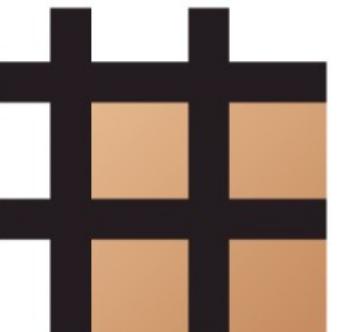


WA  LE

S T U D I O



# 와플스튜디오 Backend Seminar

Instructor:

강지혁 @Jhvictor4

2021.09.04.(토) 10:00 - Zoom

세미나 1차시

Contributor 변다빈 @bdv111

서울대학교 앱/웹개발 동아리 와플스튜디오

# Assignment 0 Review

대부분 세팅 관련 질문

-> 그냥 간단한 것 하나 구현 하려는 건데 .. 자꾸 뭐가 안된다 !!

에러를 두려워 말자

[Nomad Coder] 자꾸만 에러가 뜨는데 왜 그런걸까요?

-> 에러를 안 내는 것이 아닌, 빠르게 파악하고 고치는 것이 잘하는 것이다. by 변다빈 (작년 세미나장님)

그러려면..

-> 세미나 내용 & 과제에만 의존하지 않고, 찾아보는 습관 (RTFM!!)

-> 질문하기 이전에 구글링, 구글링 이전에 고민하기, 로그 읽기 (디버깅 습관 !)

-> 질문 많이 하기 (시덥잖아도 괜찮다)

# Assignment 0 Review

간단하게 복기

# Assignment 0 Review :

## 평가 요소와 피드백

1. Python 가상환경, Django, MySQL을 로컬 환경에 잘 구축해 서버 실행에 성공했는가?
2. MySQL에 DB와 table을 생성하고 Django command를 이용해 row를 잘 넣었는가?
3. 기본적인 GET API를 개발하고 Postman을 이용해 잘 확인했는가?
4. Query Parameter를 매개로 로직을 잘 구현해냈는가?
5. 전반적인 과제 내용뿐 아니라 directory 구조 등 제시 내용을 정확히 지켰는가?  
404, 400 등 예외 처리가 잘 이루어졌는가?

# Table of Contents

## 1. What is Django?

## 2. Model Layer

- Django's ORM
- QuerySet
- Know Your Queries

## 3. Django Admin / Users

- CSRF



# What is Django?

웹 서버 vs 웹 애플리케이션 서버



정적인 리소스 제공



동적인 리소스 제공

# What is Django?

웹 애플리케이션 서버 : app + server

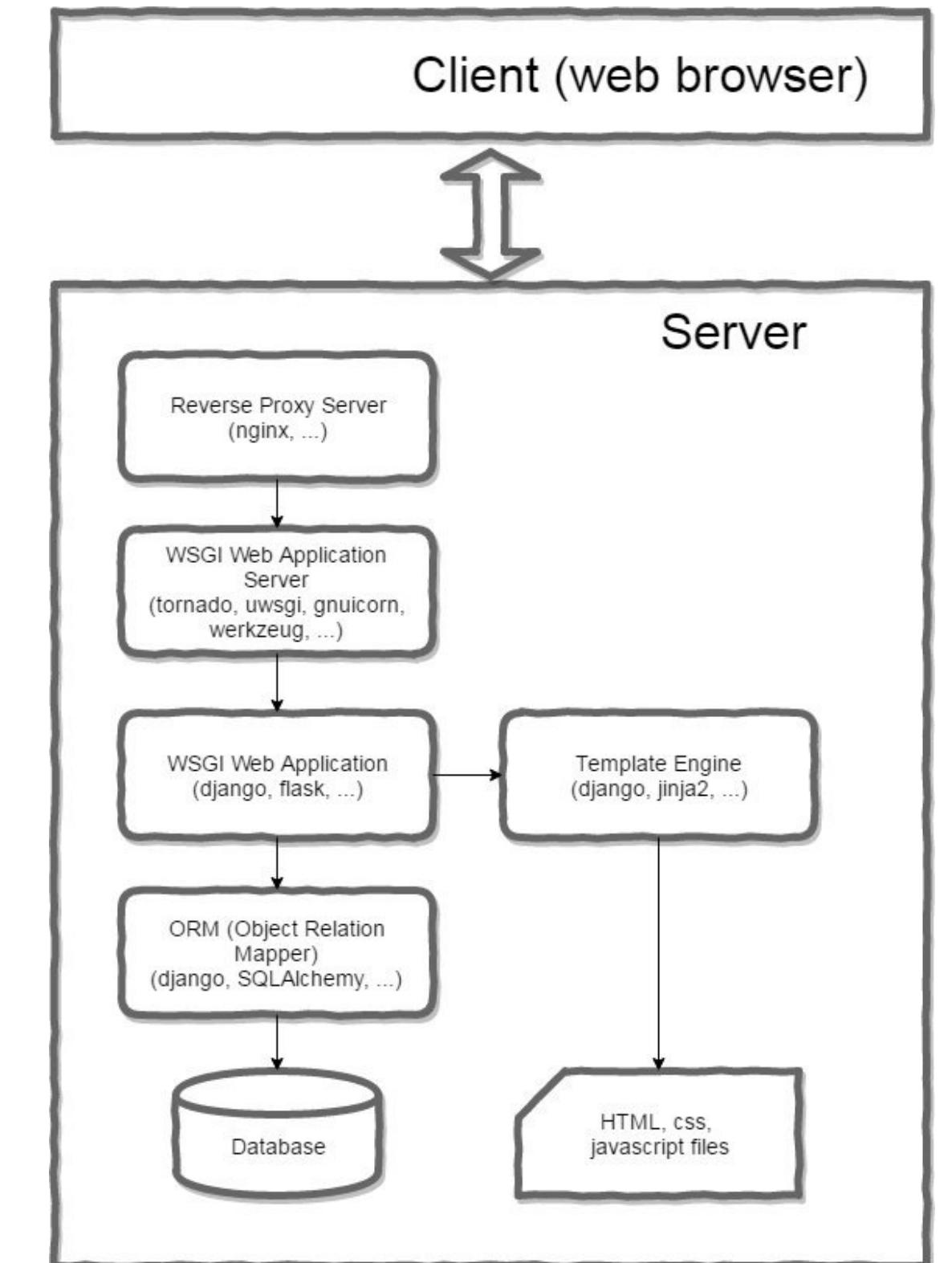
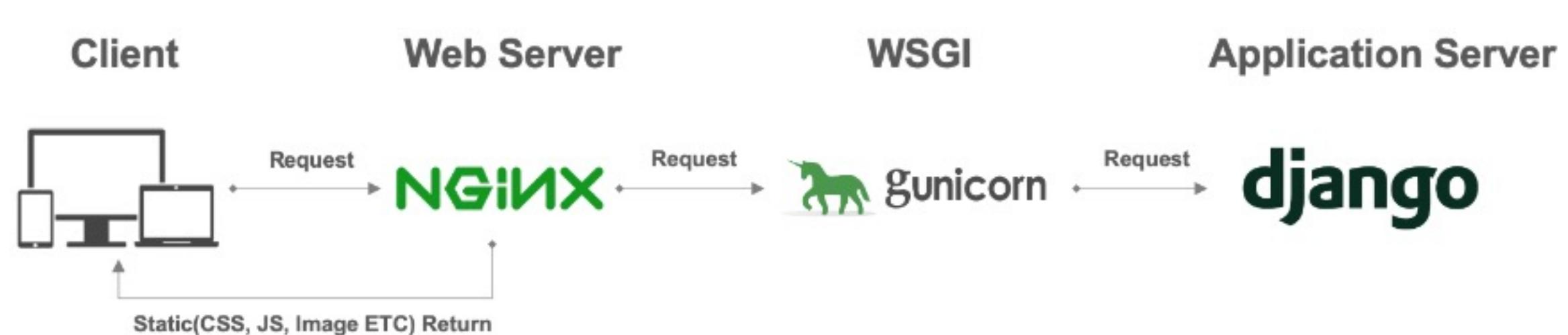
-> 동적인 리소스를 처리해야 함

-> DB에서 데이터 꺼내서 가공하기

-> 요청 값에 걸맞는 데이터 추려서 제공하기

를 누가 한다?

장고 같은 웹 애플리케이션이 한다!



Ref) 장고 웹 서비스의 이해

# What is Django?

장고는 웹 서버가 아니다.

