

Django 02

Template & URLs

• INDEX

- Template System
 - Django Template system
 - Django Template Language
- 템플릿 상속
 - 상속 관련 DTL 태그
- HTML form
 - 요청과 응답
 - form 활용

• INDEX

- Django URLs
 - Variable Routing
 - App과 URL
- URL 이름 지정
 - Naming URL patterns
 - DTL URL tag
- URL 이름 공간
 - app_name 속성
- 참고
 - 추가 템플릿 경로
 - DTL 주의사항
 - Trailing Slashes

Template System

Django Template system

Django Template system

데이터 표현을 제어하면서, 표현과 관련된 부분을 담당

HTML의 콘텐츠를 변수 값에 따라 변경하기 (1/2)

```
<!-- articles/index.html -->

<body>
    <h1>Hello, django!</h1>
</body>
```

Hello, django!

HTML의 콘텐츠를 변수 값에 따라 변경하기 (2/2)

```
# articles/views.py
def index(request):
    context = {
        'name': 'Jane',
    }
    return render(request, 'articles/index.html', context)
```

```
<!-- articles/index.html -->
<body>
    <h1>Hello, {{ name }}</h1>
</body>
```

Django Template Language

Django Template Language (DTL)

Template에서 조건, 반복, 변수 등의
프로그래밍적 기능을 제공하는 시스템

DTL Syntax

1. Variable
2. Filters
3. Tags
4. Comments

1. Variable

- `render` 함수의 세번째 인자로 딕셔너리 데이터를 사용
- 딕셔너리 key에 해당하는 문자열이 template에서 사용 가능한 변수명이 됨
- `dot('.)`를 사용하여 변수 속성에 접근할 수 있음

```
 {{ variable }}
```

```
 {{ variable.attribute }}
```

2. Filters

- 표시할 변수를 수정할 때 사용 (변수 + ‘|’ + 필터)
- chained(연결)이 가능하며 일부 필터는 인자를 받기도 함
- 약 60개의 built-in template filters를 제공

```
 {{ variable|filter }}
```

```
 {{ name|truncatewords:30 }}
```

3. Tags

- 반복 또는 논리를 수행하여 제어 흐름을 만듦
- 일부 태그는 시작과 종료 태그가 필요
- 약 24개의 built-in template tags를 제공

{% tag %}

{% if %} {% endif %}

4. Comments

- DTL에서의 주석

```
<h1>Hello, #{ name #}</h1>
```

```
{% comment %}
```

```
...
```

```
{% endcomment %}
```

DTL 예시 (1/2)

국밥 메뉴는 2글자입니다.

메뉴판

- 국밥
- 국수
- 카레
- 탕수육

아직 메뉴가 남았습니다.

DTL 예시 (2/2)

```
# urls.py

urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', views.index),
    path('dinner/', views.dinner),
]
```

```
# views.py

import random

def dinner(request):
    foods = ['국밥', '국수', '카레', '탕수육',]
    picked = random.choice(foods)
    context = {
        'foods': foods,
        'picked': picked,
    }
    return render(request, 'articles/dinner.html', context)
```

```
<!-- articles/dinner.html -->

<p>{{ picked }} 메뉴는 {{ picked|length }}글자 입니다.</p>
<h2>메뉴판</h2>
<ul>
    {% for food in foods %}
        <li>{{ food }}</li>
    {% endfor %}
</ul>

{% if foods|length == 0 %}
    <p>메뉴가 소진 되었습니다.</p>
{% else %}
    <p>아직 메뉴가 남았습니다.</p>
{% endif %}
```

이어서...

삼성 청년 SW 아카데미

템플릿 상속

기본 템플릿 구조의 한계

- 만약 모든 템플릿에 bootstrap을 적용하려면?
 - 모든 템플릿에 bootstrap CDN을 작성해야 할까?

템플릿 상속

Template inheritance

페이지의 공통요소를 포함하고,
하위 템플릿이 재정의 할 수 있는 공간을 정의하는
기본 ‘skeleton’ 템플릿을 작성하여 상속 구조를 구축

템플릿 상속

Template inheritance

- 1
- 2 페이지의 공통요소를 포함하고,
 하위 템플릿이 재정의 할 수 있는 공간을 정의하는
 기본 ‘skeleton’ 템플릿을 작성하여 상속 구조를 구축

상속 구조 만들기 (1/2)

skeleton 역할을 하게 되는 상위 템플릿(base.html) 작성

```
<!-- articles/base.html -->

<!DOCTYPE html>
<html lang="en">
<head>

    ...
    {% comment %} bootstrap CDN 생략 {% endcomment %}
</head>
<body>
    {% block content %}
    {% endblock content %}
    {% comment %} bootstrap CDN 생략 {% endcomment %}
</body>
</html>
```

상속 구조 만들기 (2/2)

기존 하위 템플릿의 변화

```
<!-- articles/index.html -->  
  
{% extends 'articles/base.html' %}  
  
{% block content %}  
  <h1>Hello, {{ name }}</h1>  
  {% endblock content %}
```

```
<!-- articles/dinner.html -->  
  
{% extends 'articles/base.html' %}  
  
{% block content %}  
  <p>{{ picked }} 메뉴는 {{ picked|length }}글자 입니다.</p>  
  
  <h2>메뉴판</h2>  
  <ul>  
    {% for food in foods %}  
      <li>{{ food }}</li>  
    {% endfor %}  
  </ul>  
  
  {% if foods|length == 0 %}  
    <p>메뉴가 소진되었습니다.</p>  
  {% else %}  
    <p>아직 메뉴가 남았습니다.</p>  
  {% endif %}  
  {% endblock content %}
```

상속 관련 DTL 태그

‘extends’ tag

```
{% extends 'path' %}
```

자식(하위)템플릿이 부모 템플릿을 확장한다는 것을 알림

- ❖ 반드시 자식 템플릿 최상단에 작성되어야 함 (2개 이상 사용 불가)

‘block’ tag

```
{% block name %}{% endblock name %}
```

하위 템플릿에서 재정의 할 수 있는 블록을 정의
(상위 템플릿에 작성하며
하위 템플릿이 작성할 수 있는 공간을 지정하는 것)

하위 템플릿이 재정의 할 수 있는 block 영역

```
<!-- articles/base.html -->  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
...  
  {% comment %} bootstrap CDN 생략 {% endcomment %}  
</head>  
<body>  
  {% block content %}  
  {% endblock content %}  
  {% comment %} bootstrap CDN 생략 {% endcomment %}  
</body>  
</html>
```

```
<!-- articles/index.html -->  
  
{% extends 'articles/base.html' %}  
  
  {% block content %}  
    <h1>Hello, {{ name }}</h1>  
  {% endblock content %}
```

이어서...

