Django, 나도 사용할 수 있다!

장고 공식 홈페이지

Django의 개요

Django는 파이썬으로 작성된 **오픈소스 웹 어플리케이션 프레임워크**입니다.
일반적으로 웹 브라우저 또는 웹 서버는 데이터베이스에서의 데이터 연결 및 판독을 하지 못합니다.
이러한 것을 해소하기 위해서는 웹과 데이터베이스 미들웨어를 사용하면 되지만, 현재 우리는 더욱 똑똑한 오픈 소스가 있습니다.
장고의 주된 목표는 **고도의 데이터베이스 기반 웹사이트 작성에 있어서 수고를 더는 것** 입니다.

목 차

1. 시작하기에 앞서...

Django는 **파이썬**으로 사용되는 오픈소스입니다.
따라서 여러분들이 이 메뉴얼을 보고 학습을 하기전에 파이썬에 대한 지식을 모르시는 분들은 공부를 하고 읽으시면 이해하시는데 훨씬 수월할 것 입니다.

2. Django 시작하기

Django project 를 구성하는 코드를 자동 생성해야 하는데, 이 과정에서 데이터베이스 설정, Django 위한 옵션들, 어플리케이션을 위한 설정들과 같은 Django 인스턴스를 구성하는 수많은 설정들이 생성되기 때문입니다.
우선 커맨드라인에서 코드를 저장할 디렉토리로 이동한 후 다음 명령어를 실행합니다.

```
$ django-admin startproject mysite
```

Python 이나 Django 에서 사용중인 이름은 피해야 합니다.
특히, django (Django 그 자체와 충돌이 일어남, 대소문자를 구별하지 않음) 나, test (Python 패키지의 이름중 하나) 같은 이름은 피해야 한다는 의미입니다.

최초의 소스 트리

```
mysite/
manage.py
mysite/
__init__.py
settings.py
urls.py
wsgi.py
```

- mysite/ 디렉토리 바깥의 디렉토리는 단순히 프로젝트를 담는 공간입니다. 이 이름은 Django 와 아무 상관이 없으니, 원하는 이름으로 변경하셔도 됩니다.

- `manage.py` : Django 프로젝트와 다양한 방법으로 상호작용 하는 커맨드라인의 유틸리티 입니다.
- mysite/ 디렉토리 내부에는 project 를 위한 실제 Python 패키지들이 저장됩니다. 이 디렉토리 내의 이름을 이용하여, (mysite.urls 와 같은 식으로) project 어디서나 Python 패키지들을 import 할 수 있습니다.
- mysite/ `__init__.py` : Python 으로 하여금 이 디렉토리를 패키지 처럼 다루라고 알려주는 용도의 단순한 빈 파일입니다.
- mysite/ `settings.py` : 현재 Django project 의 환경/구성을 저장합니다. Django settings 에서 환경 설정이 어떻게 동작하는지 확인할 수 있습니다.
- mysite/ `urls.py` : 현재 Django project 의 URL 선언을 저장합니다. Django 로 작성된 사이트의 "목차" 라고 할 수 있습니다.
- mysite/ `wsgi.py` : 현재 project 를 서비스 하기 위한 WSGI 호환 웹 서버의 진입점 입니다.

Django project 가 제대로 동작하는지 확인해 봅시다.
mysite 디렉토리로 이동하고, 다음 명령어를 실행하세요.

```
$ python manage.py runserver
```

< 제대로 실행 됐을때 커맨드라인의 출력 >

```
Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied.
Run 'python manage.py migrate' to apply them.

February 14, 2017 - 15:50:53  # 현재 시간
Django version 1.10, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

서버를 동작을 했으니 웹 브라우져에서 http://127.0.0.1:8000/ 를 통해 접속을 하면
"Welcome to Django"라는 텍스트가 띄워진 페이지를 볼 수 있습니다.

기본적으로 runserver 명령으로 내부 IP의 8000번 포트로 개발 서버를 띄우지만 포트를 변경 하고 싶으면 다음 명령어를 입력합니다.(포트번호를 8080으로 바꾼다 가정)

```
$ python manage.py runserver 8080
```

그리고 만약 서버 IP를 바꾸고 싶다면, 포트와 함께 전달해주면 됩니다.
네트워크 상의 다른 컴퓨터에게 내 작업물을 보여 줄때 유용합니다. 다음 명령어를 입력해 보세요.