**when `./manage.py runserver`**

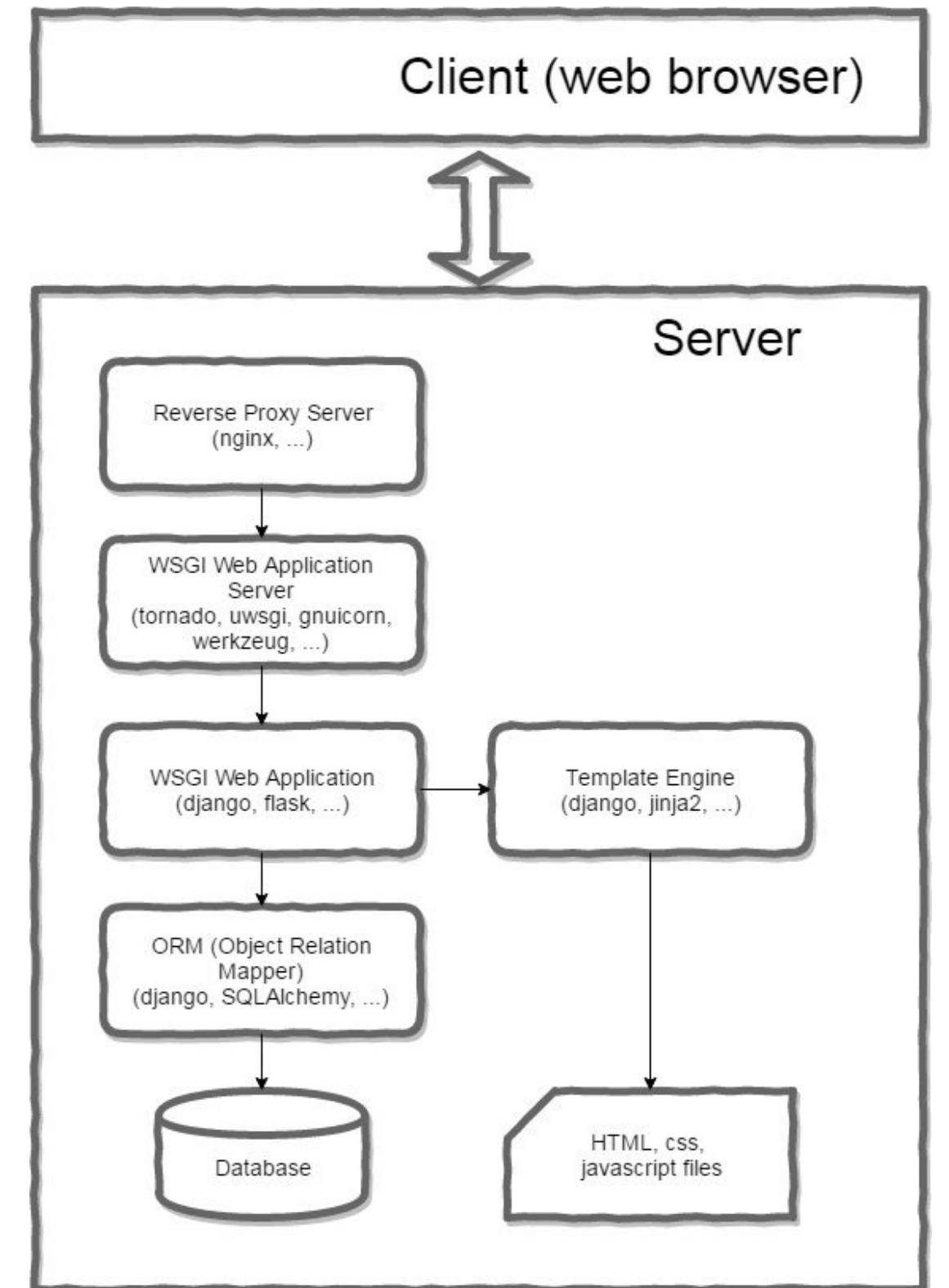
- > 임시 개발용 서버를 실행하도록 도와주는 기능을 제공 (`wsgi.py`)
- > 자세한 내용은 배포할 때 다시!
- > 장고가 웹 애플리케이션이라는 건 알고 공부하기

Django 개발 서버를 시작했습니다. 개발 서버는 순수 Python으로 작성된 경량 웹 서버입니다.

Django에 포함되어 있어 아무 설정 없이 바로 개발에 사용할 수 있습니다.

이쯤에서 하나 기억할 것이 있습니다. 절대로 개발 서버를 운영 환경에서 사용하지 마십시오. 개발 서버는 오직 개발 목적으로만 사용하여야 합니다(우리는 웹 프레임워크를 만들지 웹 서버를 만들지는 않거든요)

(장고 공식문서 발췌)

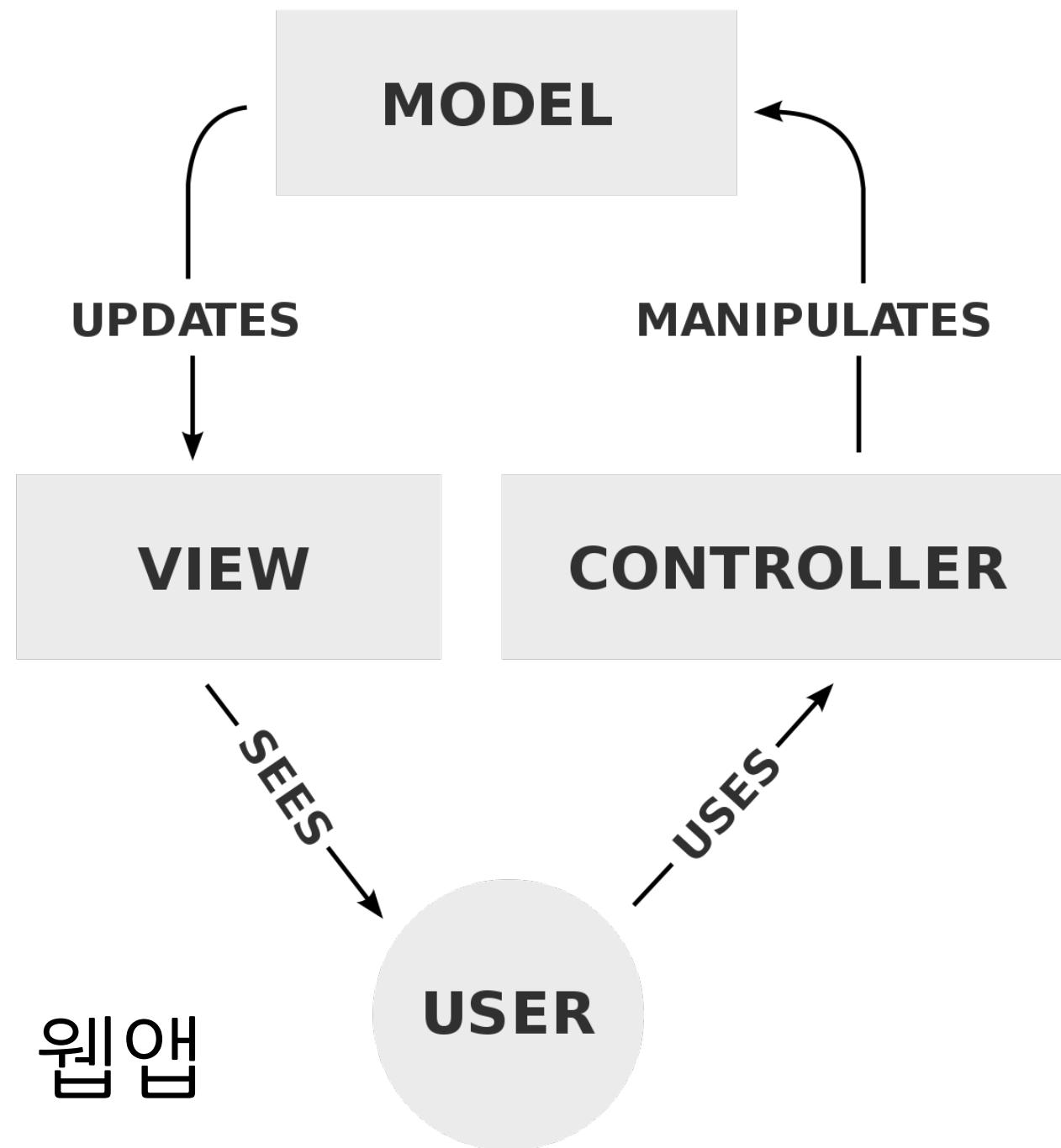


Ref) 장고는 웹 서버인가?

