

Lab04: Django Tutorial II

ESM2014-41

객체지향프로그래밍 및 실습

SKKU 시스템경영공학과

조영일

Django Model API Detail 1

- `[Model 이름].objects.all()` : 해당 Model로 만들어진 모든 Objects(instance)를 가져옴
- `[Model 이름].objects.get(arguments)` : 해당 Model로 만들어진 모든 Objects 중 arguments 로 전달되는 조건에 맞는 단 하나의 항목을 가져옴(해당 조건에 해당하는 항목이 여러 개 일 경우 Exception)
- `[Model 이름].objects.filter(arguments)` : 해당 Model로 만들어진 모든 Objects 중 arguments 로 전달되는 조건에 맞는 일부 항목을 필터링해서 가져옴
- `[Model 이름].objects.[...].exclude(arguments)` : arguments에 해당하는 조건에 맞는 항목을 제외하고 나머지 항목을 가져옴
- Chaining Filters : 위의 여러가지의 결과물은 `QuerySet(Object들의 List)`형태로 반환되며 `QuerySet`은 또다시 위의 API를 적용 할 수 있다. API를 반복적으로 조합하여 사용 할 수 있음

Django Model API Detail

```
# Question Model 의 모든 데이터를 가져옴(.all)
>>> all_questions = Question.objects.all()

# Question Model 의 데이터 중 id = 1 인 하나의 Object를 가져옴(.get)
>>> first_question = Question.objects.get(pk=1)

# Choice Model 의 데이터 중 투표수(votes)가 1이상인 데이터들을 가져옴(.filter)
>>> choices = Choice.objects.filter(votes__gte=1)

# Choice Model 의 데이터 중 투표수(votes)가 1이상인 데이터 중에서 5이상인 데이터들은
# 제외하고 가져옴(.filter & .exclude Chaining)
>>> choices = Choice.objects.filter(votes__gte=1).exclude(votes__gte=5)
```

Django Model API Detail 2

- `[Model 이름].objects.[...].order_by("attribute 명")` : QuerySet을 Model의 특정 Attribute를 기준으로 하여 정렬함.

“attribute 명” => 오름차순 / “-attribute 명” => 내림차순

- Limiting QuerySet : Model API 를 수행한 결과물은 Object들의 List 형태(QuerySet)로 반환되므로 List Indexing & Slicing을 사용하면 결과물의 일부만 선택적으로 가져올 수 있다.

Django Model API Detail

```
# Choice Model 의 데이터를 모두 가져오고 그 결과를 투표수(votes)에 따라 오름차순 정렬한다.
>>> choices = Choice.objects.all().order_by("votes")
# 내림차순 정렬
>>> choices = Choice.objects.all().order_by("-votes")

# 내림차순 정렬된 결과에서 첫 5개를 가져온다.
>>> Choice.objects.all().order_by("-votes")[:5]

# 내림차순 정렬된 결과에서 6번째 부터 10번째 결과를 가져온다.
>>> Choice.objects.all().order_by("-votes")[5:10]
```

Django Model API Detail

- Django Database(Model) API :
<https://docs.djangoproject.com/ko/2.2/topics/db/queries/>
- Related Object(객체간의 관계 표현하기) :
<https://docs.djangoproject.com/ko/2.2/topics/db/queries/#related-objects>

