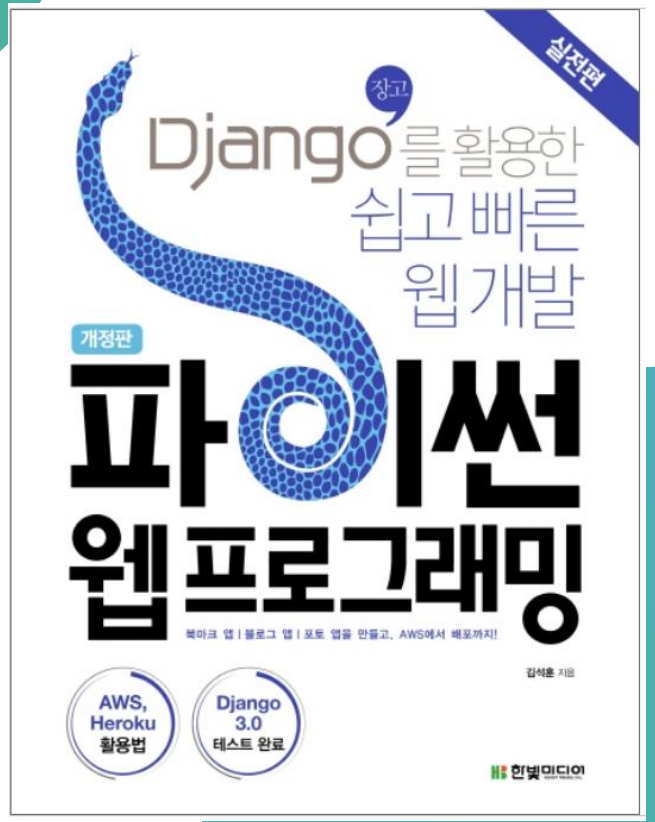
CHAPTER 11

파이썬 웹프로그래밍



가천대학교 컴퓨터공학과
왕보현

CHAPTER 11 실전 프로그램 개발 – 콘텐츠 편집 기능(Bookmark, Blog 앱)

- 콘텐츠에 대한 열람은 모든 사용자가 가능
- 콘텐츠를 새로 생성하는 것은 로그인한 사용자만 가능
- 콘텐츠를 수정 또는 삭제하는 작업은 그 콘텐츠를 생성한 사용자만 가능

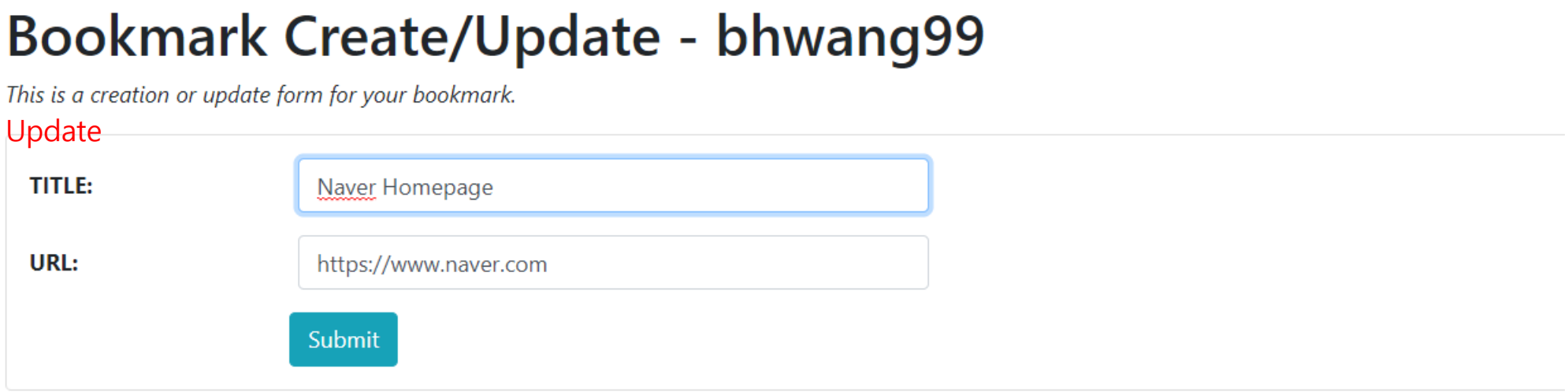
bookmark_form.html 화면

bookmark_change_list.html 화면

Bookmark Change - bhwang99

Title	Url	Owner	Update	Delete
Naver Homepage	https://www.naver.com	bhwang99	Update	Delete
가천대학교	https://www.gachon.ac.kr/main.jsp	bhwang99	Update	Delete

bookmark_form.html 화면



2. URL 설계

URL 설계

URL 패턴	뷰 이름	템플릿 파일명
/bookmark/add/	BookmarkCreateView(CreateView)	bookmark_form.html
/bookmark/change/	BookmarkChangeView(ListView)	bookmark_change_list.html
/bookmark/00/update/	BookmarkUpdateView(UpdateView)	bookmark_form.html
/bookmark/00/delete/	BookmarkDeleteView(DeleteView)	bookmark_confirm_delete.html
/blog/add/	PostCreateView(CreateView)	post_form.html
/blog/change/	PostChangeView(ListView)	post_change_list.html
/blog/00/update/	PostUpdateView(UpdateView)	post_form.html
/blog/00/delete/	PostDeleteView(DeleteView)	post_confirm_delete.html

