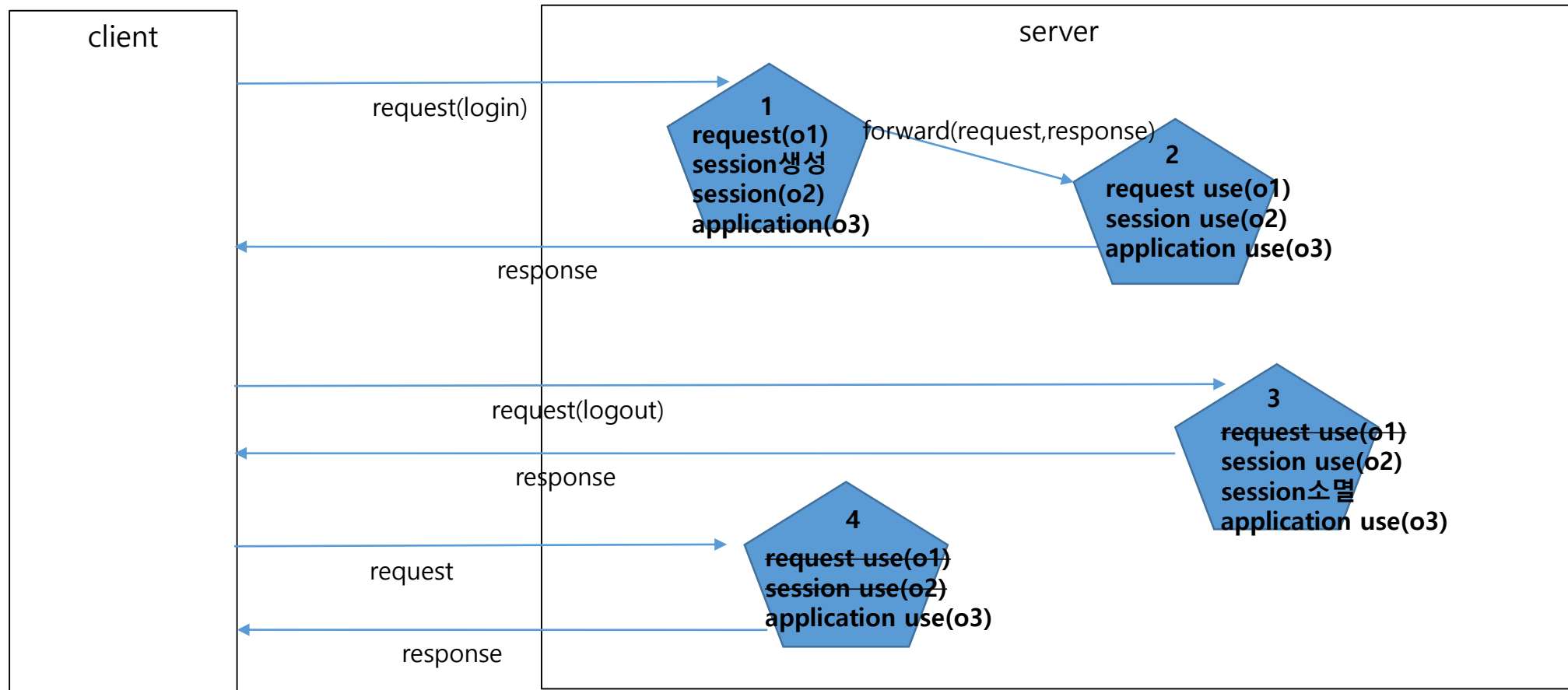


Session



HTTP – Request(scope)

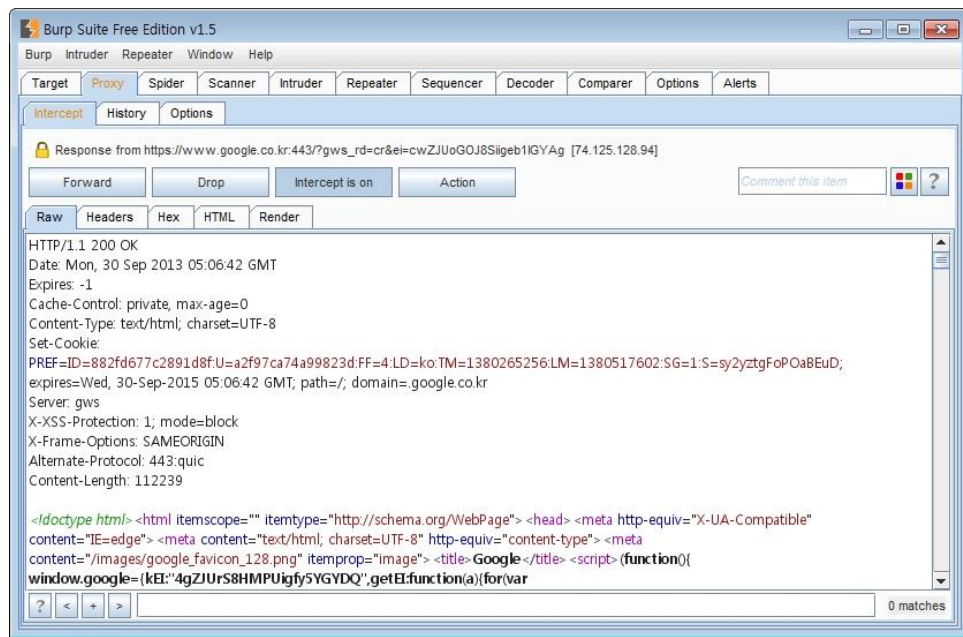
- request scope : request ~ response 유효 (thread safe)
- session(cookie) : session 생성(login) ~ session 소멸(logout 또는 timeout)
- application : service deploy(server start) ~ service undeploy(server stop)



HTTP – Response

- Response 패킷

- ✓ 클라이언트가 보낸 Request의 응답 패킷으로, 형식이 간단함



- ✓ Response 패킷에 담겨 있는 주요 내용은 서버에서 쓰이는 프로토콜 버전, HTTP 상태 코드 (200 OK) 등이며, 전달해 줄 데이터의 형식, 데이터 길이 등과 같은 추가 정보가 포함되어 있음
 - ✓ 위의 헤더 정보 뒤에 빈 줄이 하나 들어가고, 그 다음에 실제 데이터가 전달됨
 - ✓ 실제 데이터는 HTML이나 그림 파일이며, 데이터 전달이 종료되면 서버는 연결을 끊음

HTTP – Response

- HTTP 상태 코드
 - ✓ 일반적인 웹 서버 상태 코드

웹 서버 상태 코드	함축적 의미	내용
100번 대	정보 전송	임시 응답을 나타내는 것은 Status-Line과 선택적인 헤더들로 이루어져 있고 빈 줄로 끝을 맺는다. HTTP/1.0까지는 계열에 대한 어떤 정의도 이루어지지 않았기 때문에 시험용 외에는 서버 쪽의 추가 응답은 없다.
200번 대	성공	클라이언트의 요청이 성공적으로 수신되어 처리되었음을 뜻한다.
300번 대	리다이렉션	클라이언트의 요구 사항을 처리하기 위해서는 다른 곳에 있는 자원이 필요하다는 것을 의미한다.
400번 대	클라이언트 측 에러	클라이언트가 서버에 보내는 요구 메시지를 완전히 처리하지 못한 경우 등 클라이언트 측에서 오류가 발생한 것을 의미한다.
500번 대	서버 측 에러	서버 자체에서 생긴 오류 상황이나 클라이언트 요구 사항을 제대로 처리 할 수 없을 때 발생한다.

HTTP – Response

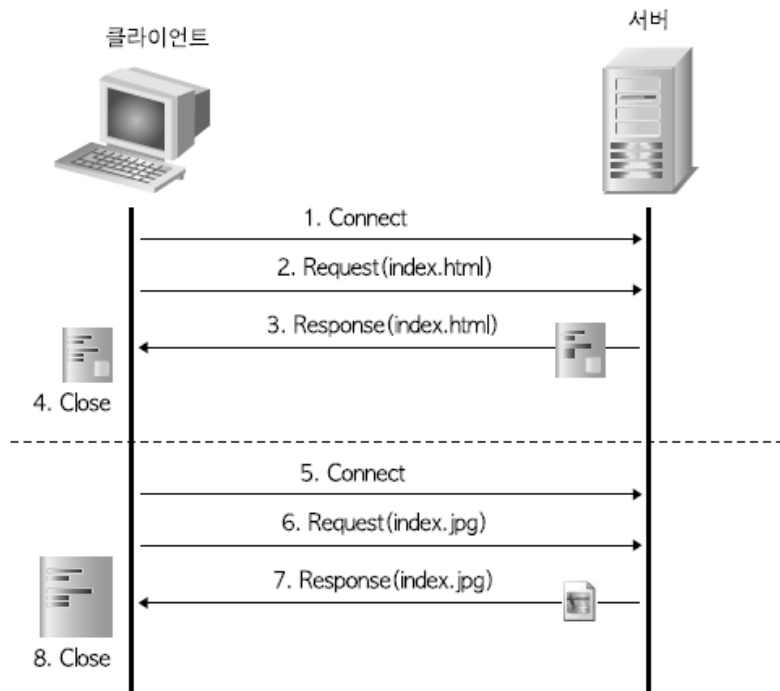
- HTTP 상태 코드
 - ✓ HTTP 상태 코드 중 눈여겨보아야 할 상태 코드