```
$ python manage.py runserver 0.0.0.0:8000
```

이제 작업을 시작하기 위한 환경(project)이 설치되었습니다.
Django는 app의 기본 디렉토리 구조를 자동으로 생성할수 있는 도구를 제공합니다.

그렇다면 project와 app의 차이가 무엇일까요?
app은 특정한 기능을 수행하는 웹 어플리케이션을 말합니다.
proejct는 이런 특정 웹사이트를 위한 app들과 설정들을 합한것입니다.
다시 말하면 project는 여러 개의 app을 포함 할수 있습니다.

그렇다면 이제 app을 생성해 볼건데 예제로 간단한 설문조사 app을 만들어 봅시다. app을 생성하기 위해서는 `manage.py` 가 존재하는 디렉토리에서 다음 명령어를 입력합니다.

```
$ python manage.py startapp polls
```

polls라는 디렉토리가 생겼습니다.

최초의 소스 트리

```
polls/
__init__.py
admin.py
```

```
apps.py
migrations/
__init__.py
models.py
tests.py
views.py
```

첫 번째 뷰를 작성해볼까요.

경로 : polls/view.py

다음 코드를 작성해봅시다.

```
from django.http import HttpResponse


def index(request):
return HttpResponse("Hello, world. You're at the polls index.")
```

Django에서 *가장 간단한 형태의 view*입니다. view를 호출하려면 이와 연결된 URL이 있어야 하는데, 이를 URLconf가 사용되는데 polls 디렉토리에서 URLconf를 생성하려면, `urls.py` 라는 파일을 생성해야 합니다.

경로 : polls/urls.py

```
from django.conf.urls import url

from . import views

urlpatterns = [
url(r'^$', views.index, name='index'),
]
```

다음단계에서는 project 최상단의 URLconf에서 polls.urls 모듈을 바라보게 설정합니다.
mysite/urls.py 파일을 열고 django.conf.urls.include 를 추가한 뒤, 다음과 같이 코드를 추가해주세요.

경로 : mysite/urls.py

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
url(r'^polls/', include('polls.urls')),
url(r'^admin/', admin.site.urls),
]
```

include() 함수는 URLconf를 참조할 수 있도록 도와줍니다.
위의 코드는 쉽게 말하면 project안에 app의 urls.py에 접근할 수 있도록 도와주는 역할을 합니다.

---

3. 데이터베이스 설정

데이터베이스를 설정해봅시다.
Django의 기본적인 SQLite로 사용하도록 구성되어 있습니다. 데이터베이스를 처음 경험한다면 그냥 그대로 사용하시고 나는 다른 데이터베이스를 사용하고 싶으시다면 별도로 DBMS를 설치하시고 아래와 같이 코드를 작성해주시면 됩니다.

```
'django.db.backends.sqlite3',       SQLite
'django.db.backends.postgresql',    PostgreSQL
'django.db.backends.mysql',         MySQL
'django.db.backends.oracle'.        Oracle
```

경로 : mysite/settings.py

기본 코드

```
DATABASES = {
'default': {
'ENGINE': 'django.db.backends.sqlite3',
'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
}
}
```

DBMS를 PostgreSQL로 바꾸고 싶다면 아래와 같이 코드를 작성해주세요.

```
DATABASES = {
'default': {
'ENGINE': 'django.db.backends.postgresql',
'NAME': '데이터베이스의 이름',
'USER': '데이터베이스 계정',
'PASSWORD': '계정 비밀번호',
'HOST': '127.0.0.1',
'PORT': '',
}
}
```

다음으로 Django에서 기본적으로 지원해주는 다음 app들을 지원해 줍니다.

```
INSTALLED_APPS = [
'django.contrib.admin', 관리자 사이트
'django.contrib.auth',  인증 시스템
'django.contrib.contenttypes', 컨텐츠 타입을 위한 프레임 워크
'django.contrib.sessions',  세션 프레임워크
'django.contrib.messages',  메세징 프레임워크
'django.contrib.staticfiles',  정적 파일을 관리하는 프레임 워크
]
```

위의 기본적으로 지원해주는 app들은 기본적으로 하나 이상의 데이터베이스 테이블을 사용합니다.
그래서 테이블을 미리 만들어 주어야 합니다.

```
$ python manage.py migrate
```

migrate 명령은 INSTALLED_APPS의 설정을 탐색하여, DBMS 설정과 app에 사용되는 데이터베이스 테이블을 생성합니다.
어떤 내용이 생성됐는지 확인하고 싶다면 데이터 베이스 클라이언트로 접속하여 확인 할 수 있습니다.

