- **RequireDebugFalse** : 이 필터는 settings.DEBUG 속성이 False인 경우만 메시지 처리를 진행합니다.
- **RequireDebugTrue** : 이 필터는 settings.DEBUG 속성이 True인 경우만 메시지 처리를 진행시킵니다.

장고에서 로깅 관련 디폴트로 설정되는 사항은 아래 2가지입니다.

- **django.request 로거에 대한 디폴트 설정** : settings.DEBUG 속성이 False이면 ERROR 또는 CRITICAL 레벨의 모든 메시지를 AdminEmailHandler 핸들러에게 보내주도록 디폴트로 설정되어 있습니다.
- **django 루트 로거에 대한 디폴트 설정** : django 루트 로거에 오는 모든 메시지는 DEBUG 속성이 True인 경우 콘솔로 보내지고, DEBUG 속성이 False인 경우 NullHandler로 보내서 무시되도록 디폴트로 설정되어 있습니다.

CHAPTER 05

실습 예제 확장하기

앞서 3장에서 장고 프로젝트와 애플리케이션을 설명하면서 간단한 실습 예제를 만들어 보았습니다. mysite 프로젝트 하위에 polls 애플리케이션을 코딩해보면서 프로젝트 뼈대를 만들고 MTV 패턴에 맞춰 애플리케이션을 개발하였습니다. 3장의 예제에서 뷰를 작성할 때는 원리부터 설명하기 위해 함수형 뷰를 사용하였습니다. 4장에서 설명하였듯이 함수형 뷰보다는 클래스형 뷰가 장점이 많기 때문에 기본적인 내용을 습득했다면 그 후에는 클래스형 뷰로 작성하는 것이 좋습니다.

이번 장에서는 3장의 예제를 확장하여 클래스형 뷰를 사용하는 애플리케이션을 만들어 보겠습니다. 우선 books라는 새로운 애플리케이션을 만들어 보면서 클래스형 뷰의 사용법을 익히고, 그 다음에 함수형 뷰로 되어 있는 기존의 polls 애플리케이션을 클래스형 뷰로 변경하는 예제를 실습하도록 하겠습니다.

5.1 새로운 애플리케이션 만들기

이번 장에서는 books라는 새로운 애플리케이션을 코딩할 예정입니다. books 애플리케이션은 책을 출판하는 데 필요한 정보들인 책, 저자, 출판사에 대한 정보들을 관리하는 웹 애플리케이션입니다. 새로운 애플리케이션을 만들기 위해서는 그 상위에 프로젝트를 먼저 만들어야 합니다. 프로젝트는 3장에서 사용한 mysite를 그대로 사용하고 그 하위에 books 애플리케이션을 코딩할 예정입

니다. 물론 다른 프로젝트를 만들어도 상관없습니다. "웹 사이트를 어떻게 구성할 것인가"라는 문제는 설계 단계에서 결정할 사항입니다.

3장에서 설명한 것처럼 개발 순서는 다음과 같습니다.

- 프로젝트 뼈대 만들기
- 애플리케이션 설계하기
- 애플리케이션 - Model 코딩하기
- 애플리케이션 - URLconf 코딩하기
- 애플리케이션 - Template 코딩하기
- 애플리케이션 - View 코딩하기

> **NOTE_ 코딩 순서(M.T.V)**
>
> 뷰와 템플릿의 코딩 순서는 정해진 것이 아니므로 개발자의 취향에 따라 진행해도 무관합니다. 필자의 경우는 화면을 생각하면서 로직을 코딩하는 것이 더 효율적이라 판단하여 보통 템플릿을 먼저 코딩합니다. 테이블은 설계 시 정해지므로 당연히 모델이 제일 먼저 코딩됩니다. 따라서 필자의 경우는 Model, Template, View 순으로 코딩하는 것을 원칙으로 삼고 있습니다. 여러분도 자신만의 코딩 순서를 정하는 것이 로직을 풀어나가는 데 일관성을 유지할 수 있고, 웹 개발 노하우도 빨리 습득할 수 있을 것이라 생각합니다.

5.1.1 프로젝트 뼈대 만들기

이번 예제에서는 mysite라는 기존의 프로젝트를 사용할 것이므로, 프로젝트 뼈대는 이미 만들어져 있습니다. 거기에 books 애플리케이션만 추가하면 됩니다. 다음 명령을 실행하여 프로젝트 뼈대 만들기를 완성합니다.

```
$ cd /home/shkim/pyBook/ch5
$ python manage.py startapp books
```

그러면 예상했던 것처럼 장고가 books라는 애플리케이션 디렉토리를 만들어주고, 그 하위에 필요한 파일들을 자동으로 생성해줍니다.

```
[shkim@localhost pyBook]$ cd /home/shkim/pyBook/ch5
[shkim@localhost ch5]$ python manage.py startapp books
[shkim@localhost ch5]$ ls -al
합계 52
drwxrwxr-x. 6 shkim shkim    92  1월   3 18:14 .
drwxrwxr-x. 7 shkim shkim    75  1월   3 18:14 ..
drwxrwxr-x. 3 shkim shkim   102  1월   3 18:14 books
-rw-r--r--. 1 shkim shkim 40960  1월   3 18:14 db.sqlite3
-rwxr-xr-x. 1 shkim shkim   249  1월   3 18:14 manage.py
drwxrwxr-x. 2 shkim shkim  4096  1월   3 18:14 mysite
drwxrwxr-x. 4 shkim shkim  4096  1월   3 18:14 polls
drwxrwxr-x. 3 shkim shkim    18  1월   3 18:14 templates
[shkim@localhost ch5]$
[shkim@localhost ch5]$ ls -al books
합계 16
drwxrwxr-x. 3 shkim shkim 102  1월   3 18:14 .
drwxrwxr-x. 6 shkim shkim  92  1월   3 18:14 ..
-rw-r--r--. 1 shkim shkim   0  1월   3 18:14 __init__.py
-rw-r--r--. 1 shkim shkim  63  1월   3 18:14 admin.py
drwxrwxr-x. 2 shkim shkim  24  1월   3 18:14 migrations
-rw-r--r--. 1 shkim shkim  57  1월   3 18:14 models.py
-rw-r--r--. 1 shkim shkim  60  1월   3 18:14 tests.py
-rw-r--r--. 1 shkim shkim  63  1월   3 18:14 views.py
[shkim@localhost ch5]$
```

그림 5-1 startapp 명령 실행 후 디렉토리 모습 – books 애플리케이션

5.1.2 애플리케이션 설계하기

우리가 개발할 books 애플리케이션의 내용은 책, 저자, 출판사의 정보를 관리하는, 즉 정보를 보여주고 입력, 수정, 삭제할 수 있는 웹 애플리케이션입니다. 다음과 같이 UI, 테이블, 뷰의 흐름을 설계하였습니다. 책, 저자, 출판사 정보 중에서 책 정보 관리 위주로 설계된 내용을 보여주고 있습니다. 저자, 출판사 관리 기능에 대한 화면 UI 및 뷰 흐름의 설계 내용도 책 정보 설계 내용과 거의 유사할 것입니다.

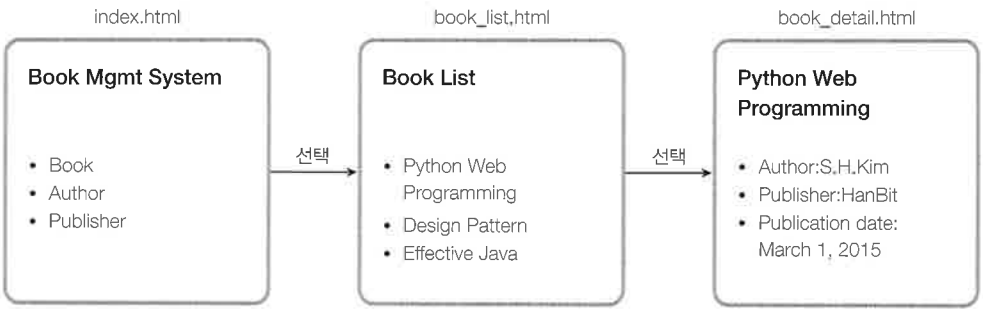


그림 5-2 books 애플리케이션 – UI 설계

표 5-1 books 애플리케이션 – Book 테이블 설계

컬럼명	타입	제약조건	설명
id	integer	NotNull, PK, AutoIncrement	Primary Key
title	varchar(100)	NotNull	책 제목
authors	integer	NotNull, FK (Author.id), index	Many-To-Many
publisher	Integer	NotNull, FK (Publisher.id), index	Foreign Key
publication_date	date	NotNull	책 출판일

표 5-2 books 애플리케이션 – Author 테이블 설계

컬럼명	타입	제약조건	설명
id	integer	NotNull, PK, AutoIncrement	Primary Key
salutation	varchar(100)	NotNull	저자 인사말
name	varchar(50)	NotNull	저자 성명
email	email	NotNull	저자 이메일

표 5-3 books 애플리케이션 – Publisher 테이블 설계

컬럼명	타입	제약조건	설명
id	integer	NotNull, PK, AutoIncrement	Primary Key
name	varchar(50)	NotNull	출판사 이름
address	varchar(200)	NotNull	출판사 주소
website	url	NotNull	출판사 홈페이지

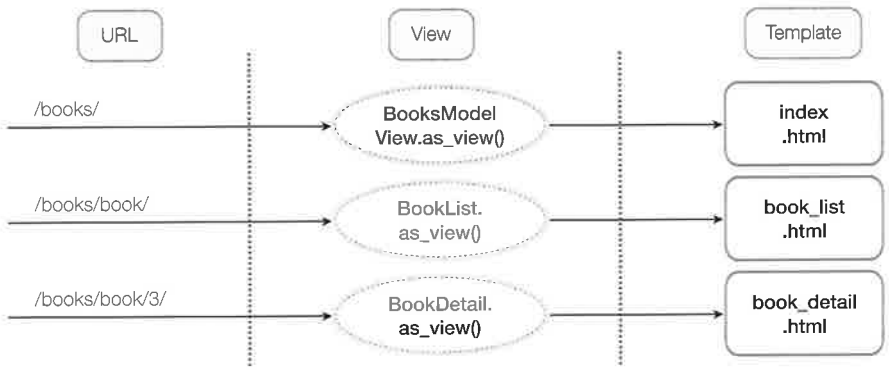


그림 5-3 books 애플리케이션 – 뷰 흐름 설계

5.1.3 애플리케이션 - Model 코딩하기

모델 작업은 다음과 같은 순서대로 진행합니다.

```
$ vi settings.py                          // 프로젝트 설정 파일에 데이터베이스를 지정함
$ vi models.py                            // 테이블을 정의함
$ vi admin.py                             // 정의된 테이블이 Admin 화면에 보이게 함
$ python manage.py makemigrations         // 데이터베이스에 변경이 필요한 사항을 추출함
$ python manage.py migrate                // 데이터베이스에 변경사항을 반영함
$ python manage.py runserver              // 현재까지 작업을 개발용 웹 서버로 확인함
```

프로젝트 설정 파일인 settings.py 파일에 필요한 사항을 지정합니다. 3장에서 지정한 Database, INSTALLED_APPS, TIME_ZONE 항목 중에서 INSTALLED_APPS 항목만 수정해주면 됩니다.

예제 5-1 mysite/settings.py 파일에 books 애플리케이션 등록

```
$ cd /home/shkim/pyBook/ch5/mysite
$ vi settings.py

# 상단 내용 동일
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'polls',
    'books',      # 추가
)
# 하단 내용 동일
```

다음은 테이블에 대해 설계된 내용에 따라 models.py 파일에 테이블을 정의합니다.

예제 5-2 books/models.py 소스제공

```
$ cd /home/shkim/pyBook/ch5/books
$ vi models.py
```

```
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=100)
    authors = models.ManyToManyField('Author')
    publisher = models.ForeignKey('Publisher')
    publication_date = models.DateField()

    def __unicode__(self): # __str__ on Python 3
        return self.title

class Author(models.Model):
    salutation = models.CharField(max_length=100)
    name = models.CharField(max_length=50)
    email = models.EmailField()

    def __unicode__(self): # __str__ on Python 3
        return self.name

class Publisher(models.Model):
    name = models.CharField(max_length=50)
    address = models.CharField(max_length=100)
    website = models.URLField()

    def __unicode__(self): # __str__ on Python 3
        return self.name
```

Admin 사이트에 보이도록 테이블을 admin.py 파일에도 등록해줍니다.

예제 5-3 books/admin.py 소스제공

```
$ cd /home/shkim/pyBook/ch5/books
$ vi admin.py

from django.contrib import admin
from books.models import Book, Author, Publisher

admin.site.register(Book)
admin.site.register(Author)
admin.site.register(Publisher)
```

다음 명령으로 방금 정의한 테이블을 데이터베이스에 반영해줍니다.

```
$ cd /home/shkim/pyBook/ch5
$ python manage.py makemigrations
$ python manage.py migrate
```

지금까지의 작업을 확인하고 싶다면 아래처럼 runserver를 실행시키고,

```
$ cd /home/shkim/pyBook/ch5
$ python manage.py runserver 0.0.0.0:8000
```

Admin 사이트에 접속해서 테이블이 보이는지 확인해보면 됩니다.

```
http://192.168.56.101:8000/admin
```

5.1.4 애플리케이션 - URLconf 코딩하기

[그림 5-3]에서 설계된 뷰 흐름의 내용을 참고해서 URLconf를 정의하면 됩니다. URLconf는 mysite/urls.py와 books/urls.py 2개의 파일에 코딩합니다. 먼저 기존의 mysite/urls.py 파일에 /books/와 관련된 한 줄을 추가합니다.

예제 5-4 books 애플리케이션 – mysite/urls.py 소스제공

```
$ cd /home/shkim/pyBook/ch5/mysite
$ vi urls.py

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

urlpatterns = patterns('',
    url(r'^polls/', include('polls.urls', namespace="polls")),
    url(r'^books/', include('books.urls', namespace="books")), # 추가
    url(r'^admin/', include(admin.site.urls)),
)
```

다음은 books/urls.py입니다. [그림 5-3]에서는 대표적으로 /book/ URL만 그림에 보여주었는데, books/urls.py 파일에는 /book/과 동일한 형식으로 /author/ 및 /publisher/ URL도 같이 정의해줍니다.

예제 5-5 books 애플리케이션 – books/urls.py

```
$ cd /home/shkim/pyBook/ch5/books
$ vi urls.py

from django.conf.urls import patterns, url
from books import views

urlpatterns = patterns('',
    url(r'^$', views.BooksModelView.as_view(), name='index'),

    url(r'^book/$', views.BookList.as_view(), name='book_list'),
    url(r'^author/$', views.AuthorList.as_view(), name='author_list'),
    url(r'^publisher/$', views.PublisherList.as_view(), name='publisher_list'),

    url(r'^book/(?P<pk>\d+)/$', views.BookDetail.as_view(), name='book_detail'),
    url(r'^author/(?P<pk>\d+)/$', views.AuthorDetail.as_view(), name='author_detail'),
    url(r'^publisher/(?P<pk>\d+)/$', views.PublisherDetail.as_view(), name='publisher_
detail'),
)
```

7개의 URL을 정의하였고, 그에 해당하는 클래스형 뷰 역시 7개를 정의하였습니다. 그리고 함수형 뷰가 아니라 클래스형 뷰로 정의하기 위해 각 URL에 따른 뷰 클래스 및 as_view() 메소드를 지정하였습니다. 참고로, 다음 절에서 살펴보겠지만 템플릿 파일도 각각 하나씩 총 7개가 필요합니다.

5.1.5 애플리케이션 - Template 코딩하기

뷰 클래스가 7개이고 템플릿 파일도 7개가 필요합니다. 파일의 위치는 4장에서 설명했듯이 /home/shkim/pyBook/ch5/books/templates/books/ 디렉토리입니다. URL/뷰/템플릿 매핑을 아래 표에 정리하였습니다.

표 5-4 books 애플리케이션 – URL/뷰/템플릿 매핑

URL 패턴	뷰 클래스명	템플릿 파일명	템플릿 설명
/books/	BooksModelView	Index.html	books 애플리케이션 첫 화면
/books/book/	BookList	book_list.html	책의 리스트를 보여줌
/books/author/	AuthorList	author_list.html	저자의 리스트를 보여줌
/books/publisher/	PublisherList	publisher_list.html	출판사의 리스트를 보여줌
/books/book/3/	BookDetail	book_detail.html	특정 책의 상세 정보를 보여줌
/books/author/3/	AuthorDetail	author_detail.html	특정 저자의 상세 정보를 보여줌
/books/publisher/3/	PublisherDetail	publisher_detail.html	특정 출판사의 상세 정보를 보여줌

※ URL 패턴에서 **3**은 예시로, 각 항목의 테이블 레코드 Primary Key가 채워지는 자리입니다.

