

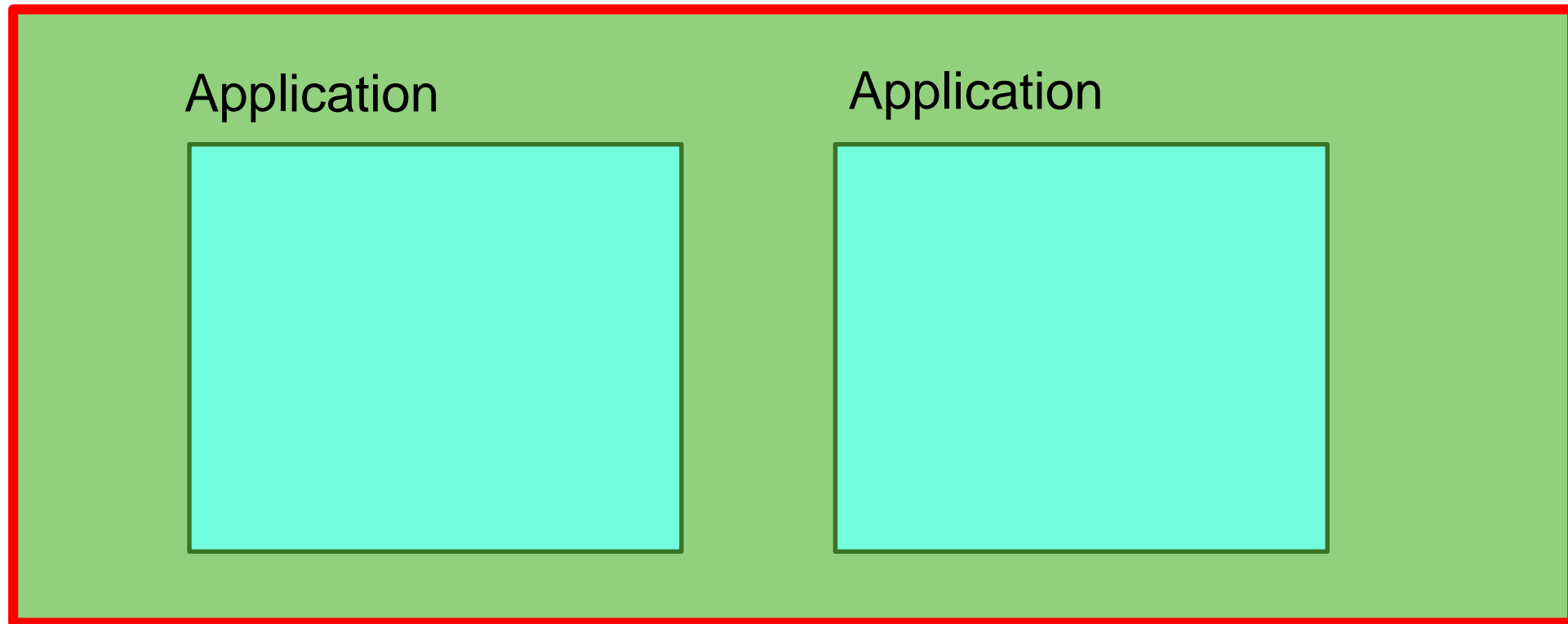
# Django 기초

# Django Project

---

- 웹 어플리케이션의 전체 구조와 설정을 포함하는 작업 단위
- URL, DB 및 기타 설정 등을 포함

Project

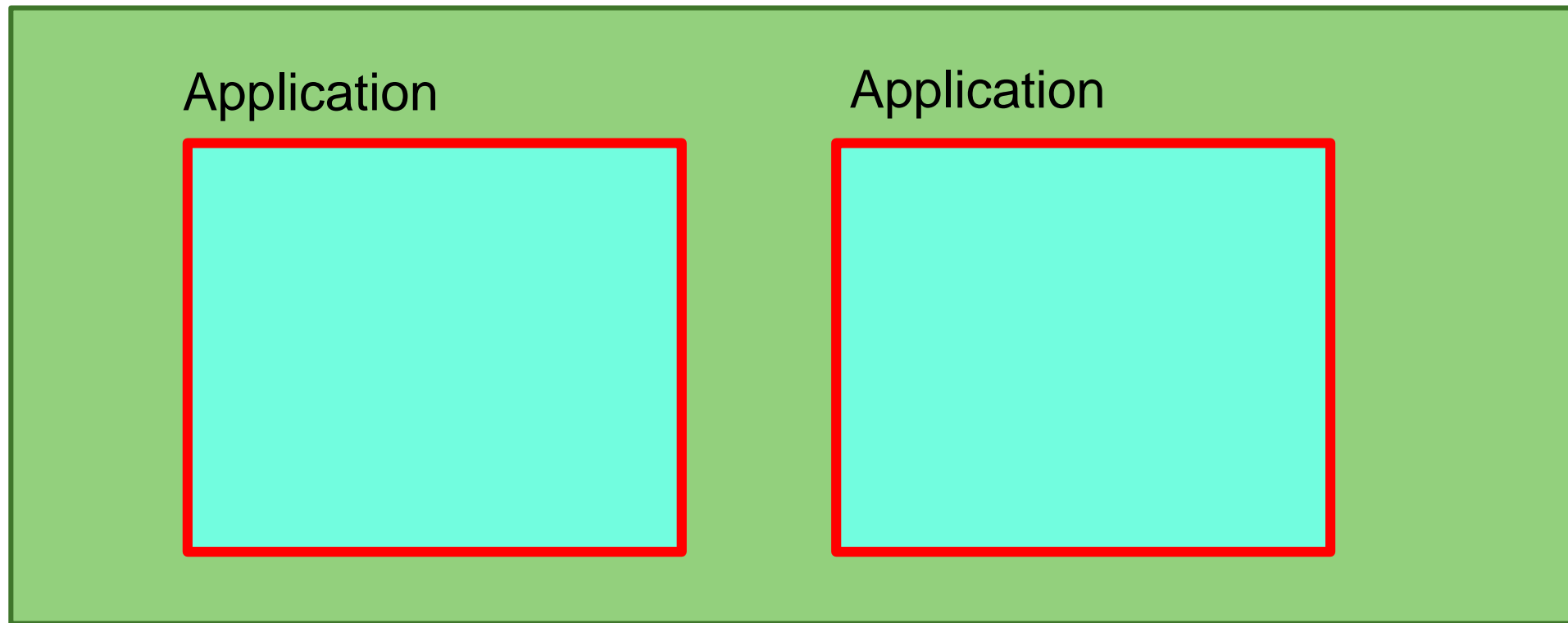


# Django Application

---

- 프로젝트의 구성 요소로 특정 기능을 담당하는 독립적인 모듈
- 여러 application이 모여 하나의 project를 형성

Project

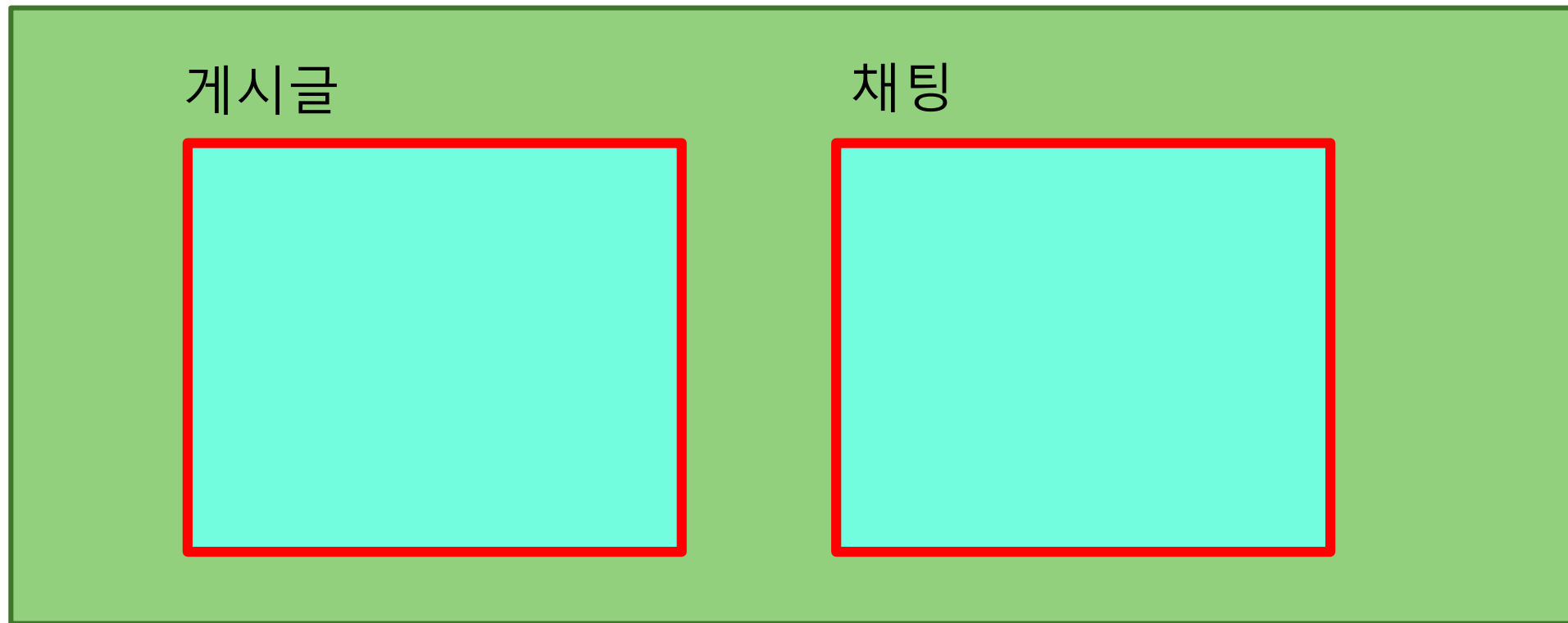


# Django project/application 예시

---

- Project: 블로그
- Application: 게시물, 채팅

블로그



# Django Application 생성

---

- Django Application 생성 명령어

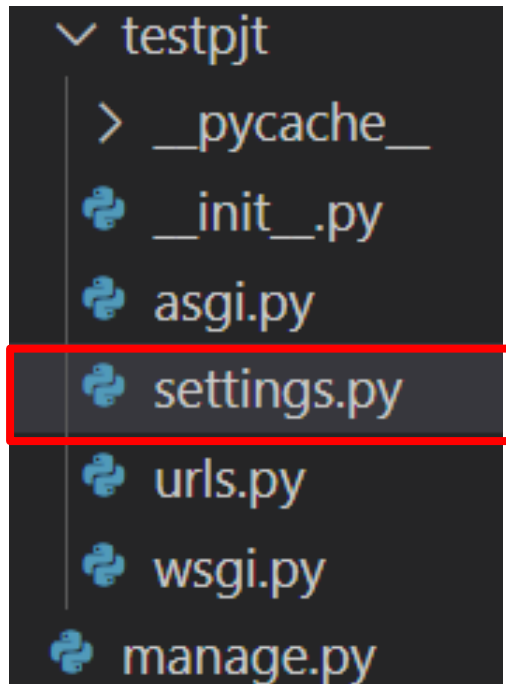
```
$ python manage.py startapp articles
```

- 생성 후 디렉토리 구조

```
▼ articles
  > migrations
  📄 __init__.py
  📄 admin.py
  📄 apps.py
  📄 models.py
  📄 tests.py
  📄 views.py
  > testpjt
  ≡ db.sqlite3
  📄 manage.py
```

# Django Application을 project에 등록

- Application 은 project 보다 하위 폴더이지만, 같은 위치에 존재하기 때문에 projec에 생성한 application 을 등록해야 함!!!
- {생성한 project}/settings.py 에서 INSTALLED\_APPS에 APP 추가



```
31 # Application definition
32
33 INSTALLED_APPS = [
34     'articles',
35     'django.contrib.admin',
