

## 0. 지난 실습에서 만들었던 Python 가상환경을 삭제하고 다시 시작합니다.

C:\work\_django\django\_mldl 폴더 안의 [ **django\_env** ] 폴더를 삭제해주세요.

## 1. 장고 프로젝트 폴더 접근 & Python 가상환경 생성 및 activate (cmd 관리자권한으로 실행)

**cd ..** 명령어로 c drive까지 나가기

1-1) c:\work\_django\django\_mldl 폴더가 없는 경우 (지난 수업에 참석 X 시)

**mkdir work\_django** -> c:\work\_django 폴더가 생깁니다. (이미 폴더가 있을 경우 Skip)

**cd work\_django**

**mkdir django\_mldl** -> c:\work\_django\django\_mldl 폴더가 생깁니다. (이미 폴더가 있을 경우 Skip)

**cd django\_mldl**

1-2) c:\work\_django\django\_mldl 폴더가 있는 경우 (지난 수업에 참석 O 시)

**cd work\_django**

**cd django\_mldl**

## 2) 가상환경 생성 및 Django 라이브러리 설치

`pip install virtualenv==16.7.7` -> 가상 환경을 만들어주는 라이브러리입니다.

`virtualenv django_env` -> `django_env` 라는 이름으로 가상 환경을 새로이 만듭니다.

`django_env\Scripts\activate` -> Scripts 폴더 내의 activate 파일을 실행해 가상 환경을 활성화합니다.

`pip install django==2.2.6` -> 가상 환경에는 최소의 라이브러리만 존재하므로 장고를 설치해줍니다.

## 2. 장고 프로젝트 생성 (C:\work\_django\django\_mdl)

: project는 하나의 웹사이트에 해당합니다.

`django-admin startproject site_2` -> `site_2` 폴더가 생깁니다.

`cd site_2`

Atom에서 프로젝트 폴더 열기

: File -> **Add Project Folder...** 에서 `django_mdl > site_2` 폴더 선택하기 (`django_mdl` 폴더가 아닙니다.)

C:\work\_django\django\_mdl\site\_2 -> **settings.py** 파일 수정 & 추가

`LANGUAGE_CODE = 'en-us'` -> **'ko-kr'**

`TIME_ZONE = 'UTC'` -> **'Asia/Seoul'**

`STATIC_ROOT = os.path.join(BASE_DIR, 'static')`

### 3. 장고 app 생성 (C:\work\_django\django\_mdl\site\_2)

: app 은 하나의 웹사이트 내에 있는 기능들에 해당합니다. (회원가입 기능, 상품 관련 기능 등)

`python manage.py startapp polls` -> polls 폴더가 생깁니다.

C:\work\_django\django\_mdl\site\_2\site\_2 -> `settings.py` 파일 수정

```
INSTALLED_APPS = [
```

```
    ... ,
```

```
    'polls',
```

```
]
```

장고 프로젝트 Server 실행 & 웹브라우저에서 확인

`python manage.py runserver` (포트번호 변경을 원할 시 : `python manage.py runserver 8888`)

브라우저에서 `127.0.0.1:8000` 접속

### 4. views.py에 index 함수 만들고 polls app의 urls.py에 등록하기

\* `site_2\site_2\urls.py` 파일의 상단 주석에 적힌 **URL Configuration** 방식 중 [ **Including another URLconf** ] 방식을 활용하는 방법입니다.

\* 기존에 site\_2 폴더 내의 `urls.py`가 모두 처리하던 전체 URL들을 App 단위로 나눠 app 폴더 내의 `urls.py`에게 권한을 위임하는 과정입니다.

4-1) C:\work\_django\django\_mdl\site\_2\polls -> `views.py` 파일 수정

```
from django.shortcuts import render
```

```
from django.http import HttpResponse
```

# Create your views here.

**def index(request):**

**return HttpResponse('Hello world')**

4-2) C:\work\_django\django\_mdl\site\_2\polls -> **urls.py** 파일 생성 후 아래 내용 작성

**from django.urls import path**

**from . import views** # 같은 폴더 내의 views.py를 import

**urlpatterns = [**

# "127.0.0.1:8000/polls/" 이후의 URL은 polls/urls.py가 handling하도록 만들 예정입니다.

**path("", views.index, name='index'),** # '127.0.0.1:8000/polls/' 를 받아내도록 만들어줄 것입니다.

**]**

4-3) C:\work\_django\django\_mdl\site\_2\site\_2 -> **urls.py** 파일 수정하여 **polls\urls.py** 파일을 **include**하기

**from django.contrib import admin**

**from django.urls import path, include**

**urlpatterns = [**

**path('admin/', admin.site.urls),**

# "127.0.0.1:8000/polls/" 이후의 URL은 polls/urls.py가 handling

**path('polls/', include('polls.urls')),**

**]**

4-4) 장고 프로젝트 Server 실행 & 웹브라우저에서 확인

**python manage.py runserver**

브라우저에서 접속 @ 127.0.0.1:8000/polls

## 5. polls app의 models.py 수정하여 DB tables 만들기

\* 먼저 강의안 PDF 파일의 슬라이드 45page 에서 만들려는 DB 구조를 살펴보고 돌아와주세요.

