Chapter-15

Real Time Project

Blog Application Development

How to specify dropdown choice field in models.py

```python
class Post(models.Model):
  STATUS_CHOICES=(('draft','Draft'),
                ('published','Published'))
status=models.CharField(max_length=10,choices=STATUS_CHOICES,default='draft')
```

slug field:

slug = models.SlugField(max_length=256,unique_for_date='publish')

- slug field can be used in urls.
- A slug is a short label containing only alphabet symbols,numbers,underscore symbol and hyphens.
- We can use slug field to build human understandable and SEO friendly URLs.
- unique_for_date specifies that slug is unique per publish date.

ForeignKey:

author = models.ForeignKey(User,related_name='blog-posts')

The value of author must be one of Users in default auth application.Multiple posts can be published by same author (same User). Hence it represents MANY-TO-ONE relationship. We can specify reverse relationship from User to Post with the related_name attribute.

DateTimeField:

publish = models.DateTimeField(default=timezone.now)

We have to use the following import ➔ from django.utils import timezone

models.py

```python
from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone
# Create your models here.
class Post(models.Model):
    STATUS_CHOICES=(('draft','Draft'),('published','Published'))
    title=models.CharField(max_length=264)
    slug=models.SlugField(max_length=264,unique_for_date='publish')
    author=models.ForeignKey(User,related_name='blog_posts')
    body=models.TextField()
    publish=models.DateTimeField(default=timezone.now)
    created=models.DateTimeField(auto_now_add=True)#datetime of create() action
```

```
    updated=models.DateTimeField(auto_now=True)#datetme of save() action
status=models.CharField(max_length=10,choices=STATUS_CHOICES,default='draft')

    class Meta:
        ordering=('-publish',)
    def __str__(self):
        return self.title
```

pytz Module (Python Timezone Module):

Timezone definitions are required for python and used by SQLite to work with datetimes. pytz module provides timezone definitions. Hence we required to install explicitly

```
pip install pytz
```

Note: Python 3.X alerady contains this module and we are not required to install.

Note: Django provides inbuilt support for timezones. In settings.py the following attribute already defined.

```
USE_TZ = True
```

Customization of Admin Interface:

We can customize admin interface for our model by defining ModelAdmin class.
In this class we can specify the following properties based on our requirement.

1) list_display:

Here we have to specify the field names which have to display.

2) list_filter

To filter records based on our provided fields
list_filter=('status','created','publish','author')
As the result, on the Right hand side side bar will be appeared for filtering purpose.

3) search_fields

search_fields=('title','body')
A search bar appeared on the page

4) prepopulated_fields

prepopulated_fields={'slug':('title',)}
If we type title then the same value will be considered automatically for the slug field also.

5) raw_id_fields

raw_id_fields=('author',)

author field is displayed with lookup widget,that can scale better than dropdown list. This is helpful if huge number of values are available.

6) date_hierarchy

date_hierarchy='publish'
Below search button,we can see a bar to navigate quickly based on published date.

7) ordering

ordering=['status','publish']
To display records according to specified order

admin.py

```python
from django.contrib import admin
from blog.models import Post

# Register your models here.
class PostAdmin(admin.ModelAdmin):
    list_display=['title','slug','author','publish','created','updated','status']
    list_filter=('status','created','publish','author')
    search_fields=('title','body')
    prepopulated_fields={'slug':('title',)}
    raw_id_fields=('author',)
    ordering=['status','publish']

admin.site.register(Post,PostAdmin)
```