웹 서버 상태 코드	내용
200 OK	클라이언트의 요청이 성공했다는 것을 나타냄
201 Created	클라이언트의 PUT 요청이 성공적이라는 것을 나타냄
301 Moved Permanently	브라우저의 요청을 다른 URL로 항시 전달한다는 것을 의미함. 다른 URL에 대한 정보는 Location 헤더에 나타남
302 Moved Temporarily	브라우저의 요청을 임시 URL로 바꾸고 Location 헤더에 임시로 변경한 URL에 대한 정보를 적음. 클라이언트가 다음에 같은 요청을 하면 기존의 URL로 돌아감
304 Not Modified	브라우저가 서버에게 요청한 자료에 대해 서버는 클라이언트 내에 복사된 캐시를 사용하면 된다는 것을 의미함. 서버는 If-Modified-Since와 If-None-Match 요청 헤더를 사용해 클라이언트가 가장 최근의 자료를 가지고 있는지 여부를 확인함
400 Bad Request	클라이언트가 서버에 잘못된 요청을 했다는 것을 나타냄
401 Unauthorized	서버가 클라이언트의 요청에 대해 HTTP 인증 확인을 요구하는 것을 의미함
403 Forbidden	클라이언트의 요청에 대해 접근을 차단하는 것을 나타냄
404 Not Found	클라이언트가 서버에 요청한 자료가 존재하지 않음을 나타냄
500 Internal Server Error	서버가 클라이언트의 요청을 실행할 수 없을 때 500 상태 코드가 발생함. 일반적으로 SQL 인젝션 취약점이 존재하는지 확인할 때 500 에러가 유용하게 사용됨

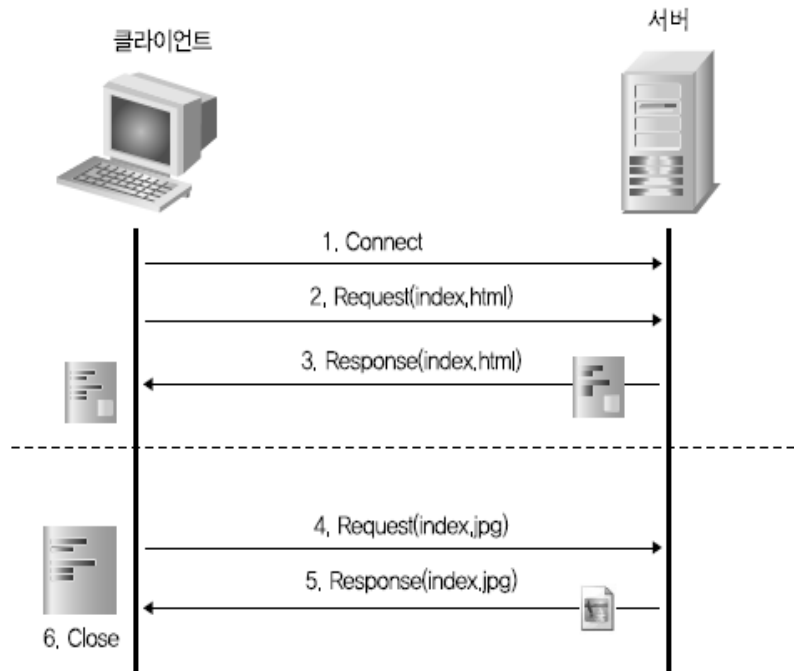
HTTP - 동작원리

- HTTP 1.0과 1.1 동작 원리

- ✓ HTTP 1.0에서는 문서에 몇 개의 그림이 있든 상관없이 텍스트가 저장된 HTML 문서를 먼저 전송받은 후 연결을 끊고 나서 다시 연결하여 그림을 전송 받음
- ✓ HTTP 1.1은 연결 요청이 계속 들어오면 HTML 문서를 받고 난 후 연결을 끊고 나서 다시 연결 요청을 하는 게 아니라 바로 그림 파일을 요청



HTTP 1.0을 이용하여 index.html 읽기



HTTP 1.1을 이용하여 index.html 읽기

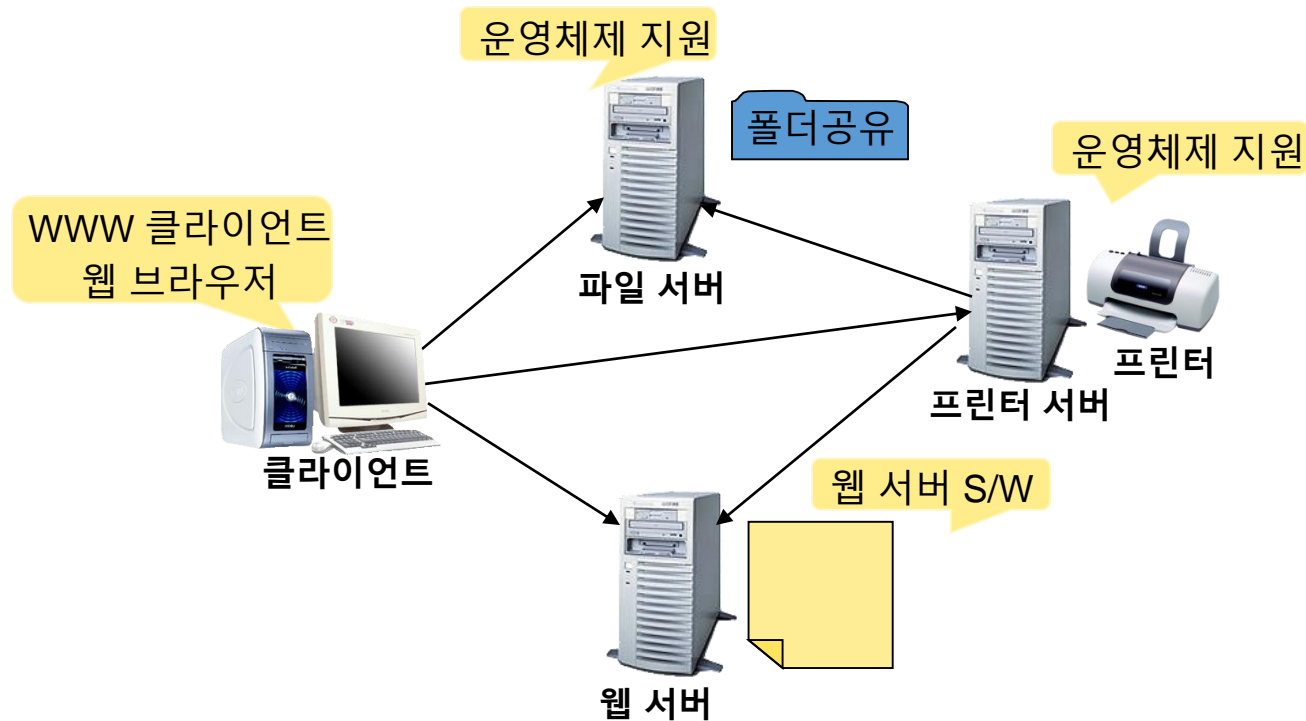
HTTP - 동작원리

- HTTPS (HTTP over Secure Socket Layer)
 - ✓ HTTP 프로토콜은 데이터가 네트워크 장비를 통과할 때 암호화되지 않은 단순한 TCP를 사용하기 때문에 네트워크에 잠입해 있는 공격자가 중간에 정보를 가로챌 수 있음
 - ✓ HTTPS는 애플리케이션 계층 프로토콜로 SSL(Secure Socket Layer)을 이용하여 클라이언트와 서버 사이에 주고받는 정보를 보호하는 데 사용
- REST
 - ✓ POST : Create(Insert)
 - ✓ GET : Read (Select)
 - ✓ PUT : Update(Update)
 - ✓ DELETE : Delete>Delete)