만약 기본적으로 제공되는 app들 중 사용하기 싫은 app이 있으시다면 주석처리 해주시거나 삭제하시고,
migrate명령을 해주시면 삭제된 app을 제외하고 migrate명령이 실행 될 겁니다.

---

4. 모델 만들기

이제 모델을 만들어 봅시다. Django는 모델을 한곳에서 관리할 수 있도록 지원해줍니다.

단순한 설문조사(poll) 앱만든다 하고, Question 과 Choice 라는 두개의 모델을 만들어 보겠습니다.
Question 은 질문(question) 과 발행일(publication date) 을 위한 두개의 필드를 가집니다.
Choice 는 선택지(choice) 와 표(vote) 계산을 위한 두개의 필드를 가지고 각 Choice 모델은 Question 모델과 연관(associated) 됩니다.

경로 : polls/models.py

```
from django.db import models

class Question(models.Model):
question_text = models.CharField(max_length=200)
pub_date = models.DateTimeField('date published')


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

각 모델은 django.db.models.Model 이라는 클래스의 서브클래스로 표현됩니다.
각 모델은 여러 개의 클래스 변수를 가지고 있으면, 각각의 클래스 변수들은 모델의 데이터베이스 필드를 나타냅니다.

데이터베이스의 필드들은 Field 클래스의 인스턴스로 표현됩니다.
각 옵션들을 부여 할수 있는데 이것을 이용하여 최대길이(max_length) 지정, default로 기본값 지정, 첫번째 인수를 전달하여 사람이 이해하기 쉬운 이름을 지정 할 수도 있습니다.

ForeignKey 를 사용한 관계설정에 대해 설명하겠습니다.
이 예제에서는 각각의 Choice 가 하나의 Question 에 관계된다는 것을 Django 에게 알려줍니다. Django 는 다-대-일(many-to-one), 다-대-다(many-to-many), 일-대-

일(one-to-one) 과 같은 모든 일반 데이터베이스의 관계들를 지원합니다.
그리고 이 필드의 이름들은 데이터 베이스에서 속성명으로 사용됩니다.

- CharField : 문자 필드
- DateTimeField : 날짜시간 필드

app 을 현재의 project 에 포함시키기 위해서는, app 의 구성 클래스에 대한 참조를 INSTALLED_APPS 설정에 추가시켜야 하므로 다음과 같이 코드를 작성해 주세요.

경로 : mysite/settings.py

```
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',아
'django.contrib.staticfiles',
'polls',   # 이부분의 app의 이름을 추가하여 준다.
]
```

그후 다음 명령어를 실행해 주세요

```
$ python manage.py makemigrations polls
```

그러면 명령창에 이렇게 띄워질 것입니다.

```
Migrations for 'polls':
polls/migrations/0001_initial.py:
- Create model Choice
- Create model Question
- Add field question to choice
```

makemigrations 을 실행시킴으로서, 당신이 모델을 변경시킨 사실과(이 경우에는 새로운 모델을 만들었음) 이 변경사항을 migration 으로 저장시키고 싶다는 것을 Django 에게 알려줍니다.
migrations는 Django가 모델의 변경사항을 저장하는 방법으로 파일로 존재합니다.
polls/migrations/0001_initial.py 파일을 보면 새롭게 저장된 모델에 대한 migrations를 볼 수 있습니다. 파일로 볼 수 있고 명령어를 치지 않고 그냥 그 파일에서 수동으로 수정할 수도 있습니다.

이제 다음 명령으로 데이터베이스에 모델과 관련된 테이블을 생성해봅시다.