Implementation of List View and Detail View By using FBVs:

views.py

```python
from django.shortcuts import render,get_object_or_404
from blog.models import Post

# Create your views here.
def post_list_view(request):
    post_list=Post.objects.all()
    return render(request,'blog/post_list.html',{'post_list':post_list})

def post_detail_view(request,year,month,day,post):
    post=get_object_or_404(Post,slug=post,
                           status='published',
                           publish__year=year,
                           publish__month=month,
                           publish__day=day)

    return render(request,'blog/post_detail.html',{'post':post})
```

urls.py

```
from django.conf.urls import url
from django.contrib import admin
from blog import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', views.post_list_view),
    url(r'^(?P<year>\d{4})/(?P<month>\d{2})/(?P<day>\d{2})/(?P<post>[-\w]+)/$', views.post_detail_view,name='post_detail'),
]
```

models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone
from django.core.urlresolvers import reverse
# Create your models here.
class CustomManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(status='published')

class Post(models.Model):
    STATUS_CHOICES=(('draft','Draft'),('published','Published'))
    title=models.CharField(max_length=264)
    slug=models.SlugField(max_length=264,unique_for_date='publish')
    author=models.ForeignKey(User,related_name='blog_posts')
    body=models.TextField()
    publish=models.DateTimeField(default=timezone.now)
  created=models.DateTimeField(auto_now_add=True)#datetime of create() action
    updated=models.DateTimeField(auto_now=True)#datetme of save() action
status=models.CharField(max_length=10,choices=STATUS_CHOICES,default='draft')
    objects=CustomManager()

    class Meta:
```

```
    ordering=('-publish',)
  def __str__(self):
    return self.title

  def get_absolute_url(self):
    return reverse('post_detail',args=[self.publish.year,
    self.publish.strftime('%m'),self.publish.strftime('%d'),self.slug])
```

Note: By using reverse() method we will get canonical URLs.

base.html

```
<!DOCTYPE html>
{%load staticfiles%}
<html lang="en" dir="ltr">
 <head>
   <meta charset="utf-8">
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
   <link rel="stylesheet" href="{% static "css/blog.css"%}">
   <title>{%block title %}{%endblock%}</title>
 </head>
 <body>
   <div class="content">
     {%block content%}{%endblock%}
   </div>
   <div class="sidebar">
     <h2>DURGA'S BLOG</h2>
     <p>Here You can see My Python Updations...</p>
   </div>
 </body>
</html>
```

post_list.html

```
<!DOCTYPE html>
{%extends 'blog/base.html'%}
{%block title %}DURGA'S BLOG{%endblock%}
{%block content%}
 <h1>DURGA's Blog</h1>
 {%for post in post_list %}
   <h2> <a href="{{post.get_absolute_url}}">{{post.title}}</a> </h2>
   <p id='date'>Published on {{post.publish}} by {{post.author|title}}</p>
```

```
    {{post.body|truncatewords:30|linebreaks}}
{%endfor%}
{%endblock%}
```

post_detail.html

```
<!DOCTYPE html>
{%extends 'blog/base.html'%}
{%block title %}{{post.title}}{%endblock%}
{%block content%}
    <h1>{{post.title}}</h1>
    <p id='date'>Published on {{post.publish}} by {{post.author|title}}</p>
    {{post.body|linebreaks}}
{%endblock%}
```

blog.css

```
.content{
  float:left;
  width: 60%;
  padding:0 0 0 30px;
}
.sidebar{
  float:right;
  width:30%;
  padding:10px;
  background: green;
  height: 100%;
}
h1{
  text-align: center;
  border-bottom: 1px solid #bbb;
  padding: 0 0 10px 0;
}
#date{
  color:yellow;
  font-size:12px;
  font-style: italic;
}
body{
  background: red;
  color:white;
}
```

Django Pagination:

- If more number of posts (records) are available then it is highly recommended to display all records across multiple pages, so that end user can feel more comfortable. This concept is called Pagination (ie. Display records across multiple pages is called pagination)
- Django can provide inbuilt support for Pagination.It contains paginator module and Paginator class.

views.py

```
from django.shortcuts import render,get_object_or_404
from django.core.paginator import Paginator,EmptyPage,PageNotAnInteger
from blog.models import Post

# Create your views here.
def post_list_view(request):
    post_list=Post.objects.all()
    paginator=Paginator(post_list,2)
    page_number=request.GET.get('page')
    try:
        post_list=paginator.page(page_number)
    except PageNotAnInteger:
        post_list=paginator.page(1)
    except EmptyPage:
        post_list=paginator.page(paginator.num_pages)
    return render(request,'blog/post_list.html',{'post_list':post_list})
```

How Pagination Works:

1) We have to create Paginator object with list of records and number of records per page.

