목차

1. Django Intro

Web application (web service) 개발
인터넷을 통해 사용자에게 제공되는 소프트웨어 프로그램을 구축하는 과정

다양한 디바이스(모바일, 태블릿, PC)에서 웹 브라우저를 통해 접근하고 사용가능

Client
서비스를 요청하는 주체
(웹 사용자의 인터넷이 연결된 장치, 웹 브라우저)

Server
클라이언트의 요청에 응답하는 주체
(웹 페이지, 앱을 저장하는 컴퓨터)

Frontend
사용자 인터페이스(UI)를 구성하고, 사용자가 애플리케이션과 상호작용할 수 있도록 함
(HTML, CSS, JavaScript, 프론트엔드 프레임워크 등)

Backend
서버 측에 동작하며, 클라이언트의 요청에 대한 처리와 데이터베이스와의 상호작용을 담당
(서버 언어, 백엔드 프레임워크, DB, API, 보안 등)

Web Framework
웹 애플리케이션을 빠르게 개발할 수 있도록 도와주는 도구
(개발에 필요한 기본 구조, 규칙, 라이브러리 등을 제공)

+

django
Python 기반의 대표적인 웹 프레임워크

django 특징

1. 다양성
   Python 기반으로 소셜 미디어 및 빅데이터 관리 등 광범위한 서비스 개발에 적합

2. 확장성
   대량의 데이터에 대해 빠르고 유연하게 확장할 수 있는 기능을 제공

3. 보안
   취약점으로부터 보호하는 보안 기능이 기본적으로 내장

4. 커뮤니티 지원
   개발자를 위한 지원, 문서, 업데이트를 제공하는 활성화 된 커뮤니티

가상환경
Python 애플리케이션과 그에 따른 패키지들을 격리하여 관리할 수 있는 독립적인 실행 환경

의존성 패키지
패키지가 다른 패키지의 기능&코드를 사용하기 때문에 그 패키지가 존재해야 제대로 작동하는 관계

가상환경을 사용하는 이유

1. 의존성 관리
   라이브러리 및 패키지를 각 프로젝트마다 독립적으로 사용가능

2. 팀 프로젝트 협업
   모든 팀원이 동일한 환경&의존성 위에서 작업해 버전간 충돌을 방지

LTS (Long-Term Support)
프레임워크&라이브러리 등에서 장기간 지원되는 안정적인 버전

기업 및 대규모 프로젝트에서는 소프트웨어 업그레이드에 많은 비용과 시간이 필요하기에 안정적이고 장기간 지원되는 버전이 필요

디자인 패턴
소프트웨어 설계에서 발생하는 문제 해결을 위한 일반적인 해결책
(공통적인 문제를 해결하는데 쓰이는 형식화된 관행)

MVC 디자인 패턴
Model, View, Controller
애플리케이션을 구조화하는 대표적인 패턴
('데이터 & 사용자 인터페이스 & 비즈니스 로직'을 분리)

-> 시각적 요소와 뒤에서 실행되는 로직을 서로 영향 없이, 독립적이고 쉽게 유지 보수할 수 있기 위함

MTV 디자인 패턴
Model, Template, View
Django에서 애플리케이션을 구조화하는 패턴
(기본 MVC 패턴과 동일, 명칭만 다르게 정의한 것 / View -> Template, Controller -> View)

1. Model
   데이터와 관련된 로직을 관리
   응용프로그램의 데이터 구조를 정의하고 데이터베이스의 기록을 관리

2. Template
   레이아웃과 화면을 처리
   화면상의 사용자 인터페이스 구조와 레이아웃을 정의

3. View
   Model & Template과 관련한 로직을 처리해서 응답을 반환
   클라이언트의 요청에 대해 처리를 분기하는 역할

Django project
애플리케이션의 집합 (DB설정, URL연결, 전체 앱 설정 등을 처리)

Django application
독립적으로 작동하는 기능 단위 모듈 (각자 특정한 기능을 담당하며 다른 앱들과 함께 하나의 프로젝트를 구성)

프로젝트 구조
settings.py : 프로젝트의 모든 설정 관리
urls.py : 요청 들어오는 URL에 따라 해당하는 views를 연결
admin.py : 관리자용 페이지 설정
models.py : DB와 관련된 Model을 정의 (MTV 패턴의 M)
views.py : HTTP 요청을 처리하고 해당 요청에 대한 응답을 반환 (MTV 패턴의 V)

URLs
http://기본주소/index/로 요청이 들어오면 view 함수의 index 함수를 호출한다는 뜻
-> url경로는 반드시 / 로 끝나야함
-> 예시 : path('index/', views.index)

View
특정 경로에 있는 template과 request 객체를 결합해 응답 객체를 반환하는 view 함수를 정의

모든 view 함수는 첫번째 인자로 request 요청 객체를 필수로 받는다.