1. 테이블 설계

ch99₩bookmark₩models.py - 테이블 수정. 기존 bookmark 테이블에 owner 컬럼 추가

```python
from django.db import models
from django.contrib.auth.models import User  # 추가


class Bookmark(models.Model):
    title = models.CharField('TITLE', max_length=100, blank=True)
    url = models.URLField('URL', unique=True)
    owner = models.ForeignKey(User, on_delete=models.CASCADE, blank=True, null=True)  # 추가 ①


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

2장에서 추가한 자료

	id	title	url	owner_id
	필터	필터	필터	필터
1	1	Naver Homepage	https://www.naver.com	NULL
2	2	Daum Homepage	https://www.daum.net	NULL
3	3	Google First Page	https://www.google.co.kr	NULL
4	4	SCI 검색싸이트-test	https://mji.clanvate.com/search-results	1
5	5	장고	https://docs.djangoproject.com/ko/3…	1
6	6	글로벌 가천	http://gachon.ac.kr	3

11장에서 추가한 컬럼

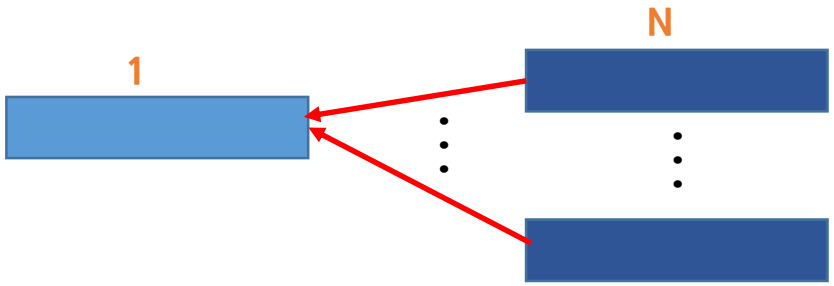
① 로그인한 사람의 id가 입력되는 컬럼 , User Table에 있는 id 값 중 하나가 입력되어야 함.
on_delete=models.CASCADE 는 User Table의 login ID가 삭제가 되면 그 ID로 입력된 bookmark 테이블의 모든 레코드도 삭제하라는 의미

null=True의 이유는 기존 이미 입력된 자료는 owner 컬럼 값이 null이어야 하기 때문
만약 migrate로 컬럼 추가가 안 된다면 테이블의 모든 자료를 지우고 추가 해 볼것

owner 컬럼이 ForeignKey 이므로 테이블에서 _id가 붙어서 컬럼명이 설정 ➔ owner_id

※ ForeignKey에 대해 : 외부 테이블의 키를 참조

User 테이블 : Bookmark 테이블 ➔ 1 : N의 관계



Bookmark (N)

	id	title	url	owner_id
	···	필터	필터	필터
1	1	Naver Homepage	https://www.naver.com	NULL
2	2	Daum Homepage	https://www.daum.net	NULL
3	3	Google First Page	https://www.google.co.kr	NULL
4	4	SCI 검색싸이트-test	https://mjl.clarivate.com/search-results	1
5	5	장고	https://docs.djangoproject.com/ko/3···	1
6	6	글로벌 가천	http://gachon.ac.kr	3

User (1)

테이블(T): auth_user Filter in any column

	id	password	last_login	is_superuser	username	first_name	
	···	필터	필터	필터	필터	필터	필터
1	1	pbkdf2_sha256$180000$bMRSBtjjR3A3···	2020-10-15 08:33:37.254645	1	bhwang99		bhwa
2	2	pbkdf2_sha256$180000$BGliyuaDn0r3$···	NULL	0	test		
3	3	pbkdf2_sha256$180000$8YnJluLdcVM···	2020-10-15 14:54:01.446936	0	testtest		

1. 테이블 설계

ch99₩blog₩models.py

```python
from django.db import models
from django.urls import reverse

from django.contrib.auth.models import User  # 추가
from django.utils.text import slugify  # 추가          ①

class Post(models.Model):
    title = models.CharField(verbose_name='TITLE', max_length=50)
    slug = models.SlugField('SLUG', unique=True, allow_unicode=True, help_text='one word for title alias.')
    description = models.CharField('DESCRIPTION', max_length=100, blank=True, help_text='simple description text.')
    content = models.TextField('CONTENT')
    create_dt = models.DateTimeField('CREATE DATE', auto_now_add=True)
    modify_dt = models.DateTimeField('MODIFY DATE', auto_now=True)
    owner = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name='OWNER', blank=True, null=True)  #추가  ②

     class Meta:
        verbose_name = 'post'
        verbose_name_plural = 'posts'
        db_table = 'blog_posts'
        ordering = ('-modify_dt',)

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse('blog:post_detail', args=(self.slug,))

    def get_previous(self):
        return self.get_previous_by_modify_dt()

    def get_next(self):
        return self.get_next_by_modify_dt()

    def save(self, *args, **kwargs):   # save함수 추가       ③
          self.slug = slugify(self.title, allow_unicode=True)
          super().save(*args, **kwargs)
```

1. 테이블 설계

ch99₩blog₩models.py

```
from django.utils.text import slugify  # 추가  ①
```

① slug 필드를 자동으로 채우기 위해 slugify() 함수를 임포트. slugify() 함수는 원래 단어를 알파벳 소문자, 숫자, 밑줄, 하이픈으로만 구성된 단어로 만들어주는 함수. 예를 들어 slugify("Django is a Python Web Framework")를 실행하면 결과는 "django-is-a-python-web-framework"가 됨.