Server & Client 구조 - 웹 서버와 클라이언트

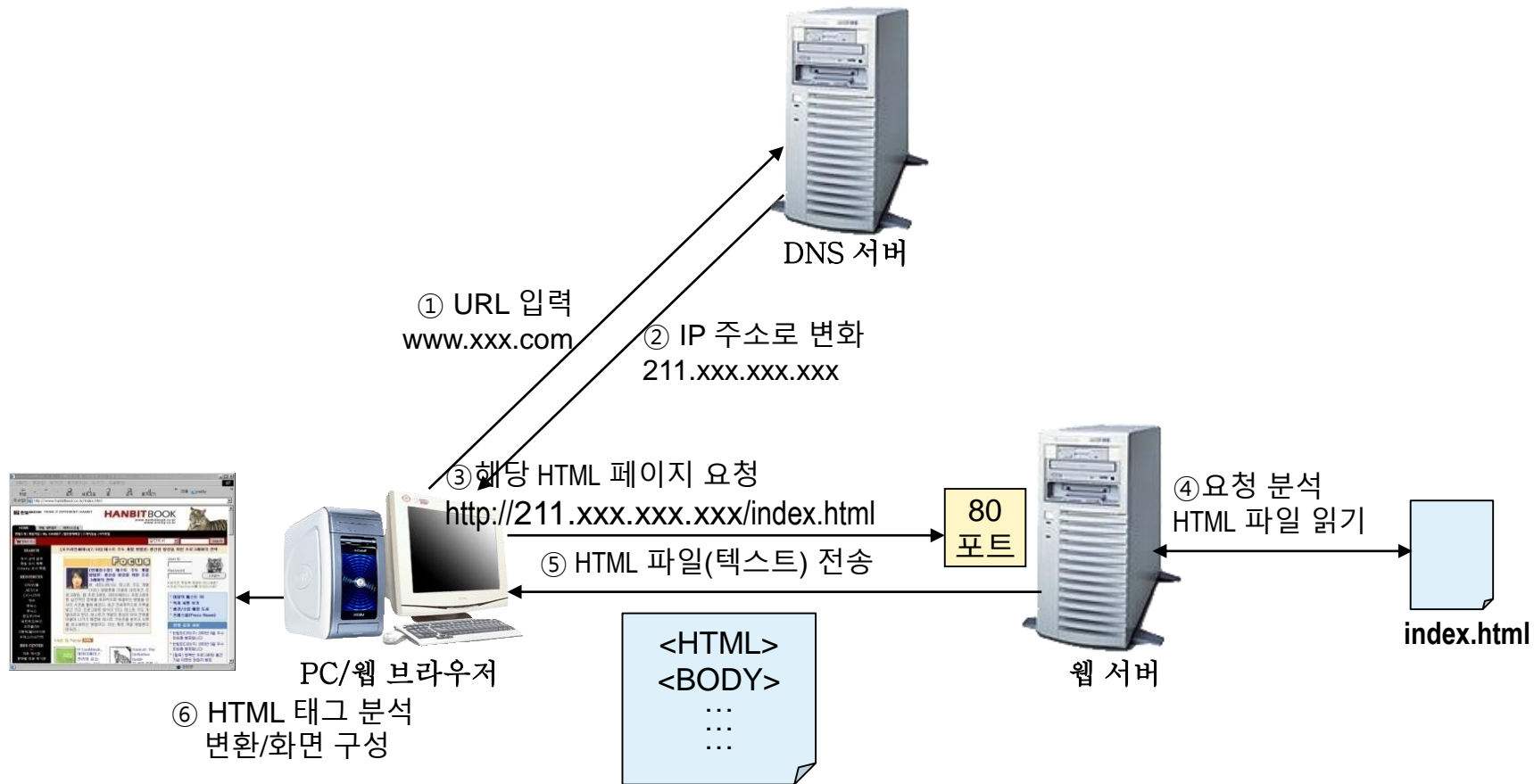
- 웹 서버와 클라이언트

- ✓ 서버: 네트워크에서 서비스를 제공하는 컴퓨터
- ✓ 클라이언트: 네트워크에서 서비스를 제공받는 컴퓨터
 - 최근 클라이언트와 서버의 하드웨어적인 구분이 없어지고 있음



Server & Client 구조 - WWW 동작원리

- 일반적인 www 서비스의 동작 과정



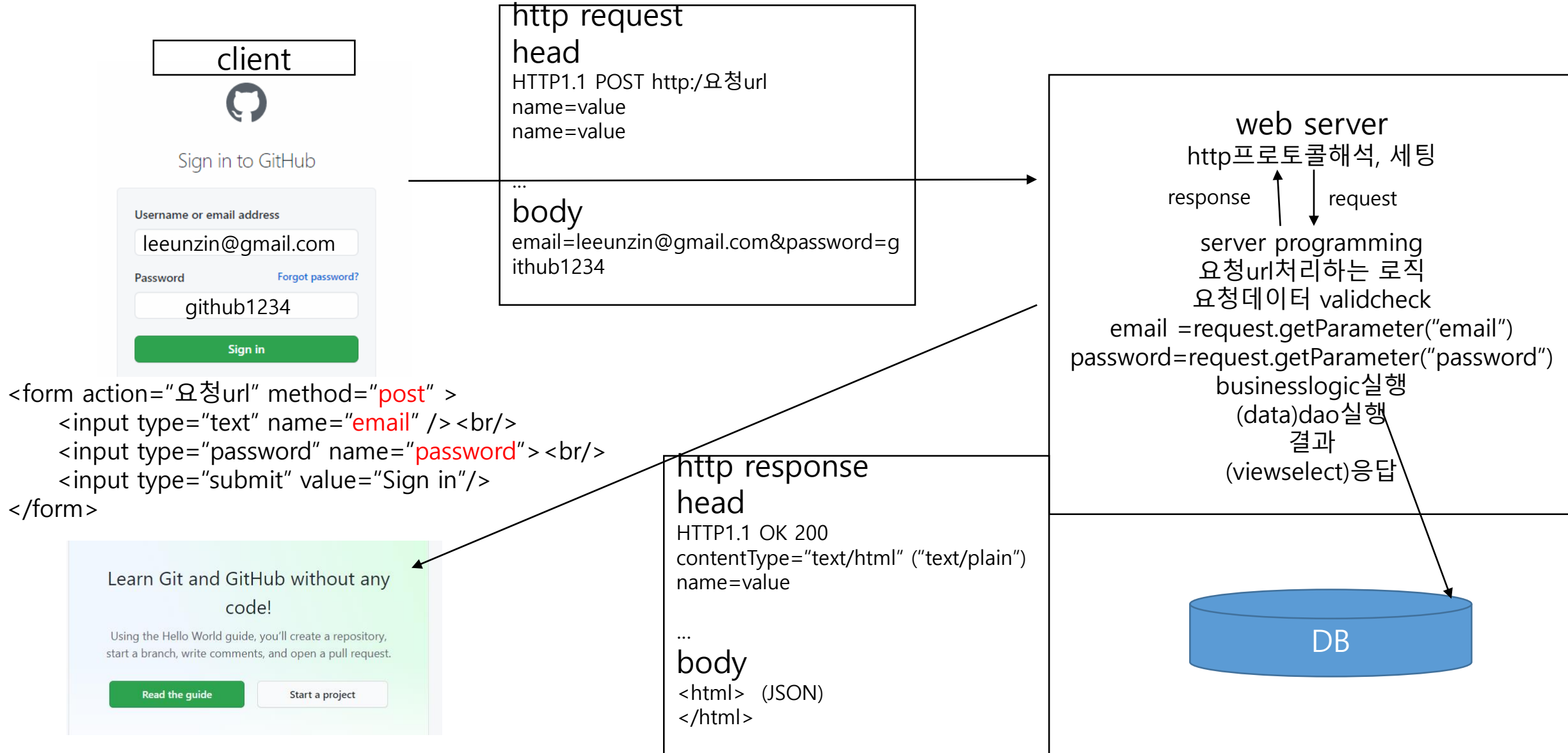
Server & Client 구조 - HTML & JavaScript

- HTML
 - ✓ HTML : 웹 브라우저에서 보여질 수 있는 화면을 만들기 위한 언어
 - ✓ HTML은 텍스트 파일로 정적인 정보만 처리 가능
 - 동적으로 변하는 정보를 처리할 수 없음
 - ✓ 동적인 내용을 처리하기 위해 CGI, Fast CGI, PHP, ASP, JSP, Servlet, Django 등의 동적인 웹 페이지 기술을 사용
- JavaScript
 - ✓ 웹 브라우저 내에서 실행될 수 있는 스크립트 형식의 언어
 - ✓ JavaScript를 이용하여 HTML에 동적인 효과를 줄 수 있음
 - ✓ 웹 브라우저가 스크립트 해석의 주체