각 템플릿 파일에서는 상속 기능을 사용하고 있습니다. 부모 템플릿 역할을 하는 base.html 및 base_books.html 템플릿은 다음 절에서 살펴보기로 하고, 우선 books 애플리케이션의 첫 화면을 보여주는 index.html 템플릿 파일을 살펴보겠습니다. 그 다음에 각 모델의 레코드 리스트를 보여주는 xxx_list.html과 특정 레코드의 상세 정보를 보여주는 xxx_detail.html 템플릿 파일을 알아보겠습니다.

index.html 템플릿을 다음과 같이 입력합니다.

예제 5-6 books 애플리케이션 – index.html 템플릿 **소스제공**

```
$ cd /home/shkim/pyBook/ch5/books/
$ mkdir templates
$ mkdir templates/books
$ cd templates/books
$ vi index.html

{% extends "base_books.html" %}

{% block content %}
    <h2>Books Management System</h2>
    <ul>
        {% for modelname in object_list %}
{% with "books:"|add:modelname|lower|add:"_list" as urlvar %}
            <li><a href="{% url urlvar %}">{{ modelname }}</a></li>
{% endwith %}
        {% endfor %}
```

```
    </ul>
{% endblock content %}
```

위 index.html 템플릿에서는 base_books.html 템플릿을 상속받아서 content 블록만을 재정의하였고, 나머지 블록은 부모 템플릿 내용을 그대로 사용하고 있습니다.

뷰로부터 object_list 컨텍스트 변수를 전달받아서 object_list에 들어 있는 모델명modelname들을 순회하면서 화면에 하나씩 보여주고 있습니다. 또한 모델명을 클릭 시 접속할 URL을 추출하기 위해 {% url urlvar %} 템플릿 태그를 사용하였고, urlvar 인자는 {% with %} 태그를 사용하여 다음과 같이 정의하고 있습니다.

```
{% with "books:"|add:modelname|lower|add:"_list" as urlvar %}
```

이 문장은 add 및 lower 템플릿 필터를 사용하여 모델명을 소문자로 변환하고, 필요한 문자열을 붙여줍니다. 예를 들어, 모델명이 Author라면 urlvar는 books:author_list가 될 것입니다.

[예제 5-7] ~ [예제 5-9]는 책, 저자, 출판사의 리스트를 보여주는 템플릿 파일들입니다. 내용은 거의 동일합니다. 공통적으로 base_books.html 템플릿을 상속받고 있고, 뷰로부터 object_list 컨텍스트 변수를 전달받아서 object_list에 들어 있는 객체들을 순회하면서 하나씩 보여주고 있습니다. 상속을 받는 하위 템플릿 파일에서는 {% block content %} 블록만 재정의하고, 나머지 부분은 부모 base_books.html 템플릿 내용을 그대로 사용하고 있다는 것을 알 수 있습니다.

예제 5-7 books 애플리케이션 - book_list.html 템플릿 **소스제공**

```
$ cd /home/shkim/pyBook/ch5/books/templates/books
$ vi book_list.html

{% extends "base_books.html" %}

{% block content %}
    <h2>Book List</h2>
    <ul>
        {% for book in object_list %}
            <li><a href="{% url 'books:book_detail' book.id %}">{{ book.title }}</a></li>
```

```
{% endfor %}
    </ul>
{% endblock content %}
```

다음 문장은 화면에 book 객체의 title 속성(book.title)을 표시하고 해당 텍스트를 클릭하는 경우, <a href> 태그 기능에 의해 books:book_detail URL 패턴으로 웹 요청을 보낸다는 의미입니다. {% url %} 태그 기능은 **4. Django의 핵심 기능**에서 자세히 설명하였으므로 생략하겠습니다.

```
<li><a href="{% url 'books:book_detail' book.id %}">{{ book.title }}</a></li>
```

다음의 author_list.html 템플릿 파일은 화면에 표시하는 내용이 {{ author.name }}이란 점만 다르고 위 템플릿과 동일합니다.

예제 5-8 books 애플리케이션 - author_list.html 템플릿 **소스제공**

```
$ cd /home/shkim/pyBook/ch5/books/templates/books
$ vi author_list.html

{% extends "base_books.html" %}

{% block content %}
    <h2>Author List</h2>
    <ul>
        {% for author in object_list %}
            <li><a href="{% url 'books:author_detail' author.id %}">{{ author.name }}</
a></li>
{% endfor %}
    </ul>
{% endblock content %}
```

다음의 publisher_list.html 템플릿 파일 역시 위의 2개와 거의 동일합니다. {{ publisher.name }}을 화면에 보여주고 있습니다.

예제 5-9 books 애플리케이션 - publisher_list.html 템플릿 소스제공

```
$ cd /home/shkim/pyBook/ch5/books/templates/books
$ vi publisher_list.html

{% extends "base_books.html" %}

{% block content %}
    <h2>Publisher List</h2>
    <ul>
        {% for publisher in object_list %}
            <li><a href="{% url 'books:publisher_detail' publisher.id %}">{{ publisher.
name }}</a></li>
        {% endfor %}
    </ul>
{% endblock content %}
```

[예제 5-10] ~ [예제 5-12]는 책, 저자, 출판사 테이블의 특정 레코드에 대한 상세 정보를 보여주는 템플릿 파일들입니다. 이 3개의 파일들도 거의 동일합니다.

공통적으로 base_books.html 템플릿을 상속받고 있고, 특정 레코드를 object라는 컨텍스트 변수로 전달받아서 object 객체, 즉 레코드에 들어 있는 컬럼값들을 보여주고 있습니다. 상속을 받는 하위 템플릿 파일에서는 {% block content %} 블록만 재정의하고, 나머지 부분은 부모 base_books.html 템플릿 내용을 그대로 사용하고 있다는 것을 알 수 있습니다.

예제 5-10 books 애플리케이션 - book_detail.html 템플릿 소스제공

```
$ cd /home/shkim/pyBook/ch5/books/templates/books
$ vi publisher_detail.html

{% extends "base_books.html" %}

{% block content %}
<h1>{{ object.title }}</h1>
<br>
<li>Authors:
{% for author in object.authors.all %}
  {{ author }}
  {% if not forloop.last %},{% else %}{% endif %}
```

```
{% endfor %}
</li>
<li>Publisher: {{ object.publisher }}</li>
<li>Publication date: {{ object.publication_date }}</li>
{% endblock content %}
```

첫 번째 템플릿은 Book 테이블에 들어 있는 특정 레코드, 즉 특정 책의 상세 정보를 표시하고 있습니다. 아래 2개의 템플릿과 다른 점은 책의 저자가 여러 명일 수 있으므로, 다음과 같이 공동 저자인 경우 저자 이름 뒤에 콤마(,)를 출력하는 로직이 추가되었습니다.

```
{% if not forloop.last %},{% else %}{% endif %}
```

예제 5-11 books 애플리케이션 – author_detail.html 템플릿 소스제공

```
$ cd /home/shkim/pyBook/ch5/books/templates/books
$ vi publisher_detail.html

{% extends "base_books.html" %}

{% block content %}
<h1>{{ object.name }}</h1>
<p>{{ object.salutation }}</p>
<li>Email: {{ object.email }}</li>
{% endblock content %}
```

예제 5-12 books 애플리케이션 – publisher_detail.html 템플릿 소스제공

```
$ cd /home/shkim/pyBook/ch5/books/templates/books
$ vi publisher_detail.html

{% extends "base_books.html" %}

{% block content %}
<h1>{{ object.name }}</h1>
<p>{{ object.website }}</p>
<li>Address: {{ object.address }}</li>
{% endblock content %}
```

5.1.6 애플리케이션 - Template 상속 기능 추가

이제 부모 템플릿인 base.html 및 base_books.html 템플릿을 작성하겠습니다. 아래 예제의 base.html 템플릿에서는 상속용으로 {% block title %}, {% block sidebar %}, {% block content %} 3개의 블록을 정의하고 있습니다.

그 다음의 base_books.html 템플릿은 base.html 템플릿을 상속받아서 이 중 title 블록과 sidebar 블록을 재정의하고 있습니다. 또한 앞 절에서 이미 보았듯이 각 xxx_list.html과 xxx_detail.html 및 index.html 템플릿에서는 content 블록만을 재정의하고 있습니다. 결과적으로 아래 그림처럼 장고에서 일반적으로 권고하고 있는 3단계 템플릿 상속 구조를 따르고 있습니다.

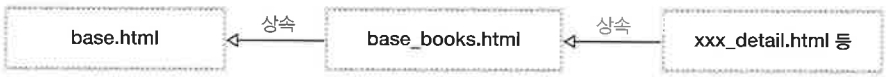


그림 5-4 템플릿 상속 구조

위에서 2개의 템플릿은 각 템플릿의 부모 템플릿 역할을 하고 있고, 개별 애플리케이션 템플릿이 아니라 공통 템플릿이므로, 다음의 디렉토리에 생성합니다.

```
/home/shkim/pyBook/ch5/templates
```

> **NOTE_** 이 위치는 setting.py 파일에서 아래처럼 TEMPLATE_DIRS 항목으로 정의한 디렉토리라는 점을 유의하기 바랍니다.
>
> TEMPLATE_DIRS = [os.path.join(BASE_DIR, 'templates')]

예제 5-13 템플릿 상속 - base.html 템플릿 소스제공

```
$ cd /home/shkim/pyBook/ch5/templates
$ vi base.html

<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}
    <link rel="stylesheet" href="{% static "admin/css/base.css" %}" />
    <title>{% block title %}My Amazing Site{% endblock %}</title>
```

```
</head>

<body>
    <div id="sidebar">
        {% block sidebar %}
        <ul>
            <li><a href="/">Project_Home</a></li>
            <li><a href="/admin/">Admin</a></li>
        </ul>
        {% endblock %}
        <br>
    </div>

    <div id="content">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

{% load static %} 템플릿 태그는 static이라는 사용자 정의 태그를 로딩해주고, 그 다음에 {% static %} 사용자 정의 태그를 통해 admin/css/base.css 스타일시트 파일을 찾게 됩니다. 이는 장고의 Admin 사이트에서 사용하는 스타일시트를 사용하여 룩앤필을 보여주는 효과를 줍니다.

base_books.html 템플릿 파일을 아래와 같이 작성합니다.

예제 5-14 템플릿 상속 - base_books.html 템플릿 소스제공

```
$ cd /home/shkim/pyBook/ch5/templates
$ vi base_books.html

{% extends "base.html" %}

<title>{% block title %}Books Application Site{% endblock %}</title>

{% block sidebar %}
{{ block.super }}
<ul>
    <li><a href="/books/">Books_Home</a></li>
</ul>
{% endblock %}
```

위 base_books.html 템플릿은 base.html 템플릿을 상속받아서 이 중 title 블록과 sidebar 블록을 재정의하고 있습니다. {{ **block.super** }} 템플릿 변수의 의미는 부모 base.html 템플릿에서 정의한 내용을 하위 base_books.html 템플릿에서 재사용한다는 의미입니다. 즉, sidebar 블록은 최종적으로 다음과 같은 코드가 됩니다.

예제 5-15 템플릿 상속 - sidebar 블록의 최종 코드

```
{% block sidebar %}
<ul>
    <li><a href="/">Project_Home</a></li>
    <li><a href="/admin/">Admin</a></li>
</ul>
<ul>
    <li><a href="/books/">Books_Home</a></li>
</ul>
{% endblock %}
```

5.1.7 애플리케이션 - 클래스형 View 코딩하기

앞에서 URLconf를 코딩하면서 뷰를 클래스형 뷰로 정의하기 위해 각 URL에 따른 해당 클래스 및 as_view() 메소드를 지정하였습니다. 이제 URLconf에서 지정한 클래스형 뷰를 코딩하겠습니다. 이미 여러 번 강조하였듯이 클래스형 뷰 또는 클래스형 지네릭 뷰는 장고 사용 시 매우 중요한 기능이고, 이번 장의 핵심 내용입니다. 아래 예제를 이해하는 것뿐만 아니라, 여러분 나름대로 예제를 수정 및 추가하면서 클래스형 뷰에 익숙해지기를 바랍니다.

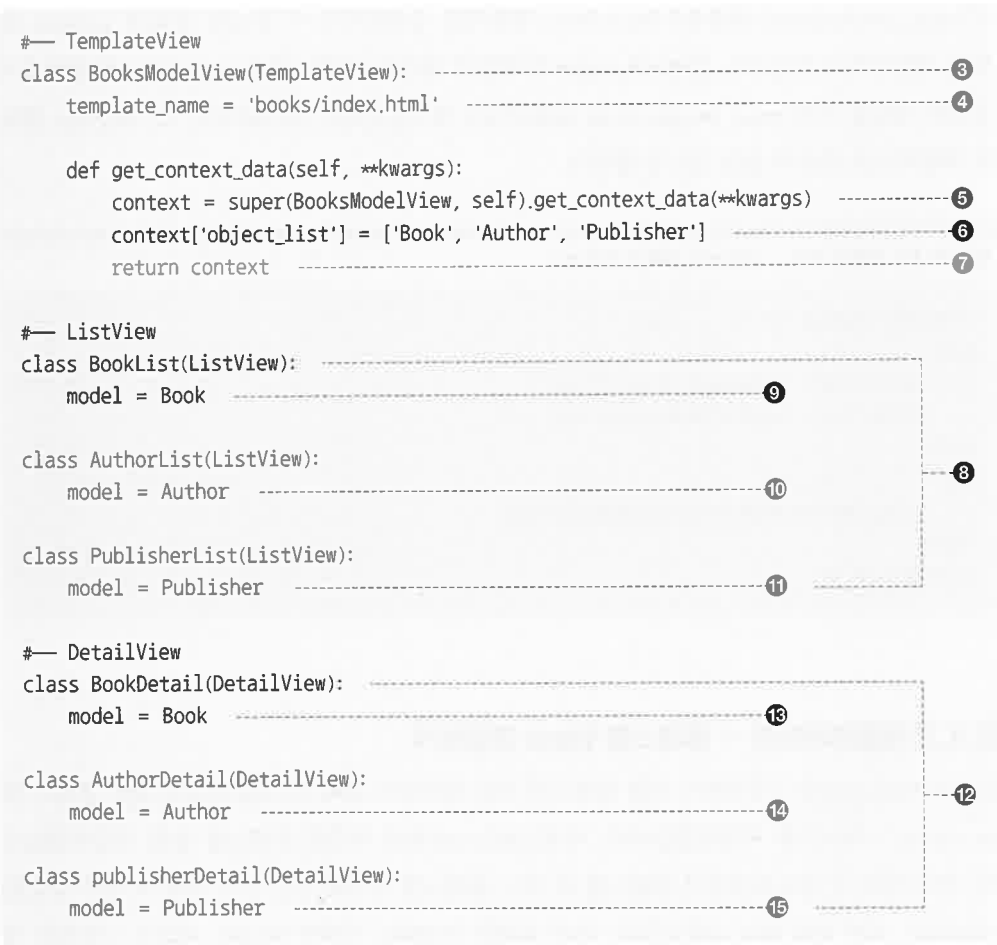
books/views.py 파일에 아래 내용을 입력합니다.

예제 5-16 books/views.py **소스제공**