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

Template
생성 순서

1. articles 앱 폴더 안에 templates 폴더 생성

2. templates 폴더 안에 artilces 폴더 생성

3. articles 폴더 안에 템플릿 파일(html) 생성

render 함수
주어진 템플릿을 주어진 context 데이터와 결합하고 렌더링 된 텍스트와 함께 HttpResponse 응답 객체를 반환하는 함수

예시 : render(request, template_name, context)

1. request : 응답을 생성하는데 사용되는 요청 객체
2. template_name : 템플릿 이름의 경로
3. context : 템플릿에서 사용할 데이터 (딕셔너리 타입으로 작성)

---

2. Templates & URLs

Django Template system
데이터 표현을 제어하면서, 표현과 관련된 부분을 담당

Django Template Language (DTL)
Template에서 조건, 반복, 변수 등의 프로그래밍적 기능을 제공하는 시스템

DTL Syntax
종류 : Variable, Filters, Tags, Comments

1. Variable
   render 함수의 세번째 인자로 딕셔너리 데이터를 사용
   딕셔너리 key에 해당하는 문자열이 template에서 사용가능한 변수명이 됨
   dot(.)을 사용하여 변수 속성에 접근할 수 있음
   예시 : {{ variable }} / {{ variable.attribute }}

2. Filters
   표시할 변수를 수정할 때 사용 (변수 + | + 필터)
   chained(연결)이 가능하며 일부 필터는 인자를 받기도 함
   예시 : {{ variable|filter }} / {{ name|truncatewords:30 }}

3. Tags
   반복 또는 논리를 수행하여 제어흐름을 만듦
   일부 태그는 시작과 종료 태그가 필요
   예시 : {{ tag }} / {% if %} {% endif %}

4. Comments
   DTL에서의 주석

템플릿 상속(Template inheritance)
페이지의 공통요소를 포함하고, **하위 템플릿이 재정의 할 수 있는 공간**을 정의하는 기본 'skeleton' 템플릿을 작성하여 상속 구조를 구축

'extends' tag
{% extends 'path' %}
자식(하위) 템플릿이 부모 템플릿을 확장한다는 것을 알림
(반드시 자식 템플릿 최상단에 1개만 작성해야 함)

'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)

HTML form (요청과 응답)
HTTP 요청을 서버에 보내는 가장 편리한 방법

'form' element
사용자로부터 할당된 데이터를 서버로 전송
-> 웹에서 사용자 정보를 입력하는 여러 방식(text, password, checkbox 등)을 제공

action & method
form의 핵심 속성 2가지
"데이터를 어디(action)로 어떤 방식(method)으로 요청할지"

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)

'name' attribute
input의 핵심 속성
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 앰퍼샌드(&)로 연결된 key=value 쌍으로 구성되며 기본 URL과는 물음표(?)로 구분됨

예시 : http://host:port/path?key=value&key=value

HTTP 'request' 객체
form으로 전송한 데이터뿐만 아니라 모든 요청 관련 데이터가 담겨 있음 (view함수 첫번째 인자)

BASE_DIR
settings에서 경로지정을 편하게 하기 위해 최상단 지점을 지정해놓은 변수

DTL 주의사항
Python코드로 실행되는 것이 아니며 관련없음 (명칭만 그렇게 설계한 것, if & for 등)
프로그래밍적 로직이 아닌 표현을 위한 것
프로그래밍적 로직은 view 함수에서 작성 및 처리할 것

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

Variable Routing
URL 일부에 변수를 포함시키는 것 (변수는 view 함수의 인자로 전달할 수 있음)

path('articles/<int:num>/', views.detial)
-> <int:num>이 variable routing이다.