36     'django.contrib.auth',
37     'django.contrib.contenttypes',
38     'django.contrib.sessions',
39     'django.contrib.messages',
40     'django.contrib.staticfiles',
41 ]
```

# Django Project를 DB에 연결

---

Mysqclient 설치 및 settings.py 설정 변경

```
$ pip install mysqlclient
```

```
# settings.py
# Database
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases

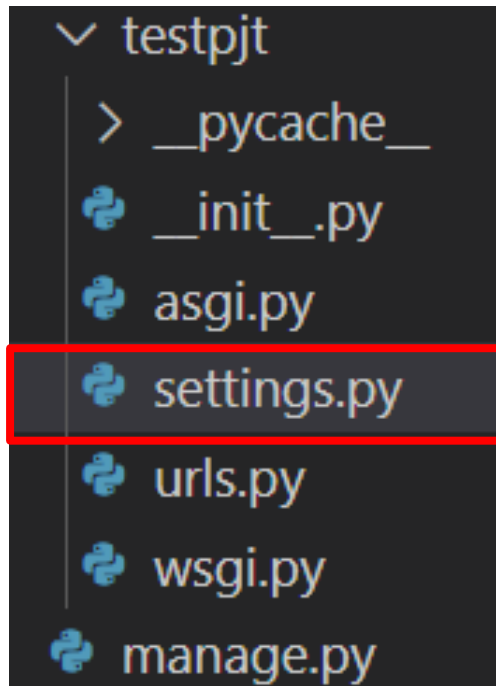
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'bigdata',
        'USER' : 'root',
        'PASSWORD' : 'root', # 설정한 비밀번호로 적어주면 된다.
        'HOST' : '127.0.0.1',
        'PORT' : '3306',
    }
}
```

# Django Project 구조

---

## 1. settings.py

- 프로젝트의 모든 설정을 관리
- App 연결/ 보안 설정 / DB 연결 설정 등 다양한 설정 관리



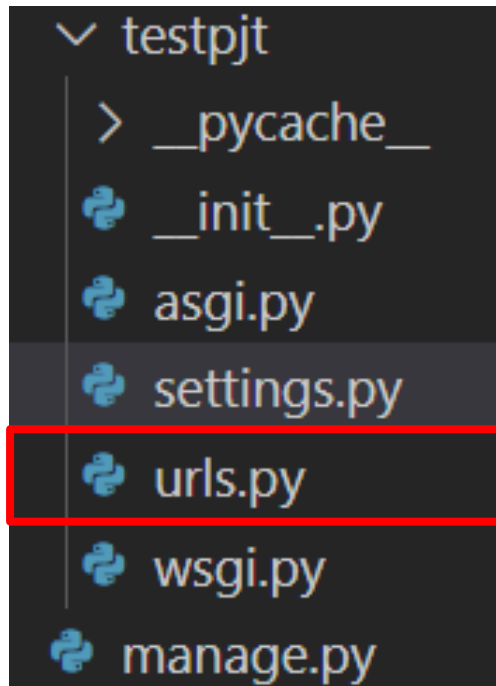


# Django Project 구조

---

## 2. urls.py

- 입력 받은 url 의 적절한 views 를 찾아 연결해주는 곳
- 가장 많이 수정하는 영역

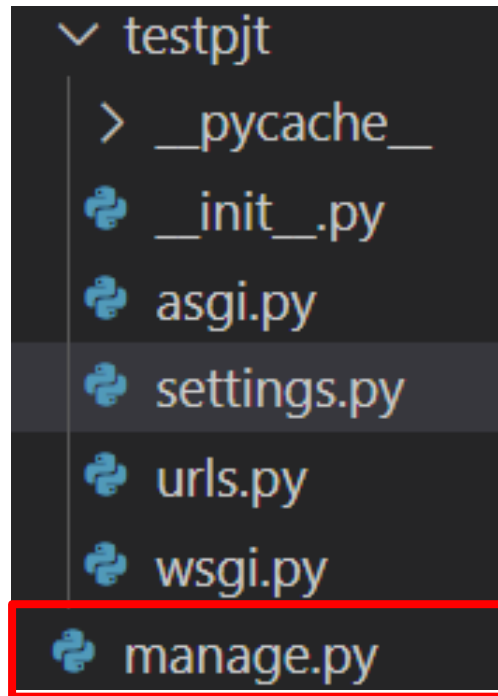


# Django Project 구조

---

## 3. manage.py

- Django 명령행 도구로 다양한 작업을 수행할 수 있도록 도와주는 영역
- DB 관리, App 관리, Server 관리 등 다양한 관리 작업을 실행