```
$ cd /home/shkim/pyBook/ch5/books
$ vi views.py

from django.views.generic.base import TemplateView
from django.views.generic import ListView
from django.views.generic import DetailView
from books.models import Book, Author, Publisher
```

❶
❷



위 소스를 라인별로 설명하겠습니다.

❶ 클래스형 지네릭 뷰를 사용하기 위해 TemplateView, ListView, DetailView 클래스를 임포트합니다.

❷ 테이블 조회를 위하여 모델 클래스들을 임포트합니다.

❸ BooksModelView는 books 애플리케이션의 첫 화면을 보여주기 위한 뷰입니다. 특별한 로직이 없고 템플릿 파일만을 렌더링하는 경우에는 이처럼 **TemplateView** 지네릭 뷰를 상속받아 사용하면 간단합니다. TemplateView를 사용하는 경우에는 필수적으로 **template_name** 클래스 변수를 오버라이딩해서 지정해 줘야 합니다. 템플릿 시스템으로 넘겨줄 컨텍스트 변수가 있는 경우에는 **get_context_data()** 메소드를 오버라이딩해서 정의해주면 됩니다.

❹ books 애플리케이션의 첫 화면을 보여주기 위한 템플릿 파일을 books/index.html로 지정하였습니다.

❺ get_context_data() 메소드를 정의할 때는 반드시 첫 줄에 super() 메소드를 호출해야 합니다.

❻ books 애플리케이션의 첫 화면에 테이블 리스트를 보여주기 위해 컨텍스트 변수 object_list에 담아서 템플릿 시스템에 넘겨주고 있습니다.

❼ return 문장도 필수입니다.

❽ 다음 3개의 클래스는 모두 **ListView** 지네릭 뷰를 사용하고 있습니다. ListView를 상속받는 경우는 객체가 들어 있는 리스트를 구성해서 이를 컨텍스트 변수로 템플릿 시스템에 넘겨주면 됩니다. 만일 이런 리스트를 테이블에 들어 있는 모든 레코드를 가져와 구성하는 경우에는 테이블명, 즉 모델 클래스명만 지정해주면 됩니다.

그리고 명시적으로 지정하지 않아도 장고에서 디폴트로 알아서 지정해주는 속성이 2가지 있습니다. 첫 번째는 컨텍스트 변수로 **object_list**를 사용하는 것이고, 두 번째는 템플릿 파일을 **모델명 소문자_list.html** 형식의 이름으로 지정하는 것입니다.

❾ Book 테이블로부터 모든 레코드를 가져와 object_list라는 컨텍스트 변수를 구성합니다. 템플릿 파일은 디폴트로 books/book_list.html 파일이 됩니다.

❿ Author 테이블로부터 모든 레코드를 가져와 object_list라는 컨텍스트 변수를 구성합니다. 템플릿 파일은 디폴트로 books/author_list.html 파일이 됩니다.

⓫ Publisher 테이블로부터 모든 레코드를 가져와 object_list라는 컨텍스트 변수를 구성합니다. 템플릿 파일은 디폴트로 books/publisher_list.html 파일이 됩니다.

⓬ 다음 3개의 클래스는 모두 **DetailView** 지네릭 뷰를 사용하고 있습니다. DetailView를 상속받는 경우는 특정 객체 하나를 컨텍스트 변수에 담아서 템플릿 시스템에 넘겨주면 됩니다. 만일 테이블에서 Primary Key로 조회해서 특정 객체를 가져오는 경우에는 테이블명, 즉 모델 클래스명만 지정해주면 됩니다. 조회 시 사용할 Primary Key 값은 URLconf에서 추출하여 뷰로 넘어온 파라미터를 사용합니다.

그리고 명시적으로 지정하지 않아도 장고에서 디폴트로 알아서 지정해주는 속성이 2가지 있습니다. 첫 번째는 컨텍스트 변수로 **object**를 사용하는 것이고, 두 번째는 템플릿 파일을 **모델명 소문자_detail.html** 형식의 이름으로 지정하는 것입니다.

⓭ Book 테이블로부터 특정 레코드를 가져와 object라는 컨텍스트 변수를 구성합니다. 템플릿 파일은 디폴트로 books/book_detail.html 파일이 됩니다. 테이블 조회 조건에 사용되는 Primary Key 값은 URLconf에서 넘겨받는데, 이에 대한 처리는 DetailView 지네릭 뷰에서 알아서 처리해줍니다.

⓮ Author 테이블로부터 특정 레코드를 가져와 object라는 컨텍스트 변수를 구성합니다. 템플릿 파일은 디폴트로 books/author_detail.html 파일이 됩니다.

⓯ Publisher 테이블로부터 특정 레코드를 가져와 object라는 컨텍스트 변수를 구성합니다. 템플릿 파일은 디폴트로 books/publisher_detail.html 파일이 됩니다.

지금까지 지네릭 뷰에 대해 살펴보았는데, 지네릭 뷰의 강력함이 보이나요? 특히 데이터베이스 객체의 리스트를 보여주거나, 특정 객체의 상세 내용을 보여주는 작업을 코딩할 때 지네릭 뷰의 장점이 확실히 드러납니다. 만일 동일한 로직을 직접 코딩한다면 테이블에 접속하고, 쿼리 조건을 지정하고, 테이블로부터 가져온 결과를 컨텍스트 변수에 담아서 템플릿 시스템에 넘겨줘야 합니다. 이

러한 복잡한 로직을 장고에서 모두 처리해주고, 개발자는 단 2줄로 코딩을 완료하였습니다. 코딩 과정에서의 버그 가능성도 크게 줄였습니다. 이것이 바로 장고의 큰 장점입니다.

5.1.8 지금까지 작업 확인하기

지금까지 클래스형 지네릭 뷰를 사용하여 books 애플리케이션 코딩을 마쳤습니다. 지금까지의 작업이 정상적으로 완료되었는지 확인하기 위해 우선 실습에 필요한 데이터를 입력하겠습니다.

runserver를 실행하고 브라우저로 Admin 사이트에 접속합니다. 아래처럼 Admin 사이트의 첫 화면이 나오면, Books, Authors, Publishers 항목의 [Add] 버튼을 클릭하여 테이블에 데이터를 입력합니다.

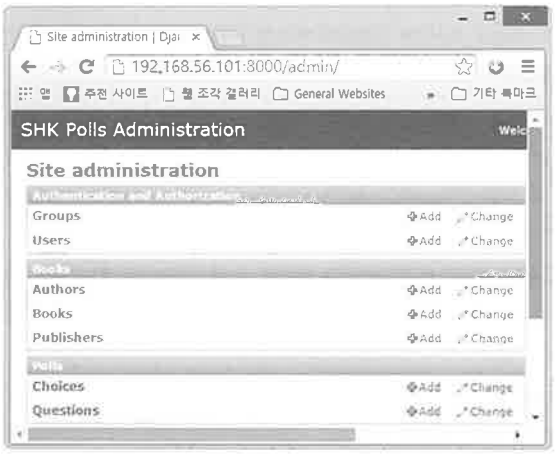


그림 5-5 books 애플리케이션 – Admin 첫 화면

Book, Author, Publisher 테이블에 다음과 같이 각각 3개의 레코드를 입력합니다.

표 5-5 Book 테이블에 입력할 데이터

Title	Authors	Publisher	Publication date
Python Web Programming	Kim Seok Hun	Hanbit Media, Inc.	2015-03-01
Design Patterns	Eric Gamma	O'Reilly	2005-12-25
Effective Java	Joshua Bloch	Pearson Education, Inc.	2008-09-02

표 5-6 Author 테이블에 입력할 데이터

Name	Salutation	Email	Publication date
Kim Seok Hun	I'm a python programmer	shkshya@daum.net	2015-03-01
Eric Gamma	Welcome to Gang of Four	ericgamma@gmail.com	2005-12-25
Joshua Bloch	Java Great Programer	joshua@gmail.com	2008-09-02

표 5-7 Publisher 테이블에 입력할 데이터

Name	Address	Website
Hanbit Media, Inc.	Seoul, Korea	http://www.hanb.co.kr/
O'Reilly	Sanfrancisco, US	http://www.oreilly.com/
Pearson Education, Inc.	United States	http://pearson.com/

데이터를 모두 입력했으면, books 애플리케이션으로 접속해보겠습니다.

```
http://192.168.56.101:8000/books
```

books 애플리케이션의 첫 화면이 나타납니다. 정상적으로 동작하고 있는 것을 확인할 수 있습니다.

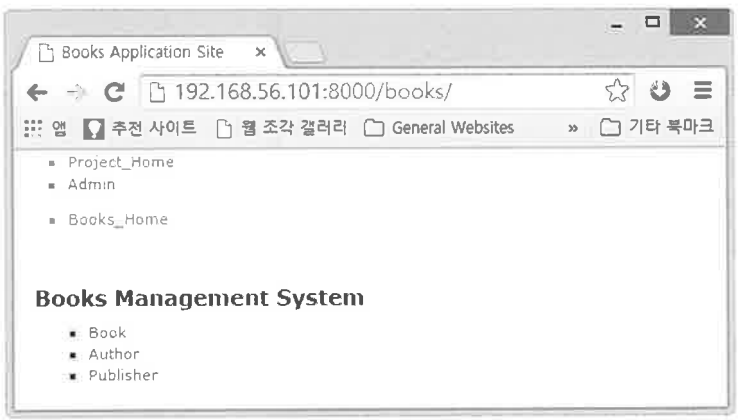


그림 5-6 books 애플리케이션 – 첫 화면

계속해서 Book, Author, Publisher 테이블을 클릭해보고, 다음에 나타나는 레코드 리스트 화면에서도 각 항목을 클릭해서 상세 정보 화면도 확인해봅니다. 또한, 위에 있는 네비게이션 항목도 클릭해서 정상적으로 해당 화면으로 이동하는지 확인해봅니다.

아래 정상적으로 처리되었을 때의 화면들을 보여주고 있으니 확인해보기 바랍니다.

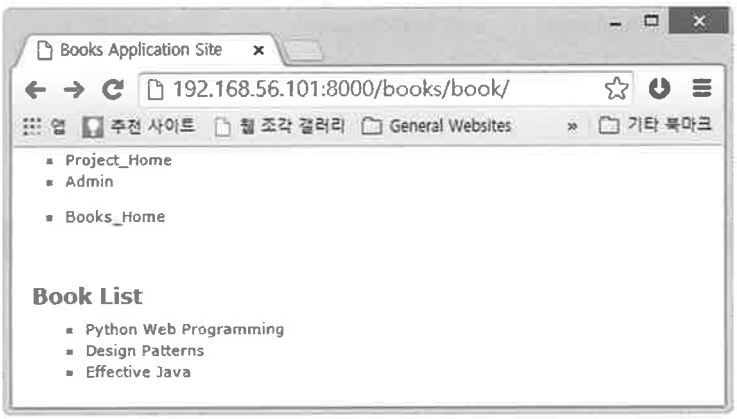


그림 5-7 books 애플리케이션 – 책 리스트 화면

그림 5-8 books 애플리케이션 – 책 상세 정보 화면

그런데 위에 있는 네비게이션 항목 중에서 Admin, Books_Home 항목은 잘 동작하지만, Project_Home 항목을 클릭하면 아래와 같은 에러 메시지가 나타납니다. 아직 이 기능은 코딩이 되지 않았기 때문입니다. 이 에러는 다음 절에서 살펴보도록 하겠습니다.

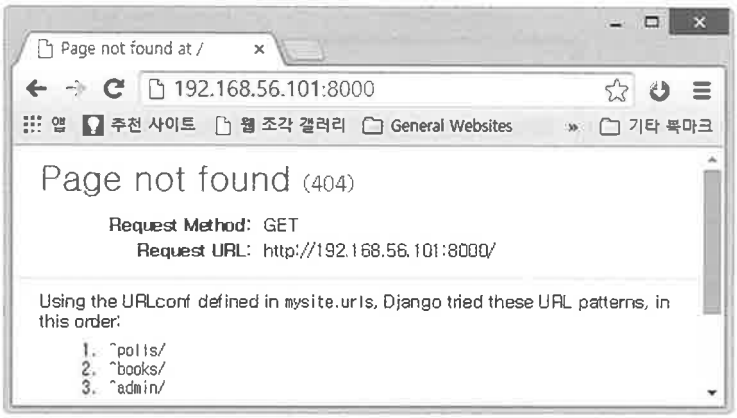


그림 5-9 books 애플리케이션 – Project_Home 에러 화면

5.2 프로젝트 첫 페이지 만들기

지금까지 mysite라는 프로젝트 하위에 polls 애플리케이션과 books 애플리케이션을 만들었습니다. 이 2개의 애플리케이션을 만들면서 각 애플리케이션의 첫 페이지인 /polls/ 및 /books/ URL에 대한 처리 로직은 개발했지만, 정작 프로젝트의 첫 페이지인 루트(/) URL에 대한 처리 로직은 개발하지 못한 상태입니다.

그래서 여기서는 프로젝트 첫 페이지인 루트(/) URL에 대한 처리 로직을 코딩하는 실습을 진행하도록 하겠습니다.

5.2.1 프로젝트 첫 페이지 설계

5.1.8 지금까지 작업 확인하기에서 화면의 네비게이션 항목 중 [Project_Home] 항목을 클릭했을 때 에러가 발생한 것도 바로 루트(/) URL에 대한 처리 로직이 없기 때문입니다. 간단하지만 코딩에 들어가기 전에 개발할 로직을 설계하면 아래 그림과 같습니다. 테이블은 변경사항이 없으므로, 화면 UI 및 뷰의 흐름만 설계하면 됩니다.

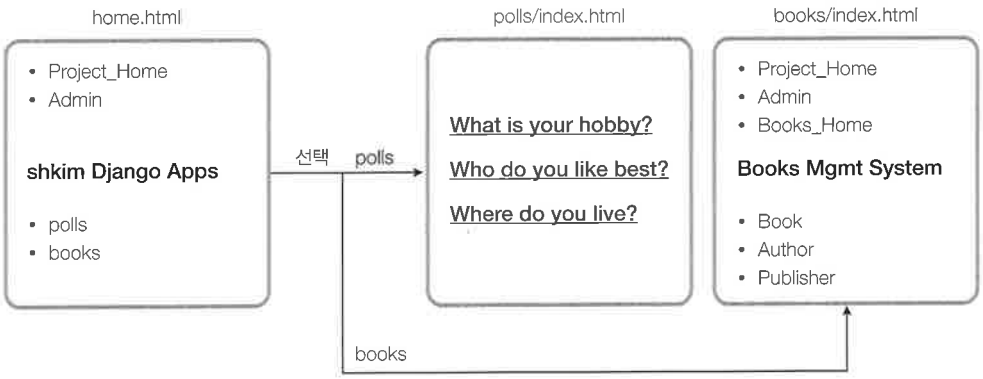


그림 5-10 프로젝트 첫 페이지 – UI 설계

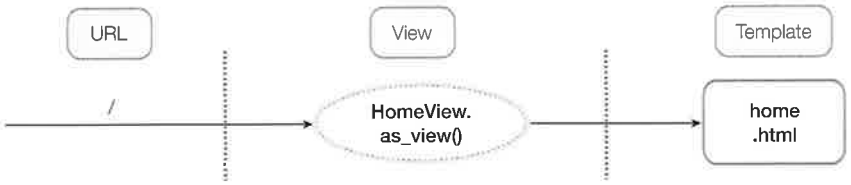


그림 5-11 프로젝트 첫 페이지 – 뷰 흐름 설계

5.2.2 URLconf 코딩하기

테이블은 변경사항이 없으므로 모델 코딩은 불필요합니다. 바로 URLconf 코딩부터 시작하겠습니다. 역시 위에서 설계된 뷰 흐름의 내용을 참고해서 URLconf를 정의하면 됩니다. 애플리케이션에 대한 URL이 아니라 프로젝트에 대한 URL이므로 mysite/urls.py 파일에 루트(/) URL 및 임포트 문장을 두 줄만 추가하면 됩니다. 뷰 이름은 HomeView라고 정의하였습니다.

예제 5-17 프로젝트 첫 페이지 – mysite/urls.py 코딩 소스제공

```
$ cd /home/shkim/pyBook/ch5/mysite
$ vi urls.py

from django.conf.urls import patterns, include, url
from django.contrib import admin
from mysite import views # 추가

urlpatterns = patterns('',
```

```
    url(r'^$', views.HomeView.as_view(), name='home'), # 추가
    url(r'^polls/', include('polls.urls', namespace="polls")),
    url(r'^books/', include('books.urls', namespace="books")),
    url(r'^admin/', include(admin.site.urls)),
)
```

5.2.3 Template 코딩하기

home.html 템플릿은 개별 애플리케이션 템플릿이 아니라 프로젝트 템플릿이므로, 상속에 사용하는 부모 템플릿의 위치와 동일하게 다음의 디렉토리에 생성합니다.

```
/home/shkim/pyBook/ch5/templates
```

home.html 템플릿에서는 base_books.html 템플릿이 아니라 base.html 템플릿을 상속받고 있다는 점을 유의하기 바랍니다.

예제 5-18 프로젝트 첫 페이지 - home.html 코딩 소스제공

```
$ cd /home/shkim/pyBook/ch5/templates
$ vi home.html

{% extends "base.html" %}

{% block content %}
    <h2>shkim Django Applications</h2>
    <ul>
        {% for appname in object_list %}
{% with appname|add:":"|add:"index" as urlvar %}
            <li><a href="{% url urlvar %}">{{ appname }}</a></li>
{% endwith %}
        {% endfor %}
    </ul>
{% endblock content %}
```

뷰로부터 object_list 컨텍스트 변수를 전달받아서 object_list에 들어 있는 애플리케이션명(appname)들을 하나씩 순회하면서 화면에 보여주고 있습니다. 또한, 애플리케이션명을 클릭

시 접속할 URL을 추출하기 위해 {% url urlvar %} 템플릿 태그를 사용하였고, urlvar 인자는 {% with %} 태그를 사용하여 다음과 같이 정의하고 있습니다.

