Module – 4

Learning Objectives

- **Task – Add Post model to application**
- Creating Models
- Accessing models using Django shell and working on ORM Query Sets
- **Task – Display the objects on the template.**
- Adding a ListView of objects on the home page using class-based and function-based views approach

Creating a Database Model

Our first task is to create a database model where we can store objects. Django will turn this model into a database table for us. Django imports module ***models*** to help us build new database models, which will “model” the characteristics of the data in our database.

blog/models.py

```
from django.db import models
from django.utils import timezone

class Post(models.Model):
    author = models.ForeignKey('auth.User', on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    text = models.TextField()
    created_date = models.DateTimeField(default=timezone.now)

    # significance of str()
    def __str__(self):
        return self.title
```

More on Fields: **https://docs.djangoproject.com/en/2.0/ref/models/fields/**

Activating models

Whenever we create or modify an existing model we’ll need to update Django in a two-step process.

1. We create a migration file with the makemigrations command which generates the SQL commands for preinstalled apps in our INSTALLED_APPS setting. Migration files do not execute those commands on our database file, rather they are a reference of all new changes to our models. This approach means that we have a record of the changes to our models over time.

2. We build the actual database with migrate which does execute the instructions in our migrations file.

```
D:\MyBlog>python manage.py makemigrations
Migrations for 'blog':
  blog\migrations\0001_initial.py
    - Create model Post

D:\MyBlog>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, blog, contenttypes, sessions
Running migrations:
  Applying blog.0001_initial... OK
```

Django Shell

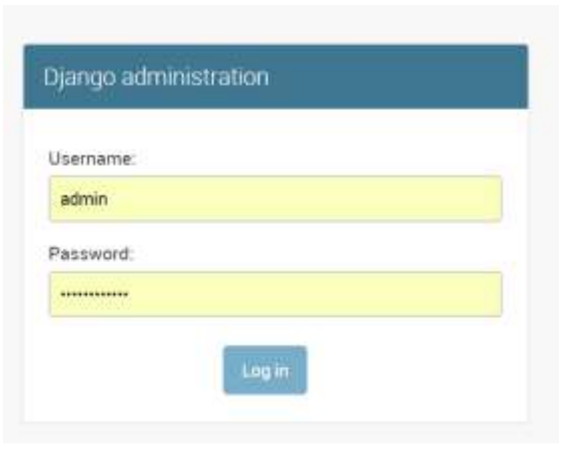
```
Example: Just for ORM querysets example.

models.py
class Icecream(models.Model):
    name = models.CharField(max_length = 30)
    price = models.IntegerField()

# significance of str()
    def __str__(self):
        return self.name
==============================================================================================================
D:\MyBlog>python manage.py shell
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
# Import the model to be accessed
>>> from blog.models import Icecream

# Create a new Object
>>> Icecream.objects.create(name = "Mango", price = 50)
<Icecream: Icecream object (1)>
>>> Icecream.objects.create(name = "Strawberry", price = 70)
<Icecream: Icecream object (2)>
>>> Icecream.objects.create(name = "Vanilla", price = 30)
<Icecream: Icecream object (3)>

# Fetch list of all objects
>>> ic_list = Icecream.objects.all()
>>> list(ic_list)
[<Icecream: Icecream object (1)>, <Icecream: Icecream object (2)>, <Icecream: Icecream object (3)>]

# Count number of objects.
>>> Icecream.objects.count()
3

# Fetch object by its id
>>> Icecream.objects.get(id=1)
<Icecream: Icecream object (1)>

# Filter objects on various criterion
>>> Icecream.objects.filter(name = "Mango")
<QuerySet [<Icecream: Icecream object (1)>]>
>>> Icecream.objects.filter(price__lte = 50)
<QuerySet [<Icecream: Icecream object (1)>, <Icecream: Icecream object (3)>]>
>>> Icecream.objects.filter(name__in = ("Mango", "Vanilla"))
<QuerySet [<Icecream: Icecream object (1)>, <Icecream: Icecream object (3)>]>
>>> Icecream.objects.filter(price__in = range(40,100))
<QuerySet [<Icecream: Icecream object (1)>, <Icecream: Icecream object (2)>]>
>>>
```

Django Admin

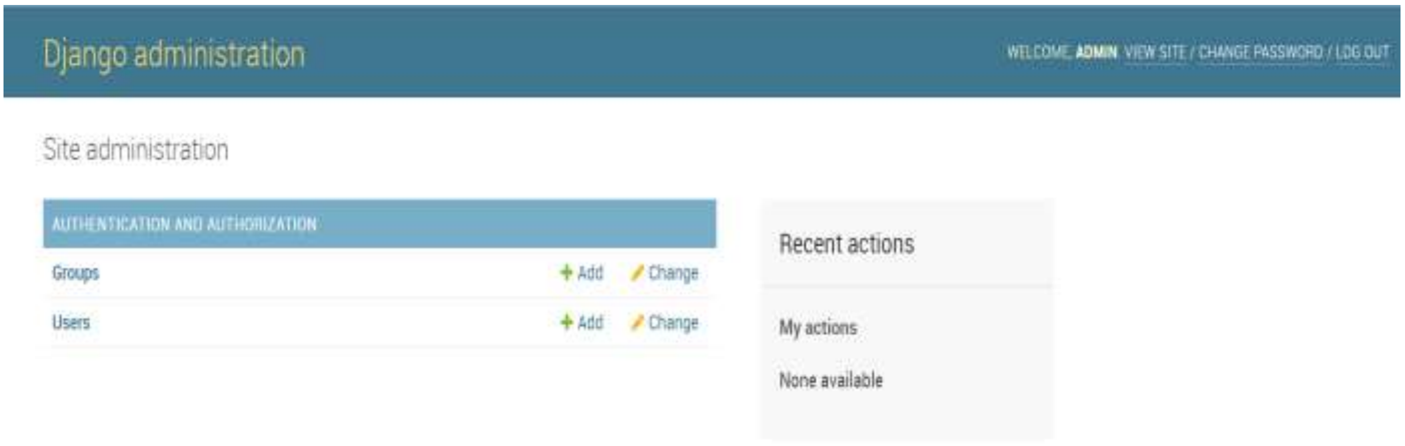
Django provides us with a robust admin interface for interacting with our database. To use the Django admin, we first need to create a superuser who can login.

```
D:\MyBog>python manage.py createsuperuser
Username (leave blank to use 'sia'): admin
Email address: vaidehi.spit@gmail.com
Password:
Password (again):
Superuser created successfully.
```