5-1) C:\work\_django\django\_mldl\site\_2\polls -> **models.py** 파일 수정

```
from django.db import models
```

```
# Create your models here.
```

```
class Question(models.Model): # DB Table for 설문조사 주제
```

```
    question_text = models.CharField(max_length=200) # 설문조사 주제 텍스트
```

```
    pub_date = models.DateTimeField('date published') # 'date published' : 관리자 페이지에서 보여질 항목명
```

```
class Choice(models.Model): # DB Table for 설문조사 주제별 선택지 (+ 선택지마다의 득표 수)
```

```
    # 자동으로 Question table의 Primary key를 Foreign Key로 세팅
```

```
    # on_delete=models.CASCADE : Question(질문) 항목 삭제 시 관계된 선택지들도 모두 자동 삭제
```

```
    question = models.ForeignKey(Question, on_delete=models.CASCADE) # 설문조사 주제의 id 값
```

```
    choice_text = models.CharField(max_length=200) # 설문조사 주제에 대한 선택지 텍스트
```

```
    votes = models.IntegerField(default=0) # 해당 선택지의 득표 수
```

## 5-2) makemigrations & migrate

**Ctrl + C 로 서버 종료**

```
python manage.py makemigrations
```

```
python manage.py migrate
```

-> C:\work\_django\django\_mdl\site\_2\polls\migrations 폴더의 0001\_initial.py 에서 실제 migration 코드를 볼 수 있음

```
python manage.py sqlmigrate polls 0001
```

-> 0001\_initial.py 에 의해 실제로 수행되는 SQL 명령어를 확인할 수 있음 (ORM)

## 6. shell 을 통해 만들어낸 model class 확인해보기

6-1) shell 에서 설문조사 항목(Question)을 직접 만들고 결과 확인해보기

@ cmd,

```
python manage.py shell
```

```
from polls.models import Question, Choice
```

```
Question.objects.all()
```

```
from django.utils import timezone
```

```
q = Question(question_text="What's the best treatment?", pub_date=timezone.now())
```

```
q.save() <- q라는 이름으로 만들어진 데이터 행을 실제로 DB에 저장합니다.
```

```
q.id
```

```
q.question_text
```

```
q.pub_date
```

```
q.question_text = "What's the worst treatment?"
```

```
q.save()
```

```
Question.objects.all()
```

```
exit()
```

-> Question model object를 출력했을 때 <Question: Question object (1)> 라고만 출력되므로 어떤 설문조사 주제인지 한 눈에 파악이 어렵습니다. **Question class**의 **\_\_str\_\_()** 함수를 고쳐서 object를 호출했을 때 설문조사 주제(question\_text)가 눈 앞에 출력되도록 고쳐봅시다.

6-2) C:\work\_django\django\_mdl\site\_2\polls -> **models.py** 파일 수정

```
from django.db import models
```

```
from django.utils import timezone
```

```
import datetime
```

```
# Create your models here.
```

```
class Question(models.Model): # DB Table for 설문조사 주제
```

```
    question_text = models.CharField(max_length=200) # 설문조사 주제 텍스트
```

```
    pub_date = models.DateTimeField('date published') # 'date published' : 관리자 페이지에서 보여질 항목명
```

```
# Shell이나 관리자 페이지 등에서 DB Table 내의 데이터를 꺼냈을 때 보여지는 텍스트를 지정합니다.
```

```
def __str__(self):
```

```
    return self.question_text
```

```
# 현재 기준으로 하루 전 시점보다 더 이후에 등록된 Question인지 여부를 확인해주는 함수(True/False)
```

```
def was_published_recently(self):
```

```
    now = timezone.now()
```

```
    return now >= self.pub_date >= now - datetime.timedelta(days=1)
```

```

class Choice(models.Model): # DB Table for 설문조사 주제별 선택지 (+ 선택지마다의 득표 수)

    # 자동으로 Question table의 Primary key를 Foreign Key로 세팅

    # on_delete=models.CASCADE : Question(질문) 항목 삭제 시 관계된 선택지들도 모두 자동 삭제

    question = models.ForeignKey(Question, on_delete=models.CASCADE) # 설문조사 주제의 id 값

    choice_text = models.CharField(max_length=200) # 설문조사 주제에 대한 선택지 텍스트

    votes = models.IntegerField(default=0) # 해당 선택지의 득표 수


    def __str__(self):

        return self.choice_text

```

### 6-3) shell 에서 변경사항 확인해보기

(Django ORM의 **filter**에 대해 다룹니다. 너무 어려울 경우 Skip하셔도 다음 단계의 실습들에 문제가 없습니다.)

@ cmd,

**python manage.py shell**

```

from polls.models import Question, Choice

```

```

Question.objects.all()

```

```

Question.objects.filter(id=1)

```

```

Question.objects.filter(question_text__startswith='What') <- question_text는 column(field), __ 이후는 조건
__ : 밑줄 1개가 아니라 2개입니다.

```

```

from django.utils import timezone

```

```

current_year = timezone.now().year

```

```

Question.objects.get(pub_date__year=current_year)

```

```

Question.objects.get(pk=1)

```

```

q = Question.objects.get(pk=1)

```

```

q.was_published_recently()

```



#### 6-4) Question object(설문조사 주제)에 대한 Choice 객체(설문조사 주제별 선택지) 만들기

`q.choice_set.all()` <- "**choice\_set**"은 Question class object인 **q**에 연결된 **Choice** table의 **object set**을 의미합니다.

즉, "What's the worst treatment?" 설문조사 주제에 연결된 선택지들의 **set**을 의미합니다.

만약 Question table에 연결된 table의 이름이 **Choice**가 아니라 **Voters**였다면,

`q.choice_set.all()` 대신 `q.voters_set.all()` 을 입력해줘야 합니다. (자동으로 소문자로 치환됨)

```
q.choice_set.create(choice_text='Treatment A', votes=0)
```

```
q.choice_set.create(choice_text='Treatment B', votes=0)
```

`c = q.choice_set.create(choice_text='Treatment C', votes=0)` <- 만들어진 **Choice table**의 **row**가 `c`에 저장됩니다.