삼성 청년 SW 아카데미

HTML form

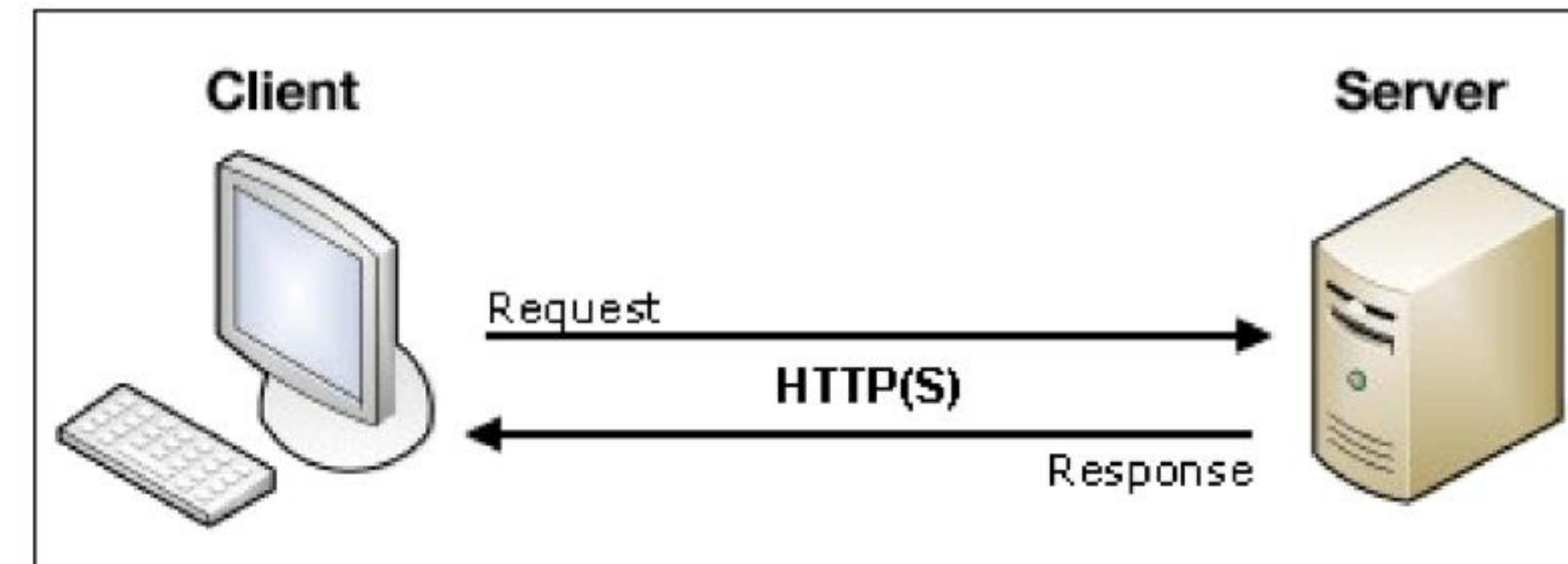
요청과 응답

데이터를 보내고 가져오기

Sending and Retrieving form data

HTML ‘form’ element를 통해
사용자와 애플리케이션 간의 상호작용 이해하기

**HTML ‘form’은
HTTP 요청을 서버에 보내는 가장 편리한 방법**



HTML ‘form’은 HTTP 요청을 서버에 보내는 가장 편리한 방법

아이디 :

패스워드 :

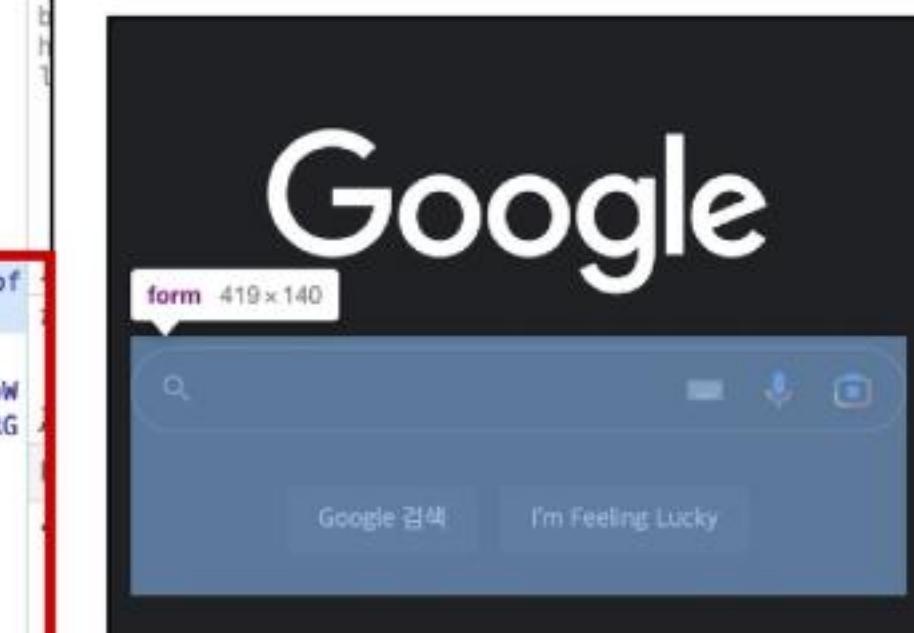
```
<form action="#" method="GET">
  <div>
    <label for="name">아이디 : </label>
    <input type="text" id="name">
  </div>
  <div>
    <label for="password">패스워드 : </label>
    <input type="password" name="password" id="password">
  </div>
  <input type="submit" value="로그인">
</form>
```