# Django Project 구조

---

## 4. `__init__.py`

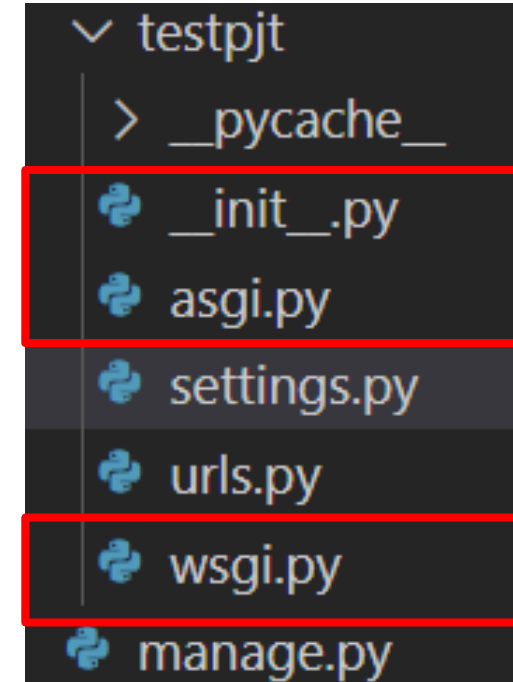
- 해당 폴더를 패키지로 인식하도록 하는 설정

## 5. `asgi.py`

- 비동기식 웹 서버와 연결 설정을 하는 곳

## 6. `wsgi.py`

- 웹 서버와 연결 설정을 하는 곳



# Design Pattern

---

소프트웨어 설계에서 발생하는 문제에 대한 해결책을 제공하는 구조

## MVC Design Pattern

- 소프트웨어 설계시 사용하는 대표적인 디자인 패턴
- Model / View / Controller 디자인으로 구조화  
=> (데이터, UI, 비즈니스 로직으로 분리해서 작업)

# Django Design Pattern

---

Django 에서는 python 철학에 맞춰 mvc 를 명칭만 다르게 해서 사용

## MTV Design Pattern

- Model / Template/ View 디자인으로 구조화  
=> (데이터, UI, 비즈니스 로직으로 분리해서 작업)
- MVC design pattern 과 완전히 동일하나 명칭만 다르게 사용

# Django Design Pattern

---

Django 에서는 python 철학에 맞춰 mvc 를 명칭만 다르게 해서 사용

## MTV Design Pattern

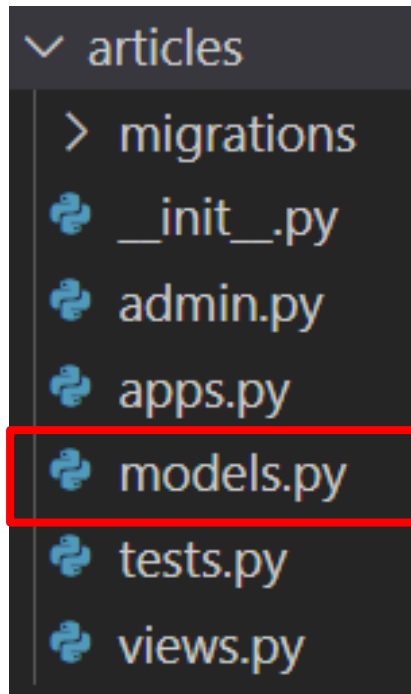
- Model / Template/ View 디자인으로 구조화  
=> (데이터, UI, 비즈니스 로직으로 분리해서 작업)
- MVC design pattern 과 완전히 동일하나 명칭만 다르게 사용

# Django Application 구조

---

## 1. models.py

- DB 와 관련된 Model 을 정의
- MTV 패턴에서 M 을 담당

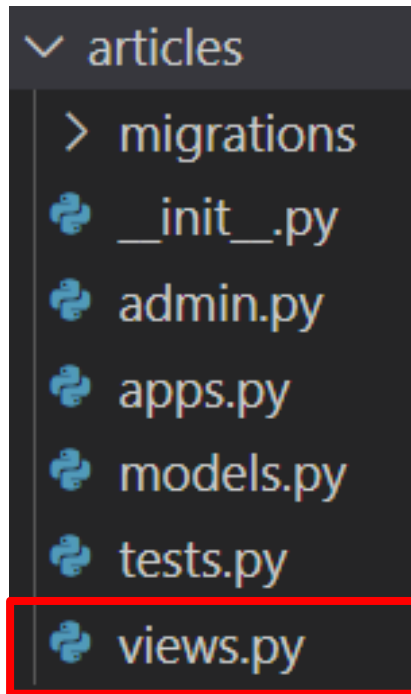


# Django Application 구조

---

## 2. views.py

- url 요청에 따라 해당 요청에 대한 응답을 반환 ( 비즈니스 로직 )
- MTV 패턴에서 V 을 담당, 가장 중요!