`c.question` <- 해당 row의 question 열의 값입니다. (슬라이드 45page의 DB 구조를 살펴봐주세요.)

```
q.choice_set.all()
```

```
q.choice_set.count()
```

```
Choice.objects.filter(question__pub_date__year=current_year)
```

\_\_ : 밑줄 1개가 아니라 2개입니다.

```
c = q.choice_set.filter(choice_text__startswith='Treat')
```

```
c.delete()
```

```
q.choice_set.all()
```

```
exit()
```

## 7. admin 페이지 세팅하기

7-1) admin page 로그인을 위한 사용자이름 & 비밀번호 생성하기 @ **cmd**

**python manage.py createsuperuser**

Username / Email / Password 입력하기 (Password는 입력 시 화면으로 출력되지 않음)

7-2) C:\work\_django\django\_mdl\site\_2\wpolls -> **admins.py** 수정하여 **Question & Choice Class** 등록하기

```
from django.contrib import admin
```

```
from .models import Question, Choice
```

```
# Register your models here.
```

```
admin.site.register(Question)
```

```
admin.site.register(Choice)
```

7-3) admin 페이지에서 확인하기

**python manage.py runserver**

브라우저에서 접속 @ 127.0.0.1:8000/admin

7-4) admin 페이지에서 **Question & Choice** 추가

### **Questions** 항목 추가

- 디자인 A/B/C 안 중 가장 좋은 것은?
- 마케팅 a/b/c 안 중 가장 좋은 것은?

### **Choice** 추가

- 디자인 A, 디자인 B, 디자인 C
- 마케팅 a, 마케팅 b, 마케팅 c

## 8. 투표 진행 및 투표 결과 확인을 위한 views.py & urls.py 수정

8-1) C:\work\_django\django\_mdl\site\_2\polls -> **views.py** 파일 수정 (아래 3개 함수 추가)

```
def detail(request, question_id):

    return HttpResponse("You're looking at question {}".format(question_id))


def results(request, question_id):

    response = "You're looking at the results of question {}."

    return HttpResponse(response.format(question_id))


def vote(request, question_id):

    return HttpResponse("You're voting on question {}".format(question_id))
```

8-2) C:\work\_django\django\_mdl\site\_2\polls -> **urls.py** 파일 수정

```
urlpatterns = [

    # "127.0.0.1:8000/polls/" 이후의 URL은 polls/urls.py가 handling하도록 만들 예정입니다.

    path("", views.index, name='index'), # '127.0.0.1:8000/polls/' 를 받아내도록 만들어줄 것입니다.

    # ex: /polls/5/

    path('<int:question_id>/', views.detail, name='detail'),

    # ex: /polls/5/results/

    path('<int:question_id>/results/', views.results, name='results'),

    # ex: /polls/5/vote/
```

```
path('<int:question_id>/vote/', views.vote, name='vote'),  
]
```

8-3) 브라우저에서 아래 URL로 각각 접속하여 결과 확인

<http://127.0.0.1:8000/polls/1/>

<http://127.0.0.1:8000/polls/1/results/>

<http://127.0.0.1:8000/polls/1/vote/>

## 9. 전체 투표(Question) 목록을 확인할 수 있는 메인페이지 template 연동하기

9-1) C:\work\_django\django\_mdl\site\_2\polls -> **views.py** 파일 수정

```
from django.shortcuts import render
```

```
from django.http import HttpResponse
```

```
from .models import Question
```

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

```
    # order_by('-pub_date')[:5] : 등록 날짜 기준 내림차순 정렬 후 앞에서 5개까지만
```

```
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
```

```
    # 지난 실습에서 render() 함수의 {~::~} 로 html 파일에게 넘겨주던 dict를 context라고 부릅니다.
```

```
    context = {'latest_question_list': latest_question_list}
```

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

...

## 9-2) **template** 파일 만들기 (index.html)

1. polls 폴더 아래에 **templates** 폴더 만들기 (복수형임을 꼭 유의해주세요!)
2. templates 폴더 아래에 **polls** 폴더 만들기
3. polls 폴더 안에 **index.html** 생성 후 내용 작성

-> Atom에서 polls 폴더 우클릭 후 New file 클릭한 다음

**pollsWtemplatesWpollsWindex.html** 입력 시 한번에 위 1~3까지의 작업을 진행할 수 있음

@ pollsWtemplatesWpollsWindex.html,

```
{% if latest_question_list %}
```

```
<ul>
```

```
    {% for question in latest_question_list %}
```

```
    <li>
```

```
        <a href="/polls/{{ question.id }}/">{{ question.question_text }}</a>
```

```
    </li>
```

```
    {% endfor %}
```

```
</ul>
```

```
{% else %}
```

```
<p>No polls are available.</p>
```

```
{% endif %}
```

## 9-3) 브라우저에서 아래 URL로 접속하여 결과 확인

<http://127.0.0.1:8000/polls/>

## 10. 투표(Question) 상세 페이지 template 연동하기 & 404 에러페이지 다루기

10-1) C:\work\_django\django\_mdl\site\_2\polls -> **views.py** 파일 수정

...

```
def detail(request, question_id):
```

```
    q = Question.objects.get(pk=question_id)
```

```
    return render(request, 'polls/detail.html', {'question':q})
```

...