Django Admin

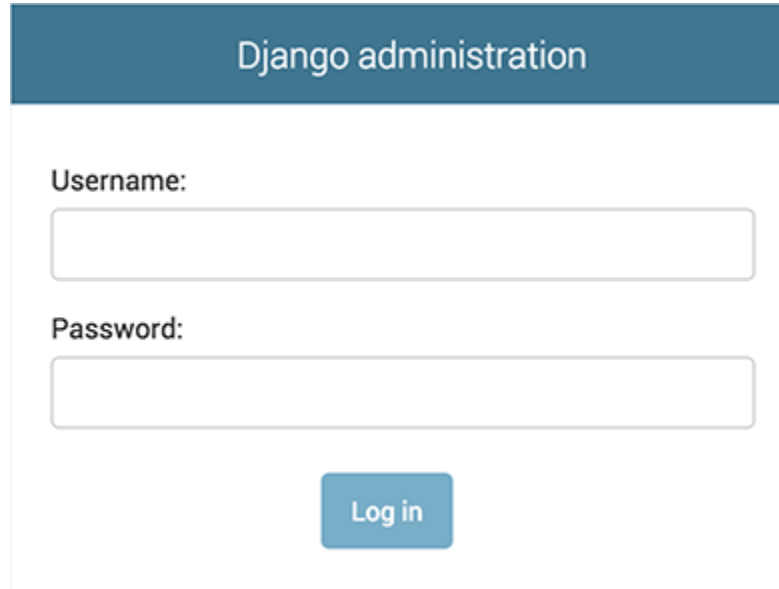
- Django Admin : 웹 서비스를 운영하는 직원들이나 고객들이 콘텐츠를 수정하기 위한 관리자 사이트를 만드는 것은 딱히 창의적일 필요 없는 반복적이고 지루한 작업. 이런 이유로, Django는 프로젝트에 등록된 모델에 대한 관리용 인터페이스를 모두 자동으로 생성합니다.
- 최고 관리자 생성하기 : “python manage.py createsuperuser”

Django Admin

- Django Admin : 웹 서비스를 운영하는 직원들이나 고객들이 콘텐츠를 수정하기 위한 관리자 사이트를 만드는 것은 딱히 창의적일 필요 없는 반복적이고 지루한 작업. 이런 이유로, Django는 프로젝트에 등록된 모델에 대한 관리용 인터페이스를 모두 자동으로 생성합니다.
- Django Admin 도 결국 프로젝트를 구성하는 어플리케이션 중 하나
- 최고 관리자 생성하기 : “python manage.py createsuperuser”

Django Admin

- <http://127.0.0.1:8000/admin/> 접속 후 로그인

A screenshot of the Django administration login page. It features a dark blue header with the text "Django administration". Below the header, there are two input fields: "Username:" and "Password:". At the bottom, there is a blue "Log in" button.

Django administration

Username:

Password:

Log in

Django Admin

- 관리자 페이지에서 polls app을 변경 가능하도록 만들기 : 사용자가 생성한 application의 model 들을 Django에서 수정 가능하도록 하기 위해서는 각 application의 model들을 등록(register) 해주는 과정이 필요
- polls/admin.py 파일을 열고 아래와 같이 수정

```
# admin module import
from django.contrib import admin

# 우리가 정의한 두개 Model을 import 한다.
from polls.models import Question, Choice

# admin.site.register(등록할 Model 이름)
admin.site.register(Question)
```

Django Admin

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

[+ Add](#)

[Change](#)

Users

[+ Add](#)

[Change](#)

POLLS

Questions

[+ Add](#)

[Change](#)

Recent Actions

My Actions

None available

Home > Polls > Questions

Select question to change

[ADD QUESTION +](#)

Action: 0 of 1 selected

☐ QUESTION

☐ What's up?

1 question

Django Admin

- Model `__str__` method : Model을 대표하여 표시될 이름을 사용자가 직접 정의 하는 기능

```
from django.db import models

class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text

class Choice(models.Model):
    # ...
    def __str__(self):
        return self.choice_text
```

Django Admin

Home > Polls > Questions > What's up?

Change question

HISTORY

Question text:

What's up?

Date published:


Date:

2015-09-06

Today | 

Time:

21:16:22

Now | 

Delete

Save and add another

Save and continue editing

SAVE

Django View

- View : Django Application이 URL로 부터 특정 요청을 받았을 때 그 요청을 처리하고 응답을 반환하는 일련의 로직을 작성하는 것
 - e.g. “Blog Application” View List
 - Blog 홈페이지 -- 가장 최근의 항목들을 보여줍니다.
 - 항목 "세부"(detail) 페이지 -- 하나의 항목에 연결하는 영구적인 링크(permalink)를 제공합니다.
 - 년도별 축적 페이지 -- 주어진 연도의 모든 월별 항목들을 표시합니다.
 - 월별 축적 페이지 -- 주어진 월의 날짜별 항목들을 표시합니다.
 - 날짜별 축적 페이지 -- 주어진 날짜의 모든 항목들을 표시합니다.
 - 댓글 기능 -- 특정 항목의 댓글을 다룰 수 있는 기능