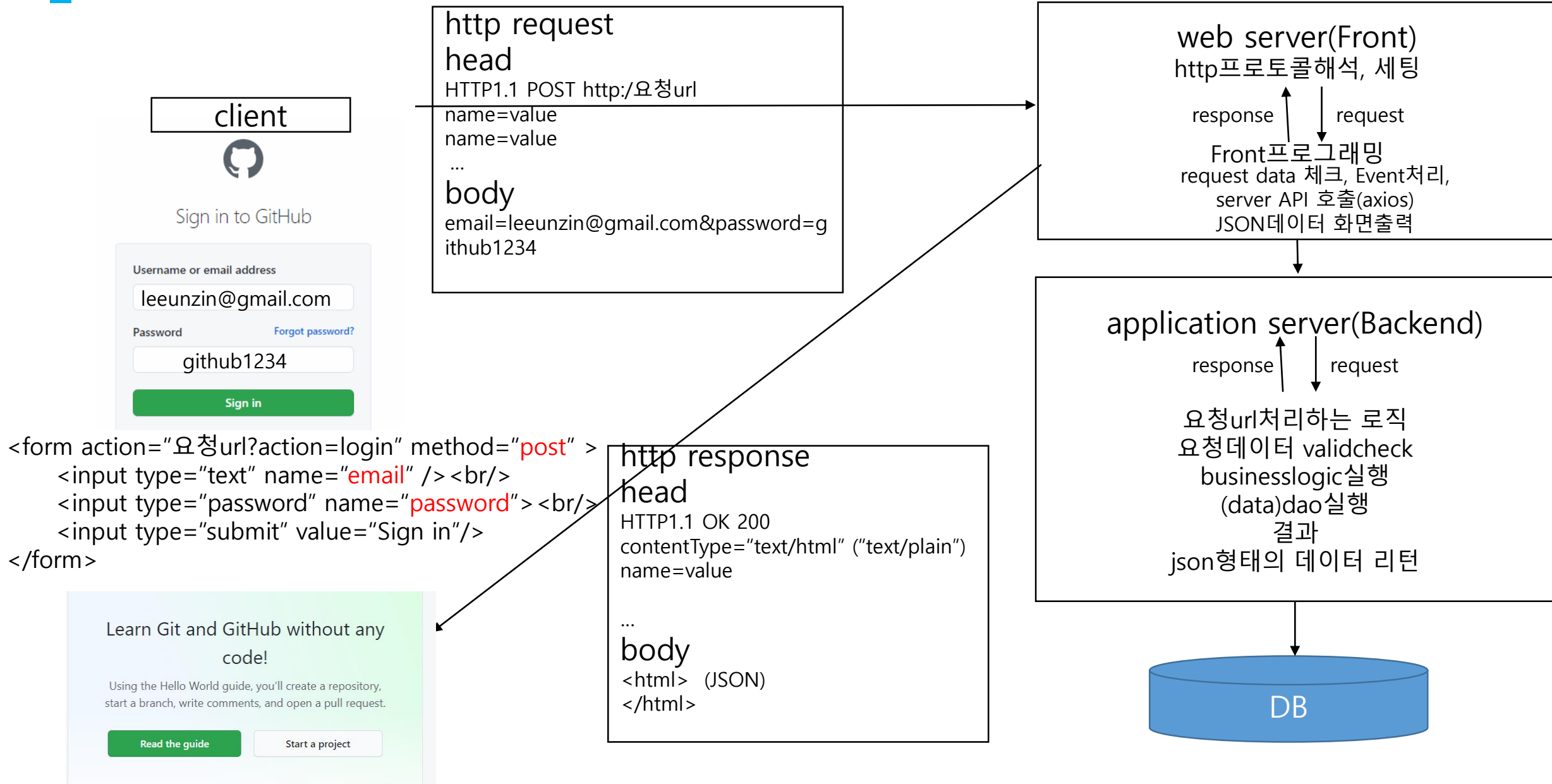
Server & Client 구조 - 서버 스크립트 기술

- 서버 스크립트 기술
 - ✓ HTML과 스크립트 언어를 함께 사용할 수 있는 기술로 웹 서버에서 해석
 - ✓ 웹 서버에서 실행되고 실행결과가 브라우저로 전송
 - ✓ 데이터베이스 연동 처리 등 다양한 구현이 가능
 - ✓ 별도의 컴파일 과정 없이 HTML 태그 수정 가능
 - ✓ ASP , PHP , JSP, Django 등

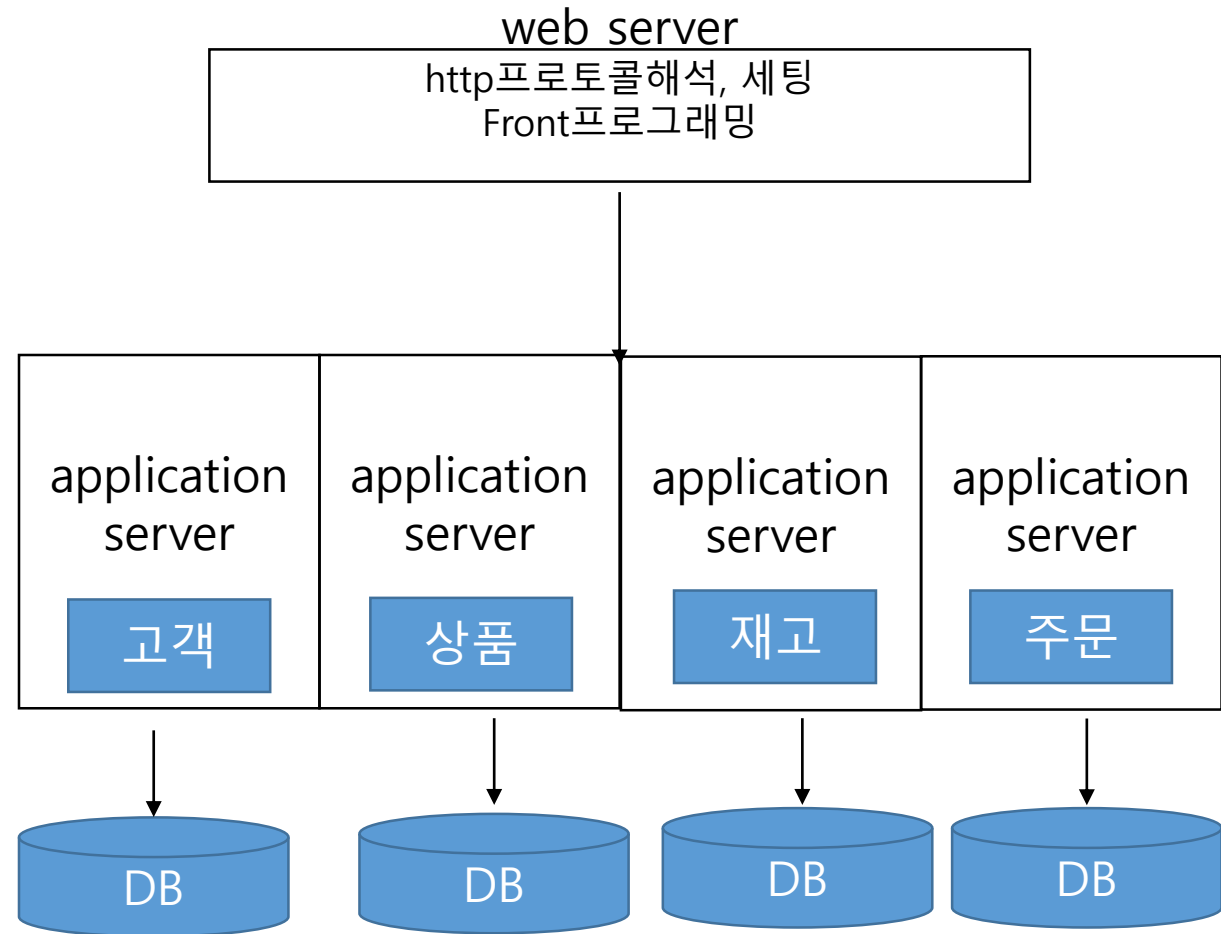
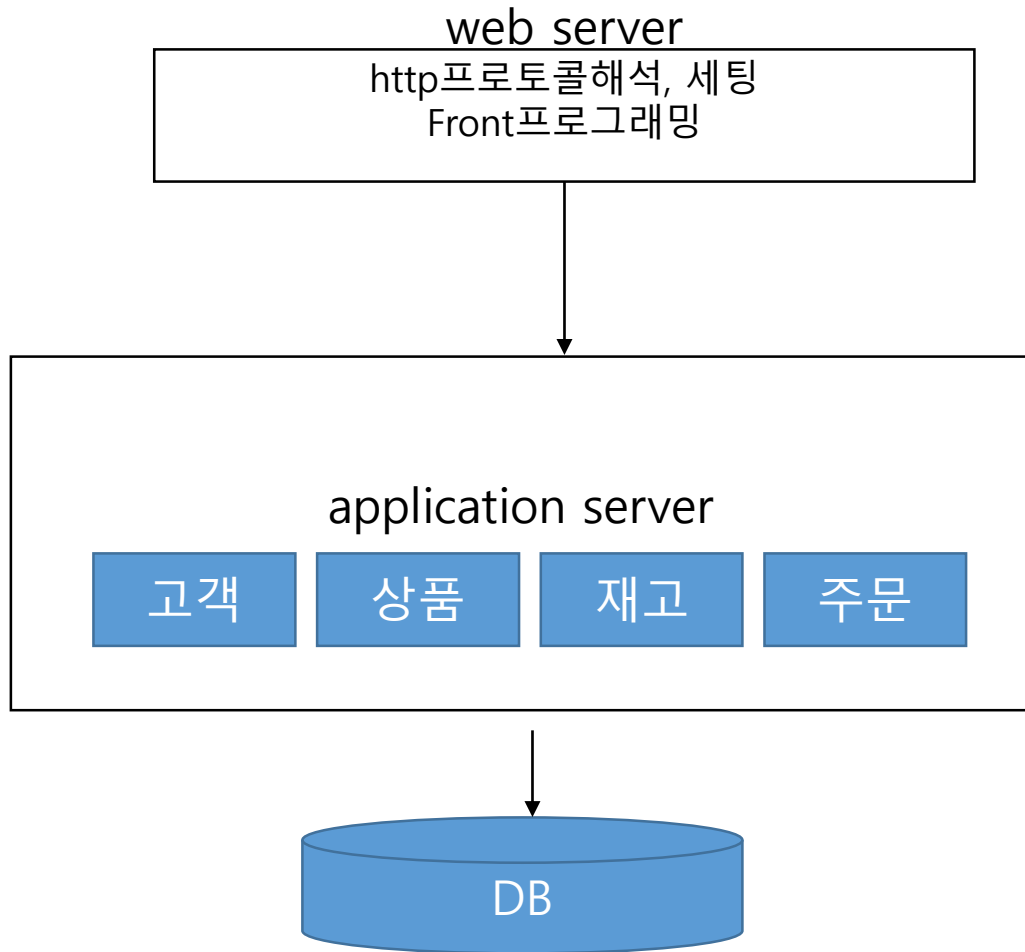
Server & Client 구조 - Web 동작 예



