DJANGOFRAMEWORK

Class 1 : Introduction of Django Framework

Framework: Framework is a fundamental structure for developing a software projects. It provides set of predefined tools and libraries that streamline and organize the projects.

Example : 1. Django is used for web development in python.
2. React is used for building user interfaces in javascript.

Django: Free and open source framework for building web apps with python.

It's not only one framework in python:

- Django
- Flask
- Tornado
- Bottle
- Falcon
- Hug

Django is a popular one because it builds a website in less time and less code.

That's why, many companies are using Django like:

1. You Tube
2. Instagram
3. Spotify
4. Drop box

Django we can also call it as, Batteries included framework which means lot of features are out of the box. So, we don't want to code them from the scratch.

Django features:

- Admin site --> To managing a data. It is a huge time saver.
- Object-Relational Mapper(ORM) --> It abstract the database so we can code this process the data without writing a lot of SQL queries.
- Authentication --> It is used for validation purpose.

Here, Django comes with a lot of features...

But you don't have to learn and use them all. Because, all these features are all optional.

Django framework: Django is a python based high level web framework and mainly is used to develop a rapid, flexible and complex web application. Which is used **MVT design pattern**.

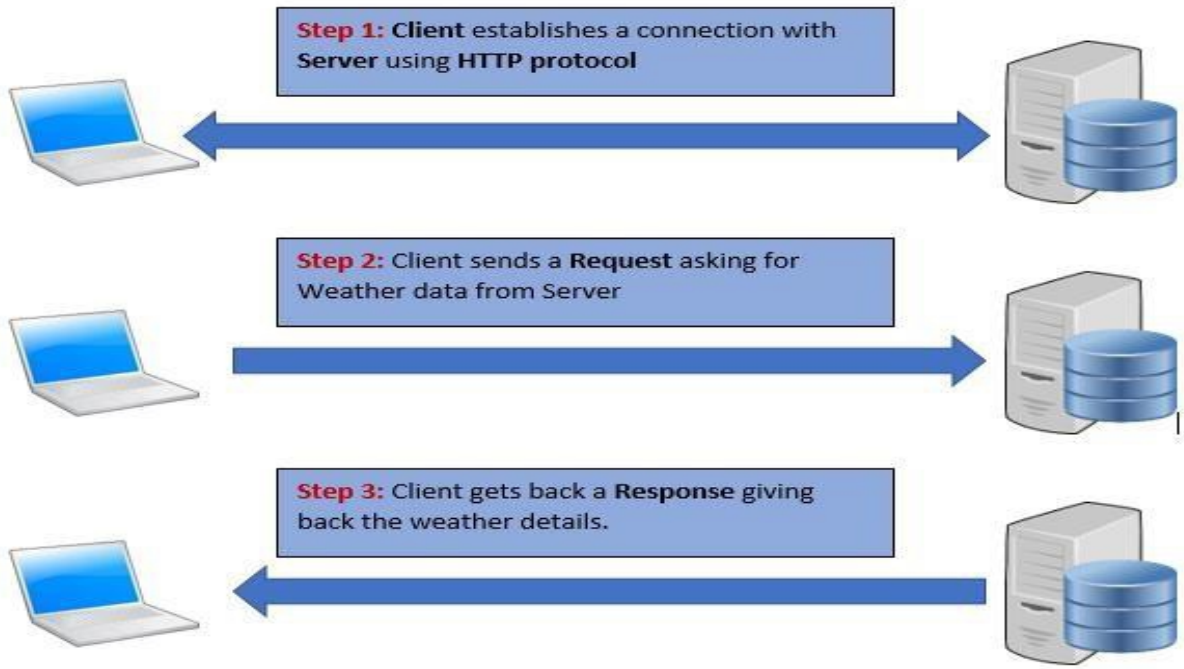
Django framework was developed by **Adrian Holovaty.**

Client server architecture: **HTTP**

Step1: Client establish a connection with server using Http Protocol.

Step 2: Client sends a request asking for data from server.

Step 3: Client gets back a response giving back the details.



Pip(Preferred install Program):

---->Goto search/type cmd

C:\Users\KITS>**mkdir DjangoApplications**

C:\Users\KITS>**cd DjangoApplications**

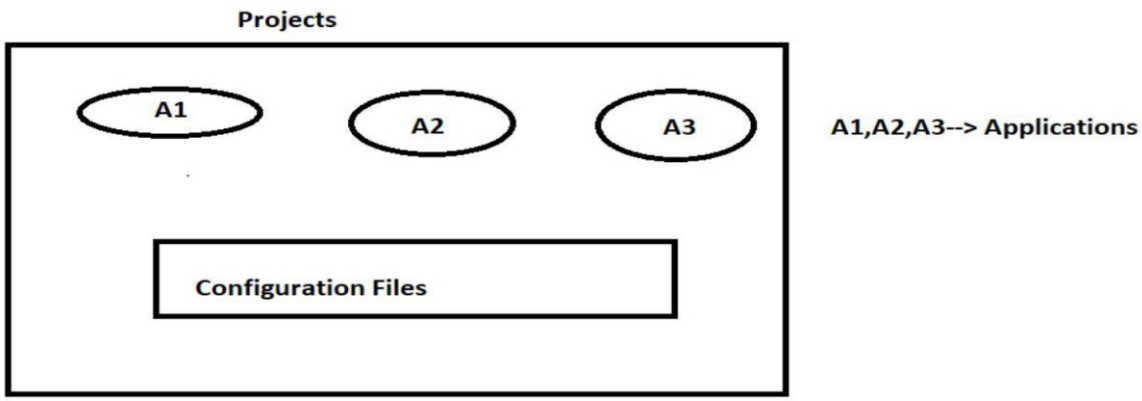
C:\Users\KITS>**DjangoApplications>cd..**

C:\Users\KITS>**pip install Django**

C:\Users\KITS>**django-admin –version**

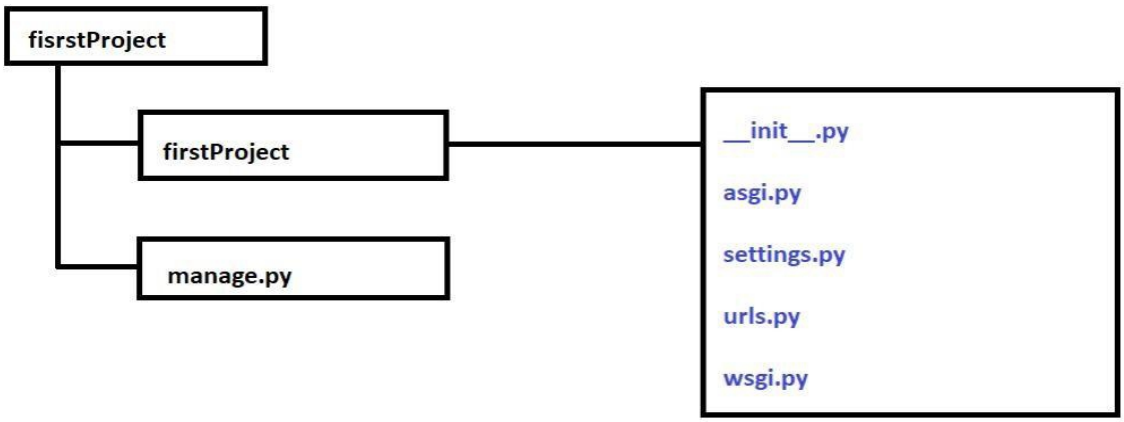
C:\Users\KITS>**cd DjangoApplications**

Django Projects and Applications:



Step 1: Create project using following command.

django-admin startproject firstProject



1. **__init__.py :** initialize the package.
2. **Asgi.py:** asgi (Asynchronous server gateway interface).

   this file is responsible for configuring the asgi application.
3. **Se翻 ng.py**This is the file is used to configure an application.

This file contains following entries.

- Installed_apps: We will add our application in this entry.
- Middleware: used as processing unit between client and server.
- Templates: is used to render the response to end user.
- Database: used to specify database. By default Django provides a database called db.sqlite3.
- Auth_password_validator: used for Validation purposes.

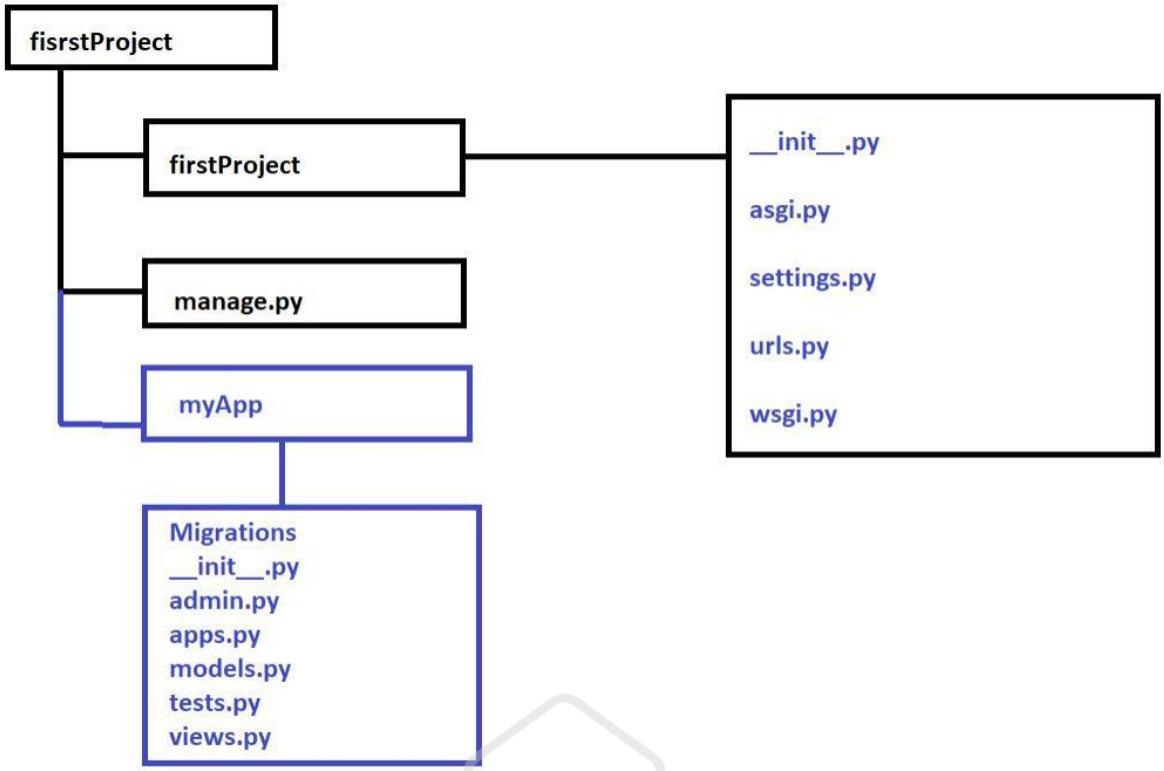
4. Urls.py: contains URL pattern CORRESPONDING to each view.
5. Wsgi.py: wsgi (Web server gateway interface) it is a standard component of django configured to use wsgi.

It serves to serve your projects.

Step 2: Create an application using following command.

C:\Users\KITS**DjangoApplications>cd firstProject**

python manage.py startapp myApp



Migration folder is used for database specific informations**.**

Admin.py: is used to register our application to admin interface.

Apps.py: Application specific configuration.

Models.py: we store application specific data models and it is used for database operations.

Tests.py: used to test our applications.

