

Django

- 파이썬 웹 프로그래밍 참조

강사: 서찬웅

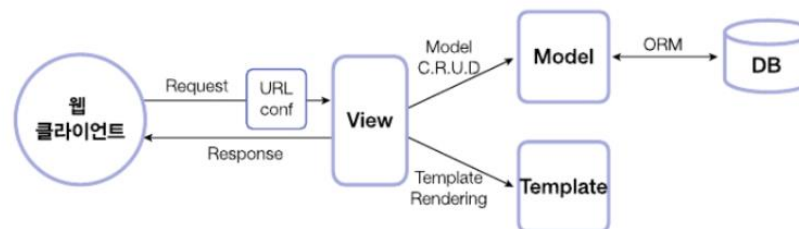
django란?

- 장고(Django)는 웹 프로그램을 개발하는 데 사용하는 파이썬 웹 프레임워크입니다.
 - 프레임워크 : 어떠한 목적을 달성하기 위해 복잡하게 얹혀있는 문제를 해결하기 위한 구조며, 소프트웨어 개발에 있어 하나의 뼈대 역할을 함
- 제공하는 기능이 풍부하고, 쉽고 빠르게 웹 개발이 가능하며 사용자도 가장 많이 존재합니다.
- 장고는 2003년 로렌스 저널-월드 신문을 만들던 웹 개발팀의 내부 프로젝트로 시작됐으며, 2005년 오픈소스 프로젝트로 공개되었습니다.
 - 신문사에서 장고를 만든 이유는 보다 빠르게 신문 기사를 업데이트할 목적으로 만들었다고 합니다.
- 특징
 - MVC 패턴 기반 MVT,
 - 화면에 해당하는 데이터의 DB를 액세스할 수 있는 Model
 - 데이터를 가져오고 변형을 할 수 있는 View
 - 화면을 담당하는 Template
 - 객체 관계 매핑
 - 객체 관계 매핑(Object-Relational Mapping)을 지원
 - ORM 기능을 사용하여 다양한 데이터베이스 시스템에 SQL 문장을 사용하지 않고도 테이블을 조작할 수 있습니다.
 - 자동으로 구성되는 관리자 화면
 - 웹 서버의 콘텐츠, 즉 데이터베이스에 대한 관리 기능을 위하여 프로젝트를 시작하는 시점에 기본 기능으로 관리자 화면을 제공
 - 우아한 URL 설계
 - 정규식을 사용하여 복잡한 URL도 표현할 수 있습니다.
 - 자체 템플릿 시스템
 - 디자인과 로직에 대한 코딩을 분리하여 독립적으로 개발 진행이 가능

참고: 개발자 홀로바티가 좋아하던 집시 재즈 기타리스트인 장고 라인하르트에서 따온 것입니다.

장고에서의 애플리케이션 개발 방식

- 웹 개발 또는 웹 서비스 개발이란 용어를 명확하게 표현하면 웹 애플리케이션 개발이라고 할 수 있습니다.
- 웹 사이트를 설계할 때 가장 먼저 해야 할 일은 프로그램이 해야 할 일을 적당한 크기로 나누어서 모듈화하는 것입니다.
 - 웹 사이트의 전체 프로그램 또는 모듈화된 단위 프로그램을 애플리케이션이라고 합니다.
 - 즉 프로그램으로 코딩할 대상을 애플리케이션이라고 부릅니다.
- 장고에서는
 - 웹 사이트에 대한 전체 프로그램을 Project라고 하고
 - 모듈화된 단위 프로그램을 애플리케이션(app)이라고 부릅니다.
- MVT 패턴
 - **M**odel (데이터), **V**iew(데이터를 처리하는 로직), **T**emplate(사용자 인터페이스), 으로 구분해서 한 요소가 다른 요소들에 영향을 주지 않도록 설계하는 방식입니다.
 - UI 디자이너는 데이터 관리나 애플리케이션 로직에 신경 쓰지 않고 화면을 개발할 수 있습니다.
 - 로직이나 데이터를 설계하는 개발자도 화면 디자인은 디자이너에게 맡기고 자신의 설계 및 개발 업무에만 집중할 수 있습니다.



Model

- 모델이란 사용될 데이터에 대한 정의를 담고 있는 장고의 클래스입니다.
- 장고는 ORM 기법을 사용하여 애플리케이션에서 사용할 데이터베이스를 클래스로 매핑해서 코딩할 수 있습니다. 즉 하나의 모델 클래스는 하나의 테이블에 매핑되고, 모델 클래스의 속성은 테이블의 컬럼에 매핑됩니다.
- ORM이란?
 - Object Relational Mapping은 쉽게 표현하면 객체와 관계형 데이터베이스를 연결해주는 역할을 합니다.
 - 직접 SQL 언어를 사용해 데이터를 요청할 필요 없이 객체를 사용해 데이터를 처리할 수 있습니다.
 - 객체를 대상으로 필요한 작업을 진행하면, ORM이 자동으로 적절한 SQL 구문이나 데이터베이스 API를 호출해서 처리합니다.

