Django Notes

Django :- Points (django-admin --version)

To Install in virtual Environment :-

1) Pip install virtualenvwrapper-win

2) mkvirtualenv aName venv

Venv is now created

Now, whenever u want to use django, typ: workon aName (of env)

in cmd, django will be activated, Now u can use it

To uninstall :- pip uninstall django → rmvirtualenv envname, pip uninstall virtualenvwrapper

Project :- Command to Create Project :- django-admin startproject name

Project is combination of diff application/apps.

e.g :- SchoolProject :- AdmissionApp, FeeApp, ResultApp etc

Project Files :-

(Python manage.py runserver) :- To run server

manage.py :- diff apps ko manage karnay k liye

Setting :- it contain database template & Password etc related info.

wsgi :- Compatable servers k liye entry Point hota hai

web server Gateway interface →

wsgi = Provide Standard for Synchronus Python APPs.

init :- this file works as a Package for the Project.

Jo file jis directory mai hoga wo as a Package file consider kya jaega

asgi :- (Asynchronous Server Gateway Interface) :- It Provides standard for both Asynchron & Synchronus APPs.

wsgi.Py File

import os importing module

Package module function

from django.core . wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SEETS_MODULE', 'first.settings')

a mapping object representing the string environment. Function Environment variable value Defining settings module to environment variable

application = get_wsgi_application()

object callable this function returns wsgi callable.

asgi.Py :- it contains same code as wsgi file.

Settings.Py :- most used file than wsgi, asgi

If u want a specific Port number on which to run the server:-

Python manage.Py runserver 55

Port number

Ctrl+C to stop server

To Create APP :- Python manage.py startapp appName

Now open settings → INSTALLED_APPS = ['django.contrib.admin', 'appName',] → Save

APP Directory Structure :-

1) migrations → __init__ :- database se related jo files generate hoti un ko apne pass rakhna

2) __init__.py :- ye file is folder mai hoga us ke as a Python Package consider kya jayega.

3) admin.py :- use to register sql tables so we could perform CRUD
Admin APP is Provided by django to perform CRUD
بنا بنایا ہے

4) apps.py :- use to config the app.

5) models.py :- to create our own model class later these classes will be converted into ~~small~~ database table by django for our app.

6) views.py :- View of our app.

models & admin :- zyada tar datab & me use krty hai.

7) tests.py :- use to create tests.

---

urls.py

Not a good approach when we have 1+ APPs

```
from course import views as reviews
from course import views as Resultviews
urlpatterns = [path('rev', review.name),
              ]
```

```
from course.views
import rNam
path('rev', rNam)
```

is not a good approach

is proper approach

When we have more than one app: hum har 1 app ko independent banane ki koshish karey gai. jis k liye har app ka apna urls.py file hoga.

Geeky Steps

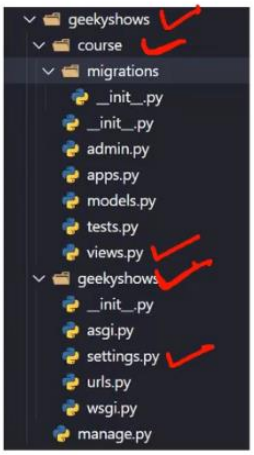
- Create Django Project: *django-admin startproject geekyshows*
- Change Directory to Django Project: *cd geekyshows*
- Create Django Application: *python manage.py startapp course*
- Add/Install Application to Django Project (course to geekyshows) using settings.py file INSTALLED_APP
- Write View Function inside **views.py** file
  - Open **views.py**
  - Import HttpResponse class from django.http module

    from django.http import HttpResponse
  - Write view Function

    def learn_django(request):

    return HttpResponse('Hello Django')
  - Save views.py

URLs



URL Pattern inside Project

urls.py file is used to define url pattern attached with application or view of application or view function of application.

urls.py file is located inside *inner project folder* not inside *application folder* which means we define url at project level for applications. Defined url name will be used by application user to get response from the application or view function of applic

Steps:-

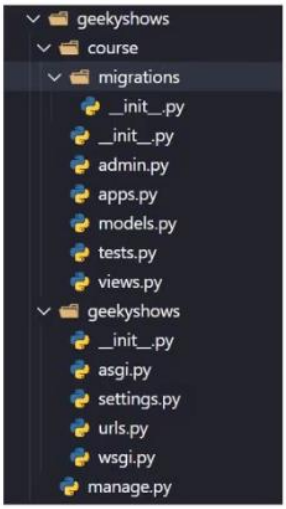
- Open urls.py
- Import Module (Python file) of the application
- Write URL Name and Map it with function

```
urlpatterns = [
        path(route, view, kwargs=None, name=None)
]
```

```
from course import views
urlpatterns = [
        path(learndj/', views.learn_django),
]
```

learndj is mapped with learn_django function which is inside views.py file.

http://127.0.0.1:8000/learndj



Single Application with Single function.

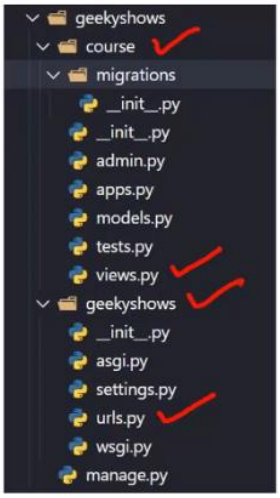
views.py

```
from django.http import HttpResponse
def learn_django(request):
        return HttpResponse('Hello Django')
```

urls.py

```
from course import views
urlpatterns = [
        path(learndj/', views.learn_django),
]
```

http://127.0.0.1:8000/learndj



Specific Manner to includ APPs in Project.urls.py

Create urls.py inside each app → write urlPatterns for each app in its own urls.py file for its views.py file.

Then write:- from django.urls import include
urlPatterns=[Path('anyPath', include('appName.urls')),
On which to access that app.

Code in Project.urls file.

Jab hum static/template folder ko Project folder mai bnatei hai to tab us ko setting mai define krna Pta hai.

Method:- 1) create templates & static files

2) settings.py:- BASE_DIR.....) below this line
TEMPLATES_DIR = BASE_DIR/'templates'
STATIC_DIR = BASE_DIR/'static'

Scroll a little down:- TEMPLATES=['DIRS':[TEMPLATES_DIR]
Scroll down:- below this:- STATIC_URL...
write:- STATICFILES_DIRS=[STATIC_DIR]

Templates and Static files are now sethetup in setting.py file.

Two Basic View.py Functions:-

1) from django.HTTP import HTTPResponse
def First(request):
return HTTPResponse('Hello world')

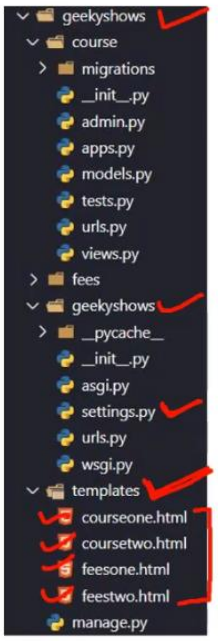
2) from django.shortcuts import render
def AName(request):
return render(request, 'myApp/index.html')
Path

Geeky Steps

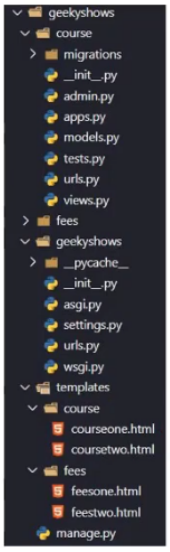
- Create Django Project: *django-admin startproject geekyshows*
- Create Django Application1: *python manage.py startapp course*
- Create Django Application2: *python manage.py startapp fees*
- Add/Install Applications to Django Project (course and fees to geekyshows) using settings.py INSTALLED_APPS
- Create **templates** folder inside Root Project Directory
- Add **templates** directory in settings.py

```
TEMPLATES_DIR   = os.path.join(BASE_DIR, 'templates')
TEMPLATES = [ {
            'DIRS': [TEMPLATES_DIR], } ]
```

- Create template files inside **templates** folder
- Write View Function inside views.py file
- Define url for view function of application using urls.py file



Dynamic Template Files



views.py

```
from django.shortcuts import render
def learn_django(request):
        cname = 'Django'
        duration = '4 Months'
        seats = 10
        django_details = {'nm':cname, 'du':duration, 'st':seats}
        return render(request, 'course/courseone.html', django_details)
```

templates/course

***courseone*.html**

```
<html>
  <body>
        <h2> Course Name:{{nm}} Duration:{{du}} and Total Seats: {{st}}</h2>
  </body>
</html>
```

STATIC_URL – This is the URL to use when referring to static file located in STATIC_ROOT. It must end in a slash if set to a non-empty value.

Example:- "/static/"

Example:- "http://static.example.com/"

STATIC_ROOT – This is absolute path to the directory where *collectstatic* will collect static files for deployment. It is by default None.

Example:- "/var/www/example.com/static/"

Example:- os.path.join(BASE_DIR, 'static/')

STATICFILES_DIRS - This setting defines the additional locations the *staticfiles* app will traverse if the *FileSystemFinder* finder is enabled, e.g. if you use the *collectstatic* or *findstatic* management command or use the static file serving view. It is by default an empty list.

```
STATICFILES_DIRS = [
    "/home/special.geek.com/geek/static",
    "/home/geek.com/geek/static",
    "/opt/webfiles/common",
]
```

www.geekyshows.com

To run a specific file from project

C:\Scripts> Python settings.py

Syntax of load:
{% load geek_mytags %}
{% load cry hel from emotags %}
both will load from emotags

When we want to use our static files in html, we must load it. Syntax:- .html file:- line 1) {% load static %}
e.g.

1) <head>... <link rel="stylesheet" href="{% static 'css/style.css' %}
directory name, file name

2) <img src="{% static "images/pic1.jpg" %}">

render()
compulsory

render(request, 'templateName.html' or dicName is sam, context=dicName, content_type=MIME_type, status=None, using=None)

Eg:-
render(request, 'jeos.html', Context=MyDi, content_type='application/xml+xml')

Eg:2]:- file = views.py

```
def courses(request):
    CourseDet = {'CName':'Django', 'CPrice':2000}
    return render(request, 'courses/course1.html', CourseDet)
```

And course1.html file = <html>
syntax to define a variable
here we have variable a Dictionary

<p>Course Name is : {{CName}}

Filters:- when we need to modify variable before displying it we use filters.

Syntax:- {{ variable | filter }}

Eg:- {{ cName | upper }}

Output Web:
Course Name is : Django

6

If truncatewords = 2 => Result :- how are ...

how are you and wher r you

where are

if truncatechars = 5,

Result = wo...

Some filter take arguments :- Syntax :- {{variable | filter : argument}}

e.g :- {{ article | truncatewords : 40 }}

Filters can be chained : e.g :- {{ article | upper | truncatewords : 40 }}

=> {{ value | default : "anything" }}

matlab agar true hojaye to yeh dikhaw nahi to default me jo value hai vo dikhao.

Value => jab false ya blank hogi to default filter run hoja.

Date and Time are important filters that we use.

Example :- 1) Views.py :- from datetime import datetime

```
def hello(req):
    wqt = datetime.now()
    return render(request, 'fees/fees.html', {'dt': wqt})
```

fees.html file :- <h2>{{dt | date: 'D d M y'}}
{{dt | time: 'H:i'}}

Output we :- Mon 10 Jan 2022
20:21

2) <h2> {{dt | date: 'SHORT_DATE_FORMAT'}}
{{dt | time: 'TIME_FORMAT'}}

Output web :- 01/10/2022
8:26 P.m

1) Float ford : 3 matlab 3 digits ka bad

Floatformat filter

e.g :- fl = 32.39827

=> {{ fl | floatformat }}

Result :- 32.4

Round off kr ta hai our decimal ka bad 1 digit chorta hai

Display kro

Jiang Baptiste Darthenay } a vscode ext for Djang Template language to see each tag in a diff color.

Pre defined Date & time formats :- E.g:-

DATE_FORMAT Default : 'N j, Y' Feb. 9, 2020

DATE_TIME_FORMAT => Feb. 9, 2020, 10 PM

SHORT_DATE_FORMAT => 12/31/2020

" " DATETIME_F... => 12/31/2020 4 P.m

TIME_FORMAT => "4:24"

If Tag (it's like if condition)

Value hai to ye run hoga.

E.g:- & Syntax:-

mtlb agar nmbr var true hai/iss mai koi

{% if nmbr %}
<h2> Number : {{nmbr}} </h>

{% endif %}

E.g:2) {% if nm|length < 100 %}
<h1> small
{% endif %}

3) {% if nmbr %}
<h2> Number : {{nmbr}} </h2>
{% else %}
<h2> No Number </h2>
{% endif %}

4) {% if nm %}
 Number : {{nm}}
{% elif st %}
 String : {{st}}
{% else %}
 No Number & String
{% endif %}

Predefined Formats

Format	Description	Example
DATE_FORMAT	Default: 'N j, Y'	Feb. 9, 2020
DATETIME_FORMAT	Default: 'N j, Y, P'	Feb. 9, 2020, 10 p.m.
SHORT_DATE_FORMAT	Default: 'm/d/Y'	12/31/2020
SHORT_DATETIME_FORMAT	Default: 'm/d/Y P'	12/31/2020 4 p.m.
TIME_FORMAT	Default: "H:i"	“01:24”

Example:- {{ value|date:"SHORT_DATE_FORMAT" }}

Example:- {{ value|time:“TIME_FORMAT" }}

Dot " . " Lookup:- E.g:- a.something

1) Agar wo dictionary hai to use dictionary leta hai nahi to "attribute or method" lookup krta hai, agar ye bhi nhi hai to "numeric index" jese leta hai. (ye aik built-in order hota hai . ka.)

for loop Tag:- 1) {% for Varible in variables %}
{{ variable }}
{% endfor %}

2) {% for variable in variables %}
{{variables}}
{% empty %} → mtlb agar variable empty ho
o/p1 — o/p1
{% endfor %}

3) views.py :- Salman={'names':['Ali','Salman','Ahmad']}
return render --
.html file:- {% for i in names %}
{{ i }}
{% endfor %}

Output Web:-
Ali Salman Ahmad

We have some reserved keywords for for loop in django.
e.g:- {% for i in names reversed %} it will show values of names list in descending order.

Predefined forloop variables

forloop.Parentloop → used in nested for loops

forloop.Counter → the current iteration of loop (1-indexed)

forloop.first → " first " " "

" .last → true if it's last time through the loop

" .Counter →

" .Counter0 → the current iteration of loop (0-indexed)

" .recounter → the nmbr of iterations from the end of loop (1-indexed)

" .recounter0 → " " " " (0-indexed)

{:} :- 1) views.file => std={'names':['Ali', Ahmad', 'Anus']}

{% for i in names %} ⟹ .html file

{{forloop.counter}} {{{i}}}
 Output web =>

{% endfor %}

1 Ali
2 Ahmad
3 Anus

2) views.py =>

StdDet = { 'std1':{'name':'Ali', 'Rn':1}
'std2':{'name':'Adil', 'Rn':2}
}

Students = {'Student':StdDet} => retur rend... (, , students)

.html File=> <body>

{{student}}

<h4> {{student.Std1.name}}

{{Student.Std1.RN}}

2 /h4> </body>

Output Web:-

{'Std1':{'name':'Ali', RN:1 ....

<b> Ali

1

3) Views.py

data={'Name':'Ali', 'RN':19}

return(- - -, {'data', data})

.html File=>

{% for key,value in data.items %}

<h6>{{key}}</h6> <h4> {{value}} </h4>

{% endfor %}

Output Web:-

Name

Ali

RN

19

Output Web:-

Std1

Name = Ali

Rn = 15

Std2

Name = Adil

Rn = 20

4) views.py:-

StdDet={

'Std1':{'Name':'Ali','Rn':15},

'Std2':{'Name':'Adil','Rn':20}

}

return render(req,'...html',{'StdDet':StdDet})

.html File=>

{% for key,value in StdDet.items %}

{{key}}

{% for i,j in key.items %}

{{i}} = {{j}}

{% endfor %}

{% endfor %}

settings.py → TEMPLATES = [
{APP_DIRS : True})

Templates Inside APP

Yeh by default true hota hai, agar yeh true nahi hoga to templates inside app kaam nahi karegi

nothing to do in settings.py. Just creat templates/appname/index.html and use it.

Static Files: inside Project

Static Files: → css, javascript, img, videos etc called Static files.

eg = Static Folder → css — style.css, custom.css
→ images — Pic1.jpg, Pic2.jpg

When we create static files inside Project we must add it in settings.py of Project → load it & give its ref to use in .html file.

{% static "images/love.jpg" as mylo %}

Now:-

<img src="{{mylo}}">

<img src="{% static 'css/rt.jpg' %}" /> ⟹ This is same as

<img src="{% get_static_prefix %} css/rt.jpg" />

<script src="{% static 'js/all.js' %}"></script>

</body> ⟹ included js from static/js/all.js

---

agar is line mei ⟹ {{forloop.parentloop.counter}} likh let to output:-

Std1
1 Name = Ali
1 Rn = 15
Std2
2 Name = Adil

Output web :-

click me

(1) alert i.e Welcome to JS will be shown when u click this button.

Eg => static/js/all.js file =>

function hello() { alert("Welcome to JS") }

And .html file =>

```
<form action="">
<input type="button" value="clickme" onclick="hello()"/>
</form>
<script src="{%static 'js/all.js' %}"></script>
</body>
```

from App/static.

we can use two link tags of CSS in one .html file (e.g:- 1 from Project/static & 2

Template Inheritence:- to inherite templates.

extends tag => this tag is used syntax => {% extends 'Parent templet name' %} / {% extends variable %}

e.g => 1) {% extends "./base.html" %} 2) {% extends Something %}

block tag => used to override specific part of a template.

Syntax:- {% block blockname %} ..... {% endblock blockname %}

e.g:- {% block title %} ... {% endblock title %}

more {% block %} tags in our base templates are better.

we write common codes in base template & create blocks for code which may vary page to page. later this templet will be inherited by child templet & will override created blocks.

Template Inheritance/ Template Extending

Template inheritance allows you to build a base "skeleton" template that contains all the common elements of your site and defines blocks that child templates can override.

The *extends* tag is used to inherit template.

extends tag tells the template engine that this template "extends" another template.

When the template system evaluates this template, first it locates the parent let's assume, "base.html".

At that point, the template engine will notice the block tags in base.html and replace those blocks with the contents of the child template.

You can use as many levels of inheritance as needed.

Creating Base/Parent Template and Child Template

We write common codes in base template and create blocks for code which may vary page to page. Later this template will be inherited by child templates and child template will override created blocks.

home.html

```
<html>
<head>
<title>Home</title>
</head>
<body>
<h1>Hello I am Home Page</h1>
</body>
</html>
```

about.html

```
<html>
<head>
<title>About</title>
</head>
<body>
<h1>Hello I am About Page</h1>
</body>
</html>
```

base.html

```
<html>
<head>
<title>{% block title %} {% endbock %}</title>
</head>
<body>
{% block content %} {% endblock content %}
</body>
</html>
```

home.html

```
{%  extends 'base.html' %}

{% block title %}
  Home
{% endblock %}

{% block content %}
  Hello I am Home Page
{% endblock content %}
```

about.html

```
{%  extends 'base.html' %}

{% block title %}
  About
{% endblock %}

{% block content %}
  Hello I am About Page
{% endblock content %}
```

www.geekyshows.com

```
urlpatterns = [
   path('about/', views.about),
]
```

```
<a href="/about">About</a>
<a href="{{ab}}">About</a>
```

```
def about(request):
 return render(request, 'core/about.html', {'ab':'/about'})
```

```
<a href="{% url 'aboutus' %}">About</a>
```

```
{% url 'aboutus' as abc %}
<a href="{{abc}}">About</a>
urlpatterns = [ path('about/', views.about, name='aboutus'),]
```

fontAwesome → contain multiple icons that are very useful in web dev. B
a website like bootstrap

base.html
Eg:- <html>
<head>
<title> {% block title %} - {% endblock %} </title>
</head>
<body> {% block content %} {% endblock content %} </body></html>

yaha par default value bhi de sakte hai.

home.html

{% extends 'base.html' %}

{% block title %} Home {% endblock %}

{% block content %} This is home Page {% endblock %}

2) base.html => <title> {% block title %} Win {% endblock %}
and home.html => {% block title %} {{block.super}} Home {% endblock %}
then, the output title will be => Win Home .

agar base.html file Project/templates mai hai to, derived.html
mai => {% extends 'base.html %} likhe gay.
Lekin agar base.html file kisi app mai hai to, derived.html
mai => {% extends 'appName/base.html' %}

Use Bootstrap :- (manually/manu without starter template)
→ download compiled CSS & js file (from bootstrap/downloads)
→ download Popper.js file (from a link below this ↑ file download button

→ download jquery (from jquery Web)

Now just leave:- bootstrap.min.js
jquery.js
Popper.js files in js folder

And bootstrap.min.css in css folder

Now copy these files to their proper static directories, load it in .html files and now u can use bootstrape.

urlPatterns = [Path('about\', views.about)

<a href="{{ab}}">About</a>

def about(req):
return render(req, 'core/about.html', {'ab': '/about'})

more Proper useable than above two methods.

Url Tag :- {%}=> route / f(x) to be called / url name

urlPatters = [Path('about', views.about, name='aboutUs')]

.html file => <a href="{% url 'aboutUs' %}">About</a>

=> {% url 'aboutUs' as abt %}
<a href="{{abt}}">About</a>

= Same as this code's output.

E.g => 1) url for core APP => views.py file = Path('', views.core, name='Home')

core.html => <a href="{% url 'Home' %}">AnyText(eg = Reload theme or PageUP etc)</a>

2) url tag in core app for courses app:-

→ urls.py of coursesapp = Path('', views.courses, name='allCourses')

→ .html of Core app = <a href="{% url 'allCourses' %}">Courses</a>

Include Tag:- it loads a whole template & renders it with current context.

Syntax:- 1) {% include "temp_var_name" %}

2) {% include "templateName.html" %}

E.g:- 1) Include from current APP:-

→ topCourses.html = <h1>Java
Python
Dj, art</h1>

→ core.html = {% includ 'core/topcourses.html' %}

Now content of topCourses.html will be shown in core.html file web.

agar is ki jga only hai to Phir sirf P var ovvr jgho pe access hoga bhai vars nahi

Agar core.html = {% includ 'core/...' with P='PHP' %}

over topCourses.html = {{P}} kr dai to P ki value topCourse.html mei add ho jayegi.

Agar views fn(x) se koi ~~Dictionary~~ Dictionary var hain ne core.html mei likha

hai (e.g. = {{d}}), to us ko topCourses ke .html mei bhi add kr sktay hai (syntax = {{d}}.
key Name

Cookies:- hmare using behaviour ko store kr k web server kerta hai.
hme jo Preference (ads, other features) Provide krta hai Cookies

ORM (object relational mapper):- it enables app to interact with database (e.g: SQLite). It maps object attributes to respective table fields using django models.

QuerySet:- a list containing all those objects we have created
It allows to read data from database, filter it and order it.
It's generally associated with CRUD apps.

Creating Models:- go to models.py file of targetted app, →

from django.db.models ——
eg = ①
Class data(models.Model):
    stId = IntegerField()
    stName = models.CharField(max_length=20)
    required in charfield

Syntax:- class ClassName(models.Model):
    fieldName = models.Field type(arg, options)

① e.g ⇒ stId = models.IntegerField(Primary_key = True)
    Comment = models.CharField(max_length=40, default='Nice')

Now Run:- 1) Python manage.py makemigrations (model class ko sql query mei convert krta hai)
2) Python manage.py sqlmigrate appName 0001 (to see sql query in terminal)
3) Python manage.py migrate (us query ko execute krta k db mei table bna deta hai.) (migrations from file name only first integers)

all() Method:- It returns a copy of QuerySet or QS subclass.
Syntax ⇒ ModelClassName.objects.all()

Object Relational Mapper (ORM)

Object-Relational Mapper (ORM), which enables application to interact with database such as SQLite, MySQL, PostgreSQL, Oracle.

ORMs automatically create a database schema from defined classes or models. It generate SQL from Python code for a particular database which means developer do not need to write SQL Code.

ORM maps objects attributes to respective table fields.

It is easier to change the database if we use ORMs hence project becomes more portable.

Django's ORM is just a way to create SQL to query and manipulate your database and get results in a pythonic fashion.

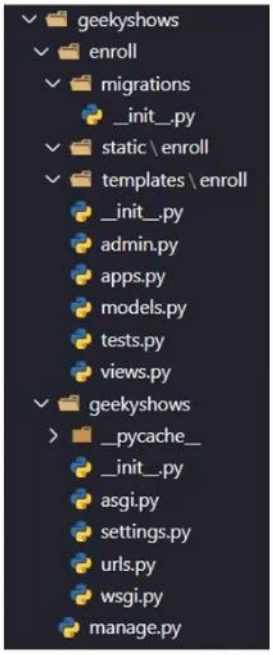
ORMs use connectors to connect databases with a web application.

Create Our Own Model Class

models.py

```
class Student(models.Model):
        stuid=models.IntegerField()
        stuname=models.CharField(max_length=70)
        stuemail=models.EmailField(max_length=70)
        stupass=models.CharField(max_length=70)
```

```
CREATE TABLE "enroll_student" (
        "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
        "stuid" integer NOT NULL,
        "stuname" varchar(70) NOT NULL,
        "stuemail" varchar(70) NOT NULL,
        "stupass" varchar(70) NOT NULL
);
```



Object Relation Mapper (ORM)

```
class Student(models.Model):
        stuid=models.IntegerField()
        stuname=models.CharField(max_length=70)
        stuemail=models.EmailField(max_length=70)
        stupass=models.CharField(max_length=70)
```

```
CREATE TABLE "enroll_student" (
        "id" integer NOT NULL PRIMARY KEY
AUTOINCREMENT,
        "stuid" integer NOT NULL,
        "stuname" varchar(70) NOT NULL,
        "stuemail" varchar(70) NOT NULL,
        "stupass" varchar(70) NOT NULL
  );
```

id	stuid	stuname	stuemail	stupass

QuerySet

A QuerySet can be defined as a list containing all those objects we have created using the Django model.

QuerySets allow you to read the data from the database, filter it and order it.

Querysets are generally associated with CRUD applications.

Model

A model is the single, definitive source of information about your data.

It contains the essential fields and behaviors of the data you're storing.

Generally, each model maps to a single database table.

Model Class

Model class is a class which will represent a table in database.

Each model is a Python class that subclasses django.db.models.Model

Each attribute of the model represents a database field.

With all of this, Django gives you an automatically-generated database-access API
Django provides built-in database by default that is sqlite database.

We can use other database like MySQL, Oracle SQL etc.

Create Our Own Model Class

models.py file which is inside application folder, is required to create our own model class.

Our own model class will inherit Python's Model Class.

Syntax:-

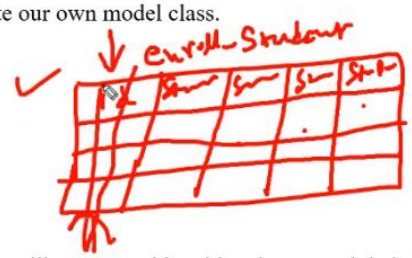
```
class ClassName(models.Model):
        field_name=models.FieldType(arg, options)
```

Example:-

Length is required in CharField Type

models.py

```
class Student(models.Model):
    stuid=models.IntegerField()
    stuname=models.CharField(max_length=70)
    stuemail=models.EmailField(max_length=70)
    stupass=models.CharField(max_length=70)
```



- This class will create a table with columns and their data types
- Table Name will be ApplicationName_ClassName, in this case it will be enroll_student
- Field name will become table's Column Name, in this case it will be stuid, stuname, stuemail, stupass with their data type.
- As we have not mentioned primary key in any of these columns so it will automatically create a new column named 'id' Data Type Integer with primary key and auto increment.

Admin Application

It is a built-in application provided by Django.

This application provides admin interface for CRUD operations without writing sql statements.

It reads metadata from your models to provide a quick, model-centric interface where trusted users can manage content on your site.

Admin Application can be accessed using http://127.0.0.1:8000/admin

Super User is required to login into Admin Application

Writing Code to get DB data in views.py

First import your own model class from models.py

Example:-

views.py

```
from enroll.models import Student
def studentinfo(request):
        stud = Student.objects.all()
        return render(request, 'enroll/studetails.html', {'stu':stud})
```

Get Data from views.py in template file

templates/enroll / studetails.html

```
<!DOCTYPE html>
<html>
    <body>
        <h1>Student Page</h1>
        {% if stu  %}
            <h1>Show Data</h1>
            {%  for st in stu %}
                <h5>{{st.stuid}}</h5>
                <h5>{{st.stuname}}</h5>
            {% endfor %}
        {% else %}
            <h1>No Data</h1>
        {% endif %}
    </body>
</html>
```

✓ flake8 = a good linter in Vscod

Pylint = also good.

get() Method :- e.g = def ... {aik specific table row ko access krny k liye} yha par idnmbr likha hai

onestudDet = Stud.objects.get(pk = 2)

return render(----, {'st': onestudDet})

→ Python manage.py createsuperuser (to get access to admin Pannel)

To register model class :- → go to admin.py file of all →

from appName.models import className | model is now registered

admin.site.register(className) | and u can see it now in admin Pannel.

__str__() method in models.py File :- Syntax ⇒ def __str__(self):
return self.fieldName

E.g ⇒ 1) class stDetC--
Name = models.CharField(max_length=20)
def __str__(self):
return self.stdName

Now u will see names u have given in table's inputs instead of class obj 1, 2, 3 e.t.c

2) def __str__(self):
return str(self.stdid)

agar int field hai to usko str mei iss trah convert krna hoga.

ModelAdmin Class :- to show table's all data on single page in admin Interface

MAC is will be written in admin.py file ⇒

e.g :- class ModelclassNameAdmin(admin.ModelAdmin):
list_display = ('ModelclassAttribute1', '2', ...)
admin.site.register(className, classNameAdmin)

Now u will see all these attributes u have written here of all table' objects on single page in admin Pannel.

Register Model by decorators:-

a decorator is used to register ModelAdmin classes.