# What is Django?

: MVC Design Pattern

- 이렇게 디자인했더니 좋더라



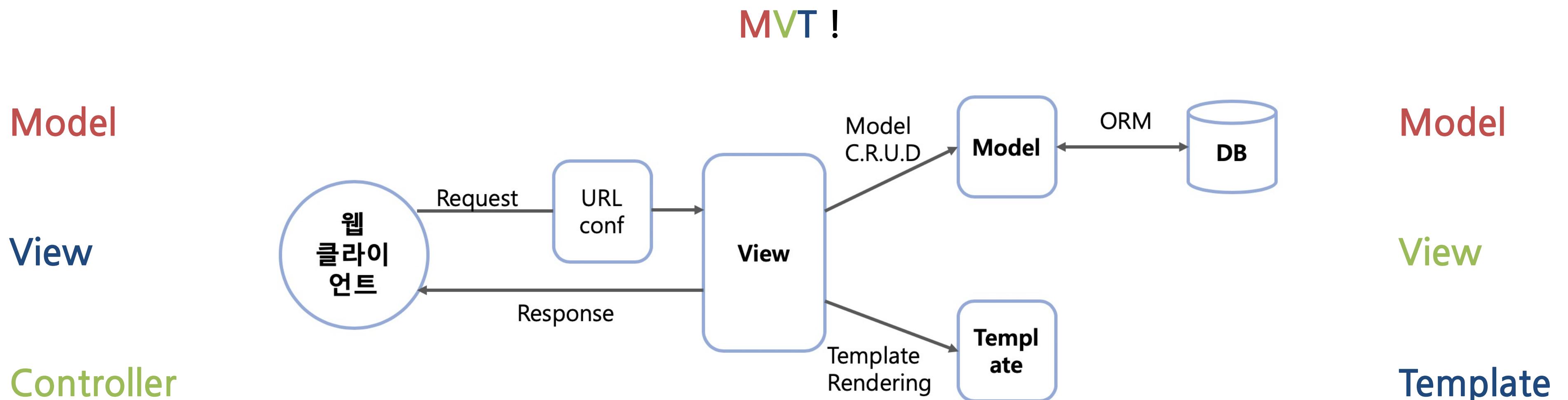
Model : holds data

View : shows data

Controller : manipulates data

# Django Design Pattern

Django는 전형적인 **MVC** 패턴; 근데 이름을 다르게 쓴다



# Django Design Pattern

각 레이어의 역할을 잘 알고,

역할에 충실하게 구현하자

MVT

---

\* Note \*

백엔드 프레임워크로서의 장고



# Django Model Layer

## ORM : 장고 기능의 핵심

```
9
10 class SurveyResult(models.Model):
11     EXPERIENCE_DEGREE =
12         (
13             (1, 'very_low'),
14             (2, 'low'),
15             (3, 'middle'),
16             (4, 'high'),
17             (5, 'very_high'),
18         )
19
20     os = models.ForeignKey(OperatingSystem, null=True, related_name='surveys', on_delete=models.SET_NULL)
21     python = models.PositiveSmallIntegerField(choices=EXPERIENCE_DEGREE)
22     rdb = models.PositiveSmallIntegerField(choices=EXPERIENCE_DEGREE)
23     programming = models.PositiveSmallIntegerField(choices=EXPERIENCE_DEGREE)
24     major = models.CharField(max_length=100)
25     grade = models.CharField(max_length=20)
26     backend_reason = models.CharField(max_length=500)
27     waffle_reason = models.CharField(max_length=500, blank=True)
28     say_something = models.CharField(max_length=500, blank=True)
29     timestamp = models.DateTimeField()
```

```
mysql> desc survey_surveyresult
      -> ;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id | int | NO | PRI | NULL | auto_increment |
| python | smallint unsigned | NO | | NULL | |
| rdb | smallint unsigned | NO | | NULL | |
| programming | smallint unsigned | NO | | NULL | |
| major | varchar(100) | NO | | NULL | |
| grade | varchar(20) | NO | | NULL | |
| backend_reason | varchar(500) | NO | | NULL | |
| waffle_reason | varchar(500) | NO | | NULL | |
| say_something | varchar(500) | NO | | NULL | |
| timestamp | datetime(6) | NO | | NULL | |
| os_id | int | YES | MUL | NULL | |
+-----+-----+-----+-----+-----+
11 rows in set (0.01 sec)
```

# Django

## Model Layer

### 1. Models

- > Class로 표현 , Instance object = row 하나
- > Meta 설정을 통해 다양한 부가 기능 구현

- DB 인덱싱 (index) (\* postgres 한정이긴 함)
- 영속성 관리 on/off (managed)
- 상속을 위한 추상화 (abstract)
- 등등..
- 아주 방대한 구현

### 2. Fields

- > Model 의 클래스 변수로 선언
- > 백문이 불여일견
- > Relationship 표현 ( 1:N, N:1, 1:1, N:M 모두 가능 )

# Django

## Model Layer

### ForeignKey (1:N, N:1)

항상! 많은 쪽에서 관계를 표현함

```
9
10 class SurveyResult(models.Model):
11     EXPERIENCE_DEGREE =
12         (1, 'very_low'),
13         (2, 'low'),
14         (3, 'middle'),
15         (4, 'high'),
16         (5, 'very_high'),
17     )
18
19     os = models.ForeignKey(OperatingSystem, null=True, related_name='surveys', on_delete=models.SET_NULL)
20     python = models.PositiveSmallIntegerField(choices=EXPERIENCE_DEGREE)
21     rdb = models.PositiveSmallIntegerField(choices=EXPERIENCE_DEGREE)
22     programming = models.PositiveSmallIntegerField(choices=EXPERIENCE_DEGREE)
23     major = models.CharField(max_length=100)
24     grade = models.CharField(max_length=20)
25     backend_reason = models.CharField(max_length=500)
26     waffle_reason = models.CharField(max_length=500, blank=True)
27     say_something = models.CharField(max_length=500, blank=True)
28     timestamp = models.DateTimeField()
29
```

```
mysql> desc survey_surveyresult
      -> ;
+-----+-----+-----+-----+-----+
| Field          | Type           | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+
| id             | int            | NO   | PRI | NULL    | auto_increment |
| python          | smallint unsigned | NO   |     | NULL    |                |
| rdb             | smallint unsigned | NO   |     | NULL    |                |
| programming    | smallint unsigned | NO   |     | NULL    |                |
| major           | varchar(100)   | NO   |     | NULL    |                |
| grade           | varchar(20)    | NO   |     | NULL    |                |
| backend_reason | varchar(500)   | NO   |     | NULL    |                |
| waffle_reason   | varchar(500)   | NO   |     | NULL    |                |
| say_something   | varchar(500)   | NO   |     | NULL    |                |
| timestamp       | datetime(6)    | NO   |     | NULL    |                |
| os_id           | int            | YES  | MUL | NULL    |                |
+-----+-----+-----+-----+-----+
11 rows in set (0.01 sec)
```

반대쪽 참조는 🤔 ?

\* related\_name \*  
( default: foo\_set )

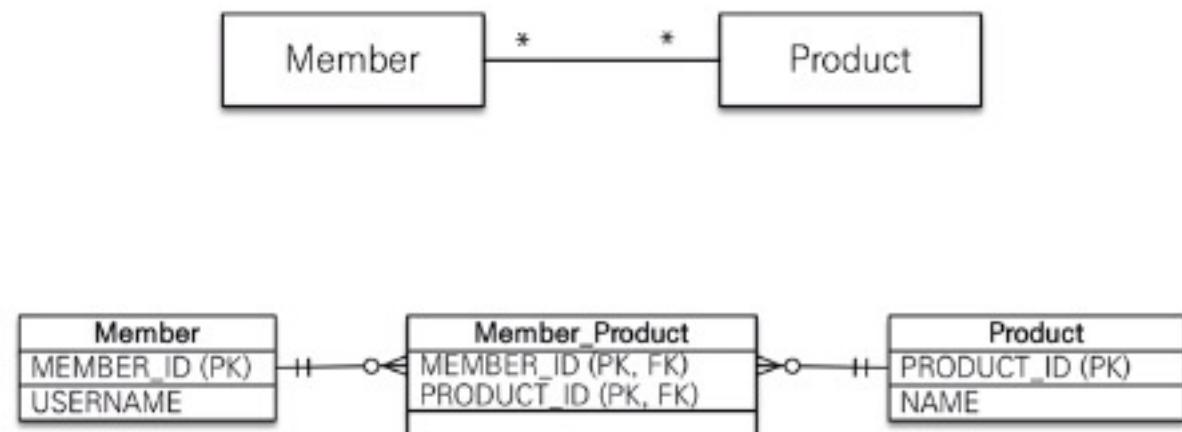
# Django

## Model Layer

### ManyToManyField (N:M)

마찬가지로, 한 쪽에서만 참조

through model 설정 가능  
*Model.m2mfield.through.objects.all()*



반대쪽 참조도 똑같다

\* related\_name \*  
( default: foo\_set )

# Django

## Model Layer

### OneToOne (1:1)

\* 클래스의 소문자 이름 \*

(os.surveyresult)

Or

\* related\_name \*

#### How are the backward relationships possible?

Other object-relational mappers require you to define relationships on both sides. The Django developers believe this is a violation of the DRY (Don't Repeat Yourself) principle, so Django only requires you to define the relationship on one end.

