점프 투 장고

<1장 장고 개발 준비>

장고는 튼튼한 웹 프레임워크

장고는 보안 공격을 기본으로 잘 막아 준다.(SQL 인젝션, XSS(cross-site scripting), CSRF(cross-site request forgery), 클릭재킹(clickjacking)과 같은 보안 공격을 기본으로 막아 준다.) 여기서, CSRF는 위조된 요청을 보내는 공격 방법이다.

---

1. 디렉터리 만들기
2. python -m venv venv

   python -m venv는 파이썬 모듈 중 venv라는 모듈을 사용한다는 의미

   뒤의 venv는 생성할 가상 환경의 이름
3. source venv/Scripts/activate

   가상환경 진입
4. source venv/Scripts/deactivate

   가상환경 벗어나기
5. pip install django django-extensions ipython
6. django-admin startproject '프로젝트 이름' .

   . 은 '현재 디렉터리를 프로젝트 디렉터리로 만들라'는 의미
7. python manage.py runserver

장고의 프로젝트는 하나의 웹 사이트

프로젝트 안에는 여러 개의 앱이 존재, 이 앱들이 모여 웹 사이트를 구성

<2장 장고 기본 요소 익히기>

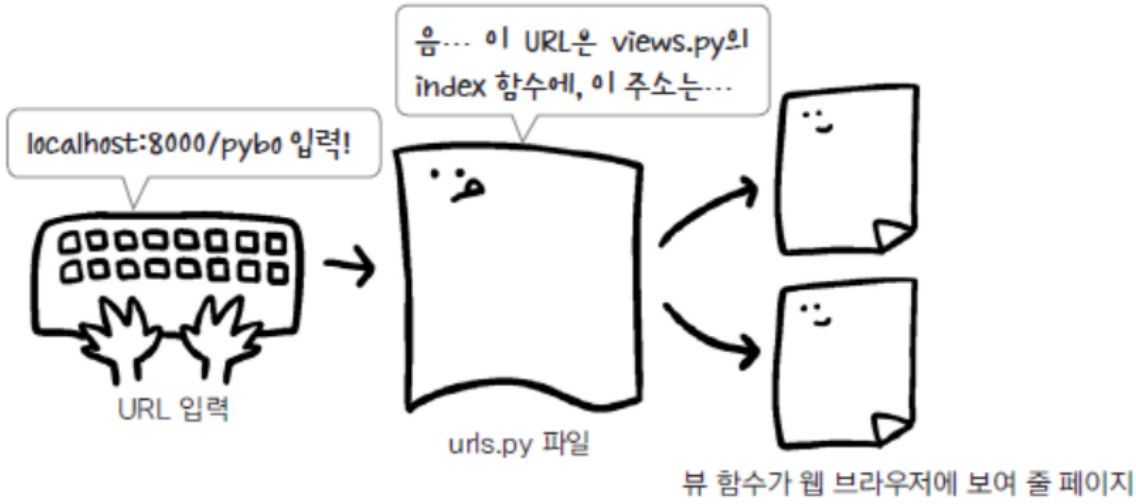
1 - 1) url

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('pybo/', views.index),  # path 함수를 사용하여 pybo/ URL과 views.index를
매핑
]
```

views.index는 views.py 파일의 index 함수를 의미

장고는 이런 식으로 URL과 뷰 함수를 매핑



1 - 2) view

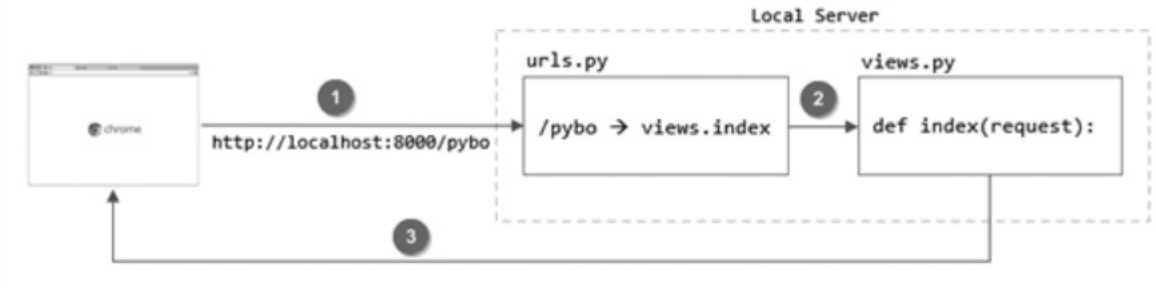
```
from django.http import HttpResponse