Views.py : used to write business logic or functions

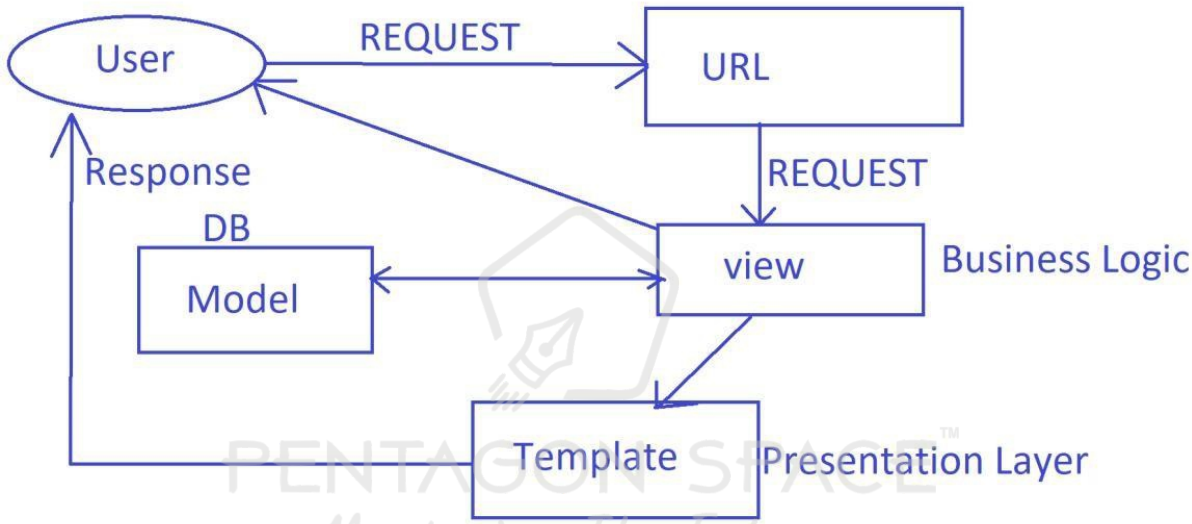
Class 2 : MVT Architecture

MVT is a **software design pattern** and it has 3 components **Model, view, Template.**

Model: The Model helps to **handle the database**.

Template: **Handles user interface**.

View: Execute the Business logic and **interact with the model**.



How to Run Django Server?

http://127.0.0.1:8000

127.0.0.1 Local Host(Base URL)

This url along with endpoint should be loaded from the browser

http://localhost:8000

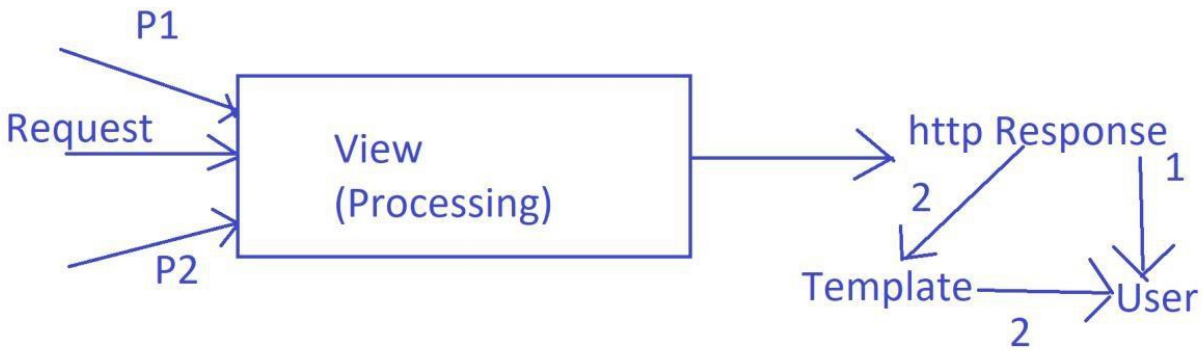
8000→ Default port number to run Django Applications

80**00**→ Default start number

Port Number	Protocol	Application
20	TCP	FTP data
21	TCP	FTP control
22	TCP	SSH
25	TCP	SMTP
53	UDP, TCP	DNS
80	TCP	HTTP (WWW)
110	TCP	POP3
443	TCP	SSL

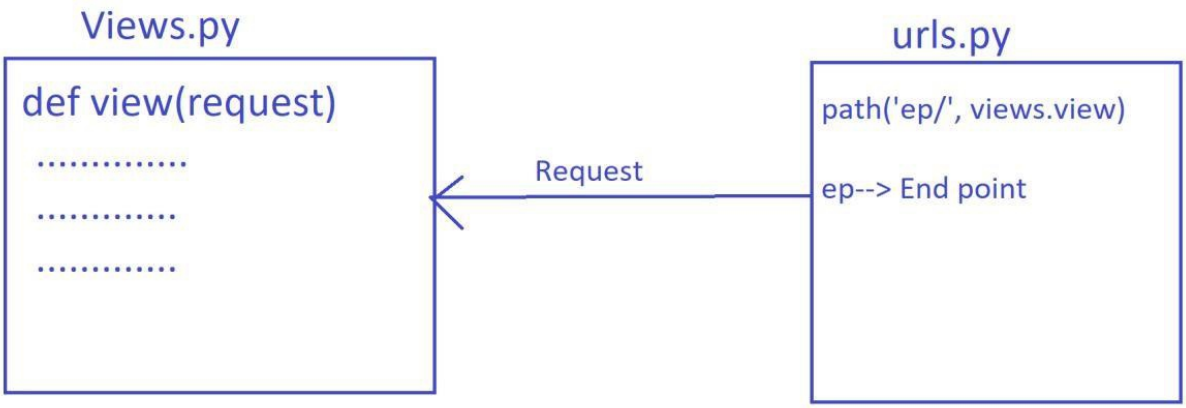
Creating views or Business Logic:

- Each view is a function that specifies business logic, used to perform a particular task.
- Each view should accept at-least a parameter called Request.
- Request specifies http request object.
- View accept request(http request) as a input and it will process the request, and response is generated.
- This Response can be directly sent to output.(html file).



Creating url pattern:

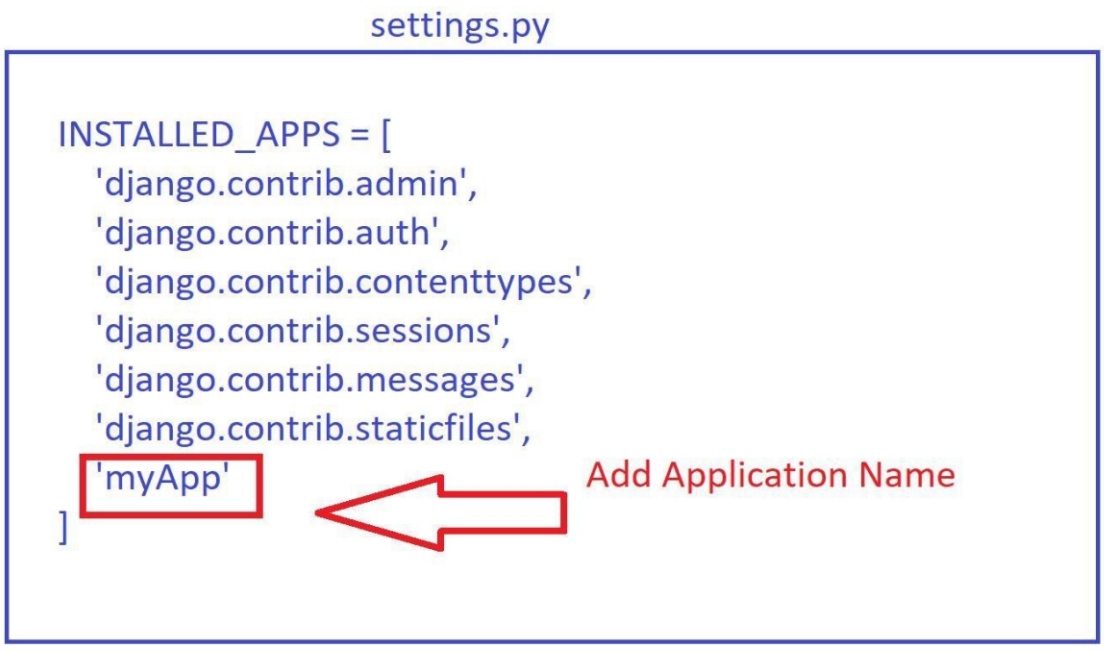
url pattern specifies End point. This end point is associated with corresponding view. Through the url request is accepted.



Manage.py: Command line utility to interact with Django project in different ways like to create an application, to run development server, to create migrations **etc**.

➔Go to sublime text, choose file

option→openfolder→fisrtProject(outer level)



Views.py

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
def view1(request):
    s="Welcome to Python Webdevelopment"
    return HttpResponse(s)
```

View

urls.py

```
from django.contrib import admin
from django.urls import path
from myApp.views import view1

urlpatterns = [
    path('response1/', view1),
]
```

End Point

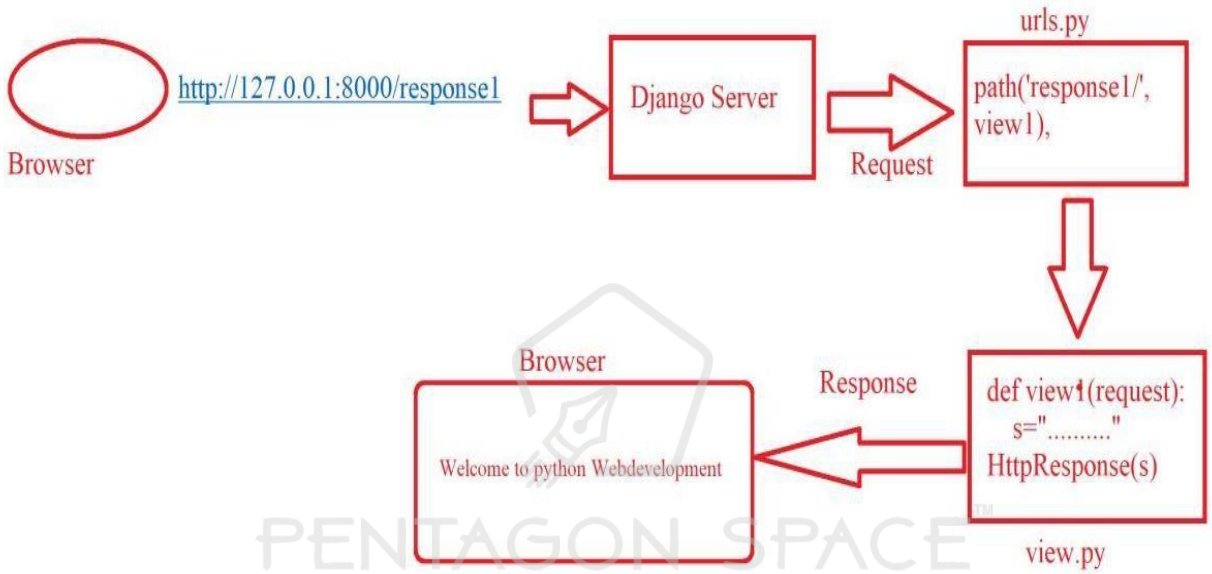
Class 3 :

Run the server by using command:

Copy the link http://127.0.0.1:8000/

Now load this link on the browser window along with the endpoint

http://127.0.0.1:8000/response1



We can write urls.py in the following way also:

```
from django.contrib import admin
from django.urls import path
from myApp import views
urlpatterns = [ path('response1/', views.view1), ]
```

Execution Steps:

Step 1: Create a folder to hold all Django projects by using mkdir

djangoProjects

Step2: change the directory to djangoProjects

Cd djangoProjects

Step3: Create a Django project

Django-admin startproject firstProject

Step 4: create an application inside firstProject

Cd firstProject py manage.py startapp myApp

step5: goto **se翩 ngs.py**and add application to **INSTALLED_APPS list, edit views.py add pattern in urls.py step6:** Run the development server **py manage.py runserver**

step7: copy paste the http link by **appending the endpoint** used **urls.py**

Creating Multiple views inside a single Application:

1. Django-admin startrproject secondProject
2. cd secondProject
3. py manage.py startapp myApp

se翩 ngs.py:

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

Views.py:

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

# Create your views here.
def view1(request):
s="<h1>This is 1st Response"
return HttpResponse(s) def
view2(request):
n1=int(input("Enter 1st num:"))
n2=int(input("Enter 2nd num:"))
n3=n1+n2 return
HttpResponse(str(n3))
```

urls.py:

```
from django.contrib import admin
from django.urls import path
from myApp.views import*
urlpatterns = [
    path('admin/', admin.site.urls),
    path('response1/',view1),
    path('response2/',view2),
    ]
```