```
paginator = Paginator(post_list,2)
```

2) We have to get list of records related to current page as follows.

```
page_number = request.GET.get('page')
post_list = paginator.page(page_number)
```

3) If the page_number is not an integer, means that we are not passing any page number in the query string with page attribute,it means it is first page,then we have to get first page records as follows

```
post_list=paginator.page(1)
```

4) If the parameter is higher than last page number then we will get EmptyPage.Instead of displaying EmptyPage we have to display last page records as follows

```
post_list = paginator.page(paginator.num_pages)
```

page_list.html

We have to add only one include tag to include paginator where ever pagination is required.

```
<!DOCTYPE html>
{%extends 'blog/base.html'%}
{%block title %}DURGA'S BLOG{%endblock%}
{%block content%}
 <h1>DURGA's Blog</h1>
 {%for post in post_list %}
   <h2> <a href="{{post.get_absolute_url}}">{{post.title}}</a> </h2>
   <p id='date'>Published on {{post.publish}} by {{post.author|title}}</p>
   {{post.body|truncatewords:30|linebreaks}}
{%endfor%}
{% include 'blog/pagination.html' with page=post_list%}
{%endblock%}
```

pagination.html

```
<div class="paginator">
  <span>
     {%if page.has_previous%}
        <a href="?page={{page.previous_page_number}}">Previous</a>
     {%endif%}
 </span>
  <span class='current'>
    page {{page.number}} of {{page.paginator.num_pages}}
  </span>
  <span>
     {%if page.has_next%}
        <a href="?page={{page.next_page_number}}">Next</a>
     {%endif%}
 </span>
</div>
```

Note: We can reuse this pagination template anywhere pagination is required.

Class Based Views For Pagination:

We have to use paginate_by property to specify the number of records per page.
ListView will always send 'page_obj' object which contains the current page records

```
from django.views.generic import ListView
class PostListView(ListView):
    model=Post
```

```
paginate_by=1
```

post_list.html

```
<!DOCTYPE html>
{%extends 'blog/base.html'%}
{%block title %}DURGA'S BLOG{%endblock%}
{%block content%}
 <h1>DURGA's Blog</h1>
 {%for post in post_list %}
   <h2> <a href="{{post.get_absolute_url}}">{{post.title}}</a> </h2>
   <p id='date'>Published on {{post.publish}} by {{post.author|title}}</p>
   {{post.body|truncatewords:30|linebreaks}}
{%endfor%}
{% include 'blog/pagination.html' with page=page_obj%}
{%endblock%}
```

pagination.html

```
same code
<div class="paginator">
  <span>
     {%if page.has_previous%}
        <a href="?page={{page.previous_page_number}}">Previous</a>
     {%endif%}
 </span>
  <span class='current'>
    page {{page.number}} of {{page.paginator.num_pages}}
  </span>
  <span>
     {%if page.has_next%}
        <a href="?page={{page.next_page_number}}">Next</a>
     {%endif%}
 </span>
</div>
```

Mail Sending Functionality:

In the web application sending mail is very common requirement. Django provides in built support for mail sending functionality.
Django provides mail module & send_mail() function.

To send email compulsory SMTP Server must be required.
SMTP → Simple Mail Transfer Protocol

We have to configure SMTP Server information in the settings.py file.
EMAIL_HOST: The SMTP Server Host. The default value is localhost
EMAIL_PORT: The SMTP Server Port. The default value is 25
EMAIL_HOST_USER: Username for SMTP Server
EMAIL_HOST_PASSWORD:Password for SMTP Server
EMAIL_USE_TLS: whether to use a TLS Secure Connection
(TLS: Transport Layer Security)

For gmail SMTP Server:

EMAIL_HOST= 'smtp.gmail.com'
EMAIL_PORT= 587
EMAIL_HOST_USER= Username for SMTP Server
EMAIL_HOST_PASSWORD=Password for SMTP Server
EMAIL_USE_TLS= True

We can send mail by using send_mail() function
send_mail(subject,msg,sender,list of receivers,fail_silently=False)

fail_silently is optional parameter. We are telling to raise exception if email not sending properly.

