Django

What is Django?

- Django is a web application framework written in Python programming language to develop web applications.

- It is based on MVT (Model View Template) design pattern.

- It takes less time to build application after collecting client requirement.

- Free and open source

- Encourages rapid development

- Django is widely accepted and used by various well-known sites such as: Instagram, Mozilla, Disqus, Pinterest, Bitbucket, The Washington Times etc.

Django Features

- **Rapid development**

- **Full featured:** Django includes various helping task modules and libraries which can be used to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds etc.

- **Vast and Supported community:** Django is an one of the most popular web framework. It has widely supportive community and channels to share and connect.

- **Very Secure:** Django takes security seriously and helps developers to avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery etc. Its user authentication system provides a secure way to manage user accounts and passwords.

- **Scalability:** Django is scalable in nature and has ability to quickly and flexibly switch from small to large scale application project.

- **Versatile:** Companies are using Django to build various types of applications like: content management systems, social networks sites or scientific computing platforms etc.

MVT(Model-View-Template)

- It is a collection of three important components Model View and Template. The Model helps to handle database. It is a data access layer which handles the data.
- The Template is a presentation layer which handles User Interface part completely. The View is used to execute the business logic and interact with a model to carry data and renders a template.
- Control is handled by the framework itself.

Here, a user **requests** for a resource to the Django, Django works as a controller and check to the available resource in URL.

If URL maps, **a view is called** that interact with model and template, it renders a template.

Django responds back to the user and sends a template as a **response**.

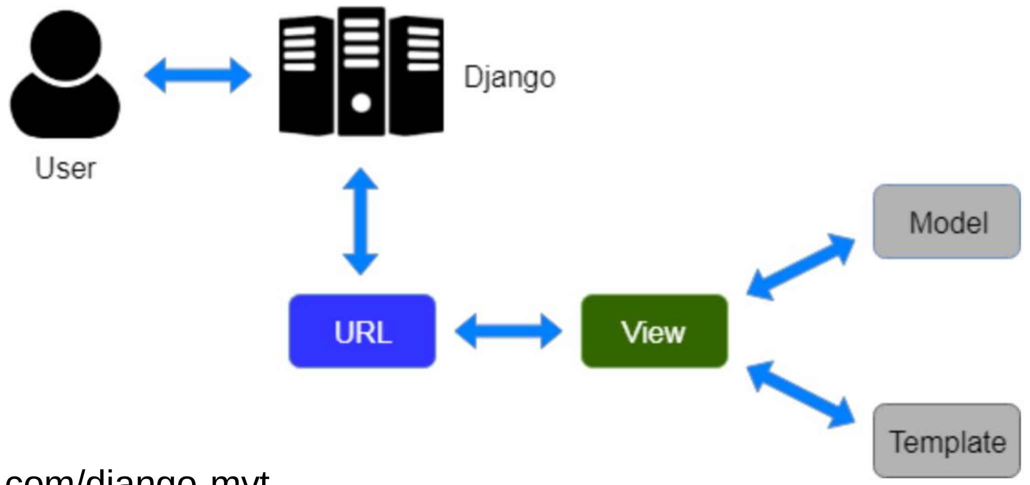


Image source: https://www.javatpoint.com/django-mvt

Django Apps

Each Project/website has separate apps.

Can have single app

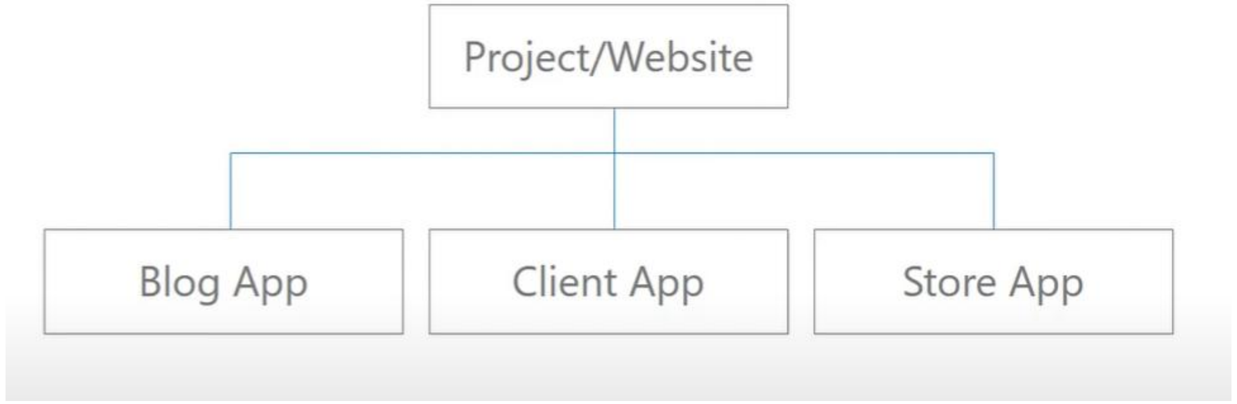
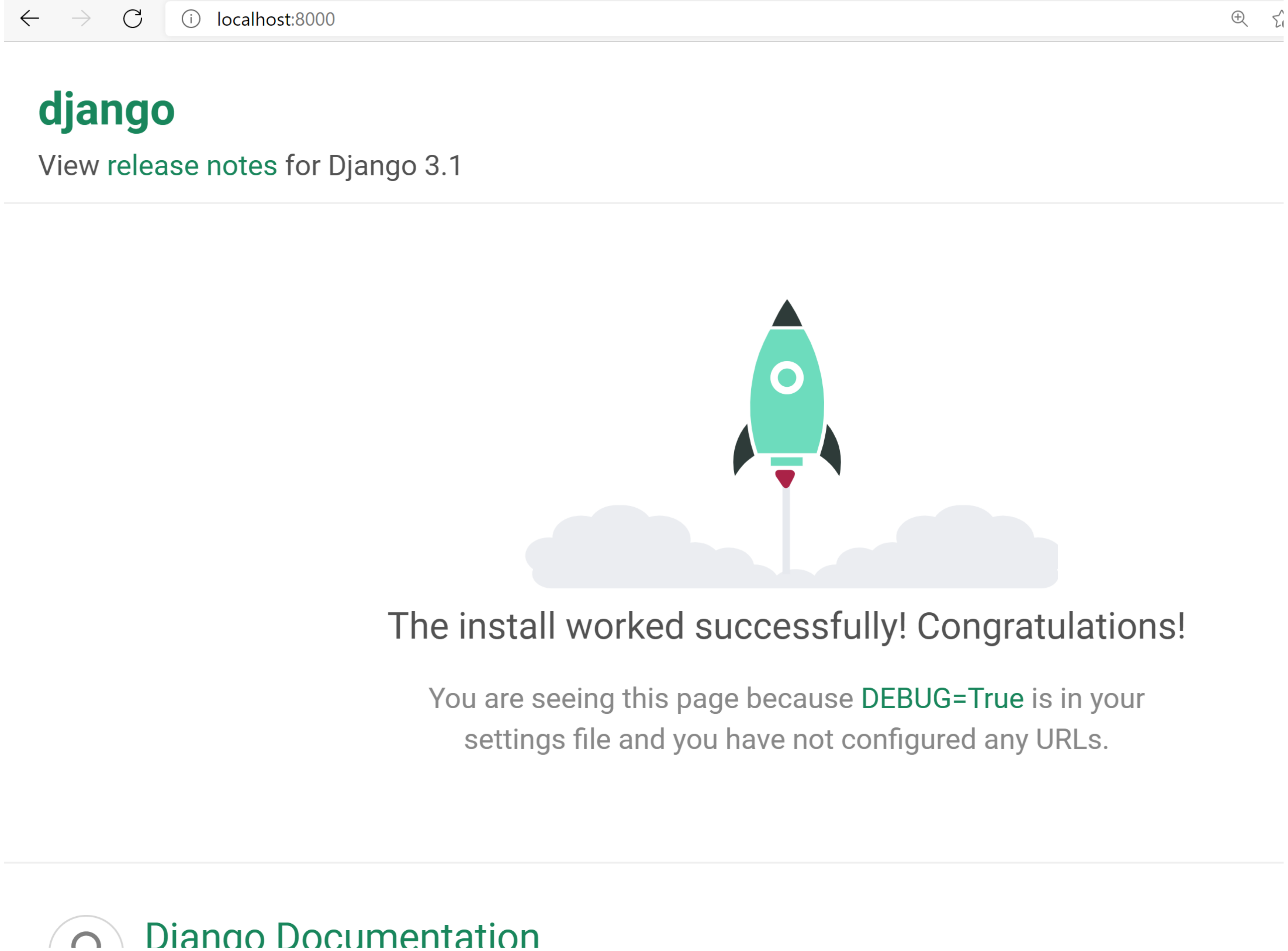