```
$ python manage.py migrate
```

migrate는 아직 적용되지 않은 migration들을 실행합니다. migration은 매우 기능이 강력하여 project를 개발할때 직접 데이터베이스에 손대지 않고도 모델을 변경하게 해줍니다.

---

5. 관리자 생성하기

기본적으로 Django에서는 관리자 사이트를 활성화 되어있습니다.
그러므로 일단 관리자 사이트를 사용할 수 있는 사용자를 생성해 봅시다. 다음 명령어를 실행 합니다.

```
$ python manage.py createsuperuser
```

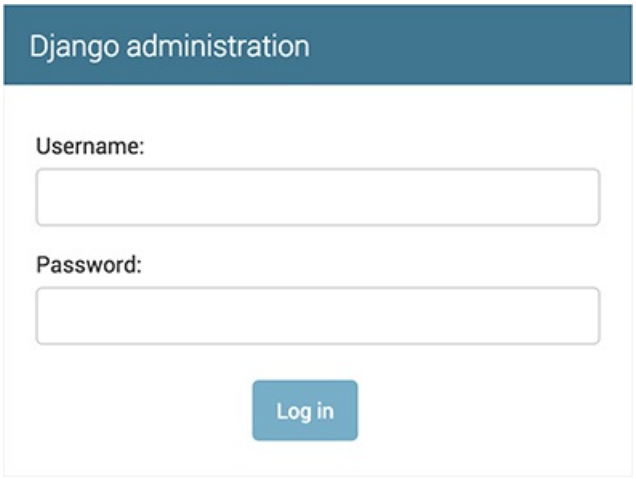
```
Username: "유저 ID"
Email address: "Email 주소"
Password: "비밀번호"
```

커맨드 창 위에서 위와 같이 작성해 줍니다.
그리고 서버를 켜기 위해 다음 명령을 실행합니다.

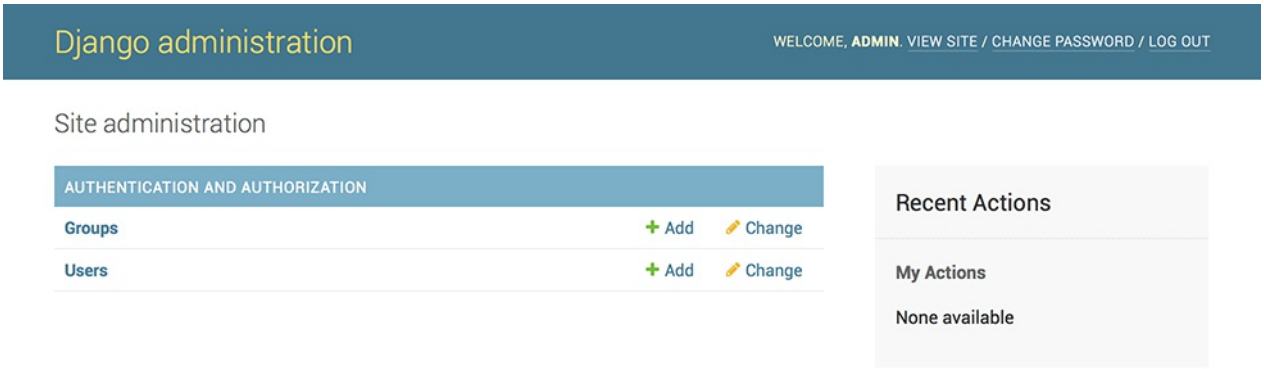
```
$ python manage.py runserver
```

그리고 웹 브라우져 주소창에 http://127.0.0.1:8000/admin/ 또는 localhost:8000/(포트를 변경 했다면 변경한 포트 입력)으로 들어갑니다.

그러면 로그인 화면이 보입니다.



위에서 생성한 계정으로 로그인 해서 들어가게 되면 다음 화면이 보일 것입니다.



polls app 관리자 페이지에 보이지 않네요.
새로 생성한 app을 관리자 페이지에서 보려면 다음 코드를 작성하세요.

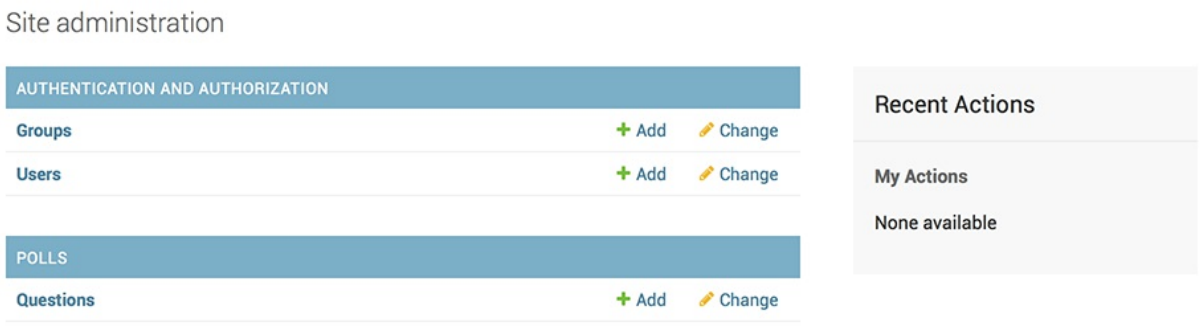
경로 : polls/admin.py

