시험 끝!

코딩 시작~

CRUD?

프레임워크?

데이터베이스?

ORM?

Intro

목표: 장고에 대한 기억 되살리기 & 숙달하기

세션 구성

1. 장고 관련 이론적 설명
2. 문제풀기(실습)

다시 만난 장고

중간고사 이후 장고 복습

9번째 세션

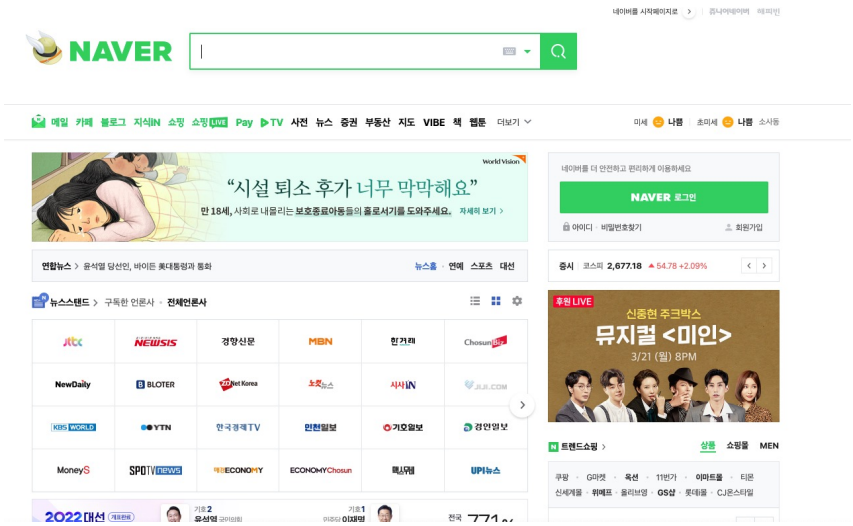
NEXT X LIKELION 박한영

웹 애플리케이션 프레임워크(풀스택)

Django 소개

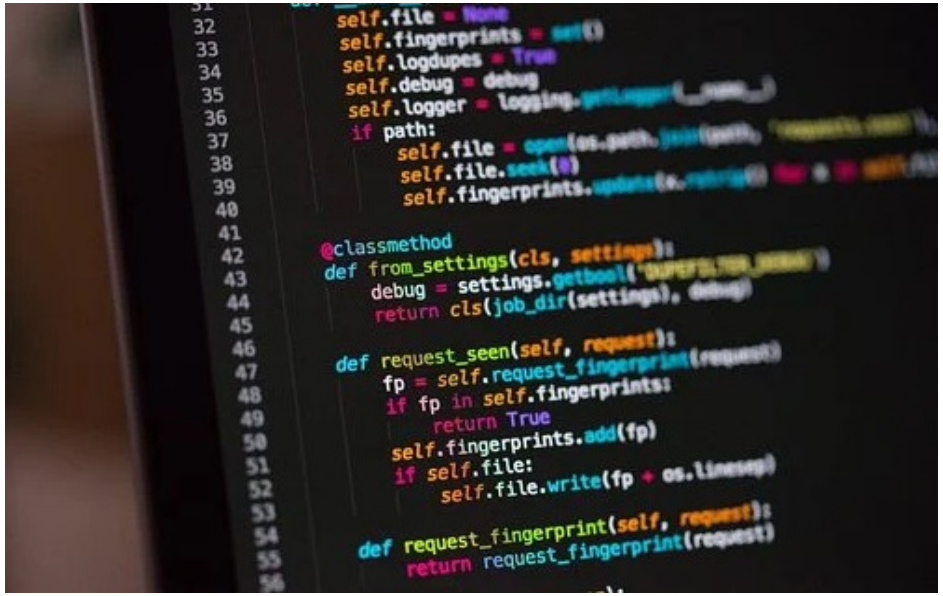
장고에 대한 정의

웹 애플리케이션 프레임워크(풀스택)



웹

+

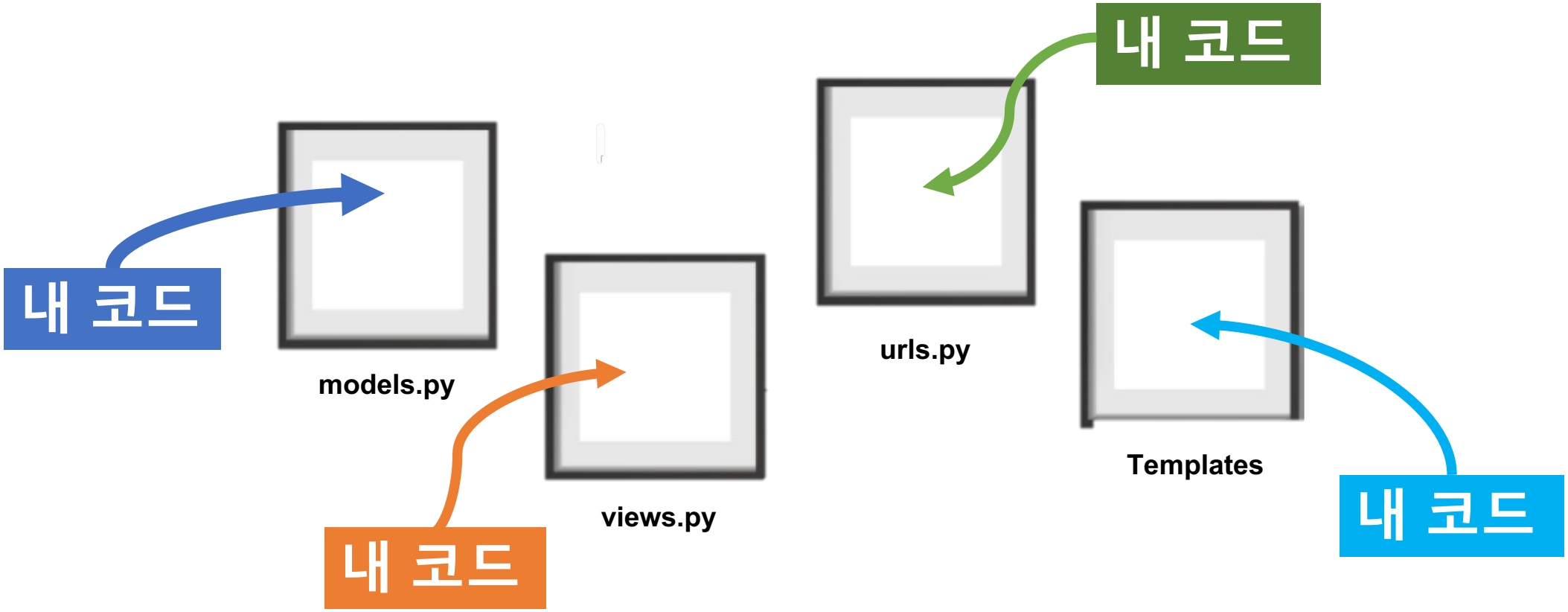


애플리케이션

Django 소개

장고에 대한 정의

웹 애플리케이션 **프레임워크**(풀스택)