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

```
class Person(models.Model):  
    first_name = models.CharField(max_length=30)  
    last_name = models.CharField(max_length=30)
```



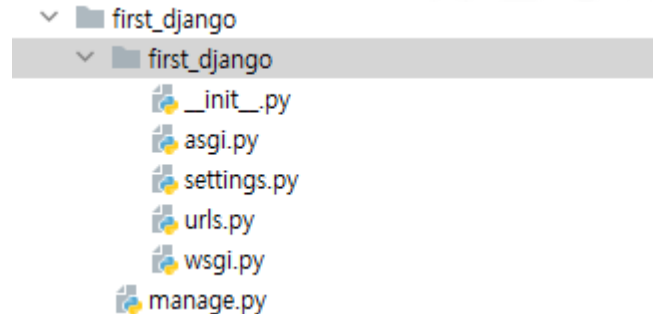
```
CREATE TABLE myapp_person (  
    "id" serial NOT NULL PRIMARY KEY,  
    "first_name" varchar(30) NOT NULL,  
    "last_name" varchar(30) NOT NULL  
);
```

- Models.py
 - Models에서 설정한 객체들이 ORM로 인해서 DB에 자동으로 매핑이 됩니다.

django 프로젝트 시작하기

- django-admin startproject [프로젝트명]

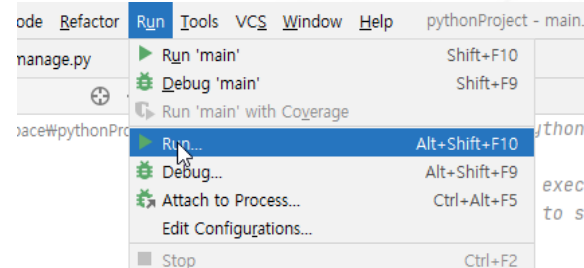
- 아래와 같이 프로젝트를 생성해보겠습니다.
- django-admin startproject first_django



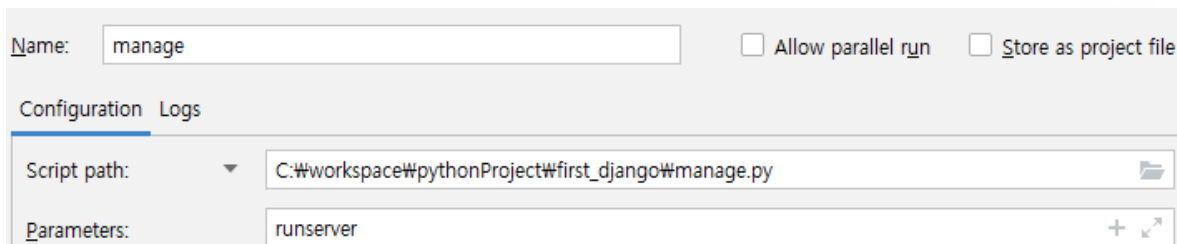
- manage.py: Django 프로젝트와 다양한 방법으로 상호작용 하는 커맨드라인의 유틸리티
- first_django/ 디렉토리 내부에는 프로젝트를 위한 실제 Python 패키지들이 저장됩니다. 이 디렉토리 내의 이름을 이용하여, (first_django.urls 와 같은 식으로) 프로젝트의 어디서나 Python 패키지들을 임포트할 수 있습니다.
- first_django/__init__.py: Python으로 하여금 이 디렉토리를 패키지처럼 다루라고 알려주는 용도의 단순한 빈 파일입니다.
- first_django/settings.py: 현재 Django 프로젝트의 환경 및 구성을 저장
- first_django/urls.py: 현재 Django project 의 URL 선언을 저장합니다. Django 로 작성된 사이트의 "목차" 라고 할 수 있습니다.
- first_django/asgi.py: (Asynchronous Server Gateway Interface) Django 3.0부터 지원하는 새로운 기능으로 비동기 웹서버 및 어플리케이션을 만들수 있도록 제공
- first_django/wsgi.py: 현재 프로젝트를 서비스하기 위한 WSGI 호환 웹 서버의 진입점입니다.