Restart the Django server with python manage.py runserver and in your browser go to **http://127.0.0.1:8000/admin/**. You should see the admin's login screen:



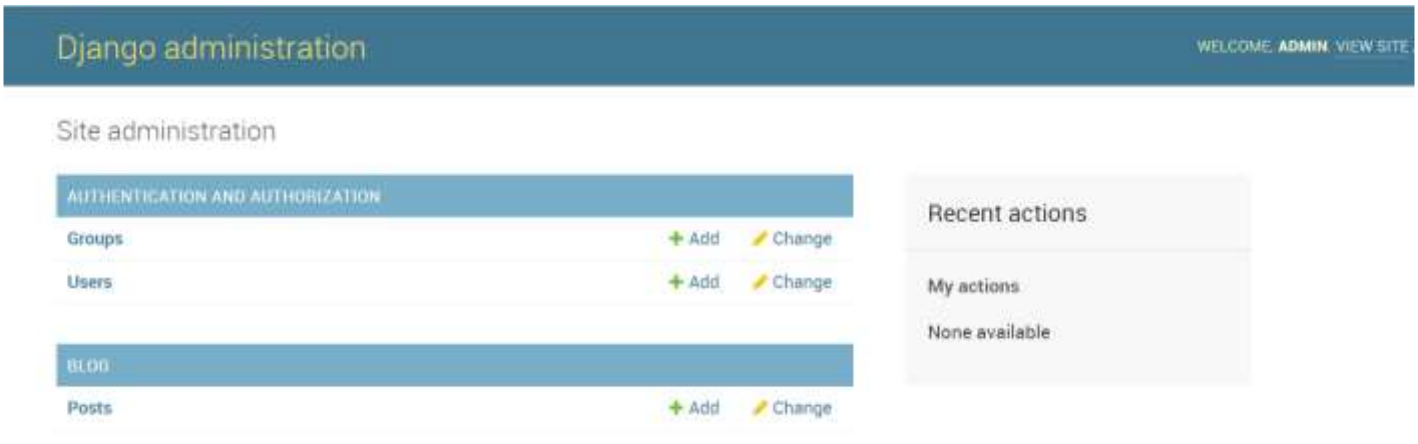
Username: admin



To make our models reflect into the admin view make following changes to admin.py file

MyBlog/ admin.py

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

admin.site.register(Post)
```



Building View for Models

blog/views.py

```
from django.views.generic import ListView
from django.shortcuts import render

from .models import Post

# List all the Posts
# Function based views
def postListView(request):
    posts = Post.objects.all()
    return render(request, 'listposts.html', {'posts': posts})

# Class Based Views
class PostListView(ListView):
    model = Post
    context_object_name = "posts"
    template_name = "listposts.html"
```

listposts.html

```
{% extends 'base.html' %}
{% block title%}Home{% endblock%}
{% block base%}
<br><br>
<div class="row">
    <div class="col-md-8">
        {% for post in posts %}
            <h3>{{ post.title }}</h3>
            {{ post.published_date }}
        {% endfor %}
    </div>
</div>
{% endblock%}
```

(modifying homepage view to show listposts)

Tests

Previously we were only testing static pages so we used **SimpleTestCase**. But now that our homepage works with a database, we need to use **TestCase** which will let us create a "test" database we can check against. In other words, we don't need to run tests on our actual database but instead can make a separate test database, fill it with sample data, and then test against it. Let's start by adding a sample post to the text database field and then check that it is stored correctly in the database. It's important that all our test methods start with test_ so Django knows to test them!

When you run your tests, the default behavior of the test utility is to find all the test cases (that is, subclasses of unittest.TestCase) in any file whose name begins with test, automatically build a test suite out of those test cases, and run that suite.

blog/tests.py

```
from django.contrib.auth import get_user_model
from django.test import Client, TestCase
from django.urls import reverse

from .models import Post


class BlogTests(TestCase):

# Performs initializations
    def setUp(self):
        self.user = get_user_model().objects.create_user(
            username='testuser',
            email='test@email.com',
            password='secret'
        )

        self.post = Post.objects.create(
            title='A good title',
            text='Nice body content',
            author=self.user,
        )

# Test - 1
    def test_string_representation(self):
        post = Post(title='A sample title')
        self.assertEqual(str(post), post.title)

# Test - 2
    def test_post_content(self):
        self.assertEqual(str(self.post.title), 'A good title')
        self.assertEqual(str(self.post.author), 'testuser')
        self.assertEqual(str(self.post.text), 'Nice body content')
```

```
# Test - 3
    def test_post_list_view(self):
        response = self.client.get('/blog/')
        response = self.client.get(reverse('viewblogs'))

# Test - 4
        print(self.post.title, response)
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'listblogs.html')
```