Server & Client 구조 - Web동작 예(Monolithic)



Server & Client 구조 – Monolithic vs. MSA



02

Django 개요

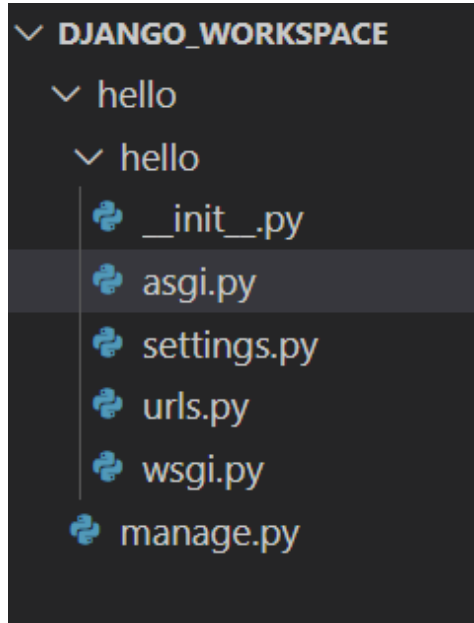
- Django 소개 및 설치
- Hello Project
- Pattern

Django 소개 및 설치

- Django 소개
 - ✓ Python 언어 기반의 웹 프레임워크
 - ✓ 다양한 라이브러리 제공
 - ✓ MVC Pattern 기반 개발환경 제공
 - ✓ 기본적인 admin 페이지 제공
 - ✓ Object model Relationship 제공 (sqlite)
 - ✓ <https://docs.djangoproject.com/ko/3.1/>
- Django 설치
 - ✓ python 3.x 설치
 - python 3.x.x - <https://www.python.org/downloads/>
 - python --version
 - ✓ Django 설치
 - python -m pip install django
 - python -m django --version
- Project
 - ✓ django-admin startproject [project이름]
 - ✓ django-admin startproject hello

Hello Project

- Hello Project
- django-admin startproject hello



project이름과 같은 이름의 package 생성
(project 관련된 config 모듈 파일들)

manage.py : Django command line 명령을 실행하기 위한 모듈 제공
python manage.py django명령어

- python manage.py runserver
- http://localhost:8000

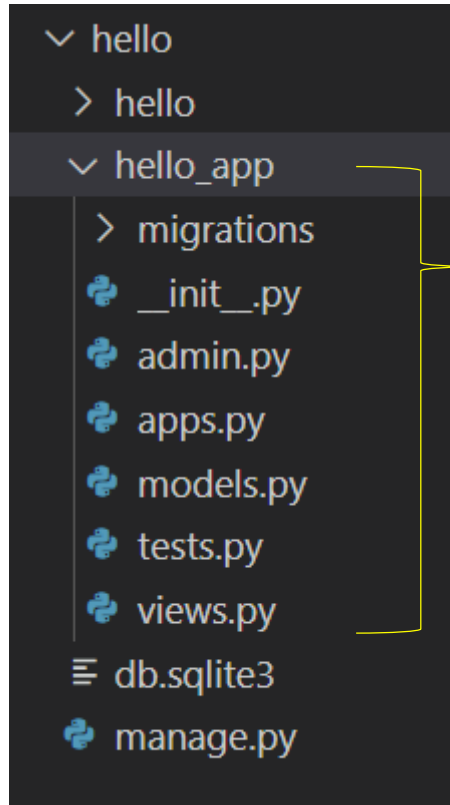
```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until
app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
January 15, 2021 - 16:34:02
Django version 3.1.5, using settings 'hello.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Hello Project - First Application hello_app

- `python manage.py startapp [application이름]`
- `python manage.py startapp hello_app`



application이름의 패키지 생성

application/migrations → models.py와 mapping Database migration결과 파일
들 저장 위치

application 구성을 위한 모듈 포함

models.py → entity

views.py → controller

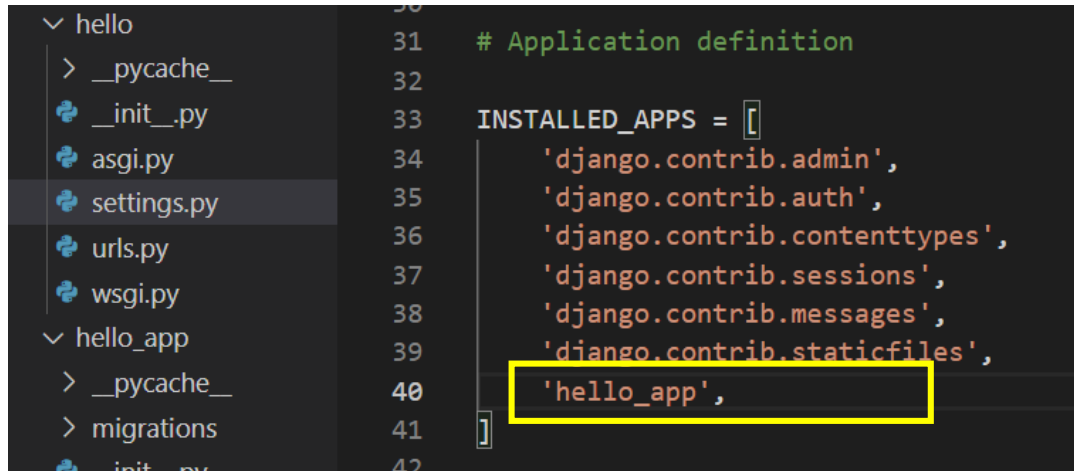
./templates/*.html → view(추가)

urls.py → request들어오는 요청 url별로 views.py의 함수 mapping 추가

- project생성 → [application 생성] * → project에 application등록

Hello Project - First Application hello_app

- project/settings.py application등록



```
30
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'hello_app',
41 ]
42
```

- hello_app/views.py controller localhost:8000/hello/요청시 응답할 index추가

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.

def index(request):
    return HttpResponse("Hello, world. You're at the Hello Django App index.")
```

Hello Project - First Application hello_app

- url추가
 - ✓ project에서 application url 추가 (hello/urls.py)

```
from django.contrib import admin
from django.urls import include, path


urlpatterns = [
    path('hello/', include('hello_app.urls')),
    path('admin/', admin.site.urls),
]
```

- application에서 요청별 url 추가 (hello_app/urls.py 파일 생성 후 추가)

```
from django.urls import path

from . import views

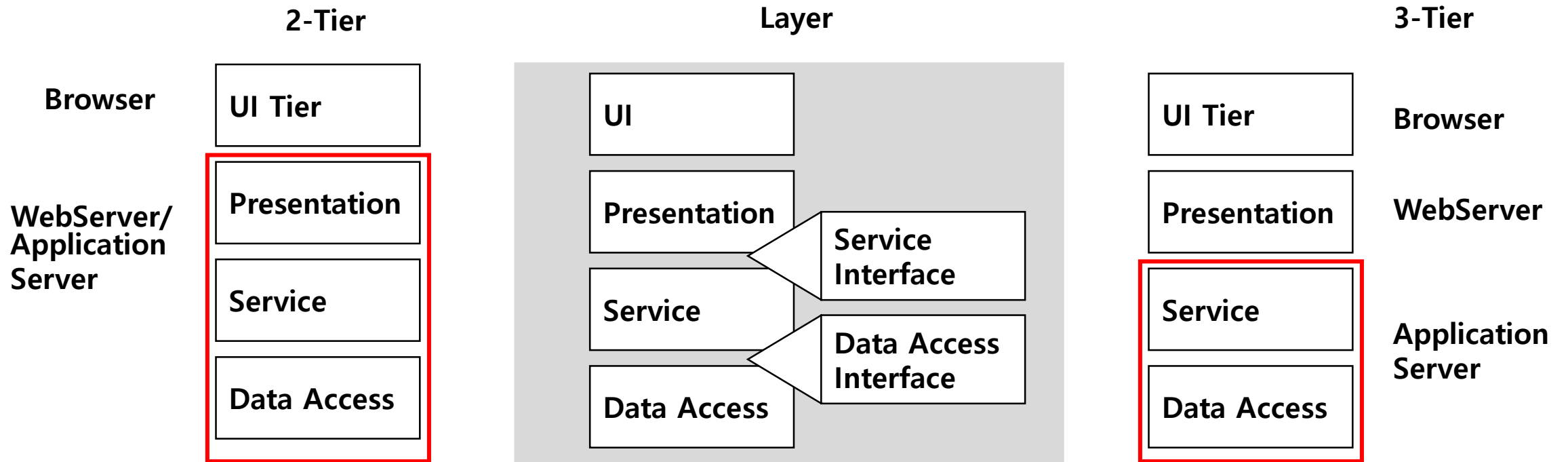
urlpatterns = [
    path('', views.index, name='index'),
]
```

A screenshot of a web browser's address bar. The address bar is highlighted with a red rectangle. It shows navigation icons (back, forward, refresh) and the URL "localhost:8000/hello/".

