구분	Django

⭐ 이전에 익힌 QuerySet API를 통해 view 함수에서 직접 CRUD 구현 해보기

▼ 사전 준비

base 템플릿 작성

- bootstrap CDN 및 템플릿 추가 경로 작성

```
<!-- templates/base.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
    <title>CRUD PJT</title>
</head>
<body>
    {% block content %}
    {% endblock content %}
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-w76AqPfDkMB
</body>
</html>
```

```
# settings.py

TEMPLATES = [
```

```
    {
        ...,
        'DIRS': [BASE_DIR / 'templates'],
        ...,
]
```

url 분리 및 연결

```
# crud/urls.py

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

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

```
# articles/urls.py

from django.urls import path
from . import views

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

index 페이지 작성

```
# articles/views.py

from django.shortcuts import render

def index(request):
    return render(request, 'articles/index.html')
```

```
<!-- templates/articles/index.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>INDEX</h1>
{% endblock content %}
```

Article Model 작성

```
from django.db import models

# Create your models here.
class Article(models.Model):
    title = models.CharField(max_length=30)
    content = models.TextField()

    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return f'{self.id}번째 글 - {self.title}'
```

CRUD with view functions

READ - index page

전체 게시글 조회

- index 페이지에서는 전체 게시글을 조회해서 출력

```
# articles/view.py

from django.shortcuts import render
from .models import Article

def index(request):
    articles = Article.objects.all()
    context = {'articles':articles}
    return render(request, 'articles/index.html', context)
```

```
<!-- templates/articles/index.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>INDEX</h1>
    <hr>
    {% for article in articles  %}
        <p>글 번호 : {{article.id}}</p>
        <p>글 제목 : {{article.title}}</p>
        <p>글 내용 : {{article.content}}</p>
        <hr>
    {% endfor %}
{% endblock content %}
```

READ - detail page

- 개별 게시글 상세 페이지 제작
- 모든 게시글 마다 뷰 함수와 템플릿 파일을 만들 수는 없음
    - 글의 번호(pk)를 활용해서 하나의 뷰 함수와 템플릿 파일로 대응
    - Variable Routing 활용

urls

- URL로 특정 게시글을 조회할 수 있는 번호를 받음

```
# articles/urls.py

urlpatterns = [
    ...
    path('<int:pk>/', views.detail, name='detail'),
]
```

views

- Article.objects.get(pk=pk)에서 오른쪽 pk는 variable routing을 통해 받은 pk, 왼쪽 pk는 DB에 저장된 레코드의 id 컬럼

```
# articles/views.py

def detail(request, pk):
    article = Article.objects.get(id=pk)
    context = {'article':article}
    return render(request, 'articles/detail.html', context)
```

templates

```
<!-- templates/articles/detail.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>DETAIL</h1>
    <hr>
```

```
    <p>글 제목 : {{article.title}}</p>
    <p>글 내용 : {{article.content}}</p>
    <p>생성 시각 : {{article.created_at}}</p>
    <p>수정 시각 : {{article.updated_at}}</p>

    <a href="{% url 'articles:index'%}">목록보기</a>

{% endblock content %}
```

제목을 누르면 상세 페이지로 이동하게 하기

```
{% extends 'base.html' %}

{% block content %}
    <h1>INDEX</h1>
    <a href="{% url 'articles:new' %}">글 작성하기</a>
    <hr>
    {% for article in articles  %}
        <p>글 번호 : {{article.id}}</p>
        <p><a href="{% url 'articles:detail' article.pk %}">글 제목 : {{article.title}}</a></p>
        <p>글 내용 : {{article.content}}</p>
        <hr>
    {% endfor %}
{% endblock content %}
```

CREATE

- 사용자의 입력을 받을 페이지를 렌더링 하는 함수 1개
  - `new` view function
- 사용자가 입력한 데이터를 전송 받아 DB에 저장하는 함수 1개
  - `create` view function

New

```
# articles/urls.py

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

```
# articles/views.py

def new(request):
    return render(request, 'articles/new.html')
```

- index 페이지에 new 페이지로 이동할 수 있는 하이퍼 링크 작성

```
<!-- templates/articles/index.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>INDEX</h1>
    <a href="{% url 'articles:new' %}">글 작성하기</a>
    <hr>
    ...
{% endblock content %}
```

Create

```
# articles/urls.py

urlpatterns = [
    ...
```

```
    path('create/', views.create, name='create'),
]
```

- 데이터를 생성하는 3가지 방법
  - 2 또는 3번째 생성 방식 사용
  - create 메서드가 더 간단해 보이지만 추후 데이터가 저장되기 전에 유효성 검사 과정을 거치게 될 예정
  - 유효성 검사가 진행된 후에 save 메서드가 호출되는 구조를 택하기 위함
- `redirect()`
  - 인자에 작성된 곳으로 다시 요청을 보냄
  - 사용 가능한 인자
    1. view name (URL pattern name) `return redirect('articles:index')`
    2. absolute or relative URL `return redirect('/articles/')`
  -