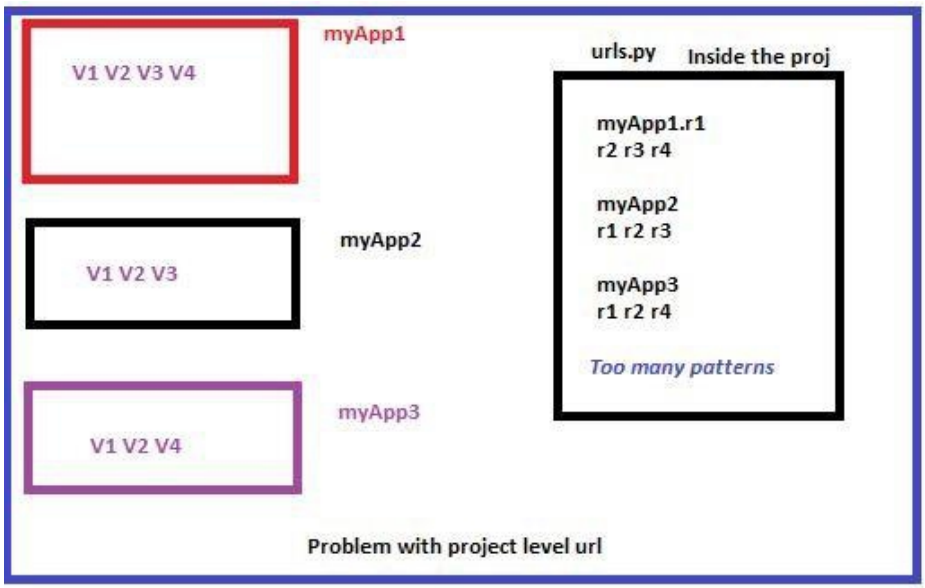
Run server: py manage.py runserver 8002

http://127.0.0.1:8002/response1/ → **This is 1st Response**

http://127.0.0.1:8002/response2/ → **500**

Creating url pattern at application level:

If we have multiple applications inside a project with multiple views,creating url patterns inside project level urls would be difficult to maintain.To avoid this we can create at application level.



Step 1: django-admin startproject thirdProject

Step 2: cd thirdProject

Step 3: py manage.py startapp myApp

Step 4:Goto se翮 ngs.py and add the application Step

4:Create a file in myApp,urls.py myApp/urls.py from

django.contrib import admin from django.urls import

path from myApp.views import * urlpatterns = [

path('admin/', admin.site.urls),

path("response1/",view1), path("response2/",view2)

]

views.py

from django.shortcuts import render from

django.http import HttpResponse # Create

your views here. def view1(request):

s="<h1>This is 1st Response" return

HttpResponse(s)

def view2(request): s="<h1>This is 2nd Response" return

HttpResponse(s) Project level urls.py[main urls.py] from

django.contrib import admin from django.urls import

path,include urlpatterns = [

path('admin/', admin.site.urls),

path('response/',include('myApp.urls'))

]

py manage.py runserver open browser and put request

http://127.0.0.1:8000/response/response1/

http://127.0.0.1:8000/response/response2/

Class 4 : A Project with Multiple Application

- cd djangoApplications
- django-admin startproject fourthProject
- py manage.py startapp myApp1
- py manage.py startapp myApp2 se翩 ngs.py:

```
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'myApp1',
'myApp2'
]
```

myApp1/views.py:

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
def view1(request):
s="Inside myApp1"
return HttpResponse(s)
```

myApp2/views.py:

```
from django.shortcuts import render from django.http import HttpResponse
# Create your views here. def view1(request):
```

s="<h1>Inside myApp2<h1>"

return HttpResponse(s)

urls.py:

from django.contrib import admin

from django.urls import path

from myApp2 import views from

myApp1 import views urlpatterns

= [path('admin/', admin.site.urls),

path('response1/',views.view1),

path('response2/',views.view1)

]

py manage.py runserver error:

Because of name collision

Solution: Aliasing from

django.contrib import admin from

django.urls import path from

myApp1 import views as v1 from

myApp2 import views as v2

urlpatterns = [path('admin/',

admin.site.urls),

path('response1/',v1.view1), path('response2/',v2.view1)

]

http://127.0.0.1:8000/response1/ Inside myApp1

http://127.0.0.1:8000/response2/ Inside myApp2 **TEMPLATES IN DJANGO:**

Templates means an HTML File

Step 1: At project level, create a folder by name, templates

templateProject ----manage.py

----myApp.py

----templateProject

---- templates

Step 2: Inside templates, creates a folder by name myApp(with application name)

TemplateProject

----manage.py

----myApp

----templateProject

----templates myApp

Step 3: Inside myApp file we can create one more html files

C:\Users\Dell\DjangoApplications>django-admin startproject templateProject

C:\Users\Dell\DjangoApplications>cd templateProject

C:\Users\Dell\DjangoApplications\templateProject>py manage.py startapp myApp

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

```
from pathlib import Path
import os
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
TEMPLATE_DIR=os.path.join(BASE_DIR,'templates')
```

To the existing BASE_DIR path,join the path related to 'templates'

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplat
        'DIRS': [TEMPLATE_DIR],
        'APP_DIRS': True,
        'OPTIONS': {
```

views.py

```
from django.shortcuts import render
# Create your views here.
def template_view(request):
        return render(request,'myApp/1.html')
```

urls.py

```
from django.contrib import admin
from django.urls import path
from myApp.views import template_view
urlpatterns = [
        path('admin/',admin.site.urls),
        path('response/',template_view)
        ]
```

templates/myApp/1.html

```
<html>
	<center>
		<body bgcolor="yellow">
			<h1 style="color: blue;">First HTML Page of DJANGO</h1>
		</body>
	</center>
</html>
```

pymanage.pyrunserver

Class 5 :

```
from django.shortcuts import render
# Create your views here.
def template_view(request):
        return render(request,'myApp/1.html')
```

Render →is used to render the request to the given template[1.html]

render(request,template path,context)

context is a dictionary:contains key value pairs

```
from django.shortcuts import render #
Create your views here.
def template_view(request):
    name=”dhoni”
    id=101
    place=”B’lore”
    return render(request,'myApp/1.html',context={‘key1’:name,’key2’:id,’key3’:place})
    or
    return render(request,'myApp/1.html',{‘ key1’:name,’key2’:id,’key3’:place}) this
```

context would be sent to 1.html

1.html

To fetch the values, Jinja 2 syntax

{{key1}}→ name dhoni

{{key2}}→ id 101

{{key3}}→ place B’lore

---

views.py

```
from django.shortcuts import render

# Create your views here.
def template_view(request):
    name="Dhoni" id=101 place= "Blore "
    context={"key1":name,"key2":id,"key3" :place}
    return render(request,'myApp/1.html',context )
```

1. html

```
from django.shortcuts import render #
Create your views here.
def template_view(request):
        name=   "Dhoni   "   id=101   place="Blore"
        context={"key1":name,"key2":id,"key3"  :place}
        return render(request,'myApp/1.html',context )
```

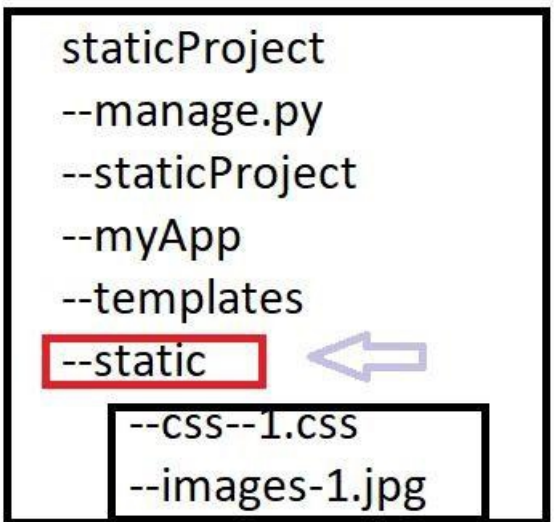
Static Files In Django:

Note:Jinja syntax uses templating language that contains variables and programming logics.
The variables or programming logics are placed inside delimiters
Ex:To fetch any result or data we use {{data}},For expressions we use {%expression%}

```
{%for i in sequesnce%}
    Action1
    Action2
{%endfor%}
```

```
{% if condition %}
        Action1
        Action2
{%else%}
        Action3
        Action4
{%endif%}
```


Step 1: At project level,create a folder static,Inside this folder create a folder by name css Inside css and images>



C:\Users\Dell\DjangoApplications>django-admin startproject staticProject

C:\Users\Dell\DjangoApplications>cd staticProject

C:\Users\Dell\DjangoApplications\staticProject>py manage.py startapp myApp

Step 2: se翩 ngs.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myApp'
]
import os
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
TEMPLATE_DIR=os.path.join(BASE_DIR,'templates')
STATIC_DIR=os.path.join(BASE_DIR,'static')

TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [TEMPLATE_DIR],
  .....
.....
}
```

At the end add the following line,

```
STATICFILES_DIRS=[STATIC_DIR] .....
.....
.....
```

1.html

```
{% load static %}
<head>
<link rel="stylesheet" href="{% static "css/1.css" %}"/>
<img src="{% static "images/1.jpg" %}" alt="....."/>
</head>
```

views.py

```
from django.shortcuts import render
# Create your views here.
def my_view(request):
        myName="Harish"
        favPlayer="Dhoni"
        favAnimal="Lion"
        favSub="Python"
        d={"name":myName,"player":favPlayer,"animal":favAnimal,"subject":"favSub"} return
        render(request,"myApp/1.html",d)
```

urls.py

```
from django.contrib import admin
from django.urls import path from
myApp.views import my_view
urlpatterns = [ path('admin/',
admin.site.urls),
        path('response/',my_view)
        ]
```

templates/myApp/1.html

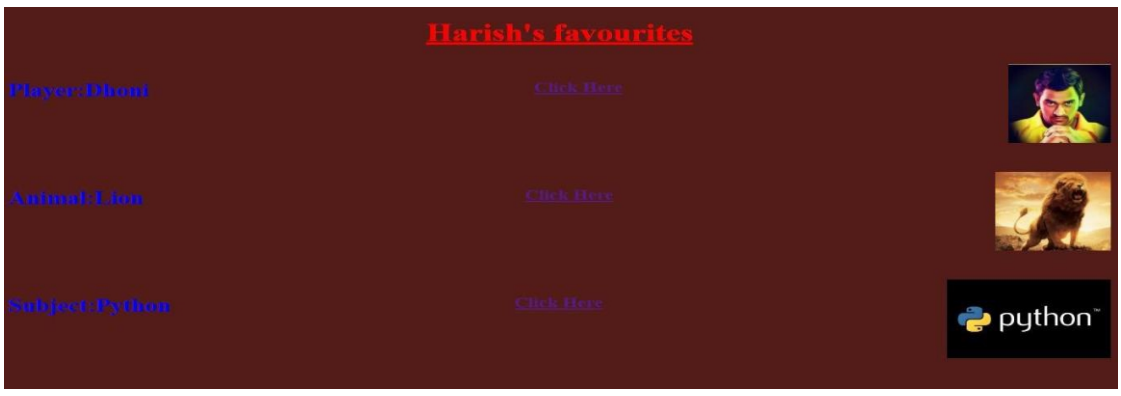
```
<!DOCTYPE html>
{%load static%}
<html lang="en" dir="ltr">
  <head> <link rel="stylesheet"href="{% static "css/1.css" %}">
  </head>
  <body bgcolor=#541c18>
    <center>
      <h1><u>{{name}}'s favourites</u></h1>
    </center>
    <div>
      <h2>Player:{{player}}</h2>
      <h3><a href="https://en.wikipedia.org/wiki/MS_Dhoni">Click Here</a></h3>
      <img src="{% static "images/dhoni.jpg" %}" alt="not available">
    </div>
```

```
    <br><br>
    <div>
      <h2>Animal:{{animal}}</h2>
      <h3><a href="https://en.wikipedia.org/wiki/Lion">Click Here</a></h3>
      <img src="{% static "images/lion.jpeg" %}" alt="not available">
    </div>
    <br><br>
    <div>
      <h2>Subject:{{subject}}</h2>
      <h3><a href="https://www.python.org/">Click Here</a></h3>
      <img src="{% static "images/python.jpg" %}" alt="not available">
    </div>
    <br><br>
  </body>
</html>
```

static/css/1.css