# Django Application 구조

---

## 3. admin.py

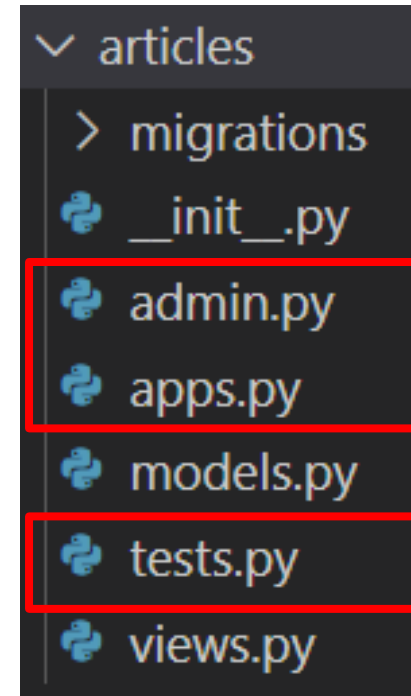
- 관리자 사이트를 설정하는 데 사용

## 4. apps.py

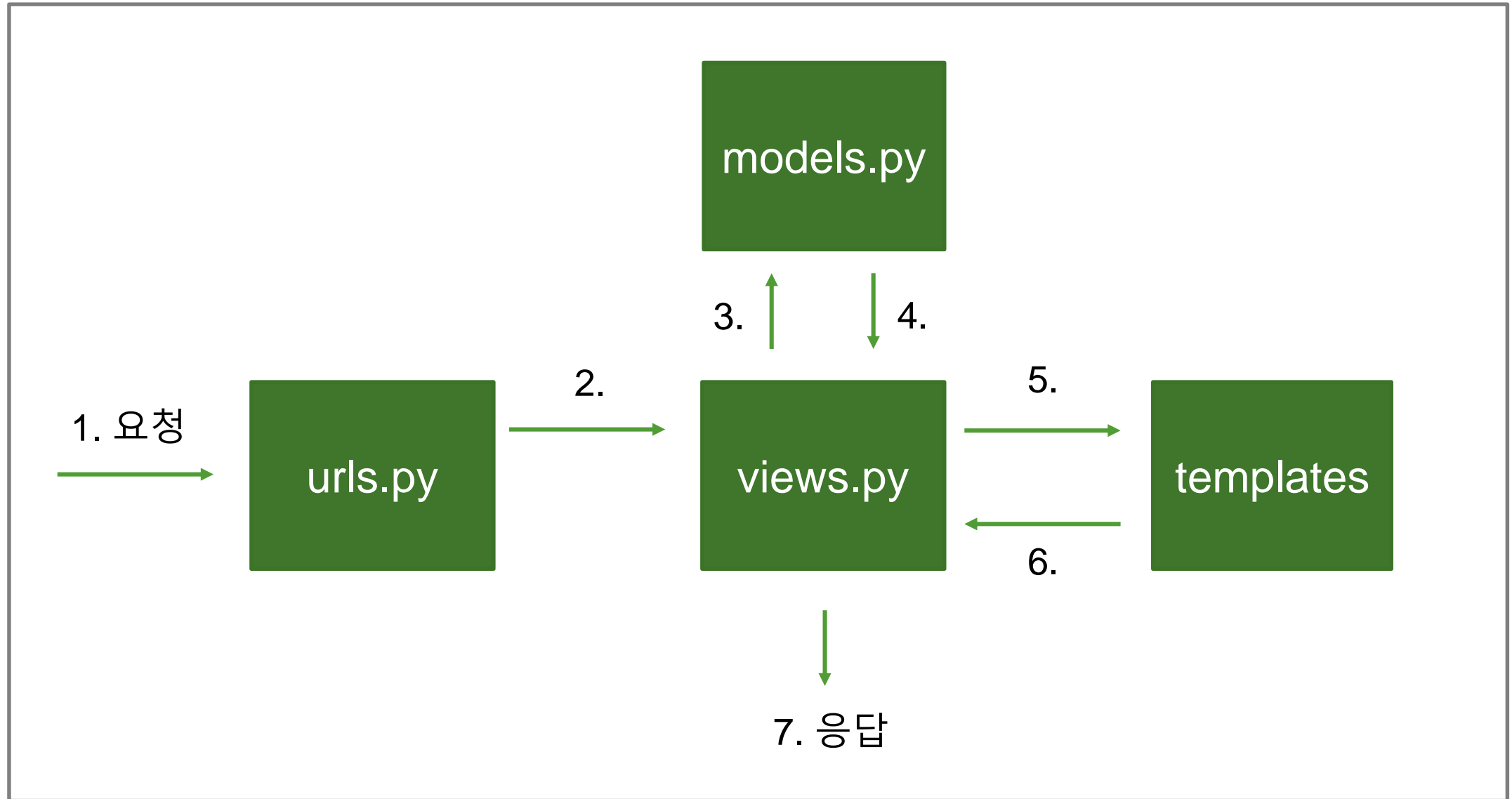
- 앱의 정보를 작성하는 곳

## 5. tests.py

- 테스트 코드를 작성하는 곳



# Django 요청과 응답 과정



# URLs

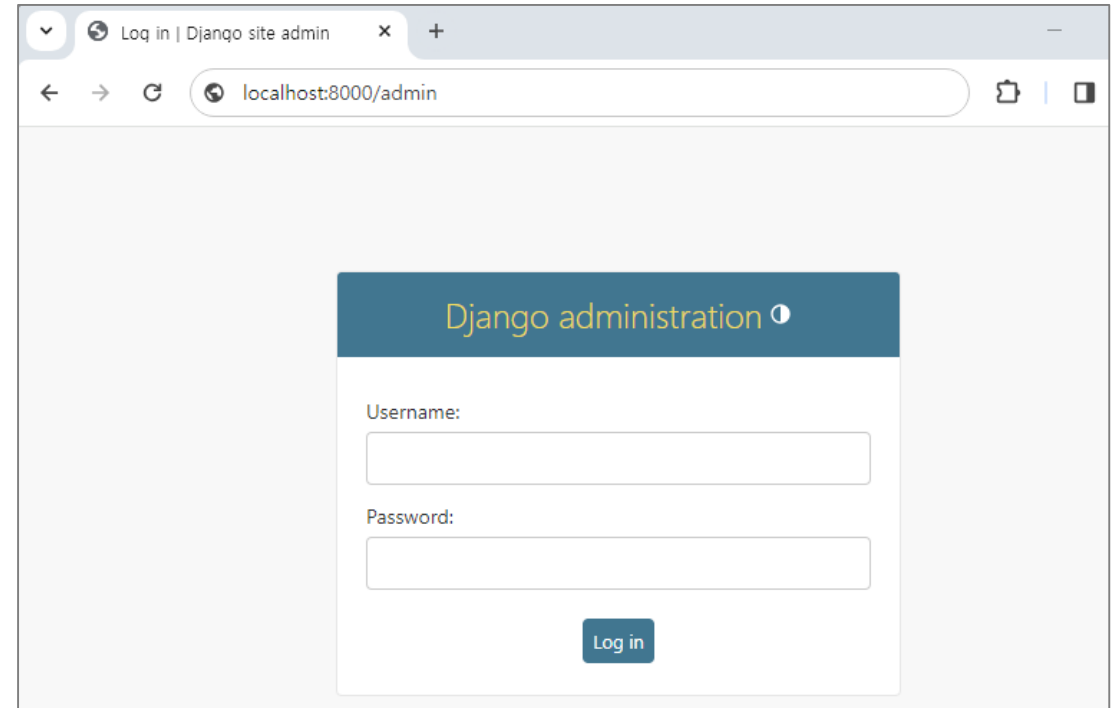
들어오는 URL을 포착해서 Views 모듈의 지정한 함수 호출

=> 서버 도메인 뒤에 작성된 경로로 들어올 경우, 2번째 파라미터 함수를 호출

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

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

Default URL pattern



localhost:8000/admin으로 접속 결과

# URLs

---

localhost:8000/articles/ 로 요청이 왔을 때 views 모듈의 함수 호출

=> 아직 views에 index 함수를 생성하지 않았지만, 미리 패턴에 추가하고 생성

```
# urls.py

from django.contrib import admin
from django.urls import path
from articles import views

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

articles URL pattern 추가

# Views

urls.py 에서 호출하기로 한 views 모듈에 index 함수 생성

```
# urls.py

from django.contrib import admin
from django.urls import path
from articles import views

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

articles URL pattern 추가

```
# views.py
from django.shortcuts import render

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

views.py 에 index 함수 생성

# View

---

urls에 의해서 요청을 받으면 특정 templat에 연결하고, request 객체의 데이터를 활용해서 결합한 후 사용자에게 반환

```
# views.py
from django.shortcuts import render

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

# View - render

---

- 요청(request)의 데이터와 결합하여 응답을 반환하는 객체
- render(request, template\_path, context)

## 1. request

- url 과 함께 날아오는 요청

## 2. template\_path

- 요청에 따라 반환해줄 template 경로

## 3. context

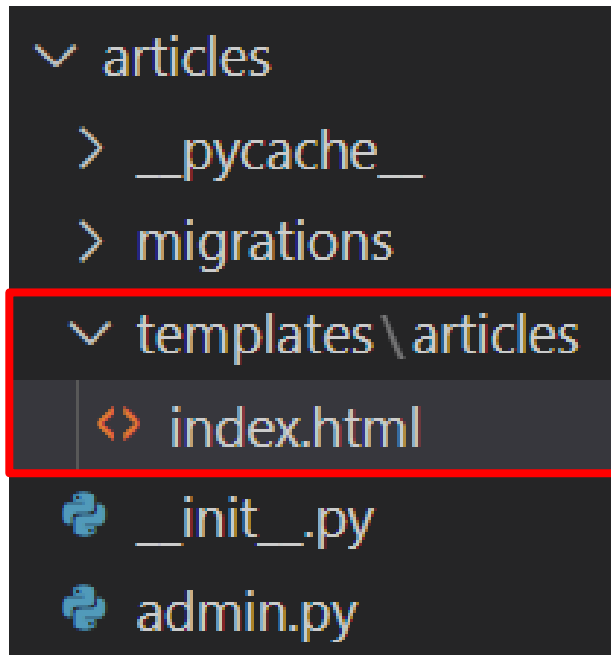
- template 에서 사용할 데이터 (dictionary 타입으로 전달)

# Template

---

views 에서 불러오는 template을 생성

(templates 폴더는 직접 만들어야 하며, 정확한 철자로 생성 )



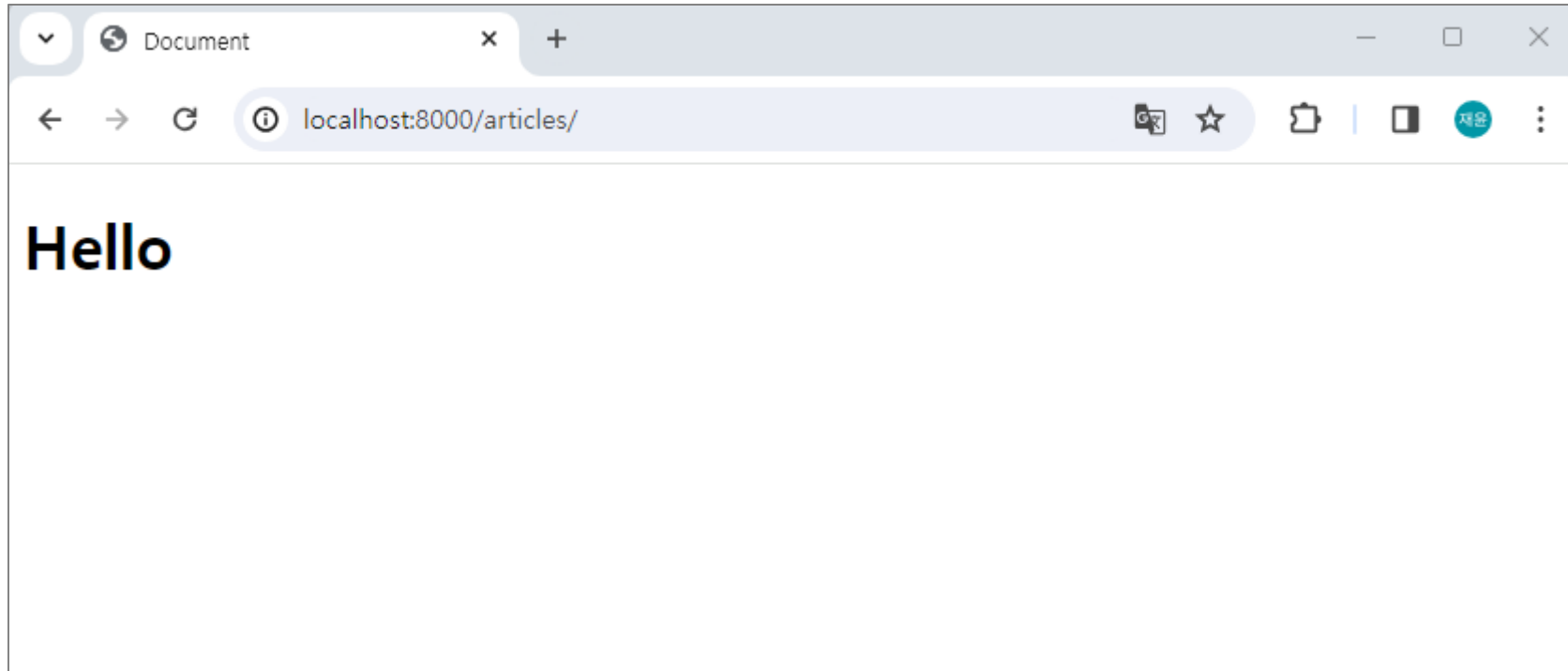
```
<!-- articles/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1>Hello</h1>
</body>
</html>
```



# 생성한 url로 접속하기

---

localhost:8000/articles url로 접속



# 데이터 흐름에 따른 코드 작성 방법

1.