실제 웹 서비스에서 form이 사용되는 예시

네이버 & 구글의 로그인 화면에서 사용하는 HTML form 요소



```
<head> ...
<body>
  <div id="wrap" class="wrap">
    <div class="u_skip"> ...
    <header class="header" role="banner"> ...
    <div id="container" class="container">
      <!-- content -->
      <div class="content">
        <div class="login_wrap">
          <ul class="menu_wrap" role="tablist"> ...
          <form id="frmNIDLogin" name="frmNIDLogin" target="_top" autocomplete="off" action="https://nid.naver.com/nidlogin.login" method="POST"> ...
            <input type="hidden" id="localechange" name="localechange" value="">
            <input type="hidden" name="dynamicKey" id="dynamicKey" value="4DfScupWA-btII8Q2DbF1zzAkv8_EVAMcBbf1UT0Zn3h49bIgtV5JuSvShGCZ6upwPmBqjEz3bgARG16jQ34XtXntVhupRdI_7Tgt8g4do">
            <input type="hidden" name="encpw" id="encpw" value="">
            <input type="hidden" name="enctp" id="enctp" value="1">
            <input type="hidden" name="svctype" id="svctype" value="1">
            <input type="hidden" name="smart_LEVEL" id="smart_LEVEL" value="1">
            <input type="hidden" name="bvsd" id="bvsd" value="">
            <input type="hidden" name="encnm" id="encnm" value="">
            <input type="hidden" name="locale" id="locale" value="ko_KR">
            <input type="hidden" name="url" id="url" value="https://www.naver.co...
          </form>
        </div>
      </div>
    </div>
  </div>
</body>
```



```
<div class="L3eUgb" data-hveid="1" data-hvtype="flex">
  <div class="o3j99 n1xJcf Ne6nSd"> ...
  <div class="o3j99 LLD4me yr19Zb LS80J"> ...
<div class="o3j99 ikrT4e om7nvf">
  <style>.om7nvf{padding:20px}</style>
</div>
<dialog class="spch-dlo" id="spch-dlo"> ...
</dialog>
<form action="/search" autocomplete="off" method="GET" role="search"> ...
  <div jsmodel="sj77Re P9Kqfe" jsdata="MuIEvd;_B0Y0LE">
    <div jscontroller="cnjECf" jsmodel="VYkzu a4L2gc QmwFZb BFDhle gx0hCb TnHSdd icvlie" class="A8SBwf" data-adhe="false" data-alt="false" data-hp="true" data-mlcf="false" data-mle="false" data-mof="false" jsdata="LVplcb;_;" jsaction="LX6RMd:w3Wsmc:DkpM8b:d3sQLd;IQ0avd:dFy0Ef;XzZZPe;jI3wzf;Aghsf:AVsnlb;iHd9U:Q7Cnrc;f5hEHc:G0jgYd;vmxUb:j3bJnb;H0ZJe:FrsEXe;R2c50:LuRugf;qiCkJd:ANdidc;htNNz:SNIJTD;N0g9L:HLgh3;uGoIkD:epUokb;zLdLw:eaGB5:H9muVd:J4e6lb;hBEIVb:nUZ9le;rcuQ6b:npT2md">
    <style> ...
    <style> CKh9sd/hackground:none;display:flex;flex:0 0 auto;</style>
  </div>
</form>
```

‘form’ element

사용자로부터 할당된 데이터를 서버로 전송

- 웹에서 사용자 정보를 입력하는 여러 방식 (text, password, checkbox 등)을 제공

fake Naver 실습 (1/4)

```
# urls.py

urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', views.index),
    path('dinner/', views.dinner),
    path('search/', views.search),
]
```

```
# views.py

def search(request):
    return render(request, 'articles/search.html')
```

```
<!-- articles/search.html -->

{% extends 'articles/base.html' %}

{% block content %}
<form action="#" method="GET">
    <label for="message">검색어</label>
    <input type="text" name="message" id="message">
    <input type="submit" value="submit">
</form>
{% endblock content %}
```

fake Naver 실습 (2/4)

input에 hello를 입력하고 제출 버튼을 누른 후 브라우저의 URL 변화 확인

http://127.0.0.1:8000/search/?**message=hello**

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

{% block content %}
    <form action="#" method="GET">
        <label for="message">검색어</label>
        <input type="text" name="message" id="message">
        <input type="submit" value="submit">
    </form>
{% endblock content %}
```

input에 입력한 데이터

fake Naver 실습 (3/4)

실제 Naver에서 검색 후 URL 확인

input의 name 속성

<https://search.naver.com/search.naver?query=hello>

목적지 URL

input에 입력한 데이터

fake Naver 실습 (4/4)