```
# aritlcles/views.py

from django.shortcuts import render, redirect
from .models import Article

def create(request):
    # 입력한 데이터 가져오기
    title = request.POST.get('title')
    content = request.POST.get('content')

    # DB에 새로운 Article 저장
    # 방법 1
    Article.objects.create(title = title, content = content)

    # 방법 2
    article = Article(title=title, content=content)
    article.save()

    # 방법 3
    article = Article()
    article.title = title
    article.content = content
    article.save()

    return redirect('articles:index')
```

```
<!-- templates/articles/new.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>글 작성하기</h1>
    <hr>

    <form action="{% url 'articles:create' %}" method="POST">
        {% csrf_token %}
        <label for ="title">제목 : </label>
        <input type="text" id="title" name="title"><br>

        <label for ="content">내용 : </label>
        <textarea name="content"></textarea><br>

        <input type="submit">

    </form>
    <hr>
    <a href="{% url 'articles:index'%}">목록보기</a>

{% endblock content %}
```

DELETE

- 삭제하고자 하는 특정 글을 조회 후 삭제해야 함

```
# articles/urls.py

urlpatterns = [
    ...
    path('<int:pk>/delete/', views.delete, name='delete'),
]
```

```
# aritlcles/views.py

from django.shortcuts import render, redirect
from .models import Article

def delete(request, pk):
    article = Article.objects.get(pk=pk)
    article.delete()
    return redirect('articles:index')
```

- Detail 페이지에 Template 작성하며, DB에 영향을 미치기 때문에 POST method 사용

```
<!-- templates/articles/detail.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>DETAIL</h1>
    <hr>

    <p>글 제목 : {{article.title}}</p>
    <p>글 내용 : {{article.content}}</p>
    <p>생성 시각 : {{article.created_at}}</p>
    <p>수정 시각 : {{article.updated_at}}</p>

    <form action="{% url 'articles:delete' article.pk %}" method="POST">
        {% csrf_token %}
        <input type="submit" value="DELETE">
    </form>

    <a href="{% url 'articles:index'%}">목록보기</a>

{% endblock content %}
```

Update

- 사용자의 입력을 받을 페이지를 렌더링 하는 함수 1개
  - `edit` view function
- 사용자가 입력한 데이터를 전송받아 DB에 저장하는 함수 1개
  - `update` view function

edit

```
# articles/urls.py

urlpatterns = [
    ...
    path('<int:pk>/edit/', views.edit, name='edit'),
]
```

```
# aritlcles/views.py

def edit(request, pk):
    article = Article.objects.get(pk=pk)
    context = {
        'article' : article,
    }
    return render(request, 'articles/edit.html', context)
```

- html 태그의 value 속성을 사용해 기존에 입력되어 있던 데이터 출력

- textarea 태그는 value 속성이 없으므로 태그 내부 값으로 작성해야 함

```
<!-- templates/articles/edit.html -->

{% extends 'base.html' %}

{% block content %}
<h1>EDIT</h1>
<form action="#" method="POST">
    {% csrf_token %}
    <label for="title">Title : </label>
    <input type="text" name="title" value="{{article.title}}"><br>
    <label for="content">Content : </label>
    <textarea name="content">{{article.content}}</textarea><br>
    <input type="submit">
</form>
<hr>
<a href="{% url 'articles:index' %}">목록보기</a>
{% endblock content %}
```

- Edit 페이지로 이동하기 위한 하이퍼링크 작성

```
<!-- templates/articles/detail.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>DETAIL</h1>
    <hr>

    <p>글 제목 : {{article.title}}</p>
    <p>글 내용 : {{article.content}}</p>
    <p>생성 시각 : {{article.created_at}}</p>
    <p>수정 시각 : {{article.updated_at}}</p>

    <a href="{% url 'articles:edit' article.pk %}">EDIT</a><br>

    <form action="{% url 'articles:delete' article.pk %}" method="POST">
        {% csrf_token %}
        <input type="submit" value="DELETE">
    </form>

    <a href="{% url 'articles:index'%}">목록보기</a>

{% endblock content %
```

Update

```
# articles/urls.py

urlpatterns = [
    ...
    path('<int:pk>/update/', views.update, name='update'),
]
```

```
# aritlcles/views.py

from django.shortcuts import render, redirect
from .models import Article

def update(request, pk):
    article = Article.objects.get(pk=pk)
    article.title = request.POST.get('title')
    article.content = request.POST.get('content')
    article.save()
    return redirect('articles:detail', article.pk)
```

