추가학습 (hashtag 구현하기)

구분	DB

hashtag 구현을 연습하기 위해서

데일리 실습 5-3 파일을 다운 받고 시작을 한다.

db_ws_5_3-master.zip

모델 관계 설정 (M:N)

- Hashtag 모델 작성하고 기존 테이블에 ManyToManyField로 연결한다.

```
# articles/models.py

class Hashtag(models.Model):
    content = models.TextField(unique=True)

    def __str__(self):
        return self.content

class Article(models.Model):
    hashtags = models.ManyToManyField(Hashtag, blank=True)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    title = models.CharField(max_length=10)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.title
```

unique arguments

- `True` 인 경우 이 필드는 테이블 전체에서 고유한 값임을 의미한다.
- 유효성 검사 단계에서 실행되며 중복 데이터를 저장하려고 하면 `.save()` 메서드로 인해 `IntegrityError` 발생한다.

URL 및 view 작성

```
# articles/urls.py

from django.urls import path
from . import views


app_name = 'articles'
urlpatterns = [
    ...,
    path('<int:hash_pk>/hashtag/', views.hashtag, name='hashtag'),
]
```

```
# articles/views.py

from .models import Hashtag

@login_required
```

```
def hashtag(request, hash_pk):
    hashtag = get_object_or_404(Hashtag, pk=hash_pk)
    articles = hashtag.article_set.order_by('-pk')
    context = {
        'hashtag': hashtag,
        'articles': articles,
    }
    return render(request, 'articles/hashtag.html', context)
```

CREATE 함수를 수정

```
@login_required
@require_http_methods(['GET', 'POST'])
def create(request):
    if request.method == 'POST':
        form = ArticleForm(request.POST)
        if form.is_valid():
            article = form.save(commit=False)
            article.user = request.user
            article.save()

            # 최종 저장된 content를 조작하기 위해 article.save()보다 아래에 작성
            for word in article.content.split():  # content를 공백기준 리스트로 변경
                if word.startswith('#'):  # '#' 로 시작하는 요소 선택 후
                    hashtag, created = Hashtag.objects.get_or_create(content=word)
                                        # 'get_or_create()' 아래 설명 참조
                    article.hashtags.add(hashtag)
            return redirect('articles:detail', article.pk)
    else:
        form = ArticleForm()
    context = {
        'form': form,
    }
    return render(request, 'articles/create.html', context)
```

- `#`으로 시작하는 단어를 찾아서 게시글과 Hashtag 모델에 데이터 추가하는데
  - get_or_create 메서드 활용했다. `get_or_create()` 은 모델 객체를 생성할 때 이미 있는 객체라면 가져오고 없으면 새로 생성한다. 그리고
  - (hashtag, created) 튜플의 형태로 저장하는데
    - hashtag 부분에는 검색 또는 생성된 객체가 저장되며
    - created 에는 새 객체 생성 여부를 지정하는 boolean 값이 저장된다.

  `get_or_create` 메서드를 간단히 정리하면 아래의 두 가지 경우로 설명이 가능하다.

  1. (content=word)에 들어간 조건에 해당하는 인스턴스가 존재 한다면 데이터베이스에서 해당 인스턴스를 꺼내와 object에 대입하고, flag는 False가 된다.
  2. 조건에 해당하는 인스턴스가 존재하지 않는다면, 해당 조건을 만족하는 인스턴스를 생성하여 objesct에 대입하고, flag는 True가 된다.

     즉, created(boolean flag)는 `get_or_create` 메서드에 의해 생성이 되면 True, 생성되지 않고 데이터베이스에서 꺼내오면 False가 된다.

- `article.hashtags.add(hashtag)` 는 add()를 활용해서 해시태그 내용을 데이터베이스에 저장 후에 반환 받은 객체를 통해 게시글과 해시태그 사이 관계 생성한다.

- 관계 설정 이후 detail 페이지로 redirect → #이 포함된 형태의 글 작성

Update 함수 수정

- 기존에 있던 hashtag 삭제 후 create와 동일한 작업 수행

```
@login_required
@require_http_methods(['GET', 'POST'])
def update(request, pk):
    article = get_object_or_404(Article, pk=pk)
    if request.user == article.user:
        if request.method == 'POST':
            form = ArticleForm(request.POST, instance=article)
            if form.is_valid():
                form.save()
                article.hashtags.clear()  # 기존에 있던 hashtag 삭제 후 다시 가져오거나 생성
                for word in article.content.split():
                    if word.startswith('#'):
                        hashtag, created = Hashtag.objects.get_or_create(content=word)
                        article.hashtags.add(hashtag)
                return redirect('articles:detail', article.pk)
        else:
            form = ArticleForm(instance=article)
    else:
        return redirect('articles:index')
    context = {
        'article': article,
        'form': form,
    }
    return render(request, 'articles/update.html', context)
```

