

# Django \_ hashtag

## 1. 해시태그 구현하기

## 1. 해시태그 구현하기

### 해시태그 기능 구현하기(M : N Relationships)

#### 해시태그 기능 구현① - 해시태그 생성

#### 해시태그 기능 구현② - 해시태그 클릭 시 해시태그 상세 페이지로 이동해서 해시태그들 모아보기

#### 해시태그 기능 구현① - 해시태그 생성

해시태그 구현하기 위한 스케레톤 코드는 데일리 실습 4-3에서 주어지는 파일은 베이스로 제작하였다.

`articles/models.py` 에서 Hashtag 모델부터 작성해 보자

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

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

한 게시글 안에서의 해시태그는 중복 없이 하나하나 고유의 값을 가져야 하므로 `unique=True` 옵션을 넣었다. True인 경우 이 필드는 테이블 전체에서 중복 없이 고유한 값이어야 한다.

그리고 Article 클래스에 hashtags 필드를 하나 추가 하자

```
class Article(models.Model):
    hashtags = models.ManyToManyField(Hashtag, blank=True)

    ...

    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
```

모델을 작성하면 서 주의해야 할 점이 있다. 새로 작성한 Hashtag 모델은 Article 모델보다 위에다가 작성을 해 줘야 한다. 그렇지 않으면 Article 모델에 방금 작성한 hashtag 필드가 인식이 되지 않는다.

Article 에 hashtags 필드는 ManyToManyField 를 통해서 `article_id`, `hashtag_id` 를 필드를 갖는

`articles_article_hashtags` 라는 중개모델이 생성된다.

`articles/views.py` 로 이동하자. 해시태그의 생성은 새로 게시글을 작성 할 때 작성 및 생성됨으로 `views.py` 에 `create` 함수 중간에 hashtag 기능 부분을 추가할 것이다.

```
from .models import Article, Comment, Hashtag
...

@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()
        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()
    context = {
        'form': form,
    }
    return render(request, 'articles/create.html', context)

```

스켈레톤 코드에서 추가된 부분은 아래와 같다.

```

from .models import Article, Comment, Hashtag

for word in article.content.split(): # content를 공백기준으로 리스트로 변경하겠다
    if word.startswith('#'):         # '#'으로 시작되는 요소만 선택
        hashtag, created = Hashtag.objects.get_or_create(content=word)
        article.hashtags.add(hashtag) # 해시태그 저장
    return redirect('articles:detail', article.pk)

```

`objects.get_or_create`를 살펴보자. `get_or_create`를 사용하면 이미 객체가 있을 경우에는 `.get`이 실행되어 객체를 저장을 하지 않고 객체가 없을 경우에는 `.create`가 실행되어 객체가 `hashtag`에 저장된다. 그리고 `objects.get_or_create`는 `(hashtag, created)` 형태의 tuple 튜플을 리턴 한다.

`hashtag` 라는 object에는 검색 또는 생성된 객체가 저장되며 `created`에는 새 객체 생성 여부를 지정하는 boolean 값 즉, True, False 형태로 저장된다. 그리고 중요한 사실은 `objects.get_or_create` 메서드는 DB가 가진 인자의 옵션이 `unique` 옵션을 강제하고 있다고 가정하고 실행하기 때문에 아까 Hashtag 모델 작성시 `unique` 옵션을 넣어 준 이유도 있다.

새 글을 작성할 때 해시태그를 만드는 부분을 `create` 함수에 넣어 주었다면 이번에는 작성했던 게시글을 수정(update) 할때 해시 코드를 새로 생성되거나 변경되는 사항을 적용시켜 보자.

`articles/views.py`

```

@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()
                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)

```

위 코드를 살펴보면 기존의 `update` 함수에서 빨간 부분만 추가 해 주었다.

```
# article = form.save() 뒤에 작성합니다.

article.hashtags.clear()
# 게시물 수정시 해당 article에 기존에 있던 hashtag 삭제후

# for문돌려 태그넣기 (for문은 create 작성할 때랑 같은코드)
for word in article.content.split():
    if word.startswith('#'):
        hashtag, created = Hashtag.objects.get_or_create(content=word)
        article.hashtags.add(hashtag)
```

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

## 해시태그 기능 구현② - 해시태그 클릭 시 해시태그 상세 페이지로 이동해서 해시태그들 모아보기

`articles/views.py` 에서 hashtag 함수를 생성하자.

```
@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)
```

`articles/urls.py` 에서 경로도 추가하자

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

그리고 `templates/articles/hashtag.html` 를 생성한 후

detail 페이지에 있는 해시태그 클릭 시 해시태그를 사용한 게시물 들을 모아 볼 수 있는 페이지를 만들어 보자

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

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

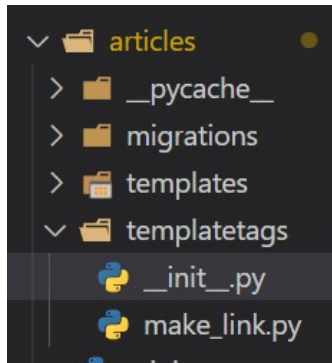
<hr>

<div>
    <h2>{{ hashtag.content }}(을)를 태그한 글</h2>
    {% for article in articles %}
    <h3>{{ article.pk }}번 게시물</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>
</div>
```

```
</div>
{% endblock %}
```

자 이번에는 해시태그를 선택 시 해당 해시태그를 모아보는 페이지로 이동 시킬 것이다.

articles app에 `templatetags` 라는 디렉토리를 새로 생성하자. 그리고 `templatetags` 디렉토리 안에 `__init__.py` `make_link.py` 파일도 생성하자.



그리고 `make_link.py` 파일에 다음과 같이 코드를 작성해 보자.

`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
```

방금 `templatetags` 라는 디렉토리를 새로 생성하였다. 이는 DB에서 넘어오는 data를 재가공 하기 위해서 생성을 한 것이다. Django 에서는 템플릿 에서 “custom filter” 를 사용할 수 있게 제공하는데 이를 사용하기 위해서는 `templatetags` 라는 디렉토리를 생성해야 한다. 이 때, 디렉터리 안에는 무조건 `__init__.py` 가 있어야 하며 `templatetags` 라는 디렉토리 명은 변경하면 안된다.