def index(request):
    return HttpResponse("안녕하세요 pybo에 오신것을 환영합니다.")
```

HttpResponse는 페이지 요청에 대한 응답을 할 때 사용하는 장고 클래스

HttpResponse에 문자열을 전달하여 문자열을 웹 브라우저에 출력

정리



1. 웹 브라우저 주소창에 localhost:8000/pybo 입력(장고 개발 서버에 /pybo 페이지 요청)
2. config/urls.py 파일에서 URL을 해석해 pybo/views.py 파일의 index 함수 호출
3. pybo/views.py 파일의 index 함수를 실행하고 그 결과를 웹 브라우저에 전달

1 - 3) url 분리

include 함수를 임포트해 `path('pybo/', include('pybo.urls'))`로 수정

pybo/로 시작되는 페이지 요청은 모두 pybo/urls.py 파일에 있는 URL 매핑을 참고하여 처리하라는 의미

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

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

---

2 - 1) 데이터 관리(모델)

장고는 모델로 데이터를 관리

보통 웹 개발에서는 데이터의 저장 · 조회를 위해 SQL 쿼리문을 이용(데이터 저장 · 조회를 위해서는 별도의 SQL 쿼리문을 배워야 한다)

BUT, **모델을 사용하면 SQL 쿼리문을 몰라도 데이터를 저장 · 조회 가능**

```
# settings.py의 DATABASES

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

settings.py를 보면 데이터베이스 엔진이 django.db.backends.sqlite3로 정의되어 있음

'NAME' 항목을 보면 데이터베이스는 BASE_DIR(프로젝트 디렉터리)에 있는 db.sqlite3이라는 파일에 저장

2 - 2) ORM

쿼리문이란 데이터베이스의 테이블을 생성, 수정, 삭제 또는 테이블 데이터의 내용을 생성, 수정, 삭제 시 사용하는 데이터베이스 질의(문법)

즉, 데이터 작업을 위한 문법

장고에는 ORM(object relational mapping)이라는 기능이 있음

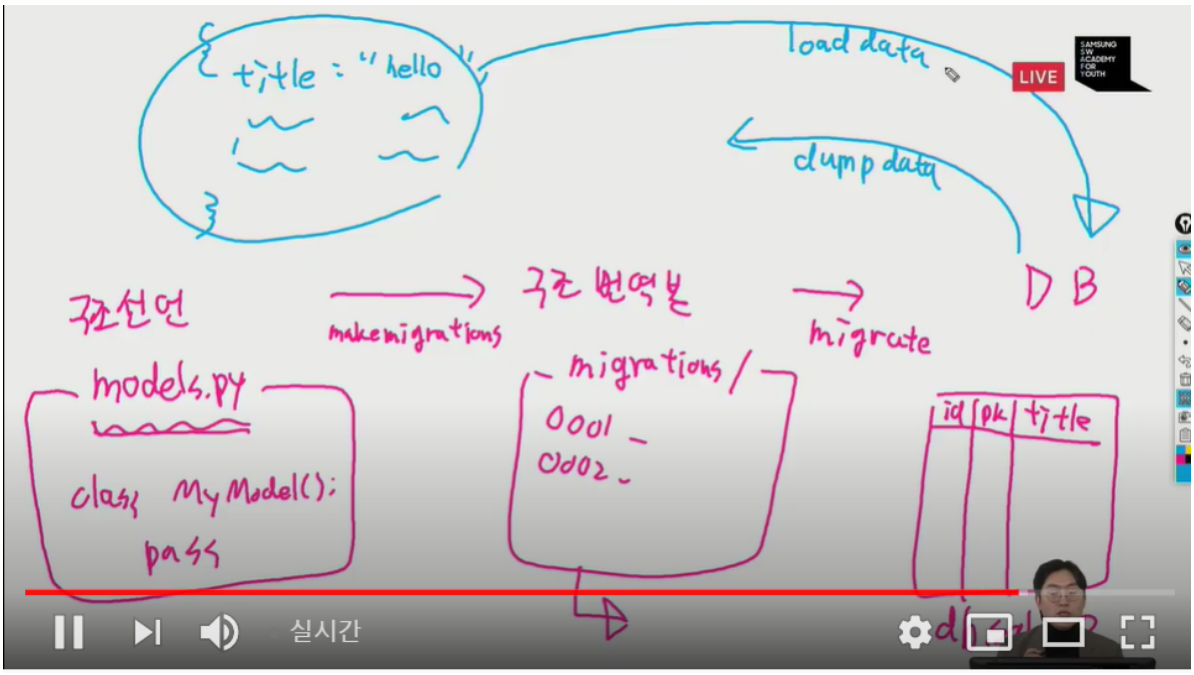
ORM은 파이썬으로 데이터 작업을 할 수 있게 해줌

즉, 장고에서는 쿼리문을 몰라도 파이썬을 안다면 데이터를 다룰 수 있다.