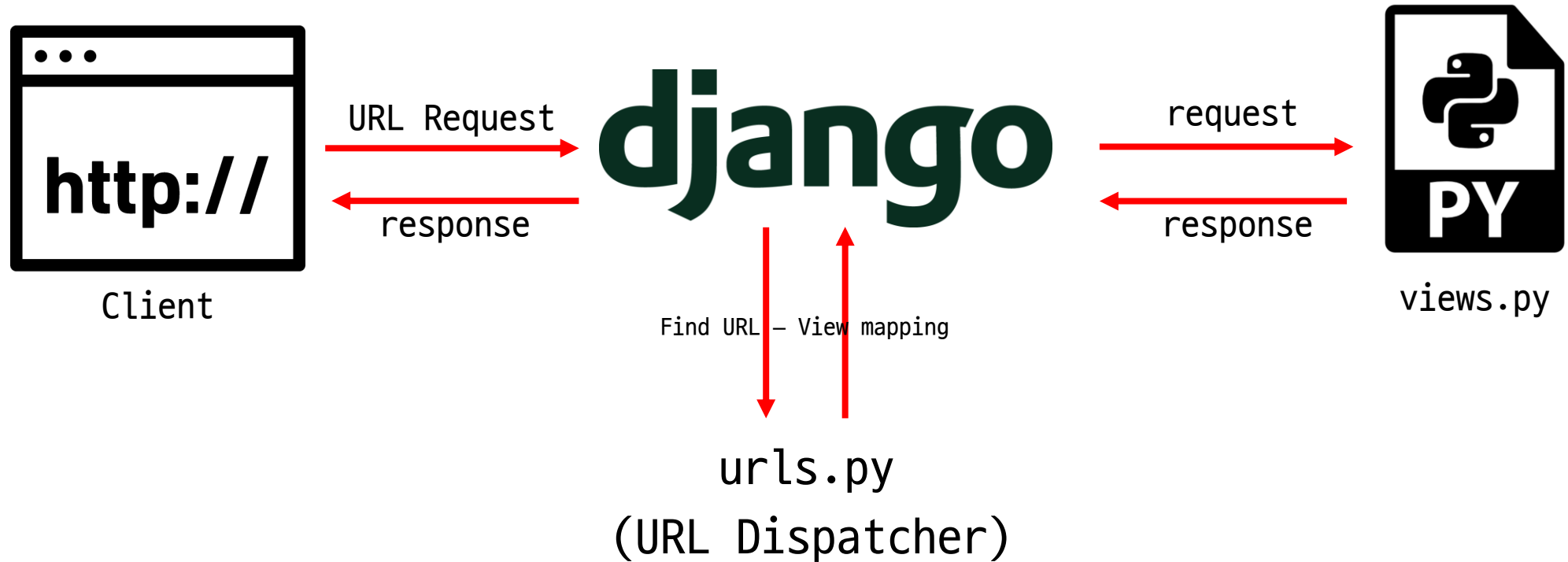
Django View

- View : Django Application이 URL로 부터 특정 요청을 받았을 때 그 요청을 처리하고 응답을 반환하는 일련의 로직을 작성하는 것
 - e.g. “Blog Application” View List
 - Blog 홈페이지 -- 가장 최근의 항목들을 보여줍니다.
 - 항목 "세부"(detail) 페이지 -- 하나의 항목에 연결하는 영구적인 링크(permalink)를 제공합니다.
 - 년도별 축적 페이지 -- 주어진 연도의 모든 월별 항목들을 표시합니다.
 - 월별 축적 페이지 -- 주어진 월의 날짜별 항목들을 표시합니다.
 - 날짜별 축적 페이지 -- 주어진 날짜의 모든 항목들을 표시합니다.
 - 댓글 기능 -- 특정 항목의 댓글을 다룰 수 있는 기능

Django View

- polls 앱에서 만들어볼 view
 - 질문 "색인" 페이지 -- 최근의 질문들을 표시합니다.
 - 질문 "세부" 페이지 -- 질문 내용과, 투표할 수 있는 서식을 표시합니다.
 - 질문 "결과" 페이지 -- 특정 질문에 대한 결과를 표시합니다
 - 투표 기능 -- 특정 질문에 대해 특정 선택을 할 수 있는 투표 기능을 제공합니다.