Hello, world. You're at the Hello Django App index.

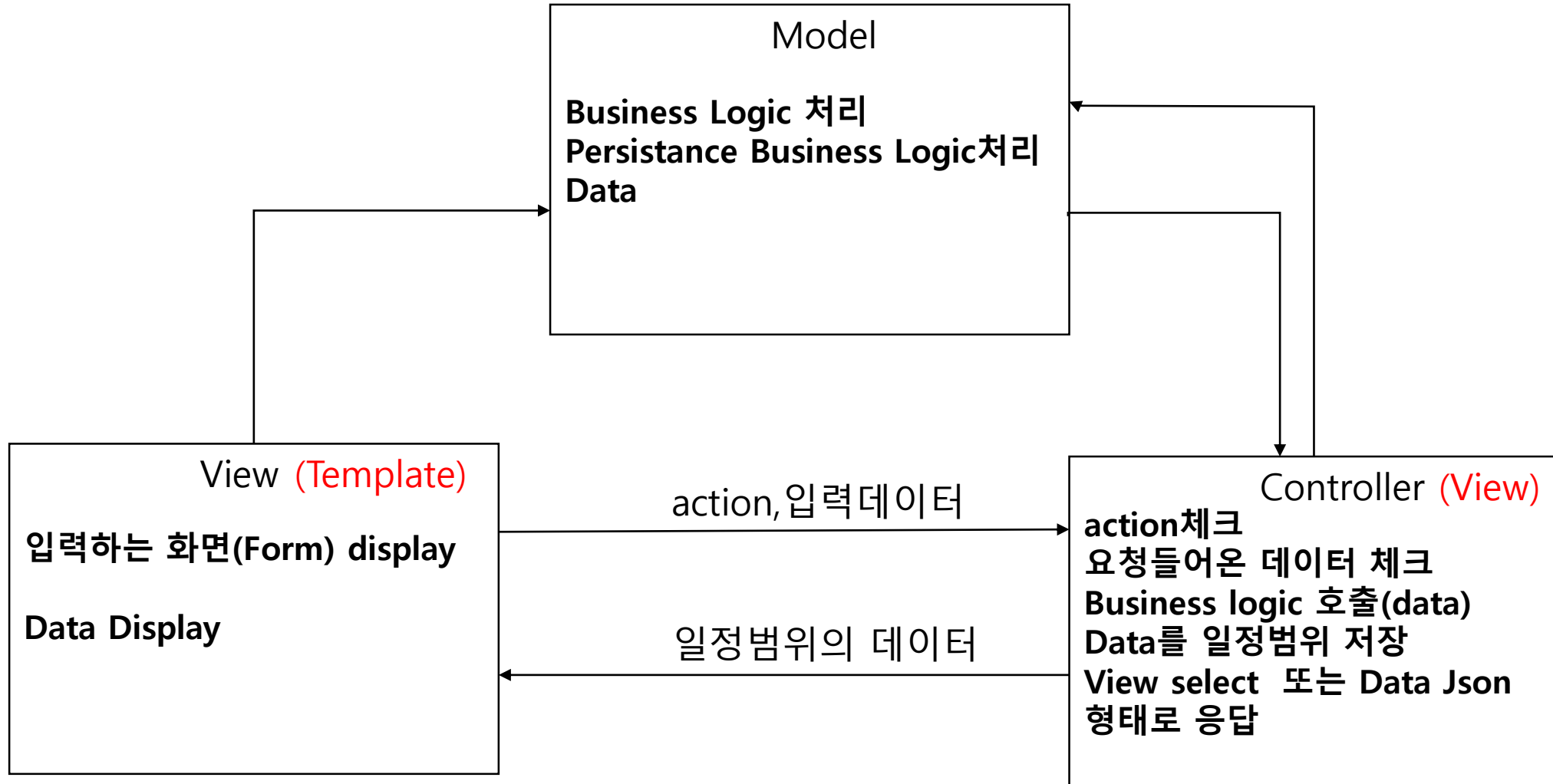
Pattern - Layer(논리적계층) & Tier(물리적인계층)

- 각 계층별 독립적으로 설계
- 바로 아래 계층에만 영향을 받고 다른 계층 변경 시 영향받지 않음



Pattern - MVC

- 역할별로 기능 나누어 설계



03

TODO 웹 프로그래밍

- 프로젝트 구성
- 프로젝트, app 생성
- MVC
- 일정등록
- 일정목록
- 일정삭제

프로젝트 구성

- Git에 프로젝트 생성
 - ✓ Todo_with_Django 프로젝트 생성하기

Project name

todo_with_django

Visibility Level ?

☒ Private

Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to mem

☒ Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project

✓ 프로젝트 clone

master todo_with_django / +

History Find file Web IDE Clone

Initial commit
eunjin Lee authored just now 565cdee9

README Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster Set up CI/CD

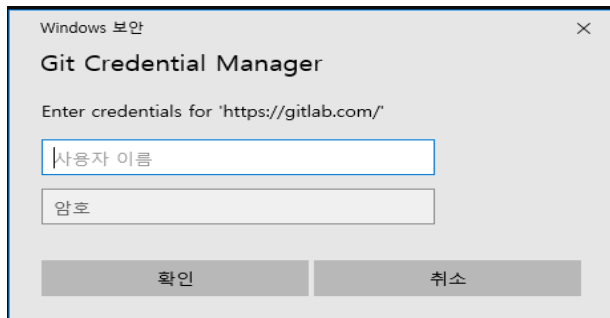
Name	Last commit	Last update
README.md	Initial commit	just now

README.md

todo_with_django

프로젝트 구성

- django_workspace 폴더 생성 후 프로젝트 clone
 - ✓ 인증 안된 경우 인증



- ✓ clone 후 폴더 생성 확인

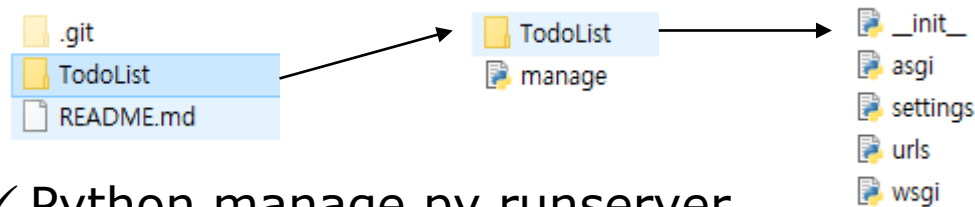
```
Cloning into 'todo_with_django'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

[todo_with_django]
```

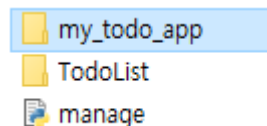
- ✓ 이후 - todo_with_Django 폴더에서 구현 -> git add . -> git commit -m "message" -> git push
- ✓ .gitignore 파일 추가
 - *.pyc
 - __pycache__

프로젝트, app 생성

- Django 프레임워크를 이용한 웹 프로젝트 생성
 - ✓ django-admin startproject 프로젝트이름
 - ✓ django-admin startproject TodoList
 - ✓ 생성된 프로젝트 구성 확인



- ✓ Python manage.py runserver
- TodoList에 대한 Application 구성
 - ✓ python manage.py startapp my_todo_app



- ✓ TodoList/setting.py에 app 등록

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'my_todo_app'
]
```

URL 구성

- TodoList/url.py
 - ✓ urlpatterns URL 추가

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('my_todo_app.urls'))
]
```

- ✓ my_todo_app/urls.py 추가

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index)
]
```

- ✓ Views.py 모듈 index 추가

```
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.
def index(request):
    return HttpResponse("my todo app")
```

- ✓ 확인
python manage.py runserver

← → ↻ ⓘ localhost:8000

my todo app

HTML 템플릿 사용

- HTML 템플릿 사용시 Django는 해당 app에서 templates라는 폴더 탐색 app과 같은 이름의 폴더에서 html파일 검색
 - ✓ my_todo_app/template/my_todo_app 폴더 생성
 - ✓ index.html 작성
 - ✓ views.py 수정

```
from django.shortcuts import render
from django.http import HttpResponse
```

```
# Create your views here.
```

```
def index(request):
```

```
    # return HttpResponse("my todo app")
```

```
    return render(request, 'my_todo_app/index.html')
```

← → ↻ ⓘ localhost:8000



To-do List with Django

메모할 내용을 적어주세요

메모하기!

HTML 템플릿 사용

- HTML 템플릿 사용시 Django는 해당 app에서 templates라는 폴더 탐색 app과 같은 이름의 폴더에서 html파일 검색
 - ✓ my_todo_app/template/my_todo_app 폴더 생성
 - ✓ index.html 작성
 - ✓ views.py 수정