```
owner = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name='OWNER', blank=True, null=True)  #추가      ②
```

② Post와 User 테이블 사이는 N:1 관계. 장고에서 N:1 관계는 외래 키로 표현. owner 필드는 NULL 값을 가질 수 있어야 함. Post 테이블에 레콘가 존재하는 상태에서 지금 owner 필드를 추가하면, 기존 레코드의 owner 필드는 NULL 값으로 채워야 하기 때문. CASCADE 옵션은 User 테이블에서 레코드가 삭제되는 경우 그 레코드에 연결된 Post 테이블의 레코드도 같이 삭제됨을 의미

1. 테이블 설계

ch99₩blog₩models.py

```
def save(self, *args, **kwargs):   # save함수 추가     ③
        self.slug = slugify(self.title, allow_unicode=True)
        super().save(*args, **kwargs)
```

③ save() 메소드를 재정의. save() 메소드는 모델 객체의 내용을 데이터베이스에 저장하는 메소드. 데이터베이스 테이블에 저장 시 slug 필드를 title 필드로부터 만들어 자동으로 채워줌. allow_unicode=True 옵션을 주면, 한글 처리도 가능
super().save() 는 부모 클래스의 save() 메소드를 호출해 객체의 내용을 테이블에 반영하는 save() 메소드의 원래 기능을 수행

※ migrate 한 후에도 DB 테이블의 컬럼이 추가 되지 않는 경우

DB browser for SQLite 를 닫고 다시 실행하기. 그래도 추가 되지 않는다면

Post 테이블과 Bookmark 테이블의 내용을 모두 지운 후
ch99 폴더에서 python manage.py makemigrations 명령과 python manage.py migrate 명령을 실행 할 것.

2. URL 설계

ch99₩bookmark₩urls.py

```
from django.urls import path
# from bookmark.views import BookmarkLV, BookmarkDV    #삭제
from bookmark import views   # 추가          ①



app_name = 'bookmark'
urlpatterns = [
    path('', views.BookmarkLV.as_view(), name='index'),  #수정
    path('<int:pk>/', views.BookmarkDV.as_view(), name='detail'),  #수정

    #하단 내용 추가
    path('add/', views.BookmarkCreateView.as_view(), name="add"),
    path('change/', views.BookmarkChangeLV.as_view(), name="change"),
    path('<int:pk>/update/', views.BookmarkUpdateView.as_view(), name="update"),
    path('<int:pk>/delete/', views.BookmarkDeleteView.as_view(), name="delete"),
]
```

① view가 많을 때는 views를 import 하여 내부 뷰 클래스를 사용

2. URL 설계

ch99₩blog₩urls.py

```python
from django.urls import path, re_path
from blog import views


app_name = 'blog'
urlpatterns = [

    # Example: /blog/
    path('', views.PostLV.as_view(), name='index'),

    # Example: /blog/post/ (same as /blog/)
    path('post/', views.PostLV.as_view(), name='post_list'),

    # Example: /blog/post/django-example/
    re_path(r'^post/(?P<slug>[-\w]+)/$', views.PostDV.as_view(), name='post_detail'),

    # Example: /blog/archive/
    path('archive/', views.PostAV.as_view(), name='post_archive'),

    # Example: /blog/archive/2019/
    path('archive/<int:year>/', views.PostYAV.as_view(), name='post_year_archive'),

    # Example: /blog/archive/2019/nov/
    path('archive/<int:year>/<str:month>/', views.PostMAV.as_view(), name='post_month_archive'),

    # Example: /blog/archive/2019/nov/10/
    path('archive/<int:year>/<str:month>/<int:day>/', views.PostDAV.as_view(), name='post_day_archive'),

    # Example: /blog/archive/today/
    path('archive/today/', views.PostTAV.as_view(), name='post_today_archive'),

    # Example: /blog/search/
    path('search/', views.SearchFormView.as_view(), name='search'),

    #하단 내용 추가
    path('add/', views.PostCreateView.as_view(), name="add"),
    path('change/', views.PostChangeLV.as_view(), name="change"),
    path('<int:pk>/update/', views.PostUpdateView.as_view(), name="update"),
    path('<int:pk>/delete/', views.PostDeleteView.as_view(), name="delete"),
]
```

3. View 설계

ch99₩bookmark₩views.py

```python
from django.views.generic import ListView, DetailView
from bookmark.models import Bookmark