template 작성

- 해시태그를 클릭했을 때 해당 해시태그를 가진 게시물들만 모아서 보여주는 페이지 작성하기

```


<!-- articles/hashtag.html -->

{% extends 'base.html' %}

{% block content %}
  <div>
    <h2>{{ hashtag.content }}</h2>
    <p>{{ articles|length }}개의 게시글</p>
  </div>

  <hr>

  <div>
    <h2>{{ hashtag.content }}(을)를 태그한 글</h2>
    {% for article in articles %}
      <h3>{{ article.pk }}번 게시글</h3>
      <h3>{{ article.title }}</h3>
      <p>{{ article.comment_set.all|length }}개의 댓글</p>
      <a href="{% url 'articles:detail' article.pk %}">상세글로 바로 가기</a>
      <hr>
    {% endfor %}
  </div>
{% endblock %}
```

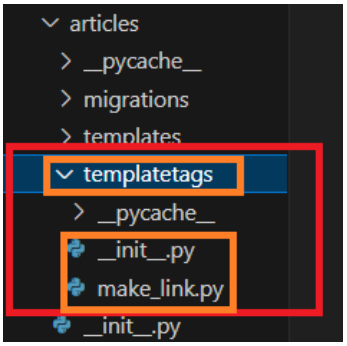
사용자 정의 템플릿 태그

- article의 content 중에서 해시태그에 해당되는 부분만 따로 링크로 연결하기 위해서 custom filter 가 필요하다. 일단 실습을 따라하고 아래 공식문서를 확인해도 좋다.

templatetags 패키지 생성하기

- articles 앱 폴더 내에 templatetag 폴더를 새로 만들고

  `__init__.py`, `make_link.py (사용자 정의 필터 함수를 작성할 파일)` 생성한다.



- `__init__.py`
  - 해당 폴더가 파이썬 패키지라는 것을 명시해주는 것
  - 안에 내용이 없어도 상관 없음

사용자 정의 템플릿 필터 작성하기

- 커스텀 필터의 인자로 들어오는 데이터에서 # 문자와 그 뒤에 오는 문자를 구분하여 반환하는 로직이 담긴 함수 포함하도록 작성할 것이다.
- template 모듈 import 하고 유효한 tag library를 만들기 위한 모듈 레벨의 인스턴스 객체인 register 변수를 작성 한다.

```
# articles/templatetags/make_link.py

from django import template

register = template.Library()
```

함수 정의하기

```
# articles/templatetags/make_link.py

from django import template

register = template.Library()

@register.filter
def hashtag_link(word):
    content = word.content + ' '
    hashtags = word.hashtags.all()
    for hashtag in hashtags:
        content = content.replace(hashtag.content + ' ', f'<a href="/articles/{hashtag.pk}/hashtag/">{hashtag.content}</a> ')
    return content  # 원하는 문자열로 치환이 완료된 content 리턴
```

templatetags 폴더를 추가하고 난 후에는 서버를 재시작 해야 정상적으로 적용된다. 템플릿 태그를 적용해 보자.

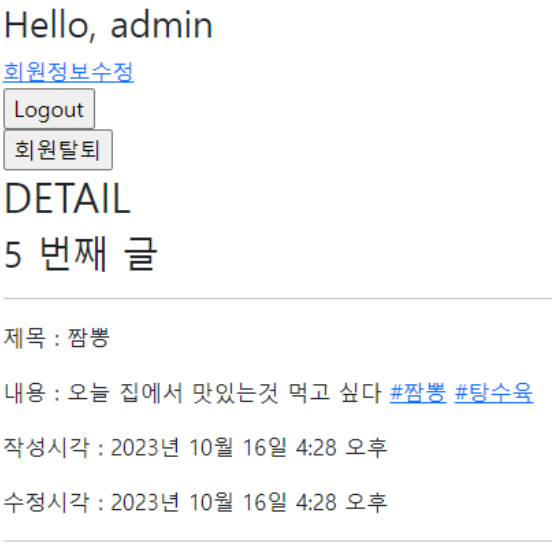
작성한 템플릿 태그 적용하기

- `detail.html`에 load 태그를 통해 직접 작성한 템플릿 태그 불러온다.
  - `{% load make_link %}`
- `|` 를 사용해 필터를 적용하고 safe 필터를 통해 출력 전 추가 HTML Escape가 필요하지 않은 형태의 문자열로 표시
- 해당 링크를 통해 `hashtag.html`로 이동하게 된다.

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

{% extends 'base.html' %}
{% load make_link %}

{% block content %}
  <h2>DETAIL</h2>
  <h3>{{ article.pk }} 번째 글</h3>
  <hr>
  <p>제목 : {{ article.title }}</p>
  <p>내용 : {{ article|hashtag_link|safe }}</p>
  <p>작성시각 : {{ article.created_at }}</p>
  <p>수정시각 : {{ article.updated_at }}</p>
  <hr>
  ...,
{% endblock content %}
```



완성된 파일은 다음과 같다.

hashtag 구현하기.zip