Syntax:- @admin.register(ModelclassName1, 2, 3)

admin.py File =>

E.g. => @admin.register(allData)
class allDataAdmin(admin.ModelAdmin):
list_display = ('id', 'Name' ...)

Better Practice than just ModelAdmin class. ~~without~~ means without decorator

Django Form:- → create forms.Py file in app → write

from django import forms

class FormName(forms.Form):
id = IntegerField()
Name = CharField()

views.Py file =>
from appName.forms import formName

def ---
any variable (regform = formName())
return render(request, '', {'regform': regform})

html file:-
{{regForm}}

agar iss mei ~~[illegible]~~ auto_id=True kr dey to har field k label attribut mei id us field ka name bn jaye gi jo hum ne forms.Py mei banaye hai.
(a Pic available for clot) optional

Form Rendering Options:-

- {{form}} → will render them all
- {{form.as_table}} → render them as table cells wrapped in <tr> tags.
- {{form.as_P}} → render them wrapped in <P> tags.
- {{form.as_ul}} → render them all in <li> tags.
- {{form.name_of_any_field}} from forms.Py file

E.g:-
<form action="" method="get">
{{form.as_P}}
<input type="submit" value="Sab">
</form>

Output:- will be in Paragraph form.

django form API har attribute name k bad ":" add krta hai, agar is ko hatana hai to views.py file mei

def —
fm = RegForm Name / label_suffix = (' ') → agar kuch our lgana hai to idher likh sktay hai. e.g:- space, char etc

def —
regForm = formName (initial = {'formAttributeName' : 'default text to display on that option at run time.'}
e.g = Name [Mr.]

forms.py

Fether_Name = charField() | ye underscore ko output mei space mei convrt krta hai our har attribute k pehle letter ko capital bna deta hai.

Ordering Form Fields:- views.py →

def...
e.g ⇒ fm = formName()
fm.order_fields(field_order = ['1st form attribute Name', '2nd', '3rd' ...])

Render FormFields Manually :- E.g ⇒ forms.py ⇒ name = forms.charField()

html file ⇒
<label for="{{form.name.id_for_label}}">Name:</label>
{{form.name}}

Output ⇒ Name: [input char box]

Render a single field in .html file:->

{{ form.fieldName.label_tag }}
{{ form.fieldName }}

{{ form.fieldName.label }} -> to see label of a field E.g = Name, etc Email,

{{ form.fieldName.value }} to see value of a field.

forms.py => name = forms.CharField(help_text = " maximum 30 chars allowed")

.html => {{ form.name.help_text }}

Access form Fields using loop :- E.g =>

<form action="" method="get">
{% for avar in dicName forms %}
{{ field.label_tag }}
{{ formdicName }}
{% endfor %} </form>

agar iss jgha visible nhi likha jay to jo hmare hidden fields hongay un ka label by default output pe show hoga.

E.g 2) => forms.py => field = forms.IntegerField(widget = forms.HiddenInput())

.html file => <form action="" method="get">

{% for avar in formdicName.visible_field %}
<P> Following are visible fields </P>
{{ field.label_tag }}
{{ field }} {% endfor %}

Form Field Aurguments:-

1) name = forms.CharField(label='Your Name')

output:- Your Name: [ ]

(label_suffix = ' = ') → Your Name = [ ]

(initial='Mr.') → Your Name = [Mr.]

(require = False) ⇒ by default true hota hai, agar false ho to is ko blank chor sktay hai warna true pe error koi input value do gay

(disabled = True) ⇒ agar ye true ho to is field ki value un-editable ho jaye gi run time mei. by defaulet = False

(helptext='ur full Name') → Your Name [ ] ur full Name

views.py file ⇒ def fm = regForm(intial = {'fieldName': 'Initial Value'}
initial argument ko idhar ↑ bhi iss trah likha ja skta hai.

E.g ⇒ ob = forms.CharField(widget=CheckboxInput(1) / PasswordInput etc) → in k bghir bhi kam kr sta hai

mare wo form vars jin k liye koi alag field django forms mei mojod nhi hai to phir hum widget=regfield ka istimal krty hai. E.g upper likh hai

E.g=2) (widget=TextArea/HiddenInput/FileInput

E.g=3) nm = forms.CharField(widget=TextInput(attrs={'class': 'myclass'}))
ab agar inspect view mei is ka html tag dekhe gay to iss field k input tag mei ak class="myclass" k name say ho gy.
Is ki zrorat tb pati hai jb hm class ko use kr k CSS apply krna chahtey hai.

Configure id attribute

auto_id – The id attribute values are generated by prepending id_ to the form field names. This behavior is configurable, though, if you want to change the id convention or remove HTML id attributes and <label> tags entirely.

Use the auto_id argument to the Form constructor to control the id and label behavior. This argument must be True, False or a string. By default, auto_id is set to the string 'id_%s'.

Example:-

fm = StudentRegistration(auto_id=False)

fm = StudentRegistration(auto_id='geeky')

- If auto_id is set to a string containing the format character '%s', then the form output will include <label> tags, and will generate id attributes based on the format string.
- If auto_id is set to True, then the form output will include <label> tags and will use the field name as its id for each form field.
- If auto_id is False, then the form output will not include <label> tags nor id attributes.
- If auto_id is set to any other true value – such as a string that doesn't include %s – then the library will act as if auto_id is True.

```
def showformdata(request):
 fm = StudentRegistration(auto_id='some_%s')
 return render(request, 'enroll/userregistration.html',
 {'form':fm})
```

GET and POST

- GET should be used only for requests that do not affect the state of the system.
- Any request that could be used to change the state of the system should use POST.
- GET would also be unsuitable for a password form, because the password would appear in the URL, and thus, also in browser history and server logs, all in plain text. Neither would it be suitable for large quantities of data, or for binary data, such as an image. A Web application that uses GET requests for admin forms is a security risk: it can be easy for an attacker to mimic a form's request to gain access to sensitive parts of the system.
- POST, coupled with other protections like Django's CSRF protection offers more control over access.
- GET, by contrast, bundles the submitted data into a string, and uses this to compose a URL. The URL contains the address where the data must be sent, as well as the data keys and values.
- Django's login form is returned using the POST method, in which the browser bundles up the form data, encodes it for transmission, sends it to the server, and then receives back its response.
- GET is suitable for things like a web search form, because the URLs that represent a GET request can easily be bookmarked, shared, or resubmitted.

Get Django Form Data in Terminal

1. First of all Create form inside forms.py file

forms.py

```
from django import forms
class StudentRegistration(forms.Form):
    name=forms.CharField()
    email=forms.EmailField()
```

2. Get submitted Data in views.py

views.py

```
from .forms import StudentRegistration
def showformdata(request):
    if request.method=='POST':
        fm = StudentRegistration(request.POST)
        if fm.is_valid():
            print('Form Validated')
            print('Name:', fm.cleaned_data['name'])
            print('email:', fm.cleaned_data['email'])
    else:
        fm = StudentRegistration()
    return render(request, 'enroll/userregistration.html', {'form':fm})
```

3. Get object from views.py in template file

templates/enroll / userregistration.html

```
<!DOCTYPE html>
<html>
    <body>
        <form method="POST"> {% csrf_token%}
            {{form.as_p}}
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

www.geekyshows.com

Pic 21 => Get & Post Method

[redacted] CSRF/XSRF → cross site request Forgery (used to Properly secure our form/Payment Method e.t.c from hackers)

if we want to use CSRF Protection then we need to add csrf_token inside form tag.

Syntax => <form method="Post">{% csrf_token %}
{{form}}
</form>

it will add a hidden input field with a random value that changes continouslly on every reload Page/refresh Page in form tag when u see it's [redacted] view option on web.
Page Source

Pic 23 ~ 24 must have.

html => <form ... novalidate> ... </form>

forms.Py => nm = CharField(error_messages={'required': 'any message to show'} when the field is left blanked instead of it's default html Provided by django forms message.

=> CharField(strip=False) by default true hota hai agar ye false hai to input me hum jo bhi spaces denge un ko wesse hi show kray
Eg. E.g: Ali Kh as

=> usrfieldempty_value = 'any text value etc') → null agar hum is field ko blank chor dey to is value ko input me le lo.

agree = forms.BooleanField() → ☐ (check box deta hai by default = False)

=> CharField(min_length=4, max_length=10) an e.g

=> IntegerField(min_value= , max_value=)

=> DecimalField(min_value= , max_value= , max_digit= , decimal_Places=)
total number of digits
"." k baad kitne digits hone chahiye

regForm > views.py > index

```python
from  regForm.forms  import regForm
# Create your views here.
def index(request):
    if request.method=='POST':
        fm=regForm(request.POST)
        # Getting valid data by django to be submitted according to their required field syntax e.g : email
        # must contain @ symbol etc. Using is_valid() method.
        if fm.is_valid():
            # Capturing form's data using cleaned_data['formFieldName'] method.
            name=(fm.cleaned_data['name'])
            email=('Email: ',fm.cleaned_data['email'])
            password=('Password: ',fm.cleaned_data['password'])
            print('this is POST request')
            #showing message of form Submitted on another web page
            return render(request,'regForm/formSubmitted.html',{'name':name})
    else:
        fm=regForm()
        print('this is GET request')
    return render (request,'regForm/regForm.html',{'form':fm})
```

regForm > views.py > ...

```python
1  from django.shortcuts import render
2
3  from  regForm.forms  import regForm
4
5  from django.http import HttpResponseRedirect
6  def formSubmitted(request):
7      return render(request,'regForm/formSubmitted.html')
8
9  # Create your views here.
10 def index(request):
11     if request.method=='POST':
12         fm=regForm(request.POST)
13         # Getting valid data by django to be submitted according to their required field syntax e.g : email
14         # must contain @ symbol etc. Using is_valid() method.
15         if fm.is_valid():
16             # Capturing form's data using cleaned_data['formFieldName'] method.
17             name=(fm.cleaned_data['name'])
18             email=('Email: ',fm.cleaned_data['email'])
19             password=('Password: ',fm.cleaned_data['password'])
20
21             return HttpResponseRedirect('/formSubmitted/')
22
23             print('this is POST request')
24             #showing message of form Submitted on another web page
25             #return render(request,'regForm/formSubmitted.html',{'name':name})
26             #by default this page will be shown in the same url to change it we wrote line:5,6,7,24 and
27             # path('formSubmitted/',views.formSubmitted) code in the urls.py of reForm app
28
29     else: ...
32     return render (request,'regForm/regForm.html',{'form':fm})
```

```
comment=forms.SlugField()
email=forms.EmailField(min_length=5, max_length=50)
website=forms.URLField(min_length=5, max_length=50)
password=forms.CharField(min_length=5, max_length=50,
widget=forms.PasswordInput)
description=forms.CharField(widget=forms.Textarea)
feedback=forms.CharField(min_length=5, max_length=50,
widget=forms.TextInput(attrs={'class':'somecss1 somecss2',
'id':'uniqueid'}))
notes=forms.CharField(widget=forms.Textarea(attrs={'rows':3,
'cols':50}))

agree = forms.BooleanField( label_suffix='', label='I Agree')
```

⇒ FloatField() → is mei jo bhi number doge, output mei uss ke saath, by default ".0" aye ga. E.g = 5 output = 5.0

Pic → 25 here

defining validation and our custom error for a Field by making our own fcn in forms.Py class—

E.g ⇒ def clean_nameofField(self):
    avar = self.cleaned_data['nameofField']
    if len(avar) < or = 4:
        raise forms.ValidationError('ur custom error')
    return avar

ye chiz se bhi error message likh sakte hai. wo to bs aik concept hai.

defining validation for full form means for all fields in forms.Py → .html file ⇒ <form — novalidate> x/forms

class—
    name = charfiel—
    email = e—
    def clean(self):
        cleaned_data = super().clean()
        nameVar = self.cleaned_data['name']
        emailVar = " " " ['email']
        if len(nameVar) < 4:
            raise forms.ValidationError('ur custom error')
        if len(emailVar) — —

define all ur fields here and write conditions for them below

To use django built-in validator:- forms.Py → e.g →

from django.core import validators
class— forms.
    name = charfield(validors = [validators.MaxlengthValidator(5)])
    MinLengthV etc

Custom validor :→ forms.Py ⇒

```
from django.core import validators

def funName(aValue):
    if (aValue[0])>='a' and aValue[0]<='z':
        raise forms.ValidationError('ur error')
```

E.g = 'Name should start with capital alphabet'
of above code.

```
class formName(forms.Form):
    name = charfield(validators=[funName])
```

Matching re-Enter Password:- forms.py->

```
Passord = charfield(widget=PasswordInput())
rPassord = ------
def clean(self):
    cleaned_data = super().clean()
    pVar = self.cleaned_data.get('Password')
    rPvar = " " " " ('rPassw')
    if pVar != rPvar:
        raise forms.ValidationError('ur error')
```

To apply css on errors → right click on web page → view source page →

note the class → use that class name for applying css.

E.g => .html file → <styles> .errorlists { color: Red; </styles>

Pic 26 another way of deeply appling css on errors.

```
<E.g=> <div>
{{ dicName.label_tag }} {{ dicNam }}
{% for error in dicName.errors %}
<span class="errorlist">{{error}}</span>
{% end for %} </div>
```

to capture errors

```
17 <body>
18  <form action="" method="POST" novalidate>
19  {% csrf_token %}
20   {{form.non_field_errors}}
21   {% for field in form %}
22    <div>
23     {{field.label_tag}} {{field}} <span class="errorlist">{
       {field.errors|striptags}}</span>
24    </div>
25   {% endfor %}
26   <input type="submit" value="Submit">
27  </form>
28 </body>
29 </html>
```

25

<script src="{% static 'js/jquery.js'
js/popper.js
js/bootstrap

Pehle jquer ko likha hai, phir bootstrap.js ko warna ye kaam nhi krayega

forms.py

class - - -
error_css_class='error'
required_css_class='required'
name = charfield - -

ab iss class ka use kr k css apply kr sktay hai hum errors pr
same as error_css_class

Pic=27,27a,b - must here

models.py → name = models.CharField(blank=True) ab hum iss field ko khali chor sktay hai.

ModelForm:- more useful than above method. Pic → 29, 29b must here

Dynamic Urls:- Pic 30, 32, 31, 33

forms.py → class regF(forms.ModelForm):
Class Meta:
1) model=model form Name
2) fields= '__all__' mtlb saray fields ko include kro.
↓
fields=['name', 'Pass'] mtlb sirf inn attrs ko add kro jo is list mei hai.
↓
exclude=['name'] mtlb is k ilawa sab fields ko include kro
exclude =('name',) → same as

ModeForm Inheritence E.g => forms.py and models.py

from django import forms
from .models import modelFormName
class formName(forms.ModelForm):
Class Meta:
model = modelFormName,
fields=['name', 'email', 'Pass']
Class SubFormName(ParentFormName):
Class Meta(ParentFormName.Meta):
fields = ['teacherName', 'email', 'Pass']

from django.db import models
Class modelClassNam(models.Model):
name = models.CharField(
teacherName = models.
email= - - -
Pass = - - - -

Pic=C1 to C11 here

Project 4
Imp here

27,27a….

regForm > forms.py > regForm

```python
from django import forms
from django.forms.fields import CharField, IntegerField
from django.forms.widgets import CheckboxInput, FileInput, PasswordInput, TextInput
from django.core import validators

class regForm(forms.Form):
    name=forms.CharField(error_messages={'required':'enter ur name'})
    email=forms.EmailField(error_messages={'required':'enter ur email'})
    password=forms.CharField(widget=PasswordInput(),error_messages={'required':'enter ur password'})


```

regForm > models.py > User

```python
from django.forms import widgets

# Create your models here.
class User(models.Model):
    name=models.CharField(max_length=50)
    email=models.CharField(max_length=50)
    password=models.CharField(max_length=50)
```

regForm > admin.py > User

```python
from django.contrib import admin

# Register your models here.
from regForm.models import User
@admin.register(User)
class User(admin.ModelAdmin):
    list_display=('id','name','email','password')
```

regForm > views.py > index

```python

from .models import User
def index(request):
    if request.method=='POST':
        fm=regForm(request.POST)
        if fm.is_valid():
            nm=fm.cleaned_data['name']
            em=fm.cleaned_data['email']
            ps=fm.cleaned_data['password']

            #save form data in db
            dbP=User(name=nm,email=em,password=ps)
            dbP.save()

             # update form data
            # dbP=User(id=1,name=nm,email=em,password=ps)#this line means update data of id=1
            # dbP.save()

             # delete form data
            # dbP=User(id=1)#mtlb id=1 pe jo data hai us ko delete krdo.
            # dbP.delete()
    else:
        fm=regForm()
    return render(request,'regForm/regForm.html',{'form':fm})
```

Model Form

```
class Registration(forms.ModelForm):
    class Meta:
        model = User
        fields = ['name', 'password', 'email']
        labels = {'name': 'Enter Name', 'password': 'Enter Password', 'email': 'Enter Email' }
        help_text = {'name': 'Enter Your Full Name' }
        error_messages = {'name': {'required': 'Naam Likhna Jaruri Hai'},
                          'password': {'required': 'Password Likhna Jaruri Hai'} }
        widgets = { 'password': forms.PasswordInput,
                    'name': forms.TextInput(attrs={'class': 'myclass', 'placeholder': 'Yaha Naam likhe'}),  }
```

29,29a…

modelForm > app > forms.py > regForm

```
3   from django.core import validators
4
5   from app.models import User
6   class regForm(forms.ModelForm):
7       #name=forms.CharField(max_length=10,required=False)#models.py me jo name field hai uss ko override krday ga
8       class Meta:                     #Meta , model , fields,labels all are predifiened attributes
9           model=User
10          fields=['name','email','password']
11          #agr label dena chahtai hai to ye trika hai. by default Name: ,Email: is trah hota hai.
12          labels={'name':'Enter Ur Name','email':'enter Email'}
13          help_text={'name':'helped_text:please enter ur name to submit'}
14          error_messages={'name':{'required':'entering ur name is compulsry(an error message)'},
15                          'password':{'required':'password leaved blanked error'}}
16          widgets={'password':forms.PasswordInput(),
17                   'email':forms.EmailInput(),
18                   'name':forms.TextInput(attrs={'class':'name','placeholder':'ur name here(placeholder)'})}
```

modelForm > app > views.py > app

```
1   from django.shortcuts import render
2   from .forms import regForm
3   from .models import User
4   # Create your views here.
5   def app(request):
6       if request.method=='POST':
7           fm=regForm(request.POST)    #To update a form data write this code instead of line 8 to 13.
8           if fm.is_valid():               #pi=User.object.get(pk=1)
9               nm=fm.cleaned_data['name']  #fm=regForm(request.POST,instance=pi)
10              em=fm.cleaned_data['email'] #if fm.is_valid():
11              ps=fm.cleaned_data['password']  #fm.save()
12              aVar=User(name=nm,email=em,password=ps) #linkin modelForm to db
13              aVar.save() #saving modelForm data in db
14      else:
15          fm=regForm()
16
17      return render(request,'app/app.html',{'form':fm})
```

30,32,31,33

EXPLORER

- 13
  - dynamicUrls
    - dynamicUrls
      - __pycache__
      - __init__.py
      - asgi.py
      - settings.py
      - urls.py
      - wsgi.py
    - myApp
      - __pycache__
      - migrations
      - templates\myApp
        - myApp.html
        - showMyApp.html
      - __init__.py
      - admin.py
      - apps.py
      - models.py
      - tests.py
      - views.py
    - db.sqlite3
    - manage.py

myApp.html | showMyApp.html | views.py | urls.py

dynamicUrls > dynamicUrls > urls.py > ...

```python
from django.contrib import admin
from django.urls import path
from myApp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('myApp',views.myApp,name='myApp'),
    path('showMyApp/<aVar>/',views.showMyApp,name='showMyApp')
]

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

dynamicUrls > myApp > views.py > myApp

```python
from django.shortcuts import render

# Create your views here.

def myApp(request):
    return render(request,'myApp/myApp.html')


def showMyApp(request,aVar):
    return render(request,'myApp/showMyApp.html',{'aVar':aVar})
```

dynamicUrls > myApp > templates > myApp > myApp.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>myApp</title>
</head>
<body>
    <h1>home . html</h1>
    <a href="{% url 'showMyApp' 1 %}">Student 1 Data</a>    <br>
    <a href="{% url 'showMyApp' 2 %}">Student 2 Data</a>    <br>
    <a href="{% url 'showMyApp' 3 %}">Student 3 Data</a>    <br>

</body>
</html>
```

dynamicUrls > myApp > templates > myApp > showMyApp.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>show</title>
</head>
<body>
    <h2>{{aVar}}</h2>
    <br><a href="{% url 'myApp'  %}">Back to Home</a>
</body>
</html>
```

Path Converters

- str - Matches any non-empty string, excluding the path separator, '/'. This is the default if a converter isn't included in the expression.

- int - Matches zero or any positive integer. Returns an int.

- slug - Matches any slug string consisting of ASCII letters or numbers, plus the hyphen and underscore characters. For example, building-your-1st-django-site.

- uuid - Matches a formatted UUID. To prevent multiple URLs from mapping to the same page, dashes must be included and letters must be lowercase. For example, 075194d3-6885-417e-a8a8-6c931e272f00. Returns a UUID instance.

- path - Matches any non-empty string, including the path separator, '/'. This allows you to match against a complete URL path rather than a segment of a URL path as with str.

Passing extra options to view functions

path(route, view, kwargs=None, name=None)

The kwargs argument allows you to pass additional arguments to the view function or method. It should be a dictionary.

urls.py

```
urlpatterns = [
        path(route, view, kwargs=None, name=None)
        ]
```

urls.py

```
urlpatterns = [
        path('', views.home, {'check': 'OK'}),
        ]
```

views.py

```
def home(request, check):
        print(check)
```

www.geekyshows.c

dynamicUrls > urls.py > ...

```python
from django.contrib import admin
from django.urls import path
from myApp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('myApp',views.myApp,name='myApp'),
    #Pic 32  /int:<aVar>/,/str:<aVar>/ by default string hota hai, iss trah is ko cnvrt kiya jata hai.
    path('showMyApp/<int:aVar>/',views.showMyApp,name='showMyApp'),
    path('showMyApp/<int:aVar>/<int:subAVar>/',views.subShowMyApp,name='subShowMyApp')

]
```

myApp > views.py > ...

```python
from django.shortcuts import render

# Create your views here.

def myApp(request):
    return render(request,'myApp/myApp.html')


def showMyApp(request,aVar):
    if aVar==1:
        aDic={'id':aVar,'name':'Salman'}
    if aVar==2:
        aDic={'id':aVar,'name':'Ali'}
    if aVar==3:
        aDic={'id':3,'name':'Ahmad'}

    return render(request,'myApp/showMyApp.html',aDic)

def subShowMyApp(request,aVar,subAVar):
    if aVar==1 and subAVar==1:
        aDic={'id':aVar,'name':'Salman','info':'my sub details'}
    if aVar==2 and subAVar==2:
        aDic={'id':aVar,'name':'Ali', 'info':'ali k sub details'}
    if aVar==3 and subAVar==3:
        aDic={'id':3,'name':'Ahmad','info':'ahmad k sub details'}

    return render(request,'myApp/showMyApp.html',aDic)
```

myApp > templates > myApp > showMyApp.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>show</title>
</head>
<body>
  Id:  {{id}}<br>Name:  {{name}} <br> Sub Info: {{info}}

    <br><br><a href="{% url 'myApp'  %}">Back to Home</a>
</body>
</html>
```

myApp > templates > myApp > myApp.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>myApp</title>
</head>
<body>
    <h1>home . html</h1>
    <a href="{% url 'showMyApp' 1 %}">Student 1 Data</a>    <br>
    <a href="{% url 'showMyApp' 2 %}">Student 2 Data</a>    <br>
    <a href="{% url 'showMyApp' 3 %}">Student 3 Data</a>    <br>


    <a href="{% url 'subShowMyApp' 1 1%}">Student 1 Sub Data</a>    <br>
    <a href="{% url 'subShowMyApp' 2 2%}">Student 2 Sub Data</a>    <br>
    <a href="{% url 'subShowMyApp' 3 3%}">Student 3 Sub Data</a>    <br>

</body>
</html>
```

C1 to c11:

urls.py - crudProject - Visual Studio Code

crudProject > urls.py > ...

```python
"""crudProject URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path,include

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

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6
```

urls.py - crudProject - Visual Studio Code

formApp > urls.py > ...

```python
from django.urls import path
from .import views
urlpatterns = [
    path('',views.base,name='base'),
    path('/update/<int:stdId>',views.update,name='update'),
    path('deleted/<int:stdId>/',views.delete,name='delete')
]
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\#1 Programming Content\#1 Web Devolopment\Django\FavPL\allDjangoProjects\14.CRUD Project\crudProject>
```

models.py

formApp > models.py > addStd

```python
from django.forms import IntegerField

# Create your models here.
class addStd(models.Model):
    stdId=models.IntegerField(primary_key=True)
    stdName=models.CharField(max_length=25)
    stdEmail=models.EmailField()
    stdPassword=models.CharField(max_length=25)

    def __str__(self):
        return str(self.stdName)

```

admin.py

formApp > admin.py > addStdAdmin

```python
from django.contrib import admin
from .models import addStd

@admin.register(addStd)
class addStdAdmin(admin.ModelAdmin):
    list_display=('stdId','stdName','stdEmail','stdPassword')
```

forms.py

formApp > forms.py > ...

```python
from django import forms
from .models import addStd
class addStdForm(forms.ModelForm):
    class Meta:
        model=addStd
        fields=['stdId','stdName','stdEmail','stdPassword']

        labels={
            'stdId':'Id ','stdName':'Name ','stdEmail':'Email ','stdPassword':'Password '
        }

        widgets={
            'stdId':forms.NumberInput(attrs={'class':'form-control'}),
            'stdName':forms.TextInput(attrs={'class':'form-control'}),
            'stdEmail':forms.EmailInput(attrs={'class':'form-control'}),
            'stdPassword':forms.PasswordInput(render_value=True,attrs={'class':'form-control'})
            }
```

views.py

formApp > views.py > base

```python
from django.shortcuts import render
from .forms import addStdForm
from .models import addStd
from django.http import HttpResponseRedirect
# Create your views here.
def base(request):
    if request.method=='POST':
        form=addStdForm(request.POST)
        if form.is_valid():
            form.save()
            form=addStdForm()
    else:
        form=addStdForm()
    allDbData=addStd.objects.all()
    context={'form':form,'allDbData':allDbData}
    return render(request,'formApp/base.html',context)

def update(request,stdId):
    if request.method=='POST':
        upd=addStd.objects.get(pk=stdId)
        newUpd=addStdForm(request.POST,instance=upd)
        if newUpd.is_valid():
            newUpd.save()
    else:
        upd=addStd.objects.get(pk=stdId)
        newUpd=addStdForm(instance=upd)
    return render(request,'formApp/update.html',{'newUpd':newUpd})


```

```python

def delete(request,stdId):
    if request.method=='POST':
        dlt=addStd.objects.get(pk=stdId)
        dlt.delete()
        return HttpResponseRedirect('/')

```

base.html

formApp > templates > formApp > base.html

```html
<!DOCTYPE html>
{%load static %}
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <link rel='stylesheet' href='{%static "formApp/css/all.min.css"%}'>
  <link rel='stylesheet' href='{%static "formApp/css/bootstrap.css"%}'>
  <link rel='stylesheet' href='{%static "formApp/css/custom.css"%}'>
  <title>CRUD Salman19861</title>
</head>
<body>

<div class="container mt-5">
    <h2 class="text-center alert alert-info">ModelForm CRUD Project</h2>
    {%block content%}

<div class="row">

    <div class="col-sm-4">
    <h4 class="text-center mt-3">Add New Student</h4>
    <form action="" class="mt-4" method="POST">{% csrf_token %}
        {{form.as_p}}
       <div> <input type="submit" class="basebtn btn btn-outline-info" value="Add"></div>
    </form>
    </div>

    <div class="col-sm-8">
        <h4 class="text-center mt-3">All Students Data</h4>
        {%if allDbData%}


        <table class="table table-striped table-hover mx-5">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Name</th>
                    <th>Email</th>
                    <th>Password</th>
                    <th>Action</th>
                </tr>
            </thead>

            <tbody>
                {%for rowData in allDbData%}
                <tr>
                    <th scope="row">{{rowData.stdId}}</th>
                    <td>{{rowData.stdName}}</td>
                    <td>{{rowData.stdEmail}}</td>
                    <td>{{rowData.stdPassword}}</td>
                    <td>
                        <a href="{%url 'update' rowData.stdId%}" class="btn btn-outline-info">Edit</a>

                        <form action="{%url 'delete' rowData.stdId %}" class="d-inline mx-1" method="POST">{% csrf_token %
                            <input type="submit" value="Delete" class="btn btn-outline-danger">
                        </form>
                    </td>
                </tr>
```

```html
                {%endfor%}
            </tbody>

          </table>


        {%else%}
        <h3 class="mx-5">No Data</h3>
        {%endif%}


      </div>
  </div>
      {%endblock content%}
  </div>
  <script src='{%static "formApp/js/jquery.js"%}'></script>
  <script src='{%static "formApp/js/popper.js"%}'></script>
  <script src='{%static "formApp/js/bootstrap.js"%}'></script>
  <script src='{%static "formApp/js/all.min.js"%}'></script>
  <script src='{%static "formApp/js/my.js"%}'></script>
  </body>
  </html>
```

update.html ×

formApp > templates > formApp > update.html

```html
<!DOCTYPE html>
{%load static %}
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <link rel='stylesheet' href='{%static "formApp/css/all.min.css"%}'>
  <link rel='stylesheet' href='{%static "formApp/css/bootstrap.css"%}'>
  <link rel='stylesheet' href='{%static "formApp/css/custom.css"%}'>
    <title>Update Data</title>
</head>
<body>
    <div class='container mt-5'>
        <h2 class="text-center alert alert-info">Update Student Data</h2>
    <div class='row mt-4'>
        <div class='col-sm-8 mx-4'>
            <form action="" method='POST'>{% csrf_token %}
                {{newUpd.as_p}}
                <input type="submit" value="Update" class='btn btn-info'>
                <a href="{%url 'base'%}" class='d-inline mx-2 btn btn-outline-info'>Back to Home</a>
            </form>
        </div>
    </div>

    </div>
<script src='{%static "formApp/js/jquery.js"%}'></script>
<script src='{%static "formApp/js/popper.js"%}'></script>
<script src='{%static "formApp/js/bootstrap.js"%}'></script>
<script src='{%static "formApp/js/all.min.js"%}'></script>
<script src='{%static "formApp/js/my.js"%}'></script>
</body>
```

ye line BASE/DIR line k upar stay mei likhne hai

import constants as messages

Messages Framework:-

Syntax:- from django.contrib import messages

messages.debug(request, '%s SQL was done'% count)

messages.info(request, 'the 22 credit remains in ur account')

messages.success , messages.warning , messages.error

~~messages.level(e.g=info(request~~ Another Syntax:-

messages.add_message(request, messages.SUCCESS, 'ur message', extra_tags => fail_silently=False)

Agr kisi tag ka name change krna ho to open settings.py → from django.contrib.messages MESSAGE_TAGS={ messages.ERROR : 'danger', } sabse akhir me likhne hai

Template file Syntax:-

{% messages %}

{% for message in messages %}

{% if message.tags %} {{ message.tags }} {% endif %}

{{ message }}

{% endfor %}

from django.contrib.messages import get_messages

avar = get_messages()

ab is var ko dic mei dal kr ontml file mei loop k zriye access kr sktay hai sarey messages ko

e.g => for avar message in avar do any action

E.g of MsgsFrameWork:-

Views.Py File :- from django.contib import messages

```
def fnName(req):
    if req.method == 'POST':
        fm = stdForm(request.POST)
            if fm.is_valid():
                fm.save()
                messages.info(request, 'ur form is submted')
                Print(messages.get_level(request))
                messages.debug(request, 'debug msg')
                messages.set_level(request, message.DEBUG)
                Print("debug after set_level")
        else:
            fm = stdForm()
        return render(req, ——, {'form': fm})
```

debug ka level 20 se km hota hai isliye iss ka msg by default show nhi hota jb tk ham ye line na likhe

• html File :-

```
<body>
<form action="" method="POST">{{ csrf_token }}
    {{ form.as_p }}
    <input type=submit value="submit">
    </form>

{% if messages %}
    {% for message in messages %}
    <span {% if message.tags %}
    {{ message }} </span>
    {% endfor %}
{% endif %}
</body>
```

agr bootstrap ka class use krna hai to

class="alert alert-{{ message.tags }}"

class="{{ message.tags }} {% endif %}"

agr wha hm ne info msg use kiya hai to iss tran agr success to is ki value success jiss ko is ki value info hogi iluse kr k hm CSS apply kr sktay hai.

username & Password k validate krnay ko authentication, jbk us user ko kon kon sa content ka access diya jai ga uss ko authorization bolte hai.