```
{% with appname|add:":"|add:"index" as urlvar %}
```

이 문장은 add 템플릿 필터를 사용하여 애플리케이션명에 필요한 문자열을 붙여주고 있습니다. 예를 들어, 애플리케이션명이 books라면 urlvar는 books:index가 될 것입니다.

5.2.4 View 코딩하기

뷰 이름은 앞에서 URLconf를 코딩하면서 HomeView라고 정의하였습니다. 파일의 위치는 어디가 좋을까요? 애플리케이션이 아니라 프로젝트와 관련된 뷰이므로, mysite/views.py 파일에 코딩하는 것이 적절합니다. 아래와 같이 내용을 입력합니다.

예제 5-19 프로젝트 첫 페이지 - mysite/views.py 코딩 **소스제공**

```
$ cd /home/shkim/pyBook/ch5/mysite
$ vi views.py

# Create your views here.
from django.views.generic.base import TemplateView ----------①

#— TemplateView
class HomeView(TemplateView): ----------②

    template_name = 'home.html' ----------③

    def get_context_data(self, **kwargs):
        context = super(HomeView, self).get_context_data(**kwargs) ----------④
        context['object_list'] = ['polls', 'books'] ----------⑤
        return context ----------⑥
```

위 HomeView 내용은 [예제 5-16]에서 본 books 애플리케이션의 BooksModelView와 거의 동일합니다. 특별한 로직 없이 템플릿만 렌더링하는 로직이므로 TemplateView 지네릭 뷰를 상속받고 있습니다. 위 소스를 라인별로 설명하면 다음과 같습니다.

❶ 클래스형 지네릭 뷰를 사용하기 위해 TemplateView 클래스를 임포트합니다.

❷ TemplateView 지네릭 뷰를 상속받아 사용하고 있습니다. TemplateView를 사용하는 경우에는 필수적으로 template_name 클래스 변수를 오버라이딩해서 지정해줘야 합니다. 템플릿 시스템으로 넘겨줄 컨텍스트 변수가 있는 경우에는 get_context_data() 메소드를 오버라이딩해서 정의해주면 됩니다.

❸ mysite 프로젝트의 첫 화면을 보여주기 위한 템플릿 파일을 home.html로 지정하였습니다. 템플릿 파일이 위치하는 디렉토리는 settings.py 파일의 TEMPLATE_DIRS 항목으로 지정되어 있습니다.

❹ get_context_data() 메소드를 정의할 때는 반드시 첫 줄에 super() 메소드를 호출해야 합니다.

❺ mysite 프로젝트 하위에 있는 애플리케이션들의 리스트를 보여주기 위해 컨텍스트 변수 object_list에 담아서 템플릿 시스템에 넘겨주고 있습니다.

❻ return 문장도 필수입니다.

5.2.5 지금까지 작업 확인하기

프로젝트 첫 페이지 코딩을 완료하였으니 결과를 확인해보겠습니다. Runserver를 실행한 후에 브라우저를 통해 루트(/) URL로 접속합니다.

```
http://192.168.56.101:8000/
```

아래와 같이 나타나면 정상입니다. 계속해서 화면을 이동하면서 네비게이션 항목의 Project_Home 항목을 클릭해봅니다. 프로젝트 첫 화면으로 이동해서 아래 화면이 나타나는지 확인해보기 바랍니다.

그림 5-12 프로젝트 첫 페이지 - Project_Home 성공 화면

5.3 polls 애플리케이션 – 클래스형 뷰로 변경하기

앞서 3장에서 mysite 프로젝트 하위에 polls 애플리케이션을 개발한 바 있습니다. 처음으로 장고의 프로젝트 및 애플리케이션을 만드는 과정이었기 때문에 비교적 이해하기 쉬운 함수형 뷰로 코딩하였습니다. 또한, 5장에서는 클래스형 뷰를 사용하여 books 애플리케이션을 만들어 보았습니다. 이번에는 이런 경험을 바탕으로 polls 애플리케이션을 클래스형 뷰로 변경하여 개발해보겠습니다. 이번 실습을 통해서 클래스형 뷰에 좀 더 친숙해지고, 함수형 뷰와 클래스형 뷰의 차이점에 대한 이해도 깊어질 것이라 생각합니다.

polls 애플리케이션에 대한 설계는 이미 3장에서 진행했던 내용을 참고하면 되므로, 설계 단계는 생략하고 코딩으로 바로 들어가겠습니다. 설계 내용에 조금 달라지는 부분이 있지만 사소한 것이므로 그 부분에 대해서는 코딩하면서 설명하도록 하겠습니다. 이번 실습 역시 테이블에는 변경사항이 없으므로 모델 코딩은 불필요하고, 바로 URLconf 코딩부터 시작하면 됩니다.

5.3.1 URLconf 코딩하기

URLconf에 대한 코딩은 기존에 URL 패턴별 함수형 뷰로 매핑했던 사항을 아래 표처럼 클래스형 뷰로 변경해서 매핑해주면 됩니다.

표 5-8 polls 애플리케이션 변경 – URL과 클래스형 뷰 매핑

URL 패턴	기존 뷰 이름 (함수형 뷰)	새로운 뷰 이름 (클래스형 뷰)	변경사항(템플릿 파일명은 동일함)
/polls/	index( )	IndexView	뷰와 템플릿 모두 변경함(index.html)
/polls/5/	detail( )	DetailView	뷰와 템플릿 모두 변경함(detail.html)
/polls/5/results/	results( )	ResultsView	뷰와 템플릿 모두 변경함(results.html)
/polls/5/vote/	vote( )	vote( )	뷰와 템플릿 모두 변경사항 없음

polls/urls.py 파일을 다음과 같이 수정합니다.

예제 5-20 polls 애플리케이션 변경 – polls/urls.py 코딩 소스제공

```
$ cd /home/shkim/pyBook/ch5/polls
```

```
$ vi urls.py

from django.conf.urls import patterns, url
from polls import views

urlpatterns = patterns('',
    url(r'^$', views.IndexView.as_view(), name='index'), # /polls/
    url(r'^(?P<pk>\d+)/$', views.DetailView.as_view(), name='detail'), # /polls/5/
    url(r'^(?P<pk>\d+)/results/$', views.ResultsView.as_view(), name='results'),
    url(r'^(?P<question_id>\d+)/vote/$', views.vote, name='vote'), # /polls/5/vote/
)
```

❶ ❷

기존 소스 대비 변경된 라인은 다음과 같습니다.

❶ 뷰 이름이 클래스형 뷰로 변경되었습니다.

❷ URL 패턴의 파라미터 이름이 〈pk〉로 변경되었습니다. 이는 DetailView 지네릭 뷰의 동작 방식 때문입니다. 즉, 테이블의 특정 레코드를 조회하는 경우 Primary Key로 검색을 하는데, Primary Key에 대한 변수명을 pk라고 사용하기 때문입니다.

5.3.2 Template 코딩하기

템플릿의 주요 변경사항은 기존에는 사용하지 않았던 상속 기능을 추가하는 것입니다. 이미 books 애플리케이션을 개발할 때 부모 템플릿이 되는 base.html을 코딩하였기 때문에 이를 상속받은 base_polls.html 템플릿 파일을 만들고, 기존 각 템플릿 파일에서 base_polls.html 템플릿을 상속받으면 됩니다.

먼저 base_polls.html 템플릿을 아래 예제처럼 코딩합니다. [예제 5-14]의 base_books.html 템플릿과 거의 동일합니다. 파일의 위치는 이제 알겠죠? 맞습니다. base_books.html 파일 위치와 동일한 디렉토리입니다. 아래 예제에 cd로 표시했습니다.

예제 5-21 polls 애플리케이션 변경 - base_polls.html 코딩 소스제공

```
$ cd /home/shkim/pyBook/ch5/templates
$ vi base_polls.html
```

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

<title>{% block title %}Polls Application Site{% endblock %}</title>  ❶

{% block sidebar %}
{{ block.super }}
<ul>
    <li><a href="/polls/">Polls_Home</a></li>  ❷
</ul>
{% endblock %}
```

위 파일은 polls 애플리케이션에 새로 추가되는 파일입니다. 이해하기 쉽도록 base_books.html 템플릿 파일과 비교해서 라인별로 다른 점을 살펴보면 다음과 같습니다.

❶ 페이지 타이틀이 변경되었습니다.

❷ 내비게이션에 사용되는 문구를 Polls_Home으로 정했습니다.

다음은 각 페이지별 템플릿 파일에 대한 수정 사항을 코딩하겠습니다. 먼저 polls/index.html 템플릿 파일을 다음과 같이 변경합니다.

예제 5-22 polls 애플리케이션 변경 - polls/index.html 코딩 소스제공

```
$ cd /home/shkim/pyBook/ch5/polls/templates/polls
$ vi index.html

{% extends "base_polls.html" %}  ❶

{% block content %}  ❷

<h2>Polls Question List</h2>  ❸

{% if latest_question_list %}
    <ul>
    {% for question in latest_question_list %}
        <li><a href="{% url 'polls:detail' question.id %}">{{ question.
question_text }}</a></li>  ❹
    {% endfor %}
    </ul>
```

```
{% else %}
    <p>No polls are available.</p>
{% endif %}

{% endblock content %}
```

❷

기존 소스 대비 변경된 라인은 다음과 같습니다.

❶ base_polls.html 템플릿을 상속받습니다.

❷ content 블록을 재정의합니다.

❸ 페이지 제목을 〈h2〉 폰트 크기로 표시합니다.

❹ 이하는 기존 소스와 동일합니다.

polls/detail.html 템플릿 파일을 다음과 같이 변경합니다.

예제 5-23 polls 애플리케이션 변경 - polls/detail.html 코딩 소스제공

```
$ cd /home/shkim/pyBook/ch5/polls/templates/polls
$ vi detail.html

{% extends "base_polls.html" %}

{% block content %}

<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>

{% endblock content %}
```

❷ ❸ ❷

기존 소스 대비 변경된 라인은 다음과 같습니다.

❶ base_polls.html 템플릿을 상속받습니다.

❷ content 블록을 재정의합니다.

❸ 이하는 기존 소스와 동일합니다.

polls/results.html 템플릿 파일을 다음과 같이 변경합니다.

예제 5-24 polls 애플리케이션 변경 - polls/results.html 코딩 소스제공

```
$ cd /home/shkim/pyBook/ch5/polls/templates/polls
$ vi results.html

{% extends "base_polls.html" %}                                    ❶

{% block content %}                                                ❷

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

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

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

{% endblock content %}                                             ❷
```

기존 소스 대비 변경된 라인은 다음과 같습니다.

❶ base_polls.html 템플릿을 상속받습니다.

❷ content 블록을 재정의합니다.

❸ 이하는 기존 소스와 동일합니다.

5.3.3 View 코딩하기

URLconf 또는 템플릿 변경사항은 그리 많지 않지만, 함수형 뷰에서 클래스형 뷰로 변경되는 사항은 그보다 많은 편입니다. 그런데 클래스형 지네릭 뷰를 사용할 것이므로 코딩량은 오히려 줄어듭니다.

클래스형 뷰를 코딩할 때 가장 먼저 고려해야 할 사항이 어떤 지네릭 뷰를 사용할 것이냐 하는 것입니다. 개발하고자 하는 애플리케이션의 로직을 분석해보고 가장 적합한 지네릭 뷰를 찾을 수 있어야 합니다. 예제의 polls 애플리케이션은 간단한 로직이므로 어려운 편은 아닙니다. 아래 표와 같이 지네릭 뷰를 선택하여 사용하겠습니다.

표 5-9 polls 애플리케이션 변경 - 지네릭 뷰 선택

URL 패턴	기존 뷰 이름 (함수형 뷰)	새로운 뷰 이름 (클래스형 뷰)	지네릭 뷰 선택
/polls/	index( )	IndexView	질문 리스트를 보여주는 로직이므로, ListView를 사용함
/polls/5/	detail( )	DetailView	질문 하나에 대한 세부 정보를 보여주는 로직이므로, DetailView를 사용함
/polls/5/results/	results( )	ResultsView	투표 결과도 각 질문에 대한 세부 정보에 해당하므로, DetailView를 사용함
/polls/5/vote/	vote( )	vote( )	뷰와 템플릿 모두 변경사항 없음

polls/views.py 파일을 다음과 같이 변경합니다.

예제 5-25 polls 애플리케이션 변경 - polls/views.py 코딩 소스제공

```
$ cd /home/shkim/pyBook/ch5/polls
$ vi views.py

from django.shortcuts import get_object_or_404, render
from django.http import HttpResponseRedirect
from django.core.urlresolvers import reverse
from django.views.generic import ListView, DetailView  ----------①

# Create your views here.
from polls.models import Choice, Question
```

```
#-- Class-based GenericView
class IndexView(ListView):                                          ❷
    template_name = 'polls/index.html'                              ❸
    context_object_name = 'latest_question_list'                    ❹

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

class DetailView(DetailView):                                       ❻
    model = Question                                                ❼
    template_name = 'polls/detail.html'                             ❽

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

#-- Funtion-based View
def vote(request, question_id):
    p = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = p.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # 설문 투표 폼을 다시 보여준다
        return render(request, 'polls/detail.html', {
            'question': p,
            'error_message': "You didn't select a choice.",         ⓬
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # POST 데이터를 정상적으로 처리하였으면,
        # 항상 HttpResponseRedirect를 반환하여 리다이렉션 처리함
        return HttpResponseRedirect(reverse('polls:results', args=(p.id,)))
```

기존 소스 대비 변경된 라인은 다음과 같습니다.

❶ 클래스형 지네릭 뷰를 사용하기 위해 ListView, DetailView 클래스를 임포트합니다.

❷ IndexView 클래스는 ListView 지네릭 뷰를 사용하고 있습니다. ListView를 상속받는 경우는 객체가 들어 있는 리스트를 구성해서 이를 컨텍스트 변수로 템플릿 시스템에 넘겨주면 됩니다. 만일 이런 리스트를 테이블에 들어 있는 모든 레코드를 가져와 구성하는 경우에는 테이블명, 즉 모델 클래스명만 지정해주면 됩니다. 그렇지 않은 경우에는 get_queryset() 메소드를 오버라이딩으로 정의하여 원하는 리스트를 구성해주면 됩니다.

컨텍스트 변수명과 템플릿 파일명은 디폴트 값을 사용할 수도 있고, 명시적으로 지정해줄 수도 있습니다.

❸ 템플릿 파일명을 디폴트로 사용하지 않고, polls/index.html로 지정해주고 있습니다.

❹ 컨텍스트 변수명을 디폴트로 사용하지 않고, latest_question_list로 지정해주고 있습니다.

❺ 컨텍스트 변수의 리스트를 구성하기 위해 get_queryset() 메소드를 오버라이딩해서 정의하고 있습니다. Question 테이블에서 pub_date 컬럼 기준으로 최신 5개를 조회하여 리스트를 구성하고 있습니다.

❻ DetailView 클래스는 DetailView 지네릭 뷰를 사용하고 있습니다. DetailView를 상속받는 경우는 특정 객체 하나를 컨텍스트 변수에 담아서 템플릿 시스템에 넘겨주면 됩니다. 만일 특정 객체를 테이블에서 Primary Key로 조회해서 가져오는 경우에는 테이블명, 즉 모델 클래스명만 지정해주면 됩니다.
테이블 조회 조건에 사용되는 Primary Key 값은 URLconf에서 pk라는 파라미터 이름으로 넘겨받는데, 이에 대한 처리는 DetailView 지네릭 뷰에서 알아서 처리해줍니다.

컨텍스트 변수명과 템플릿 파일명은 디폴트 값을 사용할 수도 있고, 명시적으로 지정해줄 수도 있습니다.

❼ Question 테이블로부터 특정 레코드를 가져와 컨텍스트 변수를 구성합니다. 컨텍스트 변수명은 디폴트 값인 object를 사용합니다.

❽ 템플릿 파일명을 디폴트로 사용하지 않고, polls/detail.html로 지정해주고 있습니다.

❾ ResultsView 클래스는 DetailView 지네릭 뷰를 사용하고 있습니다. DetailView를 상속받는 경우는 특정 객체 하나를 컨텍스트 변수에 담아서 템플릿 시스템에 넘겨주면 됩니다. 만일 특정 객체를 테이블에서 Primary Key로 조회해서 가져오는 경우에는 테이블명, 즉 모델 클래스명만 지정해주면 됩니다.

테이블 조회 조건에 사용되는 Primary Key 값은 URLconf에서 pk라는 파라미터 이름으로 넘겨받는데, 이에 대한 처리는 DetailView 지네릭 뷰에서 알아서 처리해줍니다.

컨텍스트 변수명과 템플릿 파일명은 디폴트 값을 사용할 수도 있고, 명시적으로 지정해줄 수도 있습니다.

❿ Question 테이블로부터 특정 레코드를 가져와 컨텍스트 변수를 구성합니다. 컨텍스트 변수명은 디폴트 값인 object를 사용합니다.

⓫ 템플릿 파일명을 디폴트로 사용하지 않고, polls/results.html로 지정해주고 있습니다.

⓬ 이하는 기존 소스와 동일합니다.

vote() 뷰 함수도 지네릭 뷰로 변경할 수 있습니다. 클래스형 지네릭 뷰에 익숙해졌다고 생각되면 여러분이 스스로 시도해보는 것도 좋은 공부가 될 것입니다.

5.3.4 로그 추가하기

4장에서 공부한 로깅 시스템을 적용하여 로그가 정상적으로 기록되는지 확인해보겠습니다. 로그를 남기기 위해서는 우선 settings.py 파일에 로깅 설정을 해줘야 합니다. 그리고 나서 로그 기록을 원하는 곳에서 로깅 메소드를 호출하면 됩니다.

로깅 설정을 위하여 mysite/settings.py 파일의 끝에 다음 사항을 추가합니다.