But how is this possible, given that a model class doesn't know which other model classes are related to it until those other model classes are loaded?

The answer lies in the [app registry](#). When Django starts, it imports each application listed in [INSTALLED\\_APPS](#), and then the models module inside each application. Whenever a new model class is created, Django adds backward-relationships to any related models. If the related models haven't been imported yet, Django keeps tracks of the relationships and adds them when the related models eventually are imported.

For this reason, it's particularly important that all the models you're using be defined in applications listed in [INSTALLED\\_APPS](#). Otherwise, backwards relations may not work properly.

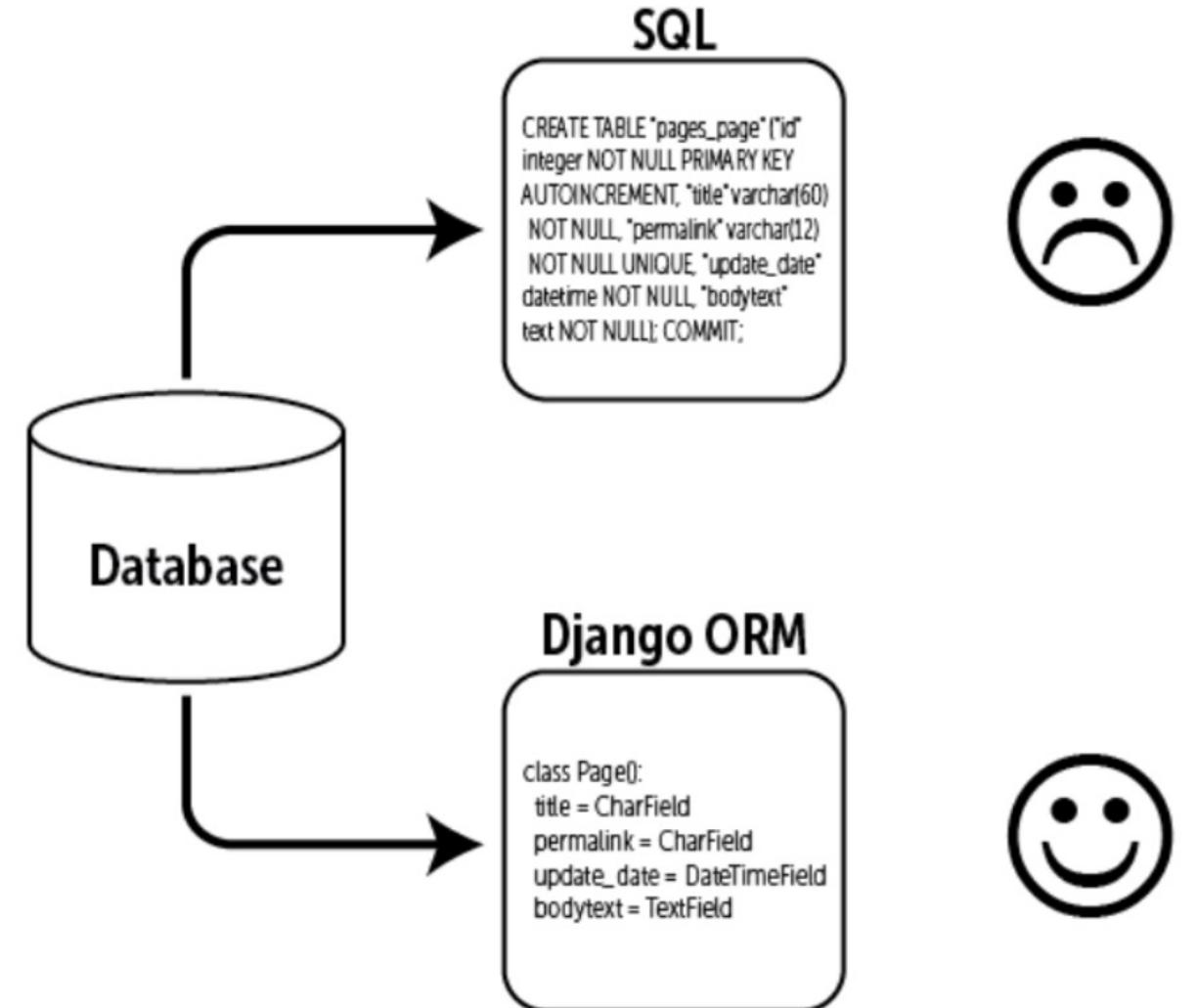
-> 한 줄 요약 : 장고는 INSTALLED\_APPS 미리 훑어봄으로써, 역 참조를 알아서 설정

# Django

## Model Layer

### 3. Manager

- > Model 과 붙어다니는 친구
- > DB에 어떻게 쿼리를 날릴지 결정
- > Model.objects ... , ORM의 중추
- > get\_queryset() override
- > 커스텀 메소드 설정



# Django

## Model Layer

### 4. QuerySet

-> 번역하면, 요청 모음

If. 식당에 10명이 와서, 웨이터에게 젓가락 좀 달라고 하는 상황

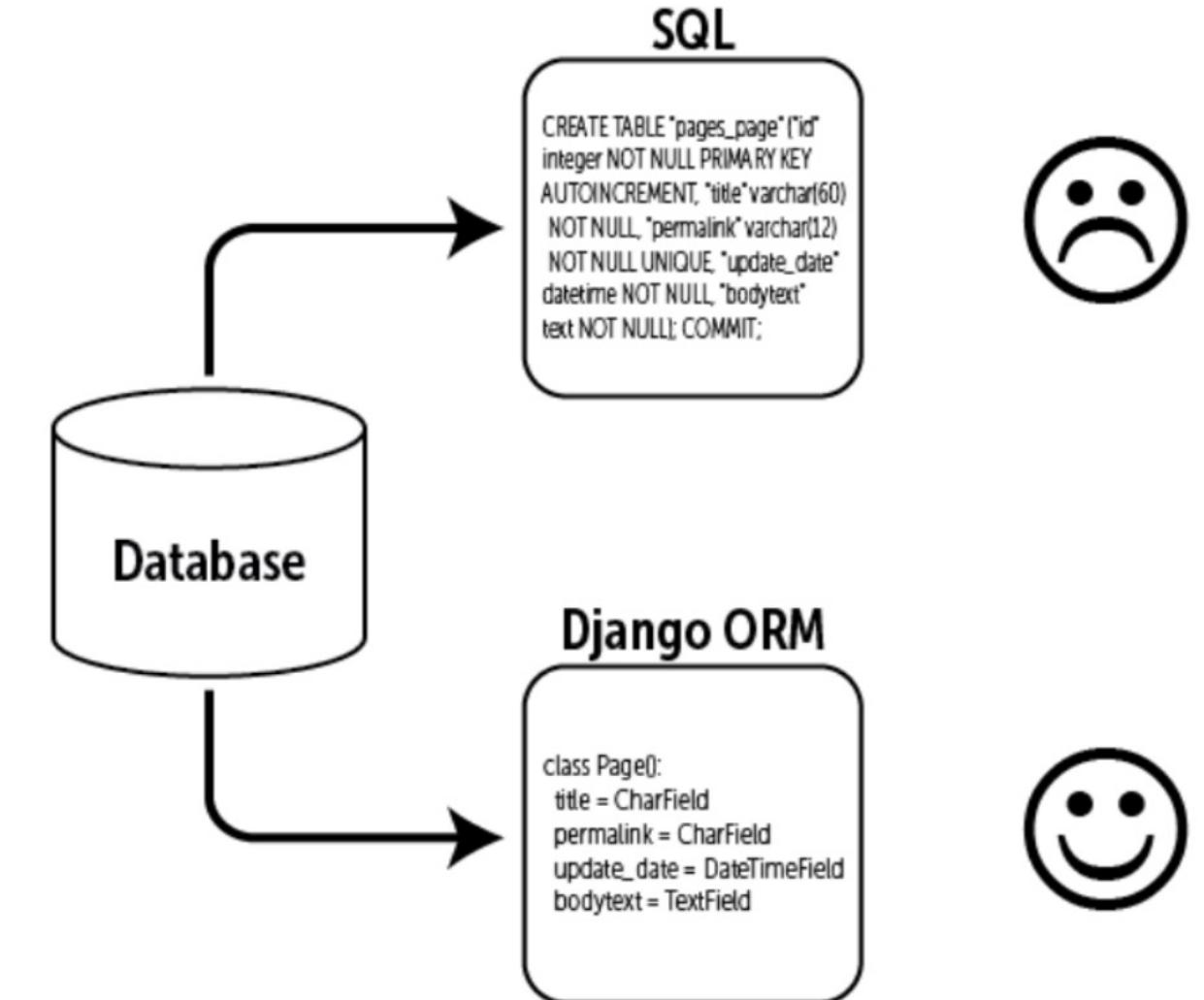
-> 한명이 한 짹씩 갖다 달라 하면 -> 20번의 요청이 필요함

-> 😱 10명이 아니라 1000만명이라고 하면..?

