One_to_Four (Day1 ~ Day4 까지 내용 복습 겸 정리)

자료	Django
구분	Django
과목	

TodoList 를 만들어 보자. (할일 목록을 만드는 프로젝트를 만들어 보자)

- 바탕화면에서 one_to_four 파일 생성 후 vscode 로 열기

- 가상환경 만들기 > activate 하기 > django 설치 후 > 프로젝트 만들기 > 앱 2개 만들기

```
$ python -m venv venv_one_to_four

$ source venv_one_to_four/Scripts/activate

$ pip install django

$ django-admin startproject todo_list_pjt .

$ python manage.py startapp accounts

$ python manage.py startapp todos
```

```
# settings.py

---

INSTALLED_APPS = [
    'accounts',
    'todos',

---
```

```
# todo_list_pjt > urls.py

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

urlpatterns = [
```

```
    path('admin/', admin.site.urls),
    path('todos/', include('todos.urls')),
    path('accounts/', include('accounts.urls')),
]
```

accounts > urls.py 파일 생성

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

app_name = 'accounts'

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

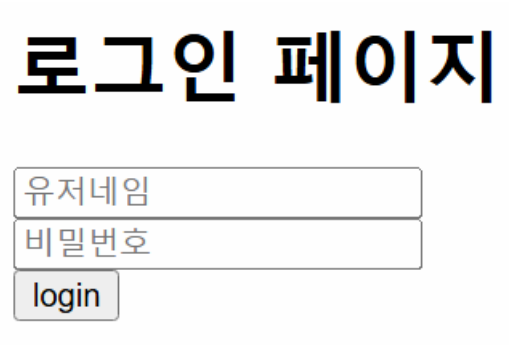
accounts > views.py

```
from django.shortcuts import render

# Create your views here.


def login(request):
    return render(request, 'accounts/login.html')
```

accounts 앱에서 templates>accounts>login.html 파일 생성 후

```
<h1>로그인 페이지</h1>
<form action="#" method="GET">
  <label for="ssafy"></label>
  <input type="text" placeholder="유저네임" /> <br />
  <input type="password" placeholder="비밀번호" id="ssafy" name="query" /> <br />
  <input type="submit" value="login" />
</form>
```

- 서버 켜서 중간 확인
  - urls.py 가서 todos/ 의 경로는 주석 처리를 하고 runserver 를 하자.
  - http://127.0.0.1:8000/accounts/login/ 화면 확인 후



- 이제 todos 앱 셋팅 하자.
  - todos/ 주석 처리 한 것을 풀자.

todos > urls.py 파일 생성

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

app_name = 'todos'

urlpatterns = [
    path('', views.index, name='index'),
    path('create_todo/', views.create_todo, name='create_todo'),
    path('<work>/', views.detail, name='detail'),
]
```

각 경로에 해당하는 함수의 역할은 다음과 같다.

index → 생성된 할일 리스트를 한눈에 보는 페이지가 될 것이다. (READ)

create_todo → 할일 목록을 새로 생성을 하는 페이지가 될 것이다. (CREATE)

detail - 특정 할일의 세부 내용을 확인하는 페이지가 될 것이다. (READ)

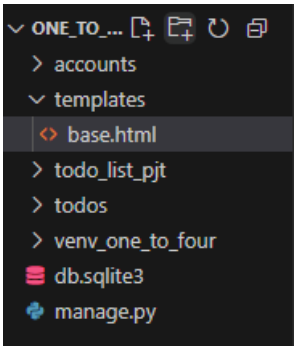
todos > views.py

```
from django.shortcuts import render

# Create your views here.


def index(request):
    pass


def create_todo(request):
```

```
    pass


def detail(request):
    pass
```

base.html 문서를 만들고 상속을 하자. 모든 html 문서에서 상속을 받게 할 것이다.

해당 프로젝트 파일 최상단 경로에 templates > base.html 만들자.



templates > base.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Don't think, Just do it</title>
  </head>
  <body>
    <a href="{% url 'todos:index' %}">[Main]</a>
    <a href="{% url 'todos:create_todo' %}">[Create]</a>
    <a href="{% url 'accounts:login' %}">[Login]</a>

    {% block content %}

    {% endblock %}
  </body>
</html>
```

그 다음

settings.py 파일의 'DIRS': [] 에

전역으로 적용시킬 템플릿이 들어가 있는 파일 경로를 추가하자.

```
TEMPLATES = [
    {
        'DIRS': [BASE_DIR/'templates'],
```

아까 만든 accounts 앱에 있는 login.html 파일에서 base.html 내용을 상속 받자.