Image source: https://www.youtube.com/watch?v=D6esTdOLXh4

Set-up

- Install latest Python version.
- Install XAMPP
- Open cmd
- >pip install virtualenvwrapper-win // to create virtual environment (The virtual environment is an environment which is used by Django to execute an application. It is recommended to create and execute a Django application in a separate environment.)
- >Create a folder // where we create the projects
- >Projects>python -m virtualenv .
- >.\scripts\activate
- >pip install Django
- >Django-admin startproject djangoproject
- >Cd djangoproject
- >code .
- >python manage.py runserver



localhost:8000

django

View release notes for Django 3.1

The install worked successfully! Congratulations!

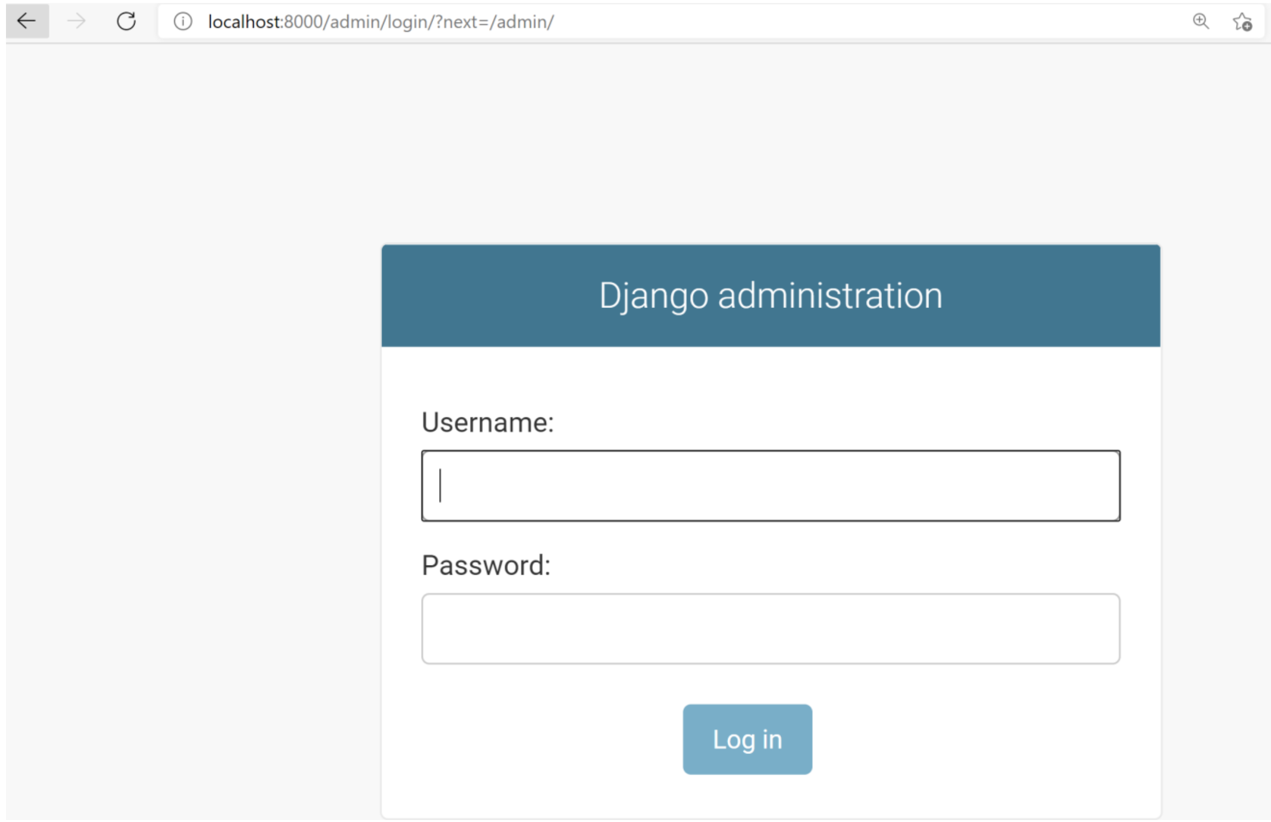
You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Django Documentation

