

■ 서식 1 : 결과보고서 요약본

Project Based Learning 결과보고서 요약본

교과목명	국문	파이썬프로그래밍(2)		
	영문	Python Programming(2)		
PBL 관련 능력단위 (능력단위코드)	PBL 관련 능력단위요소 (능력단위요소코드)			
데이터 분석 기초 기술 활용(2001010516_21v1)	데이터 분석 언어 기본문법 활용하기			
	데이터 분석 언어특성 활용하기			
학년 반	2-2	조원		
프로젝트 주제	Django를 활용하여 웹을 이용한 사이트 만들기			
지도교수	심효선	산업체 참가여부	<input type="checkbox"/> 유 <input checked="" type="checkbox"/> 무	
참여학생	이은우(202025016)			
작품 개요 (주제 선정 이유)	<p>교재의 Ch3장에 나와 있는 가장 기본적인 구조인 Main 페이지, List 페이지, Detail 페이지를 구현한다. 처음에는 홈페이지에 접속하면 Main 페이지가 표시되도록 한다. Main 페이지를 누르면 콘텐츠 리스트가 보이는 List 페이지가 표시된다. List 페이지의 한 항목을 선택하면 Detail 페이지가 보인다.</p> <p>List 페이지와 Detail 페이지에 표시되는 데이터는 DB에 저장된 값을 읽어오도록 설계해야 한다. DB의 데이터는 기본은 admin 화면을 이용하여 이용하여 입력할 수 있도록 설정하고 form 기능을 이용하여 UI 화면을 통해 입력하도록 구현하면 가산점을 부여한다.</p> <p>컨텐츠의 내용은 자율적으로 아이디어를 발굴하여 선정하며 CSS를 이용하여 화면을 잘 구성한 경우 가산점을 부여한다</p> <p>Index(메인) => List(목록) => Detail(보다 한 단계 깊게 메뉴 선택이 되도록 구성하는 것도 가능하다</p>			
작품 구조도 (문제점 제시 및 개선방안)	<p>- 교재 3장에 나와 있는 List 화면을 참조한다.</p> <p>DB[파일명만]</p> <p>DB read (list.html)</p> <p>(detail.html)</p> <p>조건 = DB에서 데이터를 가져와야함.</p>			



그림 3-30 polls 애플리케이션 첫 화면 - 데이터 입력 후

- 교재 3장에 나와 있는 Detail 화면을 참조한다.

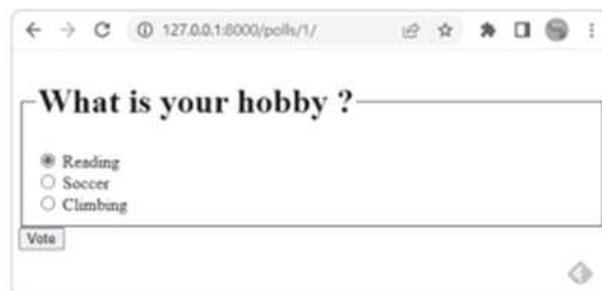


그림 3-31 polls/templates/polls/detail.html

- List화면, Detail 화면에 사용할 데이터는 DB에 admin 기능을 이용하여 입력할 수 있다.