(28)

UserCreationForm framework/modelle :-

views.py => from django.contrib.auth.forms import UserCreationForm

ab reg.is form k data ko valide wghaira kr k loop k xrye 1 alag Pattern mei display kr sktay hai.

```
def base(req):
    fm = UserCreationForm()
    return render(req, 'base.html', {'form': fm})
```

By default it Provides 3 fields i.e => UserName, Password1, Password2 (Confirmation)

.html File => {.g => <forms>

```
{% for var in form %}
    {{var.label_tag}} {{var}} {{var.errors|striptags}} <br> <br>
{% endfor %}
<input type="submit" value="signup">
</forms>
```

Pic → 34 to 38

Now to add ur own field in this form.

forms.py =>

```
from django.contrib.auth.models import User
"  "  "  "  .forms import UserCreationForm

class myForm(UserCreationForm):
    Password2 = forms.CharField(
    label='Password (again)',
    widget=forms.PasswordInput())
    class Meta:
        model = User
        fields = ['first_name', 'last_name', 'email']
        labels = {'email': 'Email'}
```

views.py =>

```
from django.contrib import messages
from .forms import myForm
fm = myForm()
```

.html File => use for loop as written above

→ Pre-defined fields as show in above Pictures 34 to 38

34 to 38

User object Fields

username - Usernames may contain alphanumeric, _, @, +, . and - characters. Its required and length is 150 characters or fewer.

first_name – Its optional (blank=True) and length is 30 characters or fewer

last_name – Its optional (blank=True) and length is 150 characters or fewer.

email – Its optional (blank=True)

password – A hash of, and metadata about, the password. Django doesn't store the raw password. Its required.

groups - Many-to-many relationship to Group.

user_permissions - Many-to-many relationship to Permission.

is_staff - Designates whether this user can access the admin site. It takes Boolean.

is_active - Designates whether this user account should be considered active. We recommend that you set this flag to False instead of deleting accounts; that way, if your applications have any foreign keys to users, the foreign keys won't break. It takes Boolean.

is_superuser - Designates that this user has all permissions without explicitly assigning them. It takes Boolean.

last_login - A datetime of the user's last login.

date_joined - A datetime designating when the account was created. Is set to the current date/time by default when the account is created.

is_authenticated - This is a way to tell if the user has been authenticated. This does not imply any permissions and doesn't check if the user is active or has a valid session. Its a read-only attribute which is always True.

is_anonymous - This is a way of differentiating User and AnonymousUser objects. It's a read-only attribute which is always False

User object Methods

get_username() - Since the User model can be swapped out, you should use this method instead of referencing the username attribute directly. It returns the username for the user.

get_full_name() – It returns the first_name plus the last_name, with a space in between.

get_short_name() – It returns the first_name.

set_password(raw_password) – It sets the user's password to the given raw string, taking care of the password hashing. Doesn't save the User object.

When the raw_password is None, the password will be set to an unusable password, as if set_unusable_password() were used.

check_password(raw_password) – It returns True if the given raw string is the correct password for the user. (This takes care of the password hashing in making the comparison.)

set_unusable_password() – It marks the user as having no password set. This isn't the same as having a blank string for a password. check_password() for this user will never return True. Doesn't save the User object.

You may need this if authentication for your application takes place against an existing external source such as an LDAP directory.

has_usable_password() – It returns False if set_unusable_password() has been called for this user.

get_user_permissions(obj=None) – It returns a set of permission strings that the user has directly.

If obj is passed in, only returns the user permissions for this specific object.

get_group_permissions(obj=None) – It returns a set of permission strings that the user has, through their groups.

get_all_permissions(obj=None) – It returns a set of permission strings that the user has, both through group and user permissions.

If obj is passed in, only returns the permissions for this specific object.

has_perm(perm, obj=None) – It returns True if the user has the specified permission, where perm is in the format "<app label>.<permission codename>". If the user is inactive, this method will always return False. For an active superuser, this method will always return True.

If obj is passed in, this method won't check for a permission for the model, but for this specific object.

has_perms(perm_list, obj=None) – It returns True if the user has each of the specified permissions, where each perm is in the format "<app label>.<permission codename>". If the user is inactive, this method will always return False. For an active superuser, this method will always return True.

has_module_perms(package_name) – It returns True if the user has any permissions in the given package (the Django app label). If the user is inactive, this method will always return False. For an active superuser, this method will always return True.

email_user(subject, message, from_email=None, **kwargs) – It sends an email to the user. If from_email is None, Django uses the DEFAULT_FROM_EMAIL. Any **kwargs are passed to the underlying send_mail() call.

UserManager Methods

The User model has a custom manager that has the following helper methods (in addition to the methods provided by BaseUserManager):

create_user(username, email=None, password=None, **extra_fields) – It creates, saves and returns an User.

The username and password are set as given. The domain portion of email is automatically converted to lowercase, and the returned User object will have is_active set to True.

If no password is provided, set_unusable_password() will be called.

The extra_fields keyword arguments are passed through to the User's __init__ method to allow setting arbitrary fields on a custom user model.

create_superuser(username, email=None, password=None, **extra_fields) – It is same as create_user(), but sets is_staff and is_superuser to True.

Group Object Fields

name - Any characters are permitted. It is required and length is 150 characters or fewer. Example: 'Awesome Users'.

permissions - Many-to-many field to Permission.

Permission Object Fields

name – It is required and length is 255 characters or fewer. Example: 'Can vote'.

content_type - A reference to the django_content_type database table, which contains a record for each installed model. It is required.

codename – It is required and length is 100 characters or fewer. Example: 'can_vote'.

29)

Login & signUP Form:- Pic → 39 to 48 i.e. Project No: 16 must here.

Loop to capture errors:- .html File ⇒ to show and edit user form in Profile .html See/Past Pic 49 Project. code in this Proj

<form>

{% if form.non_field_errors %}

{% for any var error in form.non_field_errors %}

<P> {{ error }} </P>

{% endfor %}

{% endif %}

Some Points from above Pictures:- views.py ⇒ file object

1) forms.py from django import UserCreationForm → var = SignUP(request.POST)

2) views.py = import AuthenticationForm → object syntax ⇒ var = AuthenticationForm = (request = request, data = request.POST)

3) forms.py ⇒ import User ChangeForm views.py ⇒ var = Edit(request.POST, instance = request.user)

4) views ⇒ import PasswordChangeForm obj syntax ⇒ var = PasswordChangeForm(user = request.user, data = request.POST)

5) import SetPasswordForm obj syntax = SetPasswordForm(" ")

1 = signUPForm 2 = loginForm 3 = Show & edit User Full Form

4 = Change User Passwrd (entr old Pass &) 5 = chang Password without entering old Password.

39 to 48:

urls.py - regFormUsingUserCreationForm - Visual Studio Code

regFormUsingUserCreationForm > urls.py > ...

```python
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from aLoginOrSignUp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.signUp,name='signUp'),
    path('login/',views.loginUp,name='login'),
    path('profile/',views.profile,name='profile'),
    path('userLogout/',views.UserLogout,name='userLogout'),
    path('changePass/',views.changePass,name='changePass'),
    path('chngPassWEOldPass/',views.chngPassWEOldPass,name='chngPassWEOldPass')

]
```

aLoginOrSignUp > forms.py > ...

```python
from cProfile import label
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
from django import forms

class signUpForm(UserCreationForm):
    password2=forms.CharField(          #adding our own field. By default it shows only username and password.
        label='Password (again)'
        ,widget=forms.PasswordInput()
    )
    class Meta:
        model=User
        fields=['username','first_name','last_name','email'] #these fields spellings are pre-defined.
        labels={'email':'Email'}
```

aLoginOrSignUp > views.py > ...

```python
from django.shortcuts import render
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth.forms import PasswordChangeForm,SetPasswordForm
from .forms import signUpForm
from django.contrib import messages
from django.contrib.auth import authenticate,login,logout
from django.contrib.auth import update_session_auth_hash  #to redirect to a page after changing password. bcaus
#httpresponseredirect directly doesn't work here directly.

from django.shortcuts import  HttpResponseRedirect

# Create your views here.
#Singn Up Form :
def signUp(request):
    if not request.user.is_authenticated:
        if request.method=='POST':
            fm=signUpForm(request.POST)
            if fm.is_valid():
                fm.save()
                fm=signUpForm()
                messages.info(request,'Thanks!, Ur form has been submitted. ')
        else:
            fm=signUpForm()
        return render(request,'signUp.html',{'form':fm})
    else:
        return HttpResponseRedirect('/profile/')
```

```python
#Login Form :
def loginUp(request):
    if request.method=='POST':
        lgfm=AuthenticationForm(request=request,data=request.POST)
        if lgfm.is_valid():
            uName=lgfm.cleaned_data['username']
            uPassword=lgfm.cleaned_data['password']
            user=authenticate(username=uName,password=uPassword)
            if user is not None:
                login(request,user)
                return HttpResponseRedirect('/profile/')
    else:
        lgfm=AuthenticationForm()
    return render(request,'login.html',{'lgfm':lgfm})


#Showing Dashbord or HomePage after Logging
def profile(request):
    if request.user.is_authenticated:
        return render(request,'profile.html',{'userName':request.user})
    else:
        return HttpResponseRedirect('/login/')


# f(x) for Logout button
def UserLogout(request):
    logout(request)
    return HttpResponseRedirect('/')


#Changing User password, Old password needed to create new Password :
def changePass(request):
    if request.user.is_authenticated:
        if request.method=='POST':
            fm=PasswordChangeForm(user=request.user,data=request.POST)
            if fm.is_valid():
                fm.save()
                update_session_auth_hash(request, fm.user)
                messages.success(request,'Password Changed Successfully !')
                return HttpResponseRedirect('/profile/')
        else:
            fm=PasswordChangeForm(user=request.user)
        return render(request,'changePass.html',{'form':fm} )
    else:
        return HttpResponseRedirect('/login/')


#Change Password Without entering old Password :
def chngPassWEOldPass(request):
    if request.user.is_authenticated:
        if request.method=='POST':
            fm=SetPasswordForm(user=request.user,data=request.POST)
            if fm.is_valid():
                fm.save()
                update_session_auth_hash(request, fm.user)
                messages.success(request,'Password Changed Successfully !')
                return HttpResponseRedirect('/profile/')
        else:
            fm=SetPasswordForm(user=request.user)
        return render(request,'chngPassWEOldPass.html',{'form':fm} )
    else:
        return HttpResponseRedirect('/chngPassWEOldPass/')
```

signUp.html

aLoginOrSignUp > templates > signUp.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SignUp</title>
    <style>
        .info{
            background-color: lightblue
        }
    </style>
</head>
<body>
    <form action="" method="POST">{% csrf_token %}
        {%for display in form%}
            {{display.label_tag}} {{display}} {{display.errors|striptags}}<br><br>
        {%endfor%}
        <input type="submit" value="SignUp">
        {%if messages%}
        {%for msg in messages%}
       <p {%if msg.tags%} class='{{msg.tags}}' {%endif%}> {{msg}} </p>
        {%endfor%}
        {%endif%}
    </form><br>
    <a href="{%url 'login'%}">Login here</a>
</body>
</html>
```

login.html

aLoginOrSignUp > templates > login.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
</head>
<body>
    <form action="" method='POST' novalidate>{% csrf_token %}
        {%for display in lgfm%}
        {{display.label_tag}} {{display.errors|striptags}}{{display}}<br><br>
        {%endfor%}

        {%if lgfm.non_field_errors%}
        {%for error in lgfm.non_field_errors%}
         <p>   {{error}}   </p><br>
        {%endfor%}
    {%endif%}



        <input type="submit" value="Login">
    </form><br>
    <a href="{%url 'signUp'%}">Back to SignUp</a>


</body>
</html>
```

profile.html

aLoginOrSignUp > templates > profile.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Welcome {{userName}} !</h1>



    <br><br><br>
    <a href="{%url 'userLogout'%}">Logout</a><br>
    <a href="{%url 'changePass'%}">Change Password</a>  <br>
    <a href="{%url 'chngPassWEOldPass'%}">Password Without Entering Old Password</a>
</body>
</html>
```

changePass.html

aLoginOrSignUp > templates > changePass.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Change Password</title>
</head>
<body>
    <form action="" method='POST' novalidate>{% csrf_token %}

        {%if form.non_field_erros%}
            {%for error in form.non_field_errors%}
                {{error}}
            {%endfor%}
        {%endif%}

        {%for fm in form%}
            {{fm.label_tag}} {{fm}} {{fm.errors|striptags}} <br><br>
        {%endfor%}

        <input type="submit" value="Submit">
        <a href="{%url 'userLogout'%}">Logout</a>

        {%if messages%}
        {%for msg in messages%}
            <p {%if msg.tags%} class='{{msg.tags}}' {%endif%}>{{msg}}</p>
        {%endfor%}
    {%endif%}
</body>
    </form>
</html>
```

chngPassWEOldPass.html

aLoginOrSignUp > templates > chngPassWEOldPass.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Change Password</title>
</head>
<body>
    <form action="" method='POST' novalidate>{% csrf_token %}

        {%if form.non_field_erros%}
            {%for error in form.non_field_errors%}
                {{error}}
            {%endfor%}
        {%endif%}

        {%for fm in form%}
            {{fm.label_tag}} {{fm}} {{fm.errors|striptags}} <br><br>
        {%endfor%}

        <input type="submit" value="Submit">
        <a href="{%url 'userLogout'%}">Logout</a>

        {%if messages%}
        {%for msg in messages%}
            <p {%if msg.tags%} class='{{msg.tags}}' {%endif%}>{{msg}}</p>
        {%endfor%}
    {%endif%}
    </form>
</body>
</html>
```

49:

*dj - Notepad

File Edit Format View Help

```
forms.py
from django.contrib.auth.forms import UserChangeForm

class editUserForm(UserChangeForm):
    password=None
    class Meta:
        model=User
        fields=['username','first_name','last_name','email','date_joined','last_login']



views.py
from .forms import editUserForm
#Showing Dashbord or HomePage after Logging
def profile(request):
    if request.user.is_authenticated:
        if request.method=='POST':
            fm=editUserForm(request.POST,instance=request.user)
            if fm.is_valid():
                messages.success(request,'Form Changed Successfully !')
                fm.save()
        else:
            fm=editUserForm(instance=request.user)
        return render(request,'profile.html',{'userName':request.user, 'form':fm})
    else:
        return HttpResponseRedirect('/login/')
```

20)

To Show & edit Admin Profile + show all users in admin home Profile + edit users data from admin Profile ⇒ (Project 17)

Permissions & Authorization in Django:- (Project 18)

make a simple login/signup app then add/see code of Pic 50 to 52. Pictures are from Project 18.

Cookies in Django:-

get.cookie.html file:-
{{name}}

E.g:- views.py:-

```
def set(request):
    aVar = render(request, 'htmlfilename')
    aVar.set_cookie('name', 'khan')   ——— line (3)
    return aVar
```

Key Value

```
def get(request):
    aVar = request.COOKIES['name']
```

or `aVar = request.COOKIES.get('name', 'a default text to display')`

when cookies are not set. Agr default value nhi denge to None ayega error nhi ayega.

```
    return render(request, 'htmlfilename', {'name': aVar})
def del(request):
    aVar = render(request, 'htmlfilname')
    aVar.delete_cookie('name')
    return aVar
```

By default browser k close kr may say cookie delete ho jata hai. Agr apna time duration dena hai to:- line 3 ⇒

Pic 50 to 52:

app > models.py > blog

```python
from django.forms import CharField

# Create your models here.
class blog(models.Model):
    title=CharField(max_length=50)
    description=CharField(max_length=50)
```

app > admin.py > blogAdmin

```python
from django.contrib import admin
from .models import blog
# Register your models here.
@admin.register(blog)
class blogAdmin(admin.ModelAdmin):
    list_display=['title','description']
```

```python
#from django.contrib.auth.models import Group
def signUp(request):
    if request.method=='POST':
        suf=signUpForm(request.POST)
        if suf.is_valid():
            user=suf.save()
            group=Group.objects.get(name='Editor')  #assign Editor group which we have created in admin pannel
            user.groups.add(group)                  #directly to the user by signingUp.
            messages.success(request,'Id Created Successfully !')
            return HttpResponseRedirect('/login/')
    else:
        suf=signUpForm()
    return render(request,'signUp.html',{'form':suf})
```

app > templates > home.html

```
{% extends 'base.html' %}
{% load static %}
{% block body %}
<h2>Welcom Home {{name}}</h2>
{% comment %} <input type="button" value="Create"><br><br>
<input type="button" value="Read"><br><br>
<input type="button" value="Update"><br><br> {% endcomment %}


                                    <!--whenever we create a model in models.py, these 4 permissions r created-->
                                    <!--automatically in http://127.0.0.1:8000/admin/auth/user/ which u  -->
                                    <!--can then assign to any User-->
    <!--Syntax = {%if perms.appName.action_modelName%}-->
                                        <!--action can be = delete,view,change,add-->

<form action="" method="post">{% csrf_token %}
{%if perms.app.delete_blog%}
    <input type="button" value="Delete"><br><br>
{%else%}
    <p></p>
{%endif%}


{%if perms.app.add_blog%}
    <input type="button" value="add"><br><br>
{%else%}
    <p></p>
{%endif%}
</form>
<a href="{%url 'ulogout'%}">Logout</a>
{% endblock body %}
```

Permissions and Authorization

myuser.groups.set([group_list])

myuser.groups.add(group, group, ...)

myuser.groups.remove(group, group, ...)

myuser.groups.clear()

myuser.user_permissions.set([permission_list])

myuser.user_permissions.add(permission, permission, ...)

myuser.user_permissions.remove(permission, permission, ...)

myuser.user_permissions.clear()

31)
Aam browsers 4096 bytes tk cookies ka data store krta hai
agr iss say zyada hua to store nhi hoga.

---

1) aVar.set_cookie('name','khan', max_age = (60*60))
nmbr of seconds

2) ~~from django import~~. from datetime import datetime, timedelta

line 3 => aVar.set_cookie('name','khan', expires = datatime.utcnow()+
timedelta(days = 2))
any nmbr of days u want to give

Signed_Cookie:-

def set(request):
aVar = render (request, 'intro')
aVar.set_signed_cookie('name', 'key', salt='nm', expires ...)
value
return aVar

def get(request):
aVar = request.get_signed_cookie('name', salt='nm', default='')
mtlb agr salt=ki value set ~~hai~~ k salt se match
nahi ho rahi to default mei jo text hai wo display krdo.

It's Preffered to use Session Framework than using

Pic 53 to 56 here.

Session Framework:- is fmwork ko use krny se user k
machine mei ek special key store krta aur cookies ko database mei
store krta hai same special key se. Jab user get request krta
hai to special key match hoti hai db/server k cookies k spec
key se aur cookies load ho jati hai.

Pic 53 to 56

Creating Cookies

set_cookie() – set_cookie() is used to set/create/sent cookies.

Syntax: - HttpResponse.set_cookie(key, value='', max_age=None, expires=None, path='/', domain=None, secure=False, httponly=False, samesite=None)

Example: - set_cookie(“name”, “sonam”)

key − This is the name of the cookie.

value − This sets the value of cookie. This value is stored on the clients computer.

name and value are required to set cookie.

When you omit the optional cookie fields, the browser fills them in automatically with reasonable defaults.

max_age - It should be a number of seconds, or None (default) if the cookie should last only as long as the client’s browser session. If expires is not specified, it will be calculated.

Example:- set_cookie(“name”, “sonam”, max_age=60*60*24*10) // 10 days

expires - It describes the time when cookie will be expire. It should either be a string in the format "Wdy, DD-Mon-YY HH:MM:SS GMT" or a datetime.datetime object in UTC. If expires is a datetime object, the max_age will be calculated.

Example:- set_cookie(“name”, “sonam”, expires= datetime.utcnow() + timedelta(days=2))

path - Path can be / (root) or /mydir (directory).

Example: - set_cookie(“name”, “sonam”, “/ ”)

Example: - set_cookie(“name”, “sonam”, “/home”)

domain - Use domain if you want to set a cross-domain cookie. For example, domain="example.com" will set a cookie that is readable by the domains www.example.com, blog.example.com, etc. Otherwise, a cookie will only be readable by the domain that set it.

Example: - set_cookie (“name”, “sonam”, domain=“geekyshows.com”)

note.geekyshows.com

code.geekyshows.com

Example: - set_cookie (“name”, “sonam”, “code.geekyshows.com”)

secure - Cookie to only be transmitted over secure protocol as https. When set to TRUE , the cookie will only be set if a secure connection exists.

Example: - set_cookie(“name”, “sonam”, max_age=60*60*24*10, path=“/ ”, domain=“geekyshows.com”, secure=True)

Reading/Accessing Cookie

HttpRequest.COOKIES - A dictionary containing all cookies. Keys and values are strings.

Syntax:- request.COOKIES[‘key’];

Example: - request.COOKIES[‘name’];

Syntax:- request.COOKIES.get(‘key’, default)

Example: - request.COOKIES.get(‘name’)

Example: - request.COOKIES.get(‘name’, “Guest”)

Deleting Cookies

HttpResponse.delete_cookie(key, path='/', domain=None) – This method is used to delete the cookie based on given key with same domain and path, if they were set, otherwise the cookie may not be deleted.

Example: - delete_cookie(‘name’)

```
#session cookie ka by default age 2 weeks ka hota hai
def setSession(request):
    request.session['name']='Khan'
    request.session['name2']='GullvKhan'
    return render(request,'setSession.html')

def getSession(request):
    # nm=request.session['name'] or below line:
    # nm=request.session.get('name','no session cookie')    or below line:      all r same
    nm=request.session.get('name',default='no session cookie')
    nam=request.session.get('name2',default='no session cookie')

                        #ye f(x)=setdefault() set our get dono kaam krta hai
    age=request.session.setdefault('age','19')#agr hai to ok mgr agr nhi hai to 19 return kray ga
    keys=request.session.keys()
    items=request.session.items()
    return render(request,'getSession.html',{'name':nm,
        'name2':nam,'keys':keys,'items':items,'age':age})

def delSession(request):
    request.session.flush()
    # if 'name' in request.session:   #if cndtion is liye lgaya ta k error se bch jae
    #     del request.session['name']
    return render(request,'deleteSession.html')
```

32)

E.g:- views.py ⇒

```
def setSession(request):
    request.session['key(eg: name)'] = 'khan'   # value
    return render(reque, 'set.html')

def getSession(request):
    nm = request.session.get('name', 'defaultValue')
    return render(request, 'get.html', {'name': nm})

def delSession(request):
    if 'name' in request.session:
        del request.session['name']
    return render(request, 'del.html')
```

if condition iss liye lgate hai ta k error se bach jaye.

→ iss line se data delete hotata hai mgr session our special key Phir be bagi rehta hai

agr Poray session ko khtm krna hai to:

```
def delSession(request):
    request.session.flush()
    return render(requ---)
```

→ recommended to use

Pic 56 here. Pic 57 + 58 here. Pic 59 here, Pic 60 here Pic 61 here, Pic 62 here, Pic 63 here,

Cache:- jb ham apne site/site k aik hissay ko cache me delete mei save krta hai jiss se us ko dobara access krnay mei time nhi lagta.

Storage can be:-

1) database :- Pic 64 a, b, c

Pic 65, b, c, d

2) Filesystem :- 66 no Pic

3) Local Memory :- Pic 67

The code in these Pics is for creating cache for whole web. jis ko Per site cache Implementation "delete" hai

Pic 56

```python
#session cookie ka by default age 2 weeks ka hota hai
def setSession(request):
    request.session['name']='Khan'
    request.session['name2']='GullvKhan'
    return render(request,'setSession.html')

def getSession(request):
    # nm=request.session['name'] or below line:
    # nm=request.session.get('name','no session cookie')    or below line:      all r same
    nm=request.session.get('name',default='no session cookie')
    nam=request.session.get('name2',default='no session cookie')

                                #ye f(x)=setdefault() set our get dono kaam krta hai
    age=request.session.setdefault('age','19')#agr hai to ok mgr agr nhi hai to 19 return kray ga
    keys=request.session.keys()
    items=request.session.items()
    return render(request,'getSession.html',{'name':nm,
        'name2':nam,'keys':keys,'items':items,'age':age})

def delSession(request):
    request.session.flush()
    # if 'name' in request.session:   #if cndtion is liye lgaya ta k error se bch jae
    #     del request.session['name']
    return render(request,'deleteSession.html')
```

Pic 57 and 58

```python
from django.shortcuts import render

# Create your views here.
def setSession(request):
    request.session['name']='Khan'
# set_expiry(nmbr of seconds) dete hai.Agr yha value me 0 denge to phir browser k close hone se session expire ho jae ga.
# is method se session expire(chrome://settings/siteData is path se delete hota hia) hota hai mgr data phir b db
# me rehta hai. agr db se delete krna hai to line 20 ka code likhengei.
    request.session.set_expiry(500)
    return render(request,'setSession.html')

def getSession(request):
    nm=request.session.get('name',default='no session cookie')


    age=request.session.setdefault('age','19')
    keys=request.session.keys()
    items=request.session.items()
    return render(request,'getSession.html',{'name':nm})

def delSession(request):
    request.session.flush()
    request.session.clear_expired()
    return render(request,'deleteSession.html')
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body><h2>
{{name}}    <br>
{{request.session.get_session_cookie_age}}  <br>
{{request.session.get_expiry_age}} <br>
{{request.session.get_expiry_date}} <br>
{{request.session.get_expire_at_browser_close}} <br>
</h2>
</body>
</html>
```

Pic 59

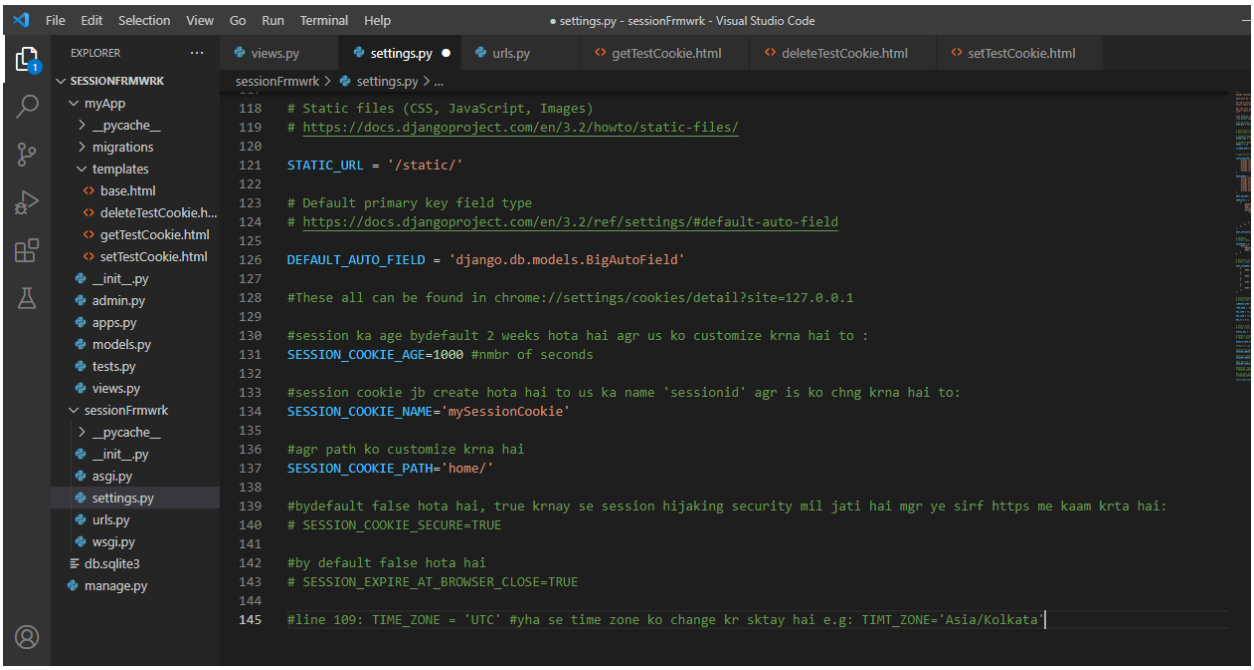
```python
from django.shortcuts import render

def setTestCookie(request):
    print(request.session.set_test_cookie())
    return render(request,'setTestCookie.html')


def checkTestCookie(request):
    request.session.test_cookie_worked()
    return render(request,'getTestCookie.html')


def delTestCookie(request):
    request.session.delete_test_cookie()
    return render(request,'deleteTestCookie.html')
```

Pic 60



Pic 61

```python
1  from django.http import HttpResponse
2  from django.shortcuts import render
3
4  def setSession(request):
5      request.session['name']='Ali'
6      return render(request,'setSession.html')
7
8
9
10 def getSession(request):
11     if 'name' in request.session:
12         nm=request.session['name']
13         request.session.modified=True   #mtlb are given time me session refresh ho to time restart hoga.
14         return render(request,'getSession.html',{'name':nm})
15     else:
16         return HttpResponse('<h3>Session Expired...<h3>')
17
18 def delSession(request):
19     request.session.flush()
20     request.session.clear_expired()
21     return render(request,'delSession.html')
22
23 #SETTINGS.PY FILE= SESSION_COOKIE_AGE=15
24 #15 seconds tk data get ho skta hai us k bad nhi. our line:13
```

Pic 62

File Edit Selection View Go Run Terminal Help • views.py - sessionFrmwrk - Visual Studio Code

EXPLORER
- SESSIONFRMWRK
  - myApp
    - __pycache__
    - migrations
    - templates
    - __init__.py
    - admin.py
    - apps.py
    - models.py
    - tests.py
    - views.py
  - sessionFiles
    - sessionidfq67z996c8...
  - sessionFrmwrk
    - __pycache__
    - __init__.py
    - asgi.py
    - settings.py
    - urls.py
    - wsgi.py
  - db.sqlite3
  - manage.py

views.py | settings.py | urls.py | getSession.html | delSession.html | setSession.html

myApp > views.py > ...