Database Migration and Admin Interface

- >pip install mysqlclient
- Go to settings.py

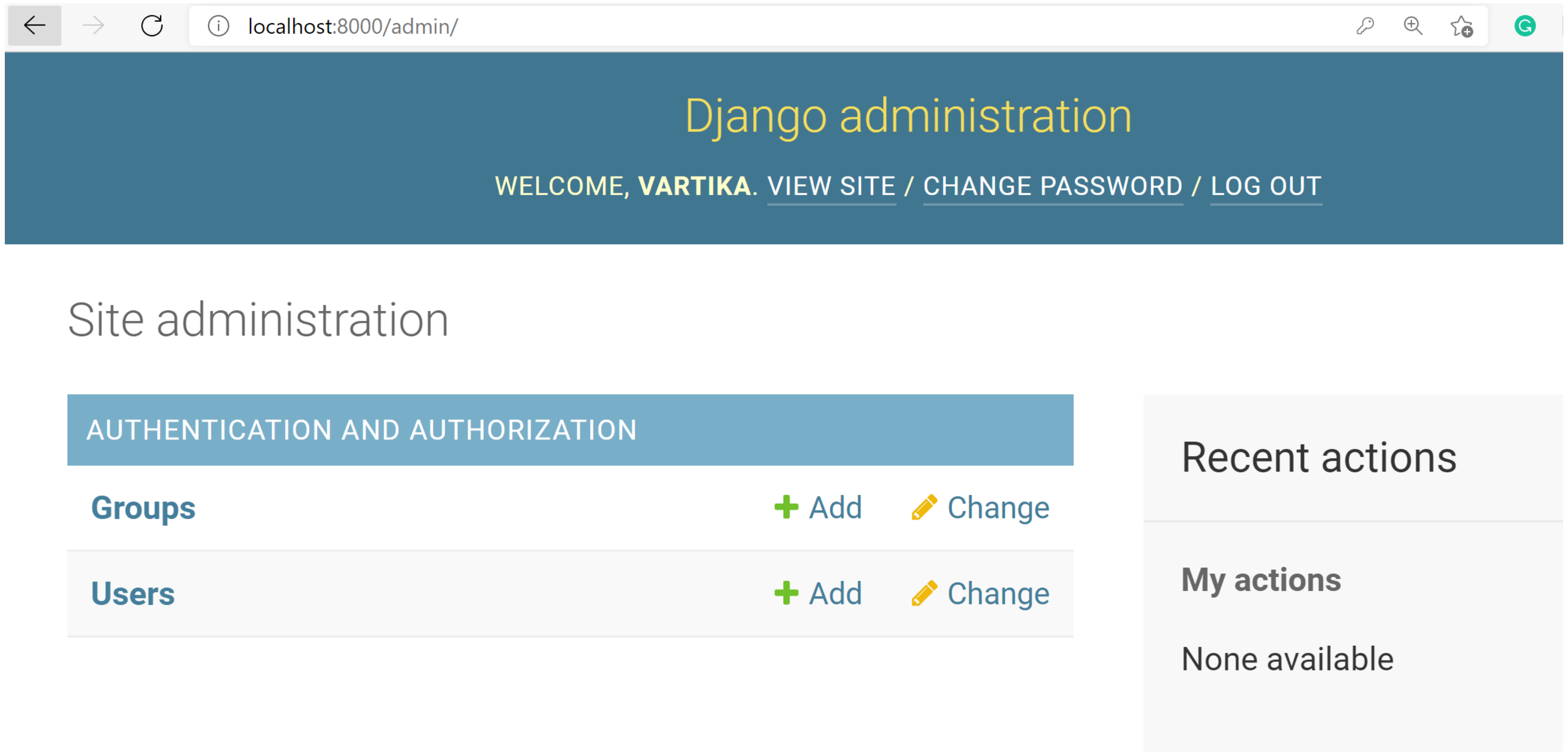
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'djangoapp',
        'USER':'root',
        'PASSWORD':'1234',
        'HOST':'localhost',
        'PORT':''
    }
```



- >python manage.py migrate // to migrate database
- >python manage.py runserver

Create User

- >python manage.py createsuperuser --username=vartika --email=vartika@gmail.com
- Password:
- >python manage.py runserver



localhost:8000/admin/

Django administration

WELCOME, **VARTIKA**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

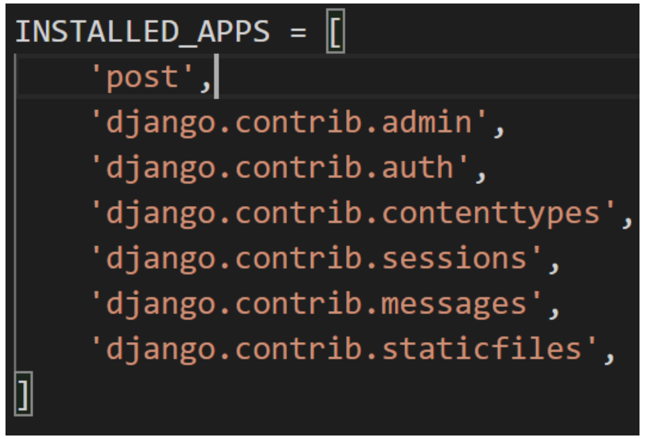
Recent actions

My actions

None available

Django app

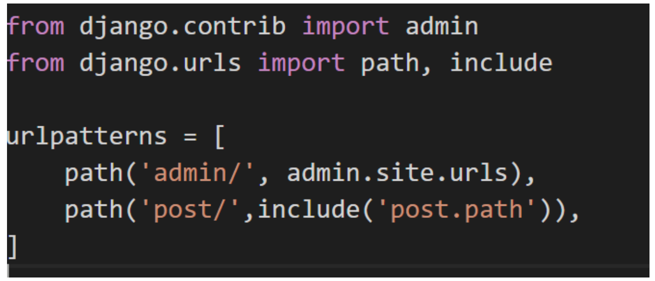
- create app inside the created project.
- a project is a collection of configuration files and apps whereas the app is a web application which is written to perform business logic.
- >python manage.py startapp post
- Go to settings.py in djangoapp, add post



```
INSTALLED_APPS = [
    'post',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

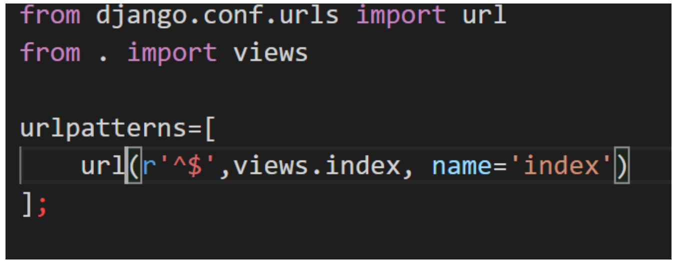
- Include post app in url.py of djangoapp called url mapping

(Django already has mentioned a URL here for the admin, need to add url for post).



```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('post/',include('post.path')),
]
```

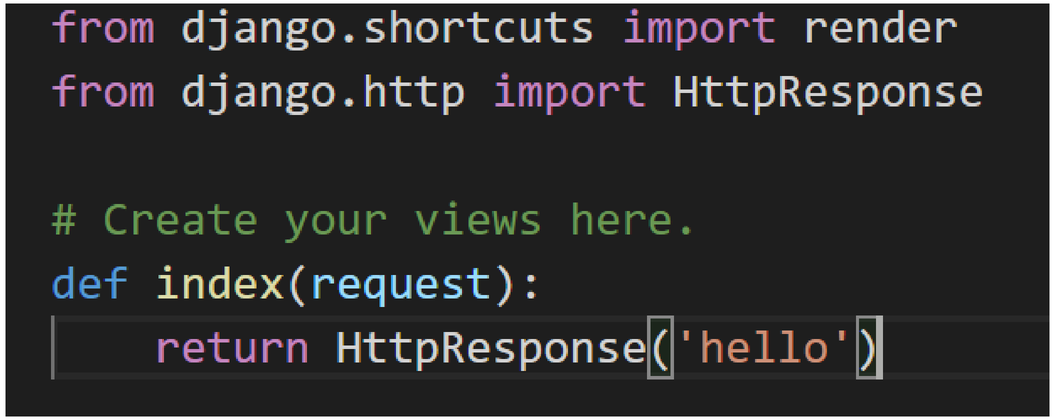
- url.py in post app



```
from django.conf.urls import url
from . import views

urlpatterns=[
    url(r'^$',views.index, name='index')
];
```