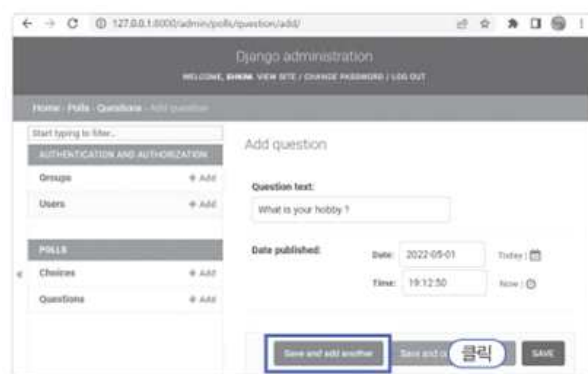


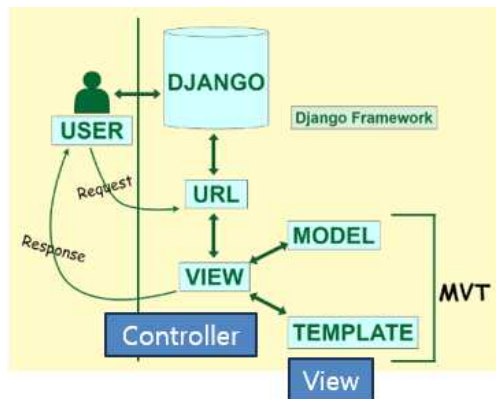
그림 3-26 Questions 데이터 입력 화면

관련 이론

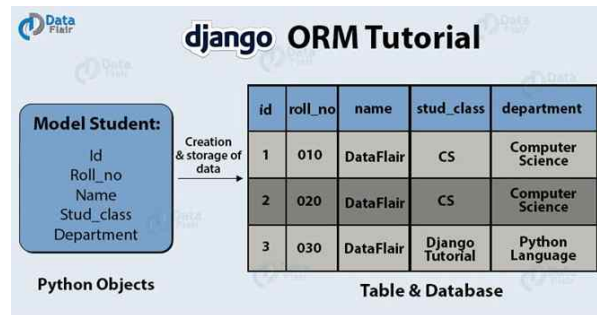
- 장고의 특징

- 1) 객체 관계 매핑 :Object-Relational Mapping(ORM)은 객체와 관계형 데이터베이스를 연결하는 역할을 하는 것이다. 데이터베이스 대신 객체(클래스)를 사용하여 데이터를 처리해 ORM이 자동으로 적절한 SQL구문이나 DB API를 호출하여 처리한다.
- 2) 자동으로 구성되는 관리자 : 데이터 베이스 관리 기능을 위한 관리자만의 화면을 제공한다. 이를 통해 애플리케이션에서 사용하는 테이블과 데이터를 쉽게 생성 및 변경이 가능함. 그러므로 개발자가 별도로 관리 기능을 개발할 필요가 없다.
- 3) 우아한 URL 설계 : 간편 URL체계를 도입했지만 장고에선 유연하면서도 강력한 URL 설계 기능을 제공하는 우아한 URL방식을 채택하였다. 또 정규 표현식을 사용하여 복잡한 URL도 표현이 가능하며, URL형태를 파이썬 함수에 1:1로 연결하도록 설계되어 이해가 쉬운 코드작성, 편리한 개발이 가능하다.
- 4) 자체 템플릿 시스템 : 내부적으로 확장이 가능하고 디자인하기 쉬운 강력한 템플릿 시스템을 가지고 있음. 이를 통해 화면 디자인과 로직에 관한 코딩을 분리해 독립적으로 프로그래밍이 가능하다. 즉 HTML과 같은 텍스트형 언어를 쉽게 다룰 수 있게 개발됨
- 5) 캐시 시스템 : 캐시용 페이지를 메모리, DB 내부, 파일 시스템 중 아무곳에나 저장할 수 있음. 또한 캐시 단위를 페이지에서부터 사이트 전체 또는 특정 뷰의 결과, 템플릿의 일부 영역만을 지정하여 저장할 수도 있다.
- 6) 다국어 지원 : 장고는 동일한 소스 코드를 다른 나라에서도 사용할 수 있도록 텍스트 번역, 날짜/시간/숫자 포맷, 타임존 지정 등과 같은 다국어 환경을 제공한다.
- 7) 풍부한 개발 환경 : 프로그래밍에 도움이 되는 여러 기능을 제공하는데 특히 테스트용 웹 서버를 포함하고 있어 개발과정에서 웹 서버가 없어도 테스트를 진행할 수 있다. 또한 디버깅 모드를 사용할 경우에는 에러를 쉽게 파악하고 해결할 수 있게 상세한 메시지를 보여준다.
- 8) 소스변경사항 자동 반영 : *.py 파일의 변경 여부를 감시하고 있다가 변경이 되면 실행파일에 변경 내역을 바로 반영함.