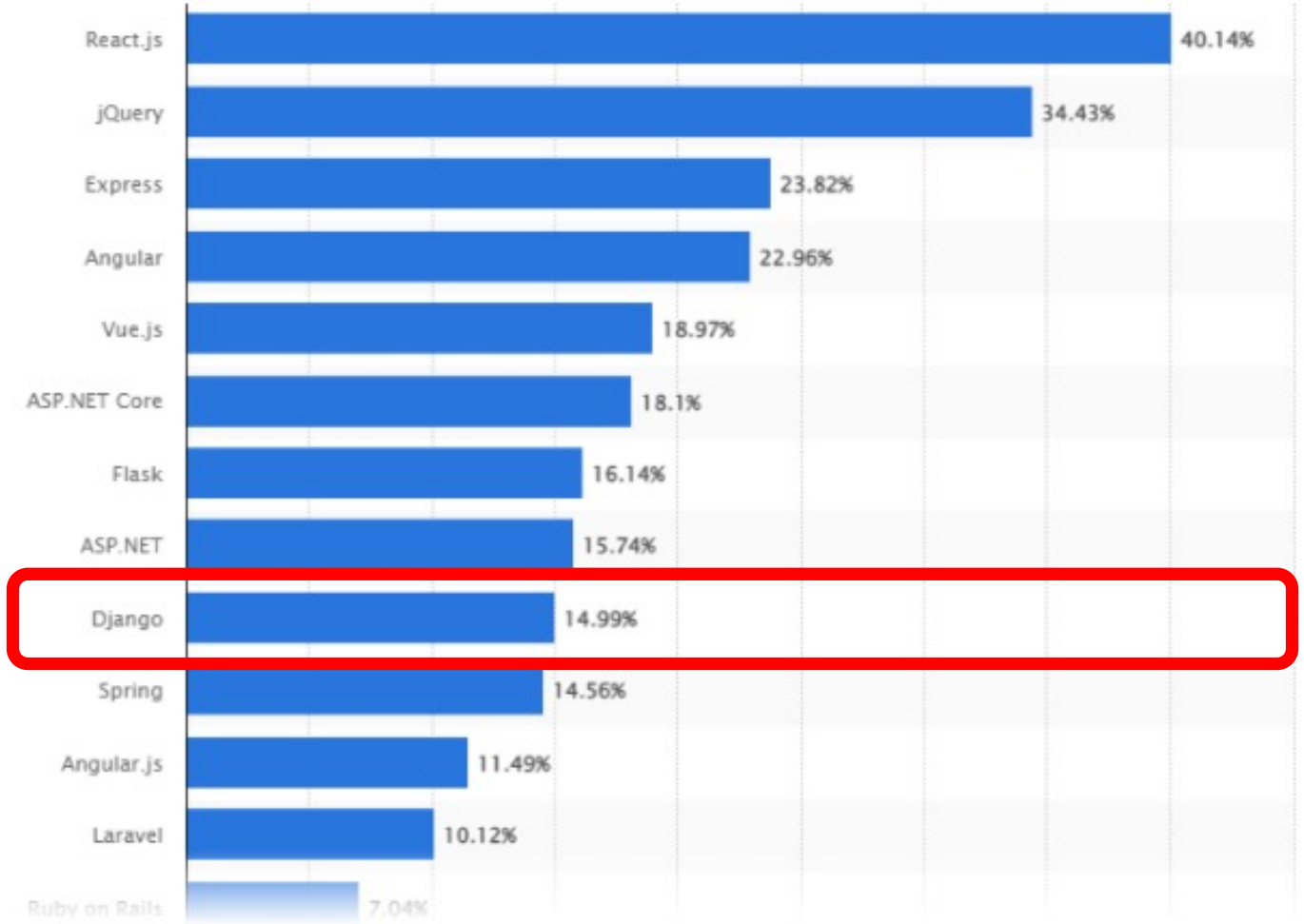
웹 애플리케이션 프레임워크(**풀스택**)

클라이언트
(프론트엔드)

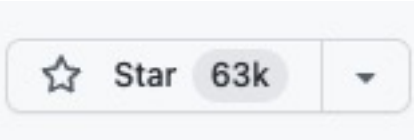
서버
(백엔드)

Django 소개

장고의 위상



9위



2021 전 세계 개발자 사이 가장 인기 있는 웹 프레임워크

Django 소개

멋사는 왜 장고를 택했나?