-> 질의는 가능한 **효율적으로** 가져와야 함 ( 꼭 한번에 X )

-> 이를 위해, 다양한 상황에서의 DB 쿼리를 가능케 하는 쿼리셋 클래스 존재

-> 앞서 본 **Manager**는 작업 수행의 결과로 **QuerySet** 반환

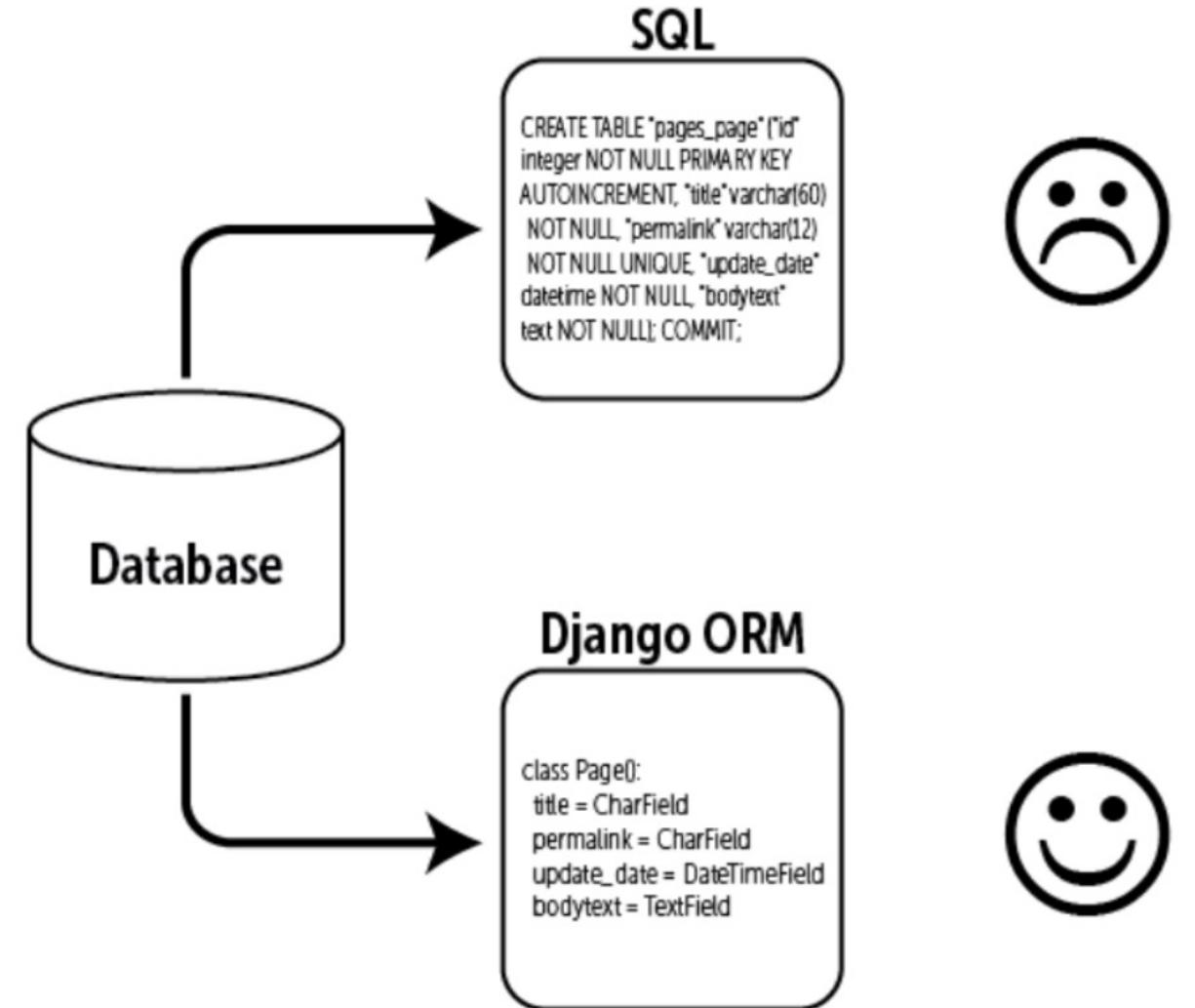


# Django

## Model Layer

### Manager

- > all, filter, exclude
- > create, update, delete, bulk\_
- > annotate, alias ..
- > prefetch\_related, select\_related ( DB JOIN 개념 )
- > 공식 문서, 구현 확인해보기



# Django

## Model Layer

### QuerySet의 중요한 특징

#### 1. QuerySets are Lazy !!

-> 진짜 필요할 때 아니면 디비까지 안 감!

#### 2. QuerySet은 캐싱이 된다

-> 결과를 잘 기억하고 있음

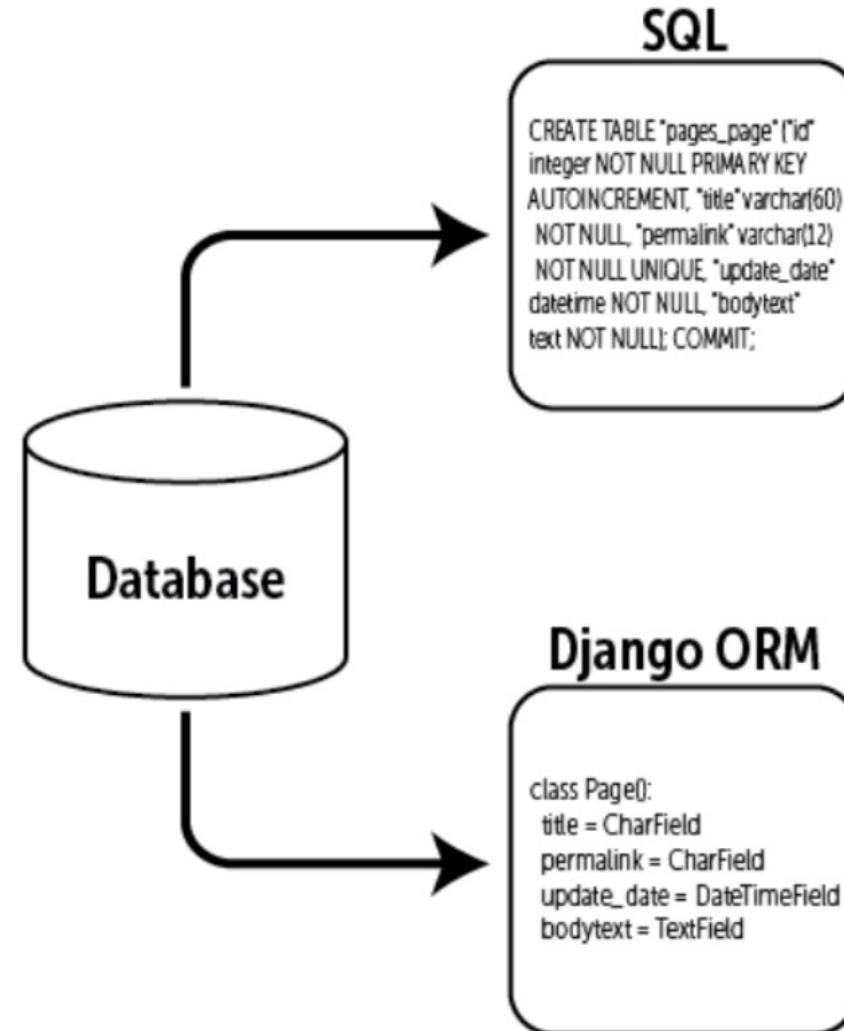
```
class OS(models.Model):
    os_brand = models.CharField(max_length=20)

# 1.
queryset = OS.objects.all()

# 2.
queryset = OS.objects.all()
if queryset:
    print(queryset)

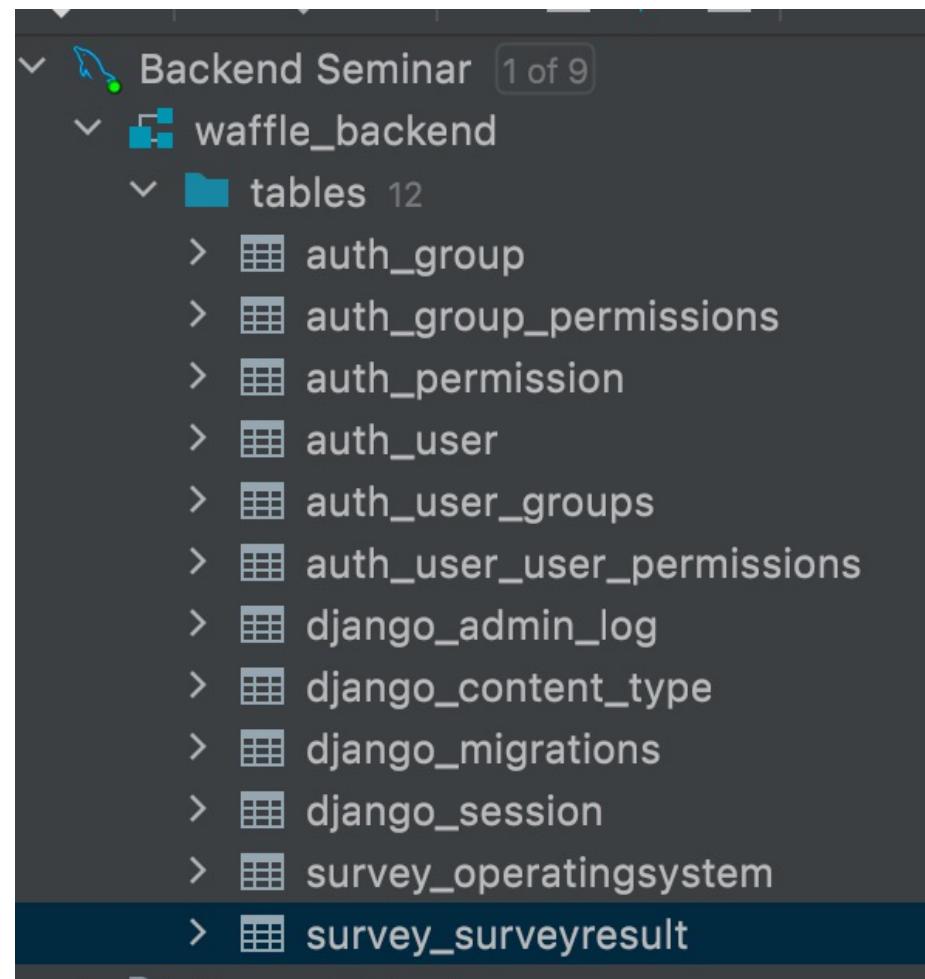
# 3.
queryset = OS.objects.all()
if queryset:
    print(queryset)

if queryset:
    print(queryset)
```