accounts > login.html

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

{% block content %}
  <h1>로그인 페이지</h1>
  <form action="#" method="GET">
    <label for="ssafy"></label>
    <input type="text" placeholder="유저네임" /> <br />
    <input type="password" placeholder="비밀번호" id="ssafy" name="query" /> <br />
    <input type="submit" value="login" />
  </form>
{% endblock %}
```

- 서버 켜서 확인해 보자. http://127.0.0.1:8000/accounts/login/

todos 앱 가자.

todos 앱은

1. 할일 목록을 생성하고 (create_todo 함수)
2. 생성된 목록을 클릭시, 해당 목록의 상세페이지 조회를 할 것이다. (detail 함수)
3. index 함수에서 생성된 목록을 전부 조회하는 것은 내일 학습 할 내용이다.

여기서 오늘은 1번과 2번을 중점적으로 지금까지 학습한 내용 가지고 연습해보자.

먼저 할일 목록을 생성하는 create_todo 함수부터 작성 해보자.

todos > views.py

```
from django.shortcuts import render

# Create your views here.


def index(request):
    pass


def create_todo(request):
    return render(request, 'todos/create_todo.html')


def detail(request):
    pass
```

todos > templates > todos > create_todo.html 파일 생성

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

{% block content %}
  <h1>이곳에 할 일을 생성합니다.</h1>

  <form action="{% url 'todos:index' %}">
    <label for="create_todo"></label>
    <input type="text" id="create_todo" name="minho" />
    <input type="submit" value="go" />
  </form>
{% endblock %}
```

할일 목록을 생성하는 create_todo.html 에서 유저가 작성한 내용을

todos앱의 할일 목록을 보여주는 index 함수로 보내자.

todos > views.py

```
from django.shortcuts import render

# Create your views here.
```

```
def index(request):
    work = request.GET.get('minho')
    context = {
        'work': work
    }
    return render(request, 'todos/index.html', context)
```

index 함수에서는 create_todo.html 의 form 을 통해서 입력받은 값을

딕셔너리의 형태로 work 라는 변수에 저장했다.

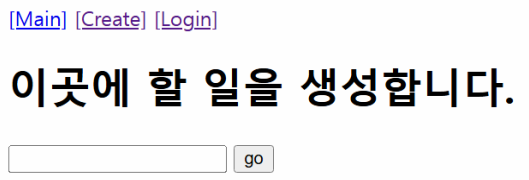
그리고 work의 값을 context로 담아서 index.html 로 보낸다.

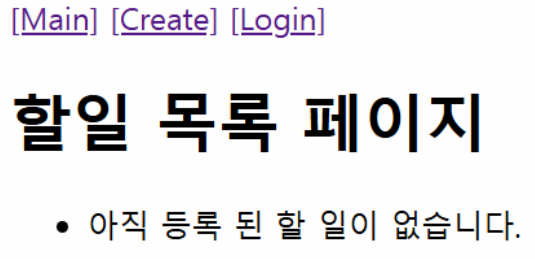
todos > templates > todos > index.html 파일 생성

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

{% block content %}
  <h1>할일 목록 페이지</h1>

  <ul>
    {% if work %}
      <li>{{ work }}</li>
    {% else %}
      <li>아직 등록 된 할 일이 없습니다.</li>
    {% endif %}
  </ul>
{% endblock %}
```

index.html 에서는 전달 받은 값을 화면에 표기했다.

- 서버 켜서 확인해 보자. http://127.0.0.1:8000/todos/create_todo/





입력창에 “test 입니다.” 라고 타이핑 후, go 버튼을 누르면 해당 내용이 바로

index 함수로 넘어가면서 할일 목록 페이지(index.html) 에서 보여질 것이다.

해당 제목을 클릭 시 work (할일)의 상세 페이지로 넘어가게 수정하자.

할일 목록을 클릭시 해당 상세 페이지로 넘어갈 수 있도록

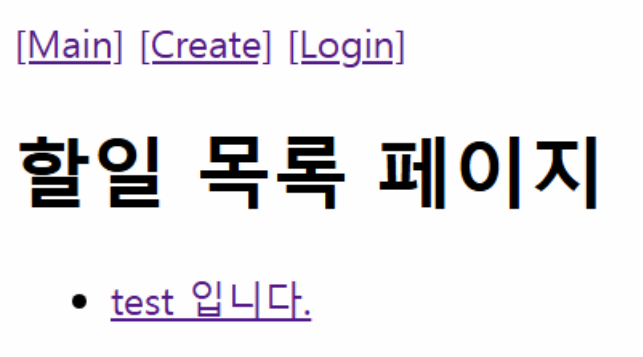
아래와 같이 링크를 달자.