예제 5-26 mysite/settings.py에 로깅 설정 추가

```
$ cd /home/shkim/pyBook/ch5/mysite
$ vi settings.py

TEMPLATE_DIRS = [os.path.join(BASE_DIR, 'templates')]
# 상단 내용 동일

# logging
LOGGING = {
    'version': 1,                                                    ❶
    'disable_existing_loggers': False,                               ❷
    'formatters': {
        'verbose': {
            'format' : "[%(asctime)s] %(levelname)s [%(name)s:%(lineno)s]
                                        %(message)s",                ❸
            'datefmt' : "%d/%b/%Y %H:%M:%S"
        },
    },
    'handlers': {
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': os.path.join(BASE_DIR, 'logs/logfile'),      ❹
            'formatter': 'verbose'
        },
    },
    'loggers': {
        'polls': {
            'handlers': ['file'],
            'level': 'DEBUG',                                        ❺
        },
    }
}
```

위 로깅 설정 내용을 라인별로 설명하겠습니다.

❶ 설정이 dictConfig version 1 형식인데, 현재는 버전이 하나뿐입니다.

❷ 기존의 로깅 설정을 그대로 유지하면서 이하의 설정 사항을 추가합니다.

❸ verbose 포맷터를 정의합니다. verbose 포맷터는 [로그 메시지를 기록한 시간], 로그 레벨 이름, [로거이름:라인번호], 로그 메시지 순서로 출력합니다.

로그 메시지 기록 시간에 대한 포맷은 **날짜/월축약형/연도 시(24시기준):분:초** 형식으로 출력합니다.

❹ file 핸들러를 정의합니다. file 핸들러는 DEBUG 및 그 이상의 메시지를 파일로 출력해주는 FileHandler 클래스를 사용합니다. FileHandler 클래스에 의해서 로그가 기록되는 파일 이름은 /home/shkim/pyBook/ch5/logs/logfile입니다. 또한 위에서 정의한 verbose 포맷터를 사용합니다.

❺ polls 로거를 정의합니다. polls 로거는 DEBUG 및 그 이상의 메시지를 file 핸들러에게 보내줍니다. 로거에서 level을 정의하면 이는 핸들러에서 정의한 level을 오버라이딩합니다.

다음은 로그를 기록하기 위한 작업으로, polls/views.py 파일에 다음 사항을 추가합니다.

예제 5-27 polls/views.py에 로거 사용

```
$ cd /home/shkim/pyBook/ch5/mysite
$ vi settings.py

from polls.models import Choice, Question
# 상단 내용 동일

# logging 추가
import logging                                                    ❶
logger = logging.getLogger(__name__)                              ❷

# 중간 생략

def vote(request, question_id):
    logger.debug("vote().question_id: %s" % question_id)    # 추가   ❸
    p = get_object_or_404(Question, pk=question_id)
# 하단 내용 동일
```

위 소스를 라인별로 설명하면 다음과 같습니다.

❶ 로깅을 위하여 파이썬의 logging 모듈을 임포트합니다.

❷ getLogger(__name__) 메소드를 호출하여 polls.views 로거 객체를 취득합니다. __name__은 파이썬 식별자로서, 모듈 이름을 담고 있는 파이썬 내장 변수입니다. 즉, views.py 파일의 모듈명은 polls.views이고, 이것이 우리가 사용하고자 하는 로거 객체의 이름입니다.

❸ 로거 객체의 debug() 메소드를 호출하여 로거에게 DEBUG 수준으로 로그 메시지를 기록하도록 요청합니다. 로거는 앞에서 수정한 settings.py 파일의 로깅 설정에 따라, file 핸들러를 사용하여 /home/shkim/pyBook/ch5/logs/logfile 파일에 메시지를 기록합니다.

만일 /home/shkim/pyBook/ch5/logs/ 디렉토리가 없다면 만들어줘야 합니다.

```
$ cd /home/shkim/pyBook/ch5
$ mkdir logs
```

5.3.5 지금까지 작업 확인하기

이제 polls 애플리케이션을 포함한 mysite 프로젝트 개발을 완료하였습니다. 수고하셨습니다. 전체 프로젝트가 정상적으로 동작하는지 확인해보겠습니다.

runserver가 실행되지 않았다면 실행한 후에 브라우저를 통해 루트(/) URL로 접속합니다.

```
http://192.168.56.101:8000/
```

각 화면에 나오는 링크를 클릭하면서 polls 및 books 애플리케이션, Admin 사이트까지 이동해 보기 바랍니다. 또한 앞 절에서 추가한 로그 기능 동작을 확인하기 위해서 polls 애플리케이션에서 투표도 진행해보기 바랍니다.

다음 그림에서 정상적인 화면을 보여주고 있으니 참고해서 화면이 이상하거나 혹은 버그가 발생하지는 않는지 확인해보기 바랍니다. 만일 버그가 발생한다면 이를 수정하고, 더 나아가서 지금까지의 예제를 자신이 원하는 기능으로 개선하는 과정을 연습한다면 장고의 강력함을 실감할 수 있을 뿐만 아니라 여러분의 실력은 훨씬 더 향상되어 있을 것입니다.

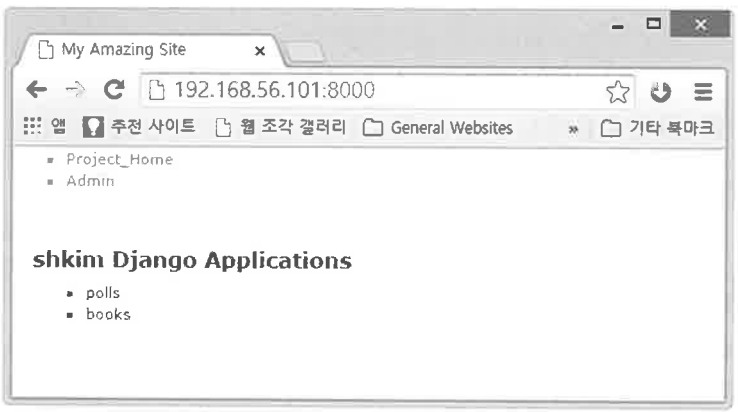


그림 5-13 mysite 프로젝트 첫 화면

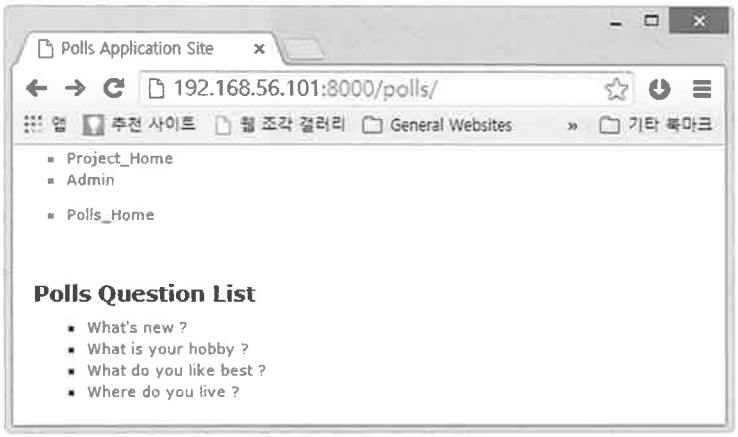


그림 5-14 polls 애플리케이션 첫 화면

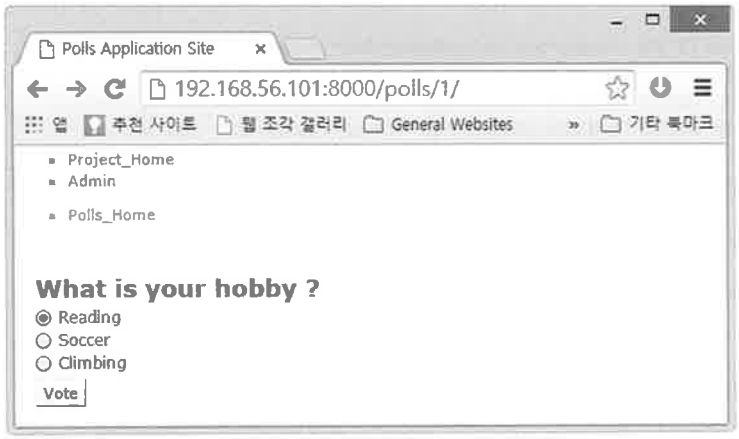


그림 5-15 polls 애플리케이션 – 투표 화면

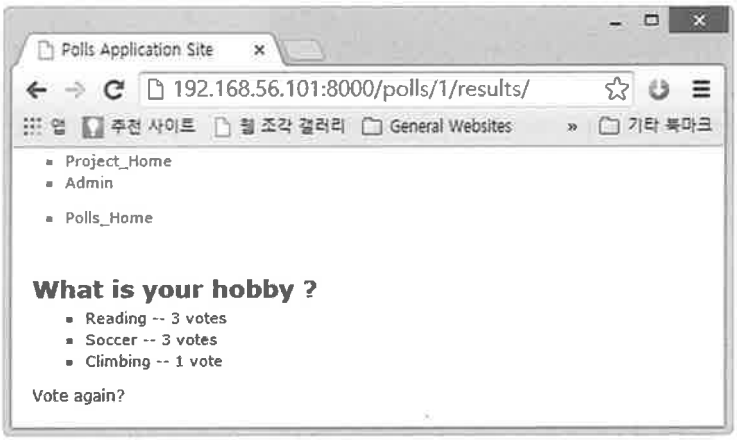


그림 5-16 polls 애플리케이션 – 투표 결과 화면

```
[shkim@localhost pyBook]$
[shkim@localhost pyBook]$ cd /home/shkim/pyBook/ch5/logs
[shkim@localhost logs]$ cat logfile
[24/Jan/2015 16:28:55] DEBUG [polls.views:32] vote().question_id: 2
[24/Jan/2015 18:40:00] DEBUG [polls.views:32] vote().question_id: 3
[24/Jan/2015 18:41:23] DEBUG [polls.views:32] vote().question_id: 1
[shkim@localhost logs]$
```

그림 5-17 polls 애플리케이션 – 로그 결과

CHAPTER 06

웹 서버(Apache)와 연동

장고를 사용하여 웹 애플리케이션을 개발한 후에 이를 실제로 서비스하기 위해서는 운영 환경에 개발한 프로그램을 배포하고 실행해야 합니다. 이렇게 개발 환경에서 운영 환경으로 옮겨가기 위해서는 우리가 개발 시 지정했던 설정 사항을 몇 가지 변경해야 합니다. 또한, 운영 환경의 웹 서버에서도 우리가 만든 애플리케이션을 인식할 수 있도록 설정 사항 변경이 필요합니다.

6장에서는 상용 웹 서버로 현재 가장 널리 사용되는 아파치에서 장고 애플리케이션을 실행하기 위해 필요한 사항들을 설명하겠습니다. 개발 환경과 운영 환경의 차이점을 이해하고, 이에 따라 설정 사항을 변경하는 것이 주요 작업입니다.

6.1 mod_wsgi 확장 모듈

아파치Apache 웹 서버의 프로그램명은 httpd이며, 전 세계에서 운영 중인 웹 사이트의 50% 이상이 사용할 정도로 인기있는 웹 서버입니다. 아파치는 추가로 필요한 기능을 모듈로 만들어 동적 로딩 방식으로 기능을 확장할 수 있는 특징이 있습니다. 클라이언트 요청 URL을 서버 내 디렉토리로 매핑해주는 mod_alias, 사용자 인증을 위한 mod_auth, 톰캣 연동에 사용되는 mod_jk, 프록시 기능을 제공하는 mod_proxy, URL rewrite를 지원하는 mod_rewrite, PHP 및 Perl 스크립트를 실행할 수 있는 mod_php, mod_perl 등의 수많은 확장 모듈이 사용되고 있습니다. 이번 장에서

설명하는 mod_wsgi도 파이썬 웹 애플리케이션을 실행할 수 있는 확장 모듈 중 하나입니다.

mod_wsgi는 파이썬 웹 애플리케이션 표준 규격인 WSGI[Web Server Gateway Interface]를 구현한 확장 모듈로서, 파이썬 웹 애플리케이션을 아파치에서 실행하는 데 사용합니다. 장고 프레임워크에서도 기본적으로 WSGI 스펙을 준수하므로, 아파치와 장고를 연동하기 위해서 mod_wsgi 모듈을 사용하도록 하겠습니다. WSGI의 스펙은 파이썬 표준 규격인 PEP[Python Enhancement Proposals] 333에 자세하게 정의되어 있습니다.

mod_wsgi를 사용해 웹 애플리케이션(또는 WSGI 애플리케이션이라고도 함)을 실행할 때, 두 가지 방식의 실행 모드를 제공합니다. 첫 번째는 내장[embedded] 모드로 WSGI 애플리케이션을 실행하는 방식입니다. 일반적인 아파치 자식 프로세스 컨텍스트 내에서 애플리케이션이 실행된다는 점에서 mod_python 방식과 유사합니다. 이 모드로 WSGI 애플리케이션을 구동하면 같은 아파치 웹 서버에서 실행되는 다른 웹 애플리케이션과 같은 아파치 자식 프로세스를 공유하게 됩니다. 내장 모드의 단점은 WSGI 애플리케이션의 소스가 변경되어 다시 적용하려면 아파치 전체를 재기동해야 한다는 것입니다. 재기동으로 인해 다른 서비스도 영향을 받게 되고, 경우에 따라서는 아파치 재기동 권한이 없을 수도 있습니다.

두 번째는 데몬[daemon] 모드로 유닉스 기반의 아파치 2.0 이상에서 지원됩니다. 이 모드는 WSGI 애플리케이션의 전용 프로세스에서 애플리케이션이 실행된다는 점에서 FCGI[Fast CGI]/SCGI[Simple CGI]와 유사합니다. 다른 점은 WSGI 애플리케이션을 구현할 때 별도의 프로세스 관리자나 WSGI 어댑터가 필요하지 않으며, 모든 처리는 mod_wsgi가 관리한다는 것입니다. WSGI 애플리케이션이 데몬 모드로 동작하면 일반적인 다른 아파치 자식 프로세스와 별도의 프로세스에서 동작합니다. 그래서 정적인 파일을 서비스하는 프로세스나 PHP, Perl 등의 아파치 모듈로 서비스하는 다른 애플리케이션에 미치는 영향이 미미합니다. 또한, 필요하다면 WSGI 애플이케이션 간에도 서로 영향을 주지 않도록 실행 유저를 달리하여 데몬 프로세스를 기동하는 것도 가능합니다.

성능적인 측면에서 mod_wsgi는 C 언어로 구현되어 있기 때문에 내부적으로 아파치가 직접 파이썬 API와 동작하므로 상대적으로 적은 메모리를 사용합니다. mod_wsgi의 이러한 구현방식 덕분에 아파치의 내장 파이썬 인터프리터가 동작하는 방식인 mod_python이나 FCGI/SCGI 같은 개선된 형태로, CGI에 비해서도 좋은 성능을 보이고 있습니다.

또한, 위에서 설명한 내장 모드와 데몬 모드의 동작 방식은 다르지만, 대용량 웹 애플리케이션에서의 처리 성능은 크게 다르지 않습니다. 처리 성능에 문제가 있다면 애플리케이션 자체 문제이거나

데이터베이스 처리로 인한 성능 저하가 더 큰 이슈가 됩니다. 장고에서는 안정성을 고려하여 내장 모드보다는 데몬 모드로 실행할 것을 권장하고 있습니다.

6.2 장고의 웹 서버 연동 원리

3장에서 장고 프로젝트의 뼈대를 만들 때 다음 명령을 실행하였습니다.

```
$ python manage.py startproject mysite
```

위 명령으로 장고의 여러 가지 기본 파일들과 함께 mysite/wsgi.py 파일이 만들어 집니다. 이 모듈이 바로 장고와 웹 서버를 연결하는 데 필요한 파일입니다. 이 모듈에는 WSGI 규격에 따라 호출 가능한callable 애플리케이션 객체를 정의하고 있습니다. 객체명은 반드시 **application**이어야 합니다. 물론 장고가 자동으로 만들어주므로 별도로 작성해줄 필요는 없습니다.

이 application 객체는 아파치와 같은 상용 웹 서버뿐만 아니라, 장고의 개발용 웹 서버인 runserver에서도 같이 사용하는 객체입니다. 다만 다른 점은 아파치에서는 httpd.conf 설정 파일에서 WSGIScriptAlias 지시자를 통해 지정하고, 개발용 runserver에서는 settings 모듈(mysite/settings.py)의 **WSGI_APPLICATION** 변수로 지정합니다.