# 하단 4개의 import 문장 추가
from django.views.generic import CreateView, UpdateView, DeleteView    ①
from django.contrib.auth.mixins import LoginRequiredMixin              ②
from django.urls import reverse_lazy                                   ③
from mysite.views import OwnerOnlyMixin                                ④

class BookmarkLV(ListView):
    model = Bookmark

class BookmarkDV(DetailView):
    model = Bookmark

# 하단 4개의 클래스형 뷰 추가
class BookmarkCreateView(LoginRequiredMixin, CreateView):              ⑤
    model = Bookmark                                                   ⑥
    fields = ['title', 'url']                                          ⑦
    success_url = reverse_lazy('bookmark:index')                       ⑧

    def form_valid(self, form):                                        ⑨
        form.instance.owner = self.request.user                        ⑩
        return super().form_valid(form)                                ⑪

class BookmarkChangeLV(LoginRequiredMixin, ListView):                  ⑫
    template_name = 'bookmark/bookmark_change_list.html'               ⑬

    def get_queryset(self):                                            ⑭
        return Bookmark.objects.filter(owner=self.request.user)

class BookmarkUpdateView(OwnerOnlyMixin, UpdateView):                  ⑮
    model = Bookmark                                                   ⑯
    fields = ['title', 'url']                                          ⑰
    success_url = reverse_lazy('bookmark:index')                       ⑱

class BookmarkDeleteView(OwnerOnlyMixin, DeleteView):                  ⑲
    model = Bookmark                                                   ⑳
    success_url = reverse_lazy('bookmark:index')                       ㉑
```

3. View 설계

ch99₩bookmark₩views.py

```
# 하단 4개의 import 문장 추가
from django.views.generic import CreateView, UpdateView, DeleteView   ①
from django.contib.auth.mixins import LoginRequiredMixin    ②
from django.urls import reverse_lazy            ③
from mysite.views import OwnerOnlyMixin     ④
```

① 편집용 제네릭 뷰인 CreateView, UpdateView, DeleteView를 임포트. CreateView는 테이블의 레코드를 생성할 때, UpdateView는 테이블의 레코드를 수정할 때, DeleteView는 테이블의 레코드를 삭제할 때 사용하는 뷰

② LoginRequiredMixin 클래스는 @login_required() 데코레이터 기능을 클래스에 적용할 때 사용. 즉, 사용자가 로그인 된 경우는 정상 처리를 하지만, 로그인이 안 된 사용자라면 로그인 페이지로 리다이렉트시킴

③ reverse_lazy() 및 reverse() 함수는 URL 패턴명을 인자로 받음. URL 패턴명을 인식하기 위해서는 urls.py 모듈이 메모리에 로딩되어야 함. 지금 작성하고 있는 views.py 모듈이 로딩되고 처리되는 시점에 urls.py 모듈이 로딩되지 않을 수도 있으므로, reverse() 함수 대신 reverse_lazy() 함수를 임포트

④ OwnerOnlyMixin 클래스를 임포트. 소유자만 콘텐츠 수정이 가능하도록 이 믹스인 클래스를 사용함.
mysite/views.py 설명 참고

3. View 설계

ch99₩bookmark₩views.py

```
# 하단 4개의 클래스형 뷰 추가
class BookmarkCreateView(LoginRequiredMixin, CreateView):          ⑤
    model = Bookmark                                               ⑥
    fields = ['title', 'url']                                      ⑦
    success_url = reverse_lazy('bookmark:index')                   ⑧

    def form_valid(self, form):                                    ⑨
        form.instance.owner = self.request.user                    ⑩
        return super().form_valid(form)                            ⑪
```

⑤ LoginRequiredMixin 및 CreateView를 상속받아 BookmarkCreateView 뷰를 작성. LoginRequiredMixin 클래스를 상속받는 클래스는 로그인 된 경우만 접근 가능. 만일 로그인되지 않은 상태에서 BookmarkCreateView 뷰를 호출하면 로그인 페이지로 이동시킴. CreateView 클래스를 상속받는 클래스는 예제처럼 중요한 몇 가지 클래스 속성만 정의하면 적절한 폼을 보여주고, 폼에 입력된 내용에서 에러 여부를 체크한 후 에러가 없으면 입력된 내용으로 테이블에 레코드를 생성함.

⑥ CreateView 기능을 적용할 대상 테이블을 Bookmark 테이블로 지정

⑦ CreateView 기능에 따라 폼을 보여줄 때, Bookmark 테이블의 title과 url 필드에 대한 입력 폼을 보여줌

⑧ 폼에 입력된 내용에 에러가 없고 테이블 레코드 생성이 완료된 후에 이동할 URL을 지정. 예제에서는 /bookmark/ URL로 이동

⑨ CreateView는 폼에 입력된 내용에 대해 유효성 검사를 수행해 에러가 없는 경우 form_valid() 메소드를 호출 함. 또한 유효성 검사를 통과하면 모델 instance(객체, 레코드)를 생성

⑩ 폼에 연결된 모델 객체의 owner 필드에는 현재 로그인된 사용자의 User 객체를 할당함

⑪ 부모 클래스, 즉 CreateView 클래스의 form_valid() 메소드를 호출. 상위 클래스의 form_valid() 메소드에 의해 form.save(), 즉 DB에 반영되고 그 후 success_url로 리다이렉트 됨.

3. View 설계

ch99₩bookmark₩views.py

```
class BookmarkChangeLV(LoginRequiredMixin, ListView):          ⑫
    template_name = 'bookmark/bookmark_change_list.html'       ⑬

    def get_queryset(self):                                    ⑭
        return Bookmark.objects.filter(owner=self.request.user)
```

⑫ LoginRequiredMixin 및 ListView를 상속받아 BookmarkChangeLV 뷰를 작성. BookmarkChangeLV 뷰의 기능은 Bookmark 테이블에서 현재 로그인 된 사용자에게 콘텐츠 변경이 허용된 레코드 리스트를 화면에 출력하는 것. 이 클래스도 LoginRequiredMixin 클래스를 상속받고 있어서, login_required() 데코레이터 기능을 제공함. 그리고 ListView 제네릭 뷰를 상속받고 있으므로 객체의 리스트만 지정하면 그 리스트를 화면에 출력해 줌

⑬ 리스트를 화면에 출력할 때 사용할 템플릿의 이름을 지정해 줌

⑭ get_queryset() 메소드는 화면에 출력할 레코드 리스트를 반환. 즉 Bookmark 테이블의 레코드 중에서 owner 필드가 로그인한 사용자인 레코드만 필터링해 그 리스트를 반환함. 이 줄에 의해 로그인한 사용자가 소유한 콘텐츠만 보이게 됨

3. View 설계

ch99₩bookmark₩views.py