2 - 3) 모델 만들기

makemigrations 명령은 장고가 테이블 작업을 수행하기 위한 파일들을 생성한다. 많은 장고 학습자가 makemigrations 명령을 실제 테이블 생성 명령으로 오해한다. 하지만 실제 테이블 생성 명령은 migrate 이다.

makemigrations 명령을 수행하면 앱이름/migrations/0001_initial.py이라는 파일이 자동으로 생성
migrate 명령을 수행하면 등록된 앱에 있는 모델을 참조하여 실제 테이블을 생성



shell로 모델 데이터 만들기

```
# python manage.py shell
>>> from 앱이름.models import Question
>>> q = Question(subject='장고 모델 질문입니다.', content='id는 자동으로 생성되나요?')
>>> q.save()
>>> q.id
1
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
```

장고에서 저장된 모델 데이터는 Question.objects를 사용하여 조회할 수 있다. 그리고 Question.objects.all()은 Question에 저장된 모든 데이터를 조회하는 함수

(1)은 장고에서 Question 모델 데이터에 자동으로 입력해 준 id

2 - 4) 모델 데이터 조회하기

데이터 유형(쿼리셋)으로 보니까 조금 불편

model에 **str**메서드를 추가하면 편하게 볼수 있다

```
class Question(models.Model):
    subject = models.CharField(max_length=200)
    content = models.TextField()
    create_date = models.DateTimeField()

    def __str__(self):
        return self.subject # 이렇게 하면, 데이터 조회 시 id가 아닌 제목을 표시
```

```
>>> from pybo.models import Question
>>> Question.objects.all()
<QuerySet [<Question: 장고 모델 질문입니다.>]>

# 여기서 주의
# makemigrations, migrate 명령은 모델의 속성이 추가되거나 변경된 경우에 실행해야 하는 명령
# 그러므로 def를 추가했다고 해서 makemigrations와 migrate를 할 필요는 없다.
```

조건을 주어 Question 모델 데이터를 조회하고 싶다면 filter 함수

get 함수를 사용하면 리스트가 아닌 데이터 하나만 조회

```
# filter
>>> Question.objects.filter(id=1)
<QuerySet [<Question: 장고 모델 질문입니다.>]>

>>> Question.objects.filter(subject__contains='장고')
<QuerySet [<Question: 장고 모델 질문입니다.>]>

# get
>>> Question.objects.get(id=1)
<Question: 장고 모델 질문입니다.>
```

filter 함수는 여러 건의 데이터를 반환하지만, get 함수는 단 한 건의 데이터를 반환

(filter는 Queryset을 반환했는데, get은 Question을 반환한게 point!)

2 - 5) 모델 데이터 수정하기

```
>>> q = Question.objects.get(id=2)
>>> q
<Question: 장고 모델 질문입니다.>
>>> q.subject = 'Django Model Question'
>>> q.save()
>>> q
<Question: Django Model Question>
```

반드시 .save()를 해야 수정된다.

2 - 6) 모델 데이터 삭제하기

```
>>> q = Question.objects.get(id=1)
>>> q.delete()
(1, {'앱이름.Question': 1})
```

delete 함수를 수행하면 해당 데이터가 데이터베이스에서 즉시 삭제되며, 삭제된 데이터의 추가 정보를 반환

앞의 1은 삭제된 Question 모델 데이터의 id를 의미

{'앱이름.Question': 1}`은 삭제된 모델 데이터의 개수를 의미

---

3 - 1) admin 사용하기

```
python manage.py createsuperuser
```

---

4 - 1) render

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

def index(request):
    question_list = Question.objects.order_by('-create_date')
    context = {'question_list': question_list}
    return render(request, '"앱이름"/question_list.html', context)
```

render 함수는 context에 있는 Question 모델 데이터 question_list를

'앱이름'/question_list.html 파일(템플릿)에 적용하여 HTML 코드로 변환

4 - 2) templates의 dir

템플릿 디렉터리 위치 config/settings.py에 등록

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

장고는 DIRS에 설정한 디렉터리 외에도 특정 앱 디렉터리 하위에 있는 templates라는 이름의 디렉터리를 자동으로 템플릿 디렉터리로 인식

4 - 3) variable routing

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

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

int:는 question_id에 숫자가 매핑되었음을 의미

question_id에 2라는 값이 저장되고 views.detail 함수가 실행 > detail 함수에 question_id 인자 필요

4 - 4) get_object_or_404

get_object_or_404 함수는 모델의 기본키를 이용하여 모델 객체 한 건을 반환한다. pk에 해당하는 건이 없으면 오류 대신 404 페이지를 반환

```
from django.shortcuts import render, get_object_or_404

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

4 - 5) namespace

서로 다른 앱에서 같은 URL 별칭을 사용하면 중복 문제가 생김

(예시: 'board' 게시판의 'detail' url과 'articles' 게시판의 'detail' url)

네임스페이스: 각각의 앱이 관리하는 독립된 이름 공간

```
# app의 urls.py

from django.urls import path
from . import views

app_name = 'pybo'

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

```
# app의 templates 중 하나

{% for question in question_list %}
    <li><a href="{% url 'pybo:detail' question.id %}">{{ question.subject }}</a>
</li>
{% endfor %}
```

```
# view의 redirect에도 사용가능

redirect('pybo:detail', question_id=question.id)
```