```
h1
{
color: red;
}
h2
{
color: blue;
}
img{
height:100px;
width:200px;
border:2px solid black;
}
h3
{
color:green;
} div{
display:flex;
justify-content: space-between;
}
```

py manage.py runserver



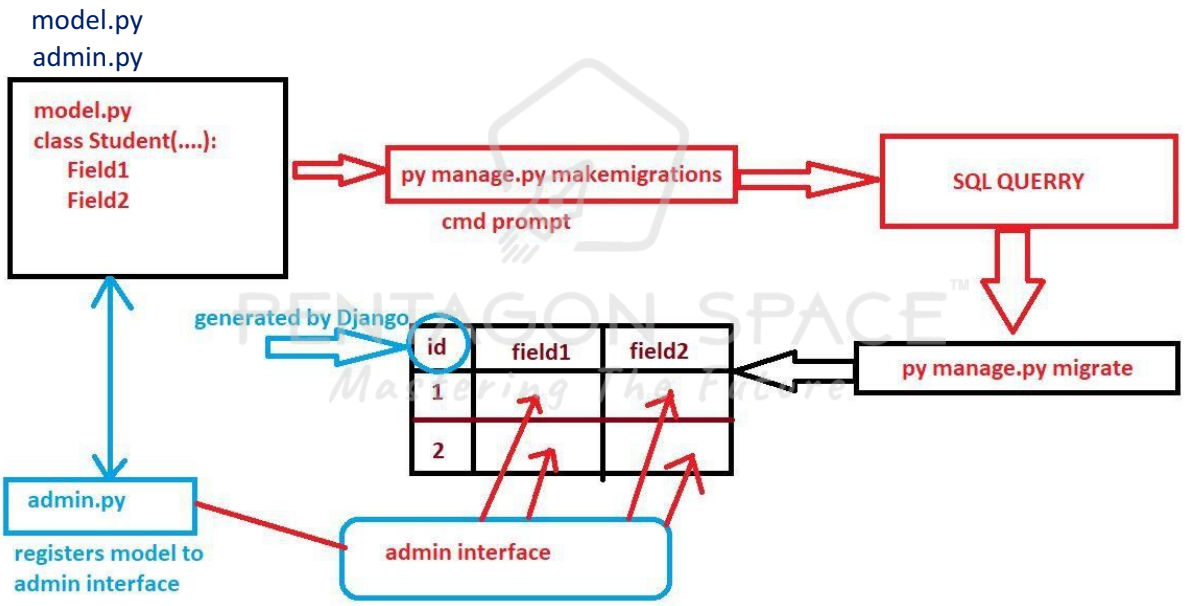
Class 6 : Models in Django

In Relational database we store the information in the form of a table, in which each row is referred to as record.

Each heading is called a field.

Name	Id(Primary Key)	Place
Rohan	1	Bangalore
Radha	2	Pune
Rosh an	3	Mumbai

Each record is identified with a unique number called primary key



C:\Users\KITS>cd DjangoApplications

C:\Users\KITS\DjangoApplications>Django-admin startproject modelProject1

C:\Users\KITS\DjangoApplications>cd modelProject1

C:\Users\KITS\DjangoApplications\modelProject1>py manage.py startapp myApp

Se翻 ngs.py

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

from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
TEMPLATE_DIR=os.path.join(BASE_DIR,'templates')

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATE_DIR],
```

Create templates/myApp/

In order to create a model class we must inherit the behaviors of Django.db.models.Model

Integer data→IntegerField(),

Float data→FloatField(),

Char data→CharField(max_lenght=30)

Max length→ max no of chars

models.py

```
from django.db import models

# Create your models here.
class Student(models.Model):
    number=models.IntegerField()
    name=models.CharField(max_length=40)
    marks=models.FloatField()
```

admin.py

This file is used to register our model to admin interface, here we specify the fields that created in models inside a class.

This class contains the fields, along with the class that we have created inside models.py should be registered.

Step1 : Import the required class from the models.py

Step2 : create an admin class which is the child of admin.ModelAdmin, Inside this class we should create a list containing all the fields of model class.

Step3 : register admin class and corresponding model class by using the function

admin.site.register(modelclassname,adminclassname)

admin.py

```
from django.contrib import admin
from myApp.models import Student
# Register your models here.
class StudentAdmin(admin.ModelAdmin):
l=['number','name','marks']
admin.site.register(Student,StudentAdmin)
```

py manage.py makemigrations

This step creates the following file

myApp\migrations\0001_initial.py

```
....
.... .... operations = [
migrations.CreateModel(
name='Student', fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True,
                serialize=False, verbose_name='ID')),#auto generated field
                ('number', models.IntegerField()),
                ('name', models.CharField(max_length=40)),
                ('marks', models.FloatField()),
            ],
        ),
    ]
```

py manage.py migrate

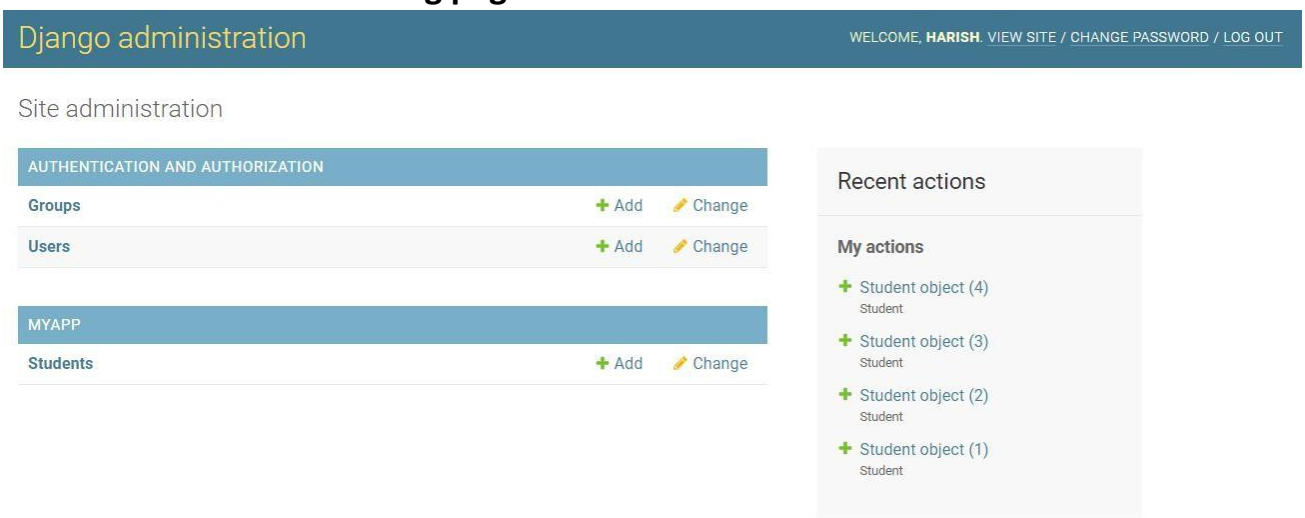
In order to have access to admin interface we must create superuser.by using the following cmd py manage.py createsuperuser

Username (leave blank to use 'dell'): shreenath
Email address: shreenath@gmail.com
Password:
Password (again):
Superuser created successfully.

Run server: py manage.py runserver

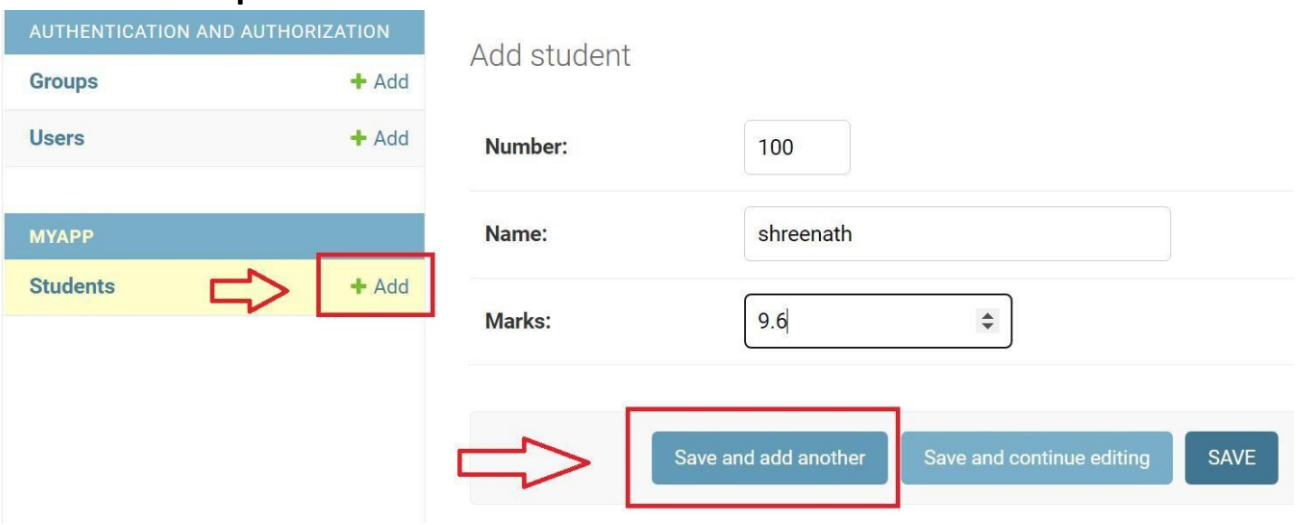
Send the request: http://127.0.0.1:8000/admin/

This will load the following page

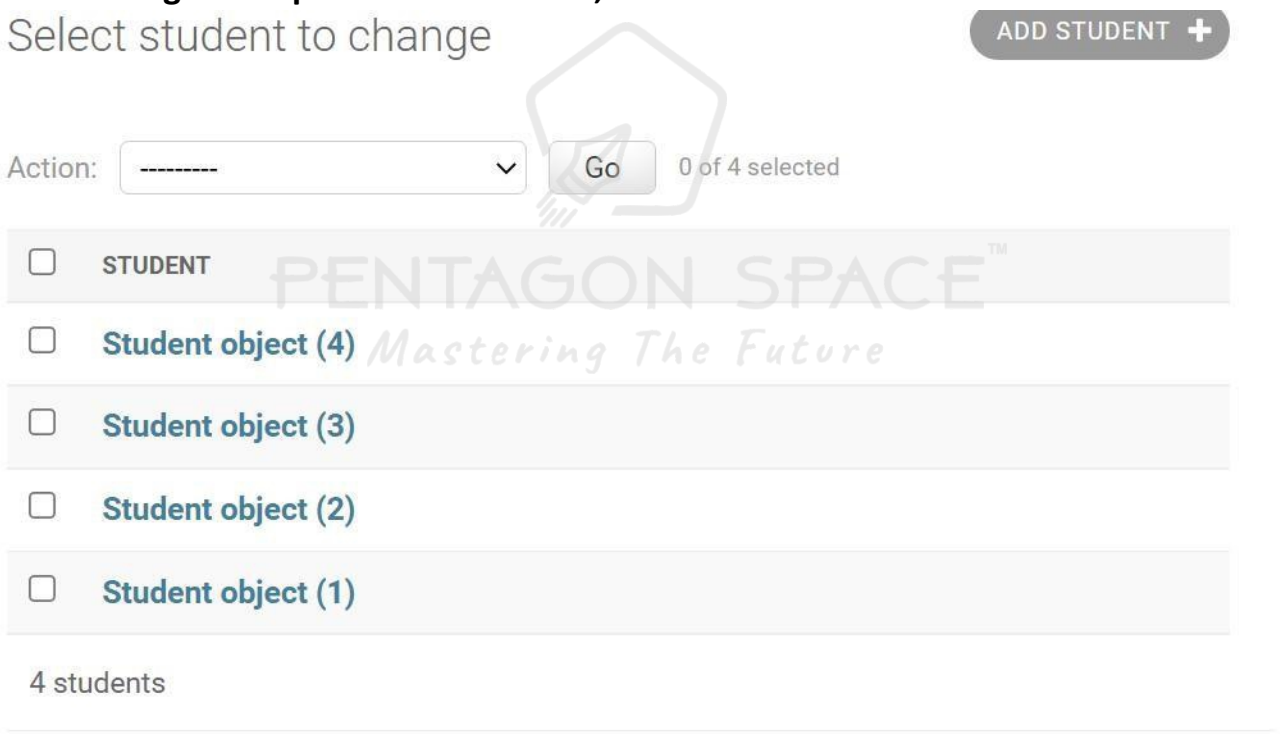


Students is the table name, if we click in students we get the following interface.