Path converters
URL 변수의 타입을 지정 (str, int 등 5가지 타입)

---

3. Model

App URL mapping
각 앱에 URL을 정의하는 것 (프로젝트와 각 앱이 URL을 나누어 관리의 편의를 위함)

include( )
프로젝트 내부 앱들의 URL을 참조할 수 있도록 매핑하는 함수
URL의 일치하는 부분까지 잘라내고, 남은 문자열 부분은 후속 처리를 위해 include된 URL로 전달

Naming URL patterns
URL에 이름을 지정하는 것 (path 함수의 name 인자를 정의해서 사용)
-> html파일에서 a태그가 아닌 url 태그를 사용해서 경로 지정해줘야 함
예시 : path('index/', views.index, name='index')

'url' tag
주어진 URL 패턴의 이름과 일치하는 절대 경로 주소를 반환
예시 : {% url 'url name'%}

URL 이름 지정 후 문제점
다른 APP의 url 이름이 같은 경우, 이름만으로 완벽하게 분리 불가능 -> 이름에 성(key)를 붙이자

'app_name' 속성 지정
각 APP의 urls.py에서 app_name 변수 값 설정한다.
예시 : app_name = 'app' (urlpatterns 위에 작성)

최종 URL tah의 변화
{% url 'articles:index'%} -> ('app name:url name')

Django Model
DB의 테이블을 정의하고 데이터를 조작할 수 있는 기능들을 제공 (테이블 구조를 설계하는 청사진)

Model 클래스

```
class Article(models.Model):
    title = models.CharField(max_length=10)
    content = models.TextField()
```

(models.Model) : models 모듈의 Model 부모 클래스를 상속 받는 다는 뜻
title, content : 클래스 변수명 (테이블 각 '필드(열) 이름')
CharField, TextField : model Field 클래스 (테이블 필드의 '데이터 타입') max_length=10 : model Field 클래스의 키워드 인자 (테이블 필드의 '제약조건' 관련 설정)

제약 조건
데이터가 올바르게 저장되고 관리되도록 하기 위한 규칙

Migrations
model 클래스의 변경사항(필드생성, 수정, 삭제 등)을 DB에 최종 반영하는 방법

Migrations 핵심 명령어
python manage.py makemigrations : model class를 기반으로 최종설계도 작성
python manage.py migrate : 최종 설계도를 DB에 전달하여 반영

Model Field
DB 테이블의 **필드(열)** 을 정의하며, 해당 필드에 저장되는 **데이터 타입**과 **제약조건**을 정의

CharField( ) : 길이 제한이 있는 문자열을 넣을 때 사용
-> max_length : 필드의 최대 길이를 결정하는 필수 인자
TextField( ) : 글자의 수가 많을 때 사용
DateTimeField( ): 날짜&시간 넣을 때 사용
-> auto_now : 데이터가 저장될 때마다 자동으로 현재 날짜&시간을 저장
-> auto_now_add : 데이터가 처음 생성될 때만 자동으로 현재 날짜&시간을 저장

Automatic admin interface
Django는 추가 설치 및 설정 없이 자동으로 관리자 인터페이스를 제공

1. admin 계정 생성
   python manage.py createsuperuser
2. DB에 생성된 admin 계정 확인
3. admin에 모델 클래스 등록
   admin.py에 작성한 모델 클래스를 등록해야 확인 가능
   from .models import Article
   admin.site.register(Article)
4. admin site에서 로그인 후 등록된 모델 클래스 확인

데이터베이스 초기화
migration 파일 삭제 + db.sqlite3 파일 삭제

SQLite
데이터베이스 관리시스템 중 하나, Django의 기본 데이터베이스로 사용

CRUD
소프트웨어가 가지는 기본적인 데이터 처리 기능
Create(저장), Read(조회), Update(수정), Delete(삭제)

---

4. ORM

ORM (Object-Relational-Mapping)
객체 지향 프로그래밍 언어를 사용하여 호환되지 않는 유형의 시스템 간 데이터를 변환하는 기술

ORM의 역할
django와 DB가 사용하는 언어가 다르기 때문에 소통 불가 -> ORM이 중간에서 이를 해석

QuerySet API
ORM에서 데이터를 검색, 필터링, 정렬, 그룹화 하는데 사용하는 도구 (SQL이 아닌 Python 코드로 데이터 처리)

python의 모델 클래스와 인스턴스를 활용해 DB에 데이터를 저장, 조회, 수정, 삭제하는 것

예시 : Article.objects.all( )
Article은 모델 클래스, objects는 Manager, all( )은 Queryset API다. (전체 게시글 주라는 뜻)

Query
데이터베이스에 특정한 데이터를 보여 달라는 요청
쿼리문을 작성한다 = 원하는 데이터를 얻기 위해 DB에 요청을 보낼 코드를 작성한다
파이썬으로 작성한 코드를 ORM이 SQL로 변환하여 DB에 전달, DB의 응답 데이터를 ORM이 QuerySet이라는 자료 형태로 변환하여 우리에게 전달