웹 서버는 이 application 객체를 호출하여 장고의 애플리케이션을 실행하게 됩니다. 다만 application 객체를 호출하기 전에 현재의 프로젝트 및 프로젝트에 포함된 모든 애플리케이션들에 대한 설정 정보를 로딩하는 작업이 필요합니다. 이 설정 정보를 담고 있는 settings 모듈의 위치를 지정해주는 방법도 다른데, 아파치 등의 웹 서버는 wsgi.py 파일에서 지정해주고, 개발용 runserver는 다음과 같이 실행 옵션으로 지정해줄 수 있습니다. 별도로 지정해주지 않은 경우에는 디폴트로 **프로젝트명.settings**를 사용합니다.

```
# 상용 웹 서버 - mysite/wsgi.py 파일에서 지정
import os
os.environ['DJANGO_SETTINGS_MODULE'] = 'mysite.settings'
```

```
# 개발용 runserver - 실행 시 지정
$ python manage.py runserver —settings=mysite.settings
```

6.3 상용 서버 적용 전 장고의 설정 변경

아파치 등과 같은 웹 서버와 연동하는 단계는 장고의 애플리케이션 개발이 완료되고, 개발용 웹 서버인 runserver에서 정상적으로 동작하는 것을 확인한 이후 시점입니다. 즉, 지금까지는 개발 모드 환경에 맞춰 설정해왔지만, 상용 서버에 적용하기 위해서는 보안, 성능 등을 고려하여 상용 환경에 맞는 설정으로 변경해야 할 사항들이 있습니다.

개발 모드에서는 에러 발생 시 디버그를 위해 브라우저에 여러 가지 정보를 출력하여 보여주었습니다. 이런 디버그 정보는 프로젝트에 관련된 중요 정보들이므로, 상용 모드에서는 settings 모듈의 DEBUG 설정값을 False로 셋팅하여 디버그 정보가 노출되지 않도록 해야 합니다.

```
DEBUG = False
```

DEBUG = False로 설정되어 있으면, 반드시 settings 모듈의 ALLOWED_HOSTS 항목을 설정해야 합니다. 악의적인 공격자가 HTTP Host 헤더를 변조하여 CSRF[Cross Site Request Forgery] 공격을 할 수 있기 때문에 이를 방지하기 위한 것입니다.

```
ALLOWED_HOSTS = [ u'192.168.56.101' ]
```

개발 서버에서는 이미지, 자바스크립트, CSS 등의 정적 파일들을 알아서 찾아 주었지만, 상용 모드에서는 아파치와 같은 웹 서버가 정적 파일들이 어디에 있는지 알 수 있도록 해주어야 합니다. settings 모듈의 STATIC_ROOT 항목은 장고의 **collectstatic** 명령 실행 시 정적 파일들을 한 곳에 모아주는 디렉토리입니다. 여기서는 /home/shkim/pyBook/ch6/www_static 디렉토리에 정적 파일들을 모아줍니다. **6.4 내장 모드 실행**에서 보게 될 아파치 설정 파일에서도 **Alias /static/** 설정이 필요합니다.

```
STATIC_ROOT = os.path.join(BASE_DIR, "www_static")
```

collectstatic 명령은 다음과 같이 실행합니다. 이 명령을 사용 시 주의할 점은 settings 모듈의 STATICFILES_DIRS 항목에 STATIC_ROOT 항목에서 정의된 디렉토리가 포함되면 안 된다는 것입니다. **STATICFILES_DIRS** 항목에 정의된 디렉토리에서 정적 파일을 찾아 **STATIC_ROOT** 디렉토리에 복사해주기 때문입니다.

```
$ python manage.py collectstatic
```

개발 모드에서는 runserver를 실행시킨 사용자의 권한으로 데이터베이스 파일이나 로그 파일을 액세스합니다. 그러나 상용 모드에서는 웹 서버 프로세스의 권한자인 apache 사용자 권한으로 해당 파일들을 액세스할 수 있어야 합니다. 이를 위해 settings 모듈의 DATABASES 항목에서 NAME 속성값의 경로를 db/db.sqlite3로 변경해주고,

```
DATABASES = {
        . . .
        'NAME': os.path.join(BASE_DIR, 'db/db.sqlite3'),
```

해당 디렉토리 및 파일의 액세스 권한을 아래처럼 변경해줘야 합니다. SQLite 데이터베이스 파일의 위치를 옮기고, SQLite 데이터베이스가 있는 디렉토리 및 파일에 apache 사용자가 접근/읽기/쓰기 가능하도록 설정한 것입니다.

```
$ cd /home/shkim/pyBook/ch6
$ mkdir db
$ mv db.sqlite3 db/
$ chmod 777 db/
$ chmod 666 db/db.sqlite3
```

settings 모듈의 LOGGING 항목에 로깅 관련 사항이 정의되어 있고, 여기에 로그 파일의 위치가 설정되어 있습니다. 로그 파일에 apache 사용자가 읽기/쓰기 가능하도록 설정한 것입니다.

```
$ cd /home/shkim/pyBook/ch6
$ chmod 777 logs/
$ chmod 666 logs/logfile
```

NOTE_ 이 외에도 상용화하기 전 확인 사항에 대한 자세한 내용은 다음 페이지를 참고하기 바랍니다.

https://docs.djangoproject.com/en/1.7/howto/deployment/checklist/

6.4 내장 모드로 실행

mod_wsgi 모듈이 아파치 프로세스에 내장되어 실행되는 방식을 먼저 살펴보겠습니다.

6.4.1 아파치 설정

아파치 웹 서버에서 mod_wsgi 모듈을 이용해 파이썬 웹 애플리케이션을 실행할 수 있도록 아파치 설정 파일인 httpd.conf에 mod_wsgi 관련 설정을 추가해야 합니다. 그리고 아파치에 대한 설정 및 웹 서버(httpd) 실행은 루트root 권한으로 작업합니다.

예제 6-1 아파치 설정 - 내장 모드로 실행하는 경우

```
# cd /etc/httpd/conf
# vi httpd.conf

# 상단 내용 동일 ----------①
WSGIScriptAlias / /home/shkim/pyBook/ch6/mysite/wsgi.py ----------②
WSGIPythonPath /home/shkim/pyBook/ch6 ----------③

<Directory /home/shkim/pyBook/ch6/mysite>
<Files wsgi.py>
Require all granted ----------④
</Files>
</Directory>
```

```
Alias /static/ /home/shkim/pyBook/ch6/www_static/                    ❺
<Directory /home/shkim/pyBook/ch6/www_static>
Require all granted                                                  ❻
</Directory>
```

위 설정 내용을 라인별로 설명하겠습니다.

❶ 기존의 설정 내용은 변경사항이 없고, 파일의 끝에 다음 내용을 추가합니다.

❷ 아파치 웹 서버로 서비스하는 URL(/)과 wsgi.py 파일의 위치를 매핑해줍니다. 루트(/) URL로 시작하는 모든 요청은 wsgi.py 파일에서 정의된 WSGI application에서 처리한다는 의미입니다.

❸ 파이썬 임포트 경로를 지정합니다. 즉, import mysite 등의 문장이 정상으로 동작하도록 합니다.

❹ 아파치가 wsgi.py 파일을 액세스할 수 있도록 mysite 디렉토리 및 wsgi.py 파일에 대한 접근 권한을 설정합니다.

❺ /static/ URL에 대한 처리를 위해 static 파일이 위치한 디렉토리를 매핑해줍니다. 이 디렉토리는 장고의 collectstaic 명령에 의해 static 파일을 모아둔 디렉토리입니다. 이는 settings.py 파일의 STATIC_ROOT 항목에 정의된 디렉토리이기도 합니다.

❻ 아파치가 www_static 디렉토리에 액세스할 수 있도록 디렉토리 접근 권한을 설정합니다.

6.4.2 지금까지 작업 확인하기

mod_wsgi 모듈에 대한 아파치 설정이 끝나면 아파치를 기동해서 동작을 확인합니다.

아파치를 기동하기 전에 한 가지 할 일이 있습니다. 설정을 정확하게 했는데도 시스템에 따라 서비스가 안 되는 경우가 있습니다. 즉, 장고의 runserver를 실행하면 정상적으로 서비스가 되는데, 아파치 웹 서버로는 서비스가 안 되는 경우가 이에 해당됩니다. 이런 경우에는 SELinux[Security Enhanced Linux] 정책이 적용되어 보안 문제로 서비스가 안 되는 것이므로, SELinux 보안 정책을 변경해줘야 합니다.

```
# setenforce permissive
```

아파치 기동 명령은 다음과 같습니다.

```
# service httpd start
```

아파치를 기동한 후에 브라우저를 통해 루트(/) URL로 접속합니다. 아파치 웹 서버는 80 포트를 사용하는데, 포트번호는 생략할 수 있습니다.

```
http://192.168.56.101:80/
```

다음과 같이 프로젝트의 첫 화면이 나타나면 정상입니다.

그림 6-1 아파치 실행 – mysite 프로젝트 첫 화면

또한, 아파치의 접속 로그를 통해서도 정상적으로 처리된 것을 확인할 수 있습니다. access_log를 보면 루트(/) URL에 대한 GET 요청에 응답 코드 200으로 성공 응답을 보내주고 있습니다.

```
[root@localhost logs]# cd /etc/httpd/logs
[root@localhost logs]# tail access_log
192.168.56.1 - - [04/Jan/2015:00:53:32 +0900] "GET / HTTP/1.1" 200
ML, like Gecko) Chrome/39.0.2171.95 Safari/537.36"
```

그림 6-2 아파치 실행 – apache access_log

NOTE_ SELinux 운용 모드

SELinux 정책은 아래와 같이 세 가지 모드로 운용될 수 있습니다. 정책 변경은 setenforce 명령으로 할 수도 있고, /etc/selinux/config 파일에서도 변경할 수 있습니다. setenforce 명령으로 변경하면 현재 쉘에서 작업하는 동안만 임시적으로 변경하는 것이며, config 파일에서 변경하면 시스템 부팅 시에 운용 모드가 적용됩니다.

- **enforcing(1) :** SELinux 보안 정책이 실제로 동작하여 위반 시 처리에 실패함
- **permissive(0) :** SELinux 보안 정책이 동작하지는 않지만, 위반 상황 발생 시 경고(warning)로 알려줌
- **disable :** SELinux 보안 정책이 동작하지 않음. config 파일에서만 동작함

6.5 데몬 모드로 실행

장고에서는 안정성을 고려하여 내장 모드보다는 데몬 모드로 실행할 것을 권장하고 있습니다.

6.5.1 아파치 설정

내장 모드로 실행 시 설정했던 방법과 유사하게 아파치 설정 파일인 httpd.conf에 mod_wsgi 관련 설정을 추가해야 합니다.

예제 6-2 아파치 설정 - 데몬 모드로 실행하는 경우

```
# cd /etc/httpd/conf
# vi httpd.conf

# 상단 내용 동일 ------------------------------------------------------------ ❶
WSGIScriptAlias / /home/shkim/pyBook/ch6/mysite/wsgi.py ------------------- ❷
WSGIDaemonProcess mysite python-path=/home/shkim/pyBook/ch6 ---------------- ❸
WSGIProcessGroup mysite ---------------------------------------------------- ❹

<Directory /home/shkim/pyBook/ch6/mysite> ---------------------------------- ❺
<Files wsgi.py>
Require all granted
</Files>
</Directory> ---------------------------------------------------------------- ❺

Alias /static/ /home/shkim/pyBook/ch6/www_static/ -------------------------- ❻
<Directory /home/shkim/pyBook/ch6/www_static> ------------------------------ ❼
Require all granted
</Directory> ---------------------------------------------------------------- ❼
```

위 설정 내용을 라인별로 설명하겠습니다.

❶ 기존의 설정 내용은 변경사항이 없고, 파일의 끝에 다음 내용을 추가합니다.

❷ 아파치 웹 서버로 서비스하는 URL(/)과 wsgi.py 파일의 위치를 매핑해줍니다. 루트(/) URL로 시작하는 모든 요청은 wsgi.py 파일에서 정의된 WSGI application에서 처리한다는 의미입니다.

❸ 별도의 데몬 프로세스에서 장고를 실행하기 위해 프로세스 속성을 설정합니다. 위 예제에서는 파이썬의 모듈 임포트 경로를 지정하였습니다. 내장 모드에서 사용했던 WSGIPythonPath 지시자는 사용할 수 없습니다. 프로세스 속성으로 프로세스의 개수와 스레드의 개수 등도 지정할 수 있습니다.

❹ 프로세스 그룹을 지정합니다. 동일한 프로세스 그룹에 할당된 애플리케이션은 같은 데몬 프로세스에서 실행됩니다.

❺ 아파치가 wsgi.py 파일을 액세스할 수 있도록 mysite 디렉토리 및 wsgi.py 파일에 대한 접근 권한을 설정합니다.

❻ /static/ URL에 대한 처리를 위해 static 파일이 위치한 디렉토리를 매핑해줍니다. 이 디렉토리는 장고의 collectstaic 명령에 의해 static 파일을 모아둔 디렉토리입니다. 이는 settings.py 파일의 STATIC_ROOT 항목에 정의된 디렉토리이기도 합니다.

❼ 아파치가 www_static 디렉토리에 액세스할 수 있도록 디렉토리 접근 권한을 설정합니다.

6.5.2 지금까지 작업 확인하기

mod_wsgi 모듈에 대한 아파치 설정이 끝나면 아파치를 기동해서 동작을 확인합니다. 아파치를 기동하고, 브라우저로 접속해서 동작을 확인하는 과정은 내장 모드와 동일합니다. 내장 모드 실행의 **6.4.2 지금까지 작업 확인하기**를 참고하기 바랍니다.

APPENDIX A

장고의 데이터베이스 연동

이 책의 본문에서는 SQLite 데이터베이스를 사용하여 장고 프로젝트 개발을 진행했습니다. 그런데 여러분의 서버에 또는 사용하고 있는 개발 환경에 다른 데이터베이스가 이미 설치되어 있다면 그 데이터베이스를 사용하여 장고 프로젝트를 개발할 수 있습니다.

사실 SQLite 데이터베이스는 작고 가벼워서 사용하기 쉽다는 장점이 있지만, 요즘과 같은 빅데이터 시대에 대규모 프로젝트에서는 거의 사용하지 않습니다. 즉, SQLite는 메모리, 디스크 등 서버 자원을 적게 차지하는 장점이 있는 반면, 멀티 프로세스 환경에서의 동시 접근 처리 능력 등이 약해 보통은 테스트 용도나 소규모의 프로젝트 또는 임베디드 환경에 주로 사용됩니다.

반면 큰 규모의 프로젝트에서는 다른 엔터프라이즈급 데이터베이스를 주로 사용하게 되는데, 장고에서는 SQLite 이외에도 MySQL, PostgreSQL, Oracle 데이터베이스를 공식적으로 지원하고 있습니다. 장고에서 이런 데이터베이스를 연동하는 방법을 소개하겠습니다.

MySQL 데이터베이스 연동

장고에서는 MySQL 5.5 이상의 버전을 지원합니다.

연동 드라이버 설치

파이썬에 MySQL 데이터베이스 연동을 위한 연동 드라이버 모듈은 여러 가지가 존재하는데, 장고는 다음과 같은 3가지 연동 드라이버를 지원합니다. 아래 3가지 중에서 자신에게 맞는 드라이버를 장고가 설치된 서버에 설치해줍니다.

- **MySQLdb** : 가장 오랫동안 사용된 드라이버로, 그만큼 안정되어 있으나 Python 3을 지원하지 않는다는 단점이 있습니다. MySQLdb 1.2.1p2 이상의 버전이 필요합니다.
- **mysqlclient** : 위의 MySQLdb를 개선한 패키지로, Python 3.3 이상의 버전도 지원하고 있어 장고에서 추천하는 드라이버입니다. mysqlclient 1.3.3 이상의 버전이 필요합니다.
- **MySQL Connector/Python** : MySQL 개발사인 오라클에서 제공하고 있는 드라이버로, 장고의 공식 도큐먼트에는 장고의 최신 버전은 지원하지 않을 수도 있다고 되어 있는데, 필자가 확인한 바로는 장고 1.7 버전에서 잘 동작하고 있습니다. 1.1.X 이상의 버전이 필요합니다.

settings.py 파일 수정

장고의 settings.py 파일의 DATABASES 항목에 MySQL 데이터베이스를 사용한다고 지정해줍니다.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dj_mysql',
        'USER': 'root',
        'PASSWORD': 'rootpswd',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

각 항목별 의미는 다음과 같습니다.

표 A-1 MySQL 연동 시 정의하는 항목 - 디폴트 계정 사용

항목	항목 설명
ENGINE	MySQL 엔진을 사용한다는 의미로, django.db.backends.mysql처럼 장고에서 지정한대로 작성하면 됩니다.

NAME	장고에서 사용할 MySQL 내의 데이터베이스 이름을 입력합니다. 즉, 다음과 같은 MySQL 명령으로 데이터베이스를 만들어 사용하면 됩니다. `mysql> create database dj_mysql;`
USER	장고에서 MySQL 데이터베이스에 연결 시 사용할 유저 이름입니다. 즉, 다음과 같은 명령에 사용하는 유저 이름(여기서는 root)을 입력하면 됩니다. `$ mysql -uroot -p`
PASSWORD	장고에서 MySQL 데이터베이스에 연결 시 사용할 유저 이름에 대한 패스워드를 입력합니다. 즉, 다음과 같은 명령에 사용하는 패스워드입니다. `$ mysql -uroot -prootpswd`
HOST	MySQL 서버가 실행되고 있는 머신의 IP 주소를 입력합니다. 장고와 동일 머신에서 MySQL 서버가 실행되고 있으면, 127.0.0.1이라고 써줘도 되고, 생략해도 됩니다.
PORT	MySQL 서버에 접속할 때 사용하는 포트번호입니다. MySQL 서버의 디폴트 포트번호인 3306을 그대로 사용하는 경우는 생략해도 됩니다.