Click on Add option and enter the details



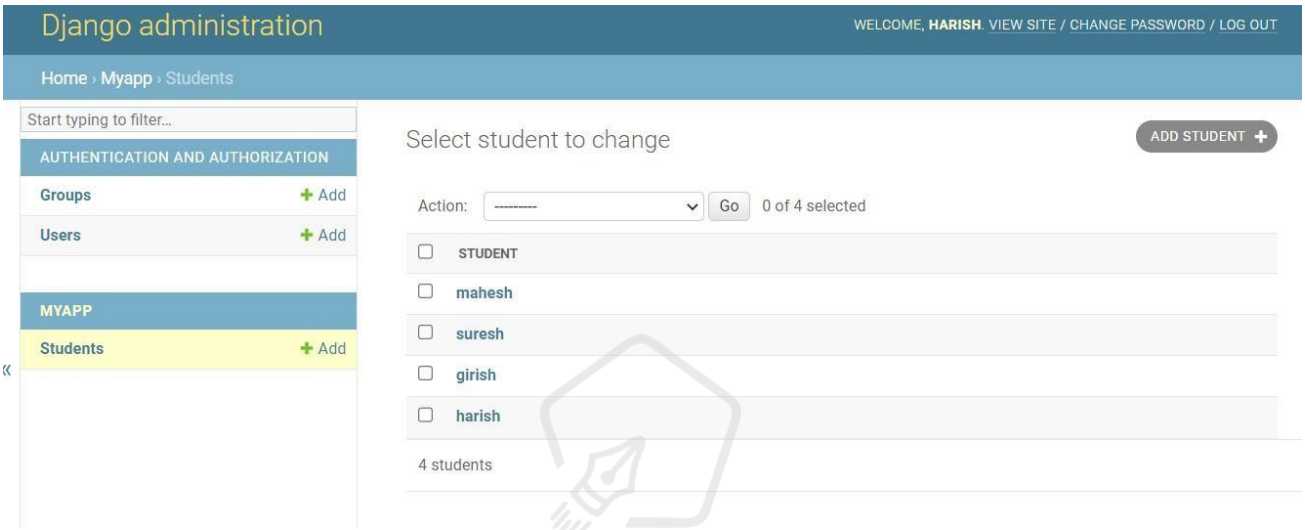
After adding the required no of records,interface looks like this



In order to convert object representation into name representation we use the method __str__ as shown below

models.py

```
from django.db import models class
Student(models.Model): #Model is a class
number=models.IntegerField()
name=models.CharField(max_length=40)
marks=models.FloatField() def __str__(self):
return self.name
```



Class 7 :ModelProjects

The data,that has been stored in admin interface should be displayed to the end user.

- **The data type associated with records is QuerySet.**
- **S=Classname.objects.all() returns all the records that have been created in admin interface.**
- **Here S is an object that refers to set of records returned from database.**
- **type(S)→QuerySet**
- **In order to pass the data from views to template,the data must be in the form of a context[dictionary]**
- **Hence we must convert the QuerySet→dictionary By using the following.**

statement d={'stud':s} stud→key s→value refers to multiple records

Now we can access the data by referring to key i.e stud inside html file

admin.py

```
from django.shortcuts import render
from myApp.models import Student
# Create your viewshere.
def myView(request):
    s=Student.objects.all()
    print(type(s))
    d={'stud':s} #qs to dict
    return render(request,'myApp/1.html',d)
```

urls.py

```
from django.contrib import admin
from django.urls import path from
myApp.views import myView
urlpatterns = [ path('admin/',
admin.site.urls),
    path('response/',myView)]
```

templates/myApp/1.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
        <h1>Student information</h1>
        <table border="2">
            <tr>
                <td>Number</td>
                <td>Name</td>
                <td>Place</td>
            </tr>
            {%for i in stud%}
            <tr>
                <td>{{i.number}}</td>
                <td>{{i.name}}</td>
                <td>{{i.marks}}</td>
            </tr>
            {%endfor%}
        </table>
    </body>
</html>
```

py manage.py runserver

send request http://127.0.0.1:8000/response/

1.django-admin startproject modelProject1
2.cd modelProject1
3.py manage.py startapp myApp
4.Goto settings.py add application and templates
5.Edit models.py and admin.py
6.py manage.py makemigrations
7.py manage.py migrate
8.py manage.py createsuperuser
9.py manage.py runserver
10.send request http://127.0.0.1:8000/admin/
11.Add the data in admin interface
12.Edit urls.py views.py,1.html
13.py manage.py runserver
14.send request http://127.0.0.1:8000/response/

Step 1 : django-admin startproject modelProject2

Step 2 : cd modelProject2

Step 3 : py manage.py startapp myApp

Step 4 : Create templates folder at project level inside that create myApp

Inside myApp create html files

Step 5 : Add application template configuration in se翩 ngs.py

models.py

```
from django.db import models class
Employee(models.Model):
eid=models.IntegerField()
ename=models.CharField(max_length=30)
edesig=models.CharField(max_length=40)
edob=models.DateField()
eexp=models.FloatField()
esal=models.IntegerField() def __str__(self):
        return self.eid
```

admin.py

```
from django.contrib import admin
from myApp.models import Employee
# Register your models here.
class EmployeeAdmin(admin.ModelAdmin):
    l=['eid','ename','edesig','edob','eexp','esal']
admin.site.register(Employee,EmployeeAdmin)
```

py manage.py makemigrations py manage.py

migrate py manage.py createsuperuser py

manage.py runserver send req

http://127.0.0.1:8000/admin/ add data and

edit the following files

views.py

```
from django.shortcuts import render
from myApp.models import Employee
# Create your views here.
def EmployeeView(request):
    e=Employee.objects.all()
    d={'emp':e}
    return render(request,'myApp/1.html',d)
```

urls.py

```
from django.contrib import admin from
django.urls import path from
myApp.views import EmployeeView
urlpatterns = [
    path('admin/', admin.site.urls),
    path('response/',EmployeeView)
    ]
```

templates/myApp/1.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
        <h1>Employee info</h1>
        <table border="2">
            <tr>
                <td>Id</td>
                <td>Name</td>
```

```
            <td>Designation</td>
            <td>DOB</td>
            <td>EXPERIENCE</td>
            <td>SALARY</td>
        </tr>
        {%for i in emp%}
        <tr>
            <td> {{i.eid}}</td>
            <td> {{i.ename}}</td>
            <td> {{i.edesig}}</td>
            <td> {{i.edob}}</td>
            <td> {{i.eexp}}</td>
            <td> {{i.esal}}</td>
        </tr>
        {%endfor%}
    </table>
</body>
</html>
```

py manage.py runserver

send request http://127.0.0.1:8000/response/

Class 8 : Django Forms

Form is used to collect details from users.Ex:Registration form,Signup Form.

```
myApp/models.py from
django.db import models class
Student(models.Model):
    sid=models.IntegerField()
    sname=models.CharField(max_length=30)
    smarks=models.FloatField()
    splace=models.CharField(max_length=30)
```

User has to create forms.py at application level

```
myApp/forms.py from django
import forms class
StudentForm(forms.Form):
sid=forms.IntegerField()
sname=forms.CharField()
smarks=forms.FloatField()
splace=forms.CharField()
```

Creating a form object

Step 1:

- Inside views.py we have to create form class object.
- For this,Inherit the form class as **from myApp.forms import StudentForm**
- f=StudentForm() where f is an object that refers to StudentForm class
- Send this form object to an html file[input.html] in the form **dictionary**

  d={'form':f} where form is key and f is value
- We must create an input html file to collect the data from

  user[input.html]

Step 2: **GET method**→we are accessing some data [read]

POST method→We are submi翮 ng some data[write]

- Inside input.html,we should **collect data** from user through **form**.
- For this we should use the context object given by views.py through the key element as **{{form}}** that displays all the form fields.
- Along with {{ form}} we should also create **submit button** for user to input the data.
- As we are collecting the input from user now we should **choose POST method** for form tag.

Step 3:

- After data is submitted from the user through form fields, these data are stored in a dictionary Called **cleaned_data.**
- Inside views.py we should perform the following steps:

1. Now **request.method [GET OR POST]** is POST because user has submitted the data through form fields.

2. If the request.method is POST perform the following tasks: I Create a form object f by calling the parameterized constructor

Parameter is request.POST as **f=StudentForm(request.POST)**

In this case the object f is ready to process the user input

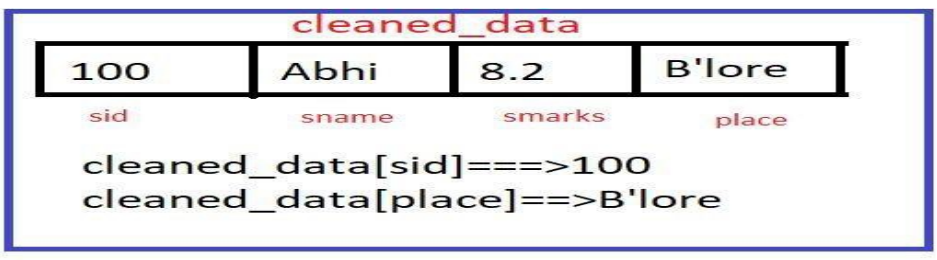
→ **Check whether form object is a valid if it is valid**

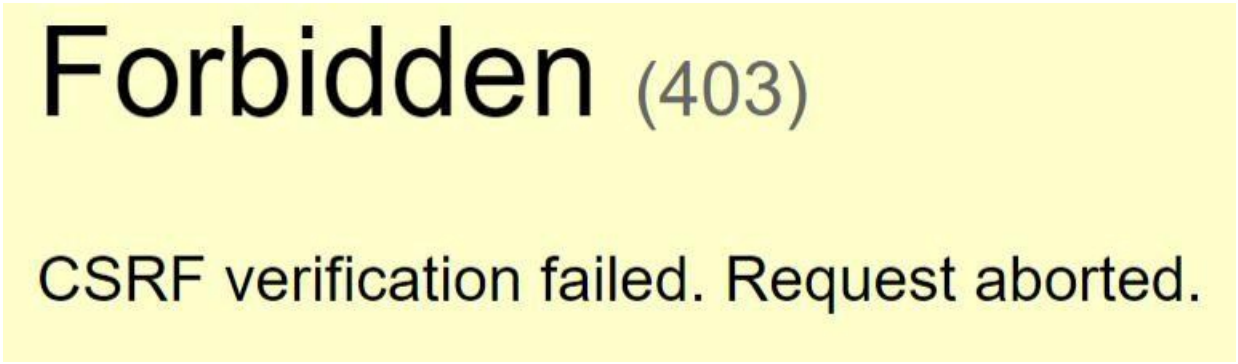
→ **By using the object f extract the data collected through form.**

name=f.cleaned_data['name'] #user submitted name data

place=f.cleaned_data['place'] #user submitted place data

→ **Now send the collected data like,name,place to html file in** the form of dictionary to display the content to end user.