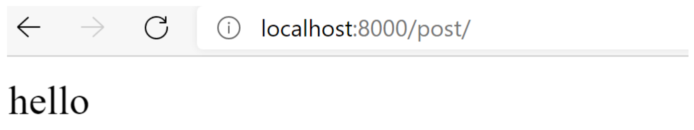
Django Views

- A view is a place where we put our business logic of the application. The view is a python function which is used to perform some business logic and return a response to the user. This response can be the HTML contents of a Web page, or a redirect, or a 404 error.
- All the view function are created inside the views.py file of the Django app.
- Views.py of post app

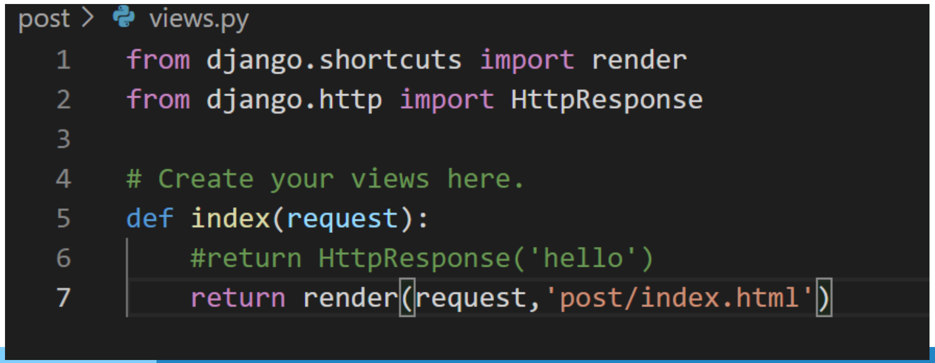


```python
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.
def index(request):
    return HttpResponse('hello')
```

Run the application
>python manage.py runserver

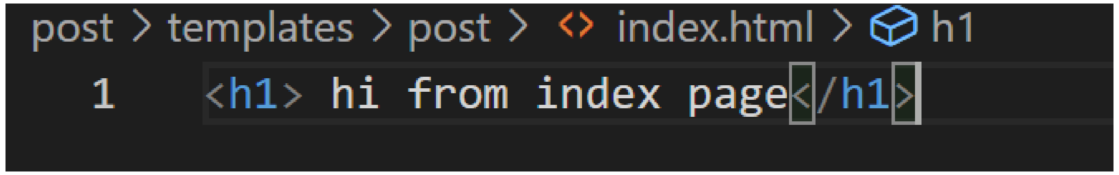


```
localhost:8000/post/

hello
```



post > views.py

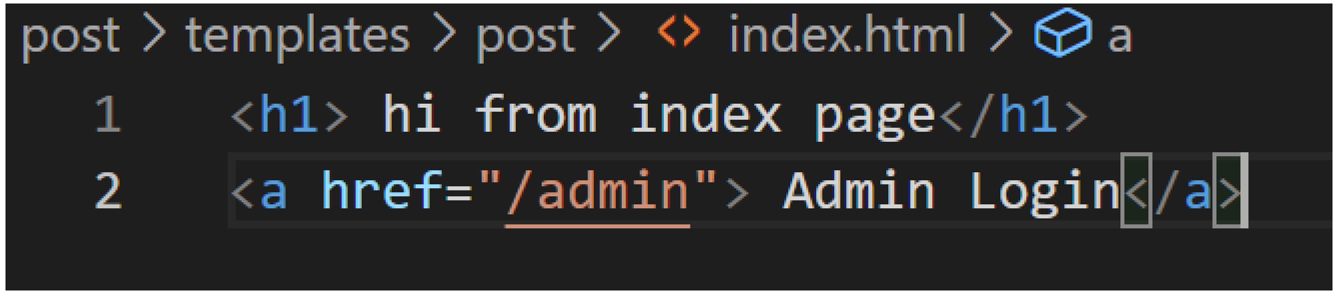
```python
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.
def index(request):
    #return HttpResponse('hello')
    return render(request,'post/index.html')
```



post > templates > post > index.html > h1

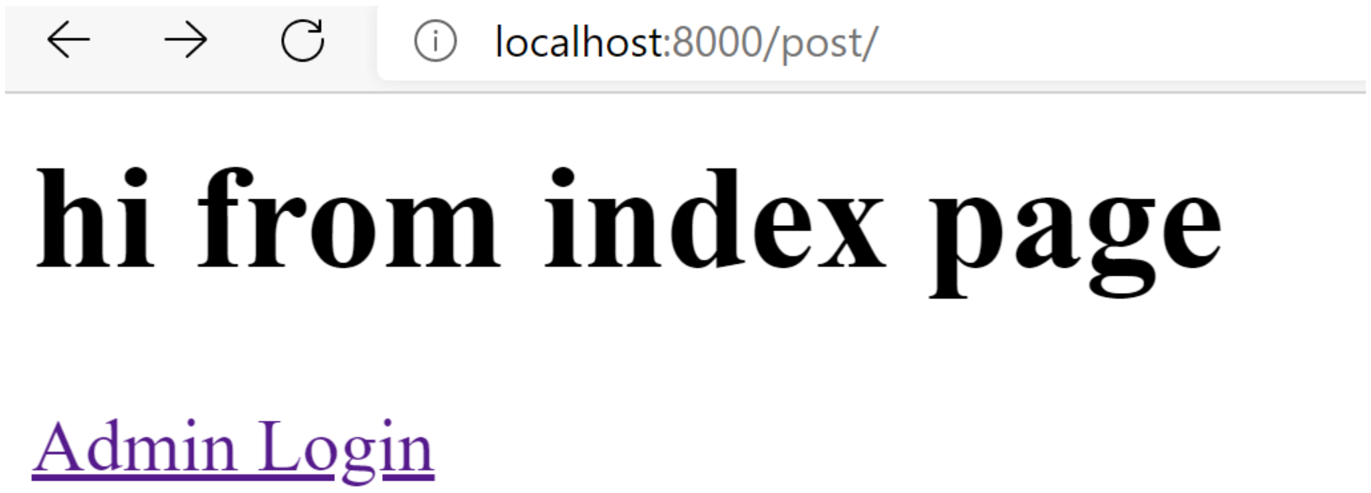
```html
<h1> hi from index page</h1>
```

Add admin interface in Django app



post > templates > post > index.html > a

```
<h1> hi from index page</h1>
<a href="/admin"> Admin Login</a>
```