```python
1  from django.http import HttpResponse
2  from django.shortcuts import render
3
4
5  #file based session view
6
7  def setSession(request):
8      request.session['name']='Ali'
9      return render(request,'setSession.html')
10
11
12
13 def getSession(request):
14         nm=request.session['name']
15         return render(request,'getSession.html',{'name':nm})
16
17 def delSession(request):
18     request.session.flush()
19     request.session.clear_expired()
20     return render(request,'delSession.html')
21
22 #Settings.py file =SESSION_ENGINE='django.contrib.sessions.backends.file'
23                                 #SESSION_FILE_PATH=BASE_DIR/'sessionFiles'
```

Pic 63

myApp > views.py > ...

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

def home(request):
    c=request.session.get('count',0)
    newC=c+1
    request.session['count']=newC
    return render(request,'home.html',{'c':newC})

#.html file:
#<meta http-equiv='refresh' content='1'> This line will auto automatically refresh
# the I page after 1 second.
#<title>Document</title>
#</head>
#<body>
#<h2>{{c}}</h2> it's value will be increment 1 by every refresh.
#</body>        jb session expire/destroy hoga to is ki value dobara 1 hojaegi.
#</html>

#Note: jb hm djagno authentication system use krtay hai to logout krnay say hmara
# session expire hota hai.
```

Pic 64a,b,c..:

File Edit Selection View Go Run Terminal Help views.py - 22 - Visual Studio Code

EXPLORER

urls.py settings.py home.html views.py

perSiteChacheImplementation > myApp > views.py > home

```python
from django.shortcuts import render

# Create your views here.
def home(request):
    return render(request,'home.html')
```

urls.py settings.py home.html views.py

perSiteChacheImplementation > perSiteChacheImplementation > settings.py > ...

```python
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',

#cache implementation. sequence is compulsory.
    'django.middleware.cache.UpdateCacheMiddleware',
    'django.middleware.common.CommonMiddleware', #ye line pehle sai hota hai.
    'django.middleware.cache.FetchFromCacheMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

urls.py | settings.py ● | home.html | views.py

perSiteChacheImplementation > perSiteChacheImplementation > settings.py > ...

```python
119 USE_TZ = True
120
121
122 # Static files (CSS, JavaScript, Images)
123 # https://docs.djangoproject.com/en/3.2/howto/static-files/
124
125 STATIC_URL = '/static/'
126
127 # Default primary key field type
128 # https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field
129
130 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
131
132
133 #expire cache time
134 CACHE_MIDDLEWARE_SECONDS=30
135
136 #cache setup in db
137 CACHES={
138     'default':{
139         'BACKEND':'django.core.cache.backends.db.DatabaseCache',
140         'Location':'myCacheTable'   #to create this table run : python manage.py createcachetable
141     }                               #command in terminal.
142 }
143
144 #ab jb surver ko run krayengai to home.html ka data show hoga.30 seconds(line:134) k andr agr hm
145 # home.html k data mei changes kray to wo show nhi hoga q k cache mei purana wala data store hai
146 # 30 seconds k bad jb cache expire ho jae ga to nya wala updated data show hoga our cache mei
147 # store ho jae ga.
148
149 #Line 113: TIME_ZONE = 'UTC'   #agr is ko change krnay se time zone change nhi hota to line:119 ko False krna hoga.
```

Pic 65,b,c..:

The per-site cache

The per-site cache – Once the cache is set up, the simplest way to use caching is to cache your entire site.

```
MIDDLEWARE = [
   'django.middleware.cache.UpdateCacheMiddleware',
   'django.middleware.common.CommonMiddleware',
   'django.middleware.cache.FetchFromCacheMiddleware',
]
```

CACHE_MIDDLEWARE_ALIAS – The cache alias to use for storage.

CACHE_MIDDLEWARE_SECONDS – The number of seconds each page should be cached.

CACHE_MIDDLEWARE_KEY_PREFIX – If the cache is shared across multiple sites using the same Django installation, set this to the name of the site, or some other string that is unique to this Django instance, to prevent key collisions. Use an empty string if you don't care.

Database Caching

Django can store its cached data in your database. This works best if you've got a fast, well-indexed database server.

```
CACHES = {
   'default': {
      'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
      'LOCATION': 'my_cache_table',
   }
}
```

The name of the database table. This name can be whatever you want, as long as it's a valid table name that's not already being used in your database.

Before using the database cache, you must create the cache table with this command:

python manage.py createcachetable

This creates a table in your database that is in the proper format that Django's database-cache system expects. The name of the table is taken from LOCATION.

If you are using multiple database caches, createcachetable creates one table for each cache.

Cache Arguments

Each cache backend can be given additional arguments to control caching behavior.

TIMEOUT: The default timeout, in seconds, to use for the cache. This argument defaults to 300 seconds (5 minutes). You can set TIMEOUT to None so that, by default, cache keys never expire. A value of 0 causes keys to immediately expire (effectively "don't cache").

OPTIONS: Any options that should be passed to the cache backend. The list of valid options will vary with each backend, and cache backends backed by a third-party library will pass their options directly to the underlying cache library.

Cache backends that implement their own culling strategy (i.e., the locmem, filesystem and database backends) will honor the following options:

MAX_ENTRIES: The maximum number of entries allowed in the cache before old values are deleted. This argument defaults to 300.

CULL_FREQUENCY: The fraction of entries that are culled when MAX_ENTRIES is reached. The actual ratio is 1 / CULL_FREQUENCY, so set CULL_FREQUENCY to 2 to cull half the entries when MAX_ENTRIES is reached. This argument should be an integer and defaults to 3.

A value of 0 for CULL_FREQUENCY means that the entire cache will be dumped when MAX_ENTRIES is reached. On some backends (database in particular) this makes culling much faster at the expense of more cache misses.

Cache Arguments

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'enroll_cache',
        'TIMEOUT': 60,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    }
}
```

Pic 66:

```python
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
CACHE_MIDDLEWARE_SECONDS = 30
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': 'C:\DjangoCode\gs79\cache',
    }
}
```

Pic 67:

Local Memory Caching

This is the default cache if another is not specified in your settings file. This cache is per-process and thread-safe.

Each process will have its own private cache instance, which means no cross-process caching is possible. This obviously also means the local memory cache isn't particularly memory-efficient.

It's probably not a good choice for production environments. It's nice for development.

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake',
    }
}
```

The cache LOCATION is used to identify individual memory stores.

Cache is preferred to be used for dynamic pages/web. Local memory cache production me use nhi kiya jata.

Per View Cache:

Mtlb agr kisi 1 view/f(x) ko cache mei dalna ho.

It has two ways to be used/done:

1.

urls.py | views.py | home.html | settings.py

perViewCaching > myApp > views.py > ...

```python
from django.shortcuts import render



#ab ye cache sirf home function k liye hoga, na k pooray web k liye.
from django.views.decorators.cache import cache_page
@cache_page(20) # (nmbr of seconds) in which cache will be expired.
def home(request):
    return render(request,'home.html')


#Settings.py file:
#iss se hmara data db mei store hoga.
# CACHES={
#     'default':{
#         'BACKEND':'django.core.cache.backends.db.DatabaseCache',
#         'LOCATION':'myCacheTable'
#     }
# }
#run python manage.py createcachetable command in terminal
```

The per-view cache

The per-view cache - A more granular way to use the caching framework is by caching the output of individual views. django.views.decorators.cache defines a cache_page decorator that will automatically cache the view's response. If multiple URLs point at the same view, each URL will be cached separately.

from django.views.decorators.cache import cache_page

@cache_page(timeout, cache, key_prefix)

def my_view(request):

@cache_page(60, cache="some_cache", key_prefix="some_key")

def home(request):

timeout - The cache timeout, in seconds.

cache – This directs the decorator to use a specific cache (from your CACHES setting) when caching view results. By default, the default cache will be used.

Key_prefix - You can also override the cache prefix on a per-view basis. It works in the same way as the CACHE_MIDDLEWARE_KEY_PREFIX setting for the middleware.

2. Note: This method is recommended and more useful than the above method.

urls.py ● views.py home.html settings.py

perViewCaching > perViewCaching > urls.py > ...

```python
#kbhi kbar aise xrorat hoti hai k function aik hi hota hai lekin kbhi us ko cache me daalnei
#ki xrorat hoti hai our kbhi nahi to uss k liye ye mothod hai:

#ab idhr f(x) aik hai lekin aik url pe cache run hoga our dosre pe nhi.
from django.views.decorators.cache import cache_page
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',cache_page(20)(views.home),name='home'),
    path('home/',views.home)
]

#Views.py file:
# def home(request):
#     return render(request,'home.html')


#Settings.py file:
#iss se hmara data db mei store hoga.
# CACHES={
#     'default':{
#         'BACKEND':'django.core.cache.backends.db.DatabaseCache',
#         'LOCATION':'myCacheTable'
#     }
# }
#run python manage.py createcachetable command in terminal
#This method is recommended more than the first method.
```

The per-view cache

Specifying per-view cache in the URLconf

```
from django.views.decorators.cache import cache_page
urlpatterns = [
    path('route/', cache_page(timeout, cache, key_prefix)(view_function)),
]

urlpatterns = [
    path('home/', cache_page(60)(views.home), name="home"),
]
```

Template Fragment Cache:

Mtlb inside template files cache ka use krna.

Template Fragment Caching

This gives you more control what to cache.

{% load cache %} – This gives access to cache tag in template.

{% cache %} - This template tag caches the contents of the block for a given amount of time.

Syntax:-

{% cache timeout name variable using=" " %} ….. {% endcache name %}

Timeout - The cache timeout, in seconds. The fragment is cached forever if timeout is None. It can be a template variable, as long as the template variable resolves to an integer value.

Name - The name to give the cache fragment. The name will be taken as is, do not use a variable.

Variable – This can be a variable with or without filters. This will cache multiple copies of a fragment depending on some dynamic data that appears inside the fragment.

using - The cache tag will try to use the given cache. If no such cache exists, it will fall back to using the default cache. You may select an alternate cache backend to use with the using keyword argument, which must be the last argument to the tag.

Example:-

```
{% load cache %}
{% cache 60 sidebar request.user.username using="localcache" %}
   .. sidebar for logged in user ..
{% endcache %}
```

```
perViewCaching > myApp > views.py > ...
from django.shortcuts import render

def home(request):
    return render(request,'home.html')


#.html file:
# {%load cache%}
# <html>
# <body><h3>
#     {% cache 20 inC %}    abb iss block mei jo data hai sirf wo data cache mei store hoga.
#         inside cache
#     {%endcache inC%}
#     <br>
# outsideCache
# </h3>
# </body>
# </html>

#Settings.py file:
#iss se hmara data db mei store hoga. iss k ilawa filesystem ya local memory mei bhi store kr
#sktay hai.
# CACHES={
#     'default':{
#         'BACKEND':'django.core.cache.backends.db.DatabaseCache',
#         'LOCATION':'myCacheTable'
#     }
# }
#run python manage.py createcachetable command in terminal
#Urls.py file:  path('',views.home)
```

Low level Cache API:

Jb hm kisi 1 object(e.g: a string or a dictionary) ko cache me store krna chahtai hai to iss ka use krtay hai.

views.py × settings.py home.html

myApp > views.py

```python
from django.shortcuts import render
from django.core.cache import cache


# Create your views here.
#Method 1:
# def home(request):
#     mv=cache.get('movie','expired')
#     if mv=='expired':
#         cache.set('movie','kabirSinglkkh not',20)
#         mv=cache.get('movie')
#     return render(request,'home.html',{'movie':mv})

#Method 2:  this and method 1 do the same task.
#version ko change kr k same key k sath multiple caches hm create kr sktay hai
#agr version change nhi krayngai to key update ho jaegi db mei.
# def home(request):
#     mv=cache.get_or_set('fee',20900,20,version=1) #agr pehle se 'fee' key set hai to us ko get kray ga.
#     return render(request,'home.html',{'movie':mv})#nhi to set kr dai ga.

# Settings.py File:
# CACHES={
#     'default':{
#         'BACKEND':'django.core.cache.backends.db.DatabaseCache',
#         'LOCATION':'myCacheTable'
#     }
# }


#Get data of a dictionary:
# def home(request):
#     data={'name':'Professor','rollNmbr':19861}
#     cache.set_many(data,20)
#     stu=cache.get_many(data)
#     print(stu)
#     return render(request,'home.html',{'stu':stu})
#ab agr hm is f(x) ko run krnay k bad line:23 ko rmove kr dengai our 20 seconds k andr
#dic mei changes krayngai to wo changes show nhi hoga q k ab hmara data cache se aarha hai.
#.html File:
# <body>
#     HOME
#     <h2>{{stu.name}}</h2> <br>
#     <h2>{{stu.rollNmbr}}</h2>
# </body>


#jb cache expire hota hai to us k bad bhi wo db mei rehta hai.
#To Delete a Cache:
# def home(request):#    cache.delete('name of key',version=) agr version nhi deingai to bydefault version=1 wala delete ho
#     cache.delete('data',version=1)
#     return render(request,'home.html')

#delete all Caches at once:
#it's recommended to clear cache through terminal:
# Run below commands in terminal:
# python manage.py shell
#from django.core.cache import cache
#cache.clear()
#quit()

#But if u want to user a right to delete the caches then:
# def home(request):
#     cache.clear()
#     return render(request,'home.html')
```

```

#pehle iss trah set krna hai us k bad home2 k f(x) se ya issi f(x) k code ko change kr k
# incrementation ya decrementation kr sktay hai.
# def home(request):
#     cache.set('students',100, 180)
#     pr=cache.get('student')
#     print(pr)
#     return render(request,'home.html')

# def home2(request):
#     pr=cache.incr('students',delta=2)# ab ye hr refresh pe iss ki value me(jo yha 100 hai) 2 increment kray ga
#     print(pr)                       #agr decrement krna hai to incr ki jgha decr likhna hai.
#     return render(request,'home.html')

```

Low-Level Cache API

from django.core.cache import cache

cache.set(key, value, timeout=DEFAULT_TIMEOUT, version=None) – This method is used to set cache.

Where,

key –It should be str.

value – It can be any pickleable Python object.

timeout – It is number of seconds the value should be stored in the cache. Timeout of None will cache the value forever. A timeout of 0 won't cache the value.

version – It is an int. You can set cache with same key but different version.

cache.get(key, default=None, version=None) – This method is used to get cache. If the key doesn't exists in the cache, it returns None.

Where,

default – This specifies which value to return if the object doesn't exist in the cache.

---

from django.core.cache import cache

cache.add(key, value, timeout=DEFAULT_TIMEOUT, version=None) – This method is used to add a key only if it doesn't already exist. It takes the same parameters as set(), but it will not attempt to update the cache if the key specified is already present. If you need to know whether add() stored a value in the cache, you can check the return value. It will return True if the value was stored, False otherwise.

cache.get_or_set(key, default, timeout=DEFAULT_TIMEOUT, version=None) – This method is used to get a key's value or set a value if the key isn't in the cache. It takes the same parameters as get() but the default is set as the new cache value for that key, rather than returned. You can also pass any callable as a default value.

cache.set_many(dict, timeout) – This method is used to set multiple values more efficiently, use set_many() to pass a dictionary of key-value pairs.

cache.get_many(keys, version=None) - There's also a get_many() interface that only hits the cache once. get_many() returns a dictionary with all the keys you asked for that actually exist in the cache (and haven't expired).

from django.core.cache import cache

cache.delete(key, version=None) – This method is used to delete keys explicitly to clear the cache for a particular object.

cache.delete_many(keys, version=None) – This method is used to clear a bunch of keys at once. It can take a list of keys to be cleared.

cache.clear() – This method is used to delete all the keys in the cache. Be careful with this; clear() will remove everything from the cache, not just the keys set by your application.

cache.touch(key, timeout=DEFAULT_TIMEOUT, version=None) – This method is used to set a new expiration for a key. touch() returns True if the key was successfully touched, False otherwise.

cache.incr(key, delta=1, version=None)

cache.decr(key, delta=1, version=None)

cache.close() - You can close the connection to your cache with close() if implemented by the cache backend.

Signals in django:

Signals

The signals are utilities that allow us to associate events with actions.

Signals allow certain senders to notify a set of receivers that some action has taken place.

- Login and Logout Signals
- Model Signals
- Management Signals
- Request/Response Signals
- Test Signals
- Database Wrappers

signals.py - 25 - Visual Studio Code

signalsAndBuiltinsignalsInDjango > myApp > signals.py > ...

```python
from django.contrib.auth.signals import user_logged_in,user_logged_out,user_login_failed
from django.contrib.auth.models import User
from django.dispatch import receiver #Connecting signal to receiver. iss k ilawa line:19 k code se bhi kr sktay hai.
@receiver(user_logged_in,sender=User)
#Signals receivers function:#first and last parameters are must def login_success(sender,**kwargs)
def login_success(sender,request,user,**kwargs):
    print('Logged In! Signals: ')
    print('sender: ',sender)
    print('request: ',request)
    print('user: ',user)
    print('user hashed password:',user.password)
    print('User Email: ',user.email)
    print(f'Kwargs{kwargs}')

# user_logged_in.connect(login_success,sender=User)  #connecting our signal to receiver
#run python manage.py migrate command
#and create a superuser
# #put :
# def ready(self):
#         import myApp.signals      code inside  apps.py/class

# And put:          'appName.apps.apps.pyClassName'
#default_app_config='myApp.apps.MyappConfig' code in _init_.py file of current app.
#or write : 'myApp.apps.MyappConfig' in settings.py installed apps instead of writing:'myApp'
#Now the above output will be shown in terminal when we logged to admin pannel.

#What can be done through using signals receiver function:
#u can show user's IP address to him.
#u can count howmuch times user have logged in.
#u can store session or a cache
#USER k successfully login honay k bad jo kam krnay hai wo aap kr sktay hai.
#And alot more things, depends upon ur requirments.
```

```python
@receiver(user_logged_out,sender=User)
def log_out(sender,request,user,**kwargs):
    print('_____________________________________')
    print('Logged Out! Signals')
    print('Sender: ',sender)
    print('request: ',request)
    print('user: ',user)
    print(f'kwargs: {kwargs}')
    print('_____________________________________')
#Now the above output will be shown in terminal when we logged out from the admin pannel.
```

```python
48
49
50
51
52  @receiver(user_login_failed)
53  def login_failed(sender,credentials,request,**kwargs):
54      print('_____________________________')
55      print('Logged Failed! Signals: ')
56      print('sender: ',sender)
57      print('request: ',request)
58      print(f'Kwargs{kwargs}')
59      print('Credentials: ',credentials) #it will show u: {'username': 'whatever user have tried to login',
60                                         #'password': 'whatever user have tried to login but in asterisk form.'}
61      print('_________________________________')
62  #user_login_failed.connect(login_failed) u can write this line instead of line 47 but here, not there.
63
64  #Now the above output will be shown in terminal when loggin is failed bcause of entering incorrect username or password.
65
66  #What can be done through using signals login failed function:
67  #kon kiss name se login honay ki try krr rha hai wo aap capture kr sktay ho.
68  #kintni baar user ko ghalat login honay ka permission dena hai. e.g(3 times).
69  #And alot more things, depends upon ur requirments.
```

Signals

Receiver Function - This function takes a sender argument, along with wildcard keyword arguments (**kwargs); all signal handlers must take these arguments. A receiver can be any Python function or method.

```
def receiver_func(sender, request, user, **kwargs):
    pass
```

Connecting/Registering Receiver Function - There are two ways you can connect a receiver to a signal:-

- Manual Connect Route
- Decorator

Manual Connect Route - To receive a signal, register a receiver function using the Signal.connect() method. The receiver function is called when the signal is sent. All of the signal's receiver functions are called one at a time, in the order they were registered.

Signal.connect(receiver_func, sender=None, weak=True, dispatch_uid=None)

Where,

receiver_func – The callback function which will be connected to signal.

sender – Specifies a particular sender to receive signals from.

weak – Django stores signal handlers as weak references by default. Thus, if your receiver is a local function, it may be garbage collected. To prevent this, pass weak=False when you call the signal's connect() method.

dispatch_uid – A unique identifier for a signal receiver in cases where duplicate signals may be sent.

Decorator - @receiver(signal or list of signal, sender)

Built-in Signals

Django provides a set of built-in signals that let user code get notified by Django itself of certain actions.

Login and Logout Signals - The auth framework uses the following signals that can be used for notification when a user logs in or out.

django.contrib.auth.signals

user_logged_in(sender, request, user) - Sent when a user logs in successfully.

sender - The class of the user that just logged in.

request - The current HttpRequest instance.

user - The user instance that just logged in.

user_logged_out(sender, request, user) - Sent when the logout method is called.

sender - The class of the user that just logged out or None if the user was not authenticated.

request - The current HttpRequest instance.

user - The user instance that just logged out or None if the user was not authenticated.

user_login_failed(sender, credentials, request) - Sent when the user failed to login successfully.

sender - The name of the module used for authentication.

credentials - A dictionary of keyword arguments containing the user credentials that were passed to authenticate() or your own custom authentication backend. Credentials matching a set of 'sensitive' patterns, (including password) will not be sent in the clear as part of the signal.

request - The HttpRequest object, if one was provided to authenticate()

Project 25:

signals.py

signalsAndBuiltinsignalsInDjango > myApp > signals.py > ...

```python
from django.db.models.signals import pre_init,pre_save,pre_delete,post_init,post_save,post_delete
from django.dispatch import receiver
@receiver(pre_save,sender=User) #agr hmara apna koi model ho our us ko use krna chahtai hai to User ki jgha
def at_beginning_save(sender,instance,**kwargs):    #us ko import kr k User ki jgha us model ka name likh lengai
    print('_____________Pre Save_________________')          #baqi ka procedure same hai.
    print('Sender: ',sender)
    print('Instance: ',instance)
    print(f'Kwrgs: {kwargs}')
    print('_____________________________')
#ab jb bhi admin pannel mei koi user add hoga to uper wali lines terminal mei print ho jaegi.
#Note: jb login wala f(x) run hota hai to uss k sath ye bydefault run hota hai.



@receiver(post_save,sender=User)
def at_ending_save(sender,instance,created,**kwargs):
    if created:             #mtlb agr new record hai.
        print('_____________Post Save New Record_________________')
        print('Sender: ',sender)
        print('Instance: ',instance)
        print('Created: ',created)
        print(f'Kwrgs: {kwargs}')
        print('_____________________________')
    else:
        print('_____________Post Save Update Record_________________')
        print('Sender: ',sender)
        print('Instance: ',instance)
        print('Created: ',created)
        print(f'Kwrgs: {kwargs}')
        print('_____________________________')
#iss method mei bhi boht saray kaam ho sktay hai jaisay k for e.g: jb record update ho to cache bhi hmara reneve/update
#ho jae.
```

pre_save (sender, instance, raw, using, update_fields) - This is sent at the beginning of a model's save() method.

sender - The model class.

instance - The actual instance being saved.

raw - A boolean; True if the model is saved exactly as presented (i.e. when loading a fixture). One should not query/modify other records in the database as the database might not be in a consistent state yet.

using - The database alias being used.

update_fields - The set of fields to update as passed to Model.save(), or None if update_fields wasn't passed to save().

post_save(sender, instance, created, raw, using, update_fields) - Like pre_save, but sent at the end of the save() method.

sender - The model class.

instance - The actual instance being saved.

created - A boolean; True if a new record was created.

raw - A boolean; True if the model is saved exactly as presented (i.e. when loading a fixture). One should not query/modify other records in the database as the database might not be in a consistent state yet.

using - The database alias being used.

update_fields - The set of fields to update as passed to Model.save(), or None if update_fields wasn't passed to save().

```python

@receiver(pre_delete,sender=User)
def at_beginning_delete(sender,instance,**kwargs):
    print('_____________Pre Delete_________________')
    print('Sender: ',sender)
    print('Instance: ',instance)
    print(f'Kwrgs: {kwargs}')
    print('_____________________________')

@receiver(post_delete,sender=User)
def at_ending_del(sender,instance,**kwargs):
    print('_____________Post Delete_________________')
    print('Sender: ',sender)
    print('Instance: ',instance)
    print(f'Kwrgs: {kwargs}')
    print('_____________________________')
#jaisay he kisi user ko admin pannel se hm delete krangai to pehle pre phir post_delete wala f(x) run hoga.
```

pre_delete(sender, instance, using) - Sent at the beginning of a model's delete() method and a queryset's delete() method.

sender - The model class.

instance - The actual instance being deleted.

using - The database alias being used.

post_delete(sender, instance, using) - Like pre_delete, but sent at the end of a model's delete() method and a queryset's delete() method.

sender - The model class.

instance - The actual instance being deleted.

Note that the object will no longer be in the database, so be very careful what you do with this instance.

using - The database alias being used.

```python

@receiver(pre_init,sender=User)
def at_beginning_init(sender,*args,**kwargs):
    print('__________Pre Init_________________')
    print('Sender: ',sender)
    print(f'args: {args}')
    print(f'kwargs: {kwargs}')
    print('_____________________________')

@receiver(post_init,sender=User)
def at_ending_init(sender,*args,**kwargs):
    print('___________Post Init_________________')
    print('Sender: ',sender)
    print(f'args: {args}')
    print(f'kwargs: {kwargs}')
    print('_____________________________')
#jaise he hm server ko run krayngay to ye dono f(x)s run ho jaengai q k srvr k rn hone se he
#hmare model ki instantiation shuru hoi.
#jb hmara django model instantiate ho tab hamei kya actions lene chahiye wo pre our post_init mei hm likhte hai.


```

Built-in Signals

Model signals - A set of signals sent by the model system.

django.db.models.signals

pre_init (sender, args, kwargs) - Whenever you instantiate a Django model, this signal is sent at the beginning of the model's __init__() method.

sender - The model class that just had an instance created.

args - A list of positional arguments passed to __init__().

kwargs - A dictionary of keyword arguments passed to __init__().

post_init (sender, instance) - Like pre_init, but this one is sent when the __init__() method finishes.

sender - The model class that just had an instance created.

instance - The actual instance of the model that's just been created.

class_prepared(sender) - Sent whenever a model class has been "prepared" – that is, once model has been defined and registered with Django's model system. Django uses this signal internally; it's not generally used in third-party applications.

Since this signal is sent during the app registry population process, and AppConfig.ready() runs after the app registry is fully populated, receivers cannot be connected in that method. One possibility is to connect them AppConfig.__init__() instead, taking care not to import models or trigger calls to the app registry.

sender - The model class which was just prepared.

signals.py × views.py

signalsAndBuiltinsignalsInDjango > myApp > signals.py > ...

```python

from django.core.signals import request_started,request_finished,got_request_exception

@receiver(request_started)
def at_beginning_request(sender,environ,**kwargs):
    print('_______Request at Start_________')
    print('Sender: ',sender)
    print(f'environ: {environ}')
    print(f'kwargs: {kwargs}')
    print('_____________________________________')


@receiver(request_finished)
def at_ending_request(sender,**kwargs):
    print('_______Finish Request_________')
    print('Sender: ',sender)
    print(f'kwargs: {kwargs}')
    print('_____________________________________')
#ab jaise he url hit hoga to request_started wala f(x) run hoga ouur data deliver ho k request_finished wala f(x) run hoga.


@receiver(got_request_exception)
def at_req_exception(sender,request,**kwargs):
    print('_______Getting Request Exception_________')
    print('Sender: ',sender)
    print('request: ',request)
    print(f'kwargs: {kwargs}')
    print('_____________________')
#ab jaise hi koi exception type error occur hoga hmare project mei (e.g: 20/0 output error) to ye f(x) run hoga.

```

Built-in Signals

Request/Response Signals - Signals sent by the core framework when processing a request.

django.core.signals

request_started(sender, environ) - Sent when Django begins processing an HTTP request.

sender - The handler class – e.g. django.core.handlers.wsgi.WsgiHandler – that handled the request.

environ - The environ dictionary provided to the request.

request_finished(sender) - Sent when Django finishes delivering an HTTP response to the client.

sender - The handler class.

got_request_exception(sender, request) - This signal is sent whenever Django encounters an exception while processing an incoming HTTP request.

sender - Unused (always None).

