Django

파이썬 웹 프레임워크

Web Programming 이란

Front End / Back End

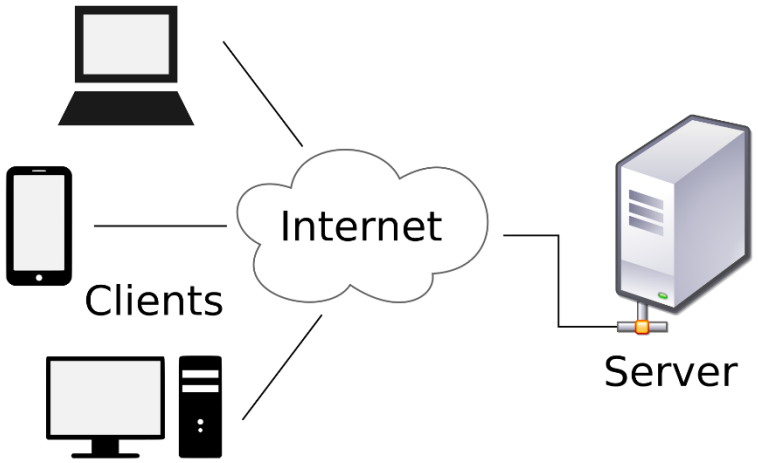
- 웹 개발은 Front End 개발과 Back End 개발로 나누어 진다.
- Front End
  - 사용자가 보는 부분을 개발한다.
    - UX 디자이너 – User Experience 디자인.
    - Web 디자이너 – 화면 디자인
    - Front end 개발자 – 사용자 쪽 웹 어플리케이션 구현
- Back End
  - 사용자 요청을 처리하는 서버 부분을 개발한다.
    - 서버 프로그램 개발자
    - DB 관리자
- Full Stack 개발자
  - front end와 back end를 모두 개발하는 개발자

인터넷

- 네트워크 (Network)
  - 컴퓨터와 컴퓨터를 연결한 것
- 인터넷 (Internet)
  - 전세계 컴퓨터 들을 연결한 통신망
- 프로토콜 (Protocol)
  - 네트워크 통신 규약
  - 네트워크의 목적은 연결된 컴퓨터 간의 데이터(정보) 교환이다.
  - 두 컴퓨터 간에 정보를 어떻게 교환할 것인지를 정의한 규약을 프로토콜이라고 한다.
  - TCP/IP 프로토콜
    - 데이터를 패킷단위로 나누어 전송하며 데이터 전송을 보장하며 보낸 순서대로 패킷을 받게 한다.
    - HTTP, FTP, SMTP 등의 기반이 되는 프로토콜

서버(Server)/클라이언트(Client)

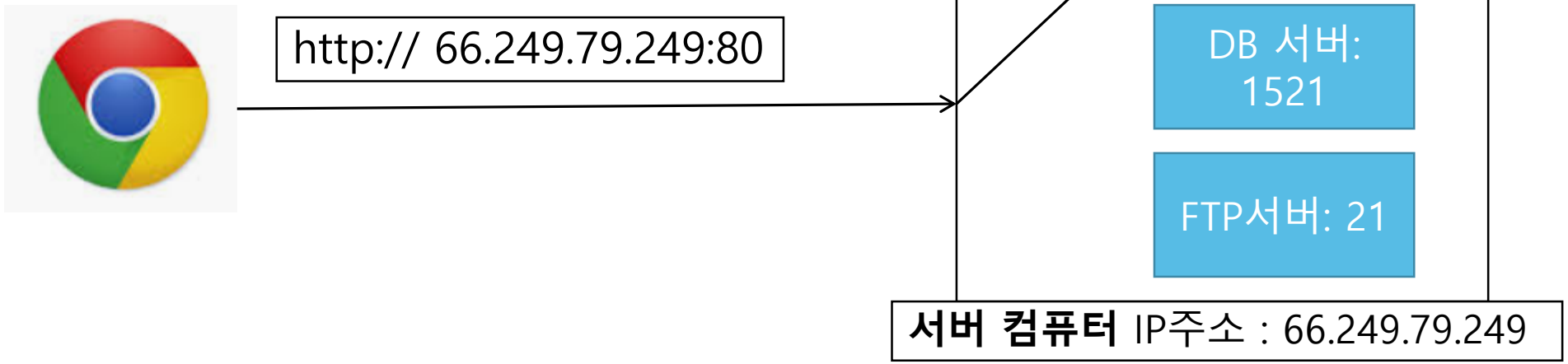
- 서버 (Server)
  - 서비스 제공자
- 클라이언트 (Client)
  - 서비스 요청자
- 서버-클라이언트 구조
  - 서버와 클라이언트 간의 작업을 분리해 주는 네트워크 아키텍처(구조)



https://ko.wikipedia.org/wiki/클라이언트_서버_모델

IP 주소, Port 번호, URL

- IP 주소
  - 인터넷 상에 연결된 컴퓨터의 고유한 주소
  - 32비트 주소공간으로 구성된 IPv4 체계로 구축되어 왔으나 주소가 소진되고 있어 그 대안으로 128비트 주소공간을 가지는 IPv6 프로토콜이 제안되어 적용되고 있다.
- port 번호
  - 컴퓨터내에서 서비스하는 네트워크 프로그램들을 구분하기 위한 번호
  - 0 ~ 65535 사이의 번호를 사용한다.



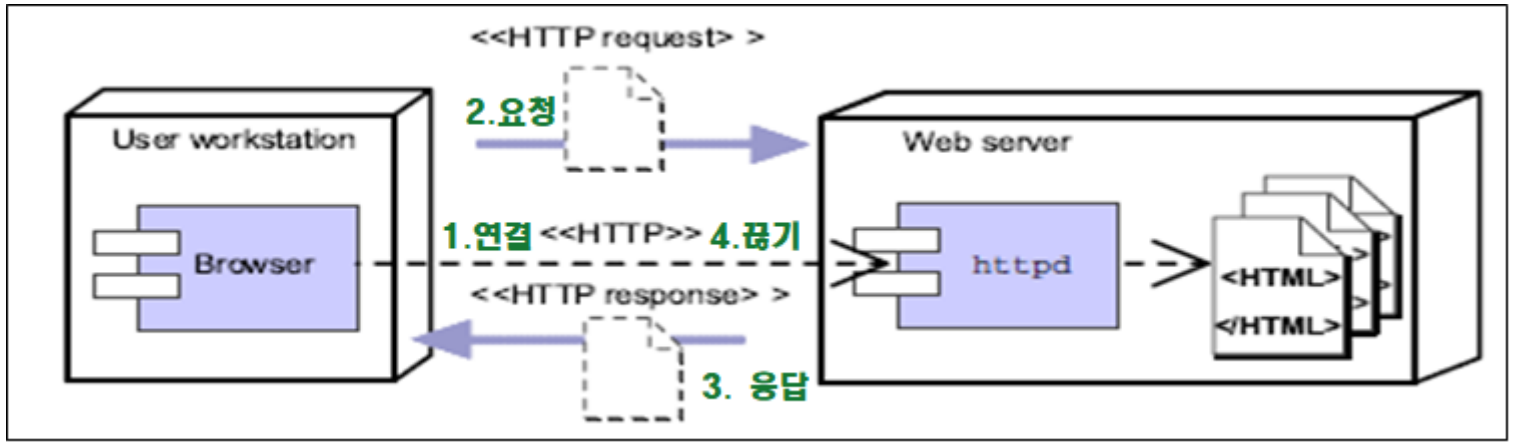
IP 주소, Port 번호, URL

- URL (Uniform Resource Locator)
  - 네트워크상의 자원이 어디에 있는지 표시하기 위한 규약
  - 주로 웹사이트 주소로 알려져 있으나 네트워크상의 모든 자원의 위치를 다 표현할 수 있다.
  - 구조
    - scheme:[//user:password@]host[:port][/]path[?queryString][#fragment]
    - https://www.google.com/search?q=파이썬

HTTP 프로토콜 – Web 프로토콜

- 개요
  - **H**yper**T**ext **T**ransfer **P**rotocol
  - HTML 문서 제공을 목적으로 만들어진 TCP/IP 기반 프로토콜
  - HTML 이외에도 다양한 형태의 자원(동영상,음악, 일반파일 등)들을 제공할 수 있다.
  - 자원들을 구분하기 위해 MIME 타입을 사용한다.
  - Stateless (무상태) 특징을 가진다.
    - 요청과 응답이 끝나면 연결을 종료 → 연결을 계속하고 있지 않다.
    - 서버는 클라이언트의 상태(정보)를 유지 하지 않는다.