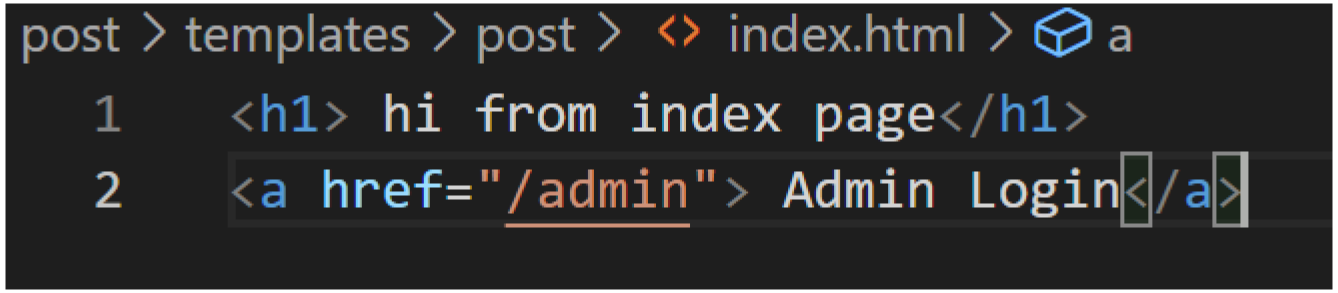
localhost:8000/post/

hi from index page

Admin Login

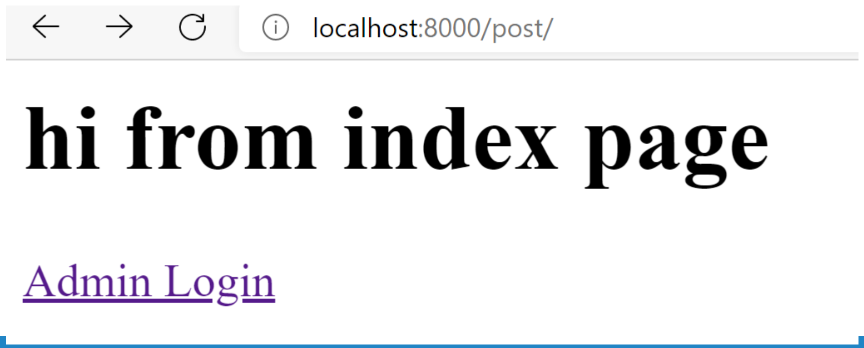
Django Template

- Django provides a convenient way to generate dynamic HTML pages by using its template system.
- A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted
- Django template engine is used to separate the design from the python code and allows us to build dynamic web pages.



post > templates > post > index.html > a

```html
<h1> hi from index page</h1>
<a href="/admin"> Admin Login</a>
```

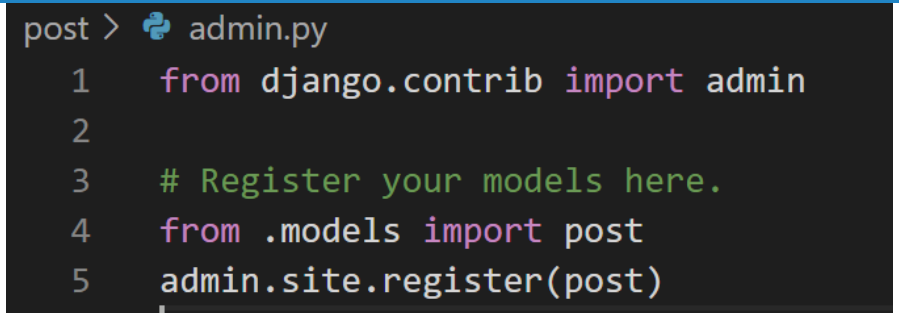


localhost:8000/post/

hi from index page

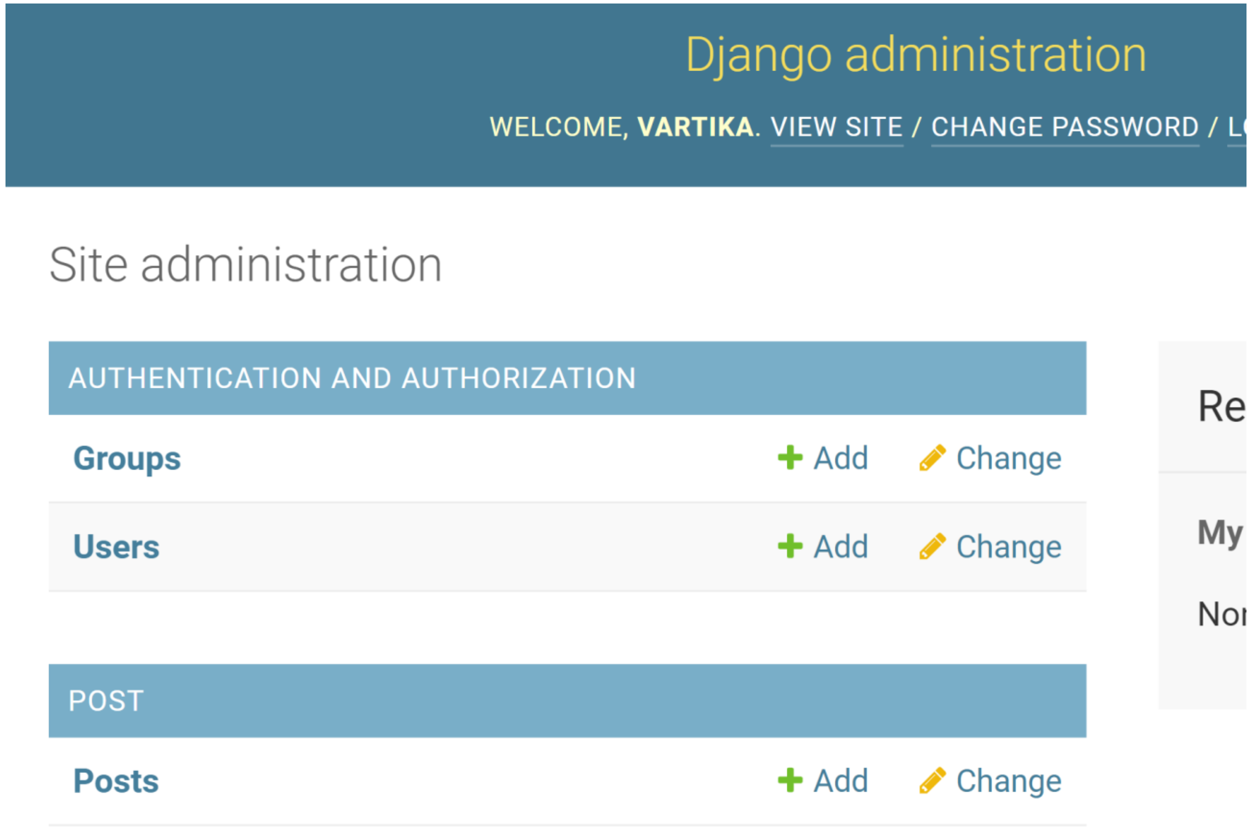
Admin Login

Register your app



post > admin.py

```python
from django.contrib import admin

# Register your models here.
from .models import post
admin.site.register(post)
```



Django administration

WELCOME, **VARTIKA**. VIEW SITE / CHANGE PASSWORD / L

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	Add	Change
Users	Add	Change