```
from django.shortcuts import render
from django.http import HttpResponse
```

```
# Create your views here.
```

```
def index(request):
```

```
    # return HttpResponse("my todo app")
```

```
    return render(request, 'my_todo_app/index.html')
```

← → ↻ ⓘ localhost:8000

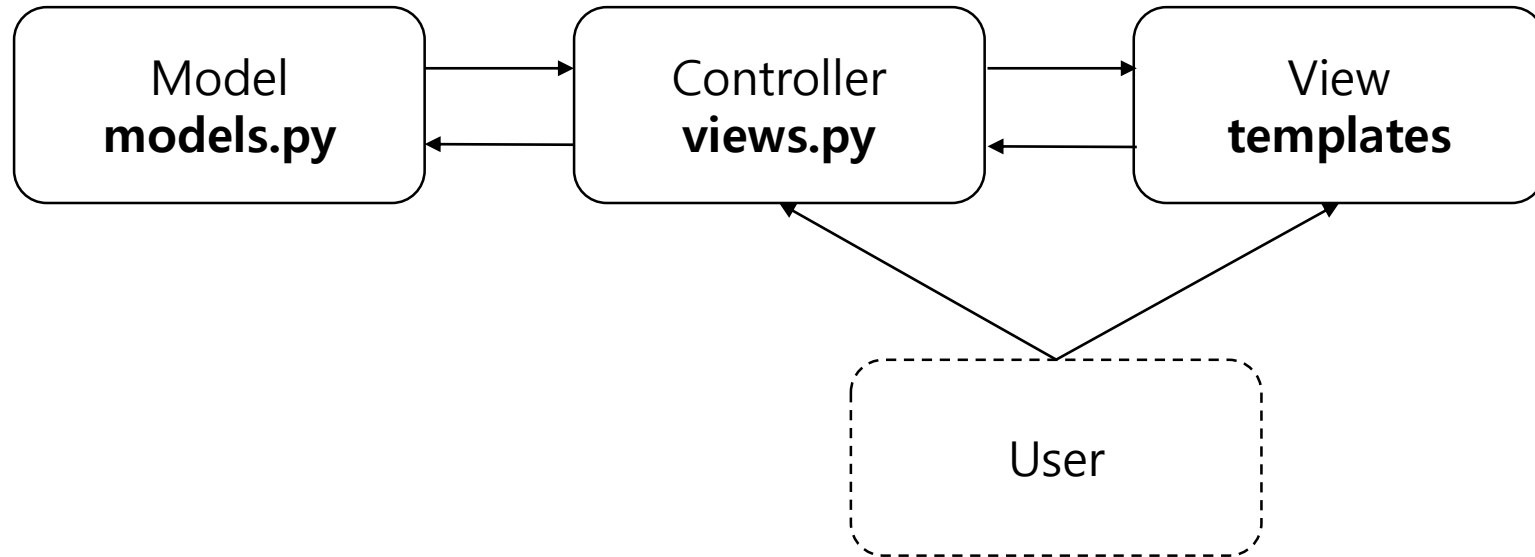
🔍 ☆ ⚙️ e

To-do List with Django

메모할 내용을 적어주세요

메모하기!

MVC



MVC - Model

- 데이터베이스 선택
 - ✓ Django : SQLite 기본설정
- 테이블 정의
 - ✓ models.py

```
from django.db import models

# Create your models here.
class Todo(models.Model) :
    content = models.CharField(max_length = 255)
```

✓ database 반영

- python manage.py makemigrations python manage.py migrate
- my_todo_app/migrations/0001.initial.py

```
Migrations for 'my_todo_app':
my_todo_app/migrations/0001_initial.py
- Create model Todo

Operations to perform:
  Apply all migrations: admin, auth, contenttypes, my_todo_app, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
```

```
from django.db import migrations, models

class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Todo',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('content', models.CharField(max_length=255)),
            ],
        ),
    ]
```

MVC - Model -Field Type

- Field Type

Field Type	설명
CharField	제한된 문자열 필드 타입. 최대 길이를 max_length 옵션에 지정해야 한다. 문자열의 특별한 용도에 따라 CharField의 파생클래스로서, 이메일 주소를 체크를 하는 EmailField, IP 주소를 체크를 하는 GenericIPAddressField, 콤마로 정수를 분리한 CommaSeparatedIntegerField, 특정 폴더의 파일 패스를 표현하는 FilePathField, URL을 표현하는 URLField 등이 있다.
TextField	대용량 문자열을 갖는 필드
IntegerField	32 비트 정수형 필드. 정수 사이즈에 따라 BigIntegerField, SmallIntegerField 을 사용할 수도 있다.
BooleanField	true/false 필드. Null 을 허용하기 위해서는 NullBooleanField를 사용한다.
DateTimeField	날짜와 시간을 갖는 필드. 날짜만 가질 경우는 DateField, 시간만 가질 경우는 TimeField를 사용한다.
DecimalField	소숫점을 갖는 decimal 필드
BinaryField	바이너리 데이터를 저장하는 필드
FileField	파일 업로드 필드
ImageField	FileField의 파생클래스로서 이미지 파일인지 체크한다.
UUIDField	GUID (UUID)를 저장하는 필드

MVC - Model – Field Options

- Field Options

Field Option	설명
null (Field.null)	null=True 이면, Empty 값을 DB에 NULL로 저장한다. DB에서 Null이 허용된다. 예: models.IntegerField(null=True)
blank (Field.blank)	blank=False 이면, 필드가 Required 필드이다. blank=True 이면, Optional 필드이다. 예: models.DateTimeField(blank=True)
primary_key (Field.primary_key)	해당 필드가 Primary Key임을 표시한다. 예: models.CharField(max_length=10, primary_key=True)
unique (Field.unique)	해당 필드가 테이블에서 Unique함을 표시한다. 해당 컬럼에 대해 Unique Index를 생성한다. 예: models.IntegerField(unique=True)
default (Field.default)	필드의 디폴트값을 지정한다. 예: models.CharField(max_length=2, default="WA")
db_column (Field.db_column)	컬럼명은 디폴트로 필드명을 사용하는데, 만약 다르게 쓸 경우 지정한다.

MVC - Model – Database Setting

- SQLITE

Database

<https://docs.djangoproject.com/en/1.9/ref/settings/#databases>

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
    }  
}
```

- MYSQL

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': DBName,  
        'USER': 'userName',  
        'PASSWORD': 'password',  
        'HOST': 'localhost',  
        'PORT': '3306',  
    }  
}
```

- DB엔진 값

django.db.backends.postgresql
django.db.backends.mysql
django.db.backends.sqlite3
django.db.backends.oracle

MVC - Model - CRUD

- INSERT

- ✓ Model 객체 생성
- ✓ model객체.save()

- SELECT

- ✓ 자동 생성되는 objects 객체 (objects = models.Manager()) 사용
- ✓ Model클래스이름.objects 사용
- ✓ all() : 모든 데이터 리턴 예) Users.objects.all()
- ✓ get(pk=값) : 하나의 데이터 리턴 예) Users.objects.get(pk=1)
- ✓ filter(조건) : 조건에 맞는 데이터들 리턴 예) Users.objects.filter(name='Lee')
- ✓ exclude(조건) : 조건을 제외한 데이터 예) Users.objects.exclude(name='Lee')
- ✓ count() : 데이터의 개수 예) Users.objects.count()
- ✓ order_by('컬럼명', '-컬럼명') : 데이터 정렬
 컬럼명 : 오름차순, -컬럼명 : 내림차순
- ✓ distinct() : 중복되는 데이터 한번 표시
- ✓ first() : 첫번째 데이터 리턴 예) Feedback.objects.order_by('name').first()
- ✓ last() : 마지막 데이터 리턴

MVC - Model – CRUD

- UPDATE

- ✓ 수정하고자하는 Model 객체 SELECT 예) `user = Users.objects.get(pk=1)`
- ✓ SELECT한 객체.수정컬럼 = 값 `user.name='Kim'`
- ✓ SELECT한 객체.save()로 수정한 데이터 저장 `user.save()`

- DELETE

- ✓ 삭제하고자하는 Model 객체 SELECT 예) `user = Users.objects.get(pk=1)`
- ✓ SELECT한 객체.delete() `user.delete()`

MVC - Model

- 데이터베이스 확인

- ✓ python manage.py dbshell

- ✓ sqlite> .tables #테이블 목록보기 <프로젝트 이름>_<모델이름>으로 생성

- ✓ sqlite> pragma table_info(my_todo_app_todo); #테이블 구조 확인

```
0|id|integer|1||1
```

```
1|content|varchar(255)|1||0
```

#순서|이름|형태|notnull여부|pk여부

여부 : 1 true , 0 false

- ✓ sqlite>select * from my_todo_app_todo

일정등록

- template

- ✓ url ./createTodo로 요청

- ✓ {%csrf_token%} : 요청한 사용자 구분. {%python code%}