10-2) template 파일 만들기 (detail.html)

polls\templates\polls\detail.html 만들고 아래와 같이 작성

```
{{ question }}
```

10-3) 브라우저에서 아래 URL로 접속하여 결과 확인

<http://127.0.0.1:8000/polls/>

-> **Question** 항목 링크 클릭

10-4) 브라우저에서 아래 URL로 접속하여 결과 확인

**\* 404 에러를 일부러 발생시켜보고 어떻게 처리할지 실습합니다.**

<http://127.0.0.1:8000/polls/20>

-> **DoesNotExist at /polls/20/** 에러 페이지가 출력되는 것을 확인

10-5) C:\work\_django\django\_mdl\site\_2\wpolls -> **views.py** 파일 수정

```
from django.shortcuts import render

from django.http import HttpResponse, Http404

from .models import Question

...

def detail(request, question_id):

    try:

        q = Question.objects.get(pk=question_id)

    except Question.DoesNotExist:

        raise Http404('Question {} does not exist'.format(question_id))

    return render(request, 'polls/detail.html', {'question':q})

...
```

10-6) 브라우저에서 아래 URL로 접속하여 결과 확인

<http://127.0.0.1:8000/polls/20>

-> **Page not found (404)** 페이지 출력

10-7) C:\work\_django\django\_mdl\site\_2\wpolls -> **views.py** 파일 수정해서 간결하게 404 처리하기

```
from django.shortcuts import render, get_object_or_404

from django.http import HttpResponse, Http404
```

```
from .models import Question
```

```
def detail(request, question_id):
```

```
    # list(QuerySet)가 return될 시에는 get_object_or_404 대신 get_list_or_404를 활용
```

```
    q = get_object_or_404(Question, pk=question_id)
```

```
    return render(request, 'polls/detail.html', {'question':q})
```

10-8) 브라우저에서 아래 URL로 접속하여 결과 확인

<http://127.0.0.1:8000/polls/20>

-> **Page not found (404)** 페이지 출력

-> 이후 **pollsWtemplatesWpolls** 폴더 내에 **404.html** 파일을 만들고 내용을 꾸며주면, 추후 본 웹사이트를 배포할 때 settings.py의 DEBUG = True가 DEBUG = False로 바뀌었을 시 모든 404 에러는 404.html 파일을 보여주게 되므로 에러 대응이 편리해집니다. (404 page 커스터마이징 방법 @ <http://j.mp/3bqxURL>)

## 11. pollsWurls.py 의 url 패턴이 갖고 있는 name을 활용해 하드코딩된 URL 바뀔주기

11-1) C:\work\_django\django\_mdl\site\_2\polls -> **urls.py** 파일 수정

```
app_name = 'polls'
```

```
urlpatterns = [
```

```
    ...
```

```
    path('<int:question_id>/', views.detail, name='detail'),
```

```
    ...
```



]

11-2) polls\templates\polls\index.html 수정해주기

\* 아래 코드에서 question.id 를 question.pk 로 변경해줘도 결과는 동일합니다.

```
{% if latest_question_list %}

<ul>

    {% for question in latest_question_list %}

    <li>

        <!-- <a href="/polls/{{ question.id }}">{{ question.question_text }}</a> -->

        <a href="{% url 'polls:detail' question.id %}">{{ question.question_text }}</a>

    </li>

    {% endfor %}

</ul>

{% else %}

    <p>No polls are available.</p>

{% endif %}
```

11-3) 브라우저에서 아래 URL로 접속하여 결과 확인

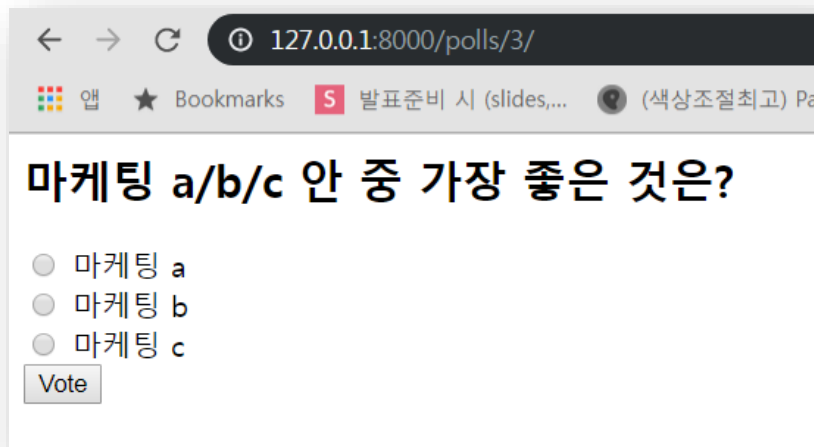
<http://127.0.0.1:8000/polls/>

-> 개발자도구를 열고 a 태그에 매겨져 있는 href 값을 확인합니다.

(위 index.html 코드에서 {% url ~ %})이 자동으로 생성해 준 URL 입니다.)

## 12. 투표(Question) 상세 페이지에서 투표를 하기 위한 Form 만들기

12-1) polls\templates\polls\detail.html 파일 수정

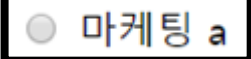


\* 강의안 PDF 에서 슬라이드 75page 를 통해 만들려는 페이지의 모습을 먼저 확인해주세요.

```
{% if error_message %}
<p><strong>{{ error_message }}</strong></p>
{% endif %}
```

\* **error\_message** 관련 코드 : 추후 선택지 옵션 중 어느 것도 선택하지 않고 Vote 버튼 클릭 시(form을 submit 할 시) 직접 만든 에러 메시지를 보여주기 위한 처리

```
<input type="radio" name="choice_select" id="choice{{ forloop.counter }}" value="{{ choice.id }}">
<label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label>
```

== 

== 

```
<input type="radio" name="choice_select" id="choice1" value="7">
<label for="choice1">마케팅 a</label>
```

\* radio input 태그의 **id** : 각 radio button과 해당 button에 대한 label을 연결해주는 역할

\* radio input 태그의 **name** : 추후 views.py에서 vote() 함수에게 넘겨진 전체 POST Request로부터 이 radio input 태그 자체를 선택할 수 있게 해주는 역할 (슬라이드 77page에서 **request.POST['choice\_select']** 참고)

\* radio input 태그의 **value** : 추후 views.py에서 vote() 함수에게 넘겨진 전체 POST Request에서 꺼내어 활용할 선