QuerySet
DB에서 전달 받은 객체 목록 (데이터 모음, 순회가 가능한 데이터로써 1개 이상의 데이터를 불러와 사용가능)
Django ORM을 통해 만들어진 자료형
다만, DB가 단일객체를 반환할 때는 모델(Class)의 인스턴스로 반환됨

Django shell
Django 환경 안에서 실행되는 python shell (입력하는 QuertySet API 구문이 Django 프로젝트에 영향을 미침)

생성 메서드
save( ) : 객체를 데이터베이스에 저장하는 메서드

조회 메서드
all( ) : 전체 데이터 조회
filter( ) : 특정 조건 데이터 조회
get( ) : 단일 데이터 조회
-> 객체를 찾을 수 없으면 DoesNotExist 예외 발생
-> 둘 이상의 객체를 찾으면 MultipleObjectReturned 예외 발생
-> 따라서 primary key와 같이 고유성을 보장하는 조회에서 사용해야 함

데이터 수정

인스턴스 변수를 변경 후 sava 메서드 호출

```
# 수정할 인스턴스 조회
>>> article = Article.objects.get(pk=1)

# 인스턴스 변수를 변경
>>> article.title = 'byebye'

# 저장
>>> article.save()

# 정상적으로 변경된 것을 확인
>>> article.title
'byebye'
```

데이터 삭제

삭제하려는 데이터 조회 후 delete 메서드 호출

```
# 삭제할 인스턴스 조회
>>> article = Article.objects.get(pk=1)

# delete 메서드 호출 (삭제된 객체가 반환)
>>> article.delete()
(1, { 'article.Article':1})
```

---

5. ORM with view

전체 게시글 조회

```python
# articles/views.py

from .models import Article


def index(request):
    articles = Article.objects.all()
    context = {
        'articles': articles,
    }
    return render(request, 'articles/index.html', context)
```

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

<h1>Articles</h1>
<hr>
{% for article in articles %}
  <p>글 번호: {{ article.pk }}</p>
  <p>글 제목: {{ article.title }}</p>
  <p>글 내용: {{ article.content }}</p>
  <hr>
{% endfor %}
```

단일 게시글 조회

```python
# articles/urls.py

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

```python
# articles/views.py

def detail(request, pk):
    article = Article.objects.get(pk=pk)
    context = {
        'article': article,
    }
    return render(request, 'articles/detail.html', context)
```

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

<h2>DETAIL</h2>
<h3>{{ article.pk }} 번째 글</h3>
<hr>
<p>제목: {{ article.title }}</p>
<p>내용: {{ article.content }}</p>
<p>작성일: {{ article.created_at }}</p>
<p>수정일: {{ article.updated_at }}</p>
<hr>
<a href="{% url 'articles:index' %}">[back]</a>
```

Create 로직을 구현하기 위해 필요한 view 함수
new : 사용자 입력 데이터를 받을 페이지를 렌더링
create : 사용자가 입력한 데이터를 받아 DB에 저장

HTTP
네트워크 상에서 데이터를 주고 받기위한 약속

HTTP request methods
데이터(리소스)에 어떤 요청(행동)을 원하는지를 나타내는 것 (GET & POST)

'GET' method
특정 리소스를 조회하는 요청 (데이터를 전달할 때 URL에서 Query String 형식으로 보내짐)
http://127.0.0.1:8000/articles/create/**?title=제목&content=내용**

'POST' method
특정 리소스에 변경(생성, 수정, 삭제)을 요구하는 요청 (데이터를 전달할 때 HTTP Body에 담겨 보내짐)

HTTP response status code
특정 HTTP 요청이 성공적으로 완료되었는지를 3자리 숫자로 표현하기로 약속한 것

403 Forbidden : 서버에 요청 전달됐지만, **권한** 때문에 거절되었다는 뜻

CSRF
Cross-Site-Request-Forgery (사이트 간 요청 위조)

CSRF Token 적용
DTL의 csrf_token 태그를 사용해 손쉽게 사용자에게 토큰 값 부여 가능 ({% csrf_token %})
요청 시 토큰 값도 함께 서버로 전송될 수 있도록 하는 것

CSRF_Token 필요성
Django가 직접 제공한 페이지에서 요청을 보낸 것인지에 대한 확인이 필요하기 때문에
(DB에 영향을 주는 요청에 대해서만)

POST일 때만 Token을 확인하는 이유
POST는 단순 조회를 위한 GET과 달리 특정 리소스에 변경(생성, 수정, 삭제)을 요구하기 때문이다.
DB에 조작을 가하는 요청에는 반드시 인증 수단을 통해 최소한의 신원 확인을 해야한다.

redirect( )
클라이언트가 인자에 작성된 주소로 다시 요청을 보내도록 하는 함수

redirect 특징