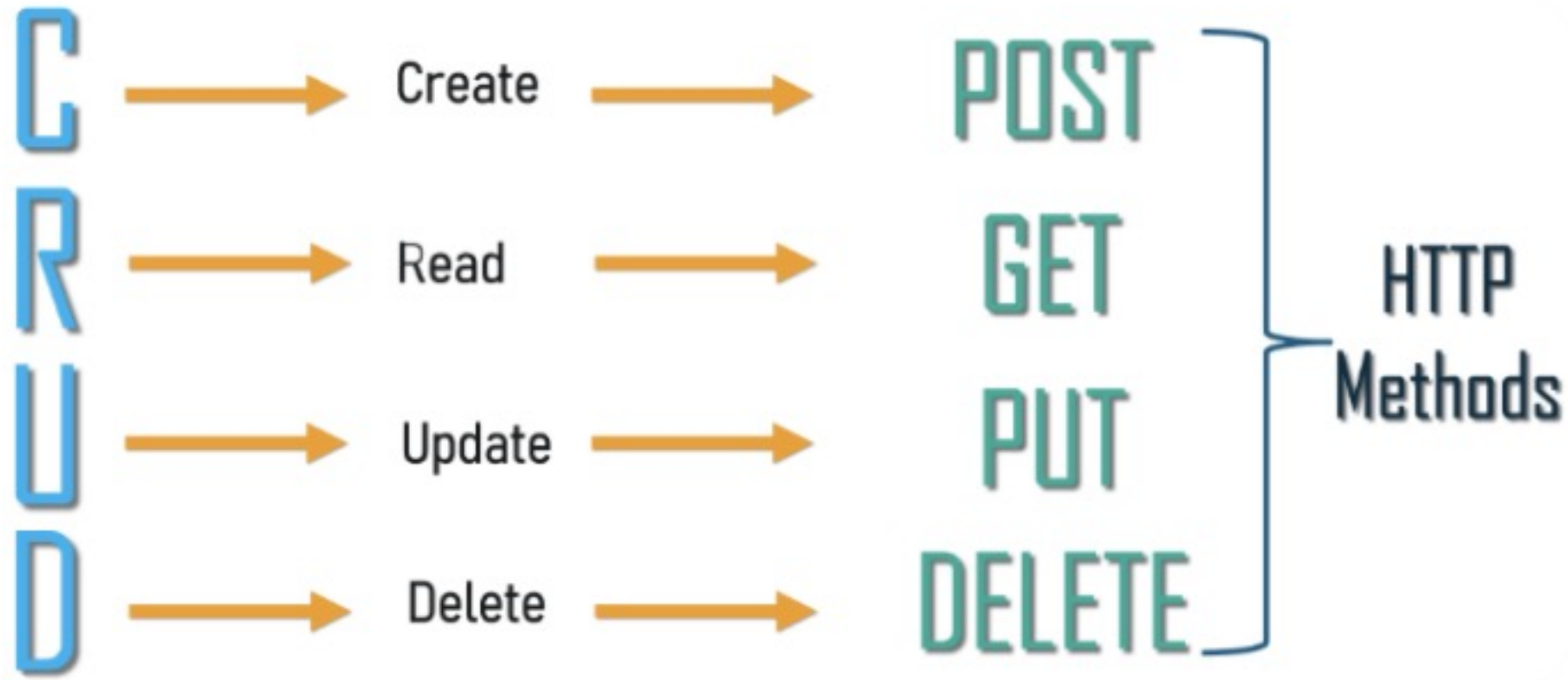
“마감 기한을 앞둔 완벽주의자들을 위한 웹 프레임워크”

>> 스타트업에 적합

데이터 처리의 기본: CRUD

CRUD

CRUD란?


C
Create
POST
R
Read
GET
U
Update
PUT
D
Delete
DELETE
HTTP
Methods

CRUD

CRUD의 대상은
데이터

데이터:

컴퓨터가 처리할 수 있는 문자, 숫자, 소리, 그림 따위의 형태로 된 자료.

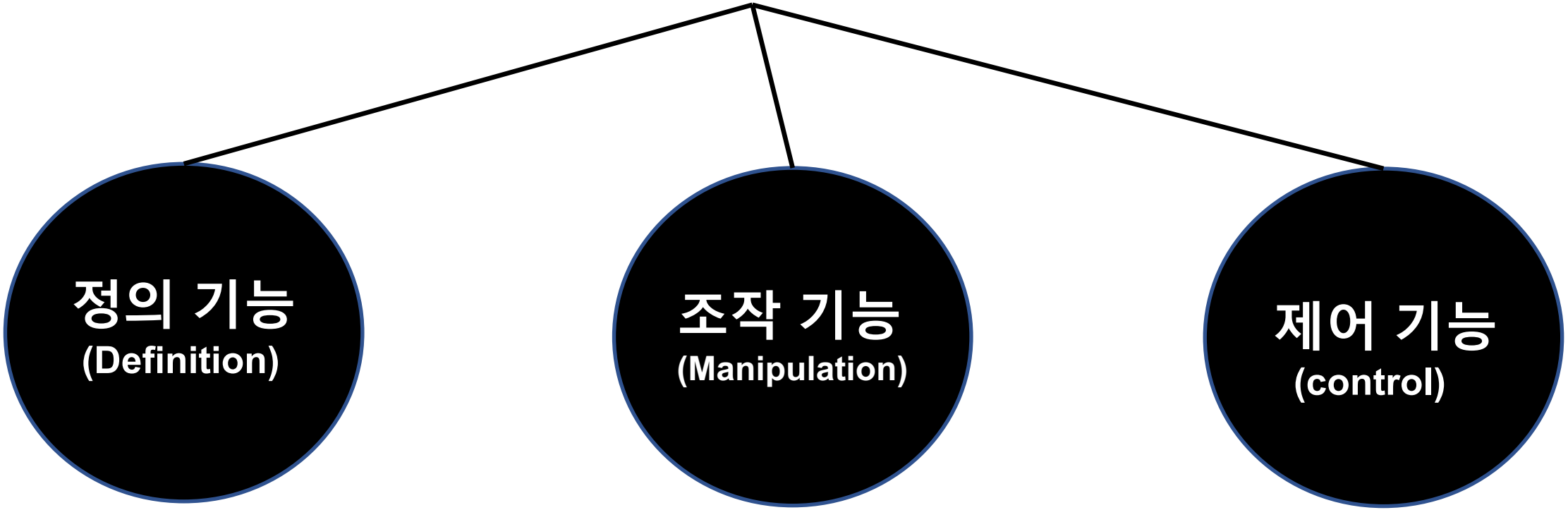
데이터베이스(DB):

여러 사람이 공유하여 사용할 목적으로 체계화해 통합, 관리하는 데이터의 집합이다.
(데이터들의 모여있는 것)

CRUD

DBMS?

DBMS: 데이터를 다룰 수 있게 해주는 소프트웨어



데이터의 구조를 정의

데이터 CRUD

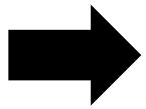
데이터 접근 권한 관리

CRUD

SQL?