공식문서의 **[Registering custom filters]** 부분을 살펴보기 바란다.

Django

The web framework for perfectionists with deadlines.

 <https://docs.djangoproject.com/en/3.2/howto/custom-template-tags/#howto-writing-custom-template-filters>

django

이제 `make_link.py` 파일을 만들어 `hashtag_link` 라는 함수에 커스텀 필터를 정의 할 것이다.

아래 코드와 주석을 통해 `make_link.py` 파일의 코드를 확인해 보자.

```
from django import template

register = template.Library() # 기존의 템플릿 라이브러리에

@register.filter                # 커스텀 필터가 추가 된다는 의미의 데코레이터

def hashtag_link(word):
    # word 는 article 객체가 들어오는데
    # article 의 content 들만 가져온다.
    content = word.content + ' '

    # content 중에 모든 해시태그에만 링크를 붙일 것이다.
    hashtags = word.hashtags.all()
    for hashtag in hashtags:
        content = content.replace(hashtag.content + ' ', f'<a href="/articles/{hashtag.pk}/hashtag/">{hashtag.content}</a> ')

    # a 태그 작성시 </a> 뒤에 공백 한 칸 띄어주세요.
    # 그렇지 않으면 게시글에 있는 해시태그가 다 붙어서 보임
    return content
```

`articles/templates/detail.htmls` 에 내용 부분을 수정해 보자.

```
{% 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>작성시간 : {{ article.created_at }}</p>
<p>수정시간 : {{ article.updated_at }}</p>
...
```

먼저 `{% load %}` 템플릿 태그를 사용하였다. `{% load %}` 는 사용자 지정 태그 및 필터를 정의한 것을 확장 사용할 때 사용하는 태그 이다.

`{% load make_link %}` 를 통해서 커스텀 한 템플릿을 로드 한 후에 `<p>내용:` 내용 부분을 수정을 하였다.

`<p>내용:` 부분을 살펴 보면 템플릿 필터 `|` 를 사용해서 변수의 출력 결과를 변경하였다.

여기서 눈에 띄는 것은 바로 `safe` filter 다.

`safe` filter는 무엇이고 왜 써야 하는가?

Django는 자동으로 autoescape 기능이 디폴트 값으로 켜져(on) 있다.그래서 `safe` filter를 이용하여 autoescape 기능을 꺼준 것이다.


그럼 autoescape 기능이 무엇일까? 템플릿 변수 `{{ }}` 를 사용할 때 request객체에 HTML TAG가 포함 되어 있는 경우 HTML TAG가 자동으로 Encoding되어 화면에는 일반 TEXT처럼 보이게 설정 해 놓은 기능이다.

쉽게 예를 들자면 Django에서 객체 안에 `<img>` 태그가 있는 경우 해당 객체를 사용하면 `<img>` 를 문자로 인식해 이미지가 display 되지 않고 `<img>` 라는 글자가 display 된다는 것이다.

## Filters and auto-escaping

Django

The web framework for perfectionists with deadlines.

 <https://docs.djangoproject.com/en/3.2/howto/custom-template-tags/#filters-auto-escaping>

# django

공식문서에서도 확인 되듯이 Django가 이러한 설정을 default로 한 이유는 보안 때문이다. Django가 외부로 부터 들어오는 HTML TAG들을 의심하지 않고 어떠한 입력 값이든 받아들이면 XSS 공격에 취약하기 때문이다. 그래서 안전 한다고 생각하는 입력 값만 태그로 받아들여야 한다. 이 때 장고 내부에서 사용하는 것이 `safe` filter 다. 그래서 결론적으로 `safe` filter 를 사용해서 autoescape 기능을 끈 것이다. autoescape 기능을 끄으로써 우리가 사용한 `<a>`가 정상적으로 작동하기 위함이다.

## Hello, sfminho1

[회원정보수정](#)

Logout

회원탈퇴

## DETAIL

### 2 번째 글

제목 : 운동

내용 : 재미있음 [#좋아요](#) [#행복](#) [#즐거움](#)

작성시각 : 2023년 4월 14일 9:56 오전

수정시각 : 2023년 4월 14일 9:56 오전

[\[UPDATE\]](#)

DELETE

[\[back\]](#)

### 댓글 목록

댓글이 없어요..

Content:

제출

## Hello, sfminho1

[회원정보수정](#)

Logout

회원탈퇴

## #좋아요

2개의 게시글

## #좋아요(을)를 태그한 글

### 2번 게시글

### 운동

0개의 댓글

[상세글로 바로 가기](#)

### 1번 게시글

### 닭갈비

0개의 댓글

[상세글로 바로 가기](#)

이상으로 해시태그 사용법을 살펴보았다. <끝>