참고로, 다음과 같은 MySQL 명령을 사용하여 새로운 MySQL 계정과 비밀번호를 만들어서 사용할 수도 있습니다.

```
$ mysql -u root -p
mysql> create database dj_test;
mysql> grant all privileges on dj_test.* to 'django'@'localhost' identified by 'pswd';
```

이 경우에는 DATABASES 항목에 다음과 같이 입력합니다.

표 A-2 MySQL 연동 시 정의하는 항목 – 새로운 계정 사용

항목명	NAME	USER	PASSWORD
입력할 내용	dj_test	django	pswd

변경사항 장고에 반영하기

데이터베이스와 관련하여 변경사항이 발생하면, 다음과 같은 명령으로 장고에 반영해줘야 합니다. 당연히 manage.py 파일이 있는 디렉토리로 이동한 후에 명령을 실행해야겠죠?

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

또한 데이터베이스 엔진을 새로 설정해서 초기화된 상태이므로, 장고 Admin 사이트에 로그인하기 위한 관리자를 새로 만들어줘야 합니다.

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

작업 확인하기

아래 명령으로 장고 runserver를 실행 시 에러가 나지 않는다면 데이터베이스 연동이 정상적으로 동작하는 것입니다.

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

좀 더 확실히 확인하고 싶다면, MySQL 데이터베이스에 테이블들이 잘 생성되었는지 확인하면 됩니다. 장고 Admin 사이트에서 테이블들이 잘 보이는지 확인하기 위해 브라우저로 Admin 사이트에 접속합니다. 아래는 Admin 사이트 주소의 한 가지 예시입니다.

```
http://192.168.56.101:8000/admin
```

> **NOTE_** 여기에서 살펴본 내용들은 본문에서 SQLite 데이터베이스 연동 시 설명한 내용들입니다. 복습의 개념으로 **3.4 프로젝트 뼈대 만들기**를 참고하기 바랍니다.

PostgreSQL 데이터베이스 연동

장고에서는 PostgreSQL 9.1 이상의 버전을 지원합니다.

연동 드라이버 설치

파이썬에서 PostgreSQL 데이터베이스 연동을 위한 연동 드라이버 모듈은 여러 가지가 존재하는데, 장고는 psycopg2 패키지를 추천하므로 장고가 설치된 서버에 **psycopg2** 최신 버전을 설치합니다. 장고에서는 psycopg2 2.5 이상의 버전을 추천합니다.

settings.py 파일 수정

장고의 settings.py 파일의 DATABASES 항목에 PostgreSQL 데이터베이스를 사용한다고 지정해줍니다.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'dj_postgres',
        'USER': 'postgres',
        'PASSWORD': 'postgrespswd',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

각 항목별 의미는 다음과 같습니다.

표 A-3 PostgreSQL 연동 시 정의하는 항목 – 디폴트 계정 사용

항목	항목 설명
ENGINE	PostgreSQL 엔진을 사용한다는 의미로, django.db.backends.postgresql_psycopg2처럼 장고에서 지정한대로 작성하면 됩니다.
NAME	장고에서 사용할 PostgreSQL 내의 데이터베이스 이름을 사용합니다. 즉, 다음과 같은 PostgreSQL 명령으로 데이터베이스를 만들어 사용하면 됩니다. `$ createdb dj_postgres`
USER	장고에서 PostgreSQL 데이터베이스에 연결 시 사용할 유저 이름입니다. PostgreSQL 데이터베이스는 기본 사용자로 postgres 유저를 사용합니다. 즉, 다음 명령과 같이 쉘에서 PostgreSQL 데이터베이스의 기본 유저인 postgres로 로그인할 때 사용하는 유저 이름을 입력합니다. `$ su - postgres`

PASSWORD	장고에서 PostgreSQL 데이터베이스에 연결 시 사용할 유저 이름에 대한 패스워드를 입력합니다. 즉, 다음 명령과 같이 쉘에서 postgres 유저로 로그인할 때 사용하는 패스워드를 입력해주면 됩니다. `$ su - postgres`
HOST	PostgreSQL 서버가 실행되고 있는 머신의 IP 주소를 입력합니다. 장고와 동일 머신에서 MySQL 서버가 실행되고 있으면 127.0.0.1이라고 입력하거나 생략합니다.
PORT	PostgreSQL 서버에 접속할 때 사용하는 포트번호입니다. PostgreSQL 서버의 디폴트 포트 번호인 5432를 그대로 사용하는 경우는 생략해도 됩니다.

참고로 다음과 같은 PostgreSQL 명령을 사용하여, PostgreSQL 데이터베이스 내에 새로운 계정과 비밀번호를 만들어서 사용할 수도 있습니다.

```
$ su - postgres
$ createdb dj_test
$ createuser django -P
Enter password for new role: pswd
Enter it again: pswd
$
```

이 경우에는 DATABASES 항목에 다음과 같이 입력합니다.

표 A-4 PostgreSQL 연동 시 정의하는 항목 - 새로운 계정 사용

항목명	NAME	USER	PASSWORD
입력할 내용	dj_test	django	pswd

장고에 반영 및 확인하기

이 이후에 데이터베이스 변경사항을 장고에 반영하는 방법이나, 작업 확인하기는 MySQL과 동일하므로, 앞에서 설명한 **MySQL 데이터베이스 연동**을 참고하기 바랍니다.

Oracle 데이터베이스 연동

장고에서는 Oracle Database Server 11.2 이상의 버전을 지원합니다.

연동 드라이버 설치

파이썬에서 Oracle 데이터베이스 연동을 위한 연동 드라이버 모듈인 **cx_Oracle** 패키지를 최신 버전으로 설치합니다. 장고에서는 cx_Oracle 5.0.1 이상의 버전을 추천하고 있으며, 5.1.3 버전부터 파이썬 3.x 이상의 버전을 지원합니다.

settings.py 파일 수정

장고의 settings.py 파일의 DATABASES 항목에 Oracle 데이터베이스를 사용한다고 지정해줍니다.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.oracle',
        'NAME': 'xe',
        'USER': 'scott',
        'PASSWORD': 'tiger',
        'HOST': '',
        'PORT': '',
    }
}
```

각 항목별 의미는 다음과 같습니다.

표 A-5 Oracle 연동 시 정의하는 항목 – 디폴트 계정 사용

항목	항목 설명
ENGINE	Oracle 엔진을 사용한다는 의미로, django.db.backends.oracle처럼 장고에서 지정한대로 입력합니다.
NAME	장고가 클라이언트 입장에서 접속할 Oracle 데이터베이스 서버 인스턴스의 이름, 즉 SID(System ID)를 입력하면 됩니다. 보통은 오라클 설치 시 디폴트 SID로 xe, orcl 등을 사용합니다. 다음 SQL 명령으로 SID를 알 수 있습니다. > select name from v$database;

USER	장고에서 Oracle 데이터베이스에 연결 시 사용할 유저 이름입니다. $ sqlplus scott/tiger처럼 데이터베이스에 접속할 때, 유저 이름은 scott입니다.
PASSWORD	장고에서 Oracle 데이터베이스에 연결 시 사용할 유저 이름에 대한 패스워드를 입력합니다. $ sqlplus scott/tiger처럼 데이터베이스에 접속할 때, 패스워드는 tiger입니다.
HOST	공란으로 둡니다. 공란으로 두면 오라클의 설정 파일인 tnsnames.ora 파일을 사용합니다. $ORACLE_HOME/network/admin/tnsnames.ora 파일에는 데이터베이스 서버의 HOST, PORT 항목이 정의되어 있습니다.
PORT	공란으로 둡니다. 위와 동일하게 공란으로 두면 오라클의 설정 파일인 tnsnames.ora 파일을 사용합니다. 설정 파일을 사용하지 않는 경우는 HOST와 PORT 항목을 지정합니다. 단 HOST와 PORT 항목을 모두 공란으로 두거나, 아니면 둘 다 지정해야 합니다.

참고로, 다음과 같은 Oracle 데이터베이스 명령을 사용하여 Oracle 데이터베이스 내에 새로운 계정과 비밀번호를 만들어서 사용할 수도 있습니다.