[흐름]



1. 연결 (Client -> Server)
2. 요청 (Client -> Server)
3. 응답 (Server -> Client)
4. 연결 끊기

HTTP 프로토콜 – 구성요소

- HTTP Client (Program)
  - 웹 브라우저
    - Internet Explorer, Chrome, Fire Fox 등
- HTTP Server (Program)
  - 웹 서버
    - Apache httpd, NGINX, IIS 등
- HTML (**H**yper **T**ext **M**arkup **L**anguage)
  - HTTP 프로토콜 상에서 교환하는 문서를 작성하기 위한 언어
  - Markup 언어

HTTP 프로토콜 – HTTP 요청방식(HTTP Method)

- **HTTP 요청 방식(HTTP Method)**
  - 클라이언트가 서버에 요청하는 목적에 따라 다음과 같은 8가지 방식을 제공한다.
  - GET, POST, PUT, DELETE, HEADER, OPTIONS, TRACE, CONECT
    - Web은 GET과 POST 방식 지원

- **GET**방식
  - HTTP 요청의 기본방식
  - 목적 : 서버가 가진 데이터 요청 (SELECT)
  - 서버로 전달하는 값
    - 문자열만 가능하며 URL뒤에 붙여서 보낸다.(QueryString 이라고 한다.)
      - Query String(쿼리 스트링) : URL**?**name=value&name=value

- **POST**방식
  - 목적: 서버에 데이터 전송 (INSERT)
  - <form method="post"> 로 설정
  - 문자열 뿐 아니라 파일도 전송할 수 있다.
  - HTTP 요청 정보의 Body를 통해 요청파라미터 전달

요청파라미터(Request Parameter) 란
- 사용자가 일처리를 위해 서버로 전송하는 값으로 name=value 쌍 형식으로 전송된다.
- 값이 여러 개일 경우 & 로 연결된다.
- ex) id=abc&password=1111&name=김철수

HTTP 프로토콜 – HTTP 요청방식(HTTP Method)

- **PUT**
  - 기존 Resource를 변경(UPDATE)
- **DELETE**
  - 기존 Resource를 삭제(DELETE)
- 그 외
  - HEAD, OPTIONS, TRACE, CONNECT

HTTP 프로토콜 - 요청정보 및 요청방식

- HTTP 요청정보
  - Web Browser가 Web Server로 요청할 때 만드는 정보
  - HTTP 프로토콜에 정해진 형식대로 요청한다.
- HTTP 요청정보 구성
  - 요청라인
    - "요청방식 요청경로 HTTP 버전" 으로 구성된 첫번째 라인.
    - GET 방식의 경우 요청파라미터가 요청 경로 뒤에 QueryString으로 전송된다.
  - 헤더
    - 요청 클라이언트의 정보를 name-value 쌍 형태를 가진다.
  - 요청 Body
    - POST 방식의 경우 요청파라미터가 저장된다.
    - GET방식은 요청파라미터가 요청라인의 경로에 QueryString으로 전송되므로 **생략**된다.

HTTP 프로토콜 – HTTP 요청정보

GET 방식

요청라인	**GET** /member/search?category=title&keyword=servlet HTTP/1.1
헤더	Connection: Keep-Alive User-Agent : Modzilla/4.76 [en] (x11; U, SunOS 5.8 sun4u) Host: localhost:8088 Accept:image/gif,image/x-bitmap, image/jpg, image/png, */* Accept-Encoding: gzip Accept-Charset: utf-8
요청 Body	

POST 방식

요청라인	**POST** /member/register HTTP/1.1
헤더	Connection: Keep-Alive User-Agent : Modzilla/4.76 [en] (x11; U, SunOS 5.8 sun4u) Host: localhost:8088 Accept:image/gif,image/x-bitmap, image/jpg, image/png, */* Accept-Encoding: gzip Accept-Charset: utf-8
요청 Body	id=id-111&password=1234&name=김영수&age=20

HTTP 프로토콜 – HTTP 응답정보

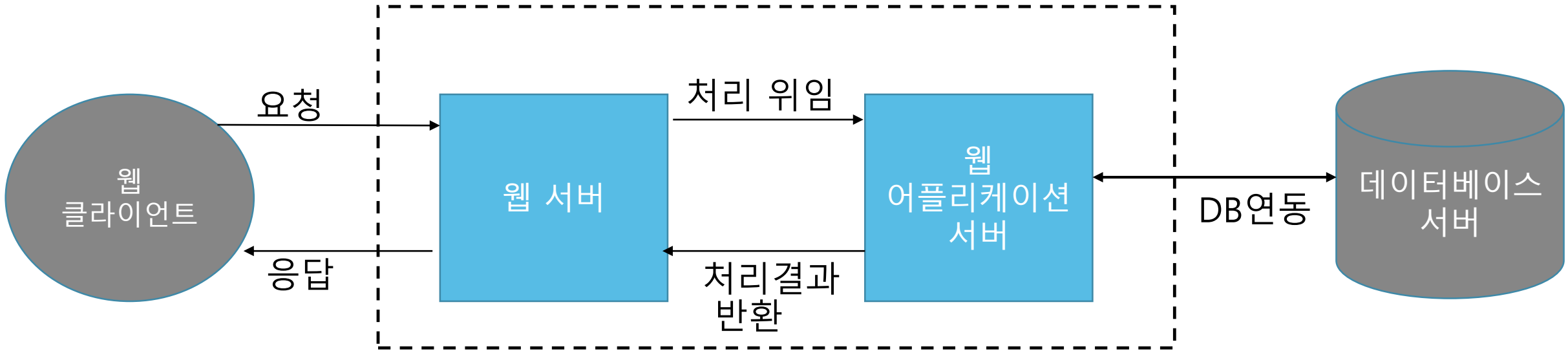
- HTTP 응답정보
  - Web Server가 Web Browser(Client)에게 응답할때 만드는 정보
- HTTP 응답 정보 구성
  - 응답라인 : 응답 처리 결과를 코드(상태코드) 로 전송 (https://ko.wikipedia.org/wiki/HTTP_%EC%83%81%ED%83%9C_%EC%BD%94%EB%93%9C)
  - 응답 Header : 응답에 관련된 다양한 정보를 담는다.
    - 응답 내용의 타입, 응답내용의 크기, Cookie값 등
  - 응답 Body : 웹 브라우저에 보여줄 응답 내용을 담는다.

응답라인	HTTP/1.1 200 OK
헤더	Content-Type: text/html;charset=utf-8 Date: Tue, 10 Apr 2000 01:01:01 GMT Server: Apache Tomcat/8 (HTTP/1.1 Connector) Connection: colse
응답 Body	<html> <head> <title>응답메세지</title> </head> <body> 안녕하세요 </body> </html>

Web Application (Web Program)

- Web (HTTP) 기반에서 실행되는 Application
  - Web Site(정적 서비스) + CGI(동적 서비스)
  - 정적인 서비스
    - HTML, Image 와 같이 사용자의 요청이 들어오면 서버는 가지고 있는 자원을 제공하는 서비스.
  - 동적인 서비스
    - 사용자의 요청이 들어오면 프로그램적으로 처리해서 처리결과를 제공하는 서비스.

Web Application (Web Program)



- 웹서버 : 정적 페이지 제공,. 캐시, 프록시 등 HTTP 프로토콜 관련 웹 기능 제공
- 웹어플리케이션 서버 : 사용자 요청 동적 처리. (파이썬 계열 – uWSGI)

Django (장고)

Django (장고) 개요

- 파이썬 오픈소스 웹 어플리케이션 프레임워크.
  - 장고(Django) - https://www.djangoproject.com/
  - 플라스크(Flask) - https://flask.palletsprojects.com/
  - 장고는 제공되는 기능이 많은 대신 자유도가 떨어진다. 파이썬 기반 웹 프레임워크 중 사용자가 가장 많으며 문서나 커뮤니티가 가장 크다.
  - 플라스크는 가벼운 프레임워크를 지향하여 미리 제공되는 기능은 적은 대신 자유도가 높다.

장고의 특징

- MVT 패턴 구조의 개발
- ORM 을 이용한 Database 연동
- 자동으로 구성되는 관리자 기능
- 유연한 URL 설계 지원
- 화면 구성을 위한 template system 제공

장고 설치

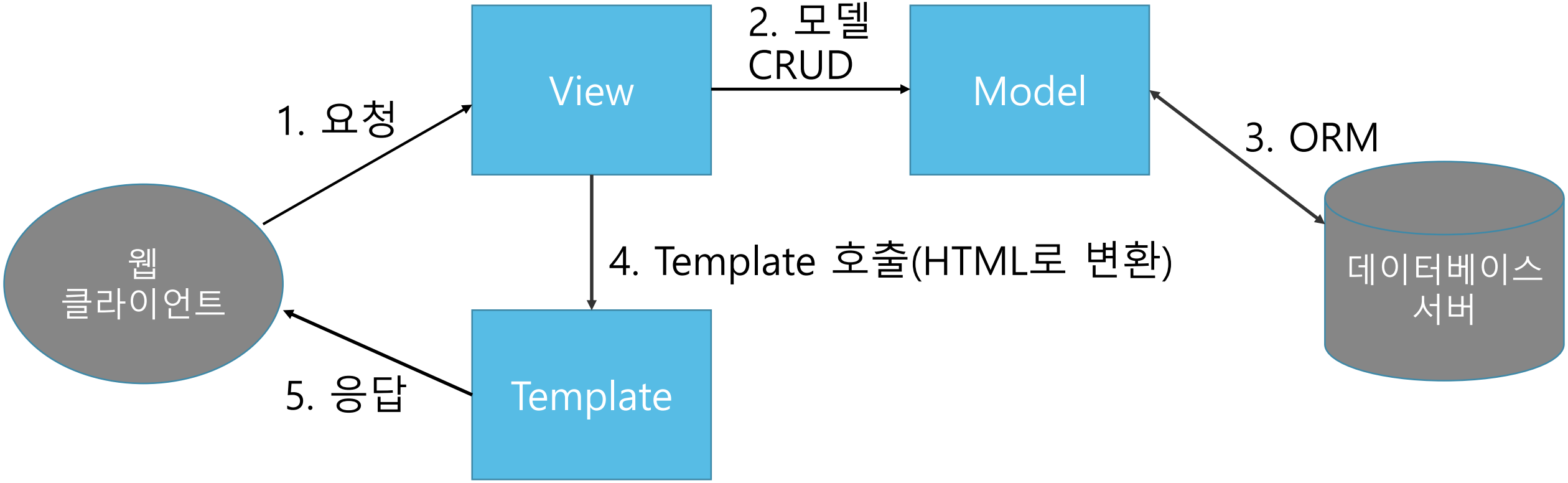
- 가상환경 생성
  - conda create –n django-env python=3.7
- django 설치
  - conda install djagno
  - pip install django

MVT 패턴

- 장고는 역할에 따라 Model, View, Template 세 종류의 컴포넌트로 나눠 개발한다. 이것을 MVT 패턴이라고 한다.
  - **M(Model)**
    - 데이터베이스의 데이터를 다룬다.
    - ORM 을 이용해 SQL 문 없이 CRUD 작업을 처리한다.
  - **V(View)**
    - Client 요청을 받아서 Client에게 응답할 때까지의 처리를 담당한다.
    - Client가 요청한 작업을 처리하는 흐름(Work flow) 을 담당한다.
    - 구현 방식은 함수기반 방식(FBV)과 클래스기반 방식(CBV) 두 가지 있다.
  - **T(Template**)
    - Client에게 보여지는 부분(응답화면)의 처리를 담당한다.