POST		
Posts	Add	Change

Re

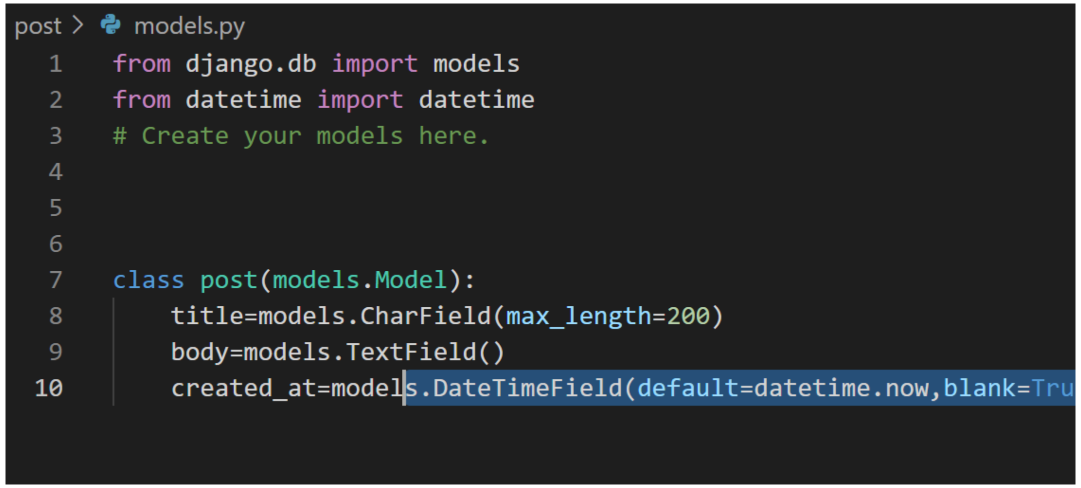
My

Nor

Django Models

In Django, a model is a class which is used to contain essential fields and methods. Each model class maps to a single table in the database.
Django Model is a subclass of **django.db.models.Model** and each field of the model class represents a database field (column).
Django provides us a database-abstraction API which allows us to create, retrieve, update and delete a record from the mapped table.

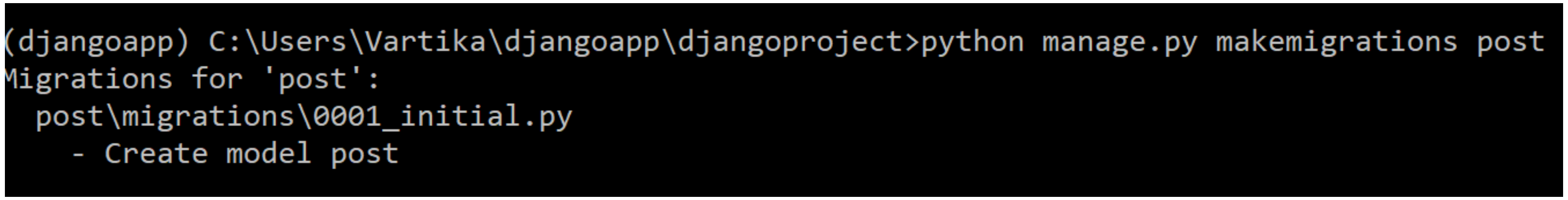
- create a model post that contains the following code in models.py file.



post > models.py

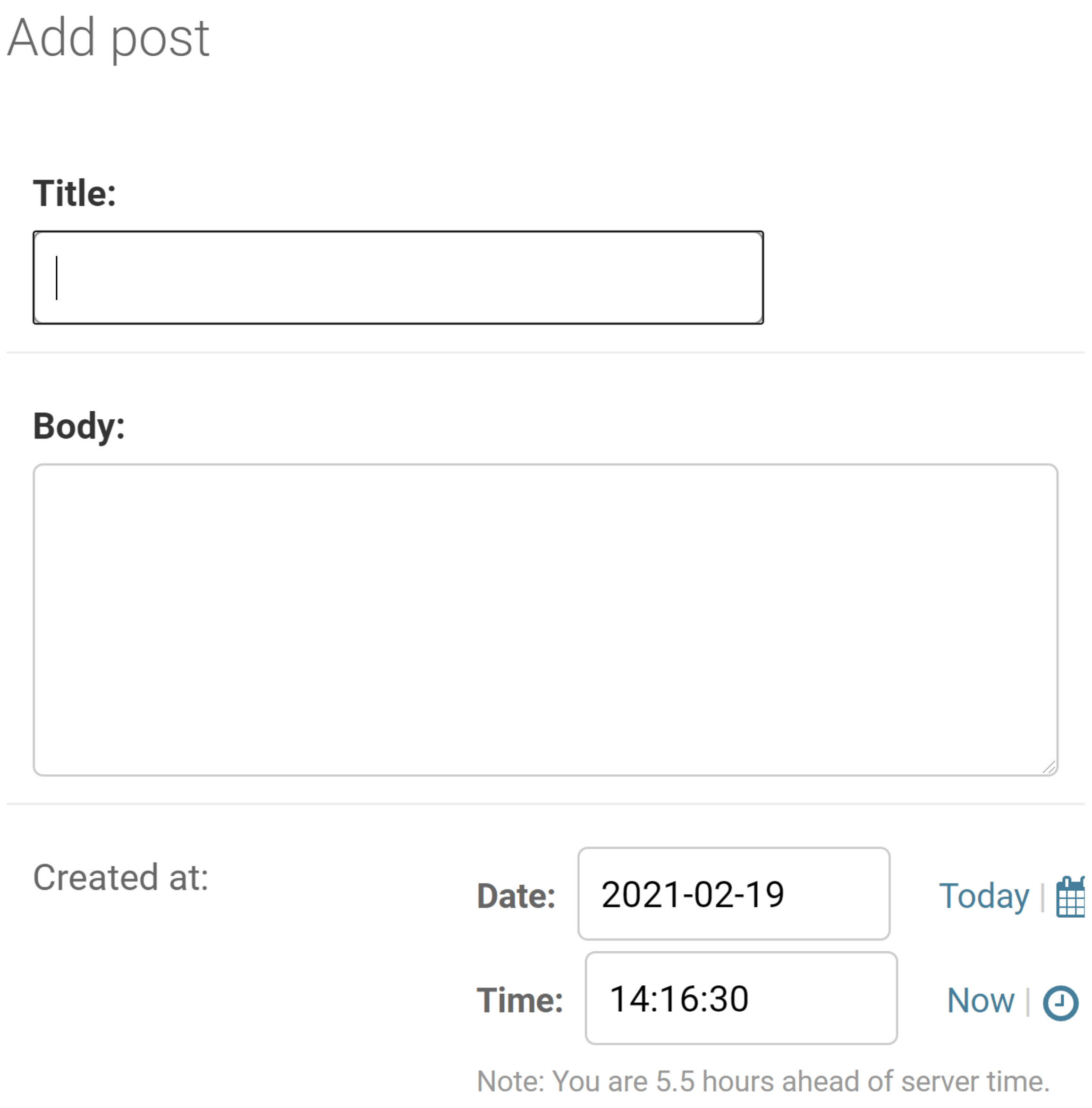
```python
from django.db import models
from datetime import datetime
# Create your models here.



class post(models.Model):
    title=models.CharField(max_length=200)
    body=models.TextField()
    created_at=models.DateTimeField(default=datetime.now,blank=Tru
```

- After that apply migration by using the following command: it will create a file to store data runtime.



```
(djangoapp) C:\Users\Vartika\djangoapp\djangoproject>python manage.py makemigrations post
Migrations for 'post':
  post\migrations\0001_initial.py
    - Create model post
```

Form is created with the fields you have mentioned with complete CRUD functionality


Add post

Title:

Body:

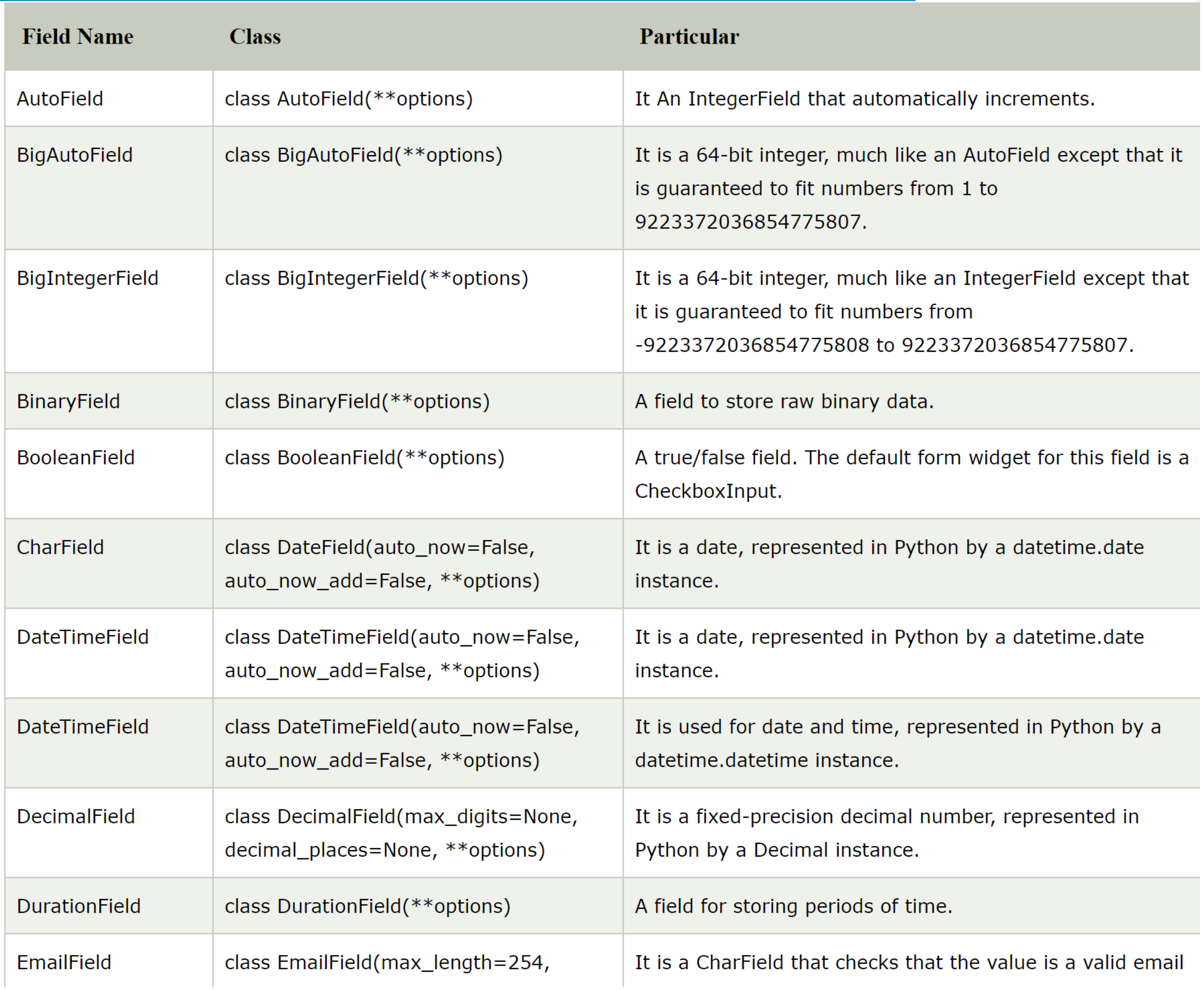
Created at:

Date: 2021-02-19

Time: 14:16:30

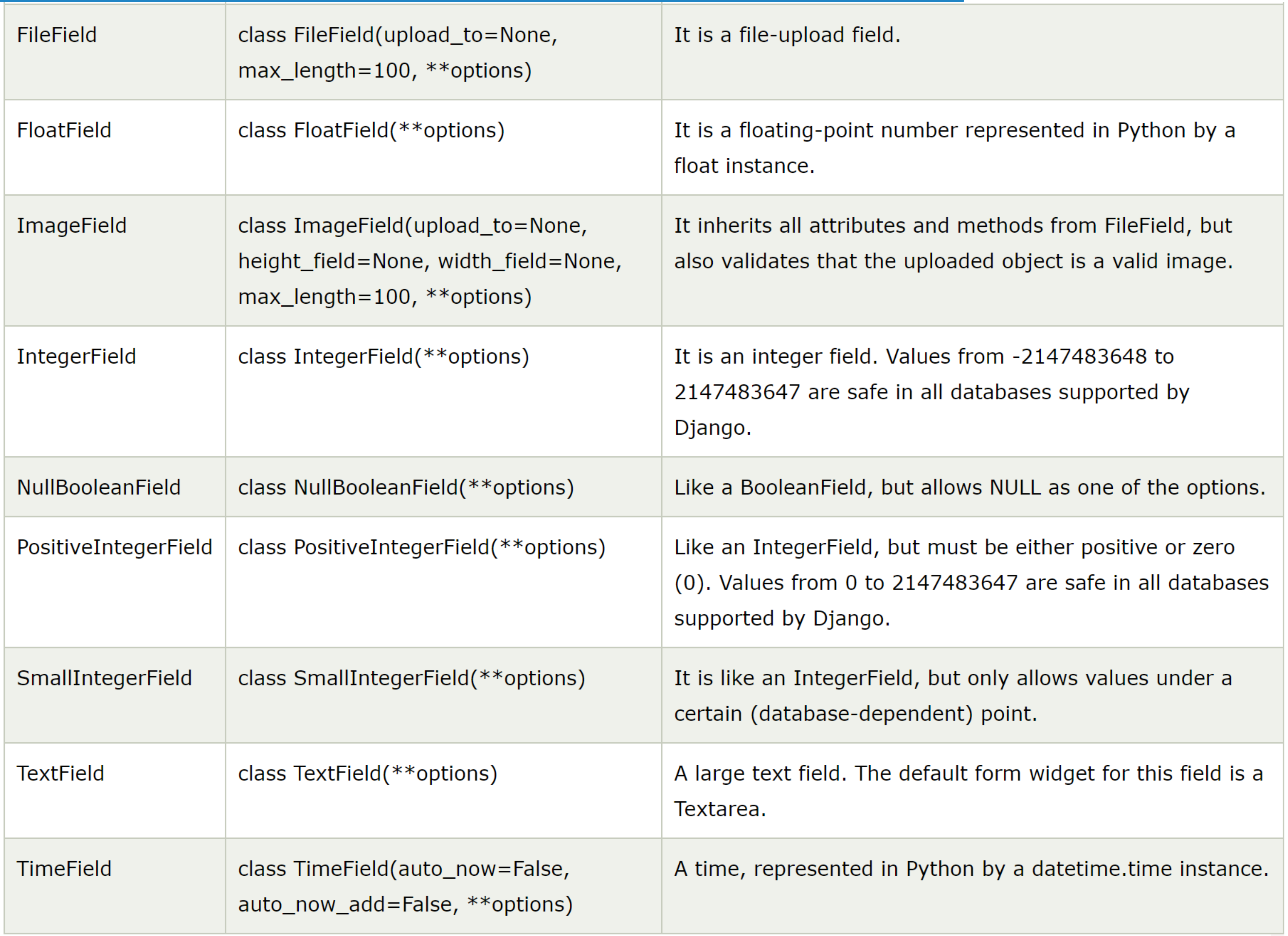
Note: You are 5.5 hours ahead of server time.