```
from django.contrib import admin

from .models import Question

admin.site.register(Question)
```

이제 관리자 페이지에서 polls app을 관리 할수 있게됩니다.



6. 뷰 추가하기

polls 어플리케이션에 관리자가 공개인터페이스인 view를 추가해보겠습니다.

view는 Django 어플리케이션이 일반적으로 특정 기능과 템플릿을 제공하는 웹페이지의 한 종류입니다.

우리가 만드는 poll 어플리케이션에서 다음과 같은 네 개의 view를 만들어 보겠습니다.

- 최근의 질문들을 표시하는 페이지
- 질문 내용과, 투표할 수 있는 서식을 표시하는 페이지
- 특정 질문에 대한 결과를 표시하는 페이지
- 특정 질문에 대해 특정 선택을 할 수 있는 투표 기능을 제공하는 페이지

Django에서는, 웹페이지와 기타 내용들이 view에 의해 제공됩니다.
각 view는 간단한 Python함수를 사용하여 작성됩니다.
Django는 요청받은 URL에 따라 view를 선택합니다.

그럼 view를 작성해봅시다.

경로 : polls/views.py

```
def detail(request, question_id):
return HttpResponse("You're looking at question %s." % question_id)

def results(request, question_id):
response = "You're looking at the results of question %s."
return HttpResponse(response % question_id)

def vote(request, question_id):
return HttpResponse("You're voting on question %s." % question_id)
```

`urls.py` 에 새로 작성된 view들을 연결하기 위해 다음과 같이 url()함수를 추가합니다.

경로 : polls/urls.py

```
from django.conf.urls import url
from . import views

urlpatterns = [
# ex: /polls/
url(r'^$', views.index, name='index'),
# ex: /polls/5/
url(r'^(?P<question_id>[0-9]+)/$', views.detail, name='detail'),
# ex: /polls/5/results/
url(r'^(?P<question_id>[0-9]+)/results/$', views.results, name='results'),
# ex: /polls/5/vote/
url(r'^(?P<question_id>[0-9]+)/vote/$', views.vote, name='vote'),
]
```

이제 웹 브라우저 주소창에 localhost:8000/polls/34를 입력하게 되면 detail()함수를 호출하여 url에 입력된 ID(34)를 출력할 것입니다.

view 는 HttpResponse 객체를 반환하거나 Http404같은 에러를 발생시킵니다. 또 데이터베이스의 레코드를 읽어 올수도 있습니다. 이번에는 데이터베이스를 다루는 방법을 해봅시다.
아래와 같이 코드를 작성해 주세요.

경로 : polls/views.py

```
from django.http import HttpResponse

from .models import Question


def index(request):
latest_question_list = Question.objects.order_by('-pub_date')[:5]
output = ', '.join([q.question_text for q in latest_question_list])
return HttpResponse(output)
```

그런데 이 뷰에는 문제가 있습니다. 바로 **디자인에 되어있지 않다는 점**입니다.
페이지를 디자인하고 싶다면 Python코드로부터 디자인을 독립시키도록 하는 Django의 템플릿 시스템을 사용할 것입니다.

일단 polls 디렉토리에 templates 라는 디렉토리를 만듭니다.
Django는 이 디렉토리 안에서 템플릿을 찾게 될 것입니다.
project의 템플릿 설정에는 Django가 어떻게 템플릿을 불러오고 렌더링 할 것인지 를 서술합니다.
기본적으로 Django는 templates 디렉토리를 탐색하게 됩니다.
templates 디렉토리 안에 app과 이름이 같은 디렉토리를 만들어 봅시다. 그리고 그안에 html파일을 만들어 아래 코드를 작성합니다.

경로 : polls/templates/polls/index.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 %}
```

Django 에서는 {%%}에는 조건문을 {{}}에는 변수를 출력을 할수 있도록 도와줍니다.
이제 이 템플릿을 페이지로 띄우기 위해 polls/views.py함수를 수정해봅시다.
아래 코드와 같이 수정해주세요.