```
from django.core.mail import send_mail
send_mail('Hello','Very imp msg','python6656@gmail.com',[
'xxx@gmail.com',yyy@gmail.com'])
```

If we are getting 1 then the mail sent successfully.

forms.py

```
from django import forms
class EmailSendForm(forms.Form):
    name=forms.CharField()
    email=forms.EmailField()
    to=forms.EmailField()
    comments=forms.CharField(required=False,widget=forms.Textarea)
```

views.py

```
from django.core.mail import send_mail
from blog.forms import EmailSendForm
...
def mail_send_view(request,id):
    post=get_object_or_404(Post,id=id,status='published')
    sent=False
    if request.method=='POST':
```

```
        form=EmailSendForm(request.POST)
        if form.is_valid():
            cd=form.cleaned_data
            post_url=request.build_absolute_uri(post.get_absolute_url())
            subject='{}({}) recommends you to read "{}"'.format(cd['name'],cd['email'],
post.title)
            message='Read Post At: \n {}\n\n{}\' Comments:\n{}'.format(post_url,cd
['name'],cd['comments'])
            send_mail(subject,message,'durga@blog.com',[cd['to']])
            sent=True
    else:
        form=EmailSendForm()
return render(request,'blog/sharebymail.html',{'post':post,'form':form,'sent':sent})
```

sharebymail.html

```
<!DOCTYPE html>
{%extends 'blog/base.html'%}
{%block title %}Share Post By Email{%endblock%}
{%block content%}
  {%if sent%}
   <h1>Email Successfully Sent !!!</h1>
    <p>"{{post.title}}" was sent by email</p>
  {%else%}
        <h1>Share {{post.title}}" by Email!!!"</h1>
       <form method="post">
         {{form.as_p}}
         {%csrf_token%}
         <input type="submit" name="" class='btn btn-lg btn-
success' value="Send Mail">
       </form>
    {%endif %}
{%endblock%}
```

urls.py

```
urlpatterns = [
     ...
     url(r'^(?P<id>\d+)/share/$', views.mail_send_view),
]
```

Add Send Mail Button in Details Page:

post_detail.html

```
<!DOCTYPE html>
{%extends 'blog/base.html'%}
{%block title %}{{post.title}}{%endblock%}
{%block content%}
    <h1>{{post.title}}</h1>
    <p id='date'>Published on {{post.publish}} by {{post.author|title}}</p>
    {{post.body|linebreaks}}
    <a href="/{{post.id}}/share" class='btn btn-lg btn-
primary'>Share Post By Email</a>
{%endblock%}
```

Adding Comments Section to the Blog:

Activities:

1) We required to create a Model to save comments
2) We required to create a model based form to submit comments
3) We required to create a view function to process comments and save to the database
4) We have to edit post_detail.html to display list of comments and add a form to submit comments

models.py

Comments are always associated with Post and without Post there is no chance of existing comments.Each Post can have multiple comments.Hence it is Many-to-One relationship, which can be specified by Foreign Key.