Structured **Q**uery **L**anguage

:데이터베이스 관리 시스템에서 자료를 처리하는 용도로 사용되는 구조적 데이터 질의 언어



ORM(Object Relational Mapping):
객체와 데이터를 연결시켜줌

무슨말이냐:

본래, SQL으로 다루어야 하는 데이터를

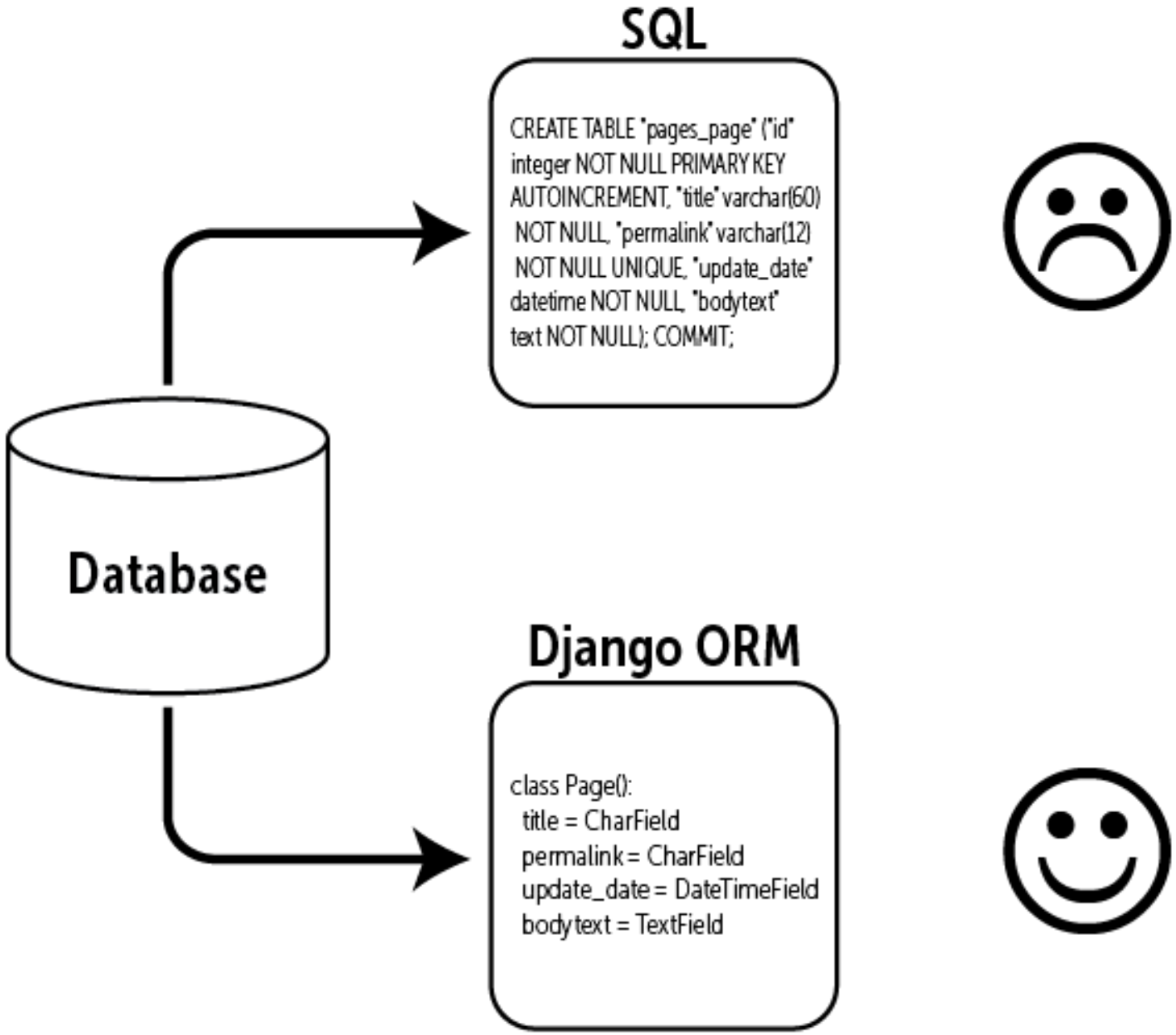
파이썬 객체처럼 다룰 수 있게 해준다!

ORM 덕분에 파이썬만 알아도 데이터를 다룰 수 있게 됨

ORM아 고마워..!

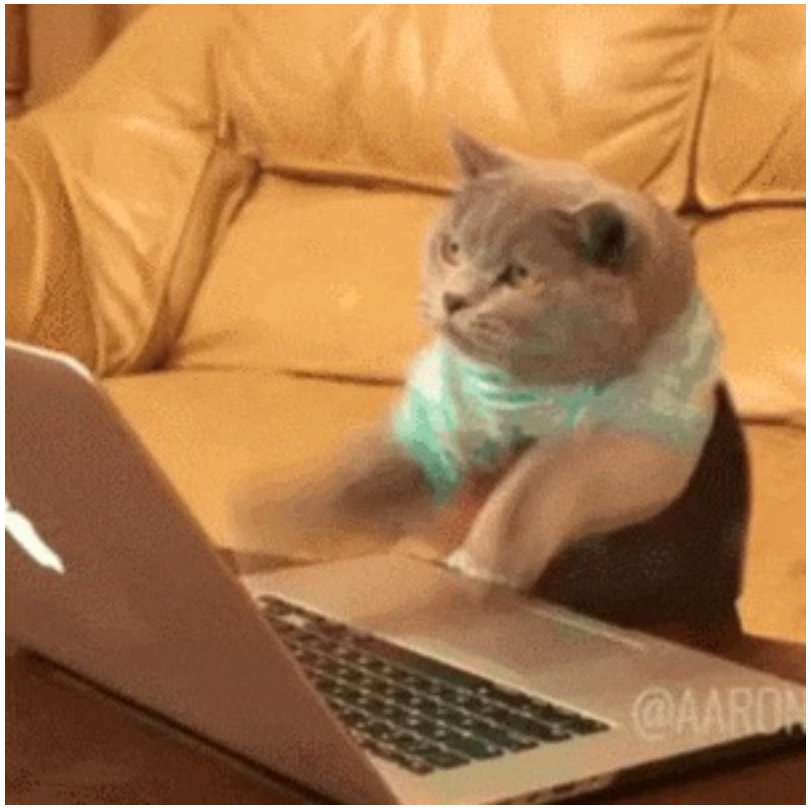
CRUD

ORM?