```
class BookmarkUpdateView(OwnerOnlyMixin, UpdateView):      ⑮
    model = Bookmark                                       ⑯
    fields = ['title', 'url']                              ⑰
    success_url = reverse_lazy('bookmark:index')           ⑱
```

⑮ OwnerOnlyMixin및 UpdateView를 상속받아 BookmarkUpdateView 뷰를 작성.

OwnerOnlyMixin 클래스에 의해 로그인 사용자가 대상 콘텐츠의 소유자인 경우에만 UpdateView 기능이 동작.
즉, 로그인을 하지 않은 경우 또는 로그인했어도 소유자가 아닌 경우는 익셉션 처리를 함. mysite/views.py 참조

⑯ UpdatViw 기능을 적용할 대상 테이블을 Bookmark 테이블로 지정

⑰ UpdateView 기능에 따라 폼을 보여줄 때 Bookmark 테이블의 특정 레코드를 선택하고 그 레코드의 title과 url 필드로 폼을 구성해 화면에 보여줌

⑱ 폼에 수정 입력된 내용에 에러가 없으면 UpdateView는 내부적으로 form_valid() 메소드를 호출하여 테이블의 레코드를 수정하고 success_url로 지정된 URL로 리다이렉트 처리. 예제에서는 /bookmark/ 로 리다리엑트

UpdateView 클래스를 상속받는 클래스는 예제처럼 중요한 몇 가지 클래스 속성만 정의하면, 테이블의 레코드들 중에서 지정된 레코드 하나에 대한 내용을 폼으로 보여주고 폼에서 수정 입력된 내용에서 에러 여부를 체크하여 에러가 없으면 입력된 내용으로 테이블의 레코드를 수정

Bookmark Change - bhwang99

Title	Url	Owner	Update	Delete
Naver Homepage	https://www.naver.com	bhwang99	Update	Delete
가천대학교	https://www.gachon.ac.kr/main.jsp	bhwang99	Update	Delete

```
<td><a href="{% url 'bookmark:update' item.id %}">Update</a></td>
```

insert bookmark values(....)
where bookmakr.id = id

BookmarkUpdateView.as_view()

/bookmark/id/update/

```
path('<int:pk>/update/', views.BookmarkUpdateView.as_view(), name="update"),
```

3. View 설계

ch99₩bookmark₩views.py

```
class BookmarkDeleteView(OwnerOnlyMixin, DeleteView):      ⑲
    model = Bookmark                                       ⑳
    success_url = reverse_lazy('bookmark:index')           ㉑
```

⑲ OwnerOnlyMixin및 DeleteView를 상속받아 BookmarkDeleteView 뷰를 작성. OwnerOnlyMixin 클래스에 의해 로그인 사용자가 대상 콘텐츠의 소유자인 경우에만 DeleteView 기능이 동작. 즉, 로그인을 하지 않은 경우 또는 로그인했어도 소유자가 아닌 경우는 익셉션 처리를 함. mysite/views.py 참조.

DeleteView 클래스를 상속받는 클래스는 예제처럼 중요한 몇 가지 클래스 속성만 정의하면 기존 레코드 중에서 지정된 레코드를 삭제할 것인지 확인하는 페이지를 보여줌. 사용자가 확인 응답을 하면 해당 레코드를 삭제함.

⑳ DeleteView 기능을 적용할 대상 테이블을 Bookmark 테이블로 지정함.

㉑ 테이블 레코드 삭제가 완료된 후 리다이렉트될 URL을 지정. 예제에서는 /bookmark/로 리다이렉트 처리됨

Bookmark Change - bhwang99

Title	Url	Owner	Update	Delete
Naver Homepage	https://www.naver.com	bhwang99	Update	Delete
가천대학교	https://www.gachon.ac.kr/main.jsp	bhwang99	Update	Delete

```
<td><a href="{% url 'bookmark:delete' item.id %}">Delete</a></td>
```

delete bookmark
where bookmakr.id = id

BookmarkDeleteView.as_view()

/bookmark/id/update/

```
path('<int:pk>/delete/', views.BookmarkDeleteView.as_view(), name="update"),
```

3. View 설계

ch99₩blog₩views.py

```
from django.views.generic import ListView, DetailView, TemplateView
from django.views.generic import ArchiveIndexView, YearArchiveView, MonthArchiveView
from django.views.generic import DayArchiveView, TodayArchiveView
from django.views.generic import FormView
from django.db.models import Q
from django.shortcuts import render

from blog.models import Post
from blog.forms import PostSearchForm

# 하단 4개의 import 문장 추가
from django.views.generic import CreateView, UpdateView, DeleteView
from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from mysite.views import OwnerOnlyMixin