```python

from django.db.models.signals import pre_migrate,post_migrate

@receiver(pre_migrate)
def before_install_app(sender,app_config,verbosity,interactive,using,plan,apps,**kwargs):
    print('______before Install App_______')
    print('print all the aruguments here step by step')

@receiver(post_migrate)
def at_end_migrate_flush(sender,app_config,verbosity,interactive,using,plan,apps,**kwargs):
    print('________after Install App_______________')
    print('print all the aruguments here step by step')

#abb jaise he "python manage.py migrate" command hm terminal me run krangai to us k start pre_migrate
#ouur end mei post_migrate wala f(x) run hoga.


```

Built-in Signals

Management signals – Signals sent by Django-admin

django.db.models.signals

pre_migrate(sender, app_config, verbosity, interactive, using, plan, apps) - Sent by the migrate command before it starts to install an application. It's not emitted for applications that lack a models module.

sender - An AppConfig instance for the application about to be migrated/synced.

app_config - Same as sender.

verbosity - Indicates how much information manage.py is printing on screen.

Functions which listen for pre_migrate should adjust what they output to the screen based on the value of this argument.

interactive - If interactive is True, it's safe to prompt the user to input things on the command line. If interactive is False, functions which listen for this signal should not try to prompt for anything.

For example, the django.contrib.auth app only prompts to create a superuser when interactive is True.

using - The alias of database on which a command will operate.

plan - The migration plan that is going to be used for the migration run. While the plan is not public API, this allows for the rare cases when it is necessary to know the plan. A plan is a list of two-tuples with the first item being the instance of a migration class and the second item showing if the migration was rolled back (True) or applied (False).

apps - An instance of Apps containing the state of the project before the migration run. It should be used instead of the global apps registry to retrieve the models you want to perform operations on

Built-in Signals

post_migrate(sender, aap_config, verbosity, interactive, using, plan, apps) - Sent at the end of the migrate (even if no migrations are run) and flush commands. It's not emitted for applications that lack a models module.

Handlers of this signal must not perform database schema alterations as doing so may cause the flush command to fail if it runs during the migrate command.

sender - An AppConfig instance for the application that was just installed.

app_config - Same as sender.

verbosity - Indicates how much information manage.py is printing on screen.

Functions which listen for post_migrate should adjust what they output to the screen based on the value of this argument.

interactive - If interactive is True, it's safe to prompt the user to input things on the command line. If interactive is False, functions which listen for this signal should not try to prompt for anything.

For example, the django.contrib.auth app only prompts to create a superuser when interactive is True.

using - The database alias used for synchronization. Defaults to the default database.

plan - The migration plan that was used for the migration run. While the plan is not public API, this allows for the rare cases when it is necessary to know the plan. A plan is a list of two-tuples with the first item being the instance of a migration class and the second item showing if the migration was rolled back (True) or applied (False).

apps - An instance of Apps containing the state of the project after the migration run. It should be used instead of the global apps registry to retrieve the models you want to perform operations on.

```python

from django.db.backends.signals import connection_created


@receiver(connection_created)
def conn_db(sender,connection,**kwargs):
    print('__________Initial Connection with db_________________________')
    print('Sender: ',sender)
    print('Connection: ',connection)
    print(f'kwargs: {kwargs}')
#db se initiate hone pe ye f(x) chalega jese for e.g ab agr hm server ko run kray to ye
#sqlite3 db se connect ho k iss f(x) ko run kray ga.
#Note: jitni baar db se hmara connection initiate hoga utni baar ye f(x) chalega.

```

Built-in Signals

Database Wrappers - Signals sent by the database wrapper when a database connection is initiated.

django.db.backends.signals

connection_created - Sent when the database wrapper makes the initial connection to the database. This is particularly useful if you'd like to send any post connection commands to the SQL backend.

sender - The database wrapper class – i.e. django.db.backends.postgresql.DatabaseWrapper or django.db.backends.mysql.DatabaseWrapper, etc.

connection - The database connection that was opened. This can be used in a multiple-database configuration to differentiate connection signals from different databases.

Track and Show Client IP Address: Project:26

Note: it's app = 19.miniBlogProject/homeApp

```
signals.py file:
from django.contrib.auth.signals import user_logged_in
from django.contrib.auth.models import User
from django.dispatch import receiver

#Settings.py file: inside INSTALLED_APPS=['homeApp.apps.HomeappConfig'] instead of writing INST..APPS=['homeApp'].
@receiver(user_logged_in,sender=User)
def logginSuccess(sender,request,user,**kwargs):
    print('________Login Success Signal___________')
    userIP=request.META.get('REMOTE_ADDR') #to get IP address of logged in user.
    print('User IP Address : ',userIP)
    request.session['userIP']=userIP #storing IP address data in session
#Abb jaise hi koi user login hoga to ye f(x) chalega.

#apps.py file: def ready(self):
        # import homeApp.signals
#views.py file:
# def userDashboard(request):
#     if request.user.is_authenticated:
#         dt=blogModel.objects.all()
#         ip=request.session.get('userIP',0)
#         return render(request,'userDashboard.html',{'dbdata':dt,'name':request.user.username,'userIP':ip})
#     else:
#         return HttpResponseRedirect('/Login/')
#.html file:   User IP Address: {{userIP}}
```

Custom Signals in django:

```
#signals.py:
from django.dispatch import Signal, receiver

#Creating Custom Signal:
notification=Signal(providing_args=['request','kuch bhi'])

#Receive Fuction for our Custom Signal:
@receiver(notification) #agr connect method use krna hai to phir receiver ki xrorat nhi hai.
def showNotification(sender,**kwargs):
    print('_______________________')
    print('Sender: ',sender)
    print(f'kwargs: {kwargs}')


#Sending Custom Signal:
#views.py file:
# from .import signals
# def home(request):
#     signals.notification.send(sender=None,request=request,user=['Ali','Khan'])
#     return HttpResponse('working home page')

# #Terminal Output:
# Sender: None
# kwargs: {'signal': <django.dispatch.dispatcher.Signal object at 0x0000016AA18CC220>,
#  'request': <WSGIRequest: GET '/'>, 'user': ['Ali', 'Khan']}

#Note: custom signals ki xrorat bht km parti hai.
```

Defining Custom Signals

All signals are django.dispatch.Signal instances.

class Signal(providing_args=list)

The providing_args is a list of the names of arguments the signal will provide to listeners. This is purely documentational, however, as there is nothing that checks that the signal actually provides these arguments to its listeners.

You’re allowed to change this list of arguments at any time.

Sending signals

There are two ways to send signals in Django.

- Signal.send(sender, **kwargs) – This is used to send a signal, all built-in signals use this to send signals. You must provide the sender argument which is a class most of the time and may provide as many other keyword arguments as you like. It returns a list of tuple pairs [(receiver, response), ...], representing the list of called receiver functions and their response values.
- Signal.send_robust(sender, **kwargs) – This is used to send a signal. You must provide the sender argument which is a class most of the time and may provide as many other keyword arguments as you like. It returns a list of tuple pairs [(receiver, response), ...], representing the list of called receiver functions and their response values.

Difference between send () and send_robust()

- send() does not catch any exceptions raised by receivers; it simply allows errors to propagate. Thus not all receivers may be notified of a signal in the face of an error.
- send_robust() catches all errors derived from Python's Exception class, and ensures all receivers are notified of the signal. If an error occurs, the error instance is returned in the tuple pair for the receiver that raised the error.

Disconnecting Signals

Signal.disconnect(receiver=None, sender=None, dispatch_uid=None) – This is used to disconnect a receiver from a signal. The arguments are as described in Signal.connect(). The method returns True if a receiver was disconnected and False if not.

Middleware in django:

Middleware

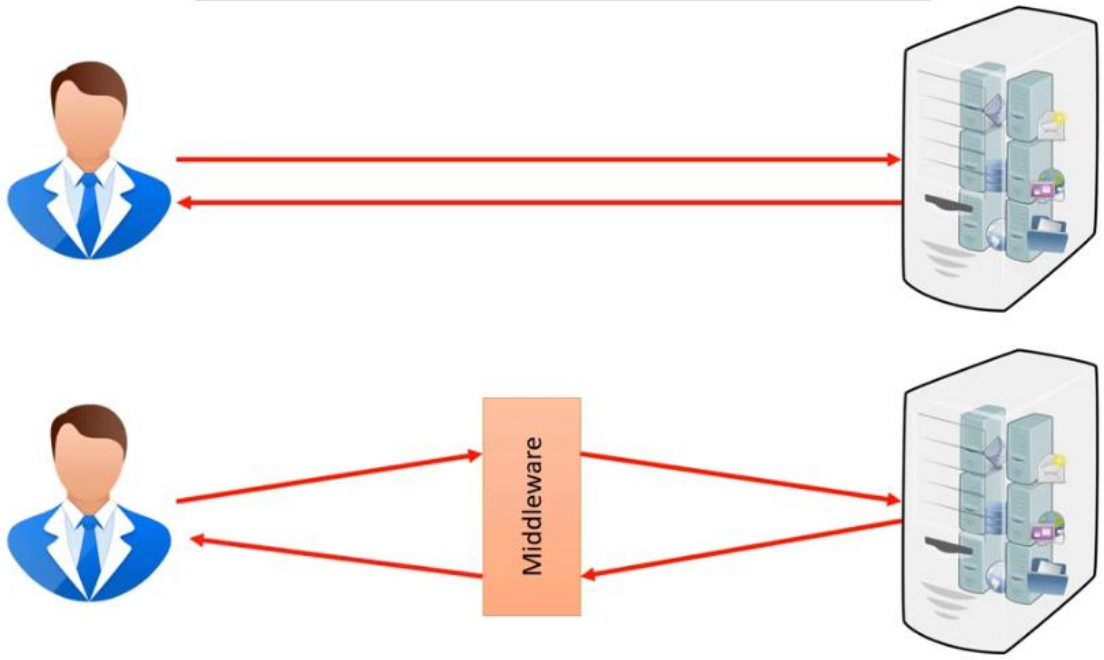
Middleware is a framework of hooks into Django's request/response processing.

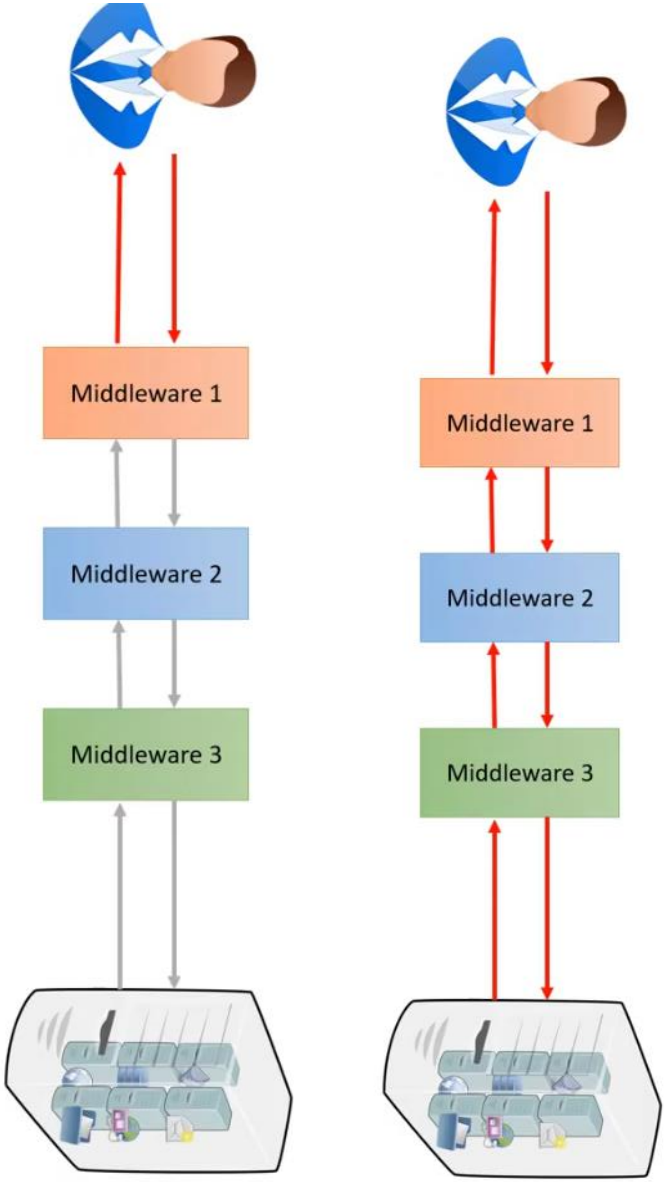
It's a light, low-level "plugin" system for globally altering Django's input or output. Each middleware component is responsible for doing some specific function.

- Built in Middleware
- Custom Middleware

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

How Middleware Works




Middleware 1
Middleware 2
Middleware 3
Middleware 1
Middleware 2
Middleware 3

```
FUNCTION BASED MIDDLEWARE:

middleware.py File:
def myMiddleware(get_response):
    print('One time Initialization/Execution') #yani ye code sirf 1 baar execute hoga, url hit hone pai.
    def myFunction(request):
        print('Run before View') #iss mei jo code hoga view k execute honay se pehle ye execute hoga.
        respnse=get_response(request)
        print('Run after View')
        return respnse
    return myFunction

views.py File:
def home(request):
    print('I am View')
    return HttpResponse('I am View')

settings.py File:
MIDDLEWARE = [
    'django.middleware.clickjacking.XFrameOptionsMiddleware', #tmam middlewares iss k neichei apne sequence k hisab
                                                                                    se likhne hai.
    'myApp.middleware.myMiddleware', #now middleware has been activated/plugged. Agr middlware ko use nhi krana to
]   #'appName.middlewares.pyFileName.f(x)NameInsidemiddlewares.pyFile'                iss line ko mat likho.


Terminal Output:
One time Initialization/Execution
Run before View
I am View
Run after View
```

Function based Middleware

A middleware factory is a callable that takes a get_response callable and returns a middleware.

A middleware is a callable that takes a request and returns a response, just like a view.

```
def my_middleware(get_response):
    # One-time configuration and initialization.
    def my_function(request):
        # Code to be executed for each request before the view are called.
        response = get_response(request)
        # Code to be executed for each request/response after the view is called.
        return response
    return my_function
```

get_response ( )

The get_response callable provided by Django might be the actual view (if this is the last listed middleware) or it might be the next middleware in the chain.

The current middleware doesn't need to know or care what exactly it is, just that it represents whatever comes next.

The get_response callable for the last middleware in the chain won't be the actual view but rather a wrapper method from the handler which takes care of applying view middleware, calling the view with appropriate URL arguments, and applying template-response and exception middleware.

Middleware can live anywhere on your Python path.

Activating Middleware

To activate a middleware component, add it to the MIDDLEWARE list in your Django settings.

In MIDDLEWARE, each middleware component is represented by a string: the full Python path to the middleware factory's class or function name. The order in MIDDLEWARE matters because a middleware can depend on other middleware. For instance, AuthenticationMiddleware stores the authenticated user in the session; therefore, it must run after SessionMiddleware.

Eg. -

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'blog.middlewares.my_middleware'
]
```

App Name .py File Name

```
CLASS BASED MIDDLEWARE:

middleware.py File:
class myMiddleware:
    def __init__(self,get_response):
        self.get_response=get_response
        print('when url hitted')

    def __call__(self,request):
        print('before view')
        response=self.get_response(request)
        print('after view')
        return response


views.py File:
def home(request):
    print('I am View')
    return HttpResponse('I am View')

settings.py File:
MIDDLEWARE = [
    'django.middleware.clickjacking.XFrameOptionsMiddleware', #tmam middlewares iss k neichei apne sequence k hisab
                                                              se likhne hai.
    'myApp.middleware.myMiddleware', #now middleware has been activated/plugged. Agr middlware ko use nhi krana to
]   #'appName.middlewares.pyFileName.classNameInsidemiddlewares.pyFile'                iss line ko mat likho.


Terminal Output:
One time Initialization/Execution #yani ye code sirf pehli bar chalega us k bad kbhi nhi chalega.

#ye code har request pe chalega.
Run before View
I am View
Run after View
```

Class based Middleware

```
class MyMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
        # One-time configuration and initialization.
    def __call__(self, request):
        # Code to be executed for each request before the view (and later middleware) are called.
        response = self.get_response(request)
        # Code to be executed for each request/response after the view is called.
        return response
```

__init__(get_response)

__init__(get_response) - Middleware factories must accept a get_response argument. You can also initialize some global state for the middleware. Keep in mind a couple of caveats:

- Django initializes your middleware with only the get_response argument, so you can't define __init__() as requiring any other arguments.
- Unlike the __call__() method which is called once per request, __init__() is called only once, when the Web server starts.

```
#agr 1 se xyada middlewares hai :

settings.py File:
MIDDLEWARE = [          #initialization iss k descending order mei hoga.
    'myApp.middleware.myMiddleware',
    'myApp.middleware.myMiddlewareTwo',
    'myApp.middleware.myMiddlewareThree',
]
```

```
Terminal Output:
Three Init
Two init
One init
before view one
before view Two
before view Three
I am View
after view Three
after view Two
after view one
```

```
class myMiddlewareTwo:
    def __init__(self,get_response):
        self.get_response=get_response
        print('Two init')

    def __call__(self,request):
        print('before view Two')
        response=HttpResponse('wapas sha')  #agr aap chahte hai k iss hi middleware se he response return ho jae
        print('after view Two')             #to line:23. iss k ilawa template waghaira bhi render kr sktay hai.
        return response
```

```
TERMINAL OUTPUT:
Three Init
Two init
One init
before view one
before view Two
after view Two
after view one

Web page Output:
wapas sha
```

Middleware Hooks

Following are special methods to class-based middleware:

process_view(request, view_func, view_args, view_kwargs) - It is called just before Django calls the view.

It should return either None or an HttpResponse object.

If it returns None, Django will continue processing this request, executing any other process_view() middleware and, then, the appropriate view.

If it returns an HttpResponse object, Django won't bother calling the appropriate view; it'll apply response middleware to that HttpResponse and return the result.

process_view(request, view_func, view_args, view_kwargs)

Where,

Request - It is an HttpRequest object.

view_func – It is the Python function that Django is about to use. (It's the actual function object, not the name of the function as a string.)

view_args – It is a list of positional arguments that will be passed to the view.

view_kwargs – It is a dictionary of keyword arguments that will be passed to the view.

Neither view_args nor view_kwargs include the first view argument (request).

```
from django.http import HttpResponse

class myProcessMiddleware:
    def __init__(self,get_request):
        self.get_request=get_request


    def __call__(self,request):
        response=self.get_request(request)
        return response

    def process_view(request,*args,**kwargs):
        print('process_view before view ')
        return HttpResponse('before view') #agr return None likhngai to django apna process continue rkhayga,mtlb aagay
                                           #ja kr hmare view f(x) ko execute krayga. Lekin yha ye 1 httpResponse
                                           # object return kr rha hai to ye iss say aagay nhi jaega,yani iss mei
                                           # jo hai wo display kr dai ga.
```

```
process_exception hook:

views.py File:
def home2(request):
    a=10/0
    print('I am View')
    return HttpResponse('I am process_exception hook')


middleware.py File:
class myExceptionMiddleware:
    def __init__(self,get_response):
        self.get_response=get_response


    def __call__(self,request):
        response=self.get_response(request)
        return response

    def process_exception(self,request,exception):
        print('process_exception before view ')
        msg=exception
        print('Process Exception')
        className=exception.__class__.__name__  #error class name.
        print(className)
        return HttpResponse(msg)
Note: agr iss middleware/middlewarehook k uper  koi dusra middleware hook hai to uss ka return None laxmi hona chahiye
wrna phir ye execute nhi hoga q k wo neichai ayega hi nhi.
```

process_exception(request, exception) - Django calls process_exception() when a view raises an exception.

It should return either None or an HttpResponse object.

If it returns an HttpResponse object, the template response and response middleware will be applied and the resulting response returned to the browser. Otherwise, default exception handling kicks in.

Where,

Request – It is an HttpRequest object.

Exception – It is an Exception object raised by the view function.

Note - Middleware are run in reverse order during the response phase, which includes process_exception. If an exception middleware returns a response, the process_exception methods of the middleware classes above that middleware won't be called at all.

```
views.py File:
from django.template.response import TemplateResponse
def home3(request):
    print('I am View')
    context={'name':'ali'}
    return TemplateResponse(request,'home.html',context)
```

```
middleware.py File:
class myTemplateResponseMiddleware:
    def __init__(self,get_response):
        self.get_response=get_response


    def __call__(self,request):
        response=self.get_response(request)
        return response

    def process_template_response(self,request,response):
        print('Process Template Response Middleware')
        response.context_data['name']='Khan' #abb jaise he home3 f(x) hit hoga to name ki value ali se
        return response                        # change ho k Khan ho jae gi jha bhi iss ka use huwa hay.
#iss middleware sei hm json data bhi pass kr sktay hai.
```

process_template_response(request, response) – This method is called just after the view has finished executing, if the response instance has a render() method, indicating that it is a TemplateResponse or equivalent.

It must return a response object that implements a render method.

It could alter the given response by changing response.template_name and response.context_data, or it could create and return a brand-new TemplateResponse or equivalent.

You don't need to explicitly render responses, responses will be automatically rendered once all template response middleware has been called.

Where,

request – It is an HttpRequest object.

response – It is the TemplateResponse object (or equivalent) returned by a Django view or by a middleware.

Note - Middleware are run in reverse order during the response phase, which includes process_template_response().

TemplateResponse

TemplateResponse - TemplateResponse is a subclass of SimpleTemplateResponse that knows about the current HttpRequest.

A TemplateResponse object can be used anywhere that a normal django.http.HttpResponse can be used. It can also be used as an alternative to calling render().

Method

__init__(request, template, context=None, content_type=None, status=None, charset=None, using=None) - It instantiates a TemplateResponse object with the given request, template, context, content type, HTTP status, and charset.

Where,

request - An HttpRequest instance.

template - A backend-dependent template object (such as those returned by get_template()), the name of a template, or a list of template names.

context - A dict of values to add to the template context. By default, this is an empty dictionary.

content_type - The value included in the HTTP Content-Type header, including the MIME type specification and the character set encoding. If content_type is specified, then its value is used. Otherwise, 'text/html' is used.

status - The HTTP status code for the response.

charset - The charset in which the response will be encoded. If not given it will be extracted from content_type, and if that is unsuccessful, the DEFAULT_CHARSET setting will be used.

using - The NAME of a template engine to use for loading the template.

TemplateResponse

There are three circumstances under which a TemplateResponse will be rendered:

When the TemplateResponse instance is explicitly rendered, using the SimpleTemplateResponse.render() method.

When the content of the response is explicitly set by assigning response.content.

After passing through template response middleware, but before passing through response middleware.

Note –

A TemplateResponse can only be rendered once.

```
#Built in Middlewares:
we have alot of built-in middlewares in django which proviedes
us diff features e.g security, url normalization and alot more
diff things.
to use a built-in middleware go to last of settings.py and write
it there.
```

Built-in Middleware

CommonMiddleware - Adds a few conveniences for perfectionists:

Forbids access to user agents in the DISALLOWED_USER_AGENTS setting, which should be a list of compiled regular expression objects.

Performs URL rewriting based on the APPEND_SLASH and PREPEND_WWW settings.

If APPEND_SLASH is True and the initial URL doesn't end with a slash, and it is not found in the URLconf, then a new URL is formed by appending a slash at the end. If this new URL is found in the URLconf, then Django redirects the request to this new URL. Otherwise, the initial URL is processed as usual.

For example, geekyshows.com/home will be redirected to geekyshows.com/home/ if you don't have a valid URL pattern for geekyshows.com/home but do have a valid pattern for geekyshows.com/home/.

If PREPEND_WWW is True, URLs that lack a leading "www." will be redirected to the same URL with a leading "www."

Both of these options are meant to normalize URLs. The philosophy is that each URL should exist in one, and only one, place. Technically a URL geekyshows.com/home is distinct from geekyshows.com/home/ a search-engine indexer would treat them as separate URLs – so it's best practice to normalize URLs.

```
#SITE UNDER CONSTRUCTION PROJECT:
middleware.py File:

from django.shortcuts import render

class myMiddleware:
    def __init__(self,get_response):
        self.get_response=get_response

    def __call__(self,request):
        response=render(request,'siteUC.html')
        return response


siteUC.html File:
<html>
<body>
    <h1>Site Under Construction</h1>
</body>
</html>

#abb user jo bhi url hit krayga to uss ko siteUC.html file display ho k mile ga
#jb aap apnay web ke errors wghaira fix kr day to simply settings.py file se
#'myApp.middlewares.myMiddleware' line ko remove krday.
```

QuerySet API Methods: Project(31)

QuerySet API

A QuerySet can be defined as a list containing all those objects we have created using the Django model.

QuerySets allow you to read the data from the database, filter it and order it.

```
from django.http import HttpResponse
from django.shortcuts import render
from .models import std
# Create your views here.
def home(request):
    allObj=std.objects.all()
    print('SQL query of allObj table : ',allObj.query)

    # allObj=std.objects.filter(passingDate='2022-02-07')

    # allObj=std.objects.exclude(city='kuhat') #mtlb peshawar city k ilawa jitne bhi bnde hai
                                                #un ka data lelo/get kro/show kro.



    return render(request,'home.html',{'stds':allObj})
#our table name in SQL query is = appName_modelsTableName
```

Methods that return new QuerySets

Retrieving all objects

all ( ) - This method is used to retrieve all objects. This returns a copy of current QuerySet.

Example:- Student.objects.all()

Retrieving specific objects

- filter (**kwargs) - It returns a new QuerySet containing objects that match the given *lookup parameters*. filter() will always give you a QuerySet, even if only a single object matches the query.

  Example:- Student.objects.filter(marks=70)

- exclude(**kwargs) - It returns a new QuerySet containing objects that do not match the given *lookup parameters*.

  Example:- Student.objects.exclude(marks=70)

```
from django.http import HttpResponse
from django.shortcuts import render
from .models import std
# Create your views here.
def home(request):
    # allObj=std.objects.order_by('marks') #abb ye marks field ko le k assending order mei data dikhayega.
                                            #koi bhi field yha de sktay hai.
    #.order_by('-marks') to marks ko le k descending order mei data show kray ga.
    #.order_by('?') to harr refresh pe alag alag sequence se data ko show kray ga.
    #Note: bydefault id k hisab se asssnding order mei hota hai.
    #Note: small 'a' se capital 'Z' ki priority xyada hai q k ye uni-code(ASCII code/system code)
    #ko le k iss ko assending/descending order mei dikhata hai.

    # allObj=std.objects.all()[:5] #mtlb data of first 5 rows from the table.
    # allObj=std.objects.order_by('id').reverse()[:5] #mtlb 'id' ko le k reverse/descending order
                                                      #me kr k first 5 rows ka data dikha do.
    # [:5] means from index 0 to 4.
    # [2:7] means from index 2 to 6.

    #to select/view some specific fields data from the table.
    # allObj=std.objects.values('name','city') #abb ye output me sirf name ouur city ka data show krayga.
    #agr iss me koi bhi field nhi likhengai to .all() method ki trah kaam kray ga, SQL query agr
    # iss ki dekhengai to [{all the data}].


    return render(request,'home.html',{'stds':allObj})
```

order_by(*fields) – It orders the fields.

- ‘field’ – Asc order
- ‘-field’ – Desc Order
- ‘?’ – Randomly

reverse() – This works only when there is ordering in queryset.

values(*fields, **expressions) - It returns a QuerySet that returns dictionaries, rather than model instances, when used as an iterable. Each of those dictionaries represents an object, with the keys corresponding to the attribute names of model objects.

distinct(*fields) - This eliminates duplicate rows from the query results.

```
from django.http import HttpResponse
from django.shortcuts import render
from .models import std
# Create your views here.
def home(request):
    #abb ye sirf id our name ka data show/get etc  krayga.
    # allObj=std.objects.values_list('id', 'name', named=True)    #it's SQL query: [(allData)]
    #agr blank chordengai to sara deta le le ga.

    #Note: jo cheiz tuple mei hota hai terminal mei print kr k to ham dekh sktay hai lekin
    #wo directly template file mei for loop waghaira se ham
    #show nhi kr sktay, uss k xror kuch code wahgaira likhenge ta k wo data hme output mei
    #show ho jae jese yha named=True hai jo k bydefault False hota hai jiss ki wja se data
    #hme show nhi hota.


    return render(request,'home.html',{'stds':allObj})
```

values_list(*fields, flat=False, named=False) - This is similar to values() except that instead of returning dictionaries, it returns tuples when iterated over.

- If you don't pass any values to values_list(), it will return all the fields in the model, in the order they were declared.
- If you only pass in a single field, you can also pass in the flat parameter. If True, this will mean the returned results are single values, rather than one-tuples.
- You can pass named=True to get results as a namedtuple.

using(alias) - This method is for controlling which database the QuerySet will be evaluated against if you are using more than one database. The only argument this method takes is the alias of a database, as defined in DATABASES.

Example:- student_data = Student.objects.using('default')

dates(field, kind, order='ASC') - It returns a QuerySet that evaluates to a list of datetime.date objects representing all available dates of a particular kind within the contents of the QuerySet.

Where,

field – It should be the name of a DateField of your model.

kind – It should be either "year", "month", "week", or "day".

"year" returns a list of all distinct year values for the field.

"month" returns a list of all distinct year/month values for the field.

"week" returns a list of all distinct year/week values for the field. All dates will be a Monday.

"day" returns a list of all distinct year/month/day values for the field.

order – It should be either 'ASC' or 'DESC'. This specifies how to order the results. defaults to 'ASC'.

Each datetime.date object in the result list is "truncated" to the given type.

Not Practiced below 1 picture:

datetimes(field_name, kind, order='ASC', tzinfo=None) – It returns a QuerySet that evaluates to a list of datetime.datetime objects representing all available dates of a particular kind within the contents of the QuerySet.

field_name – It should be the name of a DateTimeField of your model.