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

2.

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

3.

```
<!-- articles/index.html -->
<!DOCTYPE html>
<html lang="en">
|   ...
</html>
```

# DTL (Django Template Language)

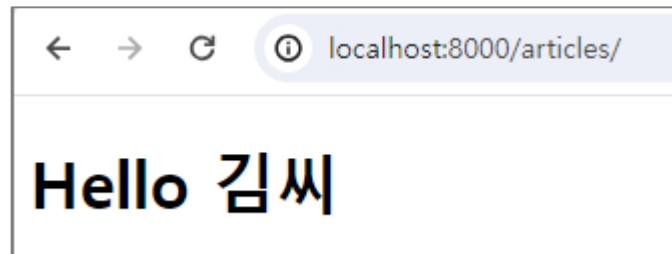
# DTL ( Django Template Language )

HTML 문서 내에 Python 코드를 삽입하여 동적인 웹 페이지 생성

=> HTML은 프로그래밍 언어가 아니기 때문에 DTL이 대신 수행

```
<!-- articles/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1>Hello {{ name }} </h1>
</body>
</html>
```

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



# DTL 문법

---

## 1. Variable (변수)

- render 함수의 3번째 인자로 dictionary 타입으로 전달
- dictionary 의 key 값을 template 에서 사용하면 됨
- {{ }} 안에 key 값을 집어넣어서 호출

```
context = {  
    'variable': '김씨'  
}  
return render(request, 'articles/index.html', context)
```

```
<h1>Hello {{ variable }} </h1>
```

# DTL 문법

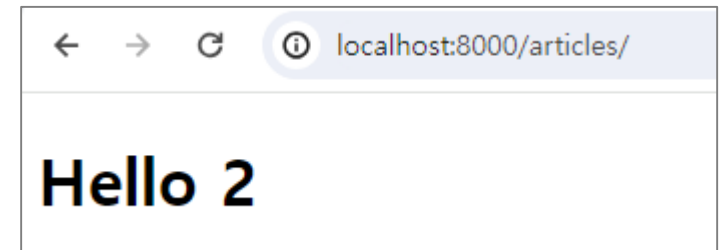
---

## 2. Filters

- 표시할 변수를 수정할 때 사용
- chaining 가능
- [Django 공식 홈페이지](#)에서 학습 및 참고

```
context = {  
    'variable': '김씨'  
}  
return render(request, 'articles/index.html', context)
```

```
<h1>Hello {{ variable|length }} </h1>
```



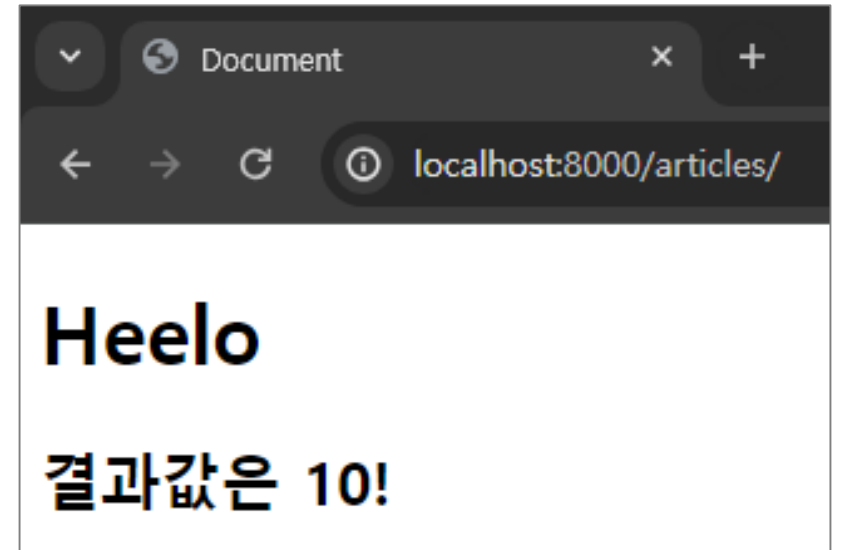
# DTL 문법

## 3. Tags

- 반복문 / 조건문 등을 수행하여 제어 흐름을 수행
- 일부 태그는 시작과 종료 태그가 짝지어서 필요
- [Django 공식 홈페이지](#)에서 학습 및 참고

```
context = {  
    'variable': '김씨',  
    "result": 10  
}  
  
return render(request, 'articles/index.html', context)
```

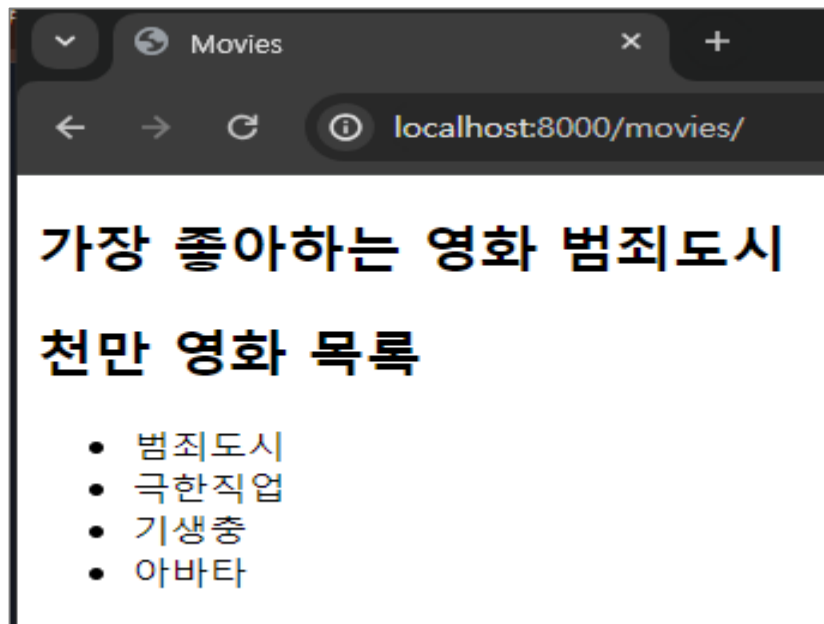
```
<h1>Heelo  {{ name }} </h1>  
{% if result == 10 %}  
    <h2> 결과값은 10! </h2>  
{% endif %}
```



# DTL 문법 예시

```
<!-- templates/articles/movies.html-->
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Movies</title>
</head>
<body>
  <h2> 가장 좋아하는 영화 {{ favorite_movie }}</h2>
  <h2> 천만 영화 목록</h2>
  {% if movie_list %}
    <ul>
      {% for movie in movie_list %}
        <li> {{ movie }} </li>
      {% endfor %}
    </ul>
  {% endif %}
</body>
</html>
```

```
# urls.py
urlpatterns = [
    path('admin/', admin.site.urls),
    path('articles/', views.index),
    path('movies/', views.movies)
]
```





# 템플릿 상속

# 템플릿 상속

Html 을 반복해서 생성할 때 공통요소를 계속 작성하는 문제를 해결

=> 기본 'skeleton' 템플릿을 작성하고, 해당 템플릿을 상속하여 해결

```
<!-- templates/articles/movies.html-->
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Movies</title>
</head>
<body>
  <h2> 가장 좋아하는 영화 {{ favorite_movie }}</h2>
  <h2> 천만 영화 목록</h2>
  {% if movie_list %}
    <ul>
      {% for movie in movie_list %}
        <li> {{ movie }} </li>
      {% endfor %}
    </ul>
  {% endif %}
</body>
</html>
```

```
<!-- templates/articles/index.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1>Heelo {{ name }} </h1>
  {% if result == 10 %}
    <h2> 결과값은 10! </h2>
  {% endif %}
</body>
</html>
```

# 템플릿 상속

## 1. Skeleton 역할을 할 부모 템플릿 작성

- block – endblock 사이에 자식 템플릿이 내용을 작성할 수 있음
- 여러 block 을 지정할 수 있으며, 각 block은 이름으로 구별됨

```
<!-- articles/base.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  {% comment %} bootstrap cdn 생략 {% endcomment %}
</head>
<body>
  {% block content %}
  {% endblock content %}
  {% comment %} bootstrap cdn 생략 {% endcomment %}
</body>
</html>
```