```
<div class="content">
  <div class="messageDiv">
    <form action="./createTodo/" method="POST">{% csrf_token %}
      <div class="input-group">
        <input id="todoContent" name="todoContent" type="text" class="form-control" placeholder="메모할 내용을 적어주세요">
        <span class="input-group-btn">
          <button class="btn btn-default" type="submit">메모하기!</button>
        </span>
      </div>
    </form>
  </div>
</div>
```

- ✓ url.py 등록

- my_todo_app/url.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('createTodo/', views.createTodo, name='createTodo')
]
```

- view.py createTodo추가

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from django.urls import reverse
from .models import *

# Create your views here.
def index(request):
    # return HttpResponseRedirect(reverse("index"))
    return render(request, 'my_todo_app/index.html')

def createTodo(request):
    todoContent = request.POST['todoContent']
    new_todo = Todo(content=todoContent)
    new_todo.save()
    return HttpResponseRedirect(reverse("index"))
```

일정목록

- template

- ✓ 등록 후 index페이지로 redirect
- ✓ view.py index 수정
 - todos = Todo.objects.all()
 - content = {'todos': todos}
 - return render(request, 'my_todo_app/index.html')

- ✓ template 반영

- {% python code %}
- {{ python data }}

```
<ul class="list-group">
  {% for todo in todos %}
    <form action="" method="GET">
      <div class="input-group" name='todo1'>
        <li class="list-group-item">{{ todo.content }}</li>
        <input type="hidden" id="id" name="id" value="{{ todo.id }}"></input>
        <span class="input-group-addon">
          <button type="submit" class="custom-btn btn btn-danger">완료</button>
        </span>
      </div>
    </form>
  {% endfor %}
</ul>
```

일정삭제

- views.py deleteTodo(request) 추가

```
def deleteTodo(request):
    delete_todo_id = request.GET['id']
    print("완료한 todo의 id", delete_todo_id)
    todo = Todo.objects.get(id = delete_todo_id)
    todo.delete()
    return HttpResponseRedirect(reverse('index'))
```

GET요청의 id 파라미터 값 받아오기

Todo Model의 id 검색
삭제

index.html로 다시 요청

- templates/my_todo_app/index.html 수정

```
<ul class="list-group">
  {% for todo in todos %}
  <form action="." deleteTodo" method="GET">
    <div class="input-group" name='todo1'>
      <li class="list-group-item">{{ todo.content }}</li>
      <input type="hidden" id="id" name="id" value="{{ todo.id }}"></input>
      <span class="input-group-addon">
        <button type="submit" class="custom-btn btn btn-danger">완료</button>
      </span>
    </div>
  </form>
  {% endfor %}
</ul>
```

GET방식으로 ./deleteTodo url로 request

hidden타입으로 request에 id={{todo.id}} 전달

- urls.py에 ,.deleteTodo/와 views.py/deleteTodo() 연결

```
urlpatterns = [
    path('', views.index, name='index'),
    path('createTodo/', views.createTodo, name='createTodo'),
    path('deleteTodo/', views.deleteTodo, name='deleteTodo'),
]
```

04

TODO Refactoring

- **Templates Composite View Pattern**
- **Django Form**
- **Static 파일 관리**

Templates Composite View Pattern

- 화면 재사용을 위해 layout이되는 base.html 추가(project/tempaltes/base.html추가)

```
<body>
  <div class="container">
    <div class="header">
      <div class="page-header">
        <h1>To-do List <small>with Django</small></h1>
      </div>
    </div>
    {% block content %} {% endblock content %}
    <div class="panel-footer">
      실전예제로 배우는 Django. Project1-TodoList
    </div>
  </div>
</body>
```

index.html의 head와 body의 div 엘리먼트의 class="container", class="header", class="panel-footer"를 base.html로 옮기고 일정등록과 목록부분이 들어갈 위치에 {% block content %} {% endblock content %} 추가

- templates/my_todo_app/index.html 수정
 - ✓ {% extends "base.html" %} 첫라인 추가
 - ✓ {% block content %} base.html로 옮기고 남은 코드 {% endblock content %}로 감싸기
- setting.py TEMPLATES 수정

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [ os.path.join(BASE_DIR, 'templates') ],
        'APP_DIRS': True,
```

프로젝트 BASE_DIR에 templates 추가
application directory의 templates 검색 True

Templates Composite View Pattern

```
38
39 <body>
40 <div class="container">
41 <div class="header">
42 <div class="page-header">
43 <h1>To-do List <small>with Django</small></div>
44 </div>
45 </div>
46
47 {% block content %}{% endblock content %}
48 <div class="panel-footer">
49 실전예제로 배우는 Django. Project1-TodoLi
50 </div>
51 </div>
52 </div>
53 </body>
54 </html>
```

Django Form

- Model 클래스로부터 폼(Form)을 자동으로 생성하는 기능 제공
 - ✓ django.forms.ModelForm 클래스로부터 파생된 사용자 폼 클래스를 정의
 - ✓ 사용자 폼 클래스 안에 Meta 클래스 (Inner 클래스)를 정의하고 Meta 클래스 안 model 속성 (attribute)에 해당 모델 클래스를 지정. 어떤 모델을 기반으로 폼을 작성할 것인지 Meta.model 에 지정

```
from django.forms import ModelForm
from .models import Todo

class TodoForm(ModelForm):
    class Meta:
        model = Todo
        fields = ('id', 'content')
```

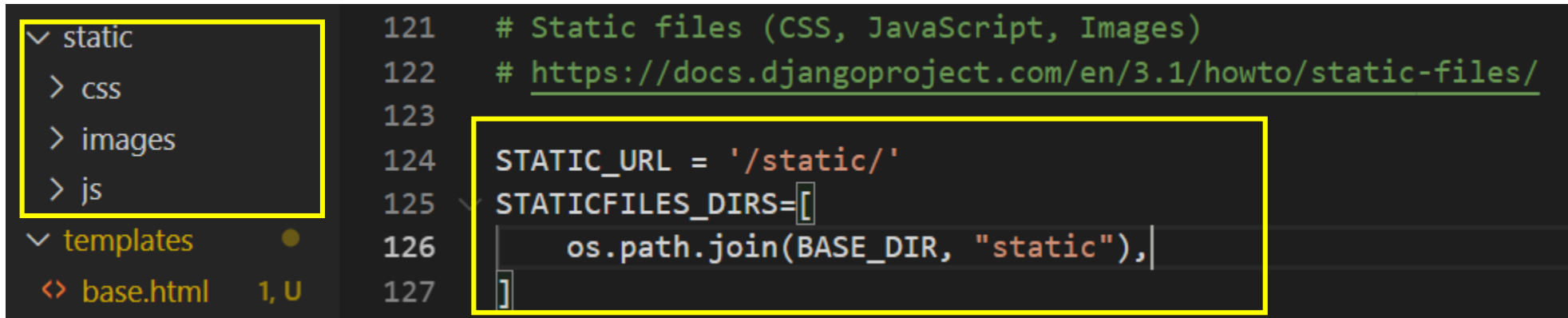
```
<div class="input-group">
  <input id="todoContent" name="content" type="text" class="form-cont
  <span class="input-group-btn">
    <button class="btn btn-default" type="submit">메모하기!<
  </span>
</div>
```

- views.py에서 폼을 핸들링하는 함수 이용
 - ✓ 주의 : templates로부터 전달되는 값의 name과 Meta.model의 fields이름 동일

```
def createTodo(request):
    # user_input_str = request.POST['todoContent']
    # new_todo = Todo(content = user_input_str)
    new_todo = TodoForm(request.POST)
    new_todo.save()
    return HttpResponseRedirect(reverse('index'))
    #return HttpResponse("create Todo를 할거야! =>" + user_input_str)
```

Static 파일 관리

- css, images, js 등 static 파일 관리는 project 홈 디렉토리(BASE_DIR)에 static 폴더를 생성하고 settings.py에 명시



```
121 # Static files (CSS, JavaScript, Images)
122 # https://docs.djangoproject.com/en/3.1/howto/static-files/
123
124 STATIC_URL = '/static/'
125 STATICFILES_DIRS=[
126     os.path.join(BASE_DIR, "static"),
127 ]
```

- application에 static파일을 관리할 때
 - ✓ 충돌 방지 : app이름/static/app이름과 같이 "app이름" 서브 폴더 권장
 - ✓ settings.py에 STATICFILES_FINDERS 추가

```
STATICFILES_FINDERS = [
    'django.contrib.staticfiles.finders.FileSystemFinder',
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',
]
```


Static 파일 관리-사용

- Templates에서 사용
 - ✓ `{% load staticfiles %}` 첫 라인에 추가
 - ✓ `"{% static '리소스이름' %}"` 로 static/ 폴더명 다음 경로 지정 참조
예) ``
- collectstatic
 - ✓ 배포 시 `"manage.py collectstatic"` 명령을 이용해 프로젝트와 app안의 static 파일 모음
 - ✓ settings.py : `STATIC_ROOT` 에 지정된 경로에 정적 파일들 복사