django 실행하기

- first_django라는 웹을 구동해 보겠습니다.
- terminal를 이용하는 방법
 - first_django 폴더 밑에 manage.py라는 파일이 있습니다. 해당 폴더로 이동합니다.
 - manage.py 파일이 있는 곳에서 아래 명령어를 실행합니다.
 - 예제 1) python manage.py runserver
 - django를 기본 포트를 사용해서 웹 구동
 - 예제 2) python manage.py runserver 6000
 - django를 실행할 때 6000번 포트를 사용해서 웹 구동



- pycharm에서 run를 사용해 실행하는 방법
 - 오른쪽 그림처럼 Run 메뉴 실행
 - Edit Configuration..를 선택하여 아래 처럼 설정



- Name은 사용자가 정하고 싶은 이름 설정
- Run 버튼을 클릭하여 실행
 - terminal를 사용하는 방법하고 같으나 버튼으로 편하게 사용 가능

App 만들기

- MVT 모델 코딩 순서

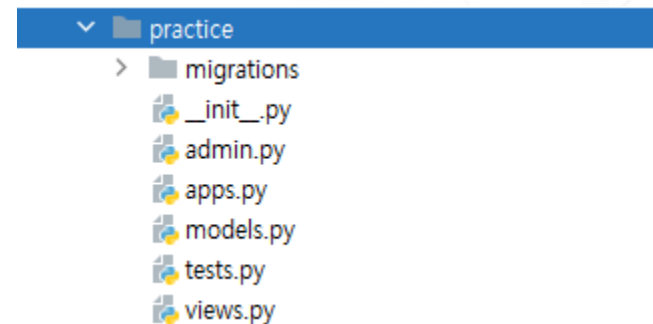
- App 뼈대 만들기
- 모델 코딩하기 : 테이블 관련 사항을 개발(models.py, admin.py 파일)
- URLconf 코딩하기 : URL 및 뷰 매핑 관계를 정의(urls.py 파일)
- 템플릿 코딩하기 : 화면 UI 개발(templates/ 디렉토리 하위의 *.html 파일들)
- 뷰 코딩하기 : 애플리케이션 로직 개발(views.py 파일)

- django에서 app이란

- 프로젝트란 개발 대상이 되는 전체 프로그램을 의미하며, 프로젝트를 몇 개의 기능 그룹으로 나누었을 때, 프로젝트 하위의 서브 프로그램을 애플리케이션이라고 합니다.
- 즉 서브 프로그램인 애플리케이션을 개발하고, 이들을 모아서 프로젝트를 개발을 완성하게 되는 것입니다.

- Practice App 만들기

- html 및 구조를 파악할 수 있는 app를 만들어 봅시다.
- `python manage.py startapp practice`



practice app 시작하기

- 수업시간에 설정한 mariadb로 설정하기

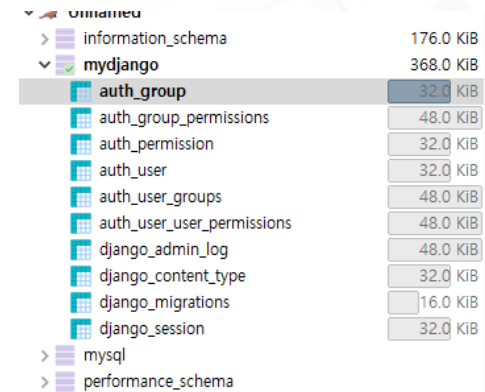
- settings.py 로 이동
- 기본으로 sqlite3를 사용하고 있지만, 수업시간에 설정한 Centos의 mariadb로 설정합니다.
- mysqldclient 패키지를 설치하여 django와 mariadb를 연동합니다.
- pip install mysqldclient

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'mydjango',           # DB명  
        'USER': 'root',              # 데이터베이스 계정  
        'PASSWORD': '123',           # 계정 비밀번호  
        'HOST': '192.168.174.100',    # 데이터베이스 주소(IP)  
        'PORT': '3306',               # 데이터베이스 포트(보통은 3306)  
    }  
}
```

mydjango라는 databases가
존재해야 합니다.

- 아래 명령어를 실행하여 데이터베이스에 변경 사항을 반영합니다.

- python manage.py migrate
- 이번 예제에서는 models를 설정하는 것이 없기 때문에 장고가 사용하는 내부 db만 설정됩니다.



information_schema	176.0 KiB
mydjango	368.0 KiB
auth_group	32.0 KiB
auth_group_permissions	48.0 KiB
auth_permission	32.0 KiB
auth_user	32.0 KiB
auth_user_groups	48.0 KiB
auth_user_user_permissions	48.0 KiB
django_admin_log	48.0 KiB
django_content_type	32.0 KiB
django_migrations	16.0 KiB
django_session	32.0 KiB
mysql	
performance_schema	

설정하기

- settings.py에 practice app 추가합니다.