```
from django.shortcuts import render, redirect

def create(requeset):
    title = request.POST.get('title')
    content = request.POST.get('content')
    article = Article(title=title, content=content)
    article.save()

    return redirect('articles:detail', article.pk)
```

해당 redirect에서 클라이언트는 detail url로 요청을 다시 보내게 됨
detail view함수가 호출되어 detial 페이지를 응답 받음
결국 사용자는 게시글 작성 후 작성된 게시글의 detail 페이지로 이동하는 것으로 느끼게 된다.

Update 로직을 구현하기 위해 필요한 view 함수
edit : 사용자 입력 데이터를 받을 페이지를 렌더링
update : 사용자가 입력한 데이터를 받아 DB에 저장

GET & POST 차이점

	GET	POST
데이터 전송 방식	URL의 Query string parameter	HTTP body
데이터 크기 제한	브라우저 제공 URL의 최대 길이	제한 없음
사용 목적	데이터 검색 및 조회	데이터 제출 및 조작

GET 요청이 필요한 경우
캐싱 및 성능
GET 요청은 캐시(Cache)될 수 있기에 동일 검색 결과를 여러번 요청하는 경우 더 빠르게 응답 가능
캐시(Cache) : 데이터 & 정보를 임시로 저장해두는 메모리, 디스크 공간 가시성 및 공유
URL에 데이터가 노출되어 있기에 사용자가 북마크하거나 공유하기 용이
RESTful API 설계
HTTP 메서드의 의미에 따라 동작하도록 디자인된 API 일관성 유지 가능

---

6. Django Form

HTML 'form'
사용자로부터 데이터를 받기 위해 활용한 방법 (유효한 데이터인지 확인인지 필터링 불가능)

유효성 검사
수집한 데이터가 정확하고 유효한지 확인하는 과정

Django Form
사용자가 입력 데이터를 수집하고, 처리 및 유효성 검사를 수행하기 위한 도구 (유효성 검사 단순화&자동화 기능을 제공)

Form class 정의

```
from Django import forms

class ArticleForm(forms.Form):
    title = forms.CharField(max_length=10)
    content = forms.CharField()
```

Form class 적용한 new 함수 & html

```
# view의 new 함수
from .forms import ArticleForm

def new(request):
    form = ArticleForm()
    context = {
        'form': form,
    }
    return render(request, 'articles/new.html', context)

# new.html
<h1>NEW</h1>
<form action="{% url 'articles:create' %}" method="POST">
    {% csrt_token %}
    {{ form }}
    <input type="submit">
</form>
```

Form rendering options
label, input 쌍을 특정 HTML 태그로 감싸는 옵션 (예시 : {{ form.as_p }}에서 as_p)

Widgets
HTML 'input' element의 표현을 담당
input 요소의 속성 및 출력되는 부분을 변경하는 것

Form과 ModelForm
Form : 사용자 입력 데이터를 DB에 저장하지 않을 때 (로그인 등)
ModelForm : 사용자 입력 데이터를 DB에 저장해야 할 때 (게시글 작성, 회원가입)

ModelForm
Model과 연결된 Form을 자동으로 생성해주는 기능을 제공 (Form + Model)

```
from django import forms
from .models import Article

class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        fields = '__all__'
```

model : 어느 모델에서 가져올 것인지 정하는 곳
fields : 어느 필드를 사용할 것인지 정하는 곳

Meta class
ModelForm의 정보를 저장하는 곳

fields의 속성
fields = ('필드이름',) : 해당 필드를 포함한다.
fields = '__all__' : 모든 필드를 포함한다.
exclude = ('필드이름',) : 해당 필드만 제외한다.

ModelForm을 적용한 create 로직

```
from .forms import ArticleForm

def create(request):
    form = ArticleForm(request.POST)
    if form.is_valid():
        article = form.save()
        return redirect('articles:detail', article.pk_)
    context = {
        'form': form,
    }
    return render(request, 'articles/new.html', context)
```

is_valid( )
여러 유효성 검사를 실행하고, 데이터가 유효한지 여부를 Boolean으로 반환

위 코드에서 유효해서 True라면 return redirect로 가고, False라면 그에 맞는 에러메세지를 포함해서 context 코드로 넘어간다.
-> 때문에 첫 줄에 정의한 form과 context의 form은 다르다. (context의 form은 에러 메시지를 포함)

save( ) 메서드가 생성과 수정을 구분하는 방법
키워드 인자 instance 여부를 통해 생성&수정을 구분하고 결정
form = ArticleForm(request.POST) : 생성
form = ArticleForm(request.POST, instance=article) : 수정

new & create view함수의 공통점과 차이점
공통점 : 데이터 생성을 구현하기 위함
차이점 : new는 GET 요청만, create는 POST 요청만 처리