```
<!-- articles/search.html -->

{% extends 'articles/base.html' %}

{% block content %}
    <form action="https://search.naver.com/search.naver/" method="GET">
        <label for="message">검색어</label>
        <input type="text" name="query" id="message">
        <input type="submit" value="submit">
    </form>
{% endblock content %}
```

‘action’ & ‘method’ **form의 핵심 속성 2가지**

“데이터를 어디(**action**)로 어떤 방식(**method**)으로 요청할지”

action과 method

- **action**
 - 입력 데이터가 전송될 URL을 지정 (목적지)
 - 만약 이 속성을 지정하지 않으면 데이터는 현재 form이 있는 페이지의 URL로 보내짐
- **method**
 - 데이터를 어떤 방식으로 보낼 것인지 정의
 - 데이터의 HTTP request methods (GET, POST)를 지정

'input' element

사용자의 데이터를 입력 받을 수 있는 요소
(**type** 속성 값에 따라 다양한 유형의 입력 데이터를 받음)

- 핵심 속성 - 'name'

‘name’ attribute

- input의 핵심 속성
- 사용자가 입력한 데이터에 붙이는 이름(key)
- 데이터를 제출했을 때 서버는 name 속성에 설정된 값을 통해서만 사용자가 입력한 데이터에 접근할 수 있음

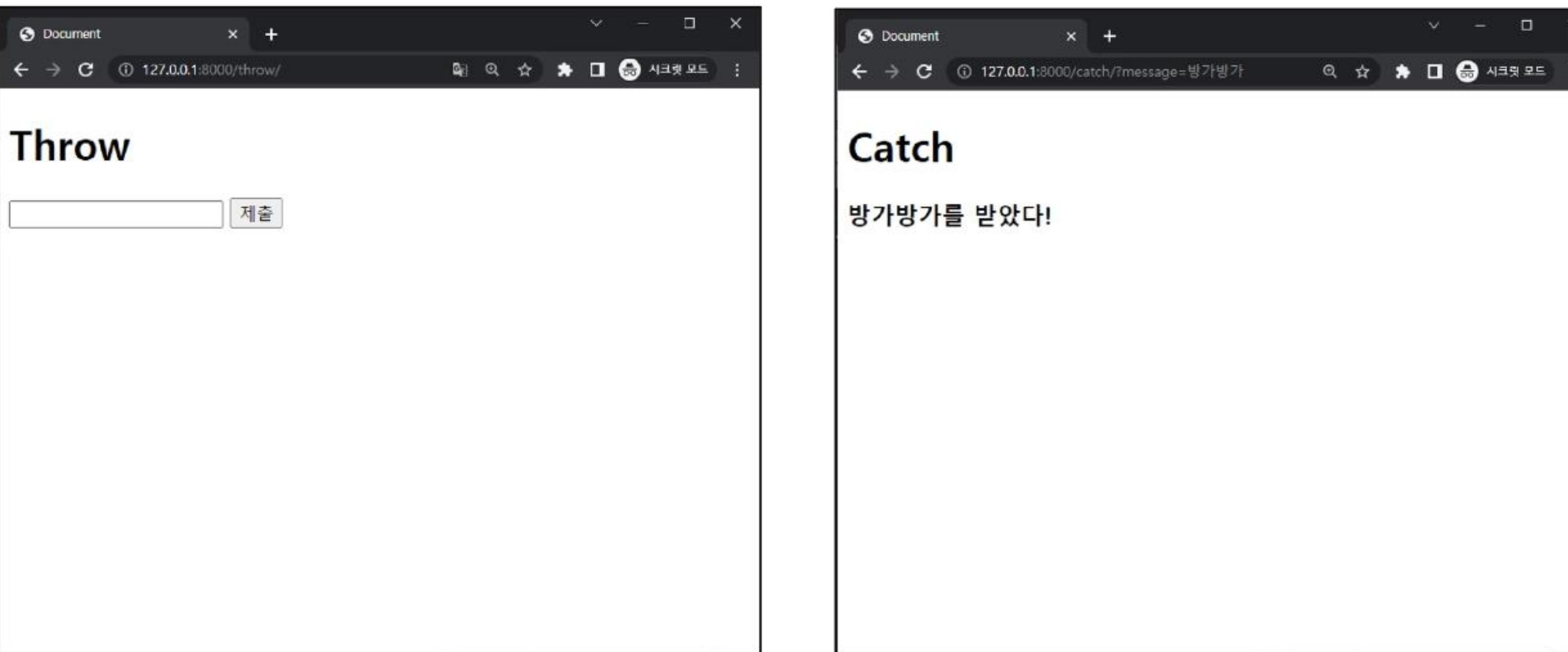
Query String Parameters

- 사용자의 입력 데이터를 URL 주소에 파라미터를 통해 서버로 보내는 방법
- 문자열은 앤퍼샌드('&)로 연결된 **key=value** 쌍으로 구성되며,
기본 URL과는 물음표('?')로 구분됨
- 예시
 - **http://host:port/path?key=value&key=value**

form 활용

사용자 입력 데이터를 받아 그대로 출력하는 서버 만들기

view 함수는 몇 개가 필요할까?



1. throw 로직 작성

```
# urls.py  
  
urlpatterns = [  
    path('throw/', views.throw),  
]
```

```
# views.py  
  
def throw(request):  
    return render(request, 'articles/throw.html')
```

```
<!-- articles/throw.html -->  
  
{% extends 'articles/base.html' %}  
  
{% block content %}  
    <h1>Throw</h1>  
    <form action="/catch/" method="GET">  
        <input type="text" id="message" name="message">  
        <input type="submit">  
    </form>  
{% endblock content %}
```

2. catch 로직 작성

throw 페이지에서 요청한 사용자 입력 데이터는 어떻게 가져와야 할까?

```
# urls.py
urlpatterns = [
    path('catch/', views.catch),
]
```

```
# views.py
def catch(request):
    context = ????
    return render(request, 'articles/catch.html')
```

```
<!-- articles/catch.html -->
{% extends 'articles/base.html' %}

{% block content %}
    <h1>Catch</h1>
    <h3>{{ ??? }}를 받았습니다!</h3>
{% endblock content %}
```

HTTP request 객체

form으로 전송한 데이터 뿐만 아니라
Django로 들어오는 모든 요청 관련 데이터가 담겨 있음
(view 함수의 첫번째 인자로 전달됨)

request 객체 살펴보기

```
# views.py

def catch(request):
    print(request)
    print(type(request))
    print(dir(request))
    print(request.GET)
    print(request.GET.get('message'))
    return render(request, 'articles/catch.html')
```

```
<WSGIRequest: GET '/catch/?message=%EC%95%88%EB%85%95%21'>

<class 'django.core.handlers.wsgi.WSGIRequest'>

['COOKIES', 'FILES', 'GET', 'META', 'POST', '__class__', ... 생략... 'parse_file_upload', 'path', 'path_info', 'read', 'readline', 'readlines', 'resolver_match', 'scheme', 'session', 'upload_handlers', 'user']

<QueryDict: {'message': ['안녕!']}>
```

안녕!

출력 결과

request 객체에서 form 데이터 추출

```
request.GET.get('message')
```

```
<QueryDict: {'message': ['안녕!']}
```

딕셔너리의 `get` 메서드를 사용해
키의 값을 조회

3. catch 로직 마무리

```
# views.py

def catch(request):
    message = request.GET.get('message')
    context = {
        'message': message,
    }
    return render(request, 'articles/catch.html', context)
```