SQL
CREATE TABLE "pages_page" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "title" varchar(60) NOT NULL, "permalink" varchar(12) NOT NULL UNIQUE, "update_date" datetime NOT NULL, "bodytext" text NOT NULL); COMMIT;
Database
Django ORM
class Page():
title = CharField
permalink = CharField
update_date = DateTimeField
bodytext = TextField

CRUD

실습



이제부터 실습

CRUD

진부하지만
게시판 만들기

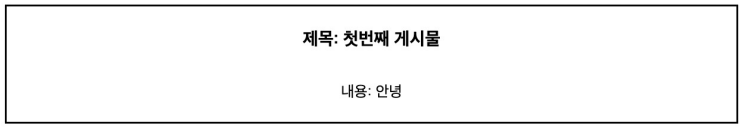
게시글 리스트 페이지

게시글 작성하기

게시글 목록(List)

게시글 상세 페이지

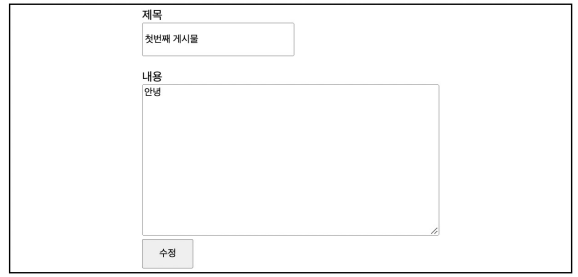
뒤로가기 수정하기 삭제하기



게시글 상세(Detail)

게시글 수정 페이지

뒤로가기



게시글 수정(edit)

CRUD

프로젝트 세팅

1) mkdir django-review

2) cd django-review

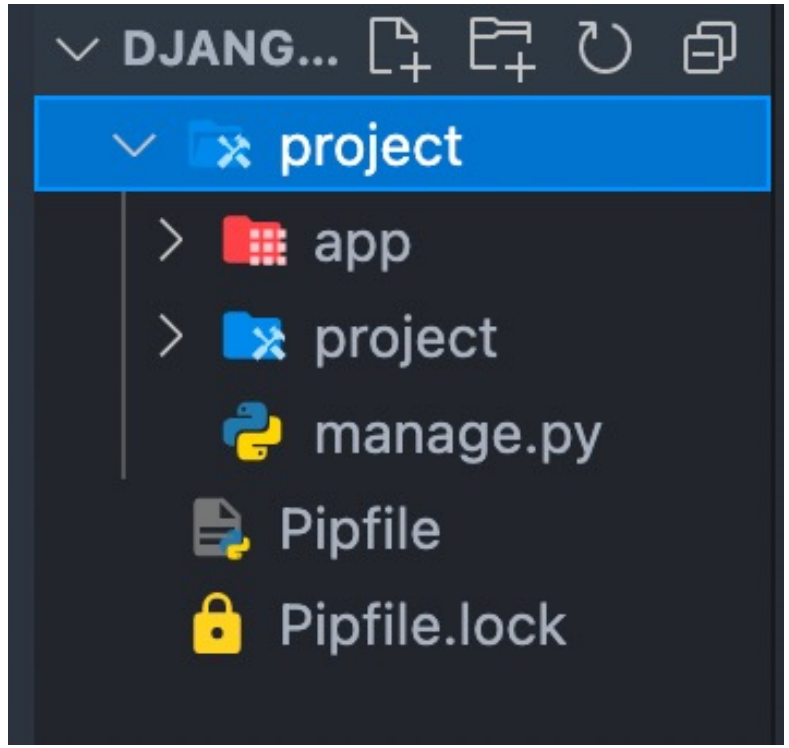
3) pipenv shell

4) pipenv install django

5) django-admin startproject project

6) cd project

7) python manage.py startapp app
(혹은 python3 manage.py startapp app)