# Django

## Built-in Models



우리가 만든 적도 없는 것들이 보인다

localhost:8000/admin

?!

장고는 어드민 페이지를 기본적으로 제공

-> 유저 인증 / 권한 관리를 쉽게 할 수 있다.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('survey.urls')),
    path('api/v1/', include('user.urls'))
]
```

# Django

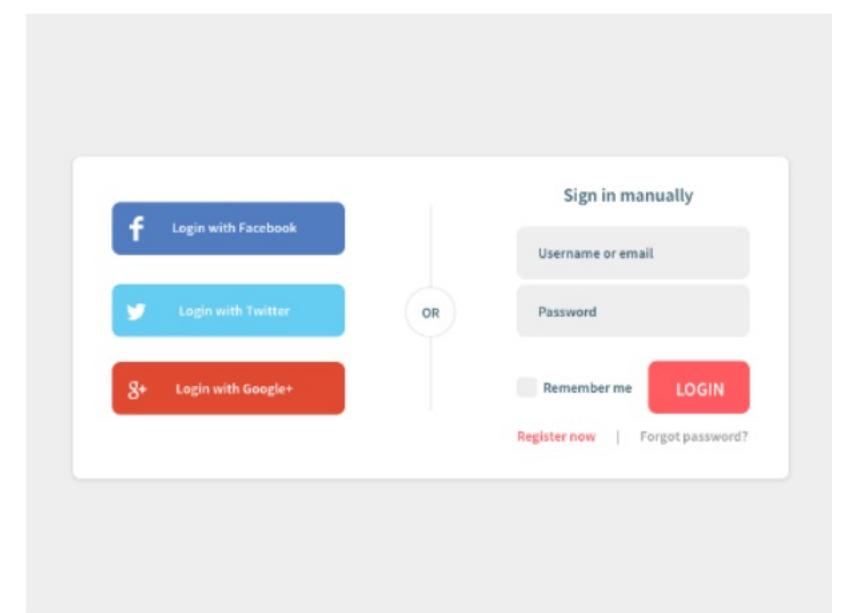
## Built-in Models

### User

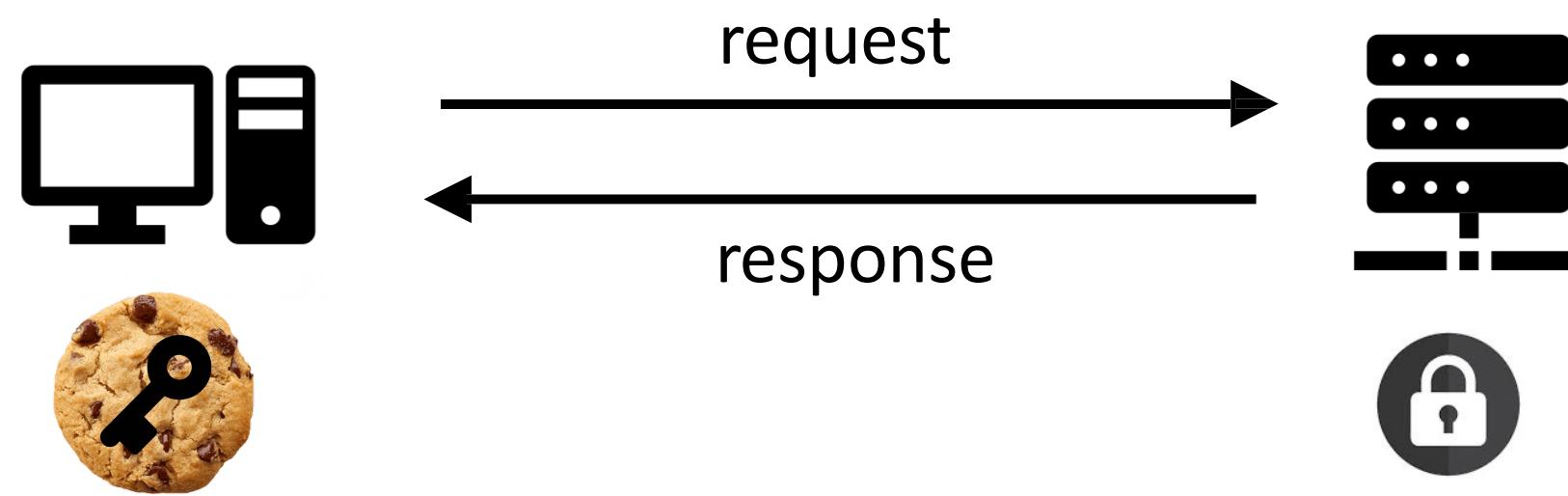
거의 모든 서비스의 기본



- Django는 기본으로 User에 대한 것들도 쉽게 여러가지를 제공해줍니다
- 이미 여러분은 User에 mapping되는 table을 로컬 환경에 갖고 있습니다
- 회원가입(sign-up), 로그인(sign-in)을 기본으로한 동작 이해 중요
- 최근엔 password, email 방식뿐 아닌 social login이 많이 이용됨



# Cookie & Session [remind]



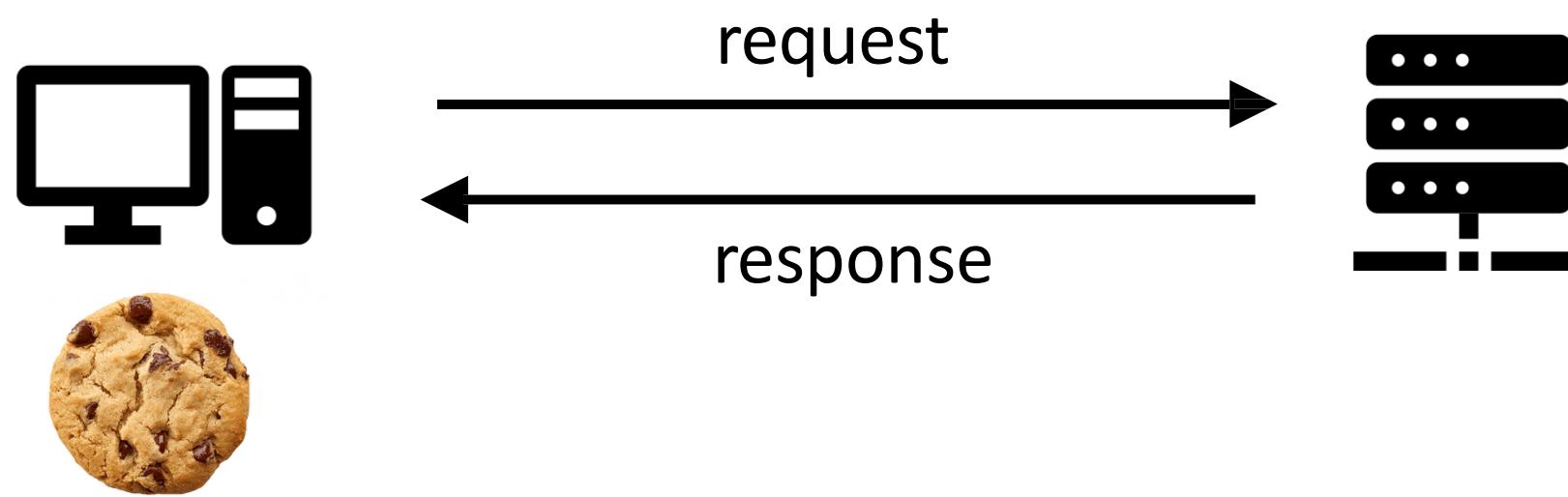
## 세션(Session)

데이터를 사용자의 브라우저에 쿠키 형태가 아닌  
접속한 서버 DB에 정보를 저장

클라이언트는 Session id를 쿠키로 메모리에 저장

Session id를 client의 메모리에 저장하므로  
브라우저가 종료되면 사라지게 됨