While processing the form request,the data submitted through the form,would get encrypted,during this time security related operations are performed.For that a token called Cross Site Request Forgery is used.Therefore we must set the csrftoken by using jinja syntax as {% csrf_token%}

1.django-admin startproject formProject1

2.cd formProject1

3.py manage.py startapp myApp

4.create templates at project level,inside that myApp inside that html files

5.goto se翩 ngs add the application,templates

6.Inside myApp create forms.py

myApp/forms.py

```
from django import forms class
StudentForm(forms.Form):
sid=forms.IntegerField()
sname=forms.CharField()
smarks=forms.FloatField()
place=forms.CharField()
```

views.py

```
from django.shortcuts import render
from myApp.forms import StudentForm
```

```
def formView(request): f=StudentForm()
#before submi翻  ng the details through
form if request.method=="POST":
        #Aft submi翻  ng the details through form
        f=StudentForm(request.POST) if
        f.is_valid():
            sid=f.cleaned_data['sid'] sname=f.cleaned_data['sname']
            smarks=f.cleaned_data['smarks']
            splace=f.cleaned_data['place']
            d={'id1':sid,'name':sname,'marks':smarks,'place':splace}
            return render(request,"myApp/output.html",d)
    d={'form':f}

    return render(request,'myApp/input.html',d)
```

templates/myApp/1.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
        <center>
            <h1>Fill Up The Following Details</h1>
            <form method="post">
                {{form.as_p}}
                {%csrf_token%}
                <input type="submit" value="Submit">
            </form>
        </center>
    </body>
</html>
```

templates/myApp/output.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
        <h1>Thank you for filling up the details</h1>
        <h2>your details are as follows</h2>
        <h3>Id={{id1}}</h3>
        <h3>Name={{name}}</h3>
        <h3>Marks={{marks}}</h3>
        <h3>Place={{place}}</h3>
    </body>
</html>
```

urls.py

```
from django.contrib import admin
from django.urls import path from
myApp.views import formView
urlpatterns = [ path('admin/',
admin.site.urls),
path('response/',formView)
    ]
```

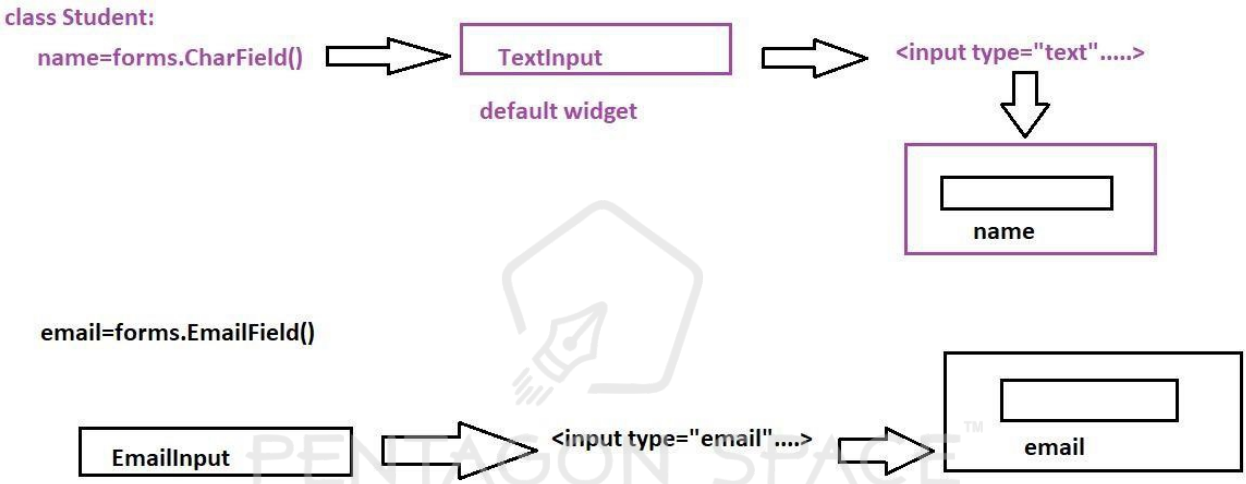
py manage.py runserver

send request: http://127.0.01:8000/response/

Creating a Django form to collect feedback from the student by validating the name field

Widgets: A widgets is Dajango represenatation of an html element. The widget handles rendering of the html and extraction of data from the GET and POST method that corresponds to widget.

When we specify a field on a form, Django will give default widget that is related to the type of data that has to be displayed.



Form validation

django-admin startproject studentFeedbackProject

cd studentFeedbackProject py manage.py startapp myApp

Create templates → myApp

Create static → css

Add application and templates entry in se翮 ngs.py

Myapp/forms.py

```
from django import forms class
FeedBackForm(forms.Form):
name=forms.CharField()
rollno=forms.IntegerField()
email=forms.EmailField()
    feedback=forms.CharField(widget=forms.Textarea)
```

views.py

```
from django.shortcuts import render
from myApp.forms import FeedBackForm
# Create your views here.
def feedbackView(request):
    f=FeedBackForm() if
    request.method=="POST":
        f=FeedBackForm(request.POST)
        if f.is_valid():
            name=f.cleaned_data['name']
            rollno=f.cleaned_data['rollno']
            email=f.cleaned_data['email']
            feedback=f.cleaned_data['feedback']
            d={'name':name,'rollno':rollno,'email':email,'feedback':feedback}
            return render(request,'myApp/output.html',d)
    d={'form':f}
    return render(request,'myApp/feedback.html',d)
```

templates/myApp/feedback.html

```
<!DOCTYPE html>
{%load static%}
<html lang="en" dir="ltr">
  <head>
    <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.
    min.css"
    integrity="sha384-
    BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vb
    dEjh4u"
```

```
    crossorigin="anonymous"> <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap
    theme.min.css" integrity="sha384-
    rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp"
    crossorigin="anonymous">
    <script
    src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
    integrity="sha384-
    Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7
    Txa " crossorigin="anonymous"></script>
  </head>
  <body>
    <div class="container" align="center">
        <h1>Student Feedback Form</h1>
        <form method="post">
          {{form.as_p}}
          {% csrf_token %}
          <br>
          <input type="submit" class="btn btn-primary " value="submit">
        </form>
    </div>
  </body>
</html>
```

templates/myApp/feedback.html

```
<!DOCTYPE html>
{%load static%}
<html lang="en" dir="ltr">
    <head>
        <link rel="stylesheet" href="{% static "css/1.css" %}">
    </head>
    <body>
        <h1>{{name}}'s Feedback</h1>
         <br>
        <h2>Rollno:{{rollno}}</h2>
        <br>
        <h2>Email:{{email}}</h2>
```

```
        <br>
        <h2>Feedback:{{feedback}}</h2>
    </body>
</html>
```

Static/css/1.css

```
h1
{
    color: red;
    text-align: center;
}
h2
{
    color: blue;
}
```

urls.py

```
from django.contrib import admin from
django.urls import path from
myApp.views import feedbackView
urlpatterns = [ path('admin/',
admin.site.urls),
    path('response/',feedbackView)
    ]
```

py manage.py runserver

send request:http://127.0.0.1:8000/response

Shreya's Feedback

Rollno:100

Email:shreya@gmail.com

Feedback:Nice Classes thank u

We can perform validations after submission

1.Length of name should be>=10

2.rollno cannot be –ve

2 ways to perform validations

1.Explicitly by using clean()

clean_fieldName(self)

The moment we submit the data through a form,django would call clean method **automatically to perform validations**,if clean method does not raise any **exception**,then only forms would be submitted.

To check name field contains minimum of 5 chars or not Rollno must be positive.

forms.py

```
from django import forms class
FeedBackForm(forms.Form):
name=forms.CharField()
rollno=forms.IntegerField()
email=forms.EmailField()
    feedback=forms.CharField(widget=forms.Textarea) def
    clean_name(self):
        n=self.cleaned_data['name']
        if len(n)<5:
            raise forms.ValidationError("Min no of chars must be >=5")
        return n
    def clean_rollno(self):
        r=self.cleaned_data['rollno']
        if r<=0:
            raise forms.ValidationError("roolno must be positive")
            return r
```

Class 9 : Faker Module

Used to generate fake data

To install faker module → **pip install faker**

The library faker is used to create fake data. Faker is an open source python library that contains various methods to generate fake data.

```
from faker import Faker
#faker-->module Faker-->class
f=Faker() name=f.name()
print("Genrated fake name=",name)
address=f.address()
print("Generated fake address=",address) text=f.text()
print("Generated fake text=",text) date=f.date()
print("Generated fake date=",date)
dob=f.date_of_birth() print("Generated
dob=",dob)
random=f.random_int(min=10,max=100)
print("Generated fake number=",random)
email=f.email() print("Generated fake
email=",email)
```

Create a file populate.py at project level,This file is used to generate fake data.

We have to execute this file after migration steps by using the command:

py populate.py

1.cd djangoApplications

2.django-admin startproject fakerProject

3. cd fakerProject

4. py manage.py startapp myApp

5.Create templates/myApp/html files

In se翻 ngs.py add application and template configuration

```
from django.db import models
# Create your models here.
class Student(models.Model):
    name=models.CharField(max_length=30)
    roll=models.IntegerField()
    marks=models.IntegerField()
    dob=models.DateField()
    email=models.EmailField()
    def __str_(self):
        return self.name
```

7. admin.py

```
from django.contrib import admin
from myApp.models import Student
# Register your models here.
class StudentAdmin(admin.ModelAdmin):
l=['name','roll','marks','dob','email']
admin.site.register(Student,StudentAdmin)
```

8.populate.py <at project level>

```
import os
import django
#Run administrative tasks
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'fakerProject.se翩  ngs')
#set environment to current project se翩  ngs
django.setup() #load the module from faker
import Faker
from myApp.models import Student
f=Faker() def generate_data(n):
    for i in range(n):#n specifies no of records
        fname=f.name()
        froll=f.random_int(min=1,max=100)
        fmarks=f.random_int(min=1,max=100)
        fdob=f.date_of_birth()
        femail=f.email
s=Student.objects.get_or_create(name=fname,roll=froll,marks=fmarks,dob=fd
ob, email=femail) generate_data(20)
```

9. py manage.py makemigration
10. py manage.py migrate
11. py populate.py

Upon execution of this step fake data would be pushed into admin interface

12. py manage.py createsuperuser
13. py manage.py runserver
14. Send the request http://127.0.0.1:8000/admin/

views.py

```
from django.shortcuts import render
from myApp.models import Student
# Create your views here. def
fakerView(request):
s=Student.objects.all() d={'stud':s}
    return render(request,'myApp/output.html',d)
```

output.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
        <center>
            <h1>Database information</h1>
            <table border="2px
```

```
            <tr>
                <td>Name</td>
                <td>Roll</td>
                <td>Marks</td>
                <td>DOB</td>
                <td>E-mail</td>
            </tr>
            {%for i in stud %}
            <tr>
                <td>{{i.name}}</td>
                <td>{{i.roll}}</td>
                <td>{{i.marks}}</td>
                <td>{{i.dob}}</td>
                <td>{{i.email}}</td>
            </tr>
            {%endfor%}
        </table>
    </center>
        </body> </html
```

urls.py

```
from django.contrib import admin
from django.urls import path from
myApp.views import fakerView
urlpatterns = [ path('admin/',
admin.site.urls),
path('response/',fakerView) ]
```

py manage.py runserver

send request:http://127.0.0.1:8000/response/

Database Information

<<<<<<<<<<<<<<<<<<<<<<<<

Name	Roll	Marks	DOB	E-mail
Theresa Reed	41	5	Sept. 19, 1929	kelseylopez@example.net
Teresa Hammond	27	91	Jan. 26, 1961	adamdavidson@example.org
Steven Short	100	73	Nov. 29, 2007	abigail37@example.com
Sara Gould	8	75	Dec. 18, 1995	gregory62@example.net
Melanie Jackson	22	51	Nov. 3, 1946	frice@example.com
Mark Myers Jr.	9	69	Aug. 8, 1960	kellybarrett@example.com
Marissa Andrade	92	41	Jan. 18, 1999	chelsea20@example.net
Maria Baker	55	98	Nov. 28, 2013	julie41@example.org
Leah Harris	50	80	Sept. 15, 1917	andrewarcher@example.org
Larry May	35	64	May 14, 1945	oyoung@example.org
Kelly Padilla	95	75	July 27, 1990	timothygonzales@example.org
Jocelyn Martinez	72	87	Sept. 20, 1925	kim04@example.net
Jay Evans	99	80	April 11, 1914	barbara66@example.com
James Lucas	51	79	July 18, 1916	dfreeman@example.net
Dr. Tina Harris MD	51	39	Aug. 23, 1962	dustin29@example.net
Denise Steele	86	92	June 20, 2003	jbrown@example.com
Cody Wolf	33	11	Dec. 7, 2020	higginspatrick@example.com
Cesar Ellis	82	10	Sept. 5, 1934	obautista@example.org
Amanda Mercer	97	22	March 10, 1987	castillowilliam@example.com
Amanda Huynh	16	27	April 11, 1918	holtmark@example.com

Class 10 : Django ORM

Object relation mapping

S=Student.objects.all()→returns all the records stored in database. Type associated with each record is QuerySet. **1.Fetch the record whose id=1**

s=Student.objects.get(id=1)

2.Filtering the record based on some condition

Ex1: Fetch all the records where marks of student >40

s=Student.objects.filter(marks__gt=40) #replace gt by lt,lte,gt,gte

Ex2: Display all the students details where marks are exactly equal to 40

s=**Student.objects.filter**(marks__exact=67)

Ex3. Fetch all records that start with “Jessica” name field

s=Student.objects.filter(name__startswith="Jessica") #check with endswith

Ex4: Fetch all the records whose id is in 1 or 2 or 3

s=Student.objects.filter(id__in=[1,2,3])

Ex5: Display all the records whose marks are in the range 70-80

s=Student.objects.filter(marks__range=(70,80))

Aggregate functions

Django ORM provides different functions to perform aggregate operations like **Avg(),Max(),Min(),Sum(),Count().**These functions are defined in **django.db.models.**