CRUD

프로젝트 세팅 해석

1) ‘django-review’라는 폴더 생성

2) ‘django-review’라는 폴더에 들어감

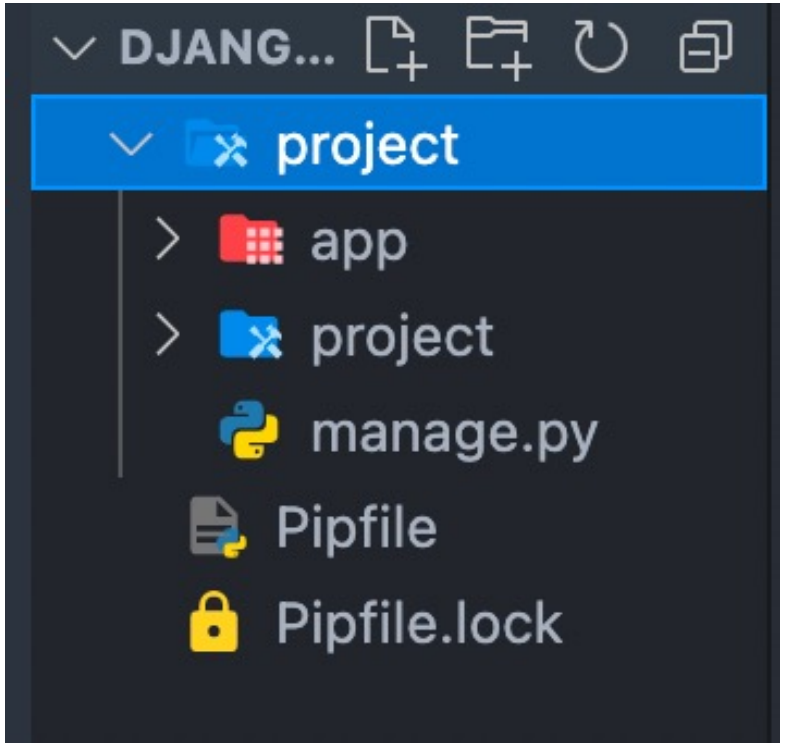
3) django-review 폴더에서 가상환경 활성화

4) 현재 가상환경에 django 설치

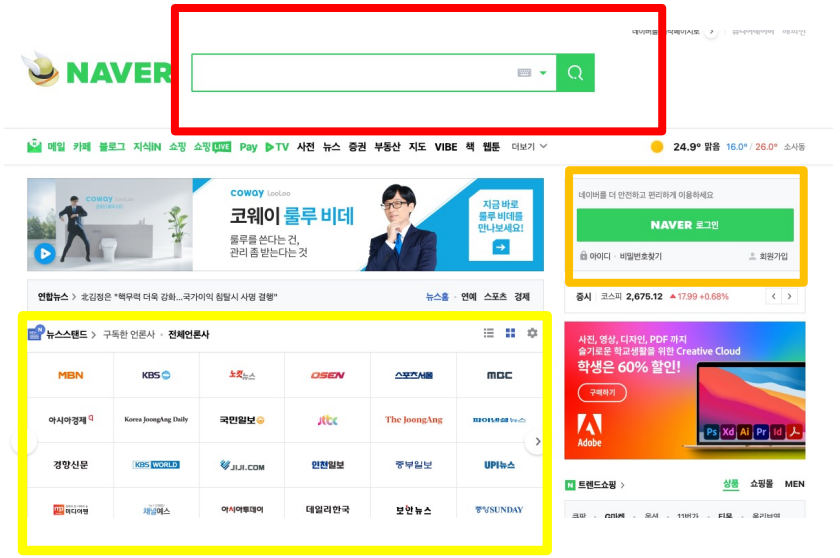
5) ‘project’라는 이름의 프로젝트를 생성
(프로젝트를 위한 가장 기본적인 세팅을 함)

6) ‘project’이라는 폴더에 들어감

7) ’app ' 이라는 앱을 생성



공부거리: 장고 project와 app
https://docs.djangoproject.com/ko/4.0/intro/tutorial01/



장고에서의 ‘project’와 ‘app’

project: 하나의 웹사이트 전체

app: 웹사이트의 기능 단위(유저 기능, 게시글 기능 ...)

*여러개의 앱(기능)이 모여 하나의 프로젝트(웹사이트)가 된다!
*하지만 처음부터 여러 개의 앱으로 나누어 프로젝트를 구성하는 것은
혼동을 유발할 수 있기에, 이번 실습에선 하나의 앱만 만들 예정~

CRUD

프로젝트 세팅

공부거리: 장고 project와 app
https://docs.djangoproject.com/ko/4.0/intro/tutorial01/

앱 생성후에는 오른쪽 사진과 같이
settings.py의 INSTALLED_APPS에 추가해주기!

```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app',  # 본인이 생성한 앱 이름
]
```

project/settings.py

CRUD

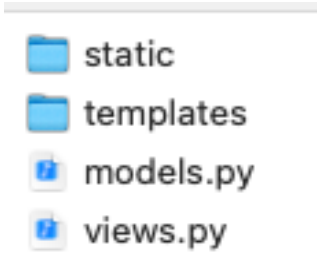
데이터 세팅

python manage.py makemigrations	models.py의 내용에 따라 DB 상의 변화를 migrations 파일을 생성
python manage.py migrate	생성된 migrations(migrate 명령어를 통해 생성됨) 파일들을 DB에 반영한다. -> 기본적으로 필요한 데이터 구조를 생성하기 위해 프로젝트 최초 생성 시 실행

CRUD

프로젝트 세팅

공부거리: 장고 project와 app
https://docs.djangoproject.com/ko/4.0/intro/tutorial01/



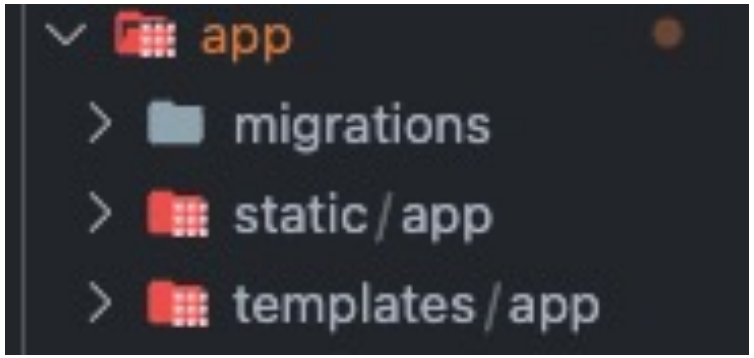
1) ‘app폴더에넣을것들’의 내용물들을 복사해서 app 폴더에 넣기

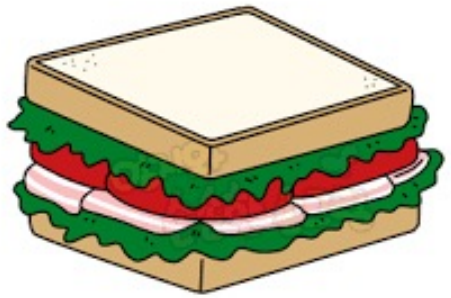
2) project 폴더의 urls.py를 이 파일로 대체

***기존 파일들을 덮어쓰기(대치)하시면 됩니다!**

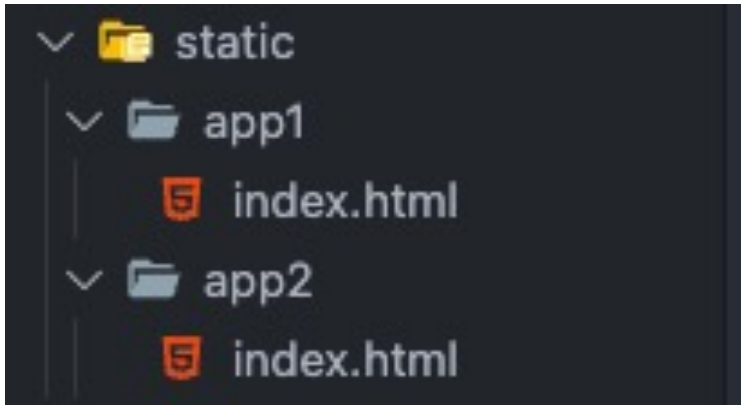
CRUD

샌드위치 구조





샌드위치 구조:
app 디렉토리 안에 static 디렉토리 안에 다시 app 디렉토리가 있음



배포 시 collectstatic 명령어를 실행해서 정적 자원들(ex. html, css 등)을
하나의 디렉토리에 모으는데,
그때 만약 파일명이 중복될 경우 문제가 발생한다.