```
{% if work %}
  <li>
    <a href="{% url 'todos:detail' work %}">{{ work }}</a>
  </li>
```

urls.py에서 상세페이지의 URL PATH 를 확인해 보니 variable routing 사용하고 있다.

따라서 url 경로안에 'todos:detai' 옆에 work를 적어 줌으로써 해당 URL에 함께 보낼 변수도

작성해 주었다.



이제 상세페이지 만들자. (detail 함수를 작성해보자)

detail 함수에서는 URL에 들어갈 variable을 매개변수 work 를 통해서 받자.

todos > views.py

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

variable routing 으로 view 함수로 넘어온 인자값을 work라는 매개변수로 받은후,

그리고 work값을 context를 통해서 detail.html 으로 전달했다.

templates > todos > detail.html 파일 생성

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

{% block content %}
  <h1>{{ work }} 의 상세 페이지 입니다.</h1>
{% endblock content %}
```

- 서버켜서 create 부터 detail 페이지 까지 잘 작동 되는지 확인해 보자

자, 그러면 이제 create에서 입력 받은 할일을 database에 저장할 수 있도록 수정해보자.

models.py에 database의 테이블을 정의하자

todos > models.py

```
from django.db import models

# Create your models here.

class Todo(models.Model):
    work = models.CharField(max_length=100)
    content = models.TextField()
    is_completed = models.BooleanField()
```

클래스를 정의 할 때 대문자로 작성하며

models 모듈의 Model 클래스를 상속받아 DB 테이블을 작성했다.

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

admin 계정도 생성하자.

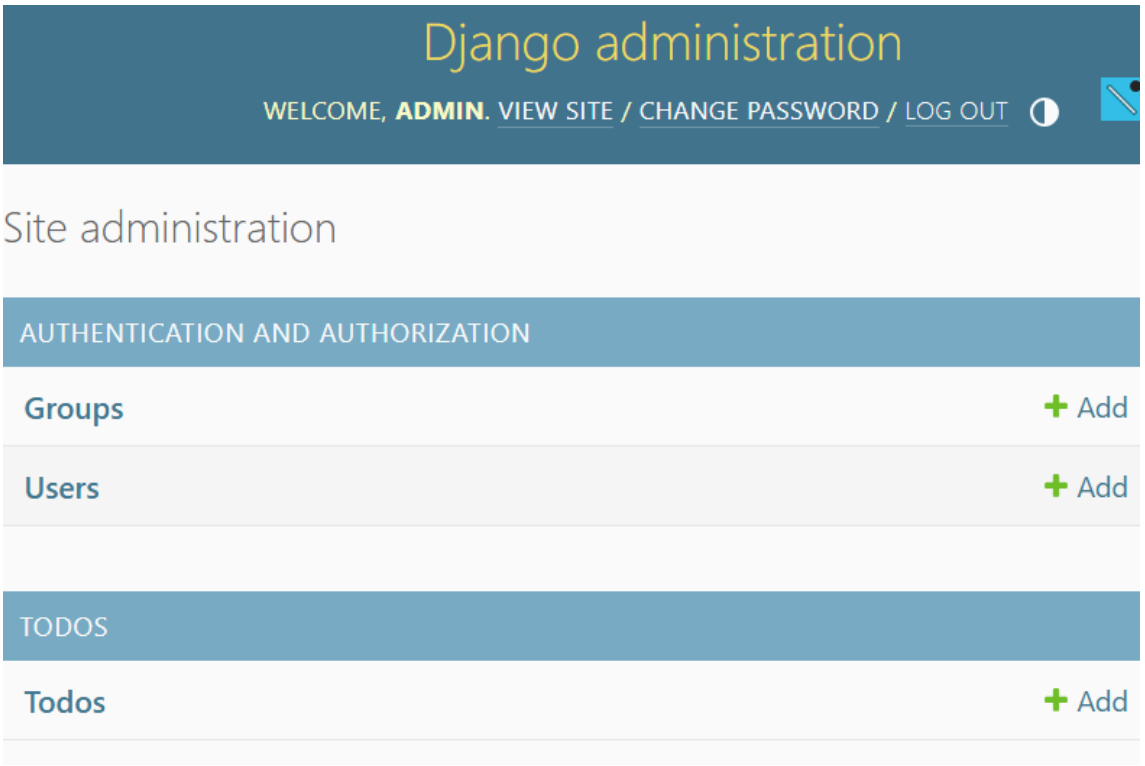
todos > admin.py

```
from django.contrib import admin
from .models import Todo
# Register your models here.
```

```
admin.site.register(Todo)
```

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

- 서버켜서 관리자 페이지 (admin 페이지)에서 로그인이 잘 되는지 확인해 보자.