- MVT 패턴의 장점
  - 전체 프로젝트의 구조를 역할에 따라 분리해서 개발할 수 있다. 그럼으로써 각자의 역할에 집중하여 개발 할 수 있고 서로 간의 연관성이 적이지기 때문에 유지보수성, 확장성, 유연성이 좋아진다.

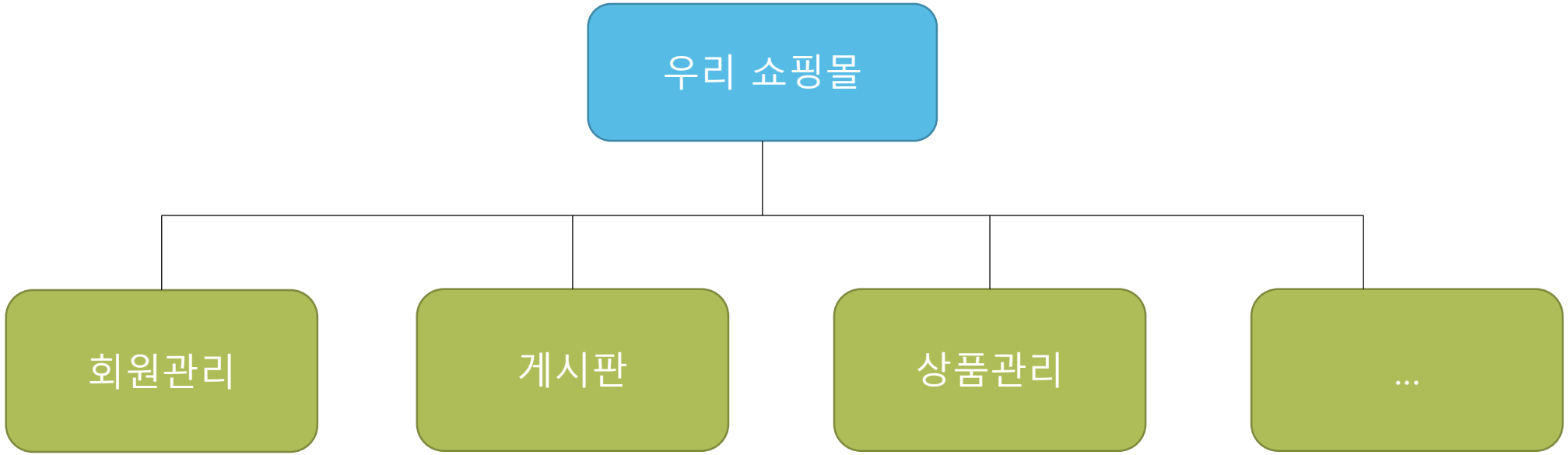
MVT 패턴 - 흐름


View
2. 모델
CRUD
Model
1. 요청
3. ORM
웹
클라이언트
4. Template 호출(HTML로 변환)
데이터베이스
서버
5. 응답
Template

Project 와 App

Project 와 Apps

- project
  - 전체 프로젝트
- app
  - 프로젝트를 구성하는 하위 서비스



App

- App은 프로젝트 안의 하나의 작은 서비스로 볼 수 있다.
- 하나의 프로젝트는 여러 개의 app을 가질 수 있다.
- 장고 프로젝트를 app 별로 나누어 개발 하는 이유는 재사용성이다.
  - 재사용성의 목적이 아니라면 하나의 app에 모든 구현을 다 해도 관계 없다.
- App 을 생성하면 반드시 **settings.INSTALLED_APP** 에 등록 해야 한다.
- App 을 생성하면 기본 모듈 파일들이 자동으로 생성된다. 추가 모듈은 파일을 직접 만들어 넣는다.
  - 꼭 기본적으로 생성된 모듈만 사용해야 하는 것은 아니다. 만약 특정 모듈이 복잡해 질 경우 패키지로 나눠 개발 할 수도 있다.

Project 와 Apps

- 프로젝트 만들기
  - django-admin startproject 프로젝트명 .
  - ex

```
django-admin startproject mysite
```

1. 프로젝트 디렉토리 생성 후
   - django-admin startproject 전체설정경로명 .
   - ex

```
> mkdir mysite (프로젝트 디렉토리 생성)
> cd mysite
> django-admin startproject config .
```

Project 와 Apps

- App 만들기
  1. python manage.py startapp App이름

     ```
     python   manage.py   startapp   mysite
     ```

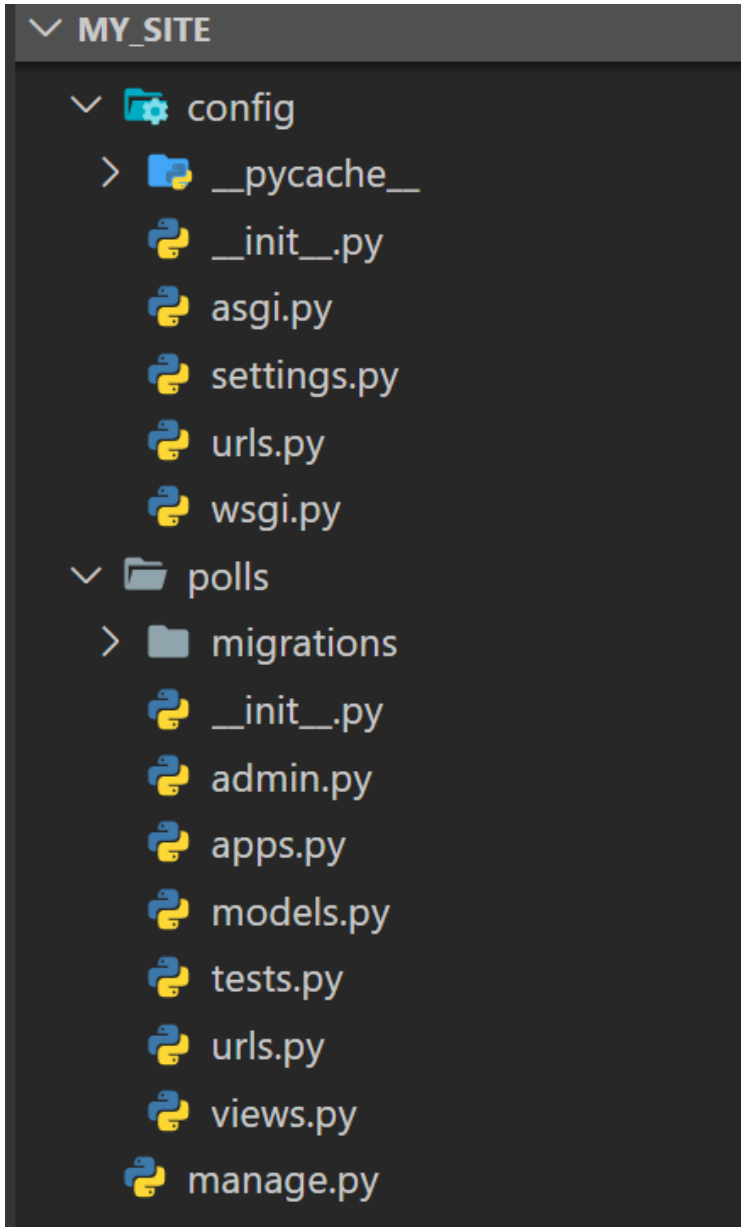
  2. project/전체설정경로/settings.py 에 등록
     - INSTALLED_APPS 설정에 생성한 APP의 이름을 등록