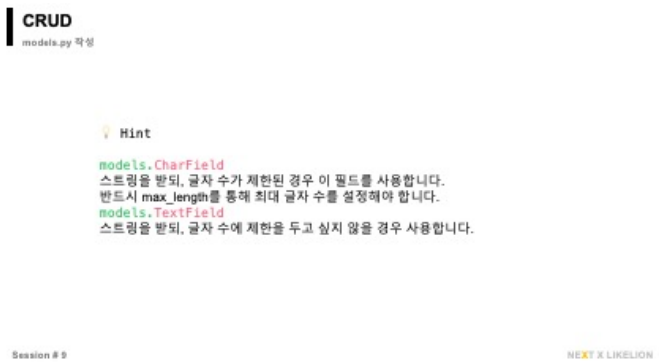
Ex. app1에도 index.html이 있고, app2에도 index.html이 있다고 하면,
이 두 파일이 하나의 디렉토리로 모일 때 문제가 발생하겠죠
그래서 하위에 디렉토리명을 다르게 해서 파일명 중복 문제를 예방한다.

CRUD

실습 안내

실습 슬라이드 구성

문제



힌트

작성 예시(정답)

CRUD

실습 안내

1) 구글링을 해도 좋으니 일단 힌트 안 보고 자기 힘만으로 풀어보기
2) 갈피가 안 잡히면 힌트 보기
3) 답은 웬만하면 끝까지 안 보기

🎯 할 일 – Post 모델 작성

models.py의 Post라는 클래스의 내용을 채워 봅시다

Post 모델에는 제목과 내용 필드가 필요합니다.

*이때 제목은 50글자를 초과하여 작성할 수 없도록 합니다.

CRUD

models.py 작성

💡 Hint

`models.CharField`
스트링을 받되, 글자 수가 제한된 경우 이 필드를 사용합니다.
반드시 max_length를 통해 최대 글자 수를 설정해야 합니다.

`models.TextField`
스트링을 받되, 글자 수에 제한을 두고 싶지 않을 경우 사용합니다.

CRUD

models.py 작성

작성 예시

```
class Post(models.Model):
    title = models.CharField(max_length=50)
    content = models.TextField()
```

+ 모델에 변경사항이 생겼으니

1) python manage.py makemigrations
2) python manage.py migrate

를 해줍시다

Read(List)와 관련된 view 함수를 작성합시다!

🎯 할 일 – list_page view 작성

list_page view 함수를 완성합시다.

모든 Post를 조회하여 그 결과물을 “posts”라는 이름으로 list_page.html에 전달해야 합니다.

CRUD

게시글 리스트 페이지
views.py 작성

💡 Hint

`SomeModel.objects.all()`: 해당 모델의 데이터를 모두 조회한다.

CRUD

게시글 리스트 페이지
views.py 작성

작성 예시

```
def list_page(request):
  posts = Post.objects.all()
  return render(request, ‘app/list_page.html', {'posts': posts})
```

Read(List)와 관련된 html을 작성합시다!

🎯 할 일 – list_page.html 작성

list_page.html을 완성합시다.

1) 조건문 이용해서 p태그(if-no-post)는 게시글이 없을 때만 보이도록 하기

2) (반복문을 이용하여) 게시글 제목들을 a태그로 감싸서 리스팅하기
* 이때, a태그의 class는 "post-title"으로 하기

CRUD

게시글 리스트 페이지
template 작성

💡 Hint

템플릿 문법

1. 변수 값 읽기
{{ 변수 }} ex. {{ post.content }}

2. 조건문
{% if 조건 %}
조건 충족 시 보여줄 문장
{% endif %}

3. 반복문
{% for item in item_list %}
{{ item }}
{% endfor %}

CRUD

게시글 리스트 페이지 template 작성

작성 예시

```
<div class="content-container">
  <h3>게시글 목록</h3>

  {% if not posts %}
  <p class=”if-no-post”>지금은 게시글이 없네요..</p>
  {% endif %}

  {% for post in posts %}
  <a class="post-title” href="{% url 'detail_page' post.pk %}">
  {{ post.title }}
  </a>
  {% endfor %}
</div>
```

Create와 관련된 view 함수를 작성합시다!

🎯 실습 – create_page view 작성

1) request.method가 POST라면,

2) Post 데이터를 생성한다.
* 데이터 생성 시, 'title'이라는 이름의 데이터를 꺼내서 title이라는 필드에 넣고, 'content'라는 이름의 데이터를 꺼내서 content라는 필드에 넣는다.

3) 데이터 생성 후에는 ‘detail_page’ url로 리다이렉트한다.

CRUD

게시글 작성 페이지
views.py 작성

💡**Hint**

1) Form을 전송하는 행위는 POST에 해당한다!

2) Form을 통해 전송한 데이터는 request.POST라는 객체로 전달된다.

3) 이때 데이터의 이름은 input혹은 textarea에 설정한 name 속성값이다.
(request.POST['input_name']과 같은 형식으로 꺼내쓸 수 있음)

4) 데이터 생성은 다음과 같은 문법으로 구현한다.

```
SomeModel.objects.create(
    필드1 = 값1,
    필드2 = 값2
)
```

5) 다른 url로 보내는 리다이렉트는 다음과 같은 문법으로 구현한다.

```
return redirect('url name')
```

CRUD

게시글 작성 페이지
views.py 작성

작성 예시

```
def create_page(request):
    if request.method == 'POST':
        new_post = Post.objects.create(
            title=request.POST['title'],
            content=request.POST['content']
        )
        return redirect('detail_page', new_post.pk)
    return render(request, 'app/create_page.html')
```

CRUD

게시글 작성 페이지 template 작성

Create와 관련된 html을 작성합시다!

🎯 실습 – create_page html 작성

1) form 태그를 만든다(이때 method 속성은 ”POST”로 설정한다.)

2) form내부에 다음을 작성한다.

2-1) csrf 공격을 막는 구문을 작성한다.

2-2) input, textarea 태그를 이용하여 입력값을 받을 요소들을 작성한다.

*받아야하는 입력값은 “titile”과 “content”이다.

*이때 label 태그를 이용하여 각각의 인풋의 이름을 명시하면 좋다.

*placeholder 속성을 통해 안내문을 작성해주면 좋다.

2-3) form을 제출하기 위한 버튼을 만든다.