==========

Django ORM 을 통해서 database에 데이터 CRUD 를 해 볼 것이다.

```
$ pip install ipython
$ pip install django-extensions
$ pip freeze > requirements.txt
```

ipython : 파이썬에서 사용하는 대화형 인터프리터 패키지

django-extensions: 장고 편의기능들이 들어있는 패키지로, shell_plus 사용하기 위해 설치

```
# settings.py
INSTALLED_APPS = [
```

```
    'accounts',
    'todos',
  'django_extensions',   # 언더바 주의
...,
]
```

```
$ python manage.py shell_plus    # Django shell 실행하기
```

자 이제 django shell 에서 python 코드를 써서 database를 조작할 것이다.

(이 과정을 도와주는 것이 ORM 이라고 했다.)

1. 자 그럼 query문으로 database에 data 객체를 생성 (Create) 볼 것이다.
2. 데이터 들을 조회 (Read) 할 것이다. - all filter get 사용
3. 데이터를 수정 (Update)를 할 것이며,
4. 데이터를 삭제 (Delete)도 할 것이다.

```
In [1]: test=Todo(work='할일정보',content='상세정보',is_completed='False')

# 모든 데이터 조회
In [2]: test.save()

In [3]: test=Todo.objects.all()

In [4]: test[0].work

# 단일 데이터 조회
In [5]: test=Todo.objects.get(pk=1)

In [6]: test.work

# 레코드 추가
In [7]: test2=Todo.objects.create(work='두번째할일',content='노래연습',is_completed='False')

# 필드값 수정
In [8]: test.work='첫번째할일'

In [9]: test.save()

In [10]: test.work

# 데이터 삭제
In [11]: test.delete()
```

- db.sqlite3 로 확인 해보자. 그리고 서버를 켜서 admin 페이지에서도 DB테이블을 확인해보자.

이번에는 레코드 작성 시간도 database에 기록이 되도록 테이블을 수정해보자.

todos > models.py

```
created_at = models.DateTimeField(auto_now_add=True, null=True)
updated_at = models.DateTimeField(auto_now=True, null=True)
```

- auto_now_add=True (최초 생성일자)
- auto_now=True (최종 수정일자)
- null=True (NULL값 허용)

Django의 모델에서 DateTimeField 필드는 기본적으로
`NOT NULL` 제약조건으로 설정된다.
이 말은 필드안에 **값이 반드시 존재해야 한다는 의미**다. 그렇지 않으면 에러를 발생한다.

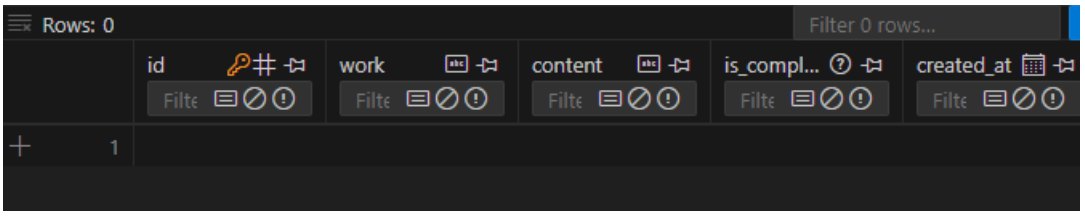
하지만 두 필드(created_at, updated_at) 을 추가하기 전에 이미 DB안에 있던 데이터에는
created_at, updated_at 필드의 값이 없을 것이다. 그렇기 때문에 null=True 옵션을 적어줬다.
이는 새 필드를 추가 전, 기존 데이터에 대해서 `NULL`을 허용하도록 설정한 것이다.

다시말해, null=True 옵션을 넣어 주면서 필드를 추가하기 전 기존에 있던 데이터베이스에 데이터가
NULL 값을 가지며 DB에 문제가 생기지 않도록 만든 것이다.

```
exit() 를 적어서 django_shell을 종료하자.
```

DB를 삭제하고 다시 migrations 그리고 migrate를 실시하자.

```
$ python manage.py makemigrations

$ python manage.py migrate

그리고 필드 추가가 잘 적용이 되어는지 db.sqlite3 로 확인 해보자.
```



오늘은 여기 까지 이다.

index.html 에는 DB안에 있는 데이터 들의 목록이 전부 보이도록 해야 하는데 이것은 내일 학습 후 연습해보자.