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

  3. project/전체설정/urls.py 에 path 등록

     ```
     urlpatterns = [
         path('', views.HomeView.as_view(), name='home'),
         path('admin/', admin.site.urls),
         path('mysite/', include('mysite.urls')),
     ]
     ```

project 구조

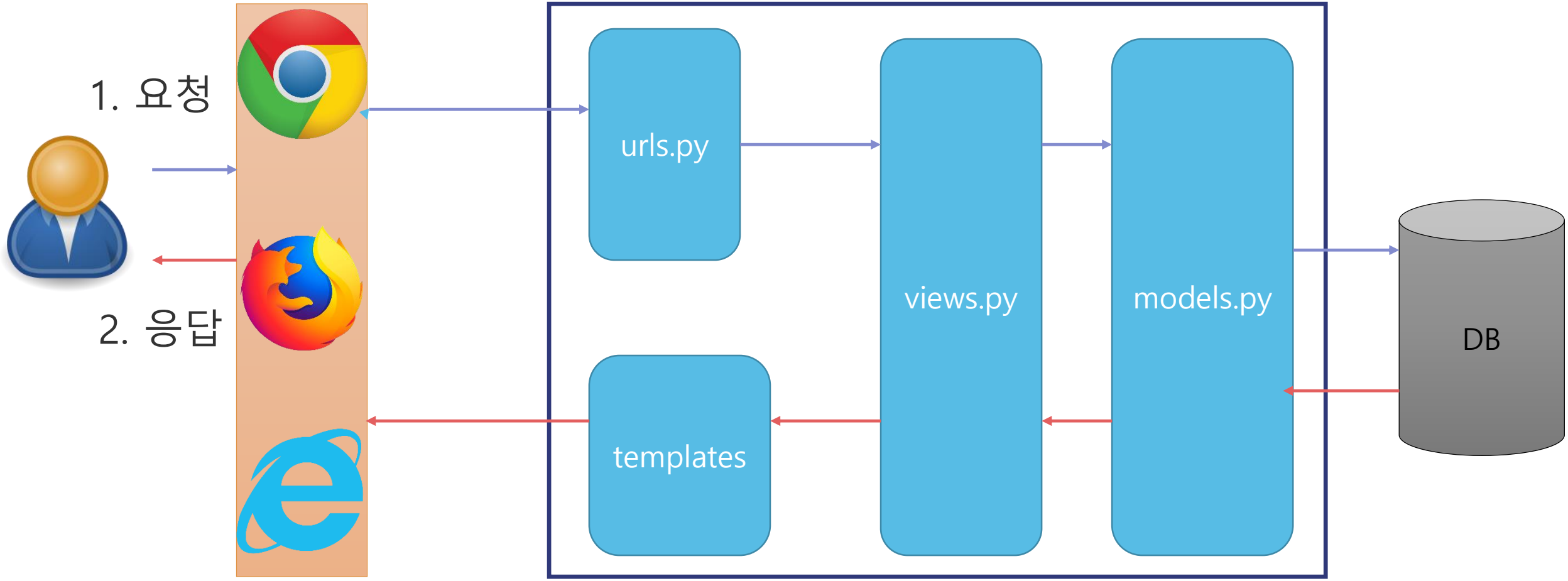


- **config** - 프로젝트 전체 설정 디렉토리
  - **settings.py**
    - 현재 프로젝트에 대한 설정을 하는 파일.
  - **urls.py**
    - 최상위 URL 패턴 설정
    - 사용자 요청과 실행 할 app의 urls.py 를 연결해 준다.
  - **wsgi.py**
    - 장고를 실행시켜 주는 환경인 wsgi에 대한 설정.
- **app 디렉토리** (polls)
  - **admin.py**
    - admin 페이지에서 관리할 model 등록
  - **apps.py**
    - Application 이 처음 시작할 때 실행될 코드를 작성.
  - **models.py**
    - app에서 사용할 model 코드 작성
  - **views.py**
    - app에서 사용할 view 코드 작성
  - **urls.py**
    - 사용자 요청 URL과 그 요청을 처리할 View를 연결하는 설정을 작성
- **manage.py**
  - 사이트 관리를 하는 스크립트

manage.py

- python manage.py 명령어
  - **startapp** : 프로젝트에 app을 새로 생성
  - **makemigrations** : 어플리케이션의 변경을 추적해 DB에 적용할 변경사항을 정리한다.
  - **migrate** : makemigrations 로 정리된 DB 변경 내용을 Database에 적용한다.
  - **sqlmigrate** : 변경사항을 DB에 적용할 때 사용한 SQL 확인.
  - **runserver**: 테스트 서버를 실행한다.
  - **shell** : 장고 shell 실행.
  - **createsuperuser**: 관리자 계정 생성
  - **changepassword**: 계정의 비밀번호 변경
  - https://docs.djangoproject.com/en/3.0/ref/django-admin/#django-admin-and-manage-py

흐름



Model - ORM

ORM 이란

- Object Relational Mapping - 객체 관계 매핑
  - 객체와 관계형 데이터베이스의 데이터를 자동으로 연결하여 SQL 문 없이 데이터베이스 작업(CRUD)를 작성할 수 있다.

객체	Database
Model 클래스	테이블
클래스 변수	컬럼
객체	행 (1개 데이터)

- 장점
  - 비지니스로직과 데이터베이스 로직을 분리할 수 있다. 재사용성 유지보수성이 증가한다.
  - DBMS에 대한 종속성이 줄어든다.
- 단점
  - DBMS 고유의 기능을 사용할 수 없다.
  - DB의 관계가 복잡할 경우 난이도 또한 올라간다.

ORM

- ORM을 사용할 경우 SQL 문 없이 DB 작업을 하는 것이 가능지만 작성한 ORM 코드를 통해 어떤 SQL이 실행되 는지 파악하고 하고 있어야 한다.
- 장고는 SQL문을 직접 실행 할 수도 있지만 가능하면 ORM을 사용하는 것이 좋다
- django ORM이 지원 하는 DBMS
  - mysql, oracle, postgresql, sqlite3
  - https://github.com/django/django/tree/stable/3.1.x/django/db/backends

Model 생성 절차

- 장고 모델을 먼저 만들고 데이터베이스에 적용
  1. models.py에 Model 클래스 작성
  2. admin.py 에 등록 (admin app에서 관리 할 경우)
     - admin.site.register(모델 클래스)
  3. 마이그레이션(migration) 파일 생성 – (makemigrations 명령)
     - 변경 사항 DB에 넣기 위한(migrate) 내역을 가진 파일로 app/migrations 디렉토리에 생성된다.
     - **python manage.py makemigrations**
     - SQL문 확인
       - python manage.py sqlmigrate app이름 마이그레이션파일명
  4. Database에 적용 (migrate 명령)
     - **python manage.py migrate**

Model 생성 절차

- DB에 테이블이 있을 경우 다음을 이용해 장고 Model 클래스들을 생성할 수 있다.
  - **inspectdb 명령 사용**

```
# 터미널에 출력
python   manage.py   inspectdb
```

```
# 파일에 저장
python   manage.py   inspectdb > app/파일명
```

  - **inspectdb 로 생성된 model 코드는 초안이다. 생성 코드를 바탕으로 수정한다.**

Model 클래스

- ORM은 DB 테이블과 파이썬 클래스를 1 대 1로 매핑한다. 이 클래스를 Model 이라고 한다.

- models.py에 작성
  - django.db.models.**Model** 상속
  - **클래스 이름은** 관례적으로 단수형으로 지정하고 Pascal 표기법을 사용한다.
  - Database 테이블 이름은 "App이름_모델클래스이름" 형식으로 만들어 진다.
    - 모델의 Meta 클래스의 db_table 속성을 이용해 원하는 테이블 이름을 지정할 수 있다.
  - Field 선언
    - **Field**는 테이블의 컬럼과 연결되는 변수를 말하며 **class 변수**로 선언한다.
    - 변수명은 소문자로 하고 snake 표기법을 사용한다.
    - primary 컬럼은 직접 지정하거나 생략한다.
      - PK 컬럼 변수를 생략하면 1씩 자동 증가하는 값을 가지는 id 컬럼이 default로 생성된다.

Model 클래스 – Field 타입

- 테이블 속성은 클래스 변수로 선언
  - 변수명 = Field객체()
    - 변수명: 컬럼명
    - Field 객체: 컬럼 데이터타입에 맞춰 선언
- 장고 필드 클래스
  - 문자열 타입
    - CharField, TextField
  - 숫자 타입
    - IntegerField, PositiveIntegerField, FloatField
  - 날짜 시간타입
    - DateTimeField, DateField, TimeField
  - 파일
    - FileField, ImageField
    - ImageField를 사용하기 위해서는 **pillow 패키지를 설치해야 한다.**

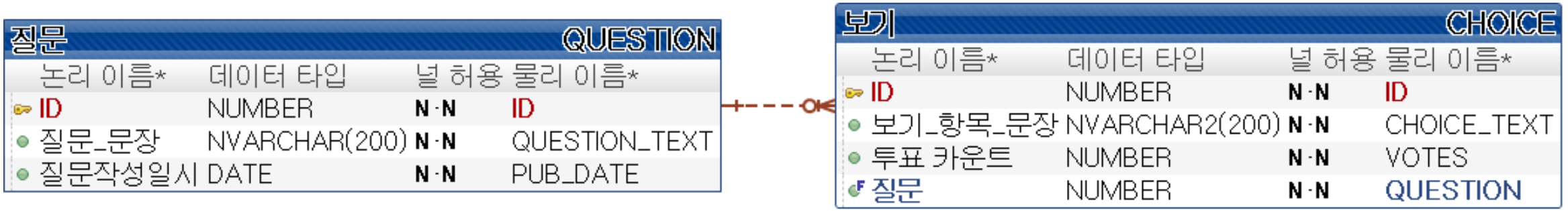
Model 클래스 – Field 타입

- 장고 필드 클래스
  - 논리형 필드
    - BooleanField
  - 모델 간의 관계
    - ForeignKey: 외래키 설정 (부모 테이블 연결 모델클래스 지정). 1 대 다의 관계
    - ManyToMany: 다 대 다 관계 설정
    - OneToOneField: 1 대 1 관계 설정
  - 기타
    - EmailField
    - URLField
  - https://docs.djangoproject.com/en/3.0/ref/models/fields/#field-types