```
<!-- articles/catch.html -->

{% extends 'articles/base.html' %}

{% block content %}
    <h1>Catch</h1>
    <h3>{{ message }}를 받았습니다!</h3>
{% endblock content %}
```

throw - catch 간 요청과 응답 정리 (1/2)



1. throw/ 로 요청 (throw 페이지를 줘!)



5. 응답 객체 전달



6. 응답 객체 해석 후 화면 출력

2. throw/ 문자열과 일치하는 `urls.py`의 path 함수 호출
3. `throw view` 함수 호출
4. `throw view` 함수가 응답 객체를 반환

throw - catch 간 요청과 응답 정리 (2/2)



1. throw 페이지에서 **form** 데이터 작성 후 제출
(**form** 요소의 **action** 속성 값으로 요청)

2. **catch/**로 요청 (+ 사용자 입력 데이터와 함께)

6. 응답 객체 전달



7. 응답 객체 해석 후 화면 출력

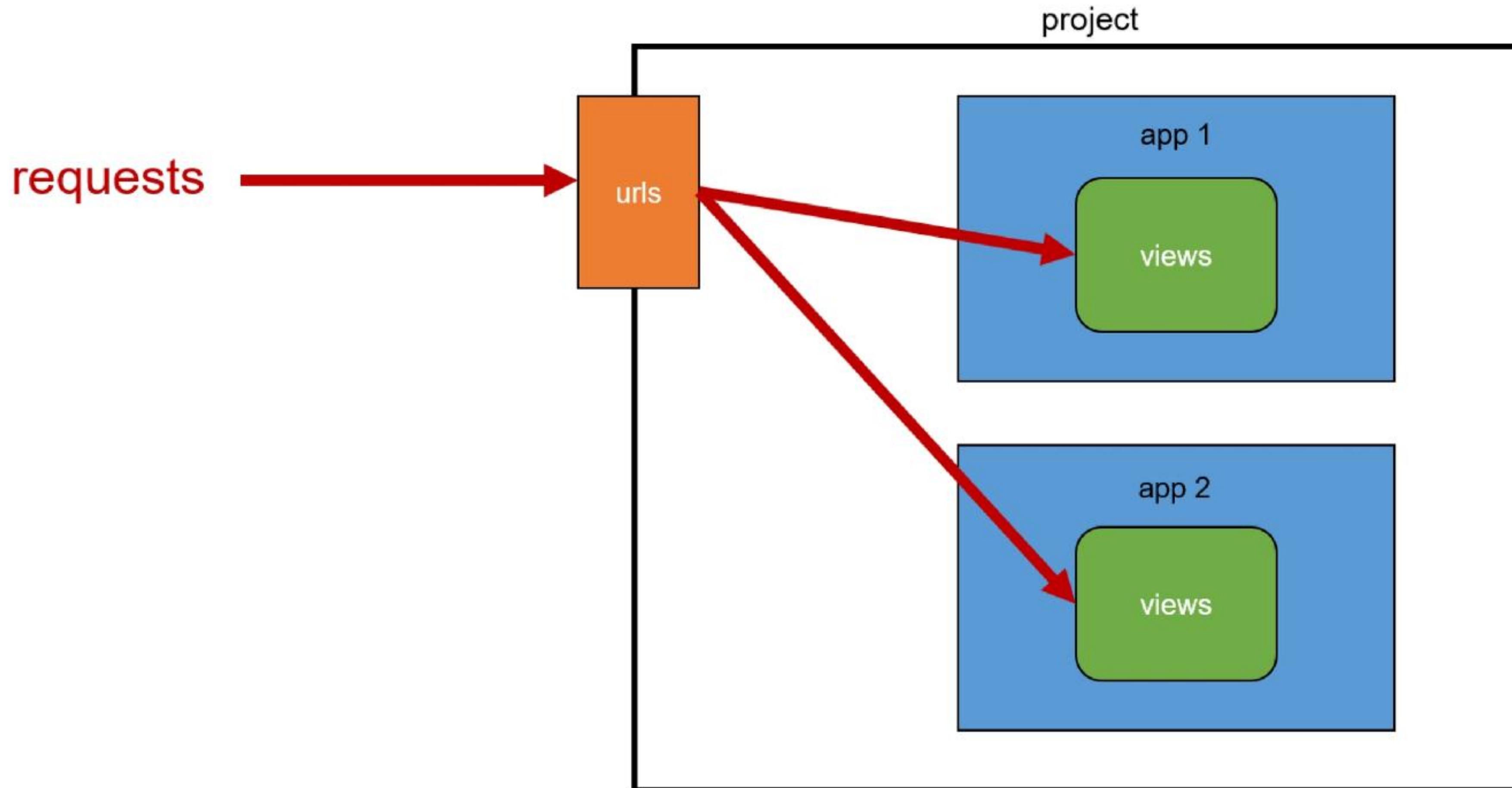
3. **catch/** 문자열과 일치하는
urls.py의 **path** 함수 호출
4. **catch view** 함수 호출
5. **catch view** 함수에서 사용자가 보낸
form 데이터 추출 후 응답 객체를 반환

이어서...

삼성 청년 SW 아카데미

Django URLs

요청과 응답에서 Django URLs의 역할



URL dispatcher

(운항 관리자, 분배기)

URL 패턴을 정의하고
해당 패턴이 일치하는 요청을 처리할 view 함수를 연결(매핑)

Variable Routing

현재 URL 관리의 문제점

템플릿의 많은 부분이 중복되고, URL의 일부만 변경되는 상황이라면
계속해서 비슷한 URL과 템플릿을 작성해 나가야 할까?

```
urlpatterns = [
    path('articles/1/', ...),
    path('articles/2/', ...),
    path('articles/3/', ...),
    path('articles/4/', ...),
    path('articles/5/', ...),
    ...,
]
```

Variable Routing

URL 일부에 변수를 포함시키는 것
(변수는 view 함수의 인자로 전달 할 수 있음)

Variable routing 작성법

<path_converter:variable_name>

```
path('articles/<int:num>', views.detail)
path('hello/<str:name>', views.greeting)
```

Path converters

URL 변수의 타입을 지정
(str, int 등 5가지 타입 지원)

Variable routing 실습 (1/2)

```
# urls.py

urlpatterns = [
    path('hello/<str:name>', views.greeting),
]
```

```
# views.py

def greeting(request, name):
    context = {
        'name': name,
    }
    return render(request, 'articles/greeting.html', context)
```

```
<!-- articles/greeting.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>Greeting</h1>
    <h3>{{ name }}님 안녕하세요 !</h3>
{% endblock content %}
```

Variable routing 실습 (2/2)

```
# urls.py

urlpatterns = [
    path('articles/<int:num>', views.detail),
]
```

```
# views.py

def detail(request, num):
    context = {
        'num': num,
    }
    return render(request, 'articles/detail.html', context)
```