```
# 새로운 new + create 함수
def create(request):
    if request.method == 'POST':
        form = ArticleForm(request.POST)
        if form.is_valid():
            article = form.save()
            return redirect('articles:detail', article.pk_)
    else:
        form = ArticleForm()
    context = {
        'form': form,
    }
    return render(request, 'articles/create.html', context)
```

두 함수의 유일한 차이점인 request method에 따른 분기
POST인 경우 : 과거 create 함수 구조였던 객체 생성 및 저장 로직 처리

POST가 아닌 경우 : 단순한 form 인스턴스 생성
urls.py에서 new url 제거, html 및 create view함수에서 new 관련 코드를 create로 변경하기

새로운 update + edit 함수

```
def update(request, pk):
    article = Article.objects.get(pk=pk)
    if request.method == 'POST':
        form = ArticleForm(requeset.POST, instance=article)
        if form.is_valid():
            form.save()
            return redirect('article:detail', article.pk)
    else:
        form = ArticleForm(instance=article)
    context = {
        'article': article,
        'form': form,
    }
    return render(request, 'articles/update.html', context)
```

---

7. Static

Static Files
정적파일, 서버 측에서 변경되지 않고 고정적으로 제공되는 파일 (이미지, JS, CSS 파일 등)

웹 서버와 정적 파일
웹 서버의 기본 동작 : 특정 위치(URL)에 있는 자원을 요청(request) 받아 응답(response)을 처리 및 제공하는 것
이는 '자원에 접근 가능한 주소가 있다'라는 의미
웹 서버는 요청 받은 URL로 서버에 존재하는 정적 자원을 제공함
-> 정적 파일을 제공하기 위한 경로(URL)가 있어야 함

static files 제공
기본 경로 & 추가 경로에서 제공할 수 있다.

기본 경로
articles/static/articles 경로에 이미지 파일 배치
static tag를 사용해 이미지 파일에 대한 경로 제공
{% load static %}
<img src="{% static 'articles/sample-1.png %}" alt="img">

STATIC_URL
기본 & 추가 경로에 위치한 정적 파일을 참조하기 위한 URL (실제 파일&디렉토리가 아닌, URL로만 존재)
settings.py에 STATIC_URL = 'static/'을 작성해야 함

Static files 추가 경로
STATICFILES_DIRS에 문자열 값으로 추가 경로 설정
<img src="{% static '/sample-2.png %}" alt="img">

STATICFILES_DIRS
정적 파일의 기본 경로 외에 추가적인 경로 목록을 정의하는 리스트
settings.py에 STATICFILES_DIRS = [BASE_DIR / 'static',]을 작성해야 함

Media Files
사용자가 웹에서 업로드하는 정적 파일 (user-uploade)

ImageField( )
이미지 업로드에 사용하는 모델 필드 (이미지 객체가 아닌 이미지 파일의 경로가 문자열로 DB에 저장)

미디어 파일 제공을 위한 사전 준비
settings.py에 MEDIA_ROOT, MEDIA_URL 설정
작성한 MEDIA_ROOT, MEDIA_URL에 대한 url 지정

MEDIA_ROOT
실제 미디어 파일들이 위치하는 디렉토리의 절대 경로
MEDIA_ROOT = BASE_DIR / 'media'

MEDIA_URL
MEDIA_ROOT에서 제공되는 미디어 파일에 대한 주소를 생성 (STATIC_URL과 동일한 역할)
MEDIA_URL = 'media/'

MEDIA_ROOT, MEDIA_URL에 대한 url 지정

```
# project.urls.py 파일에 작성

from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('아무거나 paht함수 있다치고')
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

이미지 업로드 과정 3가지
1번 : blank=True 속성을 작성해 빈 문자열이 저장될 수 있게 제약조건 설정
(이미지 없이도 게시글 작성 가능하도록)

```
# articles/models.py에 작성

class Article(models.Model):
    title = models.CharField(max_length=10)
    content = models.TextField()
    image = models.ImageField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    update_at = models.DateTimeFiled(auto_now=True)
```

기존 필드 사이에 작성해도 실제 테이블 작성 시 가장 우측(뒤)에 추가된다.