경로 : polls/views.py

```
from django.http import HttpResponse
from django.template import loader

from .models import Question


def index(request):
latest_question_list = Question.objects.order_by('-pub_date')[:5]
template = loader.get_template('polls/index.html')
context = {
'latest_question_list': latest_question_list,
}
return HttpResponse(template.render(context, request))
```

이 코드는 polls/index.html을 불러온 후, Python의 딕셔너리 형태로 context를 전달합니다.
하지만 더 간단히 표현할 수 있는 방법이 있습니다.
다음과 같이 코드를 수정해 주세요.

경로 : polls/views.py

```
from django.shortcuts import render

from .models import Question


def index(request):
latest_question_list = Question.objects.order_by('-pub_date')[:5]
context = {'latest_question_list': latest_question_list}
return render(request, 'polls/index.html', context)
```

모든 뷰에서 이 방식을 사용한다면 굳이 loader 와 HttpResponse 를 import 하지 않아도 됩니다.
render() 함수는 첫번째 인수에서 request 객체를 받고 두번째 인수로 template의 이름을 받고 세번째 인수로 context 객체를 선택적으로 인수를 받습니다. context 는 템플릿에 HttpResponse 객체로 넘어갑니다.

---

7. template 사용하기

위의 코드에서 detail()함수로 넘긴 context 변수 question이 polls/detail.html에서 어떻게 보여지는지 아래 코드를 작성해봅시다.

경로 : polls/templates/polls/detail.html

```
<h1>{{ question.question_text }}</h1>
<ul>
{% for choice in question.choice_set.all %}
<li>{{ choice.choice_text }}</li>
{% endfor %}
</ul>
```

template 시스템은 변수의 속성에 접근하기 위해 점-탐색(dot-lookup) 문법을 사용합니다.
{{ question.question_text }}을 보면, Django는 먼저 questions 객체에 대해 dictionary로 탐색한 후 실패를 하면 속성 값을 탐색하게 됩니다. {% for %} 문에서 메소드 호출이 일어나고 Python코드가 동작합니다.

template에서 하드코딩된 URL 제거하기

polls/index.html template에 링크를 적으면, 다음과 같은 하드코딩이 나오게 됩니다.

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

이러한 방법은 수많은 template 을 가진 project의 URL을 바꾸는게 어려운 일이 되버립니다.
그러므로 우리는 polls.urls 모듈의 url()함수를 통하여 이름을 정의했습니다.
그것을 이용하여 다시 코드를 작성해봅시다.

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

이렇게 코드르 작성하게 되면 url을 바꾸게 될 때 template에서 코드 수정을 하는게 아니라 polls.urls.py에서 수정하시면 됩니다.

한가지 더 맨 처음 말했듯이 하나의 project에는 여러 개의 app이 들어가지만 url의 이름을 같게 만들어야 할 상황이 있을수도 있습니다.
그럴때에는 URLconf에 namespace를 추가해주는 것입니다. 아래 코드를 작성해 주세요.

경로 : polls/urls.py

```
from django.conf.urls import url

from . import views

app_name = 'polls'
urlpatterns = [
url(r'^$', views.index, name='index'),
url(r'^(?P<question_id>[0-9]+)/$', views.detail, name='detail'),
url(r'^(?P<question_id>[0-9]+)/results/$', views.results, name='results'),
url(r'^(?P<question_id>[0-9]+)/vote/$', views.vote, name='vote'),
]
```

이렇게 하면 기존의 template의 내용을

경로 : polls/templates/polls/index.html

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

아래와 같이 수정해주시면 됩니다.

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

그럼 간단한 폼을 만들어 봅시다.
polls/detail.html 폼을 아래 처럼 HTML 요소인

태그를 추가 해봅시다. 아래 코드를 수정해주세요.