```
<!-- articles/detail.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>Detail</h1>
    <h3>{{ num }}번 글 입니다.</h3>
{% endblock content %}
```

App과 URL

App URL mapping

각 앱에 URL을 정의하는 것

- 프로젝트와 각 앱이 URL을 나누어 관리를 편하게 하기 위함

2번째 앱 pages 생성 후 발생할 수 있는 문제

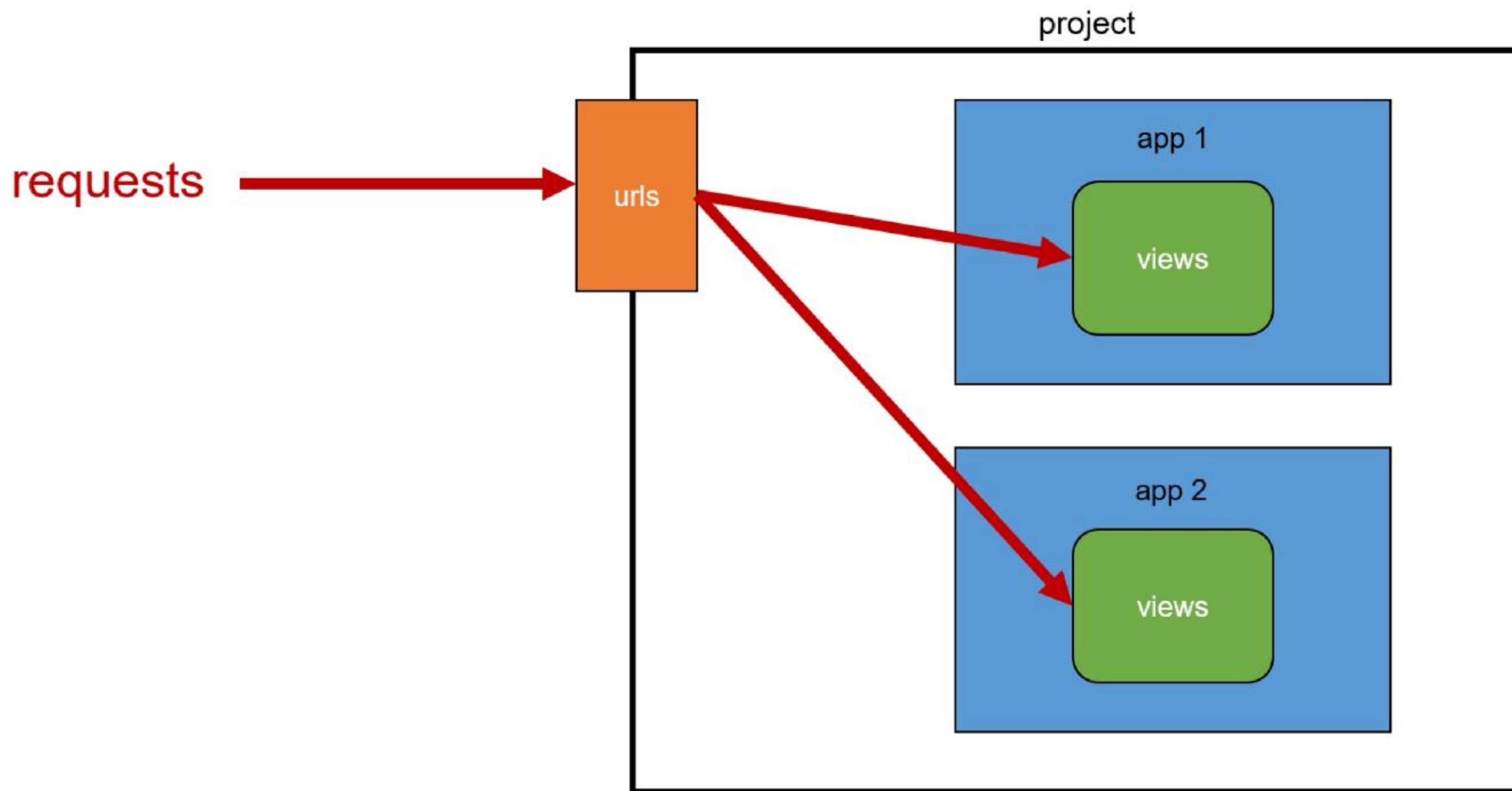
- view 함수 이름이 같거나 같은 패턴의 URL 주소를 사용하게 되는 경우
 - 아래 코드와 같이 해결해 볼 수 있으나 더 좋은 방법이 필요
- “URL을 각자 app에서 관리하자”

```
# firstpjt/urls.py

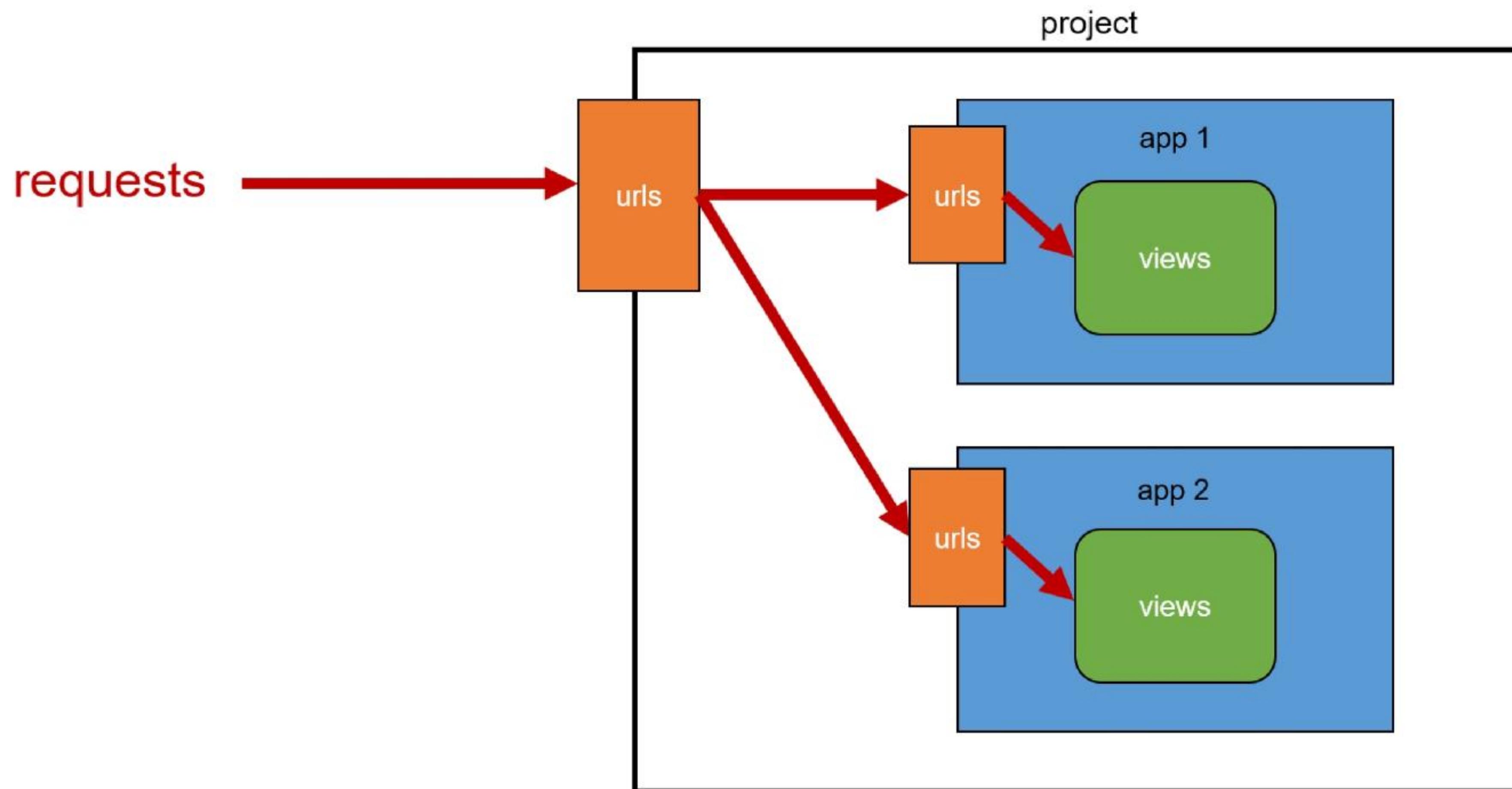
from articles import views as articles_views
from pages import views as pages_views