```
class Comment(models.Model):
  post=models.ForeignKey(Post,related_name='comments')
```

We can use related_name to access all comments associated with a particular post.
post.comments.all()

If we are not using related_name then the default related name is:
modelclassname_set
Eg: comment_set

models.py

```
...
class Comment(models.Model):
    post=models.ForeignKey(Post,related_name='comments')
    name=models.CharField(max_length=32)
    email=models.EmailField()
    body=models.TextField()
    created=models.DateTimeField(auto_now_add=True)
    updated=models.DateTimeField(auto_now=True)
    active=models.BooleanField(default=True)
    class Meta:
        ordering=('created',)
    def __str__(self):
        return 'Commented By {} on {}'.format(self.name,self.post)

py manage.py  makemigrations
py manage.py  migrate
```

admin.py

```
from django.contrib import admin
from blog.models import Post,Comment

# Register your models here.
class PostAdmin(admin.ModelAdmin):
    list_display=['title','slug','author','publish','created','updated','status']
    list_filter=('status','created','publish','author')
    search_fields=('title','body')
    prepopulated_fields={'slug':('title',)}
    raw_id_fields=('author',)
    ordering=['status','publish']
class CommentAdmin(admin.ModelAdmin):
    list_display=('name','email','post','body','created','updated','active')
    list_filter=('active','created','updated')
    search_fields=('name','email','body')

admin.site.register(Post,PostAdmin)
admin.site.register(Comment,CommentAdmin)
```

Creation of Model Based form for comments

```
from blog.models import Comment
class CommentForm(forms.ModelForm):
   class Meta:
      model=Comment
      fields=('name','email','body')
```

Display and Processing Form in views.py

```
from blog.models import Comment
from blog.forms import CommentForm

def post_detail_view(request,year,month,day,post):
   post=get_object_or_404(Post,slug=post,
                          status='published',
                          publish__year=year,
                          publish__month=month,
                          publish__day=day)
   comments=post.comments.filter(active=True)
   csubmit=False
   if request.method=='POST':
      form=CommentForm(data=request.POST)
      if form.is_valid():
         new_comment=form.save(commit=False)
         new_comment.post=post
         new_comment.save()
         csubmit=True
   else:
      form=CommentForm()
   return render(request,'blog/post_detail.html',{'post':post,'form':form,'comment
s':comments,'csubmit':csubmit})
```

Adding Comments to the Detail Page:

We have to display

1) Number of comments
2) For every comment we have to add head part
3) Comment body
4) Add Comment form to submit new comment
5) If already submitted then display success message

```
<!DOCTYPE html>
{%extends 'blog/base.html'%}
{%block title %}{{post.title}}{%endblock%}
{%block content%}
```

```
        <h1>{{post.title}}</h1>
        <p id='date'>Published on {{post.publish}} by {{post.author|title}}</p>
        {{post.body|linebreaks}}
        <a href="/{{post.id}}/share" class='btn btn-lg btn-primary'>
    Share Post By Email</a>
        {% with comments.count as comments_count %}
        <h2>{{comments_count}} Comment{{comments_count|pluralize}}</h2>
        {% endwith%}
        {%if comments %}
          {%for comment in comments %}
            <p id='ch'> comment {{forloop.counter}} by {{comment.name}} on
    {{comment.created}}</p>
            <div class="cb">{{comment.body|linebreaks}}</div>
            <hr>
           {%endfor%}
           {%else%}
           <p>There are NO Comments Yet !!!</p>
        {%endif%}
        {%if csubmit %}
          <h2>Your Comment Added Succefully</h2>
          {%else%}
          <form method="post">
           {{form.as_p}}
           {%csrf_token%}
           <input type="submit" name="" value="Submit Comment">
          </form>
        {%endif%}
{%endblock%}
```

Note:

```
{% with comments.count as comments_count %}
    <h2>{{comments_count}} Comment{{comments_count|pluralize}}</h2>
{% endwith%}
```

{%with%} can be used to assign a value to new variable. We can use that variable any number of times based on our requirement. Most of the times it can be used to avoid hitting database multiple times.

Note:
pluralize can be used to add 's' if it is required
Comment{{comments_count|pluralize}}
Eg: 0 Comments
    1 Comment
    2 Comments

Adding Tags Functionality To our Application:

Advantages of Tagging

1) For Search Engine Optimization
2) For easy navigation to the end user

By using 3rd party application 'taggit' we can implement tagging in our application.

How to install taggit Application
pip install django-taggit

After installation add this application in the settings.py