# 템플릿 상속

## 1. 자식 템플릿이 부모 템플릿을 상속받고, 콘텐츠를 작성

- extends 로 부모 템플릿을 상속 ( 2개 이상 상속 불가능 )
- 부모 템플릿의 사용할 block의 이름을 사용해서 작성

```
<!-- templates/articles/index.html-->
{% extends "articles/base.html" %}

{% block content %}
  <h1>Heelo  {{ name }} </h1>
  {% if result == 10 %}
    <h2> 결과값은 10! </h2>
  {% endif %}
{% endblock content %}
```

```
<!-- templates/articles/movies.html-->
{% extends "articles/base.html" %}

{% block content %}
  <h2> 가장 좋아하는 영화 {{ favorite_movie }}</h2>
  <h2> 천만 영화 목록</h2>
  {% if movie_list %}
    <ul>
      {% for movie in movie_list %}
        <li> {{ movie }} </li>
      {% endfor %}
    </ul>
  {% endif %}
{% endblock content %}
```

# Django 실습

---

Practice\_1.ipynb 실습문제 진행

# HTML form

# HTML form

웹 페이지에서 사용자 입력을 수집 및 서버로 전송하는 데 사용



PC방 등 공용PC라면 QR코드 로그인이 더 안전해요. X

☒ ID 로그인

[1] 일회용 번호

QR코드

☒ 로그인 상태 유지

IP보안 ☒

로그인

```
<html lang="ko">
  <head> ... </head>
  <body>
    <div id="wrap" class="wrap">
      <div class="u_skip"> ... </div>
      <header class="header" role="banner"> ... </header>
      <div id="container" class="container">
        <!-- content -->
        <div class="content">
          <div class="login_wrap">
            <ul class="menu_wrap" role="tablist"> ... </ul>
            <form id="frmNIDLogin" name="frmNIDLogin" tar
            </div>
          <div class="find_wrap" id="find_wrap"> ... </div>
          <!-- 배너 -->
          <div id="gladbanner" class="banner_wrap"> ... </div>
          <!-- //content -->
        </div>
        <!-- footer -->
      <div class="footer"> ... </div>
    </div>
    <input type="hidden" id="nclicks_nsc" name="nclicks_n
    <input type="hidden" id="nid_buk" name="nid_buk" valu
```

# HTML form 형식

---

## 1. <input> 태그

- 사용자의 데이터를 입력 받을 수 있는 요소

```
<form action="" method="">  
  <input type="text" name="" id="">  
  <input type="submit" value="submit">  
</form>
```



# HTML form 형식

---

## 2. <input> 태그 – type

- 속성 값을 의미하며 다양한 유형의 입력 데이터를 받음
- ex) text, submit, textarea, ...
- mdn input tag [공식홈페이지](#) 참고

```
<form action="" method="">  
  <input type="text" name="" id="">  
  <input type="submit" value="submit">  
</form>
```

# HTML form 형식

---

## 3. <input> 태그 – name

- 해당 태그의 이름을 의미
- 데이터를 서버에 전송할 때, 서버는 name 속성값으로 데이터에 접근
- 태그의 역할에 맞는 적절한 이름을 부여

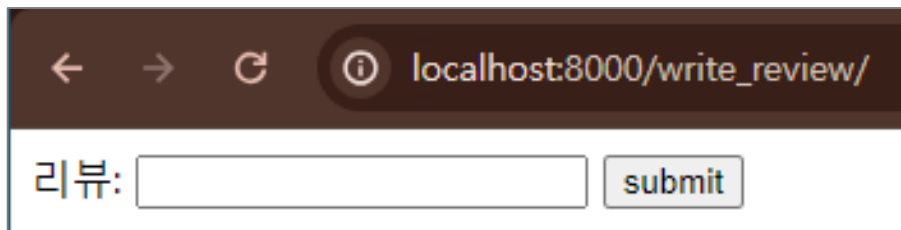
```
<form action="" method="">  
  <input type="text" name="" id="">  
  <input type="submit" value="submit">  
</form>
```

# HTML form 형식

## 4. <input> 태그 – id

- 해당 태그를 고유하게 식별할 수 있는 값
- id를 이용해서 다양한 태그들과 연결 및 조작이 가능 (대표적으로 label)

```
<form action="" method="">  
  <label for="review_text_id">리뷰: </label>  
  <input type="text" name="review_text" id="review_text_id">  
  <input type="submit" value="submit">  
</form>
```



← → ↻ ⓘ localhost:8000/write\_review/

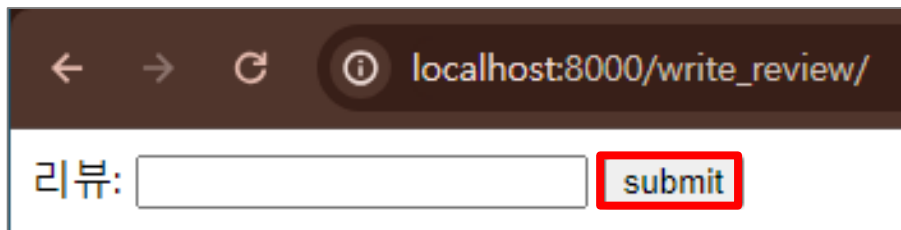
리뷰:

# HTML form 형식

## 5. <input> 태그 – value

- 입력 필드의 초기 값을 정의하는 데 사용
- JS 를 활용하면 동적으로 값을 변경하는 용도로 사용

```
<form action="" method="">
  <label for="review_text_id">리뷰: </label>
  <input type="text" name="review_text" id="review_text_id">
  <input type="submit" value="submit">
</form>
```



리뷰:

# HTML form 형식

## 5. action

- form 태그의 데이터가 제출될 URL을 작성하는 곳
- Django 의 경우 view 함수의 url 경로를 입력함

```
<form action="/save_review/" method="">
  <label for="review_text_id">리뷰: </label>
  <input type="text" name="review_text" id="review_text_id">
  <input type="submit" value="submit">
</form>
```

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

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

# HTML form 형식

---

## 6. method

- 데이터를 어떤 방식으로 요청할 지 설정
- get, post 으로 구분
- default 는 get 으로 설정
- 일반적으로 GET 은 조회, POST 는 변경(생성,수정,삭제) 일 때 사용

```
<form action="/save_review/" method="GET">  
  <label for="review_text_id">리뷰: </label>  
  <input type="text" name="review_text" id="review_text_id">  
  <input type="submit" value="submit">  
</form>
```

# HTML form 형식

---

## 6. method – get

- form 데이터가 url 에 직접 추가(쿼리 매개변수)되고, 브라우저 주소에 표시
- 서버에 데이터를 검색하며, 보안이 중요하지 않은 경우에 사용됨
- http://host?**key=value&key2=value2** 방식으로 전송
- 예시) search=“안녕”, date=“1999-12-31” 이라는 form 데이터 전송 시  
=> http://host?search=“안녕”&date=“1999-12-31” 로 서버에 전송

## HTML form 데이터 서버로 전송 후 request 출력

리뷰:



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



save reivew.html 을 생성하지 않아서 에러 발생



# HTML request data

서버에서 받은 request 데이터 출력 및 에러로그를 통해 상세히 보기

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

```
<WSGIRequest: GET '/save_review/?review_text=%EB%A6%AC%EB%B7%B0%EC%9E%85%EB%8B%88%EB%8B%A4.'>
```

Request information

USERAnonymousUser

GET

Variable	Value
review_text	'리뷰입니다.'