경로 : polls/templates/polls/detail.html

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

{% 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" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />
<label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br />
{% endfor %}
<input type="submit" value="Vote" />
</form>
```

위의 템플릿은 각 질문 선택 항목에 대한 라디오 버튼을 표시합니다.
각 라디오 버튼의 value 는 연관된 질문 선택 항목의 ID입니다.

각 라디오 버튼의 name 은 "choice"입니다.
즉, 누군가가 라디오 버튼 중 하나를 선택하여 폼을 제출하면, POST 데이터 인 choice=#을 보낼 것입니다. 여기서 #은 선택한 항목의 ID입니다. 이것은 HTML 폼의 기본 개념입니다.

폼의 action을 {% url 'polls:vote' question.id %}로 설정하고, method="post" 로 설정하였습니다.
이 폼을 전송하는 행위는 서버측 자료를 변경할 것이므로, method="post" (method="get" 와 반대로) 를 사용하는 것은 매우 중요합니다.
서버 측 자료를 변경하는 폼을 작성할 때마다, method="post" 를 사용하세요.
(이 팁은 Django에만 국한되지 않음 / 웹 개발 시의 권장사항)

forloop.counter 는 for 태그가 반복을 한 횟수를 나타냅니다.

우리는 POST 폼(자료를 수정하는 효과를 가진)을 만들고 있으므로, 사이트 간 요청 위조 (Cross Site Request Forgeries)에 대해 고민해야합니다.
고맙게도, Django는 사이트 간 요청 위조(CSRF)에 대항하기위한 사용하기 쉬운 시스템을 가지고 있기 때문에, 너무 심각하게 고민할 필요가 없습니다. 간단히 말하면, 내부 URL들을 향하는 모든 POST 폼에 템플릿 태그 {% csrf_token %}를 사용하면됩니다.

그럼 이제 부터는 입력받은 데이터를 처리하고 그 데이터를 이용하는 view 를 작성해보겠습니다.
아래 코드를 작성해 주십시오.

경로 : polls/views.py

```
from django.shortcuts import render
from django.http import HttpResponseRedirect, HttpResponse
from django.urls import reverse

from .models import Choice, Question

# ...

def vote(request, question_id):
question = get_object_or_404(Question, pk=question_id)
try:
selected_choice = question.choice_set.get(pk=request.POST['choice'])
except (KeyError, Choice.DoesNotExist):
return render(request, 'polls/detail.html', {
'question': question,
'error_message': "You didn't select a choice.",
})
else:
selected_choice.votes += 1
selected_choice.save()
return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))
```

request.POST 는 키로 전송된 자료에 접근할 수 있도록 해주는 사전과 같은 객체입니다.
이 경우, request.POST['choice'] 는 선택된 설문의 ID가 "choice"인 값을 문자열로 반환합니다. request.POST 의 값은 항상 문자열들입니다.

설문지의 수가 증가한 이후에, 코드는 일반 HttpResponse 가 아닌 HttpResponseRedirect 를 반환하고, HttpResponseRedirect 는 하나의 인수를 받습니다: 그 인수는 사용자가 재전송될 URL 입니다. (이 경우에 우리가 URL을 어떻게 구성하는지 다음 항목을 보세요).

위의 파이썬 주석이 지적했듯이, POST 데이터를 성공적으로 처리 한 후에는 항상 HttpResponseRedirect 를 반환해야합니다. 이 팁은 Django에만 국한되는것이 아닌 웹개발의 권장사항입니다.

vote() 뷰는 results 페이지를 리다이렉트합니다. 그 뷰를 작성해봅시다.

경로 : polls/views.py