택된 Choice 객체(설문조사 선택지, 위 이미지에서 '마케팅 a'에 해당)의 Primary Key 값

\* **forloop.counter** : for 문을 돌면서 1부터 차례대로 증가하며 자동으로 매겨지는 숫자 -> choice1, choice2, ...

```
<h2>{{ question.question_text }}</h2>
```

```
{% if error_message %}
```

```
    <p><strong>{{ error_message }}</strong></p>
```

```
{% endif %}
```

```
<form action="{% url 'polls:vote' question.id %}" method="post">
```

```
    {% csrf_token %}
```

```
    {% for choice in question.choice_set.all %}
```

```
        <input type="radio" name="choice_select" id="choice{{ forloop.counter }}" value="{{ choice.id }}">
```

```
        <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label>
```

```
        <br>
```

```
    {% endfor %}
```

```
    <input type="submit" value="Vote">
```

```
</form>
```

12-2) 브라우저에서 아래 URL로 접속하여 결과 확인

<http://127.0.0.1:8000/polls/>

-> Question 중 하나의 링크를 클릭하여 상세 페이지 출력 확인

### 13. 투표(Question) 상세 페이지에서 투표를 마쳤을 때 (form이 제출되었을 때) 해당 POST 요청을 처리해줄 vote 함수 수정하기

13-1) C:\work\_django\django\_mdl\site\_2\polls -> **views.py** 파일 수정

\* 강의안 PDF 슬라이드 76page에서 **try/except/else**의 작동 순서를 먼저 살펴보고 돌아와주세요.

```
from django.shortcuts import render, get_object_or_404, redirect
```

```
from django.http import HttpResponse, Http404
```

```
from .models import Question
```

```
...
```

```
def vote(request, question_id):
```

```
    question = get_object_or_404(Question, pk=question_id)
```

```
    # print(request.POST)
```

```
    # return HttpResponse('vote')
```

```
    try:
```

```
        selected_choice = question.choice_set.get(pk=request.POST['choice_select'])
```

```
        # request.POST['choice_select'] :
```

```
        # detail.html의 <input type="radio" name="choice_select" value="{{ choice.id }}">에서 날라온 값
```

```
        # form으로 제출된 POST Request 전체에서 'choice_select'가 name인 HTML 태그의 value를 꺼내는 코드
```

```
        # request.POST 는 {~~~, 'choice_select':7} 와 같은 dictionary 형태
```

```
    except:
```

```
        # request.POST['choice_select']값이 없을 경우, error_message를 가지고 details.html로 되돌아감
```

```
        context = {'question': question, 'error_message': "You didn't select a choice."}
```

```
        return render(request, 'polls/detail.html', context)
```

```
    else: # try 문에서 에러가 발생하지 않았을 경우 마지막에 실행됩니다.
```

```
        selected_choice.votes += 1
```

```
        selected_choice.save() # 실제 DB 저장
```

```
        return redirect('polls:results', question_id = question.id)
```

...

13-2) 브라우저에서 아래 URL로 접속하여 결과 확인

<http://127.0.0.1:8000/polls/>

-> Question 중 하나의 링크를 클릭하여 투표 진행 후 결과 페이지 출력 확인

13-3) 에러 발생 시 결과를 확인하기 위해 크롬 개발자도구에서 아래와 같이 수정

\* 선택한 설문조사 항목 및 선택지 항목에 따라 기존 value 값은 7이 아닐 수도 있습니다.

```
<input type="radio" name="choice_select" id="choice1" value="7">
```

```
<input type="radio" name="choice_select" id="choice1" value="99">
```

-> Vote 버튼 클릭하여 **error\_message** ("You didn't select a choice.") 출력 확인

## 14. 투표를 마쳤을 때 결과를 보여줄 result.html 만들기

14-1) C:\work\_django\django\_mdl\site\_2\wpolls -> **views.py** 파일 수정

```
def results(request, question_id):
```

```
    question = get_object_or_404(Question, pk=question_id)
```

```
    return render(request, 'polls/result.html', {'question': question})
```

14-2) pollsWtemplatesWpolls 폴더에 **result.html** 파일 생성 및 작성

```
<h1>{{ question.question_text }}</h1>
```

```
<ul>
```

```
{% for choice in question.choice_set.all %}
```

```
<li>{{ choice.choice_text }} got voted for {{ choice.votes }} times.</li>
```

```
{% endfor %}
```

```
</ul>
```

```
<a href="{% url 'polls:detail' question.id %}">Vote again?</a>
```

14-3) 브라우저에서 아래 URL로 접속하여 결과 확인

<http://127.0.0.1:8000/polls/>

-> Question 항목 클릭하여 투표 진행 후 결과 페이지 출력 확인

## 15. admin page 커스터마이징하기

\* 15번 파트의 관리자 페이지 커스터마이징 실습은 **강의안 PDF 슬라이드에서 적용하려는 변화를 각각 먼저 확인** 하신 후 해당 변화를 적용해주는 코드를 작성하시는 것을 권장드립니다.

15-1) 브라우저에서 아래 URL로 접속하여 현재 관리자 페이지의 모습을 확인

<http://127.0.0.1:8000/admin>

## 15-2) Question 추가 페이지에서 항목 순서 변경하기

(순서 변경이 가능하다는 것만 확인하기 위한 과정이며, 원래대로 다시 되돌릴 예정입니다.)