POSTNo POST data

FILESNo FILES data

COOKIESNo cookie data

# HTML request data

---

request 객체에 method를 활용해서 request data 출력

```
# articles/save_review.py
def save_review(request):
    print(request.GET)
    return render(request, 'articles/save_review.html')
```

```
<QueryDict: {'review_text': ['리뷰입니다.']}>
```

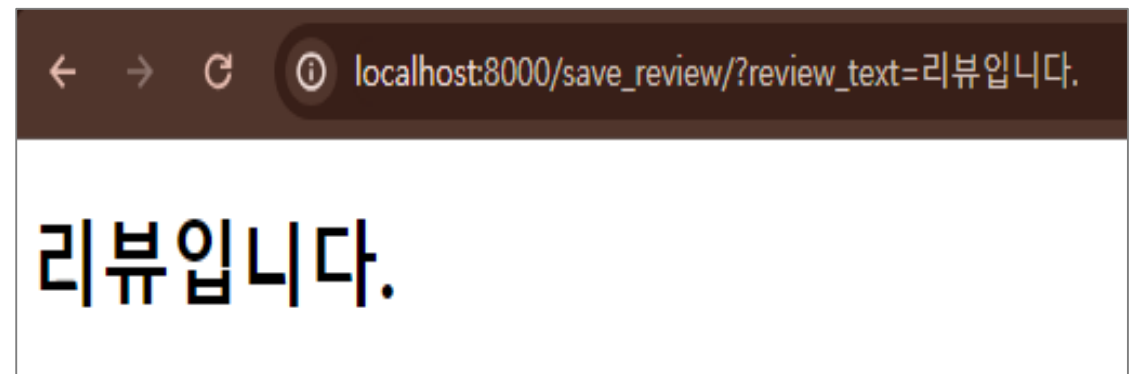
# HTML request data

save\_review.html 생성 후 request data를 context로 전달 후 출력

```
# articles/save_review.py
def save_review(request):
    req_data = request.GET
    context = {
        'review_text': req_data.get('review_text')
    }
    return render(request, 'articles/save_review.html', context)
```

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

{% block content %}
    <h1> {{ review_text }}</h1>
{% endblock content %}
```



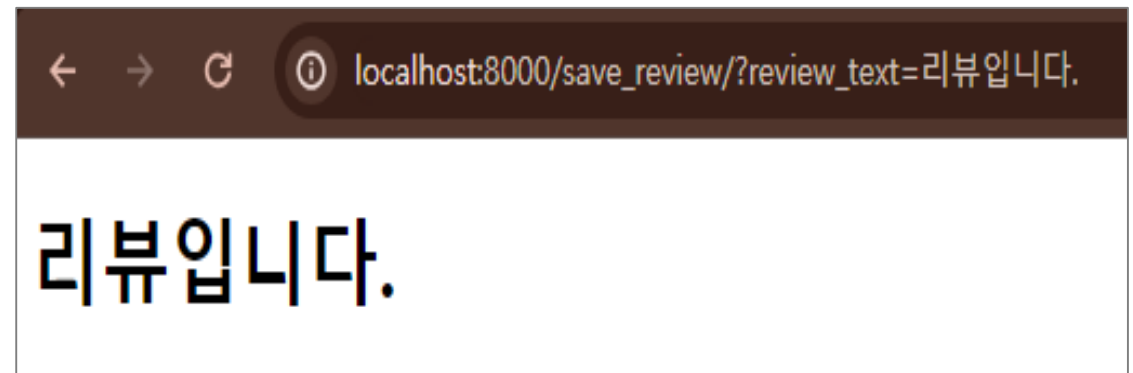
# HTML request data

save\_review.html 생성 후 request data를 context로 전달 후 출력

```
# articles/save_review.py
def save_review(request):
    req_data = request.GET
    context = {
        'review_text': req_data.get('review_text')
    }
    return render(request, 'articles/save_review.html', context)
```

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

{% block content %}
    <h1> {{ review_text }}</h1>
{% endblock content %}
```



A dark green background with a lighter green circle on the left side.

URLs

# URL 관리의 문제점

---

단순히 특정 index만 변경되는 URL을 매번 작성하는 건 불가능

- 영화 목록이 추가될 때마다 urlpatterns에 추가??

```
# urls.py
urlpatterns = [
    path('movie/1/'),
    path('movie/2/'),
    path('movie/3/'),
    path('movie/4/'),
    path('movie/5/'),
    path('movie/6/'),
    ...
]
```

# Variable Routing

---

URL 일부에 변수를 포함시키는 방법

=> view 함수에서 인자로 전달

```
# urls.py
urlpatterns = [
    path('movie/1/'),
    path('movie/2/'),
    path('movie/3/'),
    path('movie/4/'),
    path('movie/5/'),
    path('movie/6/'),
    ...
]
```

# Variable Routing 형식

---

변수에 해당하는 부분을 <타입:변수명> 으로 지정

- 타입(path converters)의 종류는 [Django 공식홈페이지](#) 참고

```
# urls.py
urlpatterns = [
    path('movie/<int:num>/' ),
    path('show/<str:screen>' ),
]
```



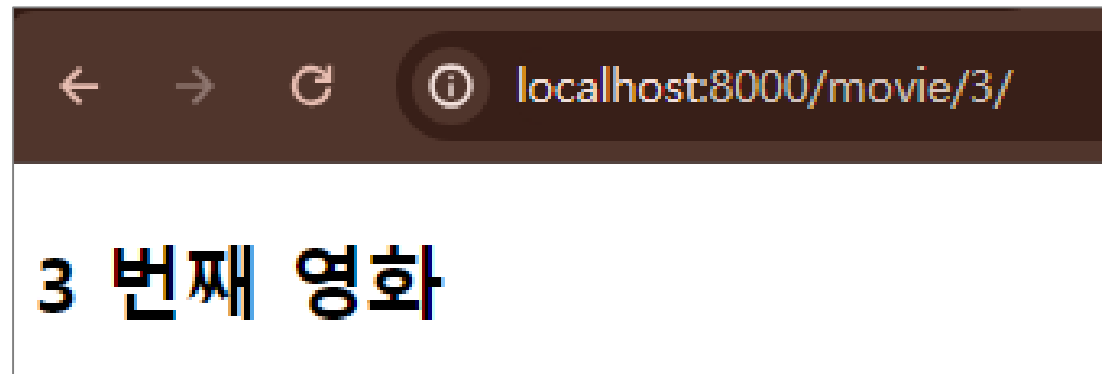
# Variable Routing 예제

```
# urls.py
urlpatterns = [
    path('movie/<int:num>/'),
    ...
]
```

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

```
<!-- templates/articles/movie.html -->
{% extends "articles/base.html" %}

{% block content %}
    <h2> {{ num }} 번째 영화</h2>
{% endblock content %}
```



# Project urls.py 문제점

추가 생성한 App과 기존 App 의 view 함수명/주소가 같은 경우

```
✓ movies
  > migrations
  ✓ templates\movies
    <> index.html
  🌀 __init__.py
  🌀 admin.py
  🌀 apps.py
  🌀 models.py
  🌀 tests.py
  🌀 views.py
```

추가 생성한 movies app

```
from articles import views
from movies import views

# urls.py
urlpatterns = [
    |   path('index', views.index),
    ]
```

어떤 views를 사용해야할까?