urlpatterns = [
    ...,
    path('pages', pages_views.index),
]
```

기존 url 구조



변경된 url 구조



url 구조 변화

```
# firstpjt/urls.py
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    # path('index/', views.index),
    # path('dinner/', views.dinner),
    # path('search/', views.search),
    # path('throw/', views.throw),
    # path('catch/', views.catch),
    # path('articles/<int:num>', views.detail),
    # path('hello/<str:name>', views.greeting),
    # path('index/', views.index),
    path('articles/', include('articles.urls')),
    path('pages/', include('pages.urls')),
]
```

```
# articles/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('index/', views.index),
    path('dinner/', views.dinner),
    path('search/', views.search),
    path('throw/', views.throw),
    path('catch/', views.catch),
    path('<int:num>', views.detail),
    path('hello/<str:name>', views.greeting),
]

# pages/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('index/', views.index)
]
```

include()

프로젝트 내부 앱들의 URL을 참조할 수 있도록 맵핑하는 함수

- URL의 일치하는 부분까지 잘라내고,
남은 문자열 부분은 후속 처리를 위해 include된 URL로 전달

include 적용

변경된 프로젝트의 urls.py

```
# firstpjt/urls.py

from django.urls import path, include

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

이어서...

삼성 청년 SW 아카데미

URL 이름 지정

Naming URL patterns

url 구조 변경에 따른 문제점

- 기존 ‘articles/’ 주소가 ‘articles/index/’로 변경됨에 따라 해당 url을 사용하는 모든 위치를 찾아가 변경해야 함
 - ▶ “URL에 이름을 지어주면 이름만 기억하면 되지 않을까?”

```
# firstpjt/urls.py  
  
path('articles/', include('articles.urls'))
```

```
# articles/urls.py  
  
path('index/', views.index, name='index')
```

Naming URL patterns

URL에 이름을 지정하는 것
(path 함수의 name 인자를 정의해서 사용)

Naming URL patterns 적용

path 함수의 name 키워드 인자를 정의

```
# articles/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('index/', views.index, name='index'),
    path('dinner/', views.dinner, name='dinner'),
    path('search/', views.search, name='search'),
    path('throw/', views.throw, name='throw'),
    path('catch/', views.catch, name='catch'),
    path('articles/<int:num>/', views.detail, name='detail'),
    path('hello/<str:name>/', views.greeting, name='greeting'),
]
```

```
# pages/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('index/', views.index, name='index'),
]
```

URL 표기 변화

url을 작성하는 모든 곳에서 변경
(a 태그의 href 속성 값 뿐만 아니라 form의 action 속성 등도 포함)

```
<!-- articles/index.html -->  
  
{% extends 'base.html' %}  
  
{% block content %}  
  <h1>Hello, {{ name }}</h1>  
  <a href="/dinner/">dinner</a>  
  <a href="/search/">search</a>  
  <a href="/throw/">throw</a>  
{% endblock content %}
```



```
<!-- articles/index.html -->  
  
{% extends 'base.html' %}  
  
{% block content %}  
  <h1>Hello, {{ name }}</h1>  
  <a href="{% url 'dinner' %}">dinner</a>  
  <a href="{% url 'search' %}">search</a>  
  <a href="{% url 'throw' %}">throw</a>  
{% endblock content %}
```

DTL URL tag

‘url’ tag

```
{% url 'url name' arg1 arg2 %}
```

주어진 URL 패턴의 이름과 일치하는 절대 경로 주소를 반환

url tag 적용 후 브라우저 출력 확인



The screenshot shows a browser's developer tools with the 'Elements' tab selected. On the left, the rendered HTML content is displayed, featuring a large 'Hello, Jane' heading and three blue underlined links: 'dinner', 'search', and 'throw'. On the right, the raw HTML source code is shown, with the 'throw' link's href attribute highlighted by a red rectangle. The source code is as follows:

```
<!DOCTYPE html>
...<html lang="en"> == $0
  <head> ...</head>
  <body>
    <h1>Hello, Jane</h1>
    <a href="/articles/dinner/">dinner</a>
    <a href="/articles/search/">search</a>
    <a href="/articles/throw/">throw</a>
  </body>
</html>
```

이어서...

삼성 청년 SW 아카데미

URL 이름 공간

app_name 속성

URL 이름 지정 후 남은 문제

- articles 앱의 url 이름과 pages 앱의 url 이름이 같은 상황
- 단순히 이름만으로는 완벽하게 분리할 수 없음
 - “이름에 성(key)을 붙이자”

```
# articles/urls.py  
  
path('index/', views.index, name='index')
```

```
# pages/urls.py  
  
path('index/', views.index, name='index')
```

'app_name' 속성 지정

app_name 변수 값 설정

```
# articles/urls.py
```

```
app_name = 'articles'  
urlpatterns = [  
    ...,  
]
```

```
# pages/urls.py
```

```
app_name = 'pages'  
urlpatterns = [  
    ...,  
]
```

URL tag의 최종 변화

마지막으로 url 태그가 사용하는 모든 곳의 표기 변경하기

{% url ‘**app_name**:**path_name**’ %}

{% url 'index' %}



{% url 'articles:index' %}

이어서...