```
INSTALLED_APPS = [
    ....,
    'blog',
    'taggit'
]
```

Add TaggableManager to our Post Model

```
from taggit.managers import TaggableManager
class Post(models.Model):
    ....
    objects=CustomManager()
    tags=TaggableManager()
    ...
```

To get all Posts:
```
Post.objects.all()
```

To get all tags associated with a post:
```
post.tags.all()
```

makemigrations and migrate:

to reflect changes made to Post model.

Operations related to taggit:

```
D:\djangoprojects\blogproject>py manage.py shell
>>> from blog.models import Post
>>> post=Post.objects.get(id=2)
>>> post.tags.add('python','django','datascience','dhoni')
>>> post.tags.all()
<QuerySet [<Tag: dhoni>, <Tag: django>, <Tag: python>, <Tag: datascience>]>
```

```
>>> post.tags.remove('datascience')
>>> post.tags.all()
<QuerySet [<Tag: dhoni>, <Tag: django>, <Tag: python>]>
```

Add Tags to the List Page:

To display tags associated with current post,we have to write code as follows.

```
 {%for post in post_list %}
   ...
   <p>Tags:{{post.tags.all|join:','}}</p>
   ..
{%endfor%}
```

post_list.html

```
 <!DOCTYPE html>
{%extends 'blog/base.html'%}
{%block title %}DURGA'S BLOG{%endblock%}
{%block content%}
 <h1>DURGA's Blog</h1>
 {%for post in post_list %}
   <h2> <a href="{{post.get_absolute_url}}">{{post.title}}</a> </h2>
   <p>Tags:{{post.tags.all|join:','}}</p>
   <p id='date'>Published on {{post.publish}} by {{post.author|title}}</p>
   {{post.body|truncatewords:30|linebreaks}}
{%endfor%}
{% include 'blog/pagination.html' with page=post_list%}
{%endblock%}
```

Editing post_list_view() for Tags

If we click any tag name then to display all posts related to that tag we have to edit view function.

```
from taggit.models import Tag
def post_list_view(request,tag_slug=None):
    post_list=Post.objects.all()
    tag=None
    if tag_slug:
        tag=get_object_or_404(Tag,slug=tag_slug)
        post_list=post_list.filter(tags__in=[tag])

    paginator=Paginator(post_list,2)
    page_number=request.GET.get('page')
    try:
        post_list=paginator.page(page_number)
```

```
  except PageNotAnInteger:
    post_list=paginator.page(1)
  except EmptyPage:
    post_list=paginator.page(paginator.num_pages)
  return render(request,'blog/post_list.html',{'post_list':post_list,'tag':tag})
```

urls.py

```
from django.conf.urls import url
from django.contrib import admin
from blog import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^mail/', views.mail_send_view),
    url(r'^$', views.post_list_view),
    url(r'^tag/(?P<tag_slug>[-
\w]+)/$', views.post_list_view, name='post_list_by_tag_name'),
    # url(r'^$', views.PostListView.as_view()),
     url(r'^(?P<year>\d{4})/(?P<month>\d{2})/(?P<day>\d{2})/(?P<post>[-
\w]+)/$', views.post_detail_view,name='post_detail'),
     url(r'^(?P<id>\d+)/share/$', views.mail_send_view),
]
```

post_list.html

```
<!DOCTYPE html>
{%extends 'blog/base.html'%}
{%block title %}DURGA'S BLOG{%endblock%}
{%block content%}
 <h1>DURGA's Blog</h1>
 {%if tag%}
 <h2>Posts tagged with "{{tag.name}}"</h2>
 {%endif%}
  {%for post in post_list %}
  <h2> <a href="{{post.get_absolute_url}}">{{post.title}}</a> </h2>
   <p>Tags:
     {%for tag in post.tags.all %}
     <a href="{%url "post_list_by_tag_name" tag.slug %}">{{tag.name}}</a>
     {%if not forloop.last%}, {%endif%}
     {%endfor%}
   </p>
  <p id='date'>Published on {{post.publish}} by {{post.author|title}}</p>
  {{post.body|truncatewords:30|linebreaks}}
{%endfor%}
```