# Cookie & Session [remind]



## 쿠키(Cookie)

서버에서 전달해 사용자의 로컬 컴퓨터에 저장하는 작은 데이터

쿠키는 같은 웹사이트를 방문할 때마다 request header에 추가되어 전달

쿠키를 통해 서버에게 client 정보와 상태를 전달 가능

# Django

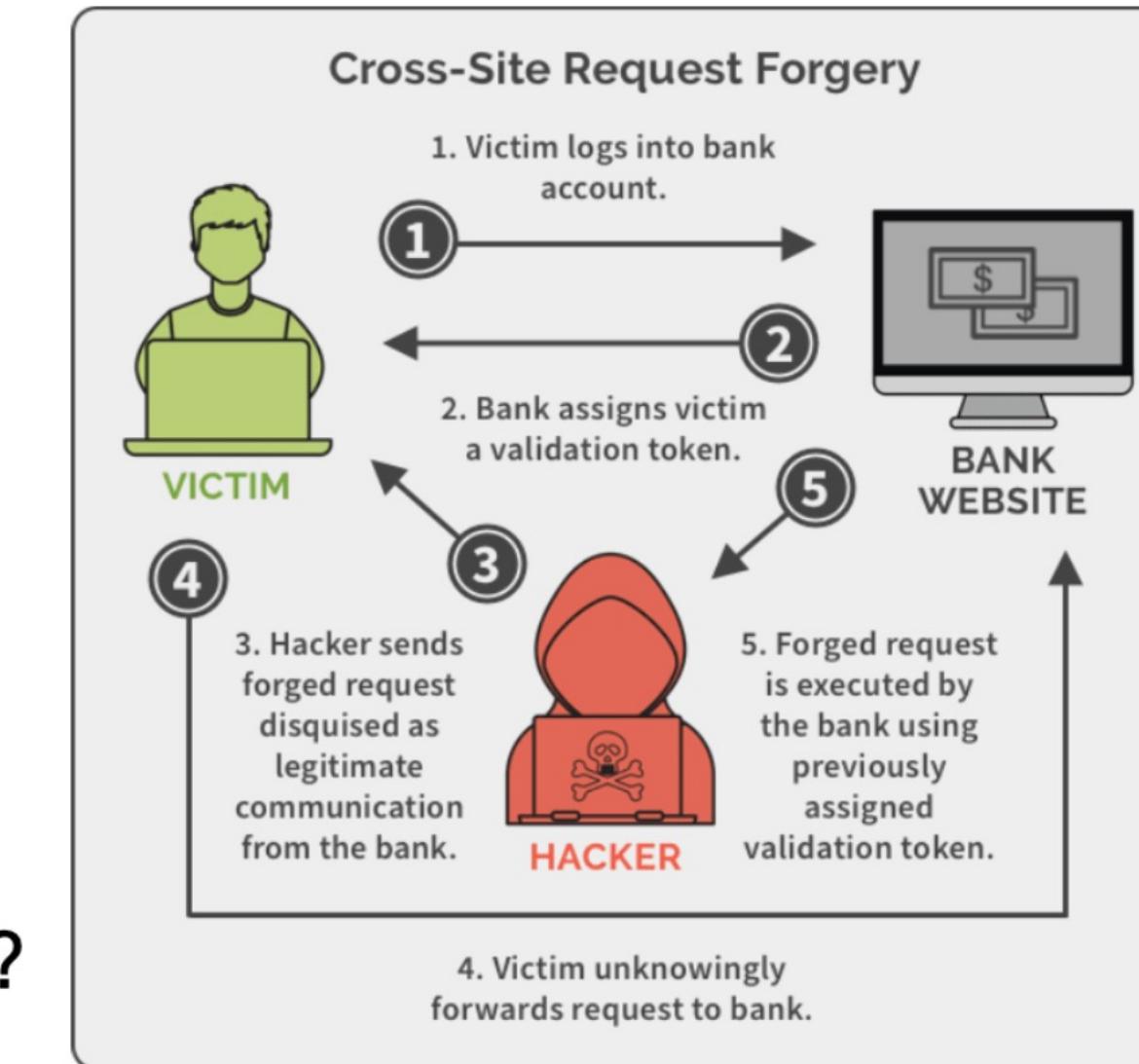
## Authentication

<https://docs.djangoproject.com/en/3.1/ref/csrf/>

## CSRF 공격

### Cross-Site Request Forgery

- 로그인만으로 보안이 충분한가? Never!
- 유저가 의도하지 않은 처리를 실행시키는 공격 수법
- 어떻게 유저가 의도하지 않은 request를 걸러낼 것인가?
- Django의 CsrfViewMiddleWare
- 임의의 token을 발행해 이를 모든 request에 대해 매번 체크
- GET 등의 요청은 예외

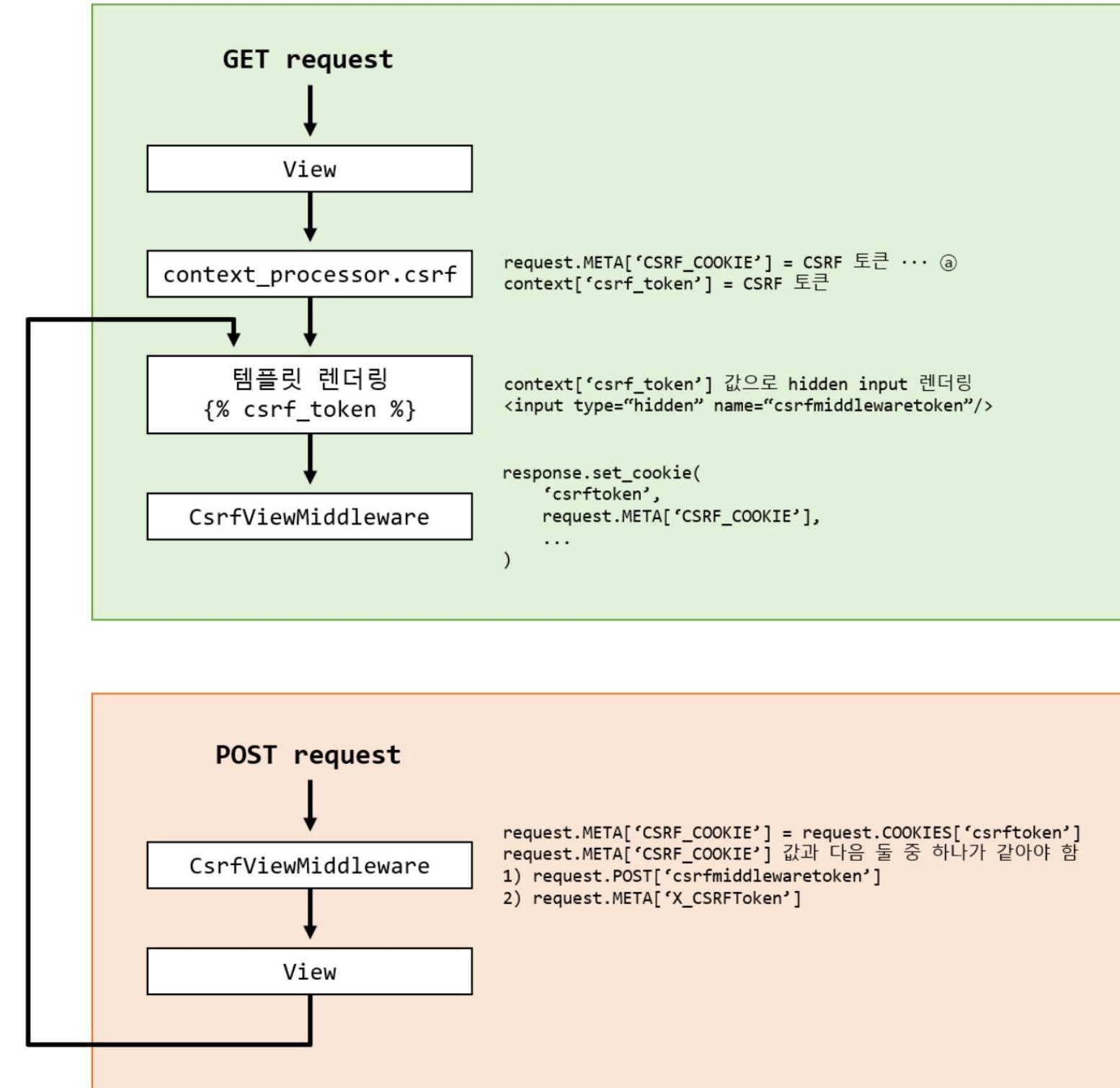
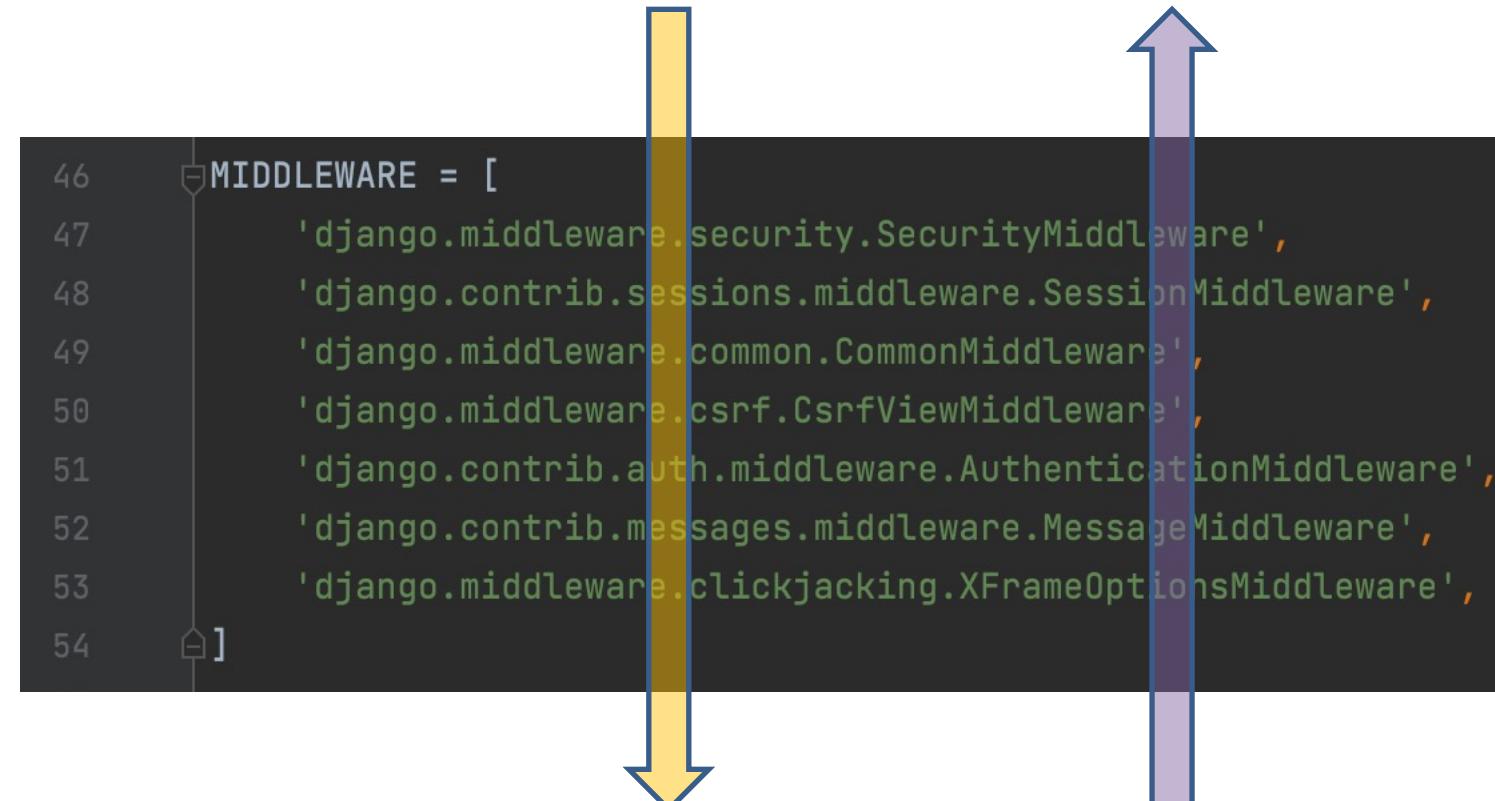


# Django Authentication

장고는 이를 막기 위해

CsrfViewMiddleware를 사용

MiddleWare ?



# Django

## Authentication

### 외부의 요청 잘 처리하기

#### 이상한 요청은 거르자

- 이런 경우는 어떡하지? 저런 경우는 어떡하지? 하는 생각하기
- 특히 POST, PUT 등의 경우 외부에서 받는 데이터 잘 체크하기