<http://127.0.0.1:8000/admin> 접속

-> Questions "추가" 버튼을 클릭하여 **Question 추가 페이지의 현재 모습**을 확인

C:\work\_django\django\_mdl\site\_2\wpolls -> **admins.py 수정하기**

```
# Register your models here.
```

```
class QuestionAdmin(admin.ModelAdmin):
```

```
    # Question 추가 페이지 내 항목들의 순서 변경(지정)
```

```
    fields = ['pub_date', 'question_text'] # fields 변수명은 고정입니다.
```

```
admin.site.register(Question, QuestionAdmin)
```

```
admin.site.register(Choice)
```

-> 브라우저에서 새로고침하여 Question 추가 페이지 변경 확인

<http://127.0.0.1:8000/admin/polls/question/add/>

## 15-3) Question 추가 페이지에서 항목들에 대한 소제목 추가해주기

C:\work\_django\django\_mdl\site\_2\wpolls -> **admins.py 수정하기**

...

# fieldsets 변수명은 고정입니다.

```
fieldsets = [  
    # ('field 집합의 소제목', {'fields': ['field 이름 1', 'field 이름 2', ...]}),  
    ("Question title", {'fields': ['question_text']}),  
    ('Date information', {'fields': ['pub_date']}),  
]
```

...

-> 브라우저에서 새로고침하여 Question 추가 페이지 변경 확인

<http://127.0.0.1:8000/admin/polls/question/add/>

15-4) Question 추가 페이지에서 Choice까지 한번에 추가할 수 있도록 수정하기

: C:\work\_django\django\_mdl\site\_2\polls -> **admins.py** 수정하기

...

```
class ChoiceInline(admin.StackedInline):
```

```
    model = Choice
```

```
    extra = 2 # Default로 보여줄 Choice 입력 slot의 수
```

```
class QuestionAdmin(admin.ModelAdmin):
```

```
    # fieldsets 변수명은 고정입니다.
```

```
    fieldsets = [  
        # ('field 집합의 소제목', {'fields': ['field 이름 1', 'field 이름 2', ...]}),  
        ("Question title", {'fields': ['question_text']}),  
        ('Date information', {'fields': ['pub_date']}),  
    ]
```

```
    inlines = [ChoiceInline]
```

...



-> 브라우저에서 새로고침하여 Question 추가 페이지 변경 확인

<http://127.0.0.1:8000/admin/polls/question/add/>

15-5) Choice 추가를 위한 Inline 영역이 너무 길어지지 않도록 간결하게 수정 (**TabularInline** 활용)

: C:\work\_django\django\_mdl\site\_2\polls -> **admins.py** 수정하기

...

```
class ChoiceInline(admin.TabularInline):
```

```
    model = Choice
```

```
    extra = 1 # Default로 보여줄 Choice 입력 slot의 수
```

...

-> 브라우저에서 새로고침하여 Question 추가 페이지 변경 확인

<http://127.0.0.1:8000/admin/polls/question/add/>

15-6) 중요하지 않은 항목을 감추기 (**collapse** class 적용)

: C:\work\_django\django\_mdl\site\_2\polls -> **admins.py** 수정하기

```
class QuestionAdmin(admin.ModelAdmin):
```

```
    fieldsets = [
```

```
        ("Question title", {'fields': ['question_text']}),
```

```
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
```

```
    ]
```

```
    inlines = [ChoiceInline]
```

-> 브라우저에서 새로고침하여 Question 추가 페이지 변경 확인

<http://127.0.0.1:8000/admin/polls/question/add/>

15-7) 생성한 Question 리스트 페이지에서 **question\_text** 이외의 정보들을 추가로 출력해 주기

: C:\work\_django\django\_mdl\site\_2\polls -> **admins.py** 수정하기

...

```
class QuestionAdmin(admin.ModelAdmin):

    # fieldsets 변수명은 고정입니다.

    fieldsets = [

        # ('field 집합의 소제목', {'fields': ['field 이름 1', 'field 이름 2', ...]}),

        ("Question title", {'fields': ['question_text']}),

        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),

    ]

    inlines = [ChoiceInline]

    # list_display 변수명은 고정입니다. (소괄호는 생략되어도 무방)

    list_display = ('question_text', 'pub_date', 'was_published_recently')
```

...

-> 브라우저에서 새로고침하여 Question 리스트 페이지 변경 확인

<http://127.0.0.1:8000/admin/polls/question/>

15-8) 생성한 Question 리스트 페이지에서 **'was\_published\_recently'** 항목을 **아이콘으로 변경**하고 정렬 기능 부여하기

\* Question 리스트 페이지(<http://127.0.0.1:8000/admin/polls/question/>)에서 'QUESTION TEXT'와 'DATE PUBLISHED'는 열의 제목을 클릭했을 때 정렬 기능이 작동됩니다. 반면 **'WAS PUBLISHED RECENTLY' 열은 클릭이 불가능(정렬이 불가능)**한데 해당 열을 설문조사 주제 생성 시간(pub\_date)을 기준으로 정렬 가능하도록 변경하겠습니다.

C:\work\_django\django\_mdl\site\_2\polls -> **models.py** 파일 수정하기

...

```
def was_published_recently(self):

    now = timezone.now()

    return now >= self.pub_date >= now - datetime.timedelta(days=1)
```

# 관리자 페이지 Question 리스트에서 True/False 문구를 아이콘으로 변경

**was\_published\_recently.boolean = True**

# 'WAS PUBLISHED RECENTLY' 열의 정렬 기준을 pub\_date(설문조사 주제 생성 시간)로 세팅

**was\_published\_recently.admin\_order\_field = 'pub\_date'**

# 'WAS PUBLISHED RECENTLY' 열의 이름 변경

**was\_published\_recently.short\_description = 'Published recently?'**

...