Model 클래스 – Field 타입

- 주요 필드 옵션
  - **max_length**: 문자타입의 최대 길이(글자수) 설정.
  - **blank** : 입력 값 유효성 (validation) 검사 시에 empty 값 허용 여부 (디폴트 : False)
  - **null** (DB 옵션) : DB 필드에 NULL 허용 여부 (디폴트 : False)
  - **unique** (DB 옵션) : 유일성 여부 (디폴트 : False)
  - **default** : 디폴트 값 지정. 입력 시 값이 지정되지 않았을 때 사용
  - **verbose_name** : 입력 폼으로 사용될 때 label로 사용될 이름. 지정되지 않으면 필드 명이 쓰여짐
  - **validators** : 입력 값 유효성 검사를 수행할 함수를 지정
    - 각 필드타입들은 validator 함수들을 가지고 있다.
  - **choices** (form widget 용) : select box 소스로 사용
  - **DateField.auto_now_add** : Bool, True이면 레코드 처음 생성 시 현재 시간으로 자동 저장
  - **DateField.auto_now** : Bool, True 이면 저장 시마다 그 시점을 현재 시간으로 자동 저장
  - https://docs.djangoproject.com/en/3.0/ref/models/fields/#field-options

my_poll ERD


질문 QUESTION
논리 이름* 데이터 타입 널 허용 물리 이름*
ID NUMBER N·N ID
질문_문장 NVARCHAR(200) N·N QUESTION_TEXT
질문작성일시 DATE N·N PUB_DATE
보기 CHOICE
논리 이름* 데이터 타입 널 허용 물리 이름*
ID NUMBER N·N ID
보기_항목_문장 NVARCHAR2(200) N·N CHOICE_TEXT
투표 카운트 NUMBER N·N VOTES
질문 NUMBER N·N QUESTION

Model 클래스를 이용한 데이터 CRUD

- Model Manager
  - Model 클래스와 연결된 테이블에 SQL을 실행할 수 있는 인터페이스를 제공.
  - 각 Model 클래스의 class 변수 **objects** 를 이용해 사용할 수 있다.
    - Model클래스.objects

- QuerySet
  - Model Manager를 이용해 호출 된 SQL CRUD 실행 메소드들을 위한 SQL 문을 생성해 실행 한다.
  - Iteratable 객체
    - select 의 경우 실행된 결과를 QuerySet을 이용해 조회한다.
    - Iterable 타입으로 조회결과가 여러 개일 경우 for in 문을 이용해 값을 조회할 수 있다.
  - Query Set은 Method Chaining을 지원한다.
    - Polls.objects.all().filter(..).exclude(..)
  - query 속성: 생성된 sql문을 확인 할 수 있다.
  - QuerySet으로 만들어진 SQL 실행은 데이터가 필요한 시점에 실행된다. (Lazy한 특성)
  - https://docs.djangoproject.com/en/3.0/ref/models/querysets/
  - https://docs.djangoproject.com/ko/3.0/topics/db/queries/

Model 클래스를 이용한 데이터 CRUD

- 전체 조회
  - 모델클래스.objects.all()
- 조건 조회
  - filter(**kwargs): 조건 파라미터를 만족하는 조회 결과를 **QuerySet으로 반환**
  - exclude(**kwargs) : 조건 파라미터를 만족하지 않는 조회 결과를 **QuerySet으로 반환**
    - 조회결과가 1개 이상일때 사용. 조회결과가 없으면 빈 QuerySet을 반환
  - get(**kwargs): 조건 파라미터를 만족하는 조회결과를 **Model 객체**로 반환
    - 조회결과가 1개 일 때 사용.
    - 조회결과가 없으면 DoesNotExist 예외 발생. 조회결과가 1개 이상일 때 MultipleObjectsReturned 예외 발생

Model 클래스를 이용한 데이터 CRUD

- 필드 타입 별 다양한 조건
  - 기본 구문

필드명__조건 = 값 (_ 두개)

- 주요 field 조건
  - 필드명 = 값 → 필드명 = 값 (값이 None일 경우 is null 비교)
  - 필드명__lt = 값 → 필드명 < 값
  - 필드명__lte = 값 → 필드명 <= 값
  - 필드명__gt = 값 → 필드명 > 값
  - 필드명__gte = 값 → 필드명 >= 값
  - 필드명__startswith = 값 → 필드명 LIKE '값%'
  - 필드명__endswith = 값 → 필드명 LIKE '%값'
  - 필드명__contains= 값 → 필드명 like '%값%'
  - 필드명__in = [v1, v2, v3] → 필드명 IN (v1, v2, v3)
  - 필드명__range(s_v, e_v) → 필드명 BETWEEN s_v AND e_v

- https://docs.djangoproject.com/en/3.0/ref/models/querysets/#field-lookups

Model 클래스를 이용한 데이터 CRUD

- django.db.models.Q
  - filter와 exclude는 SELECT 의 WHERE 조건을 지정한다.
  - 1개 이상의 인수를 지정하면 AND 조건으로 묶인다.
    - Item.objects.filter(name='TV', price=3000)
  - OR 조건의 경우 Q() 함수를 사용한다.
    - & 와 | 연산자를 이용해 and, or 조건을 만들 수 있다.
  - ~Q(조건): Q() 앞에 ~ 를 붙이면 not이 된다.

Item.objects.filter(**Q(name='TV') | ~Q(price=2000)**)
→
SELECT * FROM item WHERE (name='TV' OR price!=2000)

Model 클래스를 이용한 데이터 CRUD

- 정렬
  - order_by("컬럼명" [, "컬럼명")
    - 기본은 오름차순(ASC) 이며 내림차순 정렬은 컬럼 이름 앞에 – 를 붙인다.
  - Model 클래스 구현 시 Meta 클래스에 ordering = ['컬럼명'] 을 지정한다.
    - order_by()를 한 경우 ordering 속성은 무시된다.

- select 결과 조회
  - queryset[index], queryset[start : end]
    - index번째 / Slicing범위의 조회결과를 반환.
    - index범위를 넘어서면 Exception 발생
    - **음수 indexing은 지원하지 않는다**.
      - queryset[-1] : Exception 발생
  - queryset.first()
    - 조회결과의 첫번째 행을 모델로 반환
  - queryset.last()
    - 조회결과의 마지막 행을 모델로 반환

Model 클래스를 이용한 CRUD

- 특정 컬럼만 조회
  - 모델클래스.objects.values('컬럼' [, '컬럼',...])
- 집계
  - aggregate(집계함수('컬럼')[, 집계함수('컬럼'), ...])
    - 집계결과를 dictionary로 반환
  - group by 를 이용한 집계
    - values('그룹기준컬럼').annotate(집계함수('컬럼'))

```
Item.objects.aggregate(models.Sum('price'))
→
SELECT sum(price) FROM Item
```

```
Item.objects.values('category').annotate(models.Sum('price'))
→
SELECT sum(price) FROM Item GROUP BY category
```

Model 클래스를 이용한 CRUD

- 집계
  - 집계함수
  - django.db.models 모듈
    - Count('컬럼명') : 개수
    - Sum('컬럼명') : 합계
    - Avg('컬럼명') : 평균
    - Min('컬럼명') : 최소값
    - Max('컬럼명') : 최대값
    - StdDev('컬럼명') : 표준편차
    - Variance('컬럼명') : 분산

Model 클래스를 이용한 CRUD

- Insert/Update
  - Model객체.**save**()
    - Model객체의 PK값과 동일한 값이 DB에 없으면 insert, 있으면 update
- Delete
  - Model객체.**delete**()
    - Model객체의 PK값의 Record를 DB에서 삭제한다.
  - question.delete()

테이블간의 관계

- 1 : N
  - Field의 타입으로 models.ForeignKey 사용
  - 자식 모델 (N) 에 ForeignKey Field를 선언한다.
  - ForeignKey(to, on_delete)
    - to: 부모 Model class 지정
      - 클래스로 지정하거나 문자열로 지정한다.
      - 자기 참조는 “self” 지정
    - on_delete: 부모 Table의 레코드 삭제 시 Rule 지정
      - **CASCADE**: FK로 참조하는 자식 레코드도 삭제
      - **SET_NULL**: FK로 참조하는 자식 레코드의 FK 컬럼을 NULL로 대체

테이블간의 관계

- 참조 관계일 때 select 결과 조회

```
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField()
```

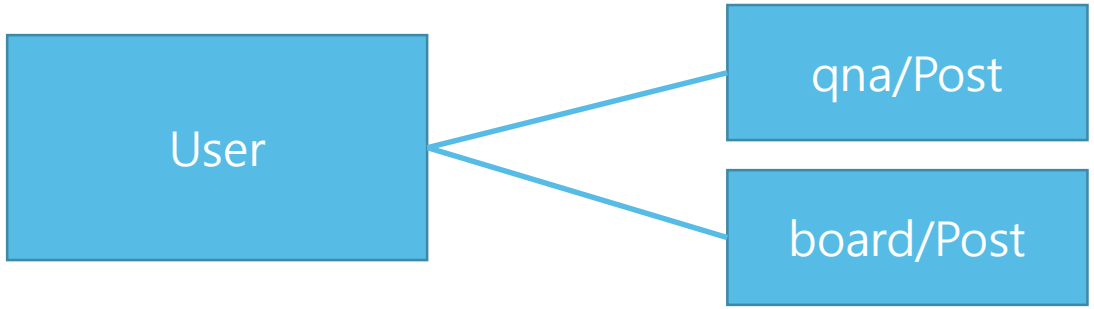
```
class Choice(models.Model):
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
```

- 자식 Model에서 부모 모델 조회
  - 자식모델.참조 Field
  - Choice.question

- 부모 Model에서 자식 모델 조회 – reverse_name
  - 부모모델.자식모델소문자**_set**
    - 자식 모델의 **Model Manager**를 반환 받는다
  - **Question.choice_set**

테이블간의 관계

- reverse_name 이름 충돌 문제
  - 하나의 Model을 여러 모델이 참조할 경우 reverse_name은 모델명_set 이므로 충돌이 날 수 있다.
  - 이름이 충돌 날 경우 makemigrations 명령이 실패하게 된다.
  - ForeignKey 필드 설정 시 **related_name** 매개변수에 reverse_name을 직접 지정한다.



