CREATING VIRTUALENVIRONMENT IN WINDOWS

# pip install virtualenvwrapper-win -> for creating wrapper

# mkvirtualenv test -> for creating a virtualenv test

# pip install Django -> for Django

# mkdir projects -> for creating a directory

#

**DJANGO GIRLS**

In settings

We can set hosts to allow to run the server

Eg: ALLOWED\_HOSTS = [

'127.0.0.1',

'.amazonaws.com',

]

Here, we allowed both amazonaws.com and localhost. Keep it empty for working in all hosts.

**FOR SETTING STATIC\_ROOT**

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static') -> for setting static root.

We can set static files directory too to check static files as follows instead of STATIC\_ROOT.

STATICFILES\_DIRS = [

Os.path.join(BASE\_DIR, ‘ static‘ )

]

But , we use STATICFILES \_DIRS in production

Do migration then

# python manage.py migrate

Run server then to test

# python manage.py runserver

Then create an app Blog

# Django-admin startapp blog

Then in settings->installed app , add ‘blog’.

MODEL

from django.db import models

from django.utils import timezone

from django.conf import settings

# Create your models here.

class Post(models.Model):

author = models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE)

title = models.CharField(max\_length=100)

text = models.TextField()

created\_date = models.DateTimeField(default=timezone.now)

published\_date = models.DateTimeField(blank=True, null=True)

def publish(self):

self.published\_date = timezone.now()

self.save()

def \_\_str\_\_(self): # used to print the post title

return self.title

Then make migration to affect changes to the model

# makemigrations

# migrate

ADMIN

For accessing the models created,

We need to register it in admin page

from django.contrib import admin

from .models import \*

# Register your models here.

admin.site.register(Post) -> registered the model Post.

Here , the model Post registered to admin , so it will shows in admin page and admin can add, delete , edit it.

SUPERUSER

To access admin page, create super user.

# python manage.py createsuperuser

Then done adding some posts in admin page

**URLS**

**Then URL c**onfiguration Done.

**VIEWS**

Views helps to get data from model and to pass to templates

Then template rendered.

**ORM AND QUERYSETS**

Just like sql, to read data from database.

# Python manage.py shell -> for opening shell

# Post.objects.all() -> for showing all objects of ‘Post”

We need to import the model first

# from blog.models import \* ->Post model imported

# From django.contrib.auth.models import User

# User.objects.all() -> to list all users

# Post.objects.create(author=me, title='Sample title', text='Test')

Here, we specified the author as me , but it will give error.

So , we need to the following.

List the superusers with User.objects.all() after importing the User module.

Then set me as the superuser .

me = User.objects.get(username=”digital”)

Now , the “ me “ contain user digital.

So ,we can create new Posts as follows .

# Post.objects.create(author=me,title=”testing”,text=”testing purpose”)

WE uses “ create “ for creation of new post.

Notes :

objects.all() -> for listing all objects

objects.get() -> to get (retreive)

objects.create() -> for creation

objects.filter(codition) -> for filtering

objects.order\_by(“condition”) -> for ordering

objects.order\_by(“ – condition”) -> if we gives a “ - ” at beginning of the condition , it will order it reverse order.

**FILTERING**

# Post.objects.filter(title=”digital”)

Here, it will list the dataset with title as “digital”

# Post.objects.filter(title\_\_contains=”digital”) ->same as above, but shows any word in a single title have digital

**ORDERING**

# Post.objects.order\_by(“created\_date”) -> for ascending order

# Post.objects.order\_by(“-created\_date”) -> forreverse order

**CHAINNIG QUERIES**

# Post.objects.filter(published\_date\_\_lte=timezone.now()).order\_by('published\_date')

We can combine queries, here, first we filtered the objects and then ordered it .

Use “ . “ after one condition .

**PASSING DATAS FROM MODEL TO TEMPLATE** WITH VIEWS AS INTERMEDIATE

This is actually MVT.

Here , the data are fetched from model to views and from views to template.

Eg codes:

In view

def post\_list(request):

posts = Post.objects.all() -> gets all objects of “Post” and sets to posts

print(posts)

return render(request,'blog/post\_list.html',{"post":posts}) -> passed the “post” to Html page

in HTML

{% for i in post %}

{{ i }} {{ i.author }} {{ i.text }}<br>

{% endfor %}