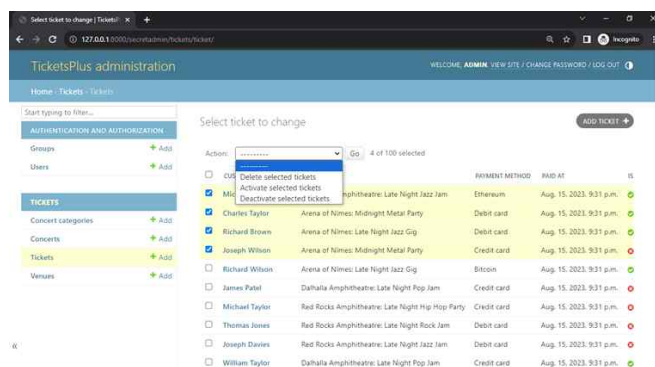
- MVT 패턴



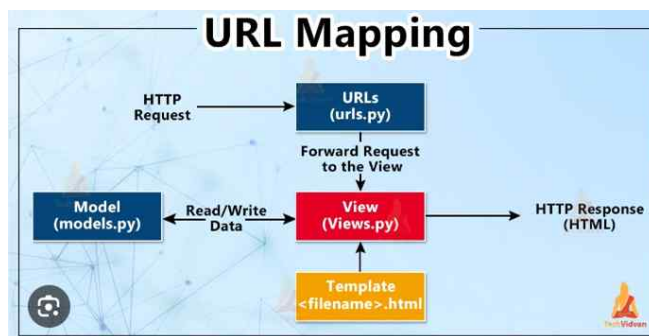
- 객체관계매핑



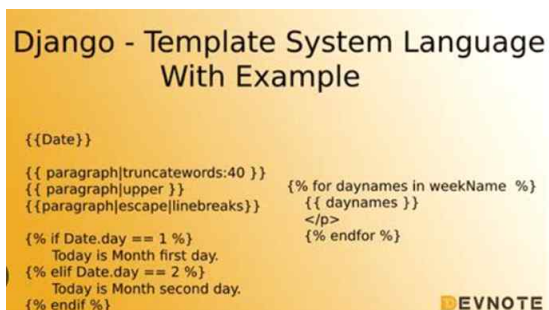
- 자동으로 구성되는 관리자 화면



- 우아한 URL 설계



- 자체 템플릿 시스템



	<p>- MVT 코딩순서</p> <p>프로젝트 뼈대 만들기 : 프로젝트 및 애플리케이션 디렉터리 생성</p> <p>모델 코딩 하기 : models.py , admin.py 파일</p> <p>URLconf 코딩 하기 : urls.py 파일</p> <p>템플릿 코딩 하기 : templates/*.html)</p> <p>뷰 코딩 하기 : views.py</p>
<p>결과물 제작 (문제점 개선사항)</p>	<p>1. 먼저 프로젝트 뼈대인 mysite 디렉터리와, 애플리케이션인 pbl 디렉터리를 생성한다.</p> <ul style="list-style-type: none"> ▶ django-admin startproject mysite(프로젝트명) . ◆ 다음 .을 붙이면 하위의 폴더만 생성한다. <p>2. 모델을 코딩하는데 models.py와 admin.py 파일을 보면</p> <p>★models.py</p> <pre> from django.db import models class Member(models.Model): name = models.CharField(max_length=100) photo = models.ImageField(upload_to='photos/') student_id = models.CharField(max_length=20) hobby = models.TextField() def __str__(self): return self.name class Friend(models.Model): name = models.CharField(max_length=100) relation = models.CharField(max_length=50) member = models.ForeignKey(Member, on_delete=models.CASCADE, related_name='friends') def __str__(self): return f"{self.name} ({self.relation})" </pre> <p>팀원의 정보를 저장할 Member 테이블과 팀원의 친구들 정보를 저장할 Friend 테이블이 필요하다. 이 테이블을 정의하는 파일이 models.py파일이며 django에선 테이블을 하나의 클래스로 정의하고 테이블의 컬럼은 클래스의 변수(속성)으로 매핑한다.</p> <p>코드해석을 하면 Member라는 테이블(클래스)를 생성하고, 테이블 컬럼(클래스 변수)로 name(팀원의 이름), photo(팀원의 사진), student_id(팀원의 학번), hobby(팀원의 취미) 등을 생성한다. 각각 타입에 맞게 varchar, text, image타입으로 변수에 저장하고, Friend라는 테이블(클래스)도 생성, 테이블 컬럼(클래스 변수)로 name(친구의 이름), relation(팀원과의 관계), member(어떤 팀원의 친구인지)을 생성하는데 여기서 member는 FK(외래키)를 통해 Friend의 모델이 Member 모델과 연결되며, related_name='friends'를 통해 Member 객체에서 member.friends를 통해 연결된 모든 friends객체를 가져올 수 있음.</p>

●Member클래스

테이블(T): **pbl_member**

	id	name	photo	student_id	hobby
	...	필터	필터	필터	필터
1	1	이은우	photos/202025016-이은우_xRGNmOV.jpg	2020165489	게임
2	2	팀원1	photos/1581304118739.jpg	2020165421	독서, 축구, 코딩
3	3	팀원2	photos/Ugh.png	2020164892	원시인

▶테이블의 컬럼은 id, name(이름), photo(팀원 사진), student_id(학번), hobby(취미)로 구성되어 있다. 즉 이 컬럼명들을 클래스 변수명에 그대로 매핑하면 된다. PK를 클래스에 지정하지 않아도 장고는 항상 PK에 대한 속성을 Not Null 및 Autoincrement로, 이름은 id로 해서 자동으로 만들

●Friend클래스

테이블(T): **pbl_friend**

	id	name	relation	member_id
	...	필터	필터	필터
1	1	홍길동	고등학교 친구	1
2	2	형길동	고등학교 친구	2
3	3	김길동	중학교 친구	1
4	4	박길동	고등학교 친구	2

▶테이블의 컬럼은 id, name(친구이름), relation(팀원과의 관계), member_id(Member클래스의 팀원들의 id값)로 구성되어 있다. FK(Foreign Key)는 항상 다른 테이블의 PK에 연결되므로 Member 클래스의 id 변수까지 지정할 필요 없이 Member클래스만 지정하면 된다. 실제 테이블에서 FK로 지정된 컬럼은 _id접미사가 붙는다. 즉 member_id는 member클래스의 외래키라고 할 수 있다.