Kind - It should be either "year", "month", "week", or "day".

"year" returns a list of all distinct year values for the field.

"month" returns a list of all distinct year/month values for the field.

"week" returns a list of all distinct year/week values for the field. All dates will be a Monday.

"day" returns a list of all distinct year/month/day values for the field.

order – It should be either 'ASC' or 'DESC'. This specifies how to order the results. defaults to 'ASC'.

tzinfo – It defines the time zone to which datetimes are converted prior to truncation. This parameter must be a datetime.tzinfo object. If it's None, Django uses the current time zone. It has no effect when USE_TZ is False.

Each datetime.datetime object in the result list is "truncated" to the given type.

```
from django.http import HttpResponse
from django.shortcuts import render
from .models import std
# Create your views here.
def home(request):
    #Agr 1 se xyada tables ka data aik saath combinely display krna hai to:
    # qs1=std.objects.values_list('id', 'name', named=True)
    # qs2=teacher.objects.values_list('id', 'name', named=True)
    # allObj=qs2.union(qs1,all=True)  #bydefault duplicate values ko nhi show krta lekin agr
                                      #all=True likhengai to full content yani saray data ko
                                      #show kray ga.

    #intersection means jo data dono tables mei hai. yani duplicate data.
    #allOjb=qs1.difference(qs2) #mtlb qs1 table se qs2 table k data ko minus kro.
    #mtlb qs1 ka wo data jo qs2 table se match krta hai wo show nhi hoga.


    return render(request,'home.html',{'stds':allObj})
```

none() - Calling none() will create a queryset that never returns any objects and no query will be executed when accessing the results. A qs.none() queryset is an instance of EmptyQuerySet.

Example:- student_data = Student.objects.none()

union(*other_qs, all=False) - Uses SQL's UNION operator to combine the results of two or more QuerySets. The UNION operator selects only distinct values by default. To allow duplicate values, use the all=True argument.

Example:- student_data = qs2.union(qs1, all=True)

intersection(*other_qs) – Uses SQL's INTERSECT operator to return the shared elements of two or more QuerySets.

Example:- student_data = qs1.intersection(qs2)

difference(*other_qs) - Uses SQL's EXCEPT operator to keep only elements present in the QuerySet but not in some other QuerySets.

Example:- student_data = qs1.difference(qs2)

Rarely needed methods below 1 picture:

- select_related(*fields)
- defer(*fields)
- only(*fields)
- prefetch_related(*lookups)
- extra(select=None, where=None, params=None, tables=None, order_by=None, select_params=None)
- select_for_update(nowait=False, skip_locked=False, of=())
- raw(raw_query, params=None, translations=None)
- annotate(*args, **kwargs)

Operators that return new QuerySets

AND (&) - Combines two QuerySets using the SQL AND operator.

Example:-

student_data = Student.objects.filter(id=6) & Student.objects.filter(roll=106)

from django.db.models import Q

student_data = Student.objects.filter(id=6, roll=106)

student_data = Student.objects.filter(Q(id=6) & Q(roll=106))

OR (|) - Combines two QuerySets using the SQL OR operator.

Example:-

Student.objects.filter(id=11) | Student.objects.filter(roll=106)

Student.objects.filter(Q(id=11) | Q(roll=106))

QuerySet API part 2: project(32)

Methods that do not return new QuerySets

Retrieving a single object

get ( ) - It returns one single object. If There is no result match it will raise DoesNotExist exception. If more than one item matches the get() query. It will raise MultipleObjectsReturned.

Example:- Student.objects.get(pk=1)

first() - It returns the first object matched by the queryset, or None if there is no matching object. If the QuerySet has no ordering defined, then the queryset is automatically ordered by the primary key.

Example:- student_data = Student.objects.first()

student_data = Student.objects.order_by('name').first()

last() - It returns the last object matched by the queryset, or None if there is no matching object. If the QuerySet has no ordering defined, then the queryset is automatically ordered by the primary key.

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

    # allObj=std.objects.get(pk=1)
                      #.get(yha pr koi bhi field likh sktay hai lekin wo
                      # uniqe hona chahiye e.g: rollNmbr. name ko bhi use kr sktay hai
                      # lekin 2 users k same name tmam table ne nhi hone chahiye
                      # jaise for e.g: .get(name='ali'))
                      #Note: id,name etc aap ke model/table k fields hone chahiye.

    # allObj=std.objects.first() #jo first nmbr pe hoga with respect to id/pk to wo show krayga.
    # allObj=std.objects.order_by('marks').first() #mtlb marks k hisab se order kr k first row dikha do/ get kro.

    #allObj=std.objects.last() #jo last pe hoga with respect to id/pk.

    return render(request,'home.html',{'stds':allObj})
```

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

     # allObj=std.objects.latest('passingDate') #sab se latest rocord ko get kro with respect to date.
     # allObj=std.objects.earliest('passingDate')#sab se oldest/earliest rocord ko get kro with respect to date.
     #Note: 'passingDate' is a dateField object from our models.py file model.
    allObj=std.objects.all()
    print(allObj.exists()) #agr ye data exist krta hai to True nhi to False output dega.

    one=std.objects.get(pk=2)
    print(allObj.filter(pk=one.pk).exists()) #agr pk=2 ka koi record hai to True output dega.


    return render(request,'home.html',{'stds':allObj})
```

latest(*fields) - It returns the latest object in the table based on the given field(s).

Example:- student_data = Student.objects.latest('pass_date')

earliest(*fields) - It returns the earliest object in the table based on the given field(s).

Example:- student_data = Student.objects.earliest('pass_date')

exists() - It returns True if the QuerySet contains any results, and False if not. This tries to perform the query in the simplest and fastest way possible, but it does execute nearly the same query as a normal QuerySet query.

Example:-

student_data = Student.objects.all()

print(student_data.exists())

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

    # newRec=std(name='Khan',roll='19861',city='London',marks=656,passingDate='2021-04-3')
    # newRec.save()
    #abb in uper 2 lines se hmara 1 nya record create ouur save hojaega db me lekin agr iis
    #cheix ko 1 line me krna hai to neichei line wala method use krayngai.
    # allObj=std.objects.create(name='Khan',roll='19861',city='London',marks=656,passingDate='2021-04-3')
    #abb ye iss object ko create and save kr day ga db mei.


    allObj,created=std.objects.get_or_create(name='Khan',roll='19861',city='London',marks=656,passingDate='2021-04-3')
    print(created) #agr ye record create ho chuka hai to iss ka output True hoga lekin agr pehle se mojod hai
    #to terminal mei False output show hoga. "allObj,created="ye python ka 1 concept hai. Note: created ki jga
    #kuch bhi likh sktay hai.


    return render(request,'home.html',{'stds':allObj})
```

Methods that do not return new QuerySets

create(**kwargs) - A convenience method for creating an object and saving it all in one step.

Example:-

s = Student(name='Sameer', roll=112, city='Bokaro', marks=60, pass_date='2020-5-4')

s.save(force_insert=True)

s = Student.objects.create(name='Sameer', roll=112, city='Bokaro', marks=60, pass_date='2020-5-4')

get_or_create(defaults=None, **kwargs) - A convenience method for looking up an object with the given kwargs (may be empty if your model has defaults for all fields), creating one if necessary.

It returns a tuple of (object, created), where object is the retrieved or created object and created is a boolean specifying whether a new object was created.

Example:-

student_data, created = Student.objects.get_or_create(name='Sameer', roll=112, city='Bokaro', marks=60, pass_date='2020-5-4')

print(student_data, created)

Methods that do not return new QuerySets

update(**kwargs) - Performs an SQL update query for the specified fields, and returns the number of rows matched (which may not be equal to the number of rows updated if some rows already have the new value).

Example:-

student_data = Student.objects.filter(id=12).update(name='Kabir', marks=80)

Update student's city Pass who has marks 60

student_data = Student.objects.filter(marks=60).update(city='Pass')

agr iss ko(yani student_data object ko) print krayngai to jitne nmbr of rows me editing hui hai un ki taadad print kr dai ga.

agr iss trah likhengai to error ayega q k ye 1 object return krta hai ouur .update() method sqlQuerySet pe kaam krta hai.

student_data = Student.objects.get(id=12).update(name='Kabir', marks=80)

update_or_create(defaults=None, **kwargs) – A convenience method for updating an object with the given kwargs, creating a new one if necessary. The defaults is a dictionary of (field, value) pairs used to update the object. The values in defaults can be callables.

It returns a tuple of (object, created), where object is the created or updated object and created is a boolean specifying whether a new object was created.

The update_or_create method tries to fetch an object from database based on the given kwargs. If a match is found, it updates the fields passed in the defaults dictionary.

Example:-

```
student_data, created = Student.objects.update_or_create(id=14, name='Kohli',
defaults={'name':'Sameer'})
```

agr id=14 and name='Kohli' ka koi record hai to uss k name ko update kr k 'Sameer' krdo nhi to create kr do.
Note : create krnay k liye defaults={saray fields} ka mention krna laxmi hai, agr iss se kaam na hojae to .update_or_create(yha pr bhi saray fields) ka mention krna hoga.

```
objs = [
 Student(name='Sonal', roll=120, city='Dhanbad', marks=40, pass_date='2020-5-4'),
 Student(name='Kunal', roll=121, city='Dumka', marks=50, pass_date='2020-5-7'),
 Student(name='Anisa', roll=122, city='Giridih', marks=70, pass_date='2020-5-9') ]
student_data = Student.objects.bulk_create(objs)
```

abb iss se ye 3 records hmare db mei createAndSave ho jaengai.

bulk_create(objs, batch_size=None, ignore_conflicts=False) – This method inserts the provided list of objects into the database in an efficient manner.

The model's save() method will not be called, and the pre_save and post_save signals will not be sent.

It does not work with child models in a multi-table inheritance scenario.

If the model's primary key is an AutoField it does not retrieve and set the primary key attribute, as save() does, unless the database backend supports it (currently PostgreSQL).

It does not work with many-to-many relationships.

It casts objs to a list, which fully evaluates objs if it's a generator. The cast allows inspecting all objects so that any objects with a manually set primary key can be inserted first.

The batch_size parameter controls how many objects are created in a single query. The default is to create all objects in one batch, except for SQLite where the default is such that at most 999 variables per query are used.

On databases that support it (all but Oracle), setting the ignore_conflicts parameter to True tells the database to ignore failure to insert any rows that fail constraints such as duplicate unique values. Enabling this parameter disables setting the primary key on each model instance.

bulk_update(objs, fields, batch_size=None) - This method efficiently updates the given fields on the provided model instances, generally with one query. QuerySet.update() is used to save the changes, so this is more efficient than iterating through the list of models and calling save() on each of them.

You cannot update the model's primary key.

Each model's save() method isn't called, and the pre_save and post_save signals aren't sent.

If updating a large number of columns in a large number of rows, the SQL generated can be very large. Avoid this by specifying a suitable batch_size.

Updating fields defined on multi-table inheritance ancestors will incur an extra query per ancestor.

If objs contains duplicates, only the first one is updated.

The batch_size parameter controls how many objects are saved in a single query. The default is to update all objects in one batch, except for SQLite and Oracle which have restrictions on the number of variables used in a query.

Example:-

```
all_student_data = Student.objects.all()
for stu in all_student_data:
 stu.city = 'Bhel'
student_data = Student.objects.bulk_update(all_student_data, ['city'])
```

iss code say jitne bhi users/students hai db mei, unn ka sb k city field me Bhel aajaega.
Note: Bhel is the name of a city.

in_bulk(id_list=None, field_name='pk') – It takes a list of field values (id_list) and the field_name for those values, and returns a dictionary mapping each value to an instance of the object with the given field value. If id_list isn't provided, all objects in the queryset are returned. field_name must be a unique field, and it defaults to the primary key.

Example:-

```
student_data = Student.objects.in_bulk([1, 2])
print(student_data[1].name)
print()
student_data1 = Student.objects.in_bulk([])
print(student_data1)
print()
student_data2 = Student.objects.in_bulk()
print(student_data2)
```

1 and 2 r primary keys. mtlb sirf in dono fields ka querySet object lelo/get kro.

empty list of objects

it will return all the objects of the querySet of Student table.

delete() - The delete method, conveniently, is named delete(). This method immediately deletes the object and returns the number of objects deleted and a dictionary with the number of deletions per object type.

Example:-

Delete One Record

student_data = Student.objects.get(pk=22)

deleted = student_data.delete()

Delete in Bulk

You can also delete objects in bulk. Every QuerySet has a delete() method, which deletes all members of that QuerySet.

Example:- student_data = Student.objects.filter(marks=50).delete() mtlb jiss k bhi 50 marks hai un ka record delete krdo.

Delete All Records

Example:- student_data = Student.objects.all().delete()

count() - It returns an integer representing the number of objects in the database matching the QuerySet. A count() call performs a SELECT COUNT(*) behind the scenes.

Example:-

student_data = Student.objects.all()

print(student_data.count())

jitne nmbrs of rows/records hai hmare querySet/db mei unn ki taadad ko print krdaiga.

QuerySet API part 3: i.e Field Lookups: (project:33)

Field Lookups

Field lookups are how you specify the meet of an SQL WHERE clause.

They're specified as keyword arguments to the QuerySet methods filter(), exclude() and get().

If you pass an invalid keyword argument, a lookup function will raise TypeError.

Syntax:- field__lookuptype=value

Example:- Student.objects.filter(marks__lt='50')

SELECT * FROM myapp_student WHERE marks < '50';

The field specified in a lookup has to be the name of a model field.

In case of a ForeignKey you can specify the field name suffixed with _id. In this case, the value parameter is expected to contain the raw value of the foreign model's primary key.

Example:- Student.objects.filter(stu_id=10)

```
from django.shortcuts import render
from .models import std
# Create your views here.
def home(request):
    # allObj=std.objects.all()  #Note: jb bhi hm yha .get() use krtay hai to wo aik single object ko get krta hai to use
    # print('allObjects: ',allObj)    #waqt template file mei for loop use nhi krna. agr kiya hai to error ayega.
    # print()
    # print('SQLQuery: ',allObj.query)


    # allObj=std.objects.filter(name__exact='Ali')    #bydefault ye case-sensitive hota hai agr case-sensitivity
                                                     #htani hai to name__iexact='ali' iss trah exact k saath
                                                     #i likhna hai. Note: ye lookup diff databases mei diff reactions
                                                         #krta hai.
    # allObj=std.objects.filter(name__contains='a') #abb name field k jis record mei bhi 'a' hoga wo show/get krdai ga.
                                                     #Note: case-sensitive nhi hai iss db mei. Lekin agr aap chahte hai
                                                     #k iss db me bad mei koi masla na aye case-sensitivity
                                                     # ki wja se to 'icontains' likhye.

    # allObj=std.objects.filter(id__in=[1,4,5])   #mtlb id 1,4,5 wale records get/show krdo.
    # allObj=std.objects.filter(marks__in=[400])   #another example.

    return render(request,'home.html',{'std':allObj})
```

exact - Exact match. If the value provided for comparison is None, it will be interpreted as an SQL NULL. This is case sensitive

Example:- Student.objects.get(name__exact='sonam')

iexact - Exact match. If the value provided for comparison is None, it will be interpreted as an SQL NULL. This is case insensitive

Example:- Student.objects.get(name__iexact='sonam')

contains - Case-sensitive containment test.

Example:- Student.objects.get(name__contains='kumar')

icontains - Case-insensitive containment test.

Example:- Student.objects.get(name__icontains='kumar')

in - In a given iterable; often a list, tuple, or queryset. It's not a common use case, but strings (being iterables) are accepted.

Example:- Student.objects.filter(id__in=[1, 5, 7])

```
from django.shortcuts import render
from .models import std
# Create your views here.
def home(request):

    # allObj=std.objects.filter(marks__gt=400) #jin k marks 400 se xyada hai un ka record filter/get/show kro.
    # allObj=std.objects.filter(marks__gte=400) #greater than or equal to 400.
    # allObj=std.objects.filter(marks__lt=400) #less than 400.
    # allObj=std.objects.filter(marks__lte=400) #less than or equal to 400.


    return render(request,'home.html',{'std':allObj})
```

gt - Greater than.

Example: - Student.objects.filter(marks__gt=50)

gte - Greater than or equal to.

Example: - Student.objects.filter(marks__gte=50)

lt - Less than.

Example: - Student.objects.filter(marks__lt=50)

lte - Less than or equal to.

Example: - Student.objects.filter(marks__lte=50)

startswith - Case-sensitive starts-with.

Example: - Student.objects.filter(name__startswith='r')

istartswith - Case-insensitive starts-with.

Example: - Student.objects.filter(name__istartswith='r')

endswith - Case-sensitive ends-with.

Example:- Student.objects.filter(name__endswith='j')

iendswith - Case-insensitive ends-with.

Example:- Student.objects.filter(name__iendswith='j')

```
from django.shortcuts import render
from .models import std
# Create your views here.
def home(request):

    # allObj=std.objects.filter(marks__range=(320,420)) #320 se 420 tak. 320 ouur 420 bhi include honge.
                                                #iss mei date waghaira k fields bhi likh sktay hai.

    # from datetime import date
    # allObj=std.objects.filter(admiDatetime__date=date(2022,1,5))#jiss ka admission time ka date ye hoga, un ka re-
                                                                #-cord filter kr dayga.
                                                                #Note: ye sirf DateTimeField() wale object k
                                                                #k saath kaam krta hai.
    # allObj=std.objects.filter(admiDatetime__date__gt=date(2022,1,5))#iss trah diff filters ko combine bhi kr
    #sktay hai. Abb jiss ka admission time ka date iss se greater hoga.

    # allObj=std.objects.filter(passDate__year=2022)
    # allObj=std.objects.filter(admiDatetime__year=2022)
    # allObj=std.objects.filter(admiDatetime__year__lt=2022)


    return render(request,'home.html',{'std':allObj})
```

range - Range test (inclusive).

Example:- Student.objects.filter(passdate__range=('2020-04-01', '2020-05-05'))

SQL: - SELECT ... WHERE admission_date BETWEEN '2020-04-01' and '2020-05-05';

You can use range anywhere you can use BETWEEN in SQL — for dates, numbers and even characters.

date - For datetime fields, casts the value as date. Allows chaining additional field lookups. Takes a date value.

Example:-

Student.objects.filter(admdatetime__date=date(2020, 6, 5))

Student.objects.filter(admdatetime__date__gt=date(2020, 6, 5))

year - For date and datetime fields, an exact year match. Allows chaining additional field lookups. Takes an integer year.

Example:-

Student.objects.filter(passdate__year=2020)

Student.objects.filter(passdate__year__gt=2019)

month - For date and datetime fields, an exact month match. Allows chaining additional field lookups. Takes an integer 1 (January) through 12 (December).

Example:-

Student.objects.filter(passdate__month=6)

Student.objects.filter(passdate__month__gt=5)

day - For date and datetime fields, an exact day match. Allows chaining additional field lookups. Takes an integer day.

Example:-

Student.objects.filter(passdate__day=5)

Student.objects.filter(passdate__day__gt=3)

This will match any record with a pub_date on the third day of the month, such as January 3, July 3, etc.

```
from django.shortcuts import render
from .models import std
# Create your views here.
def home(request):

    # allObj=std.objects.filter(admiDatetime__week=22) #1 year mei 52 ya 53 weeks hote hai uss k hisaab se dene hai.
    # allObj=std.objects.filter(passDate__week=22) #another e.g
    # allObj=std.objects.filter(passDate__week__lt=50) #another e.g

    # allObj=std.objects.filter(passDate__week_day=1) #jiss ka bhi passing date sunday pe hoga un ka record filter krdayga.
    # allObj=std.objects.filter(passDate__week_day__gt=1)#another e.g.  Note: week k baad 1 underscore hai.


    return render(request,'home.html',{'std':allObj})
```

week - For date and datetime fields, return the week number (1-52 or 53) according to ISO-8601, i.e., weeks start on a Monday and the first week contains the year's first Thursday.

Example:-

Student.objects.filter(passdate__week=23)

Student.objects.filter(passdate__week__gt=22)

week_day - For date and datetime fields, a 'day of the week' match. Allows chaining additional field lookups.

Takes an integer value representing the day of week from 1 (Sunday) to 7 (Saturday).

Example:-

Student.objects.filter(passdate__week_day=6)

Student.objects.filter(passdate__week_day__gt=5)

This will match any record with a admission_date that falls on a Monday (day 2 of the week), regardless of the month or year in which it occurs. Week days are indexed with day 1 being Sunday and day 7 being Saturday.

quarter - For date and datetime fields, a 'quarter of the year' match. Allows chaining additional field lookups. Takes an integer value between 1 and 4 representing the quarter of the year.

Example to retrieve entries in the second quarter (April 1 to June 30):

Student.objects.filter(passdate__quarter=2)

time - For datetime fields, casts the value as time. Allows chaining additional field lookups. Takes a datetime.time value.

from datetime import time

Example:- Student.objects.filter(admdatetime__time__gt=time(6,00))

e.g no 2: Student.objects.filter(admdatetime__time__gt=9

time(hour,minutes)

hour - For datetime and time fields, an exact hour match. Allows chaining additional field lookups. Takes an integer between 0 and 23.

Example:- Student.objects.filter(admdatetime__hour__gt=5)

minute - For datetime and time fields, an exact minute match. Allows chaining additional field lookups. Takes an integer between 0 and 59.

Example:- Student.objects.filter(admdatetime__minute__gt=50)

second - For datetime and time fields, an exact second match. Allows chaining additional field lookups. Takes an integer between 0 and 59.

Example:- Student.objects.filter(admdatetime__second__gt=30)

isnull - Takes either True or False, which correspond to SQL queries of IS NULL and IS NOT NULL, respectively.

Example:- Student.objects.filter(roll__isnull=False)

agr hm ne apne mdel k roll field mei null=False kiya hai to ye data ko filter kr layga otherwise nhi kray ga.

regex

iregex

Imp Note: allObj=std.objects.filter(marks__gt=400).filter(passDate__week_day__gt=1) iss trah bhi 2 ya 2 se xyada fieldLookups ko combine kiya ja skta hai.

Aggregation

sometimes you will need to retrieve values that are derived by summarizing or aggregating a collection of objects.

aggregate() - It is a terminal clause for a QuerySet that, when invoked, returns a dictionary of name-value pairs. The name is an identifier for the aggregate value; the value is the computed aggregate. The name is automatically generated from the name of the field and the aggregate function.

Syntax:- aggregate(name=agg_function('field'), name=agg_function('field'),)

field - It describes the aggregate value that we want to compute.

name - If you want to manually specify a name for the aggregate value, you can do so by providing that name when you specify the aggregate clause.

```
from django.shortcuts import render
from .models import std
from django.db.models import Avg,Sum,Min,Max,Count

def home(request):
    allObj=std.objects.all()
    average=allObj.aggregate(Avg('marks'))
    print('Average of all the marks of the Students is :',average)

    sum=allObj.aggregate(Sum('marks')) #sum of all the values in marks field.
    mini=allObj.aggregate(Min('marks'))#Min of all the values in marks field.
    maxi=allObj.aggregate(Max('marks'))#Max of all the values in marks field.
    countAll=allObj.aggregate(Count('marks'))#Nmbr of all the values in marks field.
    return render(request,'home.html',{'std':allObj,'average':average,'sum':sum,'min':mini,'max':maxi,'count':countAll})
#Note: it's app file = project 33/homeApp

#These values r stored in dictionary form if u print it, here is how to display it properly
#on web page: home.html File:                                                    Output:

# <h2>Average of marks is : {{average}}</h2>                Average of marks is : {'marks__avg': 423.6}
# <h2>Average of marks is : {{average.marks__avg}}</h2>     Average of marks is : 423.6

# <p>Sum  : {{sum}}</p>                                     Sum : {'marks__sum': 2118}
# <p>Sum : {{sum.marks__sum}}</p>                           Sum : 2118

# <p>Minimum : {{min}}</p>                                  Minimum : {'marks__min': 320}
# <p>Minimum : {{min.marks__min}}</p>                       Minimum : 320

# <p>Maximum : {{max}}</p>                                  Maximum : {'marks__max': 500}
# <p>Maximum : {{max.marks__max}}</p>                       Maximum : 500

# <p>Count All : {{count}}</p>                              Count All : {'marks__count': 5}
# <p>Count All : {{count.marks__count}}</p>                 Count All : 5
#Note: marks is a field of our db model it's giving output according to the data stored in that
#db table of that field(which is marks here).
```

Q Objects

Q object is an object used to encapsulate a collection of keyword arguments. These keyword arguments are specified as in "Field lookups" .

If you need to execute more complex queries, you can use Q objects.

Q objects can be combined using the & and | operators. When an operator is used on two Q objects, it yields a new Q object.

from django.db.models import Q

& (AND) Operator

Example:- Student.objects.filter(Q(id=6) & Q(roll=106))

| (OR) Operator

Example:- Student.objects.filter(Q(id=6) | Q(roll=108))

~ Negation Operator

Example:- Student.objects.filter(~Q(id=6))

mtlb id=6 k ilawa saaray records ko filter kr lo.

Limiting QuerySets

Use a subset of Python's array-slicing syntax to limit your QuerySet to a certain number of results. This is the equivalent of SQL's LIMIT and OFFSET clauses.

Student.objects.all()[:5] - This returns First 5 objects

Student.objects.all()[5:10] - This returns sixth through tenth objects

Student.objects.all()[-1] - This is not valid.

Student.objects.all()[:10:2] - This returns a list of every second object of the first 10.

Model Inheritance In django: Project(35)

Model Inheritance

Model inheritance in Django works almost identically to the way normal class inheritance works in Python. The base class should subclass django.db.models.Model.

- Abstract Base Classes
- Multi-table Inheritance
- Proxy Models

```
ABSTRACT BASE CLASS DEFINITION:
jb ham chahte hai k jo common fields ka model hai hmara, uss ka table db me na bne lekin wo
modles.py mei hmare un models ko data inherite krday jin ka table db me bnayga to tb hmm
abstract base class use krtay hai.

Example:
models.py File:

from django.db import models

# Create your models here.
class commonFields(models.Model):
    name=models.CharField(max_length=70)
    age=models.IntegerField()
    date=models.DateField()
    class Meta:
        abstract=True

class std(commonFields):
    fees=models.IntegerField()
    date=None   #iss ko likhne se iss table mei date field nhi bnay ga.

class teacher(commonFields):
    salary=models.IntegerField()

class contractor(commonFields):
    date=models.DateTimeField()     #iss se date field jo k commonFields model se inherite ho rha hai
                                    #wo override ho jae ga.
```

Abstract Base Classes

Abstract base classes are useful when you want to put some common information into a number of other models.

You write your base class and put *abstract=True* in the *Meta* class.

This model will then not be used to create any database table. Instead, when it is used as a base class for other models, its fields will be added to those of the child class.

It does not generate a database table or have a manager, and cannot be instantiated or saved directly.

Fields inherited from abstract base classes can be overridden with another field or value, or be removed with None.

```
from django.db import models
class CommonInfo(models.Model):
    name = models.CharField(max_length=70)
    age = models.IntegerField()
    class Meta:
        abstract = True

class Student(CommonInfo):
    fees = models.IntegerField()

class Teacher(CommonInfo):
    salary = models.IntegerField()
```

Meta Inheritance - When an abstract base class is created, Django makes any Meta inner class you declared in the base class available as an attribute.

If a child class does not declare its own Meta class, it will inherit the parent's Meta.

If the child wants to extend the parent's Meta class, it can subclass it.

Django does make one adjustment to the Meta class of an abstract base class: before installing the Meta attribute, it sets abstract=False.

This means that children of abstract base classes don't automatically become abstract classes themselves.

You can make an abstract base class that inherits from another abstract base class. You just need to remember to explicitly set abstract=True each time.

```
from django.db import models
class CommonInfo(models.Model):
    name = models.CharField(max_length=70)
    age = models.IntegerField()
    class Meta:
        abstract = True

class Student(CommonInfo):
    fees = models.IntegerField()
    class Meta:
        abstract = False
```

MULTI-TABLE INHERITANCE:

Example:
models.py File:

```
from django.db import models

# Create your models here.
class examCenter(models.Model):
    centerName=models.CharField(max_length=70)
    city=models.CharField(max_length=70)

class std(examCenter):
    name=models.CharField(max_length=70)
    roll=models.IntegerField()      #abb iss mei examCenter model wale wale 2 fields inherite ho
    #jaengai lekin data dono ka alag alag table mei save hoga.
    #jb hm iss models(std) mei koi record insert kreingai to centerName ouur city fields ka data bydefault
    #examCenter wale model mei automatically save ho jaega 1 foreign key k saath/same id of that record.
```

Multi-table Inheritance

In this inheritance each model have their own database table, which means base class and child class will have their own table.

The inheritance relationship introduces links between the child model and each of its parents (via an automatically-created OneToOneField).