```
from django.shortcuts import get_object_or_404, render

def results(request, question_id):
question = Question.obejcts.get(pk=question_id)
return render(request, 'polls/results.html', {'question': question})
```

이제 이걸을 출력할 template를 만듭니다.

경로 : polls/templates/polls/results.html

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

<ul>
{% for choice in question.choice_set.all %}
<li>{{ choice.choice_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>

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

이제 사이트에서 투표를 하면 값이 변경되는 것을 확인 할 수 있습니다.

---

8. 제네릭 뷰 사용

제네릭 뷰를 사용해 봅시다.

지금까지 만들었던 뷰들은 매우 간단한 뷰들입니다. 그리고 중복이 되는 부분이 있습니다.
이러한 뷰는 URL에서 전달 된 매개 변수에 따라 데이터베이스에서 데이터를 가져 오는 것과 템플릿을 로드하고 렌더링 된 템플릿을 리턴하는 기본 웹 개발의 일반적인 경우를 나타냅니다. Django는 이런 매우 일반적인 경우를 위해 **제너릭 뷰 시스템**이라는 지름길을 제공합니다.

제너릭 뷰는 일반적으로 나오는 패턴을 추상화하여 앱을 작성하기 위해 중복된 코드를 작성하지 않아도 됩니다.
우리는 설문조사 app을 제네릭 뷰 시스템을 사용하도록 코드를 수정할 것입니다. 그러기 위해선 몇 단계의 과정을 거쳐야 합니다.

1. URLconf를 변환하십시오.
2. 불필요한 뷰 중 일부를 삭제하십시오.
3. Django의 제너릭 뷰를 기반으로 새로운 뷰를 도입하십시오.

먼저 URLconf를 수정합니다. 다음과 같이 코드를 수정해주세요.

경로 : polls/urls.py

```
from django.conf.urls import url

from . import views

app_name = 'polls'
urlpatterns = [
url(r'^$', views.IndexView.as_view(), name='index'),
url(r'^(?P<pk>[0-9]+)/$', views.DetailView.as_view(), name='detail'),
url(r'^(?P<pk>[0-9]+)/results/$', views.ResultsView.as_view(), name='results'),
url(r'^(?P<question_id>[0-9]+)/vote/$', views.vote, name='vote'),
]
```

두 번째와 세 번째 패턴의 정규식에서 일치하는 패턴의 이름이 에서 로 변경되었습니다.

그리고 index, detail, results 뷰를 제거하고 Django 제네릭 뷰를 대신 사용하겠습니다.
다음과 같이 코드를 수정해주세요.

경로 : polls/views.py

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from django.urls import reverse
from django.views import generic

from .models import Choice, Question


class IndexView(generic.ListView):
template_name = 'polls/index.html'
context_object_name = 'latest_question_list'

def get_queryset(self):
"""Return the last five published questions."""
return Question.objects.order_by('-pub_date')[:5]


class DetailView(generic.DetailView):
model = Question
template_name = 'polls/detail.html'


class ResultsView(generic.DetailView):
model = Question
template_name = 'polls/results.html'


def vote(request, question_id):
... # 변경할 필요 없습니다.
```

DetailView 제너릭 뷰는 URL에서 캡쳐 된 기본 키 값이 "pk"라고 기대하기 때문에 question_id를 제너릭 뷰를 위해 pk로 변경합니다.

기본적으로 DetailView 제너릭 뷰는 `<app name>/<model name>_detail.html` 템플릿을 사용합니다. 우리의 경우에는 **polls/question_detail.html**템플릿을 사용할

것입니다.
template_name 속성은 Django에게 자동 생성 된 기본 템플릿 이름 대신에 특정 템플릿 이름을 사용하도록 알려주기 위해 사용됩니다. results리스트 뷰에 대해서 template_name을 지정합니다.(결과 뷰와 상세 뷰가 렌더링 될 때 서로 다른 모습을 갖도록 함)
이들이 둘다 동일한 DetailView 를 사용하고 있더라도 말이지요.

마찬가지로, ListView 제네릭 뷰는 `<app name>/<model name>_list.html` 템플릿을 기본으로 사용합니다. 이미 있는 **polls/index.html** 템플릿을 사용하기 위해 ListView 에 template_name 를 전달했습니다.

코드 수정 전 템플릿에서는 question 및 latest_question_list context 변수가 포함된 context가 값으로 넘겨와 사용되었지만 DetailView 의 경우 question 변수가 자동으로 값으로 넘어옵니다.
Django 모델을 사용하기 때문에 Django는 context의 이름을 결정할 수 있습니다.
ListView 의 경우 자동 생성 된 컨텍스트 변수는 question_list 입니다. 이것을 바꿔서 사용하려면 context_object_name 속성을 제공하고, 대신에 latest_question_list 를 사용하도록 지정하세요.