#--- ListView
class PostLV(ListView):
    model = Post
    template_name = 'blog/post_all.html'
    context_object_name = 'posts'
    paginate_by = 2


#--- DetailView
class PostDV(DetailView):
    model = Post
```

```
#--- ArchiveView
class PostAV(ArchiveIndexView):
    model = Post
    date_field = 'modify_dt'


class PostYAV(YearArchiveView):
    model = Post
    date_field = 'modify_dt'
    make_object_list = True


class PostMAV(MonthArchiveView):
    model = Post
    date_field = 'modify_dt'


class PostDAV(DayArchiveView):
    model = Post
    date_field = 'modify_dt'


class PostTAV(TodayArchiveView):
    model = Post
    date_field = 'modify_dt'
```

3. View 설계

ch99₩blog₩views.py

```
#--- FormView
class SearchFormView(FormView):
    form_class = PostSearchForm
    template_name = 'blog/post_search.html'

    def form_valid(self, form):
        searchWord = form.cleaned_data['search_word']
        post_list = Post.objects.filter(Q(title__icontains=searchWord) |  Q(description__icontains=searchWord) |
Q(content__icontains=searchWord)).distinct()

        context = {}
        context['form'] = form
        context['search_term'] = searchWord
        context['object_list'] = post_list

        return render(self.request, self.template_name, context)   # No Redirection

# 하단 4개의 클래스형 뷰 추가
class PostCreateView(LoginRequiredMixin, CreateView):
    model = Post
    fields = ['title', 'slug', 'description', 'content']
    initial = {'slug':'auto-filling-do-not-input'}                  ①
    #fields = ['title', 'description', 'content']                   ②
    success_url = reverse_lazy('blog:index')

    def form_valid(self, form):
        form.instance.owner = self.request.user
        return super().form_valid(form)
```

```
class PostChangeLV(LoginRequiredMixin, ListView):
    template_name = 'blog/post_change_list.html'

    def get_queryset(self):
        return Post.objects.filter(owner=self.request.user)


class PostUpdateView(OwnerOnlyMixin, UpdateView):
    model = Post
    fields = ['title', 'slug', 'description', 'content']
    success_url = reverse_lazy('blog:index')


class PostDeleteView(OwnerOnlyMixin, DeleteView):
    model = Post
    success_url = reverse_lazy('blog:index')
```

① 폼의 slug 입력 항목에 대한 초기값을 지정. slug 필드는 title 필드로부터 자동으로 채워지는 필드. 이 기능은 models.py 파일의 Post 모델 정의에 있는 save() 함수에서 수행됨. 따라서 PostCreateView 뷰에서 레코드 생성 폼을 보여줄 때 slug 필드는 입력하지 말라는 의미로 초기값을 "auto-filling-do-not-input"으로 지정

② slug 필드를 처리하는 또 다른 방법은 fields 속성에서 제외해 폼에 나타나지 않도록 하는 방법임. 폼에는 보이지 않지만 Post 모델의 save() 함수에 의해 테이블의 레코드에 자동으로 채워짐

3. View 설계

ch99₩mysite₩views.py

```
from django.views.generic import TemplateView
from django.views.generic import CreateView
from django.contrib.auth.forms import UserCreationForm
from django.urls import reverse_lazy

from django.contrib.auth.mixins import AccessMixin   #추가

#--- TemplateView
class HomeView(TemplateView):
    template_name = 'home.html'

#--- User Creation
class UserCreateView(CreateView):
    template_name = 'registration/register.html'
    form_class = UserCreationForm
    success_url = reverse_lazy('register_done')


class UserCreateDoneTV(TemplateView):
    template_name = 'registration/register_done.html'

# 하단 코드 추가
class OwnerOnlyMixin(AccessMixin):                                  ②
    raise_exception = True                                          ③
    permission_denied_message = "Owner only can update/delete the object"  ④

def dispatch(self, request, *args, **kwargs):                       ⑤
    obj = self.get_object()                                         ⑥
    if request.user != obj.owner:                                   ⑦
        return self.handle_no_permission()                          ⑧
    return super().dispatch(request, *args, **kwargs)               ⑨
```

① 뷰 처리 진입 단계에서 적절한 권한을 갖추었는지 판별할 때 사용하는 믹스인 클래스

mixin: 다중 상속을 위한 메커니즘

3. View 설계

ch99₩mysite₩views.py

```
# 하단 코드 추가
class OwnerOnlyMixin(AccessMixin):                                    ②
    raise_exception = True                                            ③
    permission_denied_message = "Owner only can update/delete the object"        ④
```

② OwnerOnlyMixin 클래스를 정의. 로그인한 사용자가 콘텐츠의 소유자인지를 판별. 소유자면 정상처리를 하고, 소유자가 아니면 ③처럼 두 가지 방법으로 처리

③ 소유자가 아닌 경우 이 속성이 True 이면 403 익셉션 처리를 하고 False 이면 로그인 페이지로 리다이렉트 처리됨 예제는 403 익셉션 처리를 하게 되어 있음

④ 403 응답 시 보여줄 메시지를 지정함. 403.html 템플릿 파일에서 사용함.

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

{% block title %}403.html{% endblock %}

{% block content %}

    <h1>Permission Denied (403)</h1>
    <br>

    <div class="alert alert-danger">
        <div class="font-weight-bold">{{ exception }}</div>
    </div>

{% endblock content %}
```

3. View 설계

ch99₩mysite₩views.py

```
def dispatch(self, request, *args, **kwargs):                    ⑤
    obj = self.get_object()                                      ⑥
    if request.user != obj.owner:                                ⑦
        return self.handle_no_permission()
    return super().dispatch(request, *args, **kwargs)            ⑧
```

⑤ 메인 메소드인 get() 처리 이전 단계의 dispatch() 메소드를 오버라이딩 함. 여기서 소유자 여부를 판단함.