```
INSTALLED_APPS = [  
    'django.contrib.admin',      # 관리자 기능 관련 기본 앱  
    'django.contrib.auth',      # 인증 처리  
    'django.contrib.contenttypes', # 모델 관리  
    'django.contrib.sessions',   # 세션 및 방문자 관리  
    'django.contrib.messages',   # 메시지 처리  
    'django.contrib.staticfiles', # 정적 파일 처리  
    'practice'                  # 우리가 처음으로 만든 app  
]
```

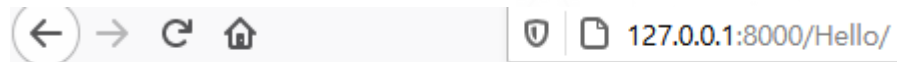
- url 설정하기

```
from practice import views  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path("Hello/", views.TellHello)  
]
```

- practice.views.py 수정하기

```
from django.http import HttpResponse  
  
def TellHello(request):  
    html = "<h1> 우리들의 첫번째 장고 App입니다. </h1>"  
    return HttpResponse(html)
```

- runserver로 실행



우리들의 첫번째 장고 App입니다.

Template 사용하기

- 이전 장표는 views.py 안에서 웹에 표시할 내용을 코딩했지만, 사실 이렇게 할 경우는 거의 없습니다.
- 수천 줄이 넘는 복잡한 애플리케이션을 이렇게 개발할 순 없습니다. 그래서 template 파일로 작성하여 개발합니다.

- app 추가하기

- python manage.py startapp html_practice

- settings.py에서 app 추가하기

- urls.py에 추가하기

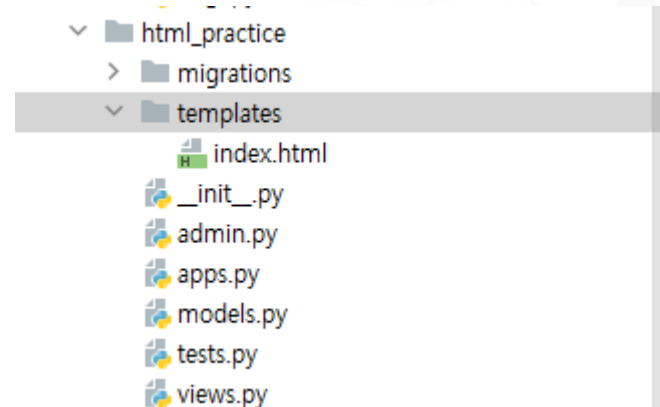
```
INSTALLED_APPS = [  
    'django.contrib.admin',    # 관리자 기능 관련 기본 앱  
    'django.contrib.auth',    # 인증 처리  
    'django.contrib.contenttypes', # 모델 관리  
    'django.contrib.sessions', # 세션 및 방문자 관리  
    'django.contrib.messages', # 메시지 처리  
    'django.contrib.staticfiles', # 정적 파일 처리  
    'practice'                # 우리가 처음으로 만든 app  
    "html_practice"           # html 연습을 위한 app  
]
```