삼성 청년 SW 아카데미

참고

추가 템플릿 경로

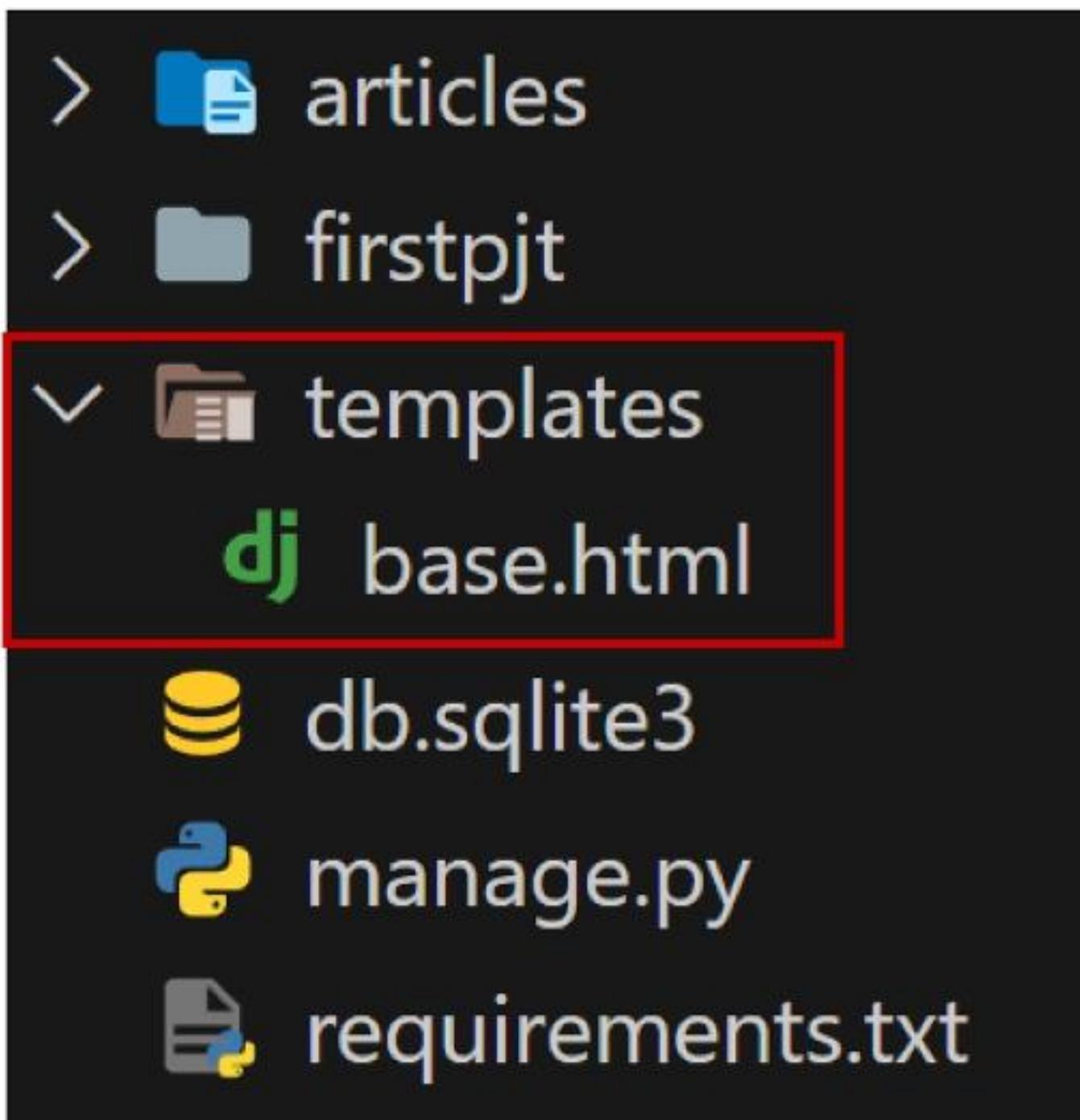
추가 템플릿 경로 지정 (1/2)

템플릿 기본 경로 외 커스텀 경로 추가하기

```
# settings.py

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            BASE_DIR / 'templates',
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            ...
        },
    },
]
```

추가 템플릿 경로 지정 (2/2)



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

각 하위 템플릿에서
extends 경로가 수정됨

새로운 템플릿 경로 생성

BASE_DIR

settings에서 경로지정을 편하게 하기 위해 최상단 지점을 지정해 둔 변수

```
# settings.py  
  
BASE_DIR = Path(__file__).resolve().parent.parent
```

03_DJANGO_TEMPLATE

- > articles
- > firstpjt
- > templates
- db.sqlite3
- manage.py
- requirements.txt

BASE_DIR 경로 위치

- ❖ Python의 객체 지향 파일 시스템 경로
- ❖ <https://docs.python.org/ko/3.9/library/pathlib.html#module-pathlib>

DTL 주의사항

DTL 주의사항

- Python처럼 일부 프로그래밍 구조(if, for 등)를 사용할 수 있지만 명칭을 그렇게 설계 했을 뿐이지 **Python 코드로 실행되는 것이 아니며 Python과는 관련 없음**
- 프로그래밍적 로직이 아니라 표현을 위한 것임을 명심하기
- 프로그래밍적 로직은 되도록 view 함수에서 작성 및 처리할 것
- 공식문서를 참고해 다양한 태그와 필터 사용해보기
 - <https://docs.djangoproject.com/en/4.2/ref/templates/builtins/>

Trailing Slashes

URL의 Trailing Slashes

- Django는 URL 끝에 ‘/’가 없다면 자동으로 붙임
- “기술적인 측면에서, `foo.com/bar`와 `foo.com/bar/`는 서로 다른 URL”
 - 검색 엔진 로봇이나 웹 트래픽 분석 도구에서는 이 두 주소를 서로 다른 페이지로 보기 때문
- 그래서 Django는 검색 엔진이 혼동하지 않게 하기 위해 무조건 붙이는 것을 선택한 것
- 그러나 모든 프레임워크가 이렇게 동작하는 것은 아니니 주의

다음 시간에
만나요!

삼성 청년 SW 아카데미