- request의 data
  - header
  - body
  - query string (parameter)

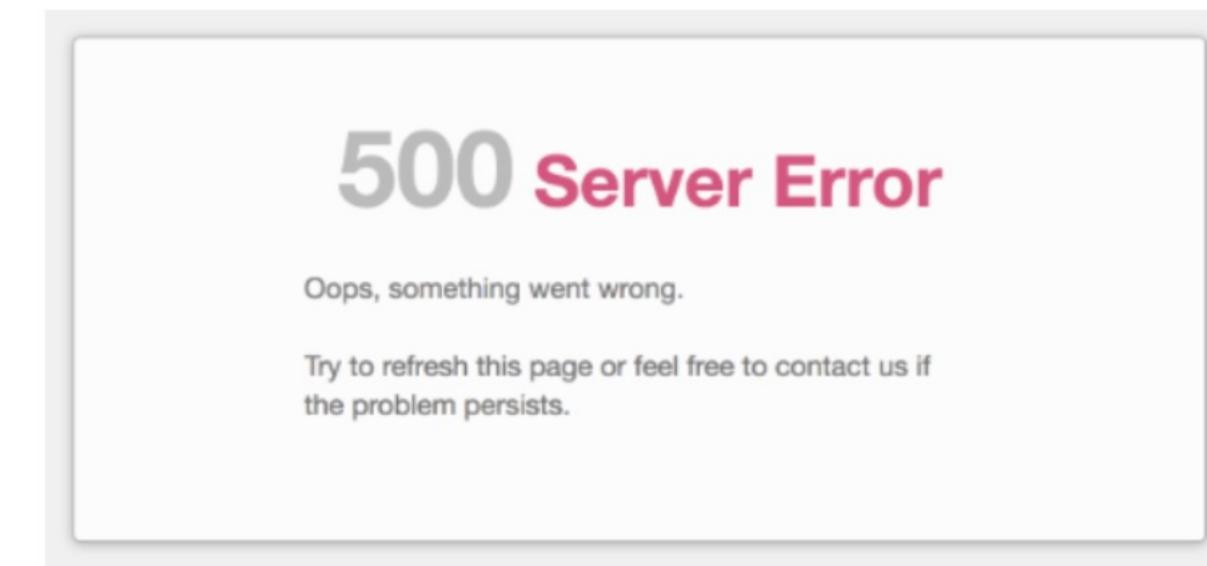
# Django

## 잠깐 땐소리

## 500을 피하자

### INTERNAL SERVER ERROR

- logic상 status code 500이 발생할 수 있는 여지는 최대한 없애기
- 모든 프로그래밍이 그렇지만, 서버 프로그래밍은 염려와 강박을 탑재할 것
- 안정적인 서버 자체가 서비스의 가치
- 유저가 서버의 존재를 느낄 일이 없어야



- 문제될 상황에 대해선 합리적인 처리가 중요
  - 예상되는 원인을 코드상에 드러나게 짚어서 대응
  - 문제의 책임과 원인을 찾고 외부에 알리는 프로그래밍

# Django

## 사실 요즘은

## Cloud Computing 서비스

### Amazon Web Services revisited

- 왜 돈 주고 남의 컴퓨터를 쓸까?
  - 내 컴의 성능은 어느 정도?
  - 내 컴이 꺼지면?
  - 모니터링 부탁할 수 있을까?
  - 관리를 부탁할 수 있을까? (재부팅하기, 갑자기 늘리기, 줄이기)



**kubernetes** Google Cloud Platform



실제 동작 확인하기 AWS EC2, RDS

WA STUDIO

The Waffle logo consists of the word "Waffle" in a black sans-serif font, where the letter "A" has a brown square pattern over it.

# Q&A