⑥ 대상이 되는 객체를 테이블로부터 가져옴. update 하고자 하는 object

⑦ 현재의 사용자(request.user)와 객체의 소유자(obj.owner)를 비교해서 다르면 handle_no_permission() 메소드를 호출. 이 메소드에 의해 403 익셉션 처리, 즉 클라이언트에게 403 응답(HttpResponseForbidden)을 보냄.

⑧ 같으면 상위 클래스의 dispatch() 메소드를 호출해서 정상 처리함.

3. View 설계

ch99₩mysite₩views.py

Title	Url	Owner	Update	Delete
SCI 검색싸이트	https://mjl.clarivate.com/search-results	bhwang99	Update	Delete

```
path('<int:pk>/update/', views.BookmarkUpdateView.as_view(), name="update"),
```

```
class BookmarkUpdateView(OwnerOnlyMixin, UpdateView):
    model = Bookmark
    fields = ['title', 'url']
    success_url = reverse_lazy('bookmark:index')
```

```
class OwnerOnlyMixin(AccessMixin):
    raise_exception = True
    permission_denied_message = "Owner only can update/delete the object"
```

```
def dispatch(self, request, *args, **kwargs):
    obj = self.get_object()    #update 하고자 하는 object
    if request.user != obj.owner:
        return self.handle_no_permission()
    return super().dispatch(request, *args, **kwargs)
```

4. 템플릿 코딩

ch99₩mysite₩base.html

```
<li class="nav-item dropdown mx-1 btn btn-primary">
    <a class="nav-link dropdown-toggle text-white" href="#" data-toggle="dropdown">Util</a>
    <div class="dropdown-menu">
        <a class="dropdown-item" href="{% url 'admin:index' %}">Admin</a>
        <div class="dropdown-divider"></div>
        <a class="dropdown-item" href="{% url 'blog:post_archive' %}">Archive</a>
        <a class="dropdown-item" href="{% url 'blog:search' %}">Search</a>
    </div>
</li>

#하단 코드 추가
 <li class="nav-item dropdown mx-1 btn btn-primary">
    <a class="nav-link dropdown-toggle text-white" href="#" data-toggle="dropdown">Add</a>
    <div class="dropdown-menu">
        <a class="dropdown-item" href="{% url 'bookmark:add' %}">Bookmark</a>
        <a class="dropdown-item" href="{% url 'blog:add' %}">Post</a>
        <div class="dropdown-divider"></div>
        <a class="dropdown-item" href="">Album</a>
        <a class="dropdown-item" href="">Photo</a>
    </div>
</li>

 <li class="nav-item dropdown mx-1 btn btn-primary">
    <a class="nav-link dropdown-toggle text-white" href="#" data-toggle="dropdown">Change</a>
    <div class="dropdown-menu">
        <a class="dropdown-item" href="{% url 'bookmark:change' %}">Bookmark</a>
        <a class="dropdown-item" href="{% url 'blog:change' %}">Post</a>
        <div class="dropdown-divider"></div>
        <a class="dropdown-item" href="">Album</a>
        <a class="dropdown-item" href="">Photo</a>
    </div>
</li>
```

4. 템플릿 코딩

ch99₩bookmark₩templates₩bookmark₩bookmark_form.html

```
{% extends "base.html" %}
{% load widget_tweaks %}

{% block title %}bookmark_form.html{% endblock %}

{% block content %}

    <h1>Bookmark Create/Update - {{user}}</h1>
    <p class="font-italic">This is a creation or update form for your bookmark.</p>

    {% if form.errors %}
    <div class="alert alert-danger">
        <div class="font-weight-bold">Wrong! Please correct the error(s) below.</div>
        {{ form.errors }}
    </div>
    {% endif %}

    <form action="." method="post" class="card pt-3">{% csrf_token %}

        <div class="form-group row">
            {{ form.title|add_label_class:"col-form-label col-sm-2 ml-3 font-weight-bold" }}
            <div class="col-sm-5">
                {{ form.title|add_class:"form-control"|attr:"autofocus" }}
            </div>
        </div>

        <div class="form-group row">
            {{ form.url|add_label_class:"col-form-label col-sm-2 ml-3 font-weight-bold" }}
            <div class="col-sm-5">
                {{ form.url|add_class:"form-control" }}
            </div>
        </div>

        <div class="form-group">
            <div class="offset-sm-2 col-sm-5">
                <input type="submit" value="Submit" class="btn btn-info"/>
            </div>
        </div>

    </form>

{% endblock %}
```

※ CreateView를 상속받는 뷰클래스는 디폴트로 model명소문자_form.html을 호출

① {{user}} 는 django.contrib.auth 앱에서 제공하는 컨텍스트 변수로 현재의 User 객체가 저장

※ 여기에서 사용하는 form은 CreateView와 UpdateView가 스스로 만든 ModelForm 임.