Django View



Django View

- polls/views.py

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

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

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

Django View

- polls/urls.py

```
from django.urls import path

from polls import views

urlpatterns = [
    # ex: /polls/
    path('', views.index, name='index'),
    # ex: /polls/5/
    path('<int:question_id>/', views.detail, name='detail'),
    # ex: /polls/5/results/
    path('<int:question_id>/results/', views.results, name='results'),
    # ex: /polls/5/vote/
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

Django View

- polls/urls.py

```
from django.urls import path

from polls import views

urlpatterns = [
    # ex: /polls/
    path('', views.index, name='index'),
    # ex: /polls/5/
    path('<int:question_id>/', views.detail, name='detail'),
    # ex: /polls/5/results/
    path('<int:question_id>/results/', views.results, name='results'),
    # ex: /polls/5/vote/
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

Django View

- <http://127.0.0.1:8000/polls/34/>
- <http://127.0.0.1:8000/polls/34/results/>
- <http://127.0.0.1:8000/polls/34/vote/>
- 사용자가 웹사이트의 페이지를 요청할 때, 예로 `"/polls/34/"`를 요청했다고 하면, Django는 `hello_world.urls` 파이썬 모듈을 불러오게 됨. `ROOT_URLCONF` 설정에 의해 해당 모듈을 바라보도록 지정되어 있기 때문.
- `hello_world.urls`에서 `urlpatterns`라는 변수를 찾고, 순서대로 패턴을 따라감. `'polls/'`를 찾은 후엔, 일치하는 텍스트(`"polls/"`)를 버리고, 남은 텍스트인 `"34/"`를 `'polls.urls'` `URLconf`로 전달하여 남은 처리를 진행. 거기에서 `'<int:question_id>/'`와 일치하여, 결과적으로 다음과 같이 `detail()` 뷰 함수가 호출됨.
- `detail(request=<HttpRequest object>, question_id=34)`

Django View

- 실제로 Model과 상호 작용하는 View 만들기 : 데이터베이스에 저장된 최소한 5 개의 투표 질문이 콤마로 분리되어, 발행일에 따라 출력되도록 하는 View(Django Tutorial I 에서 만든 index view를 수정함)

```
# Django에서 일반적인 HTTP 응답을 생성하기 위한 함수 - HttpResponse
from django.http import HttpResponse
from polls.models import Question

def index(request):
    # Django Database API
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    # latest_question_list 를 for 문을 통해 iteration하면서 각각의 question_text를 comma로
    연결(.join method)
    output = ', '.join([q.question_text for q in latest_question_list])
    return HttpResponse(output)
```

Django Template

- 앞에서 작성한 index view의 문제점 : View 는 Request를 해석하여 Response를 만들기 위한 일련의 로직만을 포함하여야 함.
- 앞에서 작성한 index view의 경우 로직뿐만 아니라 처리된 결과가 어떻게 사용자에게 보여질지 까지 구현하고 있음
- 이렇게 구현 할 경우, 추후 웹 서비스의 로직은 그대로 두고, 서비스의 디자인/사용자 인터페이스만을 변화하고자 할 때 Python 코드까지 모두 수정해야 하는 문제점을 가지고 있음
- Django Template : Django MVT(Model-View-Template)의 마지막 요소, View에서 처리된 Response를 사용자에게 어떻게 보여줄지 구현하는 요소