# Project urls.py 문제점

---

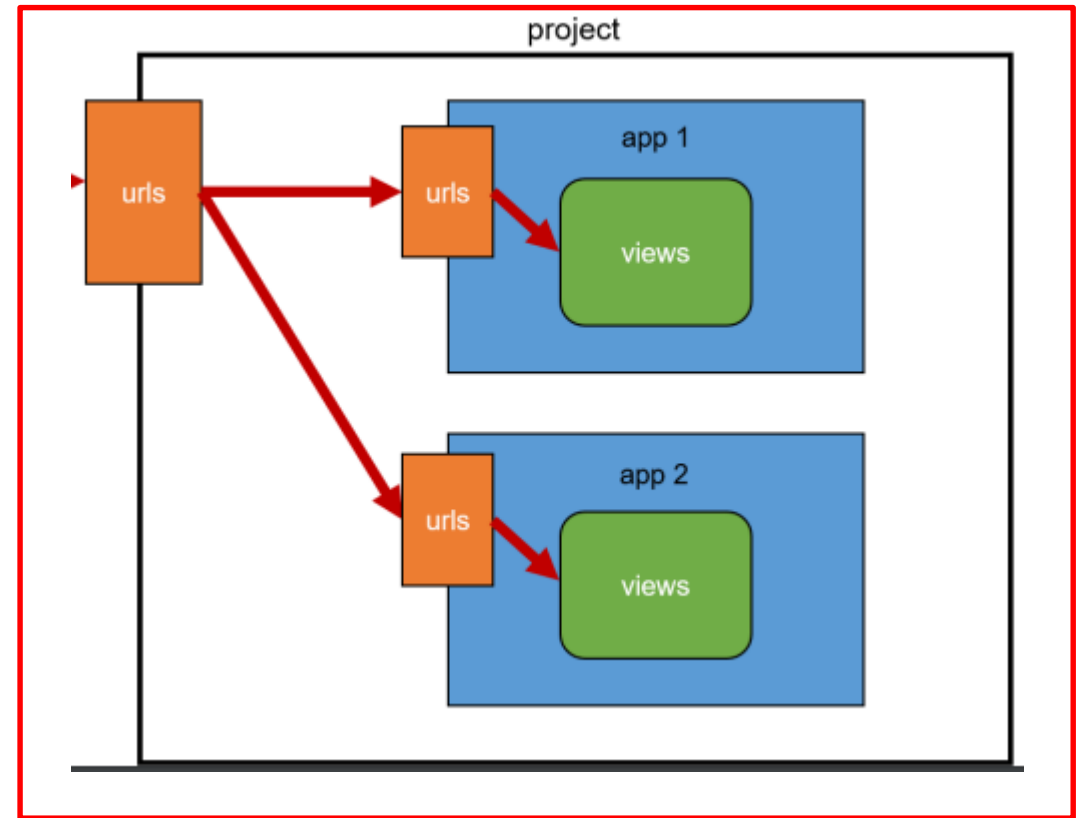
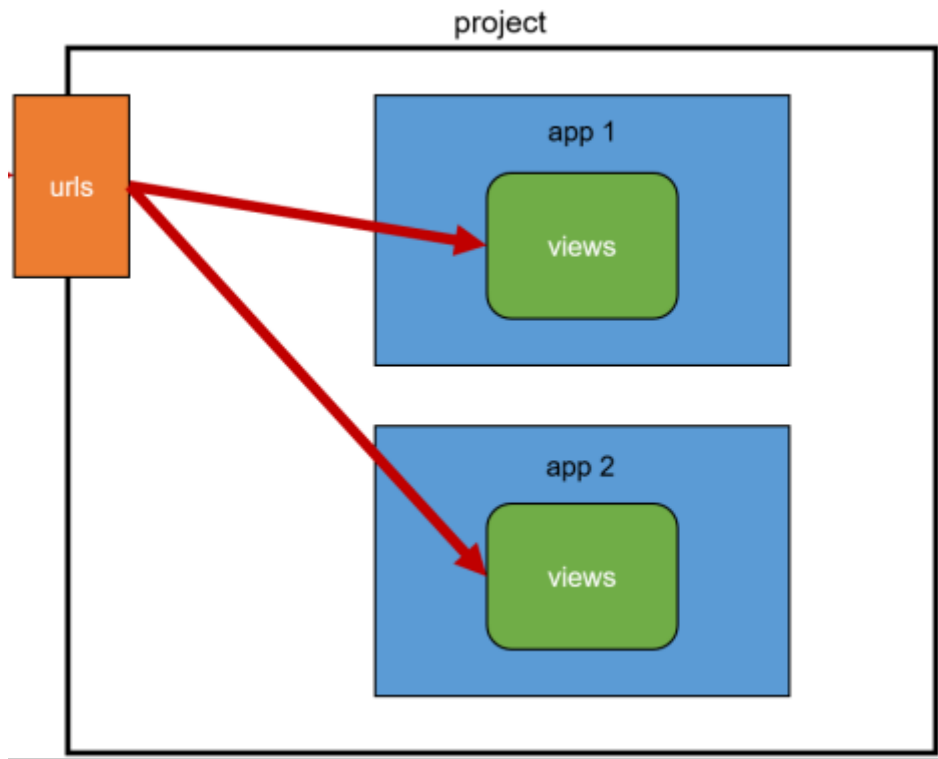
아래 코드처럼 해결이 가능하지만 더 좋은 방법을 사용하자!

```
from django.contrib import admin
from django.urls import path
from articles import views as articles_views
from movies import views as movies_views

# urls.py
urlpatterns = [
    path('articles_index', articles_views.index),
    path('movies_index', movies_views.index),
]
```

# App urls.py

프로젝트에서 관리했던 urls 을 각 App에서 관리



# App urls.py

프로젝트에서 관리했던 urls 을 각 App에서 관리

```
from django.contrib import admin
from django.urls import path, include
from articles import views
from movies import views as movie_views

# urls.py
urlpatterns = [
    # path('movie/<int:num>/', views.movie),
    # path('save_review/', views.save_review),
    # path('admin/', admin.site.urls),
    # path('articles/', views.index),
    # path('movies/', views.movies),
    # path('write_review/', views.write_review),
    # path('index/', movie_views.index),

    path('admin/', admin.site.urls),
    path('articles', include('articles.urls')),
    path('movies', include('movies.urls'))
]
```

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

urlpatterns = [
    path('movie/<int:num>/', views.movie),
    path('save_review/', views.save_review),
    path('', views.index),
    path('movies/', views.movies),
    path('write_review/', views.write_review),
]
```

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

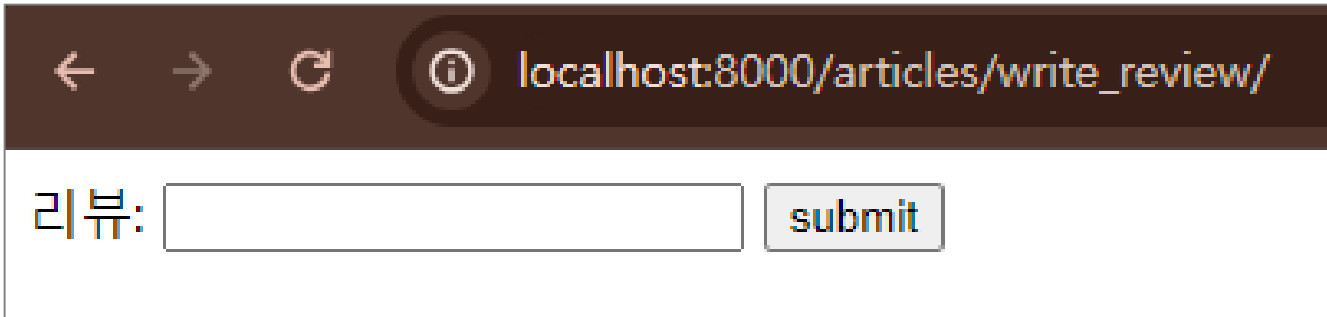
urlpatterns = [
    path('index/', views.index)
]
```

# App urls.py

---

기존: localhost:8000/write\_review 로 접근

변경 후 : localhost:8000/articles/write\_review로 접근해야 함



The screenshot shows a web browser window. The address bar displays the URL `localhost:8000/articles/write_review/`. Below the address bar, there is a form with the label "리뷰:" followed by a text input field and a "submit" button.

# App urls.py 문제점 - 1

경로 변경에 따라 해당 경로에 접근하는 모든 코드를 수정해야 함

=> 바뀌야할 경로가 무수히 많다면..?

```
<form action="/save_review/" method="GET">
  <label for="review_text_id">리뷰: </label>
  <input type="text" name="review_text" id="review_text_id">
  <input type="submit" value="submit">
</form>
```



```
<form action="articles/save_review/" method="GET">
  <label for="review_text_id">리뷰: </label>
  <input type="text" name="review_text" id="review_text_id">
  <input type="submit" value="submit">
</form>
```

# App urls.py 해결방안

---

## Naming URL Patterns

- URL 에 이름을 지정해서 사용

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

urlpatterns = [
    path('movie/<int:num>/', views.movie, name='movie'),
    path('save_review/', views.save_review, name='save_review'),
    path('', views.index, name='index'),
    path('movies/', views.movies, name='movies'),
    path('write_review/', views.write_review, name='write_review'),
]
```



# App urls.py 해결방안

## Naming URL Patterns

- {% url '{url\_name}' path\_key =value, ...%}
- ex) {% url 'movie' num=1 %}

```
<form action="articles/save_review/" method="GET">  
  <label for="review_text_id">리뷰: </label>  
  <input type="text" name="review_text" id="review_text_id">  
  <input type="submit" value="submit">  
</form>
```



```
<form action="{% url 'save_review' %}" method="GET">  
  <label for="review_text_id">리뷰: </label>  
  <input type="text" name="review_text" id="review_text_id">  
  <input type="submit" value="submit">  
</form>
```

# App urls.py 문제점 - 2

---

movies app의 name 과 articles app의 name이 동일해서 참조 불가

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

```
# articles/urls.py
urlpatterns = [
    path('movie/<int:num>', views.movie, name='movie'),
    path('save_review/', views.save_review, name='save_review'),
    path('', views.index, name='index'),
    path('movies/', views.movies, name='movies'),
    path('write_review/', views.write_review, name='write_review'),
]
```

# App urls.py 해결방안

app\_name 속성을 지정하고, url 태그 변경하기

```
# movies/urls.py
app_name = 'movies'
urlpatterns = [
    path('index/', views.index, name='index')
]
```

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

```
<form action="{% url 'articles:save_review' %}" method="GET">
  <label for="review_text_id">리뷰: </label>
  <input type="text" name="review_text" id="review_text_id">
  <input type="submit" value="submit">
</form>
```

리뷰:

```
<form action="/articles/save_review/" method="GET">
  <label for="review_text_id">리뷰: </label>
  <input type="text" name="review_text" id="review_text_id">
  <input type="submit" value="submit">
</form>
</body>
```

# Django 실습

---

Practice\_2.ipynb 실습문제 진행