```
$ sqlplus system
SQL> CREATE TABLESPACE ts_django DATAFILE '$ORACLE_HOME/rdbms/dbs/ts_django.dbf' SIZE
40M ONLINE;
SQL> CREATE USER django IDENTIFIED BY pswd DEFAULT TABLESPACE ts_django;
SQL> GRANT connect, resource TO django;
```

이 경우에는 DATABASES 항목에 다음과 같이 입력합니다.

표 A-6 Oracle 연동 시 정의하는 항목 – 새로운 계정 사용

항목명	NAME	USER	PASSWORD
입력할 내용	xe	django	pswd

장고에 반영 및 확인하기

이 이후에 데이터베이스 변경사항을 장고에 반영하는 방법이나, 작업 확인하기는 MySQL과 동일하므로, 앞에서 설명한 **MySQL 데이터베이스 연동**을 참고하기 바랍니다.

APPENDIX B

HTTP 상태 코드 전체 요약

HTTP 응답에 포함되는 상태 코드Status code는 IANAInternet Assigned Numbers Authority라는 인터넷 할당 번호 관리기관이 HTTP 상태 코드 레지스트리라는 이름으로 관리하고 있습니다.

> **NOTE_** 상태 코드 레지스트리 내용은 아래 웹 페이지를 참고하기 바랍니다.
> http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml

아래 표에 상태 코드와 그 의미를 요약 정리하였으니 참고하기 바랍니다.

상태 코드	상태 텍스트	응답 문구	서버 측면에서의 의미
1xx	Informational	정보 제공	**클라이언트의 요청을 받았으며 작업을 계속 진행하고 있다.** 1xx 계열의 응답은 HTTP/1.1 클라이언트에게만 보낼 수 있으며, 응답은 바디없이 상태라인, 헤더(생략 가능), 빈 줄로 종료됩니다.
100	Continue	계속	**계속 진행하라.** 클라이언트는 요청 헤더에 'Expect: 100-continue'를 보내고, 서버는 이를 처리할 수 있으면 이 코드로 응답합니다.
101	Switching Protocols	프로토콜 전환	**프로토콜을 전환하라.** 프로토콜을 HTTP 1.1에서 업그레이드할 때 Upgrade 응답 헤더에 표시합니다. 현재는 HTTP 1.1이 최신이므로 사용할 일이 없습니다.

102	Processing	처리중	**(WebDAV) 처리 중이다.** 서버가 처리하는 데 오랜 시간이 예상되어 클라이언트에서 타임아웃이 발생하지 않도록 이 응답코드를 보냅니다.
103–199	Unassigned		현재 할당되지 않은 상태 코드입니다.
2xx	**Success**	**성공**	**클라이언트가 요청한 동작을 수신하여 이해했고, 승낙했으며 성공적으로 처리했다.**
200	OK	성공	**서버가 요청을 성공적으로 처리했다.**
201	Created	작성됨	**요청이 처리되어서 새로운 리소스가 생성되었다.** 응답 헤더 Location에 새로운 리소스의 절대 URI를 기록합니다.
202	Accepted	허용됨	**요청은 접수했지만 처리가 완료되지 않았다.** 응답 헤더의 Location, Retry-After를 참고하여 클라이언트는 다시 요청을 보냅니다.
203	Non-Authoritative Information	신뢰할 수 없는 정보	**응답 헤더가 오리지널 서버로부터 제공된 것이 아니다.** 프록시 서버가 응답 헤더에 주석을 덧붙인 경우가 하나의 예입니다.
204	No Content	콘텐츠 없음	**처리가 성공했지만, 클라이언트에게 돌려줄 콘텐츠가 없다.** DELETE 요청에 대한 응답에 많이 사용됩니다.
205	Reset Content	콘텐츠 재설정	**처리가 성공했고, 브라우저의 화면을 리셋하라.** 예를 들어, 브라우저가 입력 폼을 보여주고 있을 때 이 응답 코드를 받으면 브라우저는 모든 입력 항목을 리셋하고 재입력할 수 있는 상태가 됩니다.
206	Partial Content	일부 콘텐츠	**콘텐츠의 일부만을 보낸다.** 응답 헤더의 Content-Range에 응답 콘텐츠의 범위를 기록합니다. 예를 들어, 1,500 바이트의 리소스 중에서 처음의 500 바이트만을 보낼 때 사용할 수 있습니다.
207	Multi-Status	다중 상태	**(WebDAV) 처리 결과의 스테이터스가 여러 개이다.** 207 응답은 성공을 뜻하지만, 각각의 처리 결과가 성공인지는 바디를 봐야 알 수 있습니다.
208–299	Unassigned		현재 할당되지 않은 상태 코드입니다.
3xx	**Redirection**	**리다이렉션**	**클라이언트는 요청을 마치기 위해 추가적인 동작을 취해야 한다.**

300	Multiple Choices	여러 선택항목	**선택 항목이 여러 개 있다.** 지정한 URI에 대해서 콘텐츠 협상을 수행한 결과 서버에서 콘텐츠를 결정하지 못하고, 클라이언트에게 복수 개의 링크를 응답할 때 사용합니다.
301	Moved Permanently	영구 이동	**지정한 리소스가 새로운 URI로 이동했다.** 이동할 곳의 새로운 URI는 응답 헤더 Location에 기록합니다.
302	Found	다른 위치 찾음	**요청한 리소스를 다른 URI에서 찾았다.** 요청한 URI가 없으므로, 클라이언트는 메소드를 그대로 유지한 채 응답 헤더 Location에 표시된 다른 URI로 요청을 재송신할 필요가 있습니다. 302의 의미를 정확하게 개선해서 307을 정의하였기 때문에 이 응답코드의 사용은 권장하지 않습니다.
303	See Other	다른 위치 보기	**다른 위치로 요청하라.** 요청에 대한 처리 결과를 응답 헤더 Location에 표시된 URI에서 GET으로 취득할 수 있습니다. 브라우저의 폼 요청을 POST로 처리하고 그 결과 화면으로 리다이렉트시킬 때, 자주 사용하는 응답 코드입니다.
304	Not Modified	수정되지 않음	**마지막 요청 이후, 요청한 페이지는 수정되지 않았다.** If-Modified-Since와 같은 조건부 GET 요청일 때, 지정한 리소스가 갱신되지 않았음을 알려줍니다. 이 응답 코드에는 바디가 없습니다.
305	Use Proxy	프록시 사용	**지정한 리소스에 액세스하기 위해서는 프록시를 통해야 한다.** 응답 헤더 Location에 프록시의 URI를 기록합니다.
306	(Unused)		예전 버전에서 사용하다가 현재는 사용하지 않는 상태 코드입니다.
307	Temporary Redirect	임시 리다이렉션	**임시로 리다이렉션 요청이 필요하다.** 요청한 URI가 없으므로, 클라이언트는 메소드를 그대로 유지한 채, 응답 헤더 Location에 표시된 다른 URI로 요청을 재송신할 필요가 있습니다. 클라이언트는 향후 요청 시 원래 위치를 계속 사용해야 합니다. 302의 의미를 정확하게 재정의해서 HTTP/1.1의 307 응답으로 추가되었습니다.
308–399	Unassigned		현재 할당되지 않은 상태 코드입니다.
4xx	Client Error	클라이언트 에러	**클라이언트의 요청에 오류가 있다.**

400	Bad Request	잘못된 요청	**요청의 구문이 잘못되었다.** 클라이언트가 모르는 4xx 계열 응답 코드가 반환된 경우에도 클라이언트는 400과 동일하게 처리하도록 규정하고 있습니다.
401	Unauthorized	권한 없음	**지정한 리소스에 대한 액세스 권한이 없다.** 응답 헤더 WWW-Authenticate에 필요한 인증 방식을 지정합니다.
402	Payment Required	결재 필요	**지정한 리소스를 액세스하기 위해서는 결재가 필요하다.** 이 응답 코드는 실제로는 사용되지 않고 있습니다.
403	Forbidden	금지됨	**지정한 리소스에 대한 액세스가 금지되었다.** 401 인증 처리 이외의 사유로 리소스에 대한 액세스가 금지되었음을 의미합니다. 리소스의 존재 자체를 은폐하고 싶은 경우는 404 응답 코드를 사용할 수 있습니다.
404	Not Found	찾을 수 없음	**지정한 리소스를 찾을 수 없다.**
405	Method Not Allowed	허용되지 않은 메소드	**요청한 URI가 지정한 메소드를 지원하지 않는다.** 응답 헤더 Allow에 이 URI가 지원하는 메소드 목록을 기록합니다.
406	Not Acceptable	수용할 수 없음	**클라이언트가 Accept-* 헤더에 지정한 항목에 대해 처리할 수 없다.** 응답 바디에는 300 응답처럼 서버가 수용 가능한 다른 선택지 리스트가 기록됩니다.
407	Proxy Authentication Required	프록시 인증 필요	**클라이언트는 프록시 서버에 인증이 필요하다.** 프록시 서버의 응답 헤더 Proxy-Authenticate에 필요한 인증 방식을 지정합니다.
408	Request Timeout	요청 시간초과	**요청을 기다리다 서버에서 타임아웃했다.**
409	Conflict	충돌	**서버가 요청을 수행하는 중에 충돌이 발생했다.** 예를 들어, 사용자명을 new_name으로 변경하려 했지만, 서버에 이미 new_name이라는 사용자가 존재하는 경우입니다. 응답 헤더 Location에는 충돌이 발생한 리소스의 URI를 기록합니다.
410	Gone	사라짐	**지정한 리소스가 이전에는 존재했었지만, 현재는 존재하지 않는다.** 예를 들어, 기간이 한정된 프로모션 사이트에서 사용할 수 있는 응답 코드입니다.
411	Length Required	길이 필요	**요청 헤더에 Content-Length를 지정해야 한다.**

412	Precondition Failed	사전조건 실패	**If-Match와 같은 조건부 요청에서 지정한 사전 조건이 서버와 맞지 않는다.**
413	Request Entity Too Large	요청 객체가 너무 큼	**요청 메시지가 너무 크다.** 서버는 접속을 끊습니다.
414	Request-URI Too Large	요청 URI가 너무 김	**요청 URI가 너무 길다.**
415	Unsupported Media Type	지원되지 않는 미디어 유형	**클라이언트가 지정한 미디어 타입을 서버가 지원하지 않는다.** 예를 들어, 서버가 지원하는 이미지는 JPG, PNG 뿐인데, 클라이언트가 GIF 형식의 이미지를 요청하는 경우입니다.
416	Range Not Satisfiable	처리할 수 없는 요청범위	**클라이언트가 지정한 리소스의 범위가 서버의 리소스 사이즈와 맞지 않는다.**
417	Expectation Failed	예상 실패	**클라이언트가 지정한 Expect 헤더를 서버가 이해할 수 없다.**
418–421	Unassigned		현재 할당되지 않은 상태 코드입니다.
422	Unprocessable Entity	처리할 수 없는 엔티티	**(WebDAV) 클라이언트가 송신한 XML이 구문은 맞지만 의미상 오류가 있다.**
423	Locked	잠김	**(WebDAV) 지정한 리소스는 잠겨있다.**
424	Failed Dependency	의존관계로 실패	**(WebDAV) 다른 작업의 실패로 인해 본 요청도 실패했다.**
425–499	Unassigned		현재 할당되지 않은 상태 코드입니다.
5xx	Server Error	서버 에러	**클라이언트의 요청은 유효한데, 서버가 처리에 실패했다.**
500	Internal Server Error	내부 서버 오류	**서버쪽에서 에러가 발생했다.** 클라이언트가 모르는 5xx 계열의 응답 코드가 반환된 경우에도 클라이언트는 500과 동일하게 처리하도록 규정하고 있습니다.
501	Not Implemented	구현되지 않음	**요청한 URI의 메소드에 대해 서버가 구현하고 있지 않다.**
502	Bad Gateway	불량 게이트웨이	**게이트웨이 또는 프록시 역할을 하는 서버가 그 뒷단의 서버로부터 잘못된 응답을 받았다.**
503	Service Unavailable	서비스 제공불가	**현재 서버에서 서비스를 제공할 수 없다.** 보통은 서버의 과부하나 서비스 점검 등 일시적인 상태입니다.

504	Gateway Timeout	게이트웨이 시간초과	**게이트웨이 또는 프록시 역할을 하는 서버가 그 뒷단의 서버로부터 응답을 기다리다 타임아웃이 발생했다.**
505	HTTP Version Not Supported	HTTP 버전 미지원	**클라이언트가 요청에 사용한 HTTP 버전을 서버가 지원하지 않는다.**
506	Unassigned		현재 할당되지 않은 상태 코드입니다.
507	Insufficient Storage	용량 부족	**(WebDAV) 서버에 저장 공간 부족으로 처리에 실패했다.**
512–599	Unassigned		현재 할당되지 않은 상태 코드입니다.

102, 207, 422, 423, 424, 507 항목은 HTTP/1.1 규격에 추가된 WebDAV용 상태 코드들입니다.

> **NOTE_ WebDAV(웹 분산 저작 및 버전 관리)란?**
> WebDAV는 Web Distributed Authoring and Versioning의 약자로, 웹에 있는 문서와 파일을 읽기뿐만 아니라, 서로 협업하면서 사용 가능하록 규정한 HTTP/1.1 프로토콜의 확장 규격입니다.

APPENDIX C

장고의 설계 원칙

당시 장고 개발자들이 프레임워크를 개발하면서 사용했던 기본 원칙들입니다. 이는 어떤 철학을 갖고 장고를 개발해 왔는지를 알려주는 것뿐만 아니라, 앞으로 계속 개발하는 과정에서도 개발 원칙에 대한 가이드로 삼기 위해서 장고 개발자들이 정리한 도큐먼트입니다.

> **NOTE_** 장고 설계 원칙에 대한 영어 원문은 장고 홈페이지를 참고하기 바랍니다.
> https://docs.djangoproject.com/en/1.7/misc/design-philosophies/

| 일반 사항 |

약한 결합(Loose coupling)

장고의 스택들은 각 레이어별로 서로 독립적으로 개발되었습니다. 그래서 꼭 필요한 경우가 아니라면 서로 의존성이 없어야 합니다. 예를 들어, 템플릿 시스템은 웹 요청이 무엇인지 알지 못하며, 데이터베이스 레이어는 데이터가 어떻게 표시되는지 모릅니다. 또한 뷰 시스템은 템플릿 시스템이 무엇인지 고려하지 않습니다. 장고는 편리함을 위하여 풀스택으로 개발되었지만, 스택의 각 모듈은 서로 독립적입니다.

경량 코드(Less code)

장고는 되도록이면 적은 코드로 애플리케이션을 작성할 수 있어야 하고, 반복적으로 사용되는 코

드를 줄일 수 있어야 합니다. 또한 파이썬의 인트로스펙션(introspection)과 같은 동적 특성을 완전히 활용해야 합니다.

신속 개발(Quick development)

요즘의 웹 프레임워크는 공통적이고 반복적인 사항들을 빠르게 개발할 수 있도록 하는 것이 중요합니다. 장고 역시 그에 맞춰 웹 프로그래밍을 높은 수준으로 빠르게 개발할 수 있도록 해야 합니다.

반복 방지(DRY, Don't Repeat Yourself)

중복을 방지하고 정규화될 수 있도록 모든 로직이나 데이터는 오직 한 곳에만 존재해야 합니다. 따라서 장고는 적은 코드로도 많은 기능을 제공할 수 있어야 합니다.

암시보다는 명시적으로 표현(Explicit is better than implicit)

이는 PEP 20에 명시된 파이썬의 핵심 원칙입니다. 마치 마술처럼 암시적으로 무엇인가가 처리되는 것은 개발자를 혼동시킬 우려가 있으므로, 되도록이면 명시적으로 표현해야 합니다.

일관성(Consistency)

장고 프레임워크는 파이썬 코딩 스타일과 같은 하위 레벨에서부터 장고의 사용 방법과 같은 상위 레벨까지 일관성을 유지해야 합니다.

| 모델 |

암시적보다는 명시적으로 표현(Explicit is better than implicit)

테이블의 필드는 이름만으로 그 의미를 암시적으로 가정해서는 안 되며, 필드 타입이나 키워드 인자를 통해 필드에 대해 명시적으로 표현해야 합니다.

관련 도메인 로직을 모두 포함(Include all relevant domain logic)

모델 클래스는 마틴 파울러의 Active Record 디자인 패턴을 따라 객체의 모든 면을 포함해야 합니다. 이는 데이터와 그 데이터에 대한 정보들이 모델 클래스로 정의되는 이유이기도 합니다. 주어진 모델에 대해 알아야 할 사항은 모두 모델 클래스에 정의되어야 합니다.

| 데이터베이스 API |

SQL 효율성(SQL efficiency)

SQL 문장은 되도록이면 적은 횟수로, 최적화된 문장으로 실행되어야 합니다. 이는 개발자들이 save() 함수를 호출해야 하는 이유이고, 또한 select_related() QuerySet 메소드가 존재하는 이유이기도 합니다. 이 메소드를 사용하면 관련 객체를 검색하는 속도를 향상시킬 수 있습니다.

간결하고 강력한 문법(Terse, powerful syntax)

데이터베이스 API는 간단한 문법으로도 다양한 표현이 가능해야 합니다. 다른 모듈이나 헬퍼 객체를 임포트해야만 표현이 가능한 문법은 바람직하지 않습니다. 조인은 필요하다면 백그라운드에서 자동적으로 실행되어야 하며, 모든 객체는 관련된 객체 모두를 양방향으로 액세스할 수 있어야 합니다.

필요 시 쉽게 작성할 수 있는 SQL(Option to drop into raw SQL easily, when needed)

데이터베이스 API는 간단한 문법(shortcut)을 제공하는 것이지 모든 SQL 문장을 제공하는 것이 아닙니다. 따라서 장고에서는 사용자가 정의할 수 있는 SQL 문장을 작성하는 것이 쉬워야 합니다. 범위는 문장 전체일 수도 있고, API 호출 시 넘겨주는 파라미터에 대응되는 WHERE 절일 수도 있습니다.

| URL 설계 |

약한 결합(Loose coupling)

장고 애플리케이션의 URL은 처리함수와 묶여서는 안 됩니다. URL과 처리함수 이름을 동일하게 사용하는 것도 좋지 않습니다. 그렇게 해야만 동일한 애플리케이션을 다른 사이트로 이동시킬 때에도 변경이 쉬워집니다. 예를 들어, 어느 사이트에서 /stories/라는 URL을 사용한 경우, 동일한 기능을 다른 사이트에서는 /news/라고 지정할 수 있습니다.

제약없는 유연성(Infinite flexibility)

URL은 가능한한 유연해야 합니다. 그래서 생각할 수 있는 모든 URL이 가능해야 합니다.

베스트 프랙티스 권장(Encourage best practices)

개발자가 간편 URL을 설계하는 것이 쉽도록 해야 합니다. 또한, URL에 파일 확장자가 들어가거나, 비네트 스타일로 URL에 콤마(,)를 넣는 것은 피해야 합니다.

결정적인 URL(Definitive URLs)

기술적으로 보면 foo.com/bar와 foo.com/bar/는 서로 다른 URL입니다. 검색 엔진 로봇(또는 웹 트래픽 분석 툴)은 이들을 다른 페이지로 처리합니다. 장고는 검색 엔진 로봇에 혼동을 주지 않도록 URL을 정규화해야 합니다. 그래서 APPEND_SLASH 세팅이 필요한 것입니다.

| 템플릿 시스템 |

표현과 로직의 분리(Separate logic from presentation)

템플릿 시스템은 표현 및 표현에 관련된 로직을 제어하기 위한 툴입니다. 그 뿐입니다. 템플릿 시스템이 처리 기능에 관여해서는 안 됩니다.

중복 배제(Discourage redundancy)

대부분의 웹 사이트는 동일한 헤더 및 푸터, 네비게이션 바 등 시스템 전체적으로 일관된 설계를 사용합니다. 장고 템플릿 시스템에서는 그런 요소들을 한 곳에 넣고 코드 중복을 방지하는 것을 쉽게 할 수 있어야 합니다. 이는 템플릿 상속 기능의 원칙이기도 합니다.

HTML과 분리(Be decoupled from HTML)

템플릿 시스템은 HTML만 출력하도록 설계되어서는 안 됩니다. HTML 이외의 다른 텍스트 포맷이나 텍스트 그 자체도 출력할 수 있어야 합니다.

템플릿 언어로 XML 금지(XML should not be used for template languages)

템플릿 분석에 XML 엔진을 사용하면 템플릿 편집 시 휴먼 에러를 야기할 수 있고, 템플릿을 처리할 때 오버헤드가 많이 발생합니다.

디자이너의 능력 가정(Assume designer competence)

템플릿 시스템은 드림위버와 같은 WYSIWYG 편집기에서 멋지게 표현될 필요는 없습니다. 제약

이 너무 많고 문법이 복잡하기 때문입니다. 장고는 템플릿 작성자가 HTML을 직접 다루는 데 익숙하기를 기대합니다.

여백을 확실하게 처리(Treat whitespace obviously)

템플릿 시스템에서는 명확한 여백 처리가 필요합니다. 템플릿이 여백을 포함한다면 템플릿 시스템은 여백을 텍스트처럼 처리해서 그대로 출력해야 합니다. 템플릿 태그에 있는 여백이 아니라면 출력됩니다.

프로그래밍 언어를 만들지 말자(Don't invent a programming language)

템플릿 시스템은 변수에 값 할당 및 복잡한 로직을 의도적으로 금지합니다. 이는 프로그래밍 언어를 새로 만들지 않고도 프로그래밍과 같은, 즉 표현에 관련된 기능에 필수적인 분기 및 루프와 같은 기능을 충분히 제공하는 것이 목적입니다.

템플릿은 프로그래머가 아니라 파이썬 지식이 없는 디자이너에 의해 작성된다는 점을 고려해야 합니다.

안정성과 보안(Safety and security)

템플릿 시스템은 데이터베이스 레코드를 삭제하는 명령과 같은 악성 코드가 삽입되는 것을 방지해야 합니다. 이것은 템플릿 시스템이 파이썬 코드 모두를 허용하지 않는 또 다른 이유입니다.

확장성(Extensibility)

템플릿 시스템은 고급 작성자가 기능을 확장할 수 있어야 합니다. 이는 사용자 정의 템플릿 태그와 필터의 원칙이기도 합니다.

| 뷰 |

간단함(Simplicity)

뷰를 작성하는 것은 파이썬 함수를 작성하는 것만큼 간단해야 합니다. 개발자는 함수로 가능한 상황에서 클래스로 인스턴스를 만들 필요가 없어야 합니다.

요청 객체의 사용(Use request objects)

뷰는 진행 중인 요청에 대한 메타 데이터를 담고 있는 request 객체에 접근이 가능해야 합니다. 글로벌 변수를 통해 뷰 함수가 요청 데이터에 액세스하기보다는 객체가 뷰 함수에 직접 전달되어야 합니다. 이런 방식이 시험용 요청 객체로 뷰를 테스트하는 기능을 가볍고 쉽게 해줍니다.

약한 결합(Loose coupling)

개발자가 어떤 템플릿 시스템을 사용하는지, 더 나아가 템플릿 시스템이 사용되는지 조차 뷰가 고려하게 해서는 안 됩니다.

GET, POST 간 차이(Differentiate between GET and POST)

GET과 POST는 다릅니다. 개발자는 확실하게 그 중 하나만을 사용해야 합니다. 장고는 GET과 POST 데이터를 구분하는 것이 쉬워야 합니다.

| 캐시 시스템 |

경량 코드(Less code)

캐시는 가능한 속도가 빨라야 합니다. 그러므로 뒷단의 캐시 시스템을 감싸고 있는 프레임워크의 모든 코드는, 특히 get() 메소드에 대해서는 절대적으로 최소화해야 합니다.

일관성(Consistency)

캐시 API는 뒷단의 다른 캐시 시스템들과도 일관된 인터페이스를 제공해야 합니다.

확장성(Extensibility)

캐시 API는 개발자의 필요에 따라 애플리케이션 레벨에서 확장이 가능해야 합니다.

INDEX

기호

A~B

C

D

INDEX

INDEX

지금은 모던 웹 시대!

모던 웹 디자인을 위한
HTML5 + CSS3 입문 개정판

HTML5 분야 부동의 1위 도서

HTML5 표준안 확정에 맞춘 완전 개정판의 귀환!
HTML5 권고안과 최신 웹 브라우저 환경 대응

윤인성 저 | 624쪽 | 30,000원

모던 웹을 위한
JavaScript + jQuery 입문 개정판

자바스크립트에서 제이쿼리, 제이쿼리 모바일까지 한 권으로 끝낸다!

시대의 흐름에 맞춰 다시 쓴 자바스크립트 교과서

윤인성 저 | 980쪽 | 32,000원

모던 웹을 위한
Node.js 프로그래밍 개정판

페이스북, 월마트, 링크드인은 왜 Node.js를 선택했는가?

이 물음에 대한 답은 Node.js가 보여주는 빠른 처리 능력 때문이다.

윤인성 저 | 484쪽 | 25,000원

HTML5 + CSS3 정복 개정판

필요한 것만 배워 바로 현장에서 쓰는 HTML5

순서대로 읽으며 실습할 수 있는 HTML5 자습서

김상형 저 | 700쪽 | 32,000원