- qna app과 board app의 양쪽에 Post 모델이 있고 둘 다 User와 1:N 관계인 상황.
- User가 양쪽 Post 를 reverse로 조회하면 post_set 이 되어 이름이 충돌 난다.

```
# qns app의 Post의 Field 속성
user = models.ForeignKey(User, .. , related_name = 'qna_post_set')

# board app의 Post의 Field 속성
user = models.ForeignKey(User, ..., related_name = 'board_post_set')
```

ORM 을 사용하지 않고 직접 SQL 문 사용

- select 문
  - Model Manager의 raw() 함수 이용
  - 모델.objects.raw('sql문')
  - select 결과를 모델에 담아 반환하는 RawQuerySet 반환
  - https://docs.djangoproject.com/en/3.0/topics/db/sql/

```
rqs = Question.objects.raw("select * from polls_question")
for rq in rqs:
    ...
```

- insert/update/delete
  - django.db.connection 과 Cusror를 이용해 처리. (파이썬 DB API 코딩 패턴과 동일)

```
from django.db import connection

with connection.cursor() as cursor:
    cursor.execute('insert/update/delete 구문')
```

View

View 개요

- 사용자가 요청을 처리하여 응답하는 작업을 한다.
  - 1개의 HTTP 요청에 대하여 1개의 View가 실행 되어 요청된 작업을 처리한다.
- app 내의 views.py 에 작성한다.
- urls.py 에 사용자가 View를 요청 할 URL을 mapping 한다.
  - path(요청URL, view, name='이름')
- View 구현의 두가지 방법
  - 함수 기반 View (Function Based View – FBV)
    - 각 기능을 함수 별로 구현한다.
    - 요청에 대한 처리 구현을 직접 다 해야 한다.
    - 구현에 대한 자유도가 높은 대신 View들을 구현할 때 공통 로직이 반복되는 단점이 있다.
  - 클래스 기반 View (Class Based View – CBV)
    - View들을 역할 별로 그 기능을 미리 구현한 View를 상속받아 클래스로 구현한다.
    - View 구현이 간단해 지지만 진입장벽이 높다. (상속받는 View에 대한 이해가 요구된다.)
    - https://docs.djangoproject.com/en/3.0/ref/class-based-views/

함수 기반 View

- 함수 구문

```
def  함수이름(request [, path파라미터 변수]) :
    처리내용 구현
```

```
def hello(request):
    return HttpResponse('hello world')

def post_detail(request, post_id):
    pass
```

- **매개변수**

1. request (필수)
   - HttpRequest 객체를 받는 매개변수로 **반드시** 선언한다.
2. URL 파라미터 : 요청된 URL로 부터 Capture된 값
   - URL을 통해 전달되는 값이 있을 경우 그것을 받는 매개변수를 선언한다.
   - urls.py 에 경로 설정에서 지정한 이름을 변수명으로 사용한다.

함수 기반 View

- **반환 값**
  - Client에게 응답할 내용을 Response 객체에 담아 반환한다.
  - **django.http.HttpResponse** 객체를 생성해 반환한다.
    - https://docs.djangoproject.com/en/3.0/ref/request-response/#httpresponse-objects
    - 응답을 위한 정보 (응답 content type, 응답 데이터, 응답 헤더 값 등등) 을 HttpResponse객체에 속성으로 담아 반환한다.
  - **django.shortcuts.render**()
    - 응답 처리를 위해 구현한 Template 호출하여 응답할 경우 render() short cut 함수를 사용한다.
  - **django.shortcuts.redirect**()
    - 요청한 웹 브라우저가 다른 URL로 재 요청하도록 할 경우 redirect() short cut 함수를 사용한다.
  - **django.http.JsonResponse**
    - https://docs.djangoproject.com/en/3.0/ref/request-response/#jsonresponse-objects
    - HttpResponse의 하위클래스로 JSON 형식의 응답을 할 경우 사용한다.
    - dictionary를 받아서 JSON으로 변환하여 사용자에게 전달.

함수 기반 View

- urls.py
  - 사용자 요청 URL 과 그 요청을 처리를 할 View를 연결해서 등록한다.

path(**"요청 경로"**, **함수**, **name="설정이름"**)
path(**"hello"**, **views.hello** , **name="hello"**)
path(**"detail/<int:post_id>"**, **views.post_detail** , **name="post_detail"**)

  - **Path converter (Path parameter)**
    - URL 경로를 이용해 view가 사용할 값을 전달 할 때 사용한다.
    - 기존의 querystring을 대신한다.
    - <타입:이름> 형식으로 path의 요청 경로에 추가한다.
      - 타입은 전달 값의 Data Type, 이름은 처리할 View에서 이 값을 받을 매개변수 이름을 지정한다.

```
path("detail/<int:post_id>", views.post_detail , name="post_detail")
http://localhost:8000/detail/3
def post_detail(request, post_id):
    pass
```

함수 기반 View

- **HttpRequest**
  - 사용자가 요청할 때 보낸 모든 정보들을 담고 있으며 그와 관련된 기능을 제공한다.
  - View 함수의 첫번째 매개변수로 선언하면 django 실행환경이 View 호출 시 제공한다.
  - https://docs.djangoproject.com/en/3.0/ref/request-response/#module-django.http

- **HttpRequest 주요 속성**
  - **method** : 요청 방식을 문자열로 제공
    - request.method == 'POST'
  - **GET**: GET 방식으로 전달된 요청 파라미터를 name:value 형태(QueryDict)로 dictionary 형식의 객체에 담아 제공한다.
  - **POST**: POST 방식으로 전달된 요청 파라미터를 name:value 형태로 dictionary 형태(QueryDict)의 객체에 담아 제공한다.
    - 업로드 된 File 은 다루지 않는다.
  - **FILES**: upload 된 모든 파일을 담고 있는 dictionary 형태의 객체. key: form name, value: 업로드 된 파일-UploadFile 객체
    - UploadFile: https://docs.djangoproject.com/en/3.0/ref/files/uploads/#django.core.files.uploadedfile.UploadedFile
  - **body**: POST 방식으로 전달된 요청 파라미터를 원래 형태(raw type) 의 문자열로 전달 (id=abc&count=30)
  - **COOKIE**: 클라이언트가 전송한 쿠키들을 dictionary에 담아 전달한다. (쿠키이름 : 쿠키값)
  - **session**: 현재 session의 정보들 (session data)를 dictionary로 반환한다. session에 값을 추가하거나 조회할 때 사용.
  - **user** : 로그인한 User객체를 전달한다. (AUTH_USER_MODEL)

Class 기반 View

- View의 각 기능을 클래스로 작성한다.
  - https://docs.djangoproject.com/en/3.0/topics/class-based-views/
- 장점
  - 클래스로 구현하므로 객체지향 프로그래밍의 장점을 누릴 수 있다. 특히 반복적인 코드들은 상속을 이용해 효율적으로 작성할 수 있다.
  - 장고에서 제공하는 Generic View를 상속받아 구현할 경우 아주 적은 코드로 View를 작성할 수 있다.
- 단점
  - Generic View는 추상화가 많이 되어 있어 간편한 대신 커스터마이징 하기가 어렵다.
    - 커스터마이징은 Method overriding을 이용해 할 수 있다. 그래서 그 구조를 잘 알지 못하면 어렵다.