4. 템플릿 코딩

ch99₩bookmark₩templates₩bookmark₩bookmark_change_list.html

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

{% block title %}bookmark_change_list.html{% endblock %}

{% block content %}

  <h1>Bookmark Change - {{user}}</h1>
  <br>

  <table class="table table-bordered table-condensed table-striped">

    <thead>
      <tr class="table-primary">
        <th>Title</th>
        <th>Url</th>
        <th>Owner</th>
        <th>Update</th>
        <th>Delete</th>
      </tr>
    </thead>

    <tbody>
      {% for item in object_list %}
      <tr>
        <td>{{ item.title }}</td>
        <td>{{ item.url }}</td>
        <td>{{ item.owner }}</td>
        <td><a href="{% url 'bookmark:update' item.id %}">Update</a></td>
        <td><a href="{% url 'bookmark:delete' item.id %}">Delete</a></td>
      </tr>
      {% endfor %}
    </tbody>

  </table>

{% endblock %}
```

① {{user}} 는 django.contrib.auth 앱에서 제공하는 컨텍스트 변수로 현재의 User 객체가 저장됨

②
③
④
⑤
⑥

Bookmark Change - bhwang99

Title	Url	Owner	Update	Delete
Naver Homepage	https://www.naver.com	bhwang99	Update	Delete
가천대학교	https://www.gachon.ac.kr/main.jsp	bhwang99	Update	Delete

4. 템플릿 코딩

ch99₩bookmark₩templates₩bookmark₩bookmark_confirm_delete.html

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

{% block title %}bookmark_confirm_delete.html{% endblock %}

{% block content %}

    <h1>Bookmark Delete</h1>
    <br>

    <form action="." method="post">{% csrf_token %}
        <p>Are you sure you want to delete "{{ object }}" ?</p>
        <input type="submit" value="Confirm" class="btn btn-danger btn-sm" />
    </form>

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

4. 템플릿 코딩

ch99₩blog₩templates₩blog₩post_form.html

```
{% extends "base.html" %}
{% load widget_tweaks %}

{% block title %}post_form.html{% endblock %}

{% block content %}
    <h1>Post Create/Update - {{user}}</h1>
    <p class="font-italic">This is a creation or update form for your post.</p>

    {% if form.errors %}
    <div class="alert alert-danger">
        <div class="font-weight-bold">Wrong! Please correct the error(s) below.</div>
        {{ form.errors }}
    </div>
    {% endif %}

    <form action="." method="post" class="card pt-3">{% csrf_token %}

        <div class="form-group row">
            {{ form.title|add_label_class:"col-form-label col-sm-2 ml-3 font-weight-bold" }}
            <div class="col-sm-5">
                {{ form.title|add_class:"form-control"|attr:"autofocus" }}
            </div>
        </div>
```

4. 템플릿 코딩

ch99₩blog₩templates₩blog₩post_form.html

```
    <div class="form-group row">
        {{ form.slug|add_label_class:"col-form-label col-sm-2 ml-3 font-weight-bold" }}
        <div class="col-sm-5">
            {{ form.slug|add_class:"form-control"|attr:"readonly" }}
        </div>
        <small class="form-text text-muted">{{ form.slug.help_text }}</small>
    </div>

    <div class="form-group row">
        {{ form.description|add_label_class:"col-form-label col-sm-2 ml-3 font-weight-bold" }}
        <div class="col-sm-5">
            {{ form.description|add_class:"form-control" }}
        </div>
        <small class="form-text text-muted">{{ form.description.help_text }}</small>
    </div>

    <div class="form-group row">
        {{ form.content|add_label_class:"col-form-label col-sm-2 ml-3 font-weight-bold" }}
        <div class="col-sm-8">
            {{ form.content|add_class:"form-control" }}
        </div>
    </div>

    <div class="form-group">
        <div class="offset-sm-2 col-sm-5">
            <input type="submit" value="Submit" class="btn btn-info"/>
        </div>
    </div>

</form>

{% endblock %}
```

4. 템플릿 코딩

ch99₩blog₩templates₩blog₩post_change_list.html

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

{% block title %}post_change_list.html{% endblock %}

{% block content %}

    <h1>Post Change - {{user}}</h1>
    <br>

    <table class="table table-bordered table-sm table-striped">

        <thead>
        <tr class="table-primary">
            <th>Title</th>
            <th>Description</th>
            <th>Owner</th>
            <th>Update</th>
            <th>Delete</th>
        </tr>
        </thead>

        <tbody>
        {% for item in object_list %}
        <tr>
            <td>{{ item.title }}</td>
            <td>{{ item.description }}</td>
            <td>{{ item.owner }}</td>
            <td><a href="{% url 'blog:update' item.id %}">Update</a></td>
            <td><a href="{% url 'blog:delete' item.id %}">Delete</a></td>
        </tr>
        {% endfor %}
        </tbody>

    </table>

{% endblock %}
```

4. 템플릿 코딩

ch99₩blog₩templates₩blog₩post_confirm_delete.html

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

{% block title %}post_confirm_delete.html{% endblock %}

{% block content %}

    <h1>Post Delete</h1>
    <br>

    <form action="." method="post">{% csrf_token %}
        <p>Are you sure you want to delete "{{ object }}" ?</p>
        <input type="submit" value="Confirm" class="btn btn-danger btn-sm" />
    </form>

{% endblock %}
```