```
{% include 'blog/pagination.html' with page=post_list%}
{%endblock%}
```

How to display similar posts:

1) Retrieve all tags for the current post
2) Get all posts that are tagged with any of those tags
3) Exclude current post from the list to avoid recommended the same post
4) Order the results by number of tags shared with the current post
5) If two or more posts with the same number of tags, recommend the most recent post
6) Limit the query to the number of posts we have to recommend

```
from django.db.models import Count

post_tags_ids=post.tags.values_list('id',flat=True)
similar_posts=Post.objects.filter(tags__in=post_tags_ids).exclude(id=post.id)
similar_posts=similar_posts.annotate(same_tags=Count('tags')).order_by('-same_tags','publish')[:4]
```

Note: By using annotate() function, we can add fields to the query. In the above code, Count() function returned value we are assigning to same_tags variable and this variable can be used in next level.

views.py

```
....
from blog.models import Comment
from blog.forms import CommentForm
from django.db.models import Count

def post_detail_view(request,year,month,day,post):
    post=get_object_or_404(Post,slug=post,
                           status='published',
                           publish__year=year,
                           publish__month=month,
                           publish__day=day)
    post_tags_ids=post.tags.values_list('id',flat=True)
    similar_posts=Post.objects.filter(tags__in=post_tags_ids).exclude(id=post.id)
    similar_posts=similar_posts.annotate(same_tags=Count('tags')).order_by('-same_tags','publish')[:4]

    comments=post.comments.filter(active=True)
    csubmit=False
    if request.method=='POST':
        form=CommentForm(data=request.POST)
        if form.is_valid():
```

```
        new_comment=form.save(commit=False)
        new_comment.post=post
        new_comment.save()
        csubmit=True
  else:
     form=CommentForm()
  return render(request,'blog/post_detail.html',{'post':post,'form':form,'comments':comments,'csubmit':csubmit,'similar_posts':similar_posts})
```

post_detail.html
In this html page we have to display similar posts

```
<h2>Similar Posts</h2>
{%if similar_posts%}
   {%for post in similar_posts%}
     <h3> <a href="{{post.get_absolute_url}}">{{post.title}}</a> </h3>
   {%endfor%}
    {%else%}
          <p>No Similar Posts yet</p>
{%endif%}
```

complete file

```
<!DOCTYPE html>
{%extends 'blog/base.html'%}
{%block title %}{{post.title}}{%endblock%}
{%block content%}
    <h1>{{post.title}}</h1>
    <p id='date'>Published on {{post.publish}} by {{post.author|title}}</p>
    {{post.body|linebreaks}}
    <a href="/{{post.id}}/share" class='btn btn-lg btn-primary'>
Share Post By Email</a>
    <h2>Similar Posts</h2>
    {%if similar_posts%}
       {%for post in similar_posts%}
         <h3> <a href="{{post.get_absolute_url}}">{{post.title}}</a> </h3>
        {%endfor%}
        {%else%}
        <p>No Similar Posts yet</p>
     {%endif%}
    {% with comments.count as comments_count %}
    <h2>{{comments_count}} Comment{{comments_count|pluralize}}</h2>
    {% endwith%}
    {%if comments %}
       {%for comment in comments %}
```