-> 브라우저에서 새로고침하여 Question 리스트 페이지 변경 확인

<http://127.0.0.1:8000/admin/polls/question/>

15-9) 생성한 Question 리스트 페이지에서 **필터 & 검색 기능** 추가하기

: C:\work\_django\django\_mdl\site\_2\polls -> **admins.py 수정하기**

...

```
class QuestionAdmin(admin.ModelAdmin):
```

...

```
list_display = ('question_text', 'pub_date', 'was_published_recently')
```

**list\_filter = ['pub\_date']** # pub\_date(설문조사 생성 시간)을 기준으로 **필터 기능** 추가

**search\_fields = ['question\_text']** # question\_text(설문조사 주제)를 기준으로 **검색 기능** 추가

...

## 16. CSS 디자인 적용하기 & 템플릿 extends 활용하여 전체 페이지 통일시키기

### 16-1) style.css 파일 만들기

- 1. polls 폴더 안에 static 폴더 만들기
- 2. static 폴더 안에 polls 폴더 만들기
- 3. polls 폴더 안에 style.css 만들고 작성 시작

-> Atom에서 **polls** 폴더 우클릭 후 **New file** 클릭한 다음

**pollsWstaticWpollsWstyle.css** 입력 시 한번에 위 작업을 진행할 수 있음

(아래 CSS 파일의 내용은 구글드라이브에 올려두었습니다. 복사해서 붙여넣으세요.)

```
body {  
  
    background-image: url("images/main_bg.jpg");  
  
    background-size: cover;  
  
    font-family: 'Do Hyeon', "Noto Sans KR", "Titillium Web", "Helvetica", sans-serif;  
  
}  
  
a {  
  
    color: black;  
  
    text-decoration: none;  
  
    font-weight: 900;  
  
    font-size: 3em;  
  
    line-height: 2em;  
  
}
```

```
h2, li {  
    font-weight: 600;  
}
```

```
label {  
    font-weight: 600;  
    font-size: 1.5em;  
}
```

```
input[type=radio] {  
    margin-top: 2%;  
    margin-bottom: 2%;  
}
```

```
input[type=submit]{  
    padding: 0.5% !important;  
    font-size: 2em !important;  
    line-height: 1em;  
    width: 100%;  
}
```

```
.navbar-default {  
    margin-top: 3%;  
}
```

```
.navbar-brand{  
    font-size: 3em;  
    color: #379CBA !important;
```

```
}
```

```
.navbar-nav {  
    float: right;
```

```
}
```

```
.navbar-nav > li > a {  
    font-size: 2em;
```

```
}
```

```
.footer {  
    z-index: 999;  
    position: fixed;  
    bottom: 0;  
    width: 100%;  
    margin-top: 350px;  
    text-align: center;  
    height: 40px;  
    background-color: #f5f5f5;  
    font-size: 2em;  
}
```

16-2) background-image로 적용될 **main\_bg.jpg** 파일 위치시키기

- C:\work\_django\django\_mld\Wsite\_2\Wpolls\Wstatic\Wpolls 폴더 안에 **images** 폴더 만들기

- **main\_bg.jpg** 파일 복사해서 붙여넣기 (@ ~\Wsite\_2\Wpolls\Wstatic\Wpolls\Wimages)

16-3) 만든 **style.css**를 **html 파일에 적용**하기

: polls\templates\polls\index.html 수정하기 (최상단에 아래 2줄 작성)

```
{% load static %}
```

```
<link rel="stylesheet" type="text/css" href="{% static 'polls/style.css' %}" />
```

...

16-4) **Static files 반영**하기 (Gathering static files)

Ctrl + C 로 서버 종료

```
python manage.py collectstatic
```

```
python manage.py runserver
```

-> 브라우저에서 **127.0.0.1:8000/polls** 접속하여 결과 확인

-> 개발자도구를 열고 body 태그에 적용된 background-image 속성의 이미지 파일 링크에 마우스를 호버링하면 "http://127.0.0.1:8000/static/polls/images/main\_bg.jpg"라는 링크를 확인할 수 있음 (collectstatic 명령어 실행의 결과로 모든 static 파일들이 모아진 polls 앱 폴더 바깥의 static 폴더에 해당)

16-5) 템플릿 extends 적용하기 - (1) : **settings.py 수정**하기

: C:\work\_django\django\_mdl\site\_2\site\_2 -> **settings.py** 파일 수정

\* 'polls' 앱이 'django.contrib.admin' 앱보다 위에 위치해있어야 **polls 앱의 css 디자인이 관리자 페이지의 기존 default css 디자인보다 우선시되어 적용**될 수 있습니다.

...

```
INSTALLED_APPS = [
```

```
    'polls', # 가장 아래에 있던 polls 앱을 가장 위로 끌어올립니다.
```

```
    'django.contrib.admin',
```

...

```

]

...

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [os.path.join(BASE_DIR, 'templates')],

        'APP_DIRS': True,

        'OPTIONS': {

            'context_processors': [

                'django.template.context_processors.debug',

                'django.template.context_processors.request',

                'django.contrib.auth.context_processors.auth',

                'django.contrib.messages.context_processors.messages',

            ],

        },

    },

]

...

```

16-6) 템플릿 extends 적용하기 - (2) **base.html** 파일 생성하기

: polls\templates\polls 폴더에 **base.html** 파일 만들고 아래 내용 작성 (구글드라이브에 올려두었습니다.)

\* 다른 html 파일들에서 공통적으로 활용되는 코드를 하나의 html 파일로 만들어두고 재사용할 수 있습니다.