```
from django.db import models

class ExamCenter(models.Model):

  cname = models.CharField(max_length=70)

  city = models.CharField(max_length=70)


class Student(ExamCenter):

 name = models.CharField(max_length=70)

 roll = models.IntegerField()
```

All of the fields of ExamCenter will also be available in Student, although the data will reside in a different database table.

PROXY MODEL INHERITANCE:

Example:
models.py File:

```
from django.db import models

class examCenter(models.Model):
    centerName=models.CharField(max_length=70)
    city=models.CharField(max_length=70)

class myExamCenter(examCenter):
    class Meta:
        proxy=True
#abb inn me se jiss bhi 1 table mei koi record save kreingai to dosre mei wo
#khud bakhud aajaega/save ho jaega.
```

```
from django.db import models
class ExamCenter(models.Model):
  cname = models.CharField(max_length=70)
  city = models.CharField(max_length=70)


class MyExamCenter(ExamCenter):
 class Meta:
   proxy = True
   ordering = ['city']
 def do_something(self):
   pass
```

iss se iss table k records ko city field k respect se asscending order mei kr do.

agr descending mei krna hai to ['-city'] likhna hoga.

Proxy Model

Sometimes, however, you only want to change the Python behavior of a model – perhaps to change the default manager, or add a new method.

This is what proxy model inheritance is for: creating a proxy for the original model. You can create, delete and update instances of the proxy model and all the data will be saved as if you were using the original (non-proxied) model. The difference is that you can change things like the default model ordering or the default manager in the proxy, without having to alter the original.

Proxy models are declared like normal models. You tell Django that it's a proxy model by setting the proxy attribute of the Meta class to True.

- A proxy model must inherit from exactly one non-abstract model class.
- You can't inherit from multiple non-abstract models as the proxy model doesn't provide any connection between the rows in the different database tables.
- A proxy model can inherit from any number of abstract model classes, providing they do not define any model fields.
- A proxy model may also inherit from any number of proxy models that share a common non-abstract parent class.
- If you don't specify any model managers on a proxy model, it inherits the managers from its model parents.
- If you define a manager on the proxy model, it will become the default, although any managers defined on the parent classes will still be available.

Model Manager in django:

Model Manager

A Manager is the interface through which database query operations are provided to Django models. At least one Manager exists for every model in a Django application.

Model manager is used to interact with database.

By default, Django adds a Manager with the name objects to every Django model class.

django.db.models.manager.Manager

```
MODEL MANAGER IN DJANGO:
models.py File:
from django.db.models import models

#Note: hr 1 model class k liye apna apna manager hota hai jo k bydefault 'objects' hota hai.
#e.g: allObj=std.objects.all()

class examCenter(models.Model):
    centerName=models.CharField(max_length=70)
    city=models.CharField(max_length=70)
    #by default backend pe :
    # objects=models.Manager()  #iss trah hota hai.
    #Agr iss ko change krna hai for e.g:
    #students=models.Manager()  #abb hm objects ki bjae students ka use krangai.
                                #jese for e.g: allObj=std.students.all()
```

Custom Model Manager

You can use a custom Manager in a particular model by extending the base Manager class and instantiating your custom Manager in your model.

A custom Manager method can return anything you want. It doesn't have to return a QuerySet.

- to modify the initial QuerySet the Manager returns
- to add extra Manager methods

```
managers.py file(created by us in the current app):

#Note: ye kaam directly models.py file mei bhi kiya ja skta hai.

# from django.db import models

#by default ye iss trah hota hai:
# class CustomManager(models.Manager):
#     def get_queryset(self):
#         return super().get_queryset().all()



#Proper Example:
# class CustomManager(models.Manager):
#     def get_queryset(self):
#         return super().get_queryset().order_by('name')

#models.py File:
# from .managers import CustomManager
# class std(models.Model):
#     name=models.CharField(max_length=90)
#     roll=models.IntegerField()

#     students=CustomManager()
     #objects=models.Manager() ye line tb likhna hai jb aap chahte hai k hmara custom apne trikay say kaam kray.
#views.py File:                 #views.py file for above line: allObjs=std.objects.all()
    # allObj=std.students.all()

#Output: iss se hmara sara data name field k hisab se order ho k mil jaega.
```

Modify the initial QuerySet

A Manager's base QuerySet returns all objects in the system. You can override a Manager's base QuerySet by overriding the Manager.get_queryset() method. get_queryset() should return a QuerySet with the properties you require.

Write Model Manager

```
class CustomManager(models.Manager):
  def get_queryset(self):        # overriding Built-in method called when we call all()
    return super().get_queryset().order_by('name')
```

Associate Manager with Model

```
Class Student(models.Model):
  objects = models.Manager()                # Default Manager
  students = CustomManager()                # Custom Manager
```

You can associate more than one manager in one Model

views.py

```
Student_data = Student.objects.all()        # Work as per default Manager
Student_data = Student.students.all()       # Work as per Custom manager
```

```
ADD EXTRA MANAGER METHODS:
#models.py File:
# from .managers import CustomManager
# class std(models.Model):
#     name=models.CharField(max_length=90)
#     roll=models.IntegerField()
#     students=CustomManager()


#managers.py file
# class CustomManager(models.Manager):
#     def stdRoll_range(self,r1,r2):
#         return super().get_queryset().filter(roll_range=(r1,r2))
# #jb aap ko bar bar koi method likhni parti hai to uss se behtar hai k uss k liye
#apna manager bnaya jaye.

#views.py File:
# allObj=std.students.astdRoll_range(5,10) #5 se 10 tak roll walo ka data show krayga.
```

Add extra Manager methods

Adding extra Manager methods is the preferred way to add "table-level" functionality to your models.

Write Model Manager

```
class CustomManager(models.Manager):
 def get_stu_roll_range(self, r1, r2):
  return super().get_queryset().filter(roll__range=(r1, r2))
```

Associate Manager with Model

```
Class Student(models.Model):
objects = models.Manager()
 students = CustomManager()
```

views.py

```
Student_data = Student.objects.all()
Student_data = Student.students.get_stu_roll_range(101, 103)
```

```
#AN EXAMPLE OF COMBININGE PROXY MODEL INHERITANCE AND MODEL MANAGER:

from django.db import models
from .managers import CustomManager
class std(models.Model):
    name=models.CharField(max_length=90)
    roll=models.IntegerField()

class stdProxy(std):
    students=CustomManager()
    class Meta:
        proxy=True
        ordering=['name']

#views.py File :
# allObj=stdProxy.students.astdRoll_range(5,10) #5 se 10 tak roll walo ka data show krayga.

#Yha prr proxy model ka use kr k hm ne dono classess ka behaviour alg alg kr dia.
```

Model Relationships: Project(36)

Model Relationship

Django offers ways to define the three most common types of database relationships

- One to One Relationship
- Many to One Relationship
- Many to Many Relationships

Note: on_delete,limit_choices,parent_link wghaira saaray methods inn 3no relationships k liye kaam krte he, mtlb k use kiye ja sktay hai.

One to One

when one row of table 'A' can be linked to only one row of table 'B' is called One-to-One relationship. e.g: jaise yha pr rahul sirf 1 page bna skta hai jo k geekyshows uss ne bnaya iss trah sonam ouur kunal bhi..

User

ID	Username	Password
1	Rahul	rahul12
2	Sonam	sonam12
3	Kunal	kunal12

Page

ID	Page Name	Page Cat	Page Publish Date
1	Geekyshows	Programming	12-12-2000
2	World News	News	11-09-2003
3	DjangoLove	Programming	07-01-2001

```
models.py File:
from django.db import models
from django.contrib.auth.models import User
class page(models.Model):           #iss mei jo 1 int value ayega(relationship ki wja se) foreign key ki shkl mei
                                    #uss ko hm ne primary key bna diya.
    usr=models.OneToOneField(User,on_delete=models.CASCADE,primary_key=True)     #user k bjae apna model bhi use krsktay hai
                                                                #hai likin time k bcht k liye hm django ka pre-defined User
                                                                #model use kr liya.
    pageName=models.CharField(max_length=90)
    pageCat=models.CharField(max_length=90)
    pagePubDate=models.DateField()

#abb aap k admin pannel mei jitne bhi users honge wo sb 1 se xyada page nhi create kr sktay
#bcause of one-to-one relationship.


#agr kisi pageModel k record ko delete kiya, to sirf whi page record delete ho jaega.
#Lekin, agr kisi user ko delete kiya to uss k saath page record bhi automatically delete ho jae ga.
#AGR page record delete hone k saath aap us k user/creater ko bhi delet krna chahtay hai to:
#signals.py File:
# from .models import page
# from django.db.models.signals import post_delete
# from django.dispatch import receiver
# @receiver(post_delete,sender=page)
# def delUsrAlso(sender,instance,**kwargs):
#     print('user also deleted successfully_______________________________')
#     instance.usr.delete()

#Agr aap chahtei hai k jis user ka koi page pageModel k records mei mojod ho
#wo user delete na ho skay, to simply CASCADE ki jga PROTECT likhna hai.

#by default sb users page create kr sktay hai,agr aap chand specific users ko ye permission
#dena chahtay hai to:
#change code of Line 7 to :usr=models.OneToOneField(User,on_delete=models.CASCADE,
# primary_key=True,limit_choices_to={'is_staff':True})
#ab jiss user ka admin pannel mei Staff Status checked hoga sirf wo page/pageModel ka record save kr skay ga.
```

```
#admins.py file:
# from django.contrib import admi
# from .models import page
# @admin.register(page)
# class pageAdmin(admin.ModelAdmi
#list_display=['pageName','pageCa
```

One to One Relationship

One to One Relationship - To define a one-to-one relationship, use **OneToOneField**. You use it just like any other Field type by including it as a class attribute of your model.

OneToOneField requires a positional argument, the class to which the model is related.

Syntax:- OneToOneField(to, on_delete, parent_link=False, **options)

pehle 2 parameters ka dena must hai.which r to and on_delete

Where,

to - The class to which the model is related.

on_delete - When an object referenced by a ForeignKey is deleted, Django will emulate the behavior of the SQL constraint specified by the on_delete argument. on_delete doesn't create an SQL constraint in the database. mtlb jb user delete ho jae to uss ka page bhi automatically delete ho jae uss k liye hm ye use krtay hai.

limit_choices_to - Sets a limit to the available choices for this field when this field is rendered using a ModelForm or the admin (by default, all objects in the queryset are available to choose). Either a dictionary, a Q object, or a callable returning a dictionary or Q object can be used.

konsa user page wghaira craete kr skta ouur kon nhi ye hm iss me likhty mai.

iss trah simple inheritance se likes wale model mei 1 foreign key bnti hai jiss se wo apne parent model se link hota hai
agr uss name ko agr aap customize krna chahtei ha to:
example: models.py file:

```
from django.db import models
from django.contrib.auth.models import User
# Create your models here.
class Page(models.Model):
 user = models.OneToOneField(User, on_delete=models.CASCADE,
 primary_key=True)
 page_name = models.CharField(max_length=70)
 page_cat = models.CharField(max_length=70)
 page_publish_date = models.DateField()

class Like(Page):
 panna = models.OneToOneField(Page, on_delete=models.CASCADE,
 primary_key=True, parent_link=True)
 likes = models.IntegerField()
```

parent_link - When True and used in a model which inherits from another concrete model, indicates that this field should be used as the link back to the parent class, rather than the extra OneToOneField which would normally be implicitly created by subclassing.

on_delete

on_delete - When an object referenced by a ForeignKey is deleted, Django will emulate the behavior of the SQL constraint specified by the on_delete argument. on_delete doesn't create an SQL constraint in the database.

The possible values for on_delete are found in django.db.models:

- CASCADE - Cascade deletes. Django emulates the behavior of the SQL constraint ON DELETE CASCADE and also deletes the object containing the ForeignKey.
- PROTECT - Prevent deletion of the referenced object by raising ProtectedError, a subclass of django.db.IntegrityError.

agr aap chahte hai k user delete ho jae lekin uss k posts/pages/records baqi reh jae to simply .CASCADE ki jgha SET_NULL,null=True likhna hai.

- SET_NULL - Set the ForeignKey null; this is only possible if null is True.
- SET_DEFAULT - Set the ForeignKey to its default value; a default for the ForeignKey must be set.
- SET() - Set the ForeignKey to the value passed to SET(), or if a callable is passed in, the result of calling it.
- DO_NOTHING - Take no action. If your database backend enforces referential integrity, this will cause an IntegrityError unless you manually add an SQL ON DELETE constraint to the database field.

```
class User(models.Model):
 user_name = models.CharField(max_length=70)
 password = models.CharField(max_length=70)

class Page(models.Model):
 user = models.OneToOneField(User, on_delete=models.CASCADE)
 page_name = models.CharField(max_length=70)
 page_cat = models.CharField(max_length=70)
 page_publish_date = models.DateField()
```

yha pr PROTECT,SET_NULL wghaira likh sktay hai.

jitne bhi methods hai jo iss page mei ya kisi ouur ye saaray relationships me use ho sktay hai. one-to-one,one-to-many, ya many-to-many.

One to One Relationship

related_name - The name to use for the relation from the related object back to this one. It's also the default value for related_query_name (the name to use for the reverse filter name from the target model).

jb hmara relationship bn jae ga our query methods ka use krayngai data ko fetch krnay k liye to uss waqt hme model class k name ko use krna hota hai to agr uss name k ilawa koi ouur name aap use krna chahtay hai to iss mehtod ka use kijiye.

If you'd prefer Django not to create a backwards relation, set related_name to '+' or end it with '+'.

related_query_name - The name to use for the reverse filter name from the target model. It defaults to the value of related_name or default_related_name if set, otherwise it defaults to the name of the model.

to_field - The field on the related object that the relation is to. By default, Django uses the primary key of the related object. If you reference a different field, that field must have unique=True.

swappable - Controls the migration framework's reaction if this ForeignKey is pointing at a swappable model. If it is True - the default - then if the ForeignKey is pointing at a model which matches the current value of settings.AUTH_USER_MODEL (or another swappable model setting) the relationship will be stored in the migration using a reference to the setting, not to the model directly.

db_constraint - Controls whether or not a constraint should be created in the database for this foreign key. The default is True, and that's almost certainly what you want; setting this to False can be very bad for data integrity. That said, here are some scenarios where you might want to do this:

You have legacy data that is not valid.

You're sharding your database.

If this is set to False, accessing a related object that doesn't exist will raise its DoesNotExist exception.

Many to One Relationship

When one or more row of table B can be linked to one row of table A.

jb 1 user bht saray posts kr skta hia to
one-to-many relationship kehlata hai.

Many to One Relationship

Post

ID	Post Title	Post Cat	Post Publish Date	User_id
1	Title 1	django	12-12-2000	1
2	Title 2	django	11-09-2003	1
3	Title 3	python	07-01-2001	2

User

ID	Username	Password
1	Rahul	rahul12
2	Sonam	sonam12
3	Kunal	kunal12

Many to One Relationship

```
class User(models.Model):
 user_name = models.CharField(max_length=70)
 password = models.CharField(max_length=70)

class Post(models.Model):
 user = models.ForeignKey(User, on_delete=models.CASCADE)
 post_title = models.CharField(max_length=70)
 post_cat = models.CharField(max_length=70)
 post_publish_date = models.DateField()
```

jb user delete hoga to uss k saray posts bhi delete hojaengai.

Many to One Relationship

Many to One Relationship - To define a many-to-one relationship, use **ForeignKey**. You use it just like any other Field type: by including it as a class attribute of your model.

A many-to-one relationship requires two positional arguments: the class to which the model is related and the on_delete option.

Syntax:- ForeignKey(to, on_delete, **options)

Many to Many Relationship

When one row of table A can be linked to one or more rows of table B, and vice-versa.

Many to Many Relationship

User

ID	Username	Password
1	Rahul	rahul12
2	Sonam	sonam12
3	Kunal	kunal12

Song

ID	Song Name	Song Duration
1	Tum hi ho	5
2	Kuch Kuch	7
3	Dil to hai dil	8

song_user

ID	Song_id	User_id
1	1	1
2	1	2
3	2	3
4	2	1

Example:-

```
class User(models.Model):
 user_name = models.CharField(max_length=70)
 password = models.CharField(max_length=70)


class Song(models.Model):
 user = models.ManyToManyField(User)
 song_name = models.CharField(max_length=70)
 song_duration = models.IntegerField()
```

by many-to-many default jb hm kisi users ka record bnatay hai to admin pannel me wo nhi dikhta k kon kon se user/creaters hai iss record k to agr wo daikhna hai to,
models.py: Song model class k sb se last mei ye function likhna hai:

```
 def funName(self):
        return ",".join([str(p) for p in self.user.all()])
```

Our f(x) k name ko admin.py file k list_display object k list me likh dena hai.

Many to Many Relationship

Many to Many Relationships - To define a many-to-many relationship, use ManyToManyField. You use it just like any other Field type: by including it as a class attribute of your model.

ManyToManyField requires a positional argument: the class to which the model is related.

Syntax:- ManyToManyField(to, **options)

MODELS RELATIONSHIP SMALL PROJECT:

```
#models.py File:
from django.db import models
from django.contrib.auth.models import User
# Create your models here.
class page(models.Model):
    usr=models.OneToOneField(User,on_delete=models.CASCADE,primary_key=True)
    pageName=models.CharField(max_length=90)
    pageCategory=models.CharField(max_length=90)
    pagePublishDate=models.DateField()

class post(models.Model):
    usr=models.ForeignKey(User,on_delete=models.CASCADE)
    postTitle=models.CharField(max_length=90)
    postCategory=models.CharField(max_length=90)
    postPublishDate=models.DateField()

class song(models.Model):
    usr=models.ManyToManyField(User)
    songName=models.CharField(max_length=90)
    songDuration=models.IntegerField()
    def writtenBy(self):
        return ",".join([str(p) for p in self.usr.all()])

#views.py File:
#allObj=User.objects.filter(page__page_cat='Programming')
#uuper wale line ka mtlb: unn users ka data filter kr lo jinho nei 'page' name k model mei koi page bnaya ho ouur uss
# k page_cat attribute k field mei 'Programming' likha ho.
#Note:agr aap page ki bjae koi ouur name use krna chahtei hai jese for e.g:
#allObj=User.objects.filter(myPage__page_cat='Programming') to line
#5:     usr=models.OneToOneField(User,on_delete=models.CASCADE,primary_key=True,related_name='myPage')


#Note: baqi ka kaam aasaan hai btt lengthy hai iss liye in-complete hai.
```

Class Based Views in django: Project(37)

Type of Views

- Function Based View

- Class Based View

Class Based View

Class-based views provide an alternative way to implement views as Python objects instead of functions.

They do not replace function-based views.

- Base Class-Based Views / Base View
- Generic Class-Based Views / Generic View

Advantages:-

- Organization of code related to specific HTTP methods (GET, POST, etc.) can be addressed by separate methods instead of conditional branching.
- Object oriented techniques such as mixins (multiple inheritance) can be used to factor code into reusable components.

Base Class-Based View

Base class-based views can be thought of as parent views, which can be used by themselves or inherited from. They may not provide all the capabilities required for projects, in which case there are Mixins which extend what base views can do.

- View
- TemplateView
- RedirectView

```
from django.http import HttpResponse
from django.shortcuts import render
#Urls.py:     path('',views.myView.as_view(),name='classBasedView')
from django.views import View
class myView(View):
    def get(self,request):
        return HttpResponse('classe based view')

#Example no 2:
# class myView(View):
#     name='Ali'
#     def get(self,request):
#         return HttpResponse(self.name)



#Example no: 3
# path('',views.myView.as_view(name='Ali'),name='classBasedView')
# class myView(View):
    #name='Khan' iss ko name='' bhi likh sktay hai lekin likhna must hai.
    # def get(self,request):
        # return HttpResponse(self.name)
#Output: Ali


#Example No 4:
#path('any/',views.subMyView.as_view(),name='any')
# from django.views import View
# class myView(View):
#     n1=100
#     def get(self,request):
#         return HttpResponse(self.n1)

# class subMyView(myView):
#     def get(self,request):
#         return HttpResponse(self.n1)                          #Output: 100
```

```
from django.http import HttpRequest, HttpResponse
from django.shortcuts import render
from django.views import View
# Create your views here.

#Example No 1:
#urls.py File:    # path('2/',myApp2.myView2.as_view(),name='classBasedView2'),
# class myView2(View):
#     def get(self,request):
#         return render(request,'home.html')

#####################################################
#Example No 2:
#urls.py File:     # path('3/',myApp2.myViewDic.as_view(),name='classBasedView2')
# class myViewDic(View):
#     def get(self,request):
#         context={'name':'he is ali'}
#         return render(request,'home.html',context)

#######################################################
```

```
from django.http import HttpRequest, HttpResponse
from django.shortcuts import render
from django.views import View

#urls.py File:    path('4/',myApp2.nameFormClass.as_view(),name='nameFormClass')
#forms.py File:
# from django import forms
# class nameForm(forms.Form):
#     name=forms.CharField(max_length=90)


# from .forms import nameForm
# class nameFormClass(View):
#     def get(self,request):
#         fm=nameForm()
#         return render(request,'home.html',{'form':fm})

#     def post(self,request):
#         if request.method=='POST':
#             fm=nameForm(request.POST)
#             if fm.is_valid():
#                 print(fm.cleaned_data['name'])
#                 return HttpResponse('form submitted!!')


#.html File:
#<form action="" method="post">{% csrf_token %}
#     <h4>{{form.as_p}}</h4>
#     <input type="submit" value="Submit">
# </form>

#################################################################
```

```
#Example 4:
#urls.py File:    path('',myApp2.egFive,{'templateName':'home.html'},name='egFive')
# def egFive(request,templateName):
    # return render(request,templateName)
#Output: it's output will be templates/home.html file

##############################################################
#Example 5:
#urls.py File:    path('egFive/',myApp2.egFive,{'templateName':'home.html'},name='egFive'),
# path('egFive222/',myApp2.egFive,{'templateName':'about.html'},name='egFive')

#views.py File:
# def egFive(request,templateName):
    # return render(request,templateName)
#here we r using one f(x) but if we hit egFive/ it's output will be templates/home.html
#and if we hit egFive222/ it's output will be templates/about.html


#Here's above thing in Class Based View:

#urls.py File: path('egFive222/',myApp2.aboveThing.as_view(templateName='home.html'),name='egFive'),
             # path('egFive1222/',myApp2.aboveThing.as_view(templateName='about.html'),name='egFive'),
#views.py File:
class aboveThing(View):
    templateName=''
    def get(self,request):
        return render(request,self.templateName)
##############################################################
```

View

views.py

Function Based View:-

```
from django.http import HttpResponse
def myview(request):
 return HttpResponse('<h1>Function Based View</h1>')
```

Class Based View:-

```
from django.views import View
class MyView(View):
 def get(self, request):
  return HttpResponse('<h1>Class Based View</h1>')
```

urls.py

Function Based View:-

```
from django.urls import path
from school import views
urlpatterns = [      path('func/', views.myview, name='func'),     ]
```

Class Based View:-

```
from django.urls import path
from school import views
urlpatterns = [
            path('cl/', views.MyView.as_view(), name='cl'),
]
```

View

django.views.generic.base.View

The master class-based base view. All other class-based views inherit from this base class. It isn't strictly a generic view and thus can also be imported from django.views.

C:\Users\GS\AppData\Local\Programs\Python\Python38-32\Lib\site-packages\django\views\generic

Attribute:-

http_method_names = ['get', 'post', 'put', 'patch', 'delete', 'head', 'options', 'trace']

The list of HTTP method names that this view will accept.

View

Methods:-

setup(self, request, *args, **kwargs) - It initializes view instance attributes: self.request, self.args, and self.kwargs prior to dispatch()

dispatch(self, request, *args, **kwargs) - The view part of the view – the method that accepts a request argument plus arguments, and returns a HTTP response.

The default implementation will inspect the HTTP method and attempt to delegate to a method that matches the HTTP method; a GET will be delegated to get(), a POST to post(), and so on.

By default, a HEAD request will be delegated to get(). If you need to handle HEAD requests in a different way than GET, you can override the head() method.

http_method_not_allowed(self, request, *args, **kwargs) - If the view was called with a HTTP method it doesn't support, this method is called instead.

The default implementation returns HttpResponseNotAllowed with a list of allowed methods in plain text.

options(self, request, *args, **kwargs) – It handles responding to requests for the OPTIONS HTTP verb. Returns a response with the Allow header containing a list of the view's allowed HTTP method names.

as_view(cls, **initkwargs) - It returns a callable view that takes a request and returns a response.

_allowed_methods(self)

TEMPLATEVIEW Project(38):

```
Example No 1:                                   Note: this "template_name" is compulsory.
urls.py File:     path('',views.TemplateView.as_view(template_name='home.html'),name='templateView')
views.py File:    from django.views.generic.base import TemplateView
#Output : home.html file will be it's output.

##########################################################################################

Example No 2:
urls.py File:     path('',views.homeTemplateView.as_view(),name='templateView')
views.py File:
from django.views.generic.base import TemplateView
class homeTemplateView(TemplateView):
    template_name='home.html'
Output: it's output will be home.html file.

##########################################################################################

Example No 3:
urls.py File: path('',views.homeTemplateView.as_view(),name='templateView')
views.py File:
from django.views.generic.base import TemplateView
class homeTemplateView(TemplateView):
    template_name='home.html'
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["name"] = "ali"
        context["roll"] = 101
Note: uper 2 lines ki bjae :   context={'name':'ali','roll':101} bhi lik sktay hai.
        return context
#now u can use these(name and roll) dictionary keys in ur templates/home.html file.

##########################################################################################

Example No 4:
urls.py File: path('',views.homeTemplateView.as_view(extra_context={'course':'Python'}),name='templateView')
views.py File:
from django.views.generic.base import TemplateView
class homeTemplateView(TemplateView):
    template_name='home.html'
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["name"] = "ali"                 Note:agr yha context={'name':'ali','roll':101} iss trah likhengai
        context["roll"] = 101                   to phir urls.py file mei extra_context kaam nhi krayga.
        return context
#now u can use these(name,roll and course) dictionary keys in ur templates/home.html file.

##########################################################################################

Example No 5:
urls.py File:     path('home/<int:id>',views.homeTemplateView.as_view(),name='templateView')
views.py File:
class homeTemplateView(TemplateView):
    template_name='home.html'
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["name"] = "ali"
        context["roll"] = 101
        print(f'{kwargs}') #**kwargs mei url ka data store hota hai. it's
        #output in terminal : {'id': 2} #2 is a value we have entered in url
        return context          #it can be any nmbr.
#Note: "id" ko(jo hm ne url path mei dei hai) hm apne template/home file mei as a dictionary key use kr sktay hai, jese
#for example: <h3>your id {{id}}</h3>

##########################################################################################
```

TemplateView

django.views.generic.base.TemplateView

It renders a given template, with the context containing parameters captured in the URL.

This view inherits methods and attributes from the following views:

- django.views.generic.base.TemplateResponseMixin
- django.views.generic.base.ContextMixin
- django.views.generic.base.View

class TemplateView(TemplateResponseMixin, ContextMixin, View):

TemplateResponseMixin

It provides a mechanism to construct a TemplateResponse, given suitable context. The template to use is configurable and can be further customized by subclasses.

Attributes:-

template_name - The full name of a template to use as defined by a string. Not defining a template_name will raise a django.core.exceptions.ImproperlyConfigured exception.

template_engine - The NAME of a template engine to use for loading the template. template_engine is passed as the using keyword argument to response_class. Default is None, which tells Django to search for the template in all configured engines.

response_class - The response class to be returned by render_to_response method. Default is TemplateResponse. The template and context of TemplateResponse instances can be altered later (e.g. in template response middleware).

If you need custom template loading or custom context object instantiation, create a TemplateResponse subclass and assign it to response_class.

content_type - The content type to use for the response. content_type is passed as a keyword argument to response_class. Default is None – meaning that Django uses 'text/html'.

Methods:-

render_to_response(context, **response_kwargs)- It returns a self.response_class instance.

If any keyword arguments are provided, they will be passed to the constructor of the response class.

Calls get_template_names() to obtain the list of template names that will be searched looking for an existent template.

get_template_names() - It returns a list of template names to search for when rendering the template. The first template that is found will be used.

If template_name is specified, the default implementation will return a list containing template_name (if it is specified).

ContextMixin

A default context mixin that passes the keyword arguments received by get_context_data() as the template context.

Attribute:-

extra_context - A dictionary to include in the context. This is a convenient way of specifying some context in as_view().

Method:-

get_context_data(**kwargs)- It returns a dictionary representing the template context. The keyword arguments provided will make up the returned context.

RedirectView

django.views.generic.base.RedirectView

It redirects to a given URL.

The given URL may contain dictionary-style string formatting, which will be interpolated against the parameters captured in the URL. Because keyword interpolation is always done (even if no arguments are passed in), any "%" characters in the URL must be written as "%%" so that Python will convert them to a single percent sign on output.

If the given URL is None, Django will return an HttpResponseGone (410).

This view inherits methods and attributes from the following view:

- django.views.generic.base.View

```
REDIRECT VIEW : Project No:39

Example No 1:
urls.py File:
path('',views.TemplateView.as_view(template_name='home.html')),
path('home/',views.RedirectView.as_view(url='/'),name='home'),#abb agr hm home/ url dengai to blank url path
                                                              #(jo iss uper line mei hai) pe redirect ho jaengai.
path('learn/',views.RedirectView.as_view(url='https://www.youtube.com')) #iss trah bi lik sktay hai.

views.py File: from django.views.generic.base import TemplateView,RedirectView

##################################################################################################

Example No 2:
urls.py File:     path('',views.youTubeRedirec.as_view())
views.py File:
from django.views.generic.base import RedirectView
class youTubeRedirec(RedirectView):
    url='https://www.youtube.com'

##################################################################################################

Example No 3:
urls.py File:
path('home/',views.TemplateView.as_view(template_name='home.html'),name='home'),
path('',views.RedirectView.as_view(pattern_name='home')) #abb jb bhi blank url path hit
                                                         #hoga to home se redirect ho k url path bhi change ho
views.py File: from django.views.generic.base import TemplateView,RedirectView

##################################################################################################
```

```
Example No 4:
urls.py File:     path('<int:roll>/',views.TemplateView.as_view(template_name='home.html'),name='rollName')

views.py File:
from django.views.generic.base import TemplateView,RedirectView
class rollRedirect(RedirectView):
    pattern_name='rollName' #agr iss ki jgha url='/' likhengai to kaam nhi krayga.

home.html File: <h1>home: {{roll}}</h1>

##########################################################################################################

Example No 5:
urls.py File:     path('<int:roll>/',views.TemplateView.as_view(template_name='home.html'),name='rollName')
views.py File:
class rollRedirect(RedirectView):
    pattern_name='rollName'
    query_string=True #bydefault fale rehata hai agr true krangai to url mei aap jo bhi
                      #string wghaira deingai(e.g: home/<slug:post>) wo wha he rhe ga yani k ghaib nhi hoga.