★admin.py

Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

+ Add Change

Users

+ Add Change

기본적으로 admin사이트에 접속하면 다음과 같이 기본으로 제공하는 User, Groups 테이블만 보이는데 아래의 코드처럼 admin.py에 코드를 추가하면

```

from django.contrib import admin
from .models import Member, Friend

#admin.site.register(Member) = models.py의 Member 클래스를 불러와 admin사이트에 등록
#admin.site.register(Friend) = models.py의 Friend 클래스를 불러와 admin사이트에 등록
@admin.register(Member)
class MemberAdmin(admin.ModelAdmin): #이는 더 가시성이 좋게 테이블의 데이터값도 표현해줌
    list_display = ('name', 'student_id', 'hobby') # 관리자 페이지에 표시할 필드

@admin.register(Friend)
class FriendAdmin(admin.ModelAdmin):
    list_display = ('name', 'relation', 'member') # 관리자 페이지에 표시할 필드

```

<input type="checkbox"/>	NAME	STUDENT ID	HOBBY
<input type="checkbox"/>	팀원2	2020164892	원시인
<input type="checkbox"/>	팀원1	2020165421	독서, 축구, 코딩
<input type="checkbox"/>	이은우	2020165489	게임

보통은 admin.site.register(클래스명) 하여 models.py에 있던 클래스를 불러와 admin사이트에 추가해준다. 하지만 각 테이블마다 상세한 정보를 표시하고 싶었기 때문에 일단 @admin.register(클래스명)을 이용해 admin사이트에 추가를 해주고, class를 이용해 admin.ModelAdmin을 상속받아 다양한 속성을 추가할 수 있게 하여 list_display를 통해 Member모델 데이터를 볼 때, 각 행에 이름, 학번, 취미 필드를 표시해준다.

Friend도 마찬가지로이다.

이렇게 models.py, admin.py파일을 작성하고 DB에 변경사항을 반영하기 위해 makemigrations(변경사항을 확인=DB에 반영X), migrate(변경사항 적용) 명령으로 반영한다.

2.1 form기능 구현을 위한 forms.py 추가

추가적인 기능으로 사용자가 보는 UI에서 팀원을 추가하기 위한 form기능을 추가한다. “새 팀원 추가”라는 버튼을 클릭하면 팀원의 정보[이름, 사진등록, 학번, 취미]를 입력하는 화면을 pbl.html에서 UI에 나타낸다. 동일한 알고리즘으로 각 팀원의 친구목록을 보여주는 friend.html에서도 “새 친구 추가”라는 버튼을 클릭하면 새로운 정보의 친구가 추가되는 form기능을 구현한다. 각 템플릿 파일에도 내용이 추가되는데 이는 템플릿 파일 파트에서 설명

새 팀원 추가

Name:

Photo: 선택된 파일 없음

Student id:

Hobby:

취미

새 팀원 추가

Name:

Relation:

Member:


```

from django import forms
from .models import Member, Friend

class MemberForm(forms.ModelForm):
    class Meta:
        model = Member
        fields = ['name', 'photo', 'student_id', 'hobby'] # 입력받을 필드 정의
        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder': '이름'}),
            'student_id': forms.TextInput(attrs={'class': 'form-control', 'placeholder': '학번'}),
            'hobby': forms.Textarea(attrs={'class': 'form-control', 'placeholder': '취미'}),
        }

class FriendForm(forms.ModelForm):
    class Meta:
        model = Friend
        fields = ['name', 'relation', 'member'] # 입력받을 필드 정의
        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder': '친구 이름'}),
            'relation': forms.TextInput(attrs={'class': 'form-control', 'placeholder': '관계'}),
            'member': forms.Select(attrs={'class': 'form-control'}), # ForeignKey 필드는 드롭다운으로 표시
        }

```

팀원을 추가하는 “새 팀원 추가” form의 동작을 위해 memberform 클래스와 친구를 추가하는 “새 친구 추가” form 동작을 위해 friendForm 클래스를 만듦. 웹에서 해당 버튼을 클릭하면 각 항목[이름, 학번, 관계 등]이 나온다.

※이것 또한 각 템플릿 파일 파트에서 상세히 소개함

3. URLconf를 코딩하기 위해 앱폴더에 urls.py를 만들

```

from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('pbl/<int:membered>/', views.member_detail, name='member_detail'),
    path('pbl/<int:membered>/friend/', views.friend_list, name='friend_list'),
]

```

클라이언트로부터 요청을 받으면 장고는 가장 먼저 요청에 들어있는 URL을 분석, 즉 urls.py 파일에 정의된 URL패턴과 매칭되는지를 분석함.

pbl/<int:membered>/ 부분은 url주소가, view.member_detail 부분은 view라는 처리함수가 매핑이 되어있음을 알 수 있다. 그리고 이 동작은 member_detail이라는 이름을 지정하여 이를 사용할 때 해당 url과 뷰함수가 매핑이 되어 처리한다. friend도 마찬가지