2번 : html 파일에 form 요소에 enctype 속성 추가
예시 : <form action="{% url 'articles:create' %}" method="POST" enctype="multipart/form-data">

3번 : view 함수에서 업로드 파일에 대한 추가 코드 작성

```
def create(request):
    if request.method == 'POST':
        form = ArticleForm(request.POST, request.FILES)
```

---

8. Authenticaiton system 1

Cookie & Session
우리가 서버로 부터 받은 페이지를 볼 때, 우리는 서버와 서로 연결되어 있는 상태가 아니다.

HTPP
HTML 문서와 같은 리소스들을 가져올 수 있도록 해주는 규약
-> 웹(WWW)에서 이루어지는 모든 데이터 교환의 기초

HTTP 특징

1. 비연결 지향 (connectionless)
   서버는 요청에 대한 응답을 보낸 후 연결을 끊음
2. 무상태 (stateless)
   연결을 끊는 순간 클라이언트와 서버 간의 통신이 끝나며 상태 정보가 유지되지 않음

상태가 없다는 것
예시 : 장바구니에 담은 상품 유지 불가능, 로그인 상태 유지 불가능 -> 상태를 유지하기 위한 기술이 필요

쿠키 (Cookie)
서버가 사용자의 웹 브라우저에 전송하는 작은 데이터 조각
클라이언트 측에 저장되는 작은 데이터 파일, 사용자인증 & 추적 & 상태유지 등에 사용되는 데이터 저장 방식
서버로부터 쿠키를 받고, 같은 서버에 다른 페이지로 재요청시마다 저장해 놓았던 쿠키를 함께 전송

쿠키 사용 원리
브라우저(클라이언트)는 쿠키를 KEY-VALUE의 데이터 형식으로 저장

이렇게 쿠키를 저장해 놓았다가, 동일서버에 재요청 시 저장된 쿠키를 함께 전송

쿠키는 두 요청이 동일한 브라우저에 들어왔는지 아닌지를 판단할 때 주로 사용됨
-> 이를 통해 사용자의 로그인 상태를 유지할 수 있음
-> 상태가 없는(stateless) HTTP 프로토콜에서 상태 정보를 기억 시켜 주기 때문

쿠키 사용 목적

1. 세션 관리 (Session management)
   로그인, 아이디 자동완성, 공지 하루 안 보기, 팝업 체크, 장바구니 등의 정보 관리

2. 개인화 (Personalization)
   사용자 선호, 테마 등의 설정

3. 트래킹 (Tracking)
   사용자 행동을 기록 및 분석

세션 (Session)
서버 측에서 생성되어 클라이언트-서버 간의 상태를 유지

-> 상태 정보를 저장하는 데이터 저장 방식
-> 쿠키에 세션 데이터를 저장하여 매 요청시마다 세션 데이터를 함께 보냄

세션 작동 원리

1. 클라이언트가 로그인을 하면 서버가 session 데이터를 생성 후 저장

2. 생성된 session 데이터에 인증할 수 있는 session id를 발급

3. 발급한 session id를 클라이언트에게 응답

4. 클라이언트는 응답 받은 session id를 쿠키에 저장

5. 클라이언트가 다시 동일 서버에 접속하면 요청과 함께 (sesiion id가 저장된)쿠기를 서버에 전달

6. 쿠키는 요청 시마다 서버에 함께 전송되어 서버에서 session id를 확인해 로그인되어있다는 것을 알게 함

정리
서버 측에서는 세션 데이터 생성 후 저장, 해당 데이터에 접글할 수 있는 세션 ID를 생성
-> 세션 데이터에 '기한' 설정 가능
-> 일정시간이 지나면 자동 로그아웃되는 것 (SWEA)

이 ID를 클라이언트 측으로 전달하고, 클라이언트는 쿠키에 이 ID를 저장

이후 클라이언트가 같은 서버에 재요청 시마다 저장해 두었던 쿠키도 요청과 함께 전송

로그아웃하면 서버가 session 데이터 삭제

예시 : 로그인 상태 유지를 위해 로그인 되었다는 사실을 입증하는 데이터를 매 요청마다 계속해서 보내는 것

쿠키와 세션의 목적
서버와 클라이언트 간의 '상태'를 유지

쿠키 종류별 수명(lifetime)
Session cookie
현재 세션(current session)이 종료되면 삭제됨, 브라우저 종료와 함께 세션이 삭제됨