```
<!-- templates/articles/edit.html -->

{% extends 'base.html' %}

{% block content %}
```

```
<h1>EDIT</h1>
<form action="{% url 'articles:update' article.pk %}" method="POST">
    {% csrf_token %}
    ...
</form>
<hr>
<a href="{% url 'articles:index' %}">목록보기</a>
{% endblock content %}
```

HTTP Method

- HTTP
  - 네트워크 상에서 데이터를 주고 받기 위한 약속
- HTTP Method
  - 데이터(리소스)에 어떤 요청(행동)을 원하는지를 나타낸 것

GET

- 특정 리소스를 가져오도록 요청할 때 사용
- 반드시 데이터를 가져올 때만 사용해야 함
- DB에 변화를 주지 않음
- CRUD에서 R 역할을 담당
- GET 방식으로 데이터를 전달하면 Query String 형식으로 보내짐
- 유저가 작성할 데이터 폼을 화면에 띄울땨 GET 사용할 것임

POST

- 서버로 데이터를 전송할 때 사용
- 서버에 변경사항을 만듦
- 리소스를 생성/변경하기 위해 데이터를 HTTP body에 담아 전송
- GET의 쿼리 스트링 파라미터와 다르게 URL로 데이터를 보내지 않음
- CRUD에서 C/U/D 역할을 담당

POST method 적용하기

```
<!-- templates/articles/new.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>글 작성하기</h1>
    <hr>

    <form action="{% url 'articles:create' %}" method="POST">
        {% csrf_token %}
        ...
    </form>
    <hr>
    <a href="{% url 'articles:index'%}">목록보기</a>

{% endblock content %}
```

- 데이터가 담긴 위치가 바뀌었기 때문에 view 함수에서도 수정 필요

```
def create(request):
    title = request.POST.get('title')
    content = request.POST.get('content')
```

403 Forbidden

- 서버에 요청이 전달되었지만, 권한 때문에 거절되었다는 것을 의미
- 서버에 요청은 도달했으나 서버가 접근을 거부할 때 반환됨
- 즉, 게시글을 작성할 권한이 없다

  → Django 입장에서는 '작성자가 누구인지 모르기 때문에 함부로 작성할 수 없다' 라는 의미
- 모델(DB)을 조작하는 것은 단순 조회와 달리 최소한의 신원 확인이 필요하기 때문

CSRF

- Cross-Site-Request-Forgery
- 사이트 간 요청 위조
- 사용자가 자신의 의지와 무관하게 공격자가 의도한 행동을 하여 특정 웹페이지를 보안에 취약하게 하거나 수정, 삭제 등의 작업을 하게 만드는 공격 방법
- CRSF 공격 방어 - Security Token 사용 방식 (CSRF Token)
  - 사용자의 데이터에 임의의 난수 값(token)을 부여해 매 요청마다 해당 난수 값을 포함시켜 전송 시키도록 함
  - 이후 서버에서 요청을 받을 때마다 전달된 token 값이 유효한지 검증
  - 일반적으로 데이터 변경이 가능한 POST, PATCH, DELETE Method 등에 적용
  - Django는 DTL 에서 csrf_token 템플릿 태그 제공

crsf_token 템플릿 태그

`{% csrf_token %}`

- 해당 태그가 없다면 Django는 서버 요청에 대해 403 forbidden으로 응답
- 템플릿에서 내부 URL로 향하는 Post form을 사용하는 경우에 사용
  - 외부 URL로 향하는 POST form에 대해서는 CSRF 토큰이 유출되어 취약성을 유발할 수 있기 때문에 사용해서는 안됨
- csrf_token은 해당 POST 요청이 내가 보낸것 인지를 검증하는 것!

Handling HTTP requests

new-create, edit-update의 view 함수

공통점

- new - create는 모두 CREATE 로직을 구현하기 위한 공통 목적
- edit - update는 모두 UPDATE 로직을 구현하기 위한 공통 목적

차이점

- new와 edit는 GET 요청에 대한 처리만을 진행
- create와 update는 POST 요청에 대한 처리만을 진행

→ 하나의 view 함수에서 method에 따라 로직이 분리되도록 변경하기!

Create

- new와 create view 함수를 합침
- 각각의 역할은 `request.method` 값을 기준으로 나뉨

```
def create(request):
    if request.method == 'POST':
        title = request.POST.get('title')
        content = request.POST.get('content')
        article = Article(title=title, content=content)
        article.save()
        return redirect('articles:index')
```

```
    else:
        return render(request, 'articles/create.html')
```

Update

- edit와 update view 함수 합침

```
def update(request, pk):
    if request.method == 'POST':
        article = Article.objects.get(pk=pk)
        article.title = request.POST.get('title')
        article.content = request.POST.get('content')
        article.save()
        return redirect('articles:detail', article.pk)
    else:
        article = Article.objects.get(pk=pk)
        context = {'article' : article}
        return render(request, 'articles/edit.html', context)
```

Delete

- POST 요청에 대해서만 삭제가 가능하도록 수정

```
def delete(request, pk):
    article = Article.objects.get(pk=pk)
    if request.method == 'POST':
        article.delete()
        return redirect('articles:index')
    else:
        return redirect('articles:detail', article.pk)
```