이때 <int:membered>처럼 꺾쇠 부분이 URL패턴의 일부 문자열을 추출하기 위한 것이며 이는 <type:name> 형식으로 사용

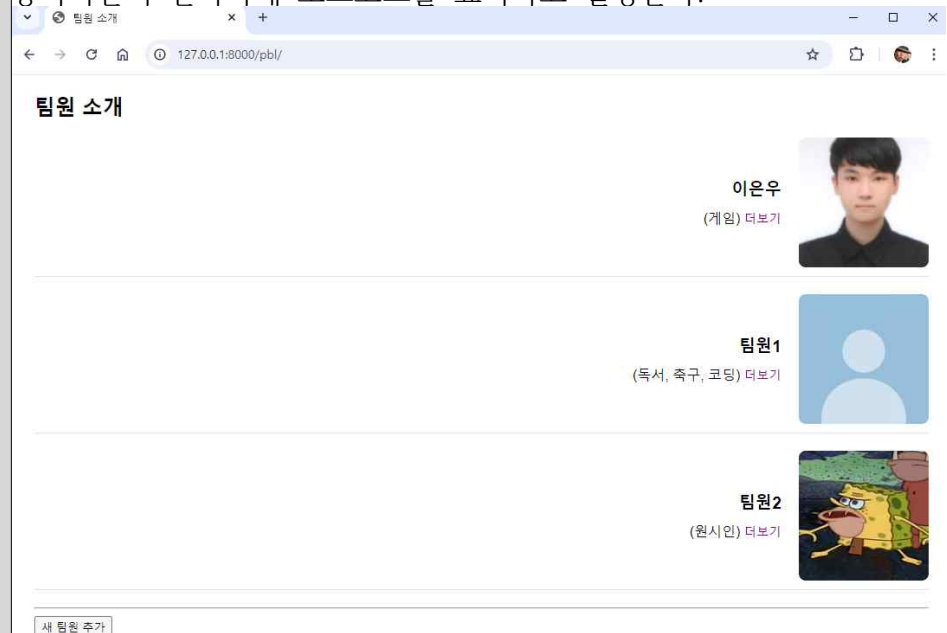
4. 템플릿(templates) 및 뷰(view) 함수 코딩하기

작품의 구조는 팀원을 소개하고 자세한 정보를 통해 그 팀원의 친구들을 알수 있는 웹페이지이며, 팀원과 팀원의 친구의 관계도 파악할 수 있다. 첫페이지(pbl.html)엔 팀원의 이름과 취미를 간단히 소개하고, 사진을 포함시켜 팀원을 간단히 소개하는 페이지를 구현한다. 이때 팀원의 자세한 정보를 보기 위해 더 보기를 누르면 해당 팀원의 학번까지 파악할 수 있는 페이지(member.html)로 이동하고, 그 외 그 팀원의 친구들 정보를 보기 위해 페이지(friend.html)를 이동하면 그 팀원의 친구가 어떤 관계인지를 나타낸다.

즉 코딩해야할 템플릿은 pbl.html, member.html, friend.html 3개의 템플릿과 그에 따른 views.py를 코딩해야 한다.

[1. pbl(home) 화면 구현]

화면을 캡처하고 화면의 의미를 기술한다. 그리고 코드가 어떻게 동작하는지 간략하게 소스코드를 표시하고 설명한다.



★pbl.html 템플릿 파일

```
{% load static %}
<title>팀원 소개</title>
<link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}">
<script src="{% static 'js/script.js' %}" defer></script>

<div class="team-container">
<h2>팀원 소개</h2>
    {% for member in members %}
    <div class="team-member">
        <div class="member-info">
            <h3>{{ member.name }}</h3>
            ({{ member.hobby }})
            <a href="{% url 'member_detail' member.id %}">더보기</a>
        </div>
        
    </div>
    {% endfor %}
</div>
<hr>
<button onclick="toggleForm()">새 팀원 추가</button>
<div id="form-container" style="display: none;">
    <form method="post" enctype="multipart/form-data">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">추가하기</button>
    </form>
</div>
</div>
```

먼저 메인 화면인 pbl.html이다. 보다시피 url이 pbl/로 시작되고 있음을 알 수 있다. {% load static %}은 정적 파일의 경로를 가져오기 위해 사용되고, 정적파일로 이용되는 것은 form기능의 동작을 원활하게

하기 위해 javascript를, 웹의 디자인을 위해 CSS를 이용하고 있다. 이 js파일은 pbl폴더 안 static/js 파일 내부 script.js가 위치 해있으며 CSS파일도 static/css 파일 내부에 style.css라는 파일을 불러온다. 그리고 그 외 <h2> 및 <div>태그를 사용하여 웹을 꾸며주었다.