Django Template

- “polls/templates/polls” 디렉토리 생성
- 해당 디렉토리에 아래와 같은 “index.html” 생성

```
{% if latest_question_list %}
    <ul>
    {% for question in latest_question_list %}
        <li><a href="/polls/{{ question.id }}/">{{ question.question_text }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```


Django Template

- index view 수정

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

from polls.models import Question

def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    # template load
    template = loader.get_template('polls/index.html')

    # template에 전달하기 위한 정보들을 dictionary 형태로 구성
    context = {
        'latest_question_list': latest_question_list,
    }

    # render : 전달된 context와 template을 결합하여 html 을 생성
    # HttpResponse : 생성된 html을 HTTP Response로 변환
    return HttpResponse(template.render(context, request))
```

Django Template

- `render()` : template을 로드하고, contex와 결합하여 HttpResponse로 반환하는 작업은 Django View 개발에서 매우 빈번한 작업. 이 작업을 간소화 시켜줌

```
from django.shortcuts import render
from polls.models import Question

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

Django Template

- detail view 수정 : Http404(Not Found) 처리

```
from django.shortcuts import render
from django.http import Http404

def detail(request, question_id):
    # 전달된 id에 해당하는 Question이 존재하지 않을 경우에 대한 처리
    try:
        question = Question.objects.get(id=question_id)
        # objects.get에서 해당 데이터가 존재하지 않을 경우 DoesNotExist 예외 발생
    except Question.DoesNotExist:
        # Http404 예외를 반환
        raise Http404("Question does not exist")
    return render(request, 'polls/detail.html', {'question': question})
```

Django Template

- detail view 수정 : `get_object_or_404()` – `get`을 했는데 없을 경우 자동으로 404를 반환하는 shortcut

```
from django.shortcuts import get_object_or_404, render

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

Django Template

- detail template : question이 주어졌을 때 그에 딸린 choice들을 포함하는 상세정보를 보여주기 위한 template

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

Django Template

- Django Template Language : HTML과 View에서 전달된 Context 속 데이터를 결합하여 HTML을 동적으로 생성하기 위한 언어
- 크게 Variable / Tag / Filter로 구성되어 있음
- Variable : context로 부터 전달된 값을 HTML 상에 표시 할 때 사용하며, {{ 와 }}로 감싸서 표현한다.

```
My first name is {{ first_name }}. My last name is {{ last_name }}.
```

With a context of {'first_name': 'John', 'last_name': 'Doe'}, this template renders to:

```
My first name is John. My last name is Doe.
```

Django Template

- Django Template Language : HTML과 View에서 전달된 Context 속 데이터를 결합하여 HTML을 동적으로 생성하기 위한 언어
- Variable / Tag / Filter로 구성되어 있음
- Variable : context로 부터 전달된 값을 HTML 상에 표시 할 때 사용하며, `{{ 와 }}`로 감싸서 표현한다.

```
My first name is {{ first_name }}. My last name is {{ last_name }}.
```

With a context of `{'first_name': 'John', 'last_name': 'Doe'}`, this template renders to:

```
My first name is John. My last name is Doe.
```

```
{{ my_dict.key }}  
{{ my_object.attribute }}  
{{ my_list.0 }}
```

Django Template

- Tag : Template 내에서 간략한 로직 구현이 필요할 경우 (python control statement와 비슷한 기능을 함)

for

Loops over each item in an array, making the item available in a context variable. For example, to display a list of athletes provided in **athlete_list**:

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
</ul>
```


Django Template

- Tag : Template 내에서 간략한 로직 구현이 필요할 경우 (python control statement와 비슷한 기능을 함)

if

The `{% if %}` tag evaluates a variable, and if that variable is "true" (i.e. exists, is not empty, and is not a false boolean value) the contents of the block are output:

```
{% if athlete_list %}
    Number of athletes: {{ athlete_list|length }}
{% elif athlete_in_locker_room_list %}
    Athletes should be out of the locker room soon!
{% else %}
    No athletes.
{% endif %}
```

Django Template

- Tag : Template 내에서 간략한 로직 구현이 필요할 경우 (python control statement와 비슷한 기능을 함)

include

Loads a template and renders it with the current context. This is a way of "including" other templates within a template.

The template name can either be a variable or a hard-coded (quoted) string, in either single or double quotes.

This example includes the contents of the template "**foo/bar.html**":