Persistent cookies
Expires 속성에 지정된 날짜 혹은 Max-Age 속성에 지정된 기간이 지나면 삭제됨

세션 in Django
Django는 'database-backed sessions' 저장 방식을 기본 값으로 사용

session 정보는 DB의 django_session 테이블에 저장됨

Django는 요청 안에 특정 session id를 포함하는 쿠키를 사용해서 각각의 브라우저와 사이트가 연결된 session 데이터를 알아냄

Django는 우리가 session 메커니즘에 대부분을 생가하지 않게 많은 도움을 줌

Authentication System
사용자 인증과 관련된 기능을 모아 놓은 시스템

Authentication (인증)
사용자가 자신이 누구인지 확인하는 것 (신원 확인)

실습 사전준비
두번째 app으로 accounts 생성 및 등록
-> auth와 관련한 경로&키워드들은 django 내부적으로 accounts라는 이름으로 사용하고 있기 때문에
accounts라는 이름을 사용하는 것을 권장

User model 대체하기
django가 기본적으로 제공하는 User name이 아닌 직접 작성한 User model을 사용하기 위해서

User 클래스를 대체하는 이유
지금까지는 별도의 User 클래스 정의 없이 내장된 auth 앱에 작성된 클래스를 사용했음
별도 설정 없이 사용할 수 있어 간편하지만, 개발자가 **직접 수정할 수 없는 문제가 존재**

대체하기

1. AbstractUser 클래스를 상속받는 커스텀 User 클래스 작성
   -기존 User 클래스도 AbstractUser를 상속받기 때문에 이것만 하면
   -커스텀 User 클래스도 기존 User 클래스와 완전히 같은 모습을 가지게 됨

2. django 프로젝트가 사용하는 기본 User 모델을 우리가 작성한 User 모델로 지정
   -수정 전 기본값은 'auth.User'

3. admin site에 대체한 User 모델 등록
   -기본 User 모델이 아니기 때문에 등록하지 않으면 출력되지 않기 때문

AUTH_USER_MODEL
Django 프로젝트의 User를 나타내는 데 사용하는 모델을 지정
※주의사항 : 프로젝트 중간에 AUTH_USER_MODEL 변경 불가능 -> 데이터베이스 초기화 후 진행해야 함

프로젝트 시작할 때 반드시 User 모델을 대체해야 한다
Django는 새 프로젝트를 시작하는 경우 커스텀 User 모델을 설정하는 것을 강력하게 권장

커스텀 User 모델은 기본 User 모델과 동일하게 작동하면서도 필요한 경우 나중에 맞춤 설정할 수 있기 때문

User 모델 대체 작업은 프로젝트의 모든 migrations, 첫 migrate를 실행하기 전에 해야 한다

Login
Session을 Create하는 과정

AuthenticationForm( )
로그인 인증에 사용할 데이터를 입력 받는 built-in form
-> modelform이 아니다!

form과 modelform의 공통점과 차이점
공통점 : 사용자 입력 데이터를 받는 역할
차이점 : form은 받은 데이터를 DB에 저장 안한다. vs modelform은 DB에 저장한다.

login(request, user)
AuthenticationForm을 통해 인증된 사용자를 로그인 하는 함수

get_user( )
AuthenticationForm의 인스턴스 메서드 -> 유효성 검사를 통과했을 경우 로그인 한 사용자 객체를 반환

Logout
Session을 Delete하는 과정

logout(request)
현재 요청에 대한 Session Data를 DB에서 삭제
클라이언트 쿠키에서도 Session Id를 삭제

Template with Authentication data
템플릿에서 인증 관련 데이터를 출력하는 방법

현재 로그인 되어있는 유저 정보 출력하기
user라는 context 데이터를 사용할 수 있는 이유는?
-> django가 미리 준비한 context 데이터가 존재하기 때문 (context processorts)

context processorts
템플릿이 렌더링 될 때 호출 가능한 context 데이터 목록

작성된 context 데이터는 기본적으로 템플릿에서 사용가능한 변수로 포함됨

django에서 자주 사용하는 데이터 목록을 미리 템플릿에 로드해둔 것

'AbstractUser' class
관리자 권한과 함께 완전한 기능을 가지고 있는 User model을 구현하는 추상 기본클래스

몇가지 공통 정보를 여러 다른 모델에 넣을 때 사용하는 클래스

DB 테이블을 만드는데 사용되지 않으며, 대신 다른 모델의 기본 클래스로 사용되는 경우 해당 필드가 하위 클래스의 필드에 추가됨