<div>태그의 사용 이유는 데이터를 그룹화하고 콘텐츠를 섹션별로 나누기 위해 사용되며, 이 특정 영역에 CSS 스타일을 적용하기 위해 필요하다. 다음은 DB에 접근할 템플릿코드를 해석한다.

{% for member in members %} : members가 팀원들의 목록을 템플릿에 전달해 for문으로 순회하고 있으며, {% url 'member_detail' member.id %}를 통해 각 멤버들의 세부정보를 불러오는 URL에 접근하고 있다. 즉 “더보기” 버튼을 클릭하면 member.html로 연결되며 각 member.id에 해당하는 정보를 불러온다. <a>는 Client가 Server로 url주소를 요청함.

또 불러온 정보는 {{member.name}}과 {{member.hobby}}에 표현한다.

그리고 이미지 필드에 접근하기 위해 {{ member.photo.url }}를 통해 DB에 저장된 이미지 파일 경로를 사용했다.

새 팀원 추가

Name:

Photo: 선택된 파일 없음

Student id:

취미

Hobby:

추가하기

추가적으로 앞전에 설명한 forms.py를 추가해 form기능을 구현하였는데 맨 좌측 아래에 보면 “새 팀원 추가”라는 버튼이 존재하고, 버튼을 누르면 아래 사진과 같이 이름[name], 이미지[photo], 학번[student_id], 취미[hobby]를 입력하는 칸이 생기고, 이를 추가함과 동시에 DB에 새롭게 추가 되는 기능을 구현하였다.

form method = “POST”로 폼이 입력된 데이터를 POST방식으로 보내고, enctype=“mulitpart/form-data를 통해 파일 데이터가 텍스트 데이터와 함께 전송될수 있게 설정한다. {{ form.as_p }}를 통해 새로운 팀원을 추가하기 위한 form화면을 표시한다.

★views.py의 home() 뷰함수

```
from django.shortcuts import render, redirect, get_object_or_404
from .models import Member
from .forms import MemberForm, FriendForm

def home(request):
    members = Member.objects.all()
    if request.method == 'POST':
        form = MemberForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            return redirect('home') # 새로 추가 후 목록 새로고침
    else:
        form = MemberForm()
    context = {'members': members, 'form': form, }

    return render(request, 'pbl.html', context)
```




views.py라는 파일에서 각 뷰 함수들과 템플릿 파일을 연동시켜 html의 동작과정을 위해 뷰함수들을 코딩한다. 먼저 home() 함수는 pbl.html 템플릿의 동작을 처리하기 위해 코딩하였고 먼저 템플릿에 넘겨줄 객체 이름은 member이다. 이 객체는 Member테이블에서 레코드를 모두 가져와 만든다. 그리고 요청이 POST방식이면 사용자 입력 데이터를 폼에 전달하고, 유효하면 데이터베이스에 저장한 뒤 **홈페이지로 리다이렉트**한다. 만약 POST가 아닌 경우 기본 폼을 초기화 한다. 그리고 템플릿에 넘겨주는 방식은 딕셔너리 타입으로 템플릿에서 사용될 변수명과 그 변수명에 해당하는 객체를 매핑하는 사전으로 context 변수를 만들어 render()함수에 보낸다. render()함수는 pbl.html에 클라이언트가 가공해서 만든 data인 request와 서버가 DB에 접근해서 만든 data인 context 변수를 적용해 사용자에게 보여줄 최종 HTML텍스트를 만들고 이를 담아 HTTPResponse객체를 반환, 즉 home() 뷰함수는 최종적으로 클라이언트에게 응답할 데이터인 HTTPResponse객체를 반환한다.

```
def add_member(request):
    if request.method == 'POST':
        form = MemberForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            return redirect('home') # 메인 페이지로 리다이렉트
        else:
            form = MemberForm()
    return render(request, 'add_member.html', {'form': form})
```

추가적으로 form기능에서 팀원을 추가하는 데에도 add_member()라는 뷰함수를 추가한다. 동작과정은 home()과 거의 동일하게 동작한다.

[2. member 화면 구현]

리스트 화면을 캡처하고 화면의 의미를 기술한다. 그리고 코드가 어떻게 동작하는지 간략하게 소스코드를 표시하고 설명한다.