\* 아래 코드 중 {% block content %} ~ {% endblock %} 을 제외한 나머지 영역을 템플릿(.html)마다 자동으로 적용

\* 추후 {% block content %} ~ {% endblock %} 사이에 다른 템플릿(.html)의 고유한 코드들이 위치하게 됨

```
<!DOCTYPE html>
```

```
<html lang="ko">
```



<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Django Polls (version 1.0)</title>

<link href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">

<link href="//fonts.googleapis.com/css?family=Do+Hyeon|Noto+Sans&display=swap" rel="stylesheet">

</head>

<body>

<div class="container">

<nav class="navbar navbar-default">

<div class="container-fluid">

<div class="navbar-header">

<span class="navbar-brand">Django Polls (version 1.0)</span>

</div>

<ul class="nav navbar-nav">

<li><a href = "{% url 'polls:index' %}">Voting</a></li>

<li class="navbar-right"><a href = "{% url 'admin:index' %}">Administrator</a></li>

</ul>

</div>

</nav>

<div>

{% block content %}

{% endblock %}

</div>

</div>

<footer class="footer">

<div class="container">

<p class="text-muted">Copyright © 2020 Replus | Django Polls | All Rights Reserved</p>

```
</div>

</footer>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>

<script src="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</body>

</html>
```

16-7) 템플릿 extends 적용하기 - (3) **load static (for style.css)** 옮겨주기 (index.html -> **base.html**)

: 강의안 PDF 파일의 슬라이드 **109 page**의 스크린샷을 참고해 **style.css** 파일과 관련된 코드를 이동시켜주세요.

- 기존 **index.html** 에서 **load static** 관련 2줄 잘라내기

- **base.html** 에 잘라낸 2줄 붙여넣기

16-8) 템플릿 extends 적용하기 - (4) 기존의 템플릿 파일들에 **base.html** 파일을 기반으로 extends 적용

: **index.html, detail.html, result.html** 모두에 대해 아래와 같이 변경해줍니다. (강의안 슬라이드 110~112p 참고)

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

```
{% block content %}
```

**index.html, detail.html, result.html** 파일에 있는 기존 **html** 코드들의 위치

```
{% endblock %}
```

16-9) 템플릿 extends 적용하기 - (5) **결과 확인**하기 (강의안 슬라이드 113~115p 참고)

: 브라우저에서 **127.0.0.1:8000/polls** 접속하여 기능들을 실행해 index/detail/result 페이지 각각 적용 확인

## 17. 관리자 페이지 CSS 디자인 적용하기

17-1) 관리자 페이지의 디자인 수정하기 - (1) : **base\_site.html** 파일 생성하기 (파일 이름을 지켜주셔야 합니다.)

admin page는 최초 Project 생성 시 보게 되는 기본 디폴트 관리자 페이지를 위한 html 파일이 존재합니다.

해당 html 파일은 "**base.html**"이며, 아래 경로에 존재합니다. (장고 라이브러리 폴더 내에 위치해 있습니다.)

C:\work\_django\django\_mld\django\_env\Lib\site-packages\

django\contrib\admin\templates\admin\base.html

강의안 슬라이드 116page에서 위 파일의 모습을 확인할 수 있습니다.

위 경로의 base.html 파일을 extends하고 바꾸고 싶은 부분들만 **block & endblock**으로 삽입하여 바꾸는 방식으로 관리자 페이지의 디자인에 변화를 줄 수 있습니다.

Atom에서 **templates** 폴더 우클릭 후 **New file** 클릭한 다음

**polls\templates\admin\base\_site.html** 입력

-> templates 폴더 내에 admin 폴더를 만들고 그 안에 base\_site.html 생성

(아래 내용은 구글드라이브에 올려두었습니다.)

```
{% extends "admin/base.html" %}
```

```
{% load static %}
```

```
{% block title %}
```

Django Polls 관리자 페이지

```
{% endblock %}
```

```
{% block extrastyle %}
```

```
<link href="//fonts.googleapis.com/css?family=Do+Hyeon|Noto+Sans&display=swap" rel="stylesheet">
```

```
<link rel="stylesheet" type="text/css" href="{% static 'polls/admin_custom.css' %}" />
```

```
{% endblock %}
```

```
{% block branding %}
```

```
<h1 id="site-name"><a href="{% url 'admin:index' %}">Django Polls 관리자 페이지 (ver 1.0)</a></h1>
```

```
{% endblock %}
```

```
{% block nav-global %}{% endblock %}
```

17-2) 관리자 페이지의 디자인 수정하기 - (2) : **admin\_custom.css** 파일 생성하기

: polls\static\polls 폴더 안에 **admin\_custom.css** 파일을 만듭니다. (style.css와 같은 폴더입니다.)

(아래 내용은 구글드라이브에 올려두었습니다.)

```
body {
```

```
    background-image: url("images/main_bg.jpg");
```

```
    background-size: cover;
```

```
    font-family: 'Do Hyeon', "Noto Sans KR", "Titillium Web", "Helvetica", sans-serif;
```

```
}
```

```
h1{
```

```
    color: black;
```

```
    font-size: 2em;
```

```
}
```

```
a {  
    font-family: 'Do Hyeon', sans-serif;  
    font-weight: 400;  
}
```

```
th {  
    font-size: 1.5em !important;  
}
```

```
th a {  
    color: black !important;  
  
}
```

```
caption, .module h2 {  
    background-color: black !important;  
    color: white !important;  
}
```

```
.breadcrumbs {  
    background-color: steelblue !important;  
}
```

```
.object-tools > li > .addlink, .historylink{  
    background-color: black !important;  
}
```

```
.field-__str__ {
```

```
font-size: 1.5em !important;

}

.module {

background: None !important;

}

#header {

background-color: black;

}

#user-tools {

font-size: 1.5em;

}
```

17-3) 관리자 페이지의 디자인 수정하기 - (3) : **collectstatic** 실행 & 결과 확인

**Ctrl + C** 로 서버 종료

python manage.py **collectstatic**

python manage.py **runserver**

-> 브라우저에서 [127.0.0.1:8000/admin/](http://127.0.0.1:8000/admin/) 접속하여 결과 확인