CRUD

게시글 작성 페이지 template 작성

💡**Hint**

1) {% csrf_token %} 구문으로 csrf 공격을 막을 수 있다.

2) Input, textarea와 같은 입력값을 받는 요소는 해당 태그의 name 속성값을 이름(key값)으로 하여 views.py에 전달이 된다.

3) label과 input/textarea는 label의 for 속성값과 input/textarea의 id 속성값이 일치할 때 연결된다.

CRUD

게시글 작성 페이지
template 작성

작성 예시

```
<div class="content-container">
<form method="POST">
{% csrf_token %}
  <label for="title">제목</label>
  <br>
  <input id="title" name="title" placeholder="제목을 입력하세요" />
  <br><br>
  <label for="content">내용</label><br>
  <textarea id="content" name="content" placeholder="내용을 입력하세요">
  </textarea>
  <br>
  <button type="submit">작성</button>
</form>
</div>
```

Update와 관련된 view 함수를 작성합시다!

🎯 실습 – update_page view 작성

1) request.method가 POST라면,
2) 위에서 조회한 Post 데이터를 수정한다.
   2-1) 데이터 수정 시, 'title'이라는 이름의 데이터를 꺼내서 title이라는 필드에 넣고,
   2-2) 'content'라는 이름의 데이터를 꺼내서 content라는 필드에 넣는다.
3) 데이터 수정 후에는 'detail_page'라는 이름의 url로 리다이렉트한다.
   3-1) detail page로 리다이렉트할 때, 수정한 post 객체의 pk를 넘겨준다.

CRUD

게시글 수정 페이지
views.py 작성

💡**Hint**

1) 데이터 수정은 다음과 같은 문법으로 구현한다.

```
some_queryset.update(
    필드1 = 값1,
    필드2 = 값2
)
```

2) 기타 사항은 Create 로직과 유사하다.

CRUD

게시글 수정 페이지
views.py 작성

작성 예시

```
def edit_page(request, post_pk):
    post = Post.objects.filter(pk=post_pk)
    if request.method == 'POST’:
        post.update(
            title=request.POST['title’],
            content=request.POST['content’]
        )
        return redirect('detail_page', post_pk)
    return render(request,  'app/edit_page.html', {'post': post[0]})
```

Update와 관련된 html을 완성합시다!

🎯 실습 – update_page html 완성

1) title input의 기본값을 해당 게시글(post)의 title 값으로 설정하기
2) content input의 기본값을 해당 게시글(post)의 content 값으로 설정하기

CRUD

게시글 수정 페이지 template 작성

💡**Hint**

1) input 요소의 값은 value라는 속성으로 설정할 수 있다.
2) textarea 요소의 값은 '<textarea>값</textarea>' 과 같이 값을 요소 내부에 작성함으로써 설정할 수 있다.

3) View 함수에서 넘긴 변수를 html에서 사용하는 템플릿 문법은 {{ 변수 }} 형태이다.

CRUD

게시글 수정 페이지
template 작성

작성 예시

```
<div class="content-container">
  <form method="POST">
    {% csrf_token %}
    <label for="title">제목</label><br>
    <input name="title" value="{{ post.title }}"/>
    <br><br>
    <label for="title">내용</label><br>
    <textarea name="content">{{ post.content }}</textarea>
    <br>
    <button type="submit">수정</button>
  </form>
</div>
```

Update와 관련된 html을 완성합시다!

🎯 실습 – delete_page view 완성

1) pk가 post_pk인 Post 객체를 조회한다.

2) 해당 Post 객체를 지운다.

3) 'list_page'라는 이름의 url로 리다이렉트한다.

CRUD

게시글 삭제 기능
view 작성

💡**Hint**

`some_object.delete()`: 해당 객체를 삭제한다.

CRUD

게시글 삭제 기능
view 작성

작성 예시

```
def delete(request, post_pk):
    post = Post.objects.get(pk=post_pk)
    post.delete()
    return redirect('list_page')
```

번외: FBV vs CBV

CBV(Class Based View): 클래스를 이용하여 view를 작성

FBV(Function Based View): 함수를 이용하여 view를 작성

CBV

```
class StudentViewSet(ModelViewSet):
    queryset = Student.objects.all()
    serializer_class = StudentSerializer
```

FBV

```
@api_view(['GET', 'PUT', 'DELETE'])
def student_view(request, pk):
    try:
        student = Student.objects.get(pk=pk)
    except Student.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
    if request.method == 'GET':
        serializer = StudentSerializer(student)
        return Response(serializer.data)
    elif request.method == 'PUT':
        serializer = StudentSerializer(student, request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
    elif request.method == 'DELETE':
        student.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

번외: FBV vs CBV

CBV(class based view)

1) 상속 덕분에 재사용 및 확장이 용이하다.
(미리 구현해둔 걸 가져다가 커스터마이징해서 쓸 수 있음).

2) 코드가 직관적이지 않고, 코드의 흐름이 암시적이기
때문에 구현 및 이해가 상대적으로 어렵다.

FBV(function based view)

1) 코드가 직관적이기 때문에 구현이 쉽고, 읽기가 좋다.
2) 상속할 수 없기 때문에 확장이나 재사용이 불가능하다.

번외: FBV vs CBV

오해

1. CBV의 코드가 뭔가 심오해 보이니, 혹은 코드가 더 짧으니 더 나은 방법이다.
2. FBV의 코드가 직관적이니 더 나은 방법이다.

진실

1. 둘 중에 어떤 것이 더 낫다고 할 수 없다. 각 상황, 취향에 맞게 잘 판단하여 사용한다(섞어쓰기도 가능).
2. FBV로 구현할 수 있는 기능은 CBV로도 구현할 수 있고, 그 역도 성립한다.
3. FBV를 충분히 이해했다고 생각되지 않으면, 우선 FBV를 숙달하는 것이 좋다고 생각한다.

과제

자유롭게 게시판 만들기

1) CRUD 네 가지 기능을 모두 포함해주세요
2) 오늘 했던 실습 코드를 그대로 내지는 말아주세요
3) 최대한 참고 자료를 보지 않고 구현해주세요

선택 과제

1) 조회수 기능을 추가해주세요
2) css로 꾸며주세요

기한
다음주 월요일(5/16)