이은우의 정보	팀원1의 정보	팀원2의 정보
		
학번: 2020165489	학번: 2020165421	학번: 2020164892
취미: 게임	취미: 독서, 축구, 코딩	취미: 원시인
친구 목록 보기	친구 목록 보기	친구 목록 보기
홈으로 돌아가기	홈으로 돌아가기	홈으로 돌아가기

★member.html 템플릿 파일

```
<title>{{ member.name }}의 페이지</title>
<h1>{{ member.name }}의 정보</h1>
<div>
    
    <p>학번: {{ member.student_id }}</p>
    <p>취미: {{ member.hobby }}</p>
</div>
<a href="{% url 'friend_list' member.id %}">친구 목록 보기</a><br>
<a href="{% url 'home' %}">홈으로 돌아가기</a>
```

해당 템플릿은 pbl.html에서 각 팀원의 “더보기”를 클릭해서 학번과 취미를 보여주고, 그 팀원의 친구 목록을 보기위한 징검다리 역할이라고 볼 수 있다. 앞에 pbl.html에서 각 팀원의 정보를 DB에서 가져왔고 내가 고른 팀원의 정보를 보는 detail.html로 member.name, member.photo.url, member.student_id, member.hobby 데이터가 넘어오면서 화면에 보여준다. 밑에는 내가 고른 팀원의 친구목록을 볼 수 있는 `{% url 'friend_list' member.id %}`이 사용되었다. views.py에 있는 friend_list() 뷰함수 파일을 불러오며 member.id를 통해 팀원을 구분함.

★views.py의 member_detail() 뷰함수

```
def member_detail(request, membered):
    member = get_object_or_404(Member, id=membered)
    return render(request, 'member.html', {'member': member})
```

url에 들어있는 membered 변수를 인자로 받으며, get_object_or_404() 단축함수를 사용한다. Member 모델 클래스로부터 id=membered 검색조건에 맞는 객체를 조회함. member.html에 클라이언트에서 가공해서 만든 data를 request로 받는건 동일하고, context변수를 적용해 사용자에게 보여줄 최종 HTML 텍스트를 만들고, 이를 담아서 HTTPResponse객체를 반환한다. 템플릿에게 넘겨주는 contest 디렉터리를 render()함수의 인자로 직접 쓰고있는 형태만 home() 뷰함수와 다르다. member라는 변수를 사용.

[3. friend 화면 구현]

디테일 화면을 캡처하고 화면의 의미를 기술한다. 그리고 코드가 어떻게 동작하는지 간략하게 소스코드를 표시하고 설명한다.

이은우의 친구들	팀원1의 친구들	팀원2의 친구들
홍길동 관계: 고등학교 친구	형길동 관계: 고등학교 친구	구길동 관계: 초등학교 친구
김길동 관계: 중학교 친구	박길동 관계: 고등학교 친구	<div>새 친구 추가</div> <div>뒤로 가기</div> <div>홈으로 돌아가기</div>
<div>새 친구 추가</div> <div>뒤로 가기</div> <div>홈으로 돌아가기</div>	<div>새 친구 추가</div> <div>뒤로 가기</div> <div>홈으로 돌아가기</div>	

★friend.html 템플릿 파일

```
{% load static %}
<title>{{ member.name }}의 친구 목록</title>
<script src="{% static 'js/script.js' %}" defer></script>
<h1>{{ member.name }}의 친구들</h1>
<div>
    {% for friend in friends %}
    <div style="...">
        <h3>{{ friend.name }}</h3>
        <p>관계: {{ friend.relation }}</p>
    </div>
    {% endfor %}
    <hr>
    <button onclick="toggleForm()">새 친구 추가</button>
    <div id="form-container" style="...">
        <form method="post" enctype="multipart/form-data">
            {% csrf_token %}
            {{ form.as_p }}
            <button type="submit">추가하기</button>
        </form>
    </div>
</div>
<a href="{% url 'member_detail' member.id %}">뒤로 가기</a><br>
<a href="{% url 'home' %}">홈으로 돌아가기</a>
```

{% for friend in friends %} : 뷰함수에서 넘어온 friends객체가 친구들 목록을 템플릿에 넘겨주어 for문으로 순회하고 있음. 내가 선택한 팀원의 친구들의 이름과 친구 관계를 Friend 테이블에서 모두 불러옴.

새 친구 추가

Name: 친구 이름

Relation: 관계

Member: ----- ▼

추가하기

Member: ----- ▼

추가하기

Member: ----- ▼

이은우

팀원1

팀원2

추가하기

뒤로 가기

홈으로 돌아가기

그리고 아래코드는 마찬가지로 pbl.html에서 했듯이 form기능을 구현하였는데 “새 팀원 추가”라는 버튼을 누르면 아래 사진과 같이 이름[name], 관계[relation]를 입력하는 칸이 생기고, 누구의 친구인지[member] 선택하는 list목록이 추가되었다.

★views.py의 firend_list() 뷰함수

```
def friend_list(request, membered):
    member = get_object_or_404(Member, id=membered)
    friends = member.friends.all()
    if request.method == 'POST':
        form = FriendForm(request.POST)
        if form.is_valid():
            friend = form.save(commit=False)
            friend.member = member # 연결된 팀원 지정
            friend.save()
            return redirect('friend_list', membered=member.id) # 새로고침
    else:
        form = FriendForm()

    context={'member': member, 'friends': friends, 'form': form,}

    return render(request, 'friend.html', context)
```


여기서도 각 팀원의 친구목록을 보기위해 url에 들어있는 membered 변수를 인자로 받는다. 똑같이 get_object_or_404() 단축함수를 사용하고 Member 모델 클래스로 부터 id=membered 검색조건에 맞는 객체를 조회함. **member.friends.all()** : member객체의 friends 관계(외래키)를 통해 친구 목록을 모두 가져온다.