```
from practice import views  
from html_practice import views as html_views  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path("Hello/", views.TellHello),  
    path("html/", html_views.main_view)  
]
```

- html_practice의 views.py 추가

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

- templates 폴더 생성 후에 index.html 파일 만들기



html 태그 정리

- html 태그를 정리해 봅시다.

태그	기능
<a>	하이퍼링크를 추가. href 속성을 사용하여 링크할 url을 지정. 메일 주소나 페이지 안의 다른 부분도 링크 가능
<h1>~ <h6>	문장의 제목을 생성. 숫자가 낮을수록 상위 레벨을 의미.
<p>	단락을 나타낸다. 달리 표시할 요소가 없는 경우에 사용하기를 권장.
 	줄바꿈을 나타낸다. 줄을 바꿀 문자의 끝에 를 붙인다. 이 태그는 </br>은 필요없다.
	번호가 없는 항목 쓰기의 범위를 나타낸다. 항목은 로 지정
	번호가 붙은 항목 쓰기의 범위를 나타낸다. 항목은 으로 지정

html 태그 정리 2

- 여러 개의 문장을 하나의 단위로 묶는 태그 소개
- 콘텐츠를 묶어서 그 뒤에 작성하는 CSS(Cascading Style Sheets)를 적용

태그	기능
<div>	콘텐츠의 단위를 나타낸다. 요소의 전후에는 줄바꿈이 들어간다. (이러한 요소를 '블록요소'라고 한다.)
	콘텐츠의 단위를 나타낸다. 요소의 전후에 줄바꿈이 들어가지 않는다. (이러한 요소를 '인라인 요소'라고 한다.)

- Tabel 태그

태그	기능
<table>	표를 작성한다. <table>요소로 둘러싼 부분이 표가 된다.
<tr>	표에 행을 추가. <tr> 요소로 둘러싼 부분이 행이 된다.
<th>	표에 헤더가 되는 셀을 추가한다. <th> 요소로 둘러싼 부분이 셀의 제목.
<td>	표에 셀을 추가. <td> 요소로 둘러싼 부분이 셀의 내용.
<caption>	표의 타이틀을 나타낸다.

html 태그 정리3

- id와 class
 - 속성 중에는 글로벌 속성이라는 거의 모든 요소 지정.
 - 그 중에서 가장 중요한 것은 id와 class속성이다.
 - 이러한 것은 스타일시트나 Java Script와 연결에 사용된다.
 - html에서 데이터를 수집할 때도 원하는 곳의 id와 class속성을 통해 데이터를 접근할 수 있다.
- id 속성
 - id 속성은 요소에 식별자를 지정
 - 하나의 문서 안에는 동일한 이름의 id를 지정할 수 없다.
- class 속성
 - class 속성은 요소에 클래스명을 지정
 - 하나의 문서 안에는 동일한 이름의 class를 지정할 수 있다.

두번째 프로젝트 : 설문지 app 만들어 보기

- 프로젝트 시작하기

- `django-admin startproject mysite2`

- app 생성

- mysite2 폴더 안에서 아래 명령어 실행
 - `python manage.py startapp polls`

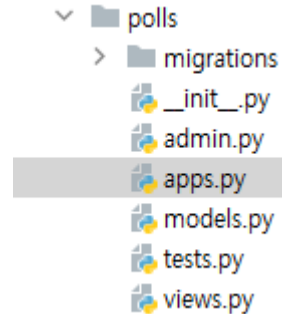
- INSTALLED_APPS에 등록하기

- settings.py 파일을 수정
- 애플리케이션을 등록할 때는 간단하게 애플리케이션의 모듈명인 'polls'만 등록해도 되지만, 애플리케이션의 설정 클래스로 등록하는 것이 더 정확한 방법입니다.
- polls 앱의 설정 클래스는 `startapp polls` 명령 시에 자동 생성된 `apps.py` 파일에 `PollsConfig`라고 정의되어 있습니다.

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'polls.apps.PollsConfig'  
]
```

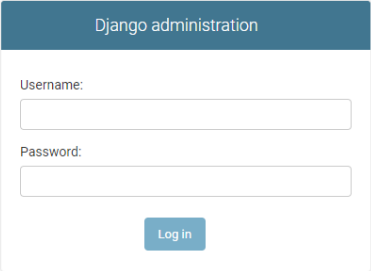
- 데이터베이스 엔진 선택하기

- 이전 예제처럼 우리가 설정한 mariadb로 설정합니다.
- db명을 **mysite2**로 설정합니다.



두번째 프로젝트 : 설문지 app 만들어 보기2

- 기본 테이블 생성
 - `python manage.py migrate`
- django admin 사이트 접속하기
 - <http://127.0.0.1:8000/admin>
- admin 계정 만들기
 - `python manage.py createsuperuser`



```
Username (leave blank to use 'howil'): gen
Email address:
Password:
Password (again):
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

Site administration

AUTHENTICATION AND AUTHORIZATION	
Groups	+ Add Change
Users	+ Add Change

두번째 프로젝트 : polls의 models.py 만들기

- ORM을 활용하여 DB에 관련된 내용을 생성하겠습니다.
- polls.models.py 설계하기

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

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

```
    def __str__(self):  
        return self.question_text
```

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

```
    def __str__(self):  
        return self.choice_text
```

테이블 컬럼명	컬럼 타입	장고의 클래스 변수	장고의 필드 클래스
id	integer	(id)	(PK는 장고에서 자동 생성해줌)
question_text	varchar(200)	question_text	models.CharField(max_length=200)
pub_date	datetime	pub_date	models.DateTimeField('date published')

컬럼명	타입	장고의 클래스 변수	장고의 필드 클래스
id	integer	(id)	(PK는 장고에서 자동 생성해줌)
choice_text	varchar(200)	choice_text	models.CharField(max_length=200)
votes	integer	votes	models.IntegerField(default=0)
question_id	integer	question	models.ForeignKey(Question)

두번째 프로젝트 : polls의 admin.py에 등록

- 앞에서 models.py에 정의한 테이블을 admin 사이트에서 보일수 있도록 등록하겠습니다.
- polls.admin.py에 등록합니다.

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

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