```
{% include "foo/bar.html" %}
```

Django Template

- Tag : Template 내에서 간략한 로직 구현이 필요할 경우 (python control statement와 비슷한 기능을 함)

`url`

Returns an absolute path reference (a URL without the domain name) matching a given view and optional parameters. Any special characters in the resulting path will be encoded using `iri_to_uri()`.

This is a way to output links without violating the DRY principle by having to hard-code URLs in your templates:

```
{% url 'some-url-name' v1 v2 %}
```

Django Template

- Django Template Built-in Tag Reference : Django Template에 존재하는 모든 Built-in tag에 대한 Documentation

<https://docs.djangoproject.com/ko/2.2/ref/templates/builtins/#ref-templates-builtins-tags>

Django Template

- Filter : Variable을 특정한 형식으로 변환하여 표시함

date

Formats a date according to the given format.

Uses a similar format as PHP's **date()** function (<https://php.net/date>) with some differences.

```
{{ my_date|date:"Y-m-d" }}
```

Django Template

- Filter : Variable을 특정한 형식으로 변환하여 표시함

truncatechars

Truncates a string if it is longer than the specified number of characters. Truncated strings will end with a translatable ellipsis character ("...").

Argument: Number of characters to truncate to

For example:

```
{{ value|truncatechars:7 }}
```

If **value** is "Joel is a slug", the output will be "Joel i...".

Django Template

- Filter : Variable을 특정한 형식으로 변환하여 표시함

length

Returns the length of the value. This works for both strings and lists.

For example:

```
{{ value|length }}
```

If **value** is `['a', 'b', 'c', 'd']` or `"abcd"`, the output will be **4**.

The filter returns **0** for an undefined variable.

Django Template

- Django Template Built-in Filter Reference : Django Template에 존재하는 모든 Built-in filter에 대한 Documentation

<https://docs.djangoproject.com/ko/2.2/ref/templates/builtins/#ref-templates-builtins-filters>

Django Template

- url tag 사용하여 템플릿 내의 하드코딩된 링크 변경

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

이러한 강력하게 결합되고 하드코딩된 접근방식의 문제는 수 많은 템플릿을 가진 프로젝트들의 URL을 바꾸는 게 어려운 일이 된다는 점입니다. 그러나, **polls.urls** 모듈의 **path()** 함수에서 인수의 이름을 정의했으므로, **{% url %}** template 태그를 사용하여 url 설정에 정의된 특정한 URL 경로들의 의존성을 제거할 수 있습니다.

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

이것이 **polls.urls** 모듈에 서술된 URL의 정의를 탐색하는 식으로 동작합니다. 다음과 같이 'detail'이라는 이름의 URL이 어떻게 정의되어 있는지 확인할 수 있습니다.

```
...  
# the 'name' value as called by the {% url %} template tag  
path('<int:question_id>/', views.detail, name='detail'),  
...
```

App URL Namespace

- 프로젝트 내에 여러 Application이 존재하는 경우 urls.py 내에 존재하는 view name의 충돌 문제 발생
(e.g. polls app에도 index view가 있고, analytics app에도 index view가 있다면?)
- 해결 방법 : app URL Namespace

```
from django.urls import path

from polls import views

# App URL NameSpace 추가
app_name = 'polls'

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

App URL Namespace

- polls/index.html 수정

polls/templates/polls/index.html

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

polls/templates/polls/index.html

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

Study Topic

- Django Tutorial Topic 2(<https://docs.djangoproject.com/ko/2.2/intro/tutorial02/>)
- Django Tutorial Topic 3(<https://docs.djangoproject.com/ko/2.2/intro/tutorial03/>)
- Django Template Language(<https://docs.djangoproject.com/en/2.2/topics/templates/>)
- <http://pythonstudy.xyz/python/article/307-Django-%ED%85%9C%ED%94%8C%EB%A6%BF-Template>