그리고 요청이 POST방식이면 사용자 입력 데이터를 폼에 전달하고, 유효하면 데이터베이스에 저장한 뒤 **홈페이지로 리다이렉트한다**.

만약 POST가 아닌 경우 기본 폼을 초기화 한다.

그리고 템플릿에 넘겨주는 방식은 딕셔너리 타입으로 템플릿에서 사용될 변수명과 그 변수명에 해당하는 객체를 매핑하는 사전으로 context 변수를 만들어 render()함수에 보낸다.

render()함수는 pbl.html에 클라이언트가 가공해서 만든 data인 request와 서버가 DB에 접근해서 만든 data인 context 변수를 적용해 사용자에게 보여줄 최종 HTML텍스트를 만들고 이를 담아 HTTPResponse객체를 반환, 즉 home() 뷰함수는 최종적으로 클라이언트에게 응답할 데이터인 HTTPResponse객체를 반환한다. 사실상 pbl.html의 home()뷰함수와 다를게 없다.

그저 다른점은 member.friends.all()를 통해 member객체의 friends관계(외래키)를 참조한다는 점이다.

새 팀원 추가

Name: 팀원3

Photo:

파일 선택

 박수.jpg

Student id: 202044922


영화

Hobby:

추가하기

팀원2

(원시인) 더보기



팀원3

(영화) 더보기



테이블(T): pbl_member

	id	name	photo	student_id	hobby
...	필터	필터		필터	필터
1	1	이은우	photos/202025016-이은우_xRGNmOV.jpg	2020165489	게임
2	2	팀원1	photos/1581304118739.jpg	2020165421	독서, 축구, 코딩
3	3	팀원2	photos/Ugh.png	2020164892	원시인
4	4	팀원3	photos/박수.jpg	202044922	영화

폼을 이용해 팀원을 추가하여 DB에 정상등록된 모습

	<div><div>팀원3의 친구 목록</div><div>← → ↺ ↻ 127.0.0.1:8000/pbl/4/friend/</div><div>팀원3의 친구들</div><div>새 친구 추가</div><div>Name: 친구1</div><div>Relation: 고등학교 친구</div><div>Member: 팀원3</div><div>추가하기</div><div>뒤로 가기</div><div>홈으로 돌아가기</div><div>테이블(T): pbl_friend</div><table><tr><th></th><th>id</th><th>name</th><th>relation</th><th>member_id</th></tr><tr><td>...</td><td></td><td>필터</td><td>필터</td><td>필터</td></tr><tr><td>1</td><td>1</td><td>홍길동</td><td>고등학교 친구</td><td>1</td></tr><tr><td>2</td><td>2</td><td>형길동</td><td>고등학교 친구</td><td>2</td></tr><tr><td>3</td><td>3</td><td>김길동</td><td>중학교 친구</td><td>1</td></tr><tr><td>4</td><td>4</td><td>박길동</td><td>고등학교 친구</td><td>2</td></tr><tr><td>5</td><td>5</td><td>구길동</td><td>초등학교 친구</td><td>3</td></tr><tr><td>6</td><td>6</td><td>친구1</td><td>고등학교 친구</td><td>4</td></tr></table><div>새 친구를 등록하여 DB에 정상 등록된 모습</div></div>		id	name	relation	member_id	...		필터	필터	필터	1	1	홍길동	고등학교 친구	1	2	2	형길동	고등학교 친구	2	3	3	김길동	중학교 친구	1	4	4	박길동	고등학교 친구	2	5	5	구길동	초등학교 친구	3	6	6	친구1	고등학교 친구	4
	id	name	relation	member_id																																					
...		필터	필터	필터																																					
1	1	홍길동	고등학교 친구	1																																					
2	2	형길동	고등학교 친구	2																																					
3	3	김길동	중학교 친구	1																																					
4	4	박길동	고등학교 친구	2																																					
5	5	구길동	초등학교 친구	3																																					
6	6	친구1	고등학교 친구	4																																					
기대 효과 및 활용방안	<p>장고를 이용하여 프로젝트를 생성하고 서버를 실행하여 UI가 있는 앱을 생성할 수 있는 방법을 배운다. DB를 이용하여 CRUD 동작을 할 수 있는 방법을 체득하고 MVT 방식으로 어떻게 구동시키는지의 방법을 배운다. 간단한 UI앱의 제작을 통해서 향후에 인공지능 기능을 앱에 연동하여 인공지능앱을 만들 수 있는 능력을 키울 수 있다.</p>																																								
2024 년 12 월 8 일																																									
지도교수 심 효 선 (인)																																									