```
class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Question',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('question_text', models.CharField(max_length=200)),
                ('pub_date', models.DateTimeField(verbose_name='date published')),
            ],
        ),
        migrations.CreateModel(
            name='Choice',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('choice_text', models.CharField(max_length=200)),
                ('votes', models.IntegerField(default=0)),
                ('question', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='polls.question')),
            ],
        ),
    ]
```

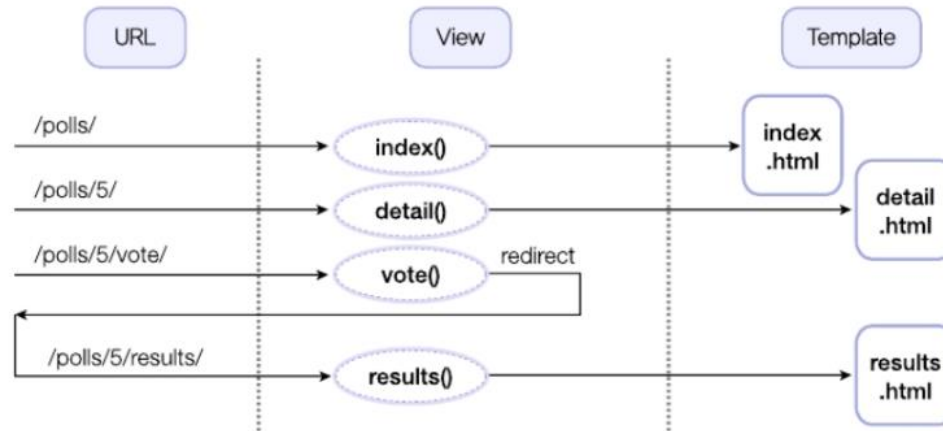
- 데이터베이스 변경사항 반영

- python manage.py makemigrations
- polls/migrations 디렉토리 하위에 마이그레이션 파일들이 생기고, 이 마이그레이션 파일들을 이용해 migrate 명령으로 데이터베이스에 테이블을 만들어줍니다.
- python manage.py migrate를 사용하여 DB에 테이블을 생성합니다.
- python manage.py sqlmigrate polls 0001 명령어를 사용하여 sql 문장을 확인할 수 있습니다.

```
CREATE TABLE `polls_question` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `question_text` varchar(200) NOT NULL, `pub_date` datetime(6) NOT NULL);
CREATE TABLE `polls_choice` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `choice_text` varchar(200) NOT NULL, `votes` integer NOT NULL, `question_id` integer NOT NULL);
ALTER TABLE `polls_choice` ADD CONSTRAINT `polls_choice_question_id_c5b4b260_fk_polls_question_id` FOREIGN KEY (`question_id`) REFERENCES `polls_question` (`id`);
```

View 및 Template 코딩

- 실제 화면에 보여질 View 및 Template를 코딩합니다.



- url과 view 매핑

URL 패턴	뷰 이름	뷰가 처리하는 내용
<code>/polls/</code>	<code>index()</code>	<code>index.html</code> 템플릿을 보여줍니다.
<code>/polls/5/</code>	<code>detail()</code>	<code>detail.html</code> 템플릿을 보여줍니다.
<code>/polls/5/vote/</code>	<code>vote()</code>	<code>detail.html</code> 에 있는 폼을 POST 방식으로 처리합니다.
<code>/polls/5/results/</code>	<code>results()</code>	<code>results.html</code> 템플릿을 보여줍니다.

urls.py 및 polls.urls.py 설정, templates 폴더 생성

- polls.urls.py를 생성하고 아래처럼 입력합니다.

```
from django.urls import path
from polls import views
```

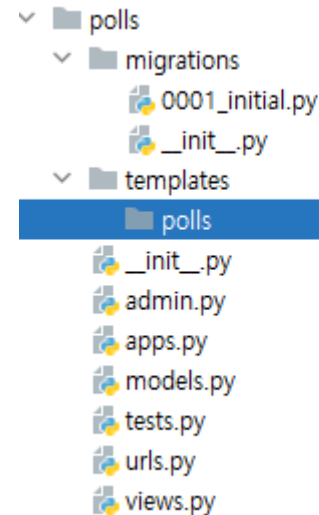
```
app_name = 'polls'
urlpatterns = [
    path("", views.index, name='index'),           # /polls/
    path("<int:question_id>/", views.detail, name='detail'), # /polls/번호/
    path("<int:question_id>/results/", views.results, name='results'), # /polls/번호/results/
    path("<int:question_id>/vote/", views.vote, name='vote'), # /polls/번호/vote/
]
```

- polls의 urls를 경로를 main의 urls.py에 등록합니다.

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

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

- polls 폴더안에 templates 폴더와 하위 polls 폴더를 생성합니다.



views.py에 코딩하기

- polls.py에 아래와 같이 코딩합니다.