Built-in Fields



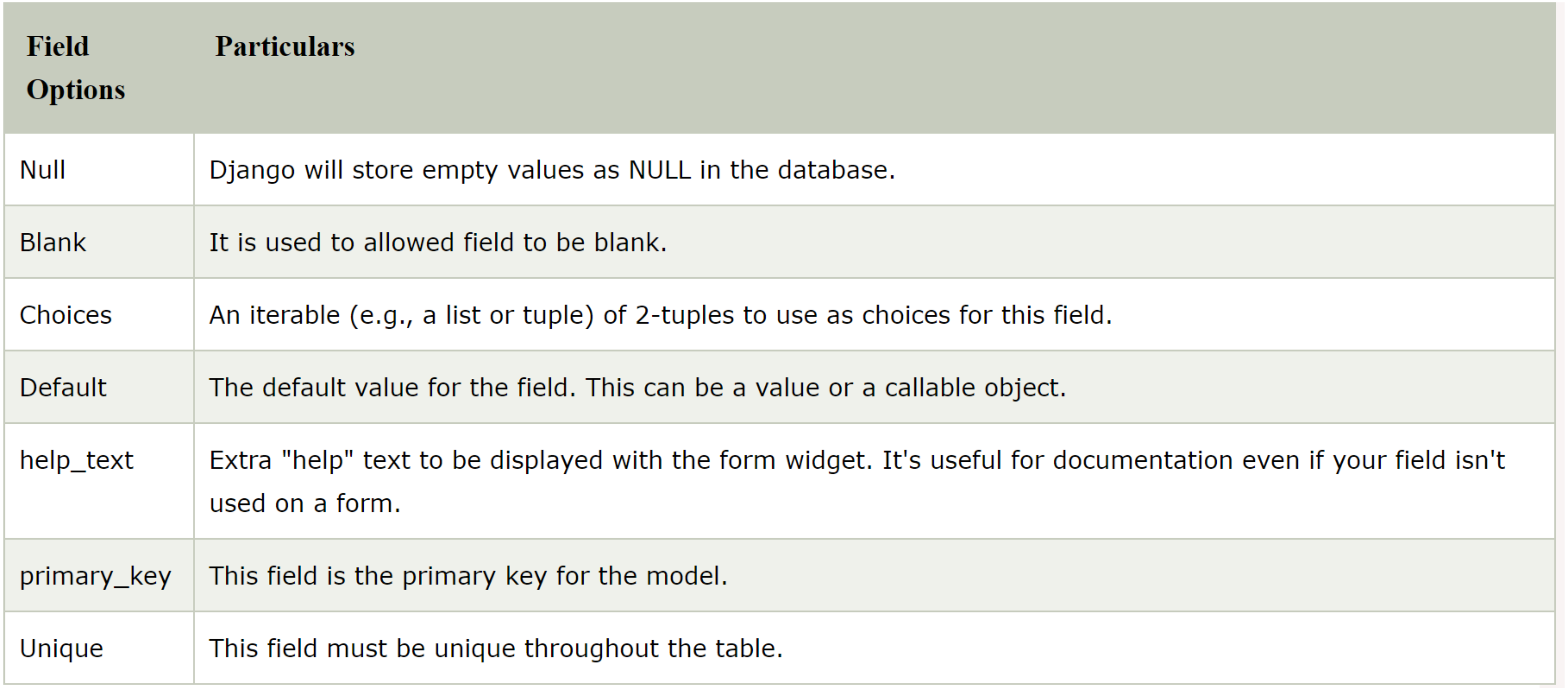
Field Name	Class	Particular
AutoField	class AutoField(**options)	It An IntegerField that automatically increments.
BigAutoField	class BigAutoField(**options)	It is a 64-bit integer, much like an AutoField except that it is guaranteed to fit numbers from 1 to 9223372036854775807.
BigIntegerField	class BigIntegerField(**options)	It is a 64-bit integer, much like an IntegerField except that it is guaranteed to fit numbers from -9223372036854775808 to 9223372036854775807.
BinaryField	class BinaryField(**options)	A field to store raw binary data.
BooleanField	class BooleanField(**options)	A true/false field. The default form widget for this field is a CheckboxInput.
CharField	class DateField(auto_now=False, auto_now_add=False, **options)	It is a date, represented in Python by a datetime.date instance.
DateTimeField	class DateTimeField(auto_now=False, auto_now_add=False, **options)	It is a date, represented in Python by a datetime.date instance.
DateTimeField	class DateTimeField(auto_now=False, auto_now_add=False, **options)	It is used for date and time, represented in Python by a datetime.datetime instance.
DecimalField	class DecimalField(max_digits=None, decimal_places=None, **options)	It is a fixed-precision decimal number, represented in Python by a Decimal instance.
DurationField	class DurationField(**options)	A field for storing periods of time.
EmailField	class EmailField(max_length=254,	It is a CharField that checks that the value is a valid email

Built-in Fields



FileField	class FileField(upload_to=None, max_length=100, **options)	It is a file-upload field.
FloatField	class FloatField(**options)	It is a floating-point number represented in Python by a float instance.
ImageField	class ImageField(upload_to=None, height_field=None, width_field=None, max_length=100, **options)	It inherits all attributes and methods from FileField, but also validates that the uploaded object is a valid image.
IntegerField	class IntegerField(**options)	It is an integer field. Values from -2147483648 to 2147483647 are safe in all databases supported by Django.
NullBooleanField	class NullBooleanField(**options)	Like a BooleanField, but allows NULL as one of the options.
PositiveIntegerField	class PositiveIntegerField(**options)	Like an IntegerField, but must be either positive or zero (0). Values from 0 to 2147483647 are safe in all databases supported by Django.
SmallIntegerField	class SmallIntegerField(**options)	It is like an IntegerField, but only allows values under a certain (database-dependent) point.
TextField	class TextField(**options)	A large text field. The default form widget for this field is a Textarea.
TimeField	class TimeField(auto_now=False, auto_now_add=False, **options)	A time, represented in Python by a datetime.time instance.

Field Options



Field Options	Particulars
Null	Django will store empty values as NULL in the database.
Blank	It is used to allowed field to be blank.
Choices	An iterable (e.g., a list or tuple) of 2-tuples to use as choices for this field.
Default	The default value for the field. This can be a value or a callable object.
help_text	Extra "help" text to be displayed with the form widget. It's useful for documentation even if your field isn't used on a form.
primary_key	This field is the primary key for the model.
Unique	This field must be unique throughout the table.

Useful Reference Links

- https://www.youtube.com/watch?v=D6esTdOLXh4
- https://www.javatpoint.com/django-tutorial