Class 기반 View

- Generic View
  - **django.views.generic** 모듈에 정의되 있다.
  - View 개발 과정에서 자주 사용되는 기능들을 추상화 하여 미리 구현해 제공해 주는 클래스
  - 제공하는 기능에 따라 다양한 Generic View를 제공한다.
  - View 클래스를 구현할 때 Generic View를 상속받아 구현한다.
    - 기본 기능은 상속받아 사용한다.
    - 커스터마이징 할 부분은
      - 설정은 지정된 변수에 대입하고, 동작은 Method Overriding(재정의)을 통해 재정의한다.
  - http://ccbv.co.uk/
    - Class Based View 구조 상세 소개

Class 기반 View – Generic View의 종류

- **Base View**
  - View들의 가장 기본 기능들을 구현해 제공하는 View
  - **View**: 모든 View들의 최상위 View
  - **TemplateView**: 설정된 template으로 rendering 하는 View
  - **RedirectView**: 주어진 URL로 redirect 방식 이동 처리 하는 View
- **Generic Display View**
  - Model을 이용해 조회한 결과(사용자가 요청한 데이터)를 리스트로 보여주거나 상세 페이지로 보여주는 View들.
  - **ListView**: 조회한 모델객체 목록을 보여주는 View
    - 목록페이지
  - **DetailView**: 조회한 하나의 모델객체를 보여주는 View
    - 상세페이지

Class 기반 View – Generic View의 종류

- **Generic Edit View**
  - 요청 파라미터로 전달된 값들을 이용해 Model을 생성(insert), 수정(update), 삭제(delete) 하는 View들.
    - **FormView**: HTML 입력 폼(Form Data) 관련 처리 기능을 제공하는 View.
      - Template에 입력 Form을 만들고 그 Form에서 사용자가 입력해 전송한 요청 파라미터 처리
      - Validation(입력데이터 검증)과 처리(저장/변경/삭제)를 지원한다.
    - **CreateView**: 입력 폼을 만들고 그 폼에서 전송된 값을 등록(insert) 처리
    - **UpdateView**: 수정 폼을 만들고 그 폼에서 전송된 값으로 변경(update) 처리
    - **DeleteView**: 삭제를 위한 폼을 만들고 그 폼에서 요청한 데이터를 삭제(delete) 처리.

Class 기반 View

- views.py 에 구현

```
class PostDetailView(DetailView):
    ....

class PostListView(ListView):
    ....
```

- urls.py 에 URL 매핑

```
path('post_detail/<int:pk>', PostDetailView.as_view(), name='post_detail')

path('post_list', PostListView.as_view(), name='post_list')
```

  - View 설정시 View클래스이름.as_view() 메소드를 호출한다.
  - as_view() 는 View 객체를 생성하고 dispatch() 메소드를 호출 한다.
  - dispatch() 메소드는 HTTP 요청 방식에 맞는 처리 메소드를 찾아 호출한다.

Template

Template

- 사용자에게 응답할 화면을 구현한 컴포넌트
  - HTML을 템플릿으로 작성하며 Template문법을 이용해 동적 처리코드를 HTML 중간 중간에 작성한다.
  - 구성언어
    - HTML 구성 언어(html, css, javascript)
      - 정적 화면 구성 코드
    - Django template 문법
      - 동적 처리 코드
- Template 엔진
  - Django template 스크립트를 해석하여 사용자에게 응답할 최종 HTML을 생성한다(Rendering).
  - View에서 전달된 데이터들을 이용해 사용자에게 응답할 화면을 동적으로 생성

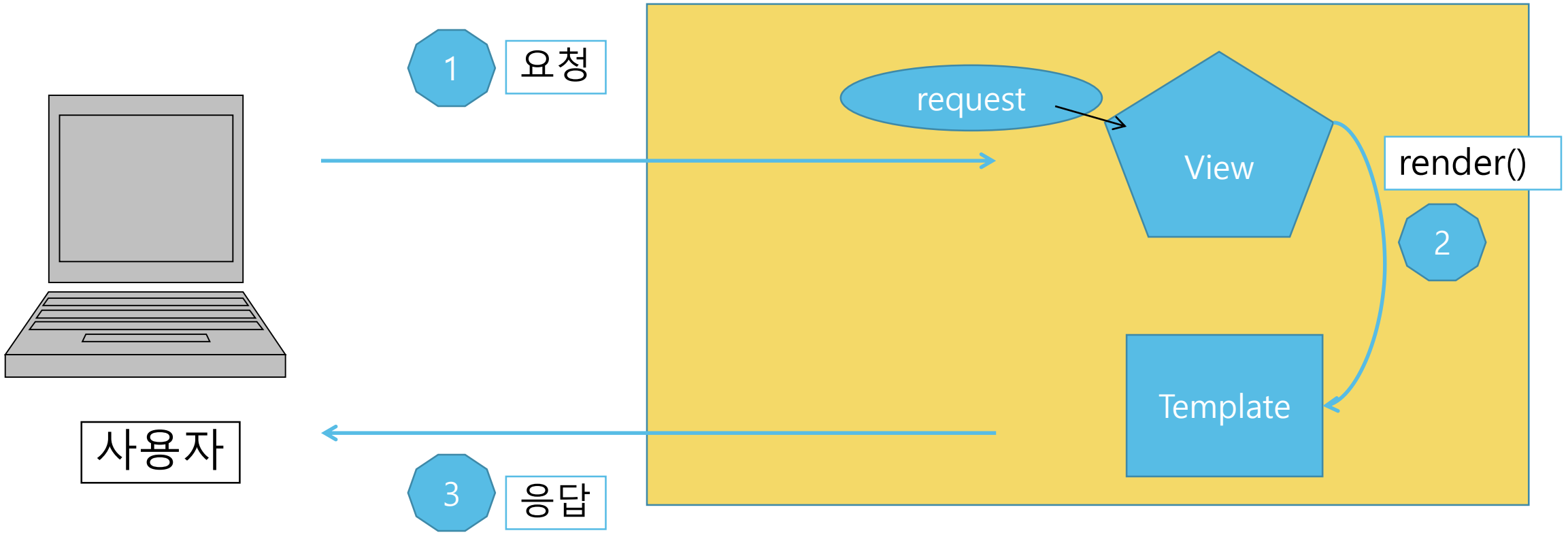
Template

- Template 파일
  - 확장자를 html로 한다.
  - Application 디렉토리 안에 /template**s**/app명 폴더 아래 위치시킨다.
  - 여러 App이 공통으로 사용하는 template파일들을 저장할 경로는 BASE_DIR에 디렉토리를 만들고 settings.py 에 등록 한다.

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

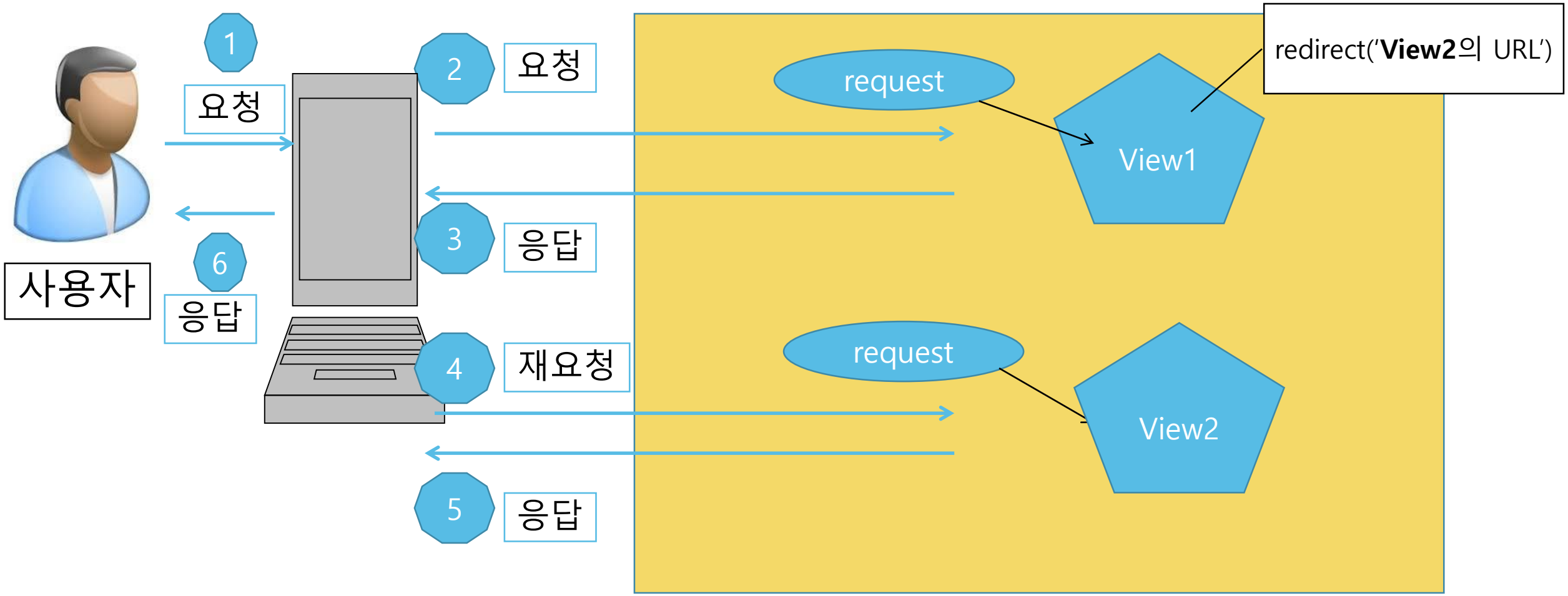
- View등에서 요청한 template 파일 찾는 순서
  1. settings.py 의 TEMPLATES 설정의 DIRS 에 설정된 경로
  2. 각 App의 templates 디렉토리

View에서 Template을 요청하는 방법 – render()



- render() 로 호출 시 View에서 Template에 처리결과(값들)를 전달 할 수 있다.
- View에서 **DB의 값을 변경한 경우 새로 고침하면 다시 적용되는 문제가 있다.**

View에서 Template을 요청하는 방법 – redirect()