```
from django.shortcuts import render
from django.shortcuts import get_object_or_404, render
from django.http import HttpResponseRedirect
from django.urls import reverse
from polls.models import Choice, Question

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

def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})

def results(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/results.html', {'question': 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,)))
```

index.html 파일

- {% if latest_question_list %}
 - latest_question_list 객체는 index() 뷰 함수에서 넘겨주는 파라미터

{% for question in latest_question_list %}

{{ question.question_text }}

{% endfor %}

- latest_question_list 객체의 내용을 순회하면서 question_text를 순서 없는 리스트로 화면에 출력
- 각 텍스트에 URL 링크를 연결 합니다.
- 해당 링크를 클릭하게 되면 아래 urls.py에 설정한 주소로 이동합니다.
 - path('<int:question_id>/', views.detail, name='detail'), # /polls/넘버/

{% else %}

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

{% endif %}

- 만일 latest_question_list 객체에 내용이 없다면 “No polls are available” 문장을 화면에 출력합니다.

index() 뷰 함수

- `def index(request):`
 - `latest_question_list = Question.objects.all().order_by('-pub_date')[:5]`
 - 템플릿에서 넘겨줄 객체의 이름일 `latest_question_list`로 정합니다.
 - `latest_question_list` 객체는 Question 테이블 객체에서 `pub_date` 컬럼의 역순으로 정렬하여 5개의 최근 Question 객체를 가져와 만듭니다.
 - `context = {'latest_question_list': latest_question_list}`
 - 템플릿에 넘겨주는 방식은 파이썬 사전 타입으로, 템플릿에 사용될 변수명과 그 변수명에 해당하는 객체를 매핑하는 사전으로 `context` 변수를 만들어서 `render()` 함수에 보내줍니다.
 - `return render(request, 'polls/index.html', context)`
 - `render()` 함수는 템플릿 파일인 `polls/index.html`에 `context` 변수를 적용하여 사용자에게 보여줄 최종 HTML 텍스트를 만들고, 이를 담아서 `HttpResponse` 객체를 반환합니다.

`index()` 뷰 함수는 최종적으로 클라이언트에게 응답할 데이터인 `HttpResponse` 객체를 반환합니다.

단축함수

- 웹 프로그램 개발 시 자주 사용되는 기능들, 예를 들어 `render()` 함수처럼 템플릿 코드를 로딩한 후에 컨텍스트 변수를 적용하고, 그 결과를 `HttpResponse` 객체에 담아 반환하는 작업 등의 공통적으로 사용되는 기능들을 장고에서는 이미 개발하여 내장 함수로 제공하고 있고, 이를 단축함수라고 합니다.

- `<h1>{{ question.question_text }}</h1>`
- `{% if error_message %}<p>{{ error_message }}</p>{% endif %}`
 - 에러가 있다면 에러 메시지를 화면에 보여줍니다. 에러를 체크하는 로직은 `vote()` 뷰 함수에 있습니다.
 - `vote()` 뷰 함수에서 익셉션이 발생되면 `error_message`를 담아서 `detail.html` 템플릿을 렌더링하고, 그에 따라 지금 보고 있는 `detail.html` 템플릿에서 에러 메시지를 보여주게 됩니다.
- `<form action="{% url 'polls:vote' question.id %}" method="post">`
 - 폼에 입력된 데이터는 POST 방식으로 보냅니다. 서버 측의 데이터를 변경하는 경우, 일반적으로 GET이 아니라 POST 방식을 사용합니다.
 - `polls:vote`는 `URLconf`에서 정의한 URL 패턴의 이름입니다.
 - `path('<int:question_id>/vote/', views.vote, name='vote'),` `# /polls/넘버/vote/`
- `{% csrf_token %}`
 - CSRF(Cross Site Request Forgery) 공격을 방지하기 위한 제공되는 기능입니다.

- {% for choice in question.choice_set.all %}
 - {% for %} 태그로 뷰 함수에서 넘어온 객체를 순회하고 있습니다.
 - index() 뷰에서처럼 이번 detail() 뷰 함수에서도 Question 객체를 템플릿으로 넘겨주고 있습니다.
 - 뷰 함수를 작성할 때 템플릿에서 무엇을 넘겨줄지는 항상 숙고해야하는 사항입니다. question.choice_set.all의 의미는 Question 객체의 choice_set 속성에 들어 있는 모든 항목을 의미합니다.
- <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />
 - 라디오 버튼으로 답변 항목을 보여주고 있습니다. 해당 라디오 버튼을 선택하면 POST 데이터가 'choice='3'(choice.id) 형태로 구성되도록 name과 value 속성을 정의하고 있습니다.
- <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label>

 - forloop.counter 변수는 for 루프를 실행한 횟수를 담고 있는 템플릿 변수입니다.
 - <label for> 속성과 <input id> 속성은 값이 같아야 서로 바인딩됩니다.
- {% endfor %}
<input type="submit" value="Vote" />
 - Vote 버튼을 클릭하면 사용자가 선택한 폼 데이터가 POST 방식으로 polls:vote URL로 전송됩니다.
 - vote() 뷰 함수에서는 request.POST['choice']구문으로 액세스합니다.
- </form>

detail() 뷰 함수

- `def detail(request, question_id):`
 - 뷰 함수를 정의합니다. `request` 객체는 필수 인자이고, 추가적으로 `question_id` 인자를 받습니다.
 - url에 있는 아래 코드에서 `question_id`의 값이 전달됩니다.
 - `path('polls/<int:question_id>/', views.detail, name='detail')`

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

- 단축함수를 사용하고 있습니다. 첫번째 인자는 모델 클래스이고, 두번째 인자부터는 검색 조건을 여러 개 사용할 수 있습니다. 이 예제는 `Question` 모델 클래스로부터 `pk=question_id` 검색 조건에 맞는 객체를 조회합니다.
 - 조건에 맞는 객체가 없다면 `Http404` 익셉션을 발생시킵니다.
- `return render(request, 'polls/detail.html', {'question': question})`

vote() 뷰 함수

- `def vote(request, question_id):`
 - request 객체는 필수 인자입니다.
 - `path(' <int:question_id>/vote/ ', views.vote, name= ' vote '),` `# /polls/넘버/vote/`
 - `urls.py`에서 `question_id`의 값을 두번째 인자로 받습니다.
- `question = get_object_or_404(Question, pk=question_id)`
- `try:`
 - `selected_choice = question.choice_set.get(pk=request.POST['choice'])`
 - `pk=request.POST['choice']`에서 해당하는 값인 `choice_id`를 스트링으로 리턴합니다.
- `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,)))`

vote() 뷰 함수

- else:

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

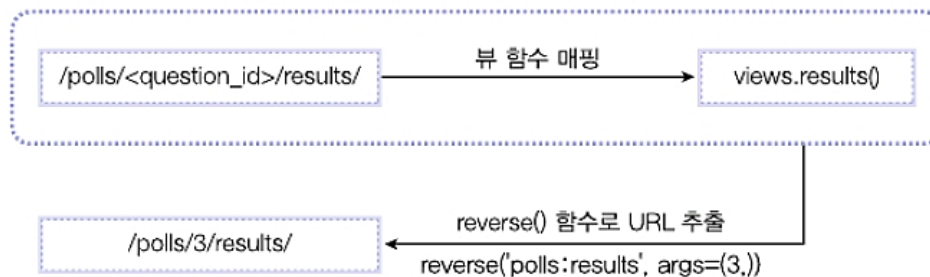
```
selected_choice.save()
```