```
from django.db.models import Avg
avg=Student.objects.all().aggregate(Avg("marks"))
print(avg)      #{'marks__avg':      58.55}      from
django.db.models import Max
max=Student.objects.all().aggregate(Max("marks"))
print(max) #{'marks__max': 100} from
django.db.models import Min
min=Student.objects.all().aggregate(Min("marks"))
print(min) #{'marks__min': 16} from
django.db.models import Sum
```

```
sum=Student.objects.all().aggregate(Sum("marks"))
print(sum) #{'marks__sum': 1171}
from django.db.models import Count
count=Student.objects.all().aggregate(Count("id"))
print(count) #{'id__count': 20}
```

Ordering the query set

In order to **arrange the queryset by ascending or descending** manner we use order_by.This function **accepts the parameter** based on which ordering must be done.Default ordering is **ascending order.**

```
s=Student.objects.all().order_by('marks') #Records are in ascending order
s=Student.objects.all().order_by('name') #Records are in ascending order
s=Student.objects.all().order_by('-name')# Records are in descending order
s=Student.objects.all().order_by('-name')[1:4]
```

Model Forms

The forms that are designed **based on model** is called **model forms.**

django-admin startproject modelFormProject1 cd

modelFormProject1 py manage.py startapp myApp

Create templates/myApp/html files

Configure se翮 ngs.py with templates and applications

models.py

```
from django.db import models # Create your
models here. class Student(models.Model):
roll=models.IntegerField()
name=models.CharField(max_length=30)
place=models.CharField(max_length=30) def
__str__(self): return self.name
```

admin.py

```
from django.contrib import admin from
myApp.models import Student class
StudentAdmin(admin.ModelAdmin):
l=['roll','name','place']
admin.site.register(Student,StudentAdmin)
```

myApp/forms.py

```
from django import forms from
myApp.models import Student class
StudentForm(forms.ModelForm):
    class Meta:#contains model,fields
        model=Student
        fields='__all__'
        #fields=('f1','f3','f4','f5')
        #exclude=['f2','f3']
```

views.py

```
from django.shortcuts import render
from myApp.models import Student
from myApp.forms import StudentForm
# Create your views here.
    def input_view(request):
        f=StudentForm()
          if request.method=="POST":
              f=StudentForm(request.POST)
              if f.is_valid():
                  f.save(commit=True)
                   #to save the changes we use commit=True
        return render(request,'myApp/1.html',{'form':f})
```

urls.py

```
from django.contrib import admin
from django.urls import path from
myApp.views import *
urlpatterns = [
path('admin/',admin.site.urls),
```

```
    path('response1/',input_view)
    ]
```

templates/myApp/1.html

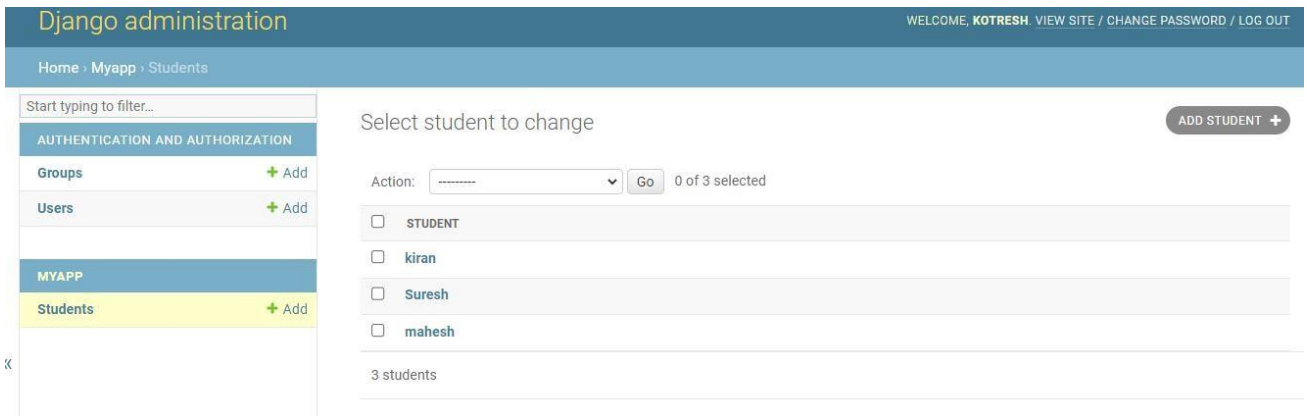
```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstra
        p.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmU
        cww7on3RYdg4Va+PmSTsz/K68vbdEjh4 u" crossorigin="anonymous">

    <!-- Optional theme -->
        <link                    rel="stylesheet"href="https://maxcdn.bootstrap
        cdn.com/bootstrap/3.3.7/css/bootstraptheme.min.css"
        integrity="sha384rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYo
        oPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">

    <!-- Latest compiled and minified JavaScript -->
        <script
        src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.mi
        n.js"integrity="sha384-
        Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNI
        cPD7Tx
        a" crossorigin="anonymous"></script>
    </head>
    <body>
        <center>
            <h1 style="color:red">Fill out the form</h1>
            <form method="post">
                {{form.as_p}}
                {%csrf_token%}
                <input type="submit" class="btn btn-primary" value="submit">
            </form>
        </center>
    </body>
</html>
```

py manage.py makemigrations
py manage.py migrate py
manage.py createsuperuser py
manage.py runserver
send request http://127.0.0.1:8000/admin/
Check admin panelpy manage.py runserver

send request http://127.0.0.1:8000/response1/



Class 11 : FBV Application2

1.django-admin startproject studentfbvproject

2.cd studentfbvproject

3.py manage.py startapp myApp

4.create templates/myApp/html files at project level

5.In se翻 ngs.py,add application and TEMPLATE_DIR

models.py

```
from django.db import models class
Student(models.Model):
sno=models.IntegerField()
sname=models.CharField(max_length=50)
smarks=models. FloatField()
saddr=models.CharField(max_length=100)
def __str__(self):
    return self.sname
```

admin.py

```
from django.contrib import admin from
myApp.models import Student class
StudentAdmin(admin.ModelAdmin):
l=['sno','sname','smarks','saddr']
admin.site.register(Student,StudentAdmin)
```

myApp/forms.py

```
from django import forms from
myApp.models import Student class
StudentForm(forms.ModelForm):
    class Meta:
        model=Student
        fields='__all__'
```

At project level,create populate.py file

populate.py

```
import osimport django
os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'studentfbvproject.se翩  ngs')
django.setup() from faker import
Faker from myApp.models import
Student f=Faker() def populate(n):
    for i in range(n):
        fno=f.random_int(min=1,max=20)
        fname=f.name()
        fmarks=f.random_int(min=1,max=100)
        faddr=f.address()

e=Student.objects.get_or_create(sno=fno,sname=fname,smarks=fmarks,
saddr= faddr) populate(20)
```

templates/myApp/parent.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/boots
        trap.min.css" integrity="sha384-
        BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/
        K68vbdEjh4 u" crossorigin="anonymous">
    <!-- Optional theme -->
        <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/boots
        trap theme.min.css" integrity="sha384-
        rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmg
        JQIXwl/Sp" crossorigin="anonymous">
    <!-- Latest compiled and minified JavaScript -->
        <script
        src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.
        min.js" integrity="sha384-
        Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wG
        NIcPD7Tx a" crossorigin="anonymous"></script>
```

```
    </head>
    <body>
        <div class="container" align="center">
            {%block for_child%}
            {%endblock%}
        </div>
    </body>
</html>
```

templates/myApp/index.html

```
<!DOCTYPE html>
{%extends 'myApp/parent.html'%}
<html lang="en" dir="ltr">
  <body>
    {%block for_child%}
    <h1 style="color:red"><b>Student Details</b></h1>
    <table border="2">
      <tr style="color:green">
        <th>Number</th>
        <th>Name</th>
        <th>Marks</th>
        <th>Address</th>
      </tr>
    {%for i in stud%} <tr>
      <td><b>{{i.sno}}</b></td>
      <td><b>{{i.sname}}</b></td>
      <td><b>{{i.smarks}}</b></td>
      <td><b>{{i.saddr}}</b></td>
      <td><a href="/update/{{i.id}}" class="btn btn-
      warning">Update</a></td>
      <td><a href="/delete/{{i.id}}" class="btn btn-danger">Delete</a></td>
    <td></td> </tr>
    {%endfor%}
    </table>
    <br><br>
    <a href="/insert" class="btn btn-primary">Do you want to insert a new
    record?</a> {%endblock%}
```

```
  </body>
</html>
```

urls.py

```
from django.contrib import admin
from django.urls import include,re_path
from myApp import views
urlpatterns = [ re_path(r'^admin/', admin.site.urls),
    re_path(r'^$',views.display),
    re_path(r'^insert/',views.insert_view),
    re_path(r'^delete/(?P<id>\d+)',views.delete_view),
    re_path(r'update/(?P<id>\d+)',views.update_view)
    ]
```

templates/myApp/insert.html

```
<!DOCTYPE html>
{%extends 'myApp/parent.html'%}
<html lang="en" dir="ltr">
  <body>
    {%block for_child%}
    <h1>Insertion Form</h1>
      <form method="post">
        {{form.as_p}}
        {%csrf_token%}
        <input type="submit" class="btn btn-primary" value="Insert New
        Record">
      </form>
    {%endblock%}
  </body>
</html>
```

templates/myApp/update.html

```
<!DOCTYPE html>
{%extends 'myApp/parent.html'%}
<html lang="'en'" dir="ltr">
   <body>
     {%block for_child%}
```

```
        <h1 style="color:blue">Updation form</h1>
        <form method="post">
            Number:<input type="text" name="sno"
            value="{{stud.sno}}"><br><br> Name:<input type="text"
            name="sname" value="{{stud.sname}}"><br><br> Marks:<input
            type="text" name="smarks" value="{{stud.smarks}}"><br><br>
            Address:<input type="text" name="saddr"
            value="{{stud.saddr}}"><br><br>
            {%csrf_token%}
            <input type="submit" class="btn btn-warning" value="submit">
        </form>
        {%endblock%}
    </body>
</html>
```

views.py

```
from django.shortcuts import render,redirect
from myApp.models import Student from
myApp.forms import StudentForm def
display(request):
    s=Student.objects.all()
    d={'stud':s}
    return render(request,'myApp/index.html',d)

def insert_view(request):
    f=StudentForm() if
    request.method=="POST":
        f=StudentForm(request.POST)
        if f.is_valid():
            f.save(commit=True)
        return redirect('/')
    d={'form':f}
    return render(request,'myApp/insert.html',d)

def delete_view(request,id):
    s=Student.objects.get(id=id)
    s.delete() return redirect('/')
```

```
def update_view(request,id):
    s=Student.objects.get(id=id) if
    request.method=="POST":
    f=StudentForm(request.POST,instance=s)
        if f.is_valid():
            f.save(commit=True)
        return redirect('/')
d={'stud':s}
return render(request,'myApp/update.html',d)
```

---

py manage.py makemigrations

py manage.py migrate py

manage.py createsuperuser py

populate.py py manage.py

runserver send the request

through

http://127.0.0.1:8000/

Class 12 : CLASS BASED VIEWS

CLASS BASED VIEWS are used to implement generic views

- **CBVs are used compare Fbv**
- **CBV internally gets converted into FBV that is CBVs acts as wrappers to FBV**
- **FBVs are more powerful as compared to CBV**
- **For simple operations like, listing of records, displaying details of a particular record, we prefer CBV**
- **For complex operations like handling multiple files we use FBVs • In CBVs we should inherit the parent class View.**
- **While defining the urls pattern to CBV we should use as_view()**
- **ListView→class available in Django.views.generic→to list all the record**
- **DetailView→class available in djnago.views.generic→to get the details of particular record specific record**

CBV-Application

Generic View	Default Template	Default Context
ListView	modelclass_list.html	modelclass_list
DetailView	modelclass_detail.html	modelclass
CreateView	modelclass_form.html	-
UpdateView	modelclass_form.html	-
DeleteView	modelclass_confirm_delete.html	modelclass

path('response/',StudentView.as_view(),name='showView') #reverse url

success_url=reverse_lazy('reverseurl name')

success_url contains a target page that should be displayed after deletion,the target page is given by reverseurl name. reverse_lazy This function waits until the record gets deleted.