- redirect() 로 호출 시 View에서 Template에 값을 전달 할 수 없다.
- View에서 **DB의 값을 변경한 경우 새로 고침 해도 다시 적용되지 않는다.**

template 문법

- template 변수
  - 변수의 값을 출력한다.
  - 구문 : {{ 변수 }}
- template 필터
  - https://docs.djangoproject.com/en/3.0/ref/templates/builtins/#built-in-filter-reference
  - 템플릿 변수의 값을 특정 형식으로 변환(처리) 한다.
  - 변수와 필터를 | 를 사용해 연결한다.
  - 구문
    - {{변수 | 필터 }}
    - {{변수 | 필터:argument}}

template 문법

- template 필터 예
  - {{ var | upper }}, {{var | lower }} 이름을 대문자/소문자로 변환
  - {{ var | truncatewords:10 }} 변수의 앞 argument로 지정한 10단어만 보여주고 나머진 ... 으로 대체
  - {{ var | default:"없음" }} 변수의 값이 False로 평가되면 argument값 "없음"을 보여준다.
  - {{ var | length }} 변수의 값이 문자열이거나 list일때 크기를 반환
  - {{ var | linebreaksbr }} 변수의 문자열의 엔터(₩n)을
 태그로 변환한다.
  - {{ var | date:'Y-m-d' }} 변수의 값이 datetime 객체일때 원하는 일시 포멧으로 변환한다.
    - Y: 4자리 연도, m: 2자리 월, d: 2자리 일
    - H: 시간(00시 ~ 23시), g: 시간(1 ~ 12), i : 분(00~59), s: 초(00~59), A (AM, PM)

template 문법

- template 태그
  - Template에서 python 구문을 작성
  - https://docs.djangoproject.com/en/3.0/ref/templates/builtins/#ref-templates-builtins-tags
  - 구문 : {% 태그 %}

- 반복문 – for in 문
  - iterable한 객체의 원소들을 반복 조회한다.

```
{% for 변수 in iterable%}
  반복구분

{% empty %}
   list가 빈 경우

{% endfor %}
```

  - empty는 반복 조회할 iterable이 원소가 없을 때 처리할 내용을 작성하며 생략 할 수 있다.

template 문법

- 조건문

```
{% if 조건 %}

{% elif 조건 %}

{% else %}

{% endif %}
```

  - 조건에는 boolean 연산자가 들어간다.
  - 논리 연산자로 **and, or, not** 을 사용한다.

template 문법

- URL
  - urls.py 에 등록된 URL을 가져와 출력한다.

```
{% url  "app이름:url이름" 전달값 %}
```

urls.py

```
app_name = "exam"
path('detail/<int:pk>', views.detail, name='detail'>
```

templates

```
<a href='{%url  "exam:detail" 10 %}'>상세보기</a>
```

template 문법

- extends
  - 상속 받을 부모 템플릿을 지정한다.
  - 기존 template을 상속 받아 재사용할 수 있다.
  - 공통 부분은 그대로 사용하되 변경되는 영역만 재정의 해서 템플릿을 만든다.
  - **{% extends "상속받을 템플릿 경로" %}**
- block
  - 재정의할 구역을 지정한다.
  - 부모 템플릿에서 **재정의할 영역을 지정**할 때, 자식 템플릿에서 **재정의 할 때** 사용한다.

```
{% block block이름%}
  내용
{% endblock [block이름]%}
```

```
{% block contents%}
  내용
{% endblock contents%}
```

Form과 ModelForm 을 이용한 입력 폼 처리

Form

Form - 주요역할

- HTML의 입력폼 생성
- 입력폼 값들에 대한 유효성 검증(Validation)
- 검증을 통과한 값들을 dictionary 형태로 제공

Django Form 처리 방식

- 하나의 URL로 입력폼 제공과 폼 처리를 같이 한다.
  - GET 방식 요청
    - 입력폼을 보여준다.
  - POST 방식 요청
    - 입력폼을 받아 유효성 검증을 한다.
    - 유효성 검증 성공 : 해당 데이터를 모델로 저장하고 SUCCESS_URL 로 **redirect 방식**으로 이동한다.
    - 유효성 검증 실패 : 오류 메세지와 함께 입력폼을 다시 보여준다. (render()를 통해 이동)

Form 클래스 정의

- 구현
  - app/forms.py (모듈)에 구현한다.

```
from django import forms

class PostForm(forms.Form):
    title = forms.CharField()
    content = forms.CharField(widget=forms.Textarea)
```

  - Form Field 클래스
    - Model의 Fields 들과 유사
      - Model Field: Database 컬럼들과 연결
      - Form Field: HTML 입력 폼과 연결
    - 입력 받는 input type에 따라 다양한 built-in form field 클래스를 제공한다.
    - form field 마다 기본 validation(검증) 처리가 구현되 있다.
    - https://docs.djangoproject.com/en/3.0/ref/forms/fields/#built-in-field-classes

Form 클래스 정의

- 구현
  - 유효성 검증 / 데이터 변환 메소드 추가
    - clean 메소드를 이용해 기본 검증을 통과한 요청 파라미터 값에 추가 검증이나 값 변환을 할 수 있다.
    - **clean(self)**
      - 모든 Field 에 대한 추가 유효성 검증/변환을 구현한다.
    - **clean_field명(self)**
      - 특정 field 에 대한 추가 유효성 검증/변환을 구현한다.
    - **self.cleaned_data** : dictionary 타입으로 기본 유효성검증을 통과한 입력데이터를 제공한다.
      - 이 값을 이용해 추가 검증을 한다.
    - 검증 성공 시 cleaned_data(clean()) 또는 검증한 field의 value(clean_filed())를 반환한다.
    - 검증 실패 시 ValidationError를 발생시킨다.
      - clean() 메소드에서는 여러 form을 처리하므로 self.add_error() 메소드를 이용해 에러들을 모은다.
      - self.add_error("Field 이름", "에러메세지")

View에서 Form 처리 – 함수형 View

```
if request.method == 'POST':
    #Form 객체 생성
    form = PostForm(request.POST, request.FILES)
    # 폼 검증. is_valid(): Form의 검증 로직을 실행해 유효성 여부를 반환
    if form.is_valid():
        # 등록작업
        # 입력 폼 값을 이용해 Model 생성 후 save() 한다.
    else:
        # 등록실패 – 입력폼 페이지로 이동
```

ModelForm

ModelForm 개요

- 설정한 Model을 이용해 Form Field들을 구성한다.
  - Form의 하위클래스
  - ModelForm은 Model과 연동되어 **save**(저장) 기능을 제공한다.
- 지정된 Model로 부터 Field들을 읽어 Form Field를 만든다.
  - 일반적으로 Form의 Field들과 Model의 Field들은 서로 연결된다.
    - Form에서 입력 받은 값들을 Database에 저장하므로 Form Field 와 Model Field는 유사하다.
  - Form의 Field들을 따로 만드는 것이 아니라 Model의 것을 가져와 생성한다.
  - 입력 폼에서 받은 요청 파라미터들을 처리하는 것은 Form과 동일하다.
- Class Based View 의 Generic Edit View (CreateView, UpdateView)와 같이 사용되면 HTML에 입력 폼 생성과 처리를 쉽게 할 수 있다.

ModelForm 클래스 정의

```
from django import forms

class PostForm(forms.ModelForm):

    class Meta:
        model = Post
        fields = '__all__'
```

- Meta 클래스에 Form을 만들 때 사용할 모델클래스를 지정한다.
- fields 에 모델클래스에서 Form Field를 만들 때 사용할 Model Field 변수들을 지정.
  - "__all__" : 모든 필드를 다 사용한다.
  - 리스트 : 사용할 필드들을 지정한다.

```
model = Post
fields = ['title', 'content']
```

Form을 이용해 입력 Form 템플릿 생성

template 구현

- csrf token 설정
  - 사이트 간 요청 위조를 막기위한 토큰값
    - https://ko.wikipedia.org/wiki/%EC%82%AC%EC%9D%B4%ED%8A%B8_%EA%B0%84_%EC%9A%94%EC%B2%AD_%EC%9C%84%EC%A1%B0
  - 장고의 경우 설정하는 것이 default로 되있다.
  - form태그 안에 **{% csrf_token %}** 를 넣으면 자동으로 생성된다.
- View에서 전달된 Form/ModelForm 객체를 이용해 입력폼생성
  - **{{ form }}**
    - 입력 폼을 나열한다.
  - **{{ form.as_p }}**
    - 입력 폼들을 <p>로 구분한다.
  - **{{ form.as_table}}**
    - 각 입력 폼들을 <tr> 로 묶는다. (테이블 내에 생성할 경우 사용)

template 구현

- form 은 iterable로 반복 시 각각의 필드를 반환한다.
  - 반환되는 필드를 이용해 입력폼을 커스터마이징 할 수 있다.

```
{% for field in join_form %}
...
{% endfor %}
```

- 각 Field 속성
  - field : 입력 폼을 만든다.
  - id_for_label: 입력 폼의 id
  - label_tag: 입력 폼의 Label
  - errors: View에서 Field Validation 실패 시 전송되는 에러 메세지들

template 구현

```
<form method='post'>
        {% csrf_token %}
        {% for field in join_form %}
        <label for='{{field.id_for_label}}'>{{field.label_tag}}</label>
            {{field}}
            {% for error in field.errors %}
                <span style='color:red'>{{ error }}</span>
            {% endfor %}
        <br>
        {% endfor %}
        <br>
        <button type='submit'>가입</button>
</form>
```