```
return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))
```

- Choice 객체.votes 속성, 즉 선택 카운트를 +1 증가시킵니다.
- 그리고 해당 choice 테이블에 저장합니다.
- HttpResponseRedirect 객체의 생성자는 리다이렉트할 타겟 URL을 인자로 받습니다.
- 타겟 URL은 reverse() 함수로 만듭니다.
- 웹 프로그램에서 POST 방식의 폼 데이터를 처리하는 경우, 그 결과를 보여줄 수 있는 페이지로 이동시키기 위해 HttpResponseRedirect 객체를 리턴하는 것이 일반적입니다.

- reverse() 함수의 원리

URLconf에 정의된 URL 패턴명 = polls:results



result() 뷰 함수 및 result.html

- `def results(request, question_id):`
 `question = get_object_or_404(Question, pk=question_id)`
 `return render(request, 'polls/results.html', {'question': question})`

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

``

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

`{{ choice.choice_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}`

- Choice 객체의 choice_text를 순서 없는 리스트로 화면에 보여줍니다. 또한 각 텍스트 옆에 투표 카운트(choice.votes)를 숫자로 보여줍니다. vote{{choice.votes|pluralize}}의 의미는 choice.votes 값에 따라 복수 접미사(s)를 붙여주는 것입니다.

•

`{% endfor %}`

``

`Vote again?`