Ex: path('response/',myView.as_view(),name='display') #reverse url
success_url=reverse_lazy(' display')

In the above line,because of reverse_lazy the control would be paused until deletion is successful.Once deletion is successful,success_url contains the target page that should be displayed after deletion that is given by reverse url display pertaining to response end point. models.py

```
from django. urls import reverse
class Student(….):
….
….
    def get_absolute_url(self):
        return reverse('detail',kwargs={'pk':self.pk})
```

get_absolute_url is used to get the target file and it uses a function reverse to perform the matching activity wrt a particular view through reverse url with required argument,here pk is id

django-admin startproject cbvproject

cd cbvproject py manage.py startapp

myApp

create templates/myApp/html files at project level

configure se翩 ngs.py

models.py

```
from django.db import models from
django.urls import reverse class
Student(models.Model):
name=models.CharField(max_length=50)
number=models.IntegerField()
marks=models.FloatField()
place=models.CharField(max_length=50)
def __str__(self):
    return self.name
def get_absolute_url(self): return
    reverse('detail',kwargs={'pk':self.pk})
```

admin.py

```
from django.contrib import admin from
myApp.models import Student # Register
your models here. class
StudentAdmin(admin.ModelAdmin):
    l=['name','number','marks','place']
admin.site.register(Student,StudentAdmin)
```

views.py

```
from django.shortcuts import render
from myApp.models import Student
from django.urls import reverse_lazy
from django.views.generic import
ListView,DetailView,CreateView,UpdateView,DeleteView
class StudentView(ListView): model=Student #default
template:student_list.html
    #default context:student_list
class StudentDetailView(DetailView):
model=Student
    #default template:student_detail.html
    #default context:student class
StudentCreateView(CreateView):
    model=Student
    fields='__all__'
```

```
        #default template:student_form.html
    class StudentUpdateView(UpdateView):
        model=Student
        fields=('name','marks')
        #default template:student_form.html
    class StudentDeleteView(DeleteView):
    model=Student
        success_url=reverse_lazy('students')
        #students-reverse url to display student list
```

urls.py

```
from django.contrib import admin
from django.conf.urls import url
from myApp.views import *
urlpatterns = [ url(r'^admin/', admin.site.urls),
    url(r'^students/',StudentView.as_view(),name='students'),
    url(r'^(?P<pk>\d+)',StudentDetailView.as_view(),name='detail'),
    url(r'^create/',StudentCreateView.as_view(),name='create'),
    url(r'^update/(?P<pk>\d+)',StudentUpdateView.as_view(),name='update'),
    url(r'^delete/(?P<pk>\d+)',StudentDeleteView.as_view(),name='delete'),
    ]
```


parent.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
        <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/boots
        trap.min.css" integrity="sha384-
        BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/
        K68vbdEjh4u" crossorigin="anonymous">

        <!-- Optional theme -->
        <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/boots
        traptheme.min.css" integrity="sha384-
        rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmg
```

```
        JQIXwl/Sp" crossorigin="anonymous">
        <!-- Latest compiled and minified JavaScript -->
        <script
        src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstr
        ap.min.js" integrity="sha384-
        Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8
        wGNIcPD7Txa" crossorigin="anonymous"></script>
    </head>
    <body>
        <div class="container" align="center">
            {%block for_child%}
            {%endblock%}
        </div>
    </body>
</html>
```

student_list.html

```
<!DOCTYPE html>
{%extends 'myApp/parent.html'%}
<html lang="en" dir="ltr">
    <body>
        <center>
            {%block for_child%}
            <h1><u>List of students</u></h1>
            {%for i in student_list%}
            <h2><a href="/{{i.id}}">{{i.name}}'s Details</a></h2>
            {%endfor%}
            {%endblock%}
        </center>
    </body>
</html>
```

student_detail.html

```
<!DOCTYPE html>
{%extends 'myApp/parent.html'%}
<html>
    <body>
```

```
        <center>
            {%block for_child%}
            <h1 style="color:blue">Student Details</h1>
            <h2>Name:{{student.name}}</h2>
            <h2>Num:{{student.number}}</h2>
            <h2>Marks:{{student.marks}}</h2>
            <h2>Place:{{student.place}}</h2>
            <h2><a href="/update/{{student.id}}" class="btn btn
            warning">Update</a></h2>
            <h2><a href="/delete/{{student.id}}" class="btn btn
            danger">Delete</a></h2>
            {%endblock%}
        </center>
    </body>
</html>
```

student_form.html

```
<!DOCTYPE html>
{%extends 'myApp/parent.html'%}<html>
    <body>
        <center>
            {%block for_child%}
            <h1 style="color:blue">Student Creation Form</h1>
            <form method="post">
                {{form.as_p}}
                {%csrf_token%}
                <input type="submit" class="btn btn-primary"
                value="Insert/Update Record">
            </form>
            {%endblock%}
        </center>
    </body>
</html>
```

student_confirm_delete.html

```
<!DOCTYPE html>
{%extends 'myApp/parent.html'%}
```

```
<html>
    <body>
        <center>
            {%block for_child%}
            <h1 style="color:red">Do you want to delete
            {{student.name}}??</h1>
            <form method="post">
                {%csrf_token%}
                <input type="submit" class="btn btn-danger" value="Delete">
                <a href="/{{student.id}}" class="btn btn-success">Cancel</a>
            </form>
            {%endblock%}
        </center>
    </body>
</html>
```

CBV	FBV
Promotes code reusability	Cannot provide code reusability
CBVs can be easily extended	Cannot be extended
Less coding	More coding
Default context,default templates are available	Not available
We should use as_view() to refer to CBV in urls	We do not use as_view()
Suitable for simple operations	Suitable for complex operations
Implicit execution flow	Explicit execution flow
We use separate get() post()	We use the statemet like request.metod==”POST”:
Less powerful	More powerful
Used to create mixins	Not used for mixin creation

Mock Questions

1)What is Django?

→Django is a free and open source framework for building web apps with python and Django is a python based high level web framework and mainly is used to develop a rapid, fledible and complex web application. Which is used MVT design pattern.

2)Name some Companies that use Django?

→Youtube, instagram , spotify, netflex, dropbox

3)What is the difference between a project and an App?

→A project is a collection of files and apps where as the app is a web application which is written to perform business logic.

4)Explain Django's Architecture?

→Django's architecture is based on the MVT framework, which stands for Model, View, Template.

5)what are the features of using a Django?

→admin site, object-relational Mapper, authentication.

6)What are the files contains in the project?

→__init__.py

Asgi.py

Se翻 ngs.py

Urls.py

Wsgi.py

7)How do you create a Django Project?

→Django-admin startproject project_name

8)How do you create a Django App?

→py manage.py startapp app_name

9)How do we start our development Server?

→py manage.py runserver

10)Give a brief about the Django admin interface?

→To register our models in the admin interface.

11)What are Django URLs?

→It contains URL pattern corresponding to each view.

12)What are the view of Django?

→It is used to write business logic or function. The view pass atleast one argument that is called request.

13)What are the models in Django?

→It is used to store application specific data models and it is used for database operations.

14)What does the se翩 ngs.py file contains?

→ It contains Installed app, database,templates,middleware, auth_password validator.

15)What is the use of Middlewares in Django?

→It act's like a processing unit between client and the server.

16)What databases are supported by Django?

→db.sqlite3.

17)What is a Framework?

→It is a fundamental structure for developing a software projects. It provides set of predefined tools and libraries that streamline and organize the projects.

18)What is the Django's default port number to run the server?

→8000

19)What is the use of context?

→context is a dictionary it contains key value pairs.

20)What is the use of jinja syntax?

→Jinja syntax uses templating language that contains variables and programming logics. The variables or Programming logics are placed inside the delimiters.

Ex:To fetch any result or data e use {{data}},for expressions we use {%expression%}

21)What is the use of Templates?

→It contains the Html Page. To rendering the response through Users.

22)What is the path we use for template?

→TEMPLATE_DIR=os.path.join(BASE_DIR,'templates')

23)Why we use StaticFiles?

→It uses for the styling purpose.

24)What are the staticfile it contents?

→CSS and Image.

25)What is Record?

→Each row is reffered to ad record.

26)What is a Table?

→Table contains a rows and columns.

27)What is Field?

→Each heading is called a field.

28)What is Primary Key?

→Each record is identified with a unique number called primary key.

29)What is the use of Admin file?

→To register our model in Admin interface that time we are going to use admin file.

30)To register the admin class and corresponding model class which command your going to use?

→admin.site.register(model class_name, admin class_name)

31)What is Query Set?

→The data type associated with records is QuerySet.

32)How do we create a all the records in the admin interface?

→S=classname.objects.all()

33)What is a Form?

→We should collect the data from the user.

34)What is the use of GET method?

→We are accessing some data[read].

35)What is the use of POST method?

→We are submi翮 ng some data [write].

36)what is Cleaned_Data?

→After data is submitted from user throght form fields, these data are stored in a dictionary called cleaned_data.

37)Expand CSRF?

→Cross Site Request Forgery

38)What is the jinja syntax for CSRF token?

→{% csrf_token %}

39)What is CSRF?

→While processing the form request, the data submitted through the form, would get encrypted, during this time security related operations are performed. For that token called Cross Site Request Forgery.

40)What is widgets?

→A widgets is Django representation of an html element. The widget handles rendering of the html and extraction of data from the GET and POST method that corresponds to widget.

41)What is the use of Clean Method?

→The moment we submit the data through a form, django would call clean method automatically to perform validations,if clean method does not raise any exception, then only forms would be submitted.

42)What is the use of Faker Module?

→It is used to Create a Fake data. Faker is an open source python library that contains various methods to generate fake data.

43)What is ORM?

→Object Relational Mapper it returns all the records stored in database.

44)What are the Aggregate function or operations in Django?

→Avg(), Max(), Min(), Sum(), Count() .

45)For ordering the Queryset which function we are going to use?

→order_by This function accepts the parameter based on which ordering must be done .the default ordering is ascending order.

46)What is the use of Class Based Views?

→It is used to implement the generic Views.

47)What is ListView?

→To list all the Record.

48)What is DetailView?

→To get the Details of particular Record.

49)What is CreateView?

→To Create a Specific Record.

50)What is UpdateView?

→To update a Specific Record.

51)What is DeleteView?

→To delete a Specific Record.

52)What are the model.py file contains?

→It contains the database attribute.

53)Difference between CBV and FBV?

→

CBV	FBV
Promotes code reusability	Cannot provide code reusability
CBVs can be easily extended	Cannot be extended
Less coding	More coding
Default context,default templates are available	Not available
We should use as_view() to refer to CBV in urls	We do not use as_view()
Suitable for simple operations	Suitable for complex operations
Implicit execution flow	Explicit execution flow
We use separate get() post()	We use the statemet like request.method==”POST”:
Less powerful	More powerful
Used to create mixins	Not used for mixin creation

54)What is the model form?

→The forms that are designed based on the model is called model forms.