```
            <div class="commentdata">
              <p id='ch'> comment {{forloop.counter}} by {{comment.name}} on
    {{comment.created}}</p>
              {{comment.body|linebreaks}}
            </div>
          {%endfor%}
          {%else%}
          <p>There are NO Comments Yet !!!</p>
      {%endif%}
      {%if csubmit %}
         <h2>Your Comment Added Succefully</h2>
         {%else%}
         <form method="post">
           {{form.as_p}}
           {%csrf_token%}
           <input type="submit" name="" value="Submit Comment">
         </form>
      {%endif%}
{%endblock%}
```

Custom Template Tags:

Django provide a big list of template tags like

{{post.title}}
{% csrf_token %}
{% if post_list %} {%endif %}
{%for post in post_list %} ..{%endfor %}
etc

Sometimes these predefined tags may not fulfill our requirement. In this case we can define our own template tag based on our required functionality,such type of tags are called Custom Template Tags.

3 Utility Functions to define Custom Template Tags:

Django provides the following 3 helper functions for creating our own template tags.

1) **simple_tag:** Perform some processing and returns a string
2) **inclusion_tag:** Perform some processing and returns a rendered template
3) **assignment_tag:** Perform some processing and assign the result to a variable in the context.

How to Create Custom Template Tag:

1) Inside our application folder ceate a folder named with 'templatetags'
2) In that 'templatetags' folder,create an empty file __init__.py,so that this folder is considered as python package
3) In that 'templatetags' folder, create a python file(blog_tags.py) to hold custom template tags.

```
...
 blog
   |--.....
   |-templatetags
      |- __init__.py
      |- blog_tags.py
```

Note: custom template tags should be placed in application folder but not in project folder

Usage of simple_tag to return number of posts published:

blog_tags.py

```python
from django import template
from blog.models import Post
register=template.Library()

@register.simple_tag
def total_posts():
    return Post.objects.count()
```

Note:

1) By using register variable we can register template tags with Library.
2) By default function name (total_posts) will become tag name.But we can specify our own name as follows

@register.simple_tag(name='my_tag')

How to use Custom Template Tag

1) We have to load custom tag functionality as follows → {% load blog_tags %}
2) We can call template tag as follows → {% total_posts %}

base.html

```html
<!DOCTYPE html>
{%load blog_tags %}
{%load staticfiles%}
<html lang="en" dir="ltr">
```

```
 <head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
  <link rel="stylesheet" href="{% static "css/blog.css"%}">
  <title>{%block title %}{%endblock%}</title>
 </head>
 <body>
  <div class="content">
   {%block content%}{%endblock%}
  </div>
  <div class="sidebar">
   <h2>DURGA'S BLOG</h2>
   <p>The total number of published posts upto today: {%total_posts%}</p>
  </div>
 </body>
</html>
```

Usage of inclusion_tag to display Latest Posts

The most commonly used and powerful utility function is inclusion_tag.

blog_tags.py

```
...
@register.inclusion_tag('blog/latest_posts123.html')
def show_latest_posts(count=5):
    latest_posts=Post.objects.order_by('-publish')[:count]
    return {'latest_posts':latest_posts}
```

latest_posts123.html

```
<ul>
 {% for post in latest_posts  %}
 <li><a href="{{post.get_absolute_url}}">{{post.title}}</a></li>
 {% endfor %}
</ul>
```

base.html

```
...
<div class="sidebar">
   <h2>DURGA'S BLOG</h2>
   <p>The total number of published posts upto today:{%total_posts %}</p>
```

```
    <h3>Latest Posts:</h3>{% show_latest_posts 100 %}
</div>
```

Usage of assignment_tag to display Most Commented Posts:

Assignment_tag is exactly same as simple_tag except that it stores the result in the given variable.

blog_tags.py

```
from django.db.models import Count

@register.assignment_tag
def get_most_commented_posts(count=5):
   return Post.objects.annotate(total_comments=Count('comments')).order_by('-total_comments')[:count]
```

base.html

```
<div class="sidebar">
    <h2>DURGA'S BLOG</h2>
    <p>The total number of published posts upto today: <span id='pcount'>{%total_posts %}</span></p>
    <h3>Latest Posts:</h3>{% show_latest_posts 100 %}
    <h3>Most Commented Posts:</h3>
    {% get_most_commented_posts as most_commented_posts%}
    <ul>
      {% for post in most_commented_posts  %}
      <li><a href="{{post.get_absolute_url}}" id='la'>{{post.title}}</a></li>
      {% endfor %}
    </ul>
</div>
```