##########################################################################################################

#agr kisi method ko smjna hai to simply likh k uss k arguments ko terminal mei print kr k
#uss k f(x) wghaira ko smjha ja skta hai.
#Jesei for e.g:
class ....(..):
    ....
    def get_redirect_url(self, *args, **kwargs):
        kwargs['roll']=100 #user jo bhi (home/ 1,2 or 3 or any nmbr dega) to uss ko 100 hi dikeiga.
        print(f'Kwargs: {kwargs}')
        return super().get_redirect_url(*args, **kwargs)

##########################################################################################################
```

RedirectView

Attributes:-

url - The URL to redirect to, as a string. Or None to raise a 410 (Gone) HTTP error.

pattern_name - The name of the URL pattern to redirect to. Reversing will be done using the same args and kwargs as are passed in for this view.

permanent - Whether the redirect should be permanent. The only difference here is the HTTP status code returned. If True, then the redirect will use status code 301. If False, then the redirect will use status code 302. By default, permanent is False.

query_string - Whether to pass along the GET query string to the new location. If True, then the query string is appended to the URL. If False, then the query string is discarded. By default, query_string is False.

Methods:-

get_redirect_url(*args, **kwargs) - It constructs the target URL for redirection.

The args and kwargs arguments are positional and/or keyword arguments captured from the URL pattern, respectively.

The default implementation uses url as a starting string and performs expansion of % named parameters in that string using the named groups captured in the URL.

If url is not set, get_redirect_url() tries to reverse the pattern_name using what was captured in the URL (both named and unnamed groups are used).

If requested by query_string, it will also append the query string to the generated URL. Subclasses may implement any behavior they wish, as long as the method returns a redirect-ready URL string.

RedirectView

views.py

```
from django.shortcuts import render
from django.views.generic.base import RedirectView, TemplateView

class GeekyShowsRedirectView(RedirectView):
 url='https://geekyshows.com'

class GeekRedirectView(RedirectView):
 pattern_name = 'mindex'
 permanent = False
 query_string = False

 def get_redirect_url(self, *args, **kwargs):
  print(kwargs)
  kwargs['pk'] = 16
  return super().get_redirect_url(*args, **kwargs)
```

urls.py

```
from django.urls import path
from django.views.generic.base import RedirectView, TemplateView
from school import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.TemplateView.as_view(template_name='school/home.html'), name='blankindex'),
    path('home/', views.RedirectView.as_view(url='/'), name='home'),
    path('index/', views.RedirectView.as_view(pattern_name='home'), name='index'),
    path('geekyshows/', views.RedirectView.as_view(url='https://geekyshows.com'), name='go-to-
geekyshows'),
    path('geekyshows/', views.GeekyShowsRedirectView.as_view(), name='go-to-geekyshows'),
    path('home/<int:pk>/', views.GeekRedirectView.as_view(), name='geek'),
    path('<int:pk>/', views.TemplateView.as_view(template_name='school/home.html'), name='mindex'),
    # path('home/<slug:post>/', views.GeekRedirectView.as_view(), name='geek'),
    # path('<slug:post>/', views.TemplateView.as_view(template_name='school/home.html'), name='index'),
]
```

Base ClassBasedView CRUD Project(40):

CLASS BASED VIEW CRUD Project: Project(40)
NOTE: This project is same as 14.CRUD Project but is in class based view.
I have edited only views.py and urls.py file of project 14.CRUD Project.

urls.py File:

```
urlpatterns = [
    path('',views.base.as_view(),name='base'),
    path('/update/<int:stdId>',views.update.as_view(),name='update'),
    path('deleted/<int:stdId>/',views.delete.as_view(),name='delete')
]
```

views.py File:

```
from django.shortcuts import render
from .forms import addStdForm
from .models import addStd
from django.http import HttpResponseRedirect
# Create your views here.
from django.views.generic import TemplateView,RedirectView
from django.views import View #iss ko uper wale path se bhi import kr sktay hai.
class base(TemplateView):
    template_name='formApp/base.html'
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        form=addStdForm()
        allDbData=addStd.objects.all()
        context={'form':form,'allDbData':allDbData}
        return context
    def post(self,request):
        if request.method=='POST':
            form=addStdForm(request.POST)
            if form.is_valid():
                form.save()
                return HttpResponseRedirect('/')
```

```
class update(View):
    def get(self,request,stdId):
        upd=addStd.objects.get(pk=stdId)
        newUpd=addStdForm(instance=upd)
        return render(request,'formApp/update.html',{'newUpd':newUpd})
    def post(self,request,stdId):
        if request.method=='POST':
            upd=addStd.objects.get(pk=stdId)
            newUpd=addStdForm(request.POST,instance=upd)
            if newUpd.is_valid():
                newUpd.save()
                return HttpResponseRedirect('/')


class delete(RedirectView):
    url='/'
    def get_redirect_url(self, *args, **kwargs):
        print(f'id:{kwargs}')
        dlt=kwargs['stdId']
        addStd.objects.get(pk=dlt).delete()
        return super().get_redirect_url(*args, **kwargs)
```

As we have discussed that Class-Based Views are of two types :

1.Base Class-Based View(which's learning has been finished with above Project(i.e:Project No:40))

Which had three parts(View,TemplateView and RedirectView)

2.Generic Class-Based View

Now it's time to learn Generic Class-Based View:

Generic Class Based View

Django's generic views are built off of those base views, and were developed as a shortcut for common usage patterns such as displaying the details of an object.

They take certain common idioms and patterns found in view development and abstract them so that you can quickly write common views of data without having to repeat yourself.

Most generic views require the queryset key, which is a QuerySet instance.

CRUD se related jo common tasks hotai hai unn k liye kuch pre-defined keywords/attributes deta hai jinn ka use kr k hm apna kaam jldi ouur km code me kr sktay hai.

- Display View – ListView, DetailView
- Editing View – FormView, CreateView, UpdateView, DeleteView
- Date Views – ArchiveIndexView, YearArchiveView, MonthArchiveView, WeekArchiveView, DayArchiveView, TodayArchiveView, DateDetailView

Generic Display View

The two following generic class-based views are designed to display data.

- ListView
- DetailView

ListView

django.views.generic.list.ListView

A page representing a list of objects.

While this view is executing, self.object_list will contain the list of objects (usually, but not necessarily a queryset) that the view is operating upon.

This view inherits methods and attributes from the following views:

- django.views.generic.list.MultipleObjectTemplateResponseMixin
- django.views.generic.base.TemplateResponseMixin
- django.views.generic.list.BaseListView
- django.views.generic.list.MultipleObjectMixin
- django.views.generic.base.View

ListView with Default Template and Context

models.py File:
```
from django.db import models
class Student(models.Model):
    name=models.CharField(max_length=90)
    roll=models.IntegerField()
```

views.py

```
from django.views.generic.list import ListView
from .models import Student
class StudentListView(ListView):
        model = Student
```

Now our template file name must be modelclassame_list.html which is student_list.html here, and it's path must be templates/urAppName/modelclassname_list.html.

.html File:
```
<body>
  {%for std in student_list%}
    {{std.name}}
    {{std.roll}}
  {%endfor%}
</body>
```

Note: here our dictionary key variable name must be as same as our template file name which is student_list here.

we can also write object_list instead of student_list , we only have these two options to use default context key.

urls.py

```
urlpatterns = [  path('student/', views.StudentListView.as_view(), name='student'),  ]
```

Default Template

Syntax:- AppName/ModelClassName_list.html

Example:- school/student_list.html

Default Context

Syntax:- ModelClassName_list

Example:- student_list

We can also use **object_list**

ListView with Custom Template and Context

views.py

```
from django.views.generic.list import ListView
from .models import Student
class StudentListView(ListView):
        model = Student
        template_name = 'school/student.html'
        context_object_name = 'students'
```

template_name='appName/urTemplateName.html' #ab urTemplateName_list.html ouur simply urTemplateName.htmlye name jo aap ne yha likha hai dono ko use krsktay hai.
Note: custom ki priority high hai default templateName se. templateName ko change krnay se context name change nhi hota.

Custom Template Name

Custom Context Name

template_name_suffix='_get' #abb jha pr bhi _list likha hai wha pr _get likhengai. _get ki jgha ko our value bhi de sktay hai.

order=['roll'] #abb ye roll k hisab se order mei ho jaega.

urls.py

```
urlpatterns = [
        path('student/', views.StudentListView.as_view(), name='student'),
]
```

Note - *school/students.html*, *school/student_list.html* These both will work

```
#Example:
class std(ListView):
   model=Student
   template_name='myApp/school.html'
def get_template_names(self):
    if self.request.COOKIES['user']=='ali':
      template_name=myApp/ali.html
    else:
      template_name=self.template_name
   return [template_name]
#ab agr user cookies mei ali store hoga to ali.html file
display hoga nhi to school.html.
#Note: iss trah iss method for loop lga k bohat saray kaam
kiye ja sktay hai.jese for e.g: agr superuser ho to ye temp
-late dikha do nhi to ye wghaira wghaira.
```

MultipleObjectMixin

Attribute:-

allow_empty - A boolean specifying whether to display the page if no objects are available. If this is False and no objects are available, the view will raise a 404 instead of displaying an empty page. By default, this is True.

model - The model that this view will display data for. Specifying model = Student is effectively the same as specifying queryset = Student.objects.all(), where objects stands for Student's default manager.

queryset - A QuerySet that represents the objects. If provided, the value of queryset supersedes the value provided for model.

ordering - A string or list of strings specifying the ordering to apply to the queryset. Valid values are the same as those for order_by().

MultipleObjectMixin

```
e.g:
class stdList(ListView):
  model=Student
def get_queryset(self):
  return
Student.objects.filter(course='Pyhton'
#abb jiss ka course course field
python hoga sirf un ka data aagay
template file mei  pass  kr   lega.
```

Methods:-

get_queryset() - Get the list of items for this view. This must be an iterable and may be a queryset (in which queryset-specific behavior will be enabled).

get_ordering() - Returns a string (or iterable of strings) that defines the ordering that will be applied to the queryset.

Returns ordering by default.

paginate_queryset(queryset, page_size) - Returns a 4-tuple containing (paginator, page, object_list, is_paginated).

Constructed by paginating queryset into pages of size page_size. If the request contains a page argument, either as a captured URL argument or as a GET argument, object_list will correspond to the objects from that page.

get_context_object_name(object_list) - Return the context variable name that will be used to contain the list of data that this view is manipulating. If object_list is a queryset of Django objects and context_object_name is not set, the context name will be the model_name of the model that the queryset is composed from, with postfix '_list' appended. For example, the model Article would have a context object named article_list.

```
def get_context_name(self,*args,**kwargs):
  context=super().get_context_data(*args,**kwargs)
  context['newStds']=Student.objects.all()
  return context
#ab hm newStds ko as a variable apne template file
mei use kr sktay hai.
```

get_context_data(**kwargs) - Returns context data for displaying the list of objects.

Context

object_list: The list of objects that this view is displaying. If context_object_name is specified, that variable will also be set in the context, with the same value as object_list.

is_paginated: A boolean representing whether the results are paginated. Specifically, this is set to False if no page size has been specified, or if the available objects do not span multiple pages.

paginator: An instance of django.core.paginator.Paginator. If the page is not paginated, this context variable will be None.

page_obj: An instance of django.core.paginator.Page. If the page is not paginated, this context variable will be None.

DetailView with Default Template & Context

jab kisi object ka detail hme chahiye hota hai to DetailView ka use krtay hai.

views.py

```
from django.views.generic.detail import DetailView
from .models import Student
class StudentDetailView(DetailView):
        model = Student
```

.html file:
```
<body>
{{student.name}}
{{student.roll}}
{{student.class}}
</body>
```

#aap jo bhi pk dogai url mei sirf ja record db se show hoga.

urls.py

```
urlpatterns = [    path('student/<int:pk>', views.StudentDetailView.as_view(), name='student'),    ]
```

Default Template

Syntax:- AppName/ModelClassName_detail.html

Example:- school/student_detail.html

Default Context

Syntax:- ModelClassName

Example:= student

DetailView with Custom Template & Context

template_name='appName/urTemplateName.html'
context_object_name='stu' #abb student ki jgha stu ko .html mei use kraingai. e.g: {{stu.name}}

views.py

```
from django.views.generic.detail import DetailView
from .models import Student
class StudentDetailView(DetailView):
        model = Student
        template_name = 'school/student.html'
        context_object_name = 'student'
```

Custom Template Name

Custom Context Name

pk_url_kwarg='id' #ab pk ki jgha id likh sktay hai.
Note: name aap jo bhi dei(jesei for e.g roll,id) uss se pk sirf rename hoga lega ye pk hei..

urls.py

```
urlpatterns = [
        path('student/<int:pk>', views.StudentDetailView.as_view(), name='student'),
]
```

views.py student_list.html urls.py

gs124 > gs124 > urls.py > ...

```python
"""
from django.contrib import admin
from django.urls import path
from school import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('student/', views.StudentListView.as_view(),
    name='studentlist'),
    path('student/<int:pk>', views.StudentDetailView.as_view(),
    name='studentdetail'),
]
```

views.py student_list.html urls.py

gs124 > school > views.py > ...

```python
from django.shortcuts import render
from .models import Student
from django.views.generic.detail import DetailView
from django.views.generic.list import ListView
# Create your views here.
class StudentDetailView(DetailView):
 model = Student
 template_name = 'school/student.html'

class StudentListView(ListView):
 model = Student
```

views.py | student_list.html × | urls.py

gs124 > school > templates > school > student_list.html

```
initial-scale=1.0">
<title>Document</title>
</head>
<body>
<h1>Student List</h1>
{% for stu in student_list %}
 <li><a href="{% url 'studentdetail' stu.id %}">{{stu.name}}</
 a></li> <br>
{% endfor %}
</body>
</html>
```

urls.py | views.py | student.html ●

gs124 > school > templates > school > student.html

```
 <title>Document</title>
</head>
<body>
 <h1>Student_detail Page</h1>
 {{student.id}}
 {{student.name}}
 {{student.roll}}
 {{student.course}}
</body>
</html>
```

views.py student_list.html urls.py

gs124 > gs124 > urls.py > ...

```python
"""
from django.contrib import admin
from django.urls import path
from school import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('student/', views.StudentListView.as_view(),
    name='studentlist'),
    path('student/<int:pk>', views.StudentDetailView.as_view(),
    name='studentdetail'),
]
```

views.py

gs125 > school > views.py > StudentDetailView

```python
from .models import Student
from django.views.generic.detail import DetailView
from django.views.generic.list import ListView
# Create your views here.
class StudentDetailView(DetailView):
 model = Student
 template_name = 'school/student.html'

 def get_context_data(self, *args, **kwargs):
  context = super().get_context_data(*args, **kwargs)
  context['all_student'] = self.model.objects.all()
  return context

class StudentListView(ListView):
```

```
model = Student
```

views.py | student.html ● | student_list.html

gs125 > school > templates > school > student.html

```
8  <body>
9   <h1>Student_detail Page</h1>
10  {{student.id}}
11  {{student.name}}
12  {{student.roll}}
13  {{student.course}}
14  <hr>
15   {% for stu in all_student %}
16    <li><a href="{% url 'studentdetail' stu.id %}">{{stu.name}}</
      a></li>
17   {% endfor %}
18 </body>
19 </html>
```

Generic Editing View

The following views are described on this page and provide a foundation for editing content:

- FormView
- CreateView
- UpdateView
- DeleteView

FormView

forms.py

```
from django import forms
class ContactForm(forms.Form):
 name = forms.CharField()
 email = forms.EmailField()
 msg = forms.CharField(widget=forms.Textarea)
```

urls.py

```
from school import views
urlpatterns = [
path('contact/', views.ContactFormView.as_view(),
name='contact'),
path('thankyou/', views.ThanksTemplateView.as_view(),
name='thankyou'),
```

views.py

```
from django.views.generic.edit import FormView
class ContactFormView(FormView):
 template_name = 'school/contact.html'
 initial = {'name': 'Mr.'}
 form_class = ContactForm
 success_url = '/thankyou/'
 def form_valid(self, form):
        print(form)
        print(form.cleaned_data['name'])
        return HttpResponse('Msg Sent')

class ThanksTemplateView(TemplateView):
 template_name = 'school/thankyou.html'
```

contact.html file:

```
<form ……metod..post..>
{{form.as_p}}
</form>
```

FormMixin

django.views.generic.edit.FormMixin

A mixin class that provides facilities for creating and displaying forms.

This view inherits methods and attributes from the following views:

- django.views.generic.base.ContextMixin

Attributes:-

initial - A dictionary containing initial data for the form.

form_class - The form class to instantiate.

success_url - The URL to redirect to when the form is successfully processed.

prefix - The prefix for the generated form.

Methods:-

get_initial() - Retrieve initial data for the form. By default, returns a copy of initial.

get_form_class() - Retrieve the form class to instantiate. By default form_class.

get_form(form_class=None) - Instantiate an instance of form_class using get_form_kwargs(). If form_class isn't provided get_form_class() will be used.

get_form_kwargs() - Build the keyword arguments required to instantiate the form.

The initial argument is set to get_initial(). If the request is a POST or PUT, the request data (request.POST and request.FILES) will also be provided.

get_prefix() - Determine the prefix for the generated form. Returns prefix by default.

get_success_url() - Determine the URL to redirect to when the form is successfully validated. Returns success_url by default.

form_valid(form) - Redirects to get_success_url().

form_invalid(form) - Renders a response, providing the invalid form as context.

ProcessFormView

django.views.generic.edit.ProcessFormView

A mixin that provides basic HTTP GET and POST workflow.

Extends

django.views.generic.base.View

Methods:-

get(request, *args, **kwargs) - Renders a response using a context created with get_context_data().

post(request, *args, **kwargs) - Constructs a form, checks the form for validity, and handles it accordingly.

put(*args, **kwargs) - The PUT action is also handled and passes all parameters through to post().

CreateView

iss se hme 1 form milta hai jiss me hm dataentry kr k db mei save kr sktay hai.

django.views.generic.edit.CreateView

A view that displays a form for creating an object, redisplaying the form with validation errors (if there are any) and saving the object.

models.py | views.py | thanks.html | urls.py | student_form.html

gs127 > school > views.py > StudentCreateView

```python
from django.shortcuts import render
from django.views.generic.edit import CreateView
from .models import Student
from django.views.generic.base import TemplateView
# Create your views here.

class StudentCreateView(CreateView):
 model = Student
 fields = ['name', 'email', 'password']
 success_url = '/create/'

class ThanksTemplteView(TemplateView):
 template_name = 'school/thanks.html'
```

iss project se hme 1 form display hoga jiss me hm entry kr k db me apna data save kr sktay hai.

models.py | views.py | urls.py | student_form.html

gs127 > school > templates > school > student_form.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width,
 initial-scale=1.0">
 <title>Document</title>
</head>
<body>
 <form action="" method="post">
  {% csrf_token %}
  {{form.as_p}}
  <input type="submit" value="Submit">
 </form>
```

models.py × views.py thanks.html urls.py student_form.html

gs127 > school > models.py > ...

```python
from django.db import models
from django.urls import reverse
# Create your models here.
class Student(models.Model):
 name = models.CharField(max_length=70)
 email = models.EmailField()
 password = models.CharField(max_length=70)

 def get_absolute_url(self):
  return reverse("thankyou")
```

iss trah idar bhi likh sktay hai. 'thankyou' url path ka name hai .

```python
class StudentCreateView(CreateView):
 model = Student
 fields = ['name', 'email', 'password']
 success_url = '/thanks/'

 def get_form(self):
  form = super().get_form()
  form.fields['name'].widget = forms.TextInput(attrs=
  {'class':'myclass'})
  return form
```

agr form mei class add krnay ho to 1 trika ye hai.

```python
from .forms import StudentCreationView
class StudentCreateView(CreateView):
 form_class = StudentForm
 template_name = 'school/student_form.html'
 success_url='/thanks/'
class ThanksTemplteView(TemplateView):
 template_name = 'school/thanks.html'
```

ye(ye ouur iss k aagay wala picture) dosra tarika hai apne form mei classes add krnay ka.

views.py urls.py models.py forms.py

gs128 > school > forms.py > StudentForm > Meta

```python
from django import forms
from .models import Student

class StudentForm(forms.ModelForm):
 class Meta:
  model = Student
  fields = ['name', 'email', 'password']
  widgets = {'name':forms.TextInput(attrs={'class':'myclass'}),
  'password':forms.PasswordInput(attrs={'class':'mypass'})}
```

Methods:-

get_form_class() - Retrieve the form class to instantiate. If form_class is provided, that class will be used. Otherwise, a ModelForm will be instantiated using the model associated with the queryset, or with the model, depending on which attribute is provided.

get_form_kwargs() - Add the current instance (self.object) to the standard get_form_kwargs().

get_success_url() - Determine the URL to redirect to when the form is successfully validated. Returns django.views.generic.edit.ModelFormMixin.success_url if it is provided; otherwise, attempts to use the get_absolute_url() of the object.

form_valid(form) - Saves the form instance, sets the current object for the view, and redirects to get_success_url().

form_invalid(form) - Renders a response, providing the invalid form as context.

UpdateView

django.views.generic.edit.UpdateView

A view that displays a form for editing an existing object, redisplaying the form with validation errors (if there are any) and saving changes to the object. This uses a form automatically generated from the object's model class (unless a form class is manually specified).

Attributes:-

template_name_suffix - The UpdateView page displayed to a GET request uses a template_name_suffix of '_form'.

object - When using UpdateView you have access to self.object, which is the object being updated.

```
19 urlpatterns = [
20     path('admin/', admin.site.urls),
21     path('create/', views.StudentCreateView.as_view(),
       name='stucreate'),
22     path('thanks/', views.ThanksTemplateView.as_view(),
       name='thankyou'),
23     path('update/<int:pk>', views.StudentUpdateView.as_view(),
       name='stuupdate'),
24 ]
```

views.py • thanks.html student_form.html urls.py

gs129 > school > views.py > ...

```
 7  model = Student
 8  fields = ['name', 'email', 'password']
 9  success_url = '/thanks/'
10
11 class ThanksTemplateView(TemplateView):
12  template_name = 'school/thanks.html'
13
14 class StudentUpdateView(UpdateView):
15  model = Student
16  fields = ['name', 'email', 'password']
17  success_url = '/thanks/'
```

```
class StudentCreateView(CreateView):
 form_class = StudentForm
 template_name = 'school/student_form.html'
 success_url='/thanks/
class StudentUpdateView(UpdateView):
 model = Student
 form_class = StudentForm
 template_name = 'school/student_form.html'
 success_url='/thanks/
```

this is to give classes to our fileds from .form/StudentForm.
This method is used when we use Form/ModelForm.

DeleteView

django.views.generic.edit.DeleteView

A view that displays a confirmation page and deletes an existing object. The given object will only be deleted if the request method is POST. If this view is fetched via GET, it will display a confirmation page that should contain a form that POSTs to the same URL.

views.py urls.py

gs131 > gs131 > urls.py > ...

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('create/', views.StudentCreateView.as_view(), name='stucreate'),
    path('thanks/', views.ThanksTemplateView.as_view(), name='thankyou'),
    path('update/<int:pk>', views.StudentUpdateView.as_view(), name='stuupdate'),
    path('thanksupdate/', views.ThanksUpdateTemplateView.as_view(), name='thankyouupdate'),
    path('delete/<int:pk>', views.StudentDeleteView.as_view(), name='studelete'),

]
```

views.py - DjangoCode - Visual Studio Code

```python
success_url = '/thanks/'

class StudentUpdateView(UpdateView):
 model = Student
 form_class = StudentForm
 template_name = 'school/student_form.html'
 success_url = '/thanksupdate/'

class StudentDeleteView(DeleteView):
  model = Student
  success_url = '/create/'



class ThanksTemplateView(TemplateView):
 template_name = 'school/thanks.html'
```

www.geekyshows.com

student_confirm_delete.html - DjangoCode - Visual Studio Code

```html
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width,
 initial-scale=1.0">
 <title>Document</title>
</head>
<body>
 <h1>Are You Sure ?</h1>
 <form action="" method="post">
  {% csrf_token %}
  <input type="submit" value="Delete">
  <a href="{% url 'stucreate' %}">Cancel</a>
 </form>
```

```
ete.html
[21/Jun/2020 22:33:41] "GET /delete/3 HTTP/1.1" 500 80893
```

www.geekyshows.com

Attribute:-

template_name_suffix - The DeleteView page displayed to a GET request uses a template_name_suffix of '_confirm_delete'.

success_url - The url to redirect to when the nominated object has been successfully deleted.

success_url may contain dictionary string formatting, which will be interpolated against the object's field attributes. For example, you could use success_url="/parent/{parent_id}/" to redirect to a URL composed out of the parent_id field on a model.

Methods:-

delete(request, *args, **kwargs) - Retrieves the target object and calls its delete() method, then redirects to the success URL.

get_success_url() - Returns the url to redirect to when the nominated object has been successfully deleted. Returns success_url by default.

Authentication Views

Django provides several views that you can use for handling login, logout, and password management.

These make use of the stock auth forms but you can pass in your own forms as well.

Django provides no default template for the authentication views.

You should create your own templates for the views you want to use.

urls.py

path('accounts/', include('django.contrib.auth.urls')),

These all url will be available:-

accounts/login/ [name='login']

accounts/logout/ [name='logout']

accounts/password_change/ [name='password_change']

accounts/password_change/done/ [name='password_change_done']

accounts/password_reset/ [name='password_reset']

accounts/password_reset/done/ [name='password_reset_done']

accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']

accounts/reset/done/ [name='password_reset_complete']

```
AUTHENTICATION_VIEWS WITH FUNCTION BASED VIEW:
urls.py File:
from django.contrib import admin
from django.urls import path,include
from myApp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/',include('django.contrib.auth.urls')),
    path('accounts/profile/',views.profile,name='profile'),
]

views.py File:

from django.shortcuts import render
from django.contrib.auth.decorators import login_required

@login_required  #ab user ko ye view access nhi hoga jb tk k wo login na kray.
def profile(request):   #iss tra @staff_member_required ouur permission_required bi hai.
    return render(request,'registration/profile.html')

templates/resgistration/
login.html File:

<body>
    <form action="" method="post">{% csrf_token %}
        {{form.as_p}}
        <input type="submit" value="Login">
        <a href="{%url 'password_reset'%}">reset </a>
    </form>
</body>

profile.html File:
<body>
    <h2>Welcome after login</h2> <hr>
    <a href="{%url 'logout'%}">Logout</a>
    <a href="{%url 'password_change'%}">change password</a>
</body>

settings.py File (at end of settings.py file):
#bydefautl login hone k baad registration/profile pe redirect hota hai agr uss ko customix krna hai
#to neichai LOGIN_REDIRECT_URL='iss mei apna path likhye'
# LOGIN_REDIRECT_URL='/'

#iss trah agr login ka path customix krna ho to for e.g:
# LOGIN_URL='/login/'

#password reset krnay pr user ko email send hota hai jo k local host/devolopment phase mei
kaam nhi krta lekin #uss ko terminal mei deikhne k liye hm ye neichei wala code likhtai hai.
EMAIL_BACKEND='django.core.mail.backends.console.EmailBackend'
```

login_required Decorator

login_required(redirect_field_name='next', login_url=None)

If the user is logged in, execute the view normally. The view code is free to assume the user is logged in.

If the user isn't logged in, redirect to settings.LOGIN_URL, passing the current absolute path in the query string. Example: /accounts/login/?next=/accounts/profile/

django.contrib.auth.decorators.login_required

Where,

redirect_field_name - If you would prefer to use a different name for this parameter, login_required() takes an optional redirect_field_name parameter. If you provide a value to redirect_field_name, you will most likely need to customize your login template as well, since the template context variable which stores the redirect path will use the value of redirect_field_name as its key rather than "next" (the default).

login_url - If you don't specify the login_url parameter, you'll need to ensure that the settings.LOGIN_URL and your login view are properly associated.

The settings.LOGIN_URL also accepts view function names and named URL patterns. This allows you to freely remap your login view within your URLconf without having to update the setting.

Example:-

```
from django.contrib.auth.decorators import login_required
@login_required
def profile(request):
 return render(request, 'registration/profile.html')
```

staff_member_required decorator

staff_member_required(redirect_field_name='next', login_url='admin:login')

This decorator is used on the admin views that require authorization. A view decorated with this function will having the following behavior:

- If the user is logged in, is a staff member (User.is_staff=True), and is active (User.is_active=True), execute the view normally.
- Otherwise, the request will be redirected to the URL specified by the login_url parameter, with the originally requested path in a query string variable specified by redirect_field_name.
  For example: /admin/login/?next=/profile/

Example:-

```
from django.contrib.admin.views.decorators import staff_member_required

@staff_member_required

def profile(request):

 return render(request, 'registration/profile.html')
```

permission_required Decorator

permission_required(perm, login_url=None, raise_exception=False)

It's a relatively common task to check whether a user has a particular permission. For that reason, Django provides a shortcut for that case: the permission_required() decorator.

Just like the has_perm() method, permission names take the form "<app label>.<permission codename>"

Example:-

```
from django.contrib.auth.decorators import permission_required

@permission_required('blog.can_add')

def profile(request):

 return render(request, 'registration/profile.html')
```

Decorating Class-Based View

iss k 2 trikai hia 1 urls.py file mei ouur dosra views.py file.

Decorating in urls.py or URLconf

The simplest way of decorating class-based views is to decorate the result of the as_view() method. The easiest place to do this is in the URLconf where you deploy your view:

```
from django.urls import path

from django.views.generic import TemplateView

from registration.views import ProfileTemplateView

from django.contrib.auth.decorators import login_required

urlpatterns = [

    path('dashboard/', login_required(TemplateView.as_view(template_name='blog/dash.html')), name='dash'),

    path('profile/', login_required(ProfileTemplateView.as_view(template_name='registration/profile.html')),
     name='profile'),

    path('blogpost/', permission_required('blog.can_add')(BlogPostView.as_view())),

]
```

```
AUTHENTICATION VIEW WITH CLASS-BASED VIEW:
from django.contrib import admin
from django.urls import path,include
from myApp import views as myApp
from django.contrib.auth.decorators import login_required
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/',include('django.contrib.auth.urls')),
    path('accounts/profile/',login_required(myApp.profile.as_view()),name='profile'),
    path('about/',login_required(myApp.about.as_view()),name='about')
]


views.py File:

from django.views.generic import TemplateView
class profile(TemplateView):
    template_name='registration/profile.html'
class about(TemplateView):
    template_name='registration/about.html'

templates/resgistration/
login.html File:
<body>
    <form action="" method="post">{% csrf_token %}
        {{form.as_p}}
        <input type="submit" value="Login">
        <a href="{%url 'password_reset'%}">reset </a>
    </form>
</body>

profile.html File:
<body>
    <h2>Welcome after login</h2> <hr>
    <a href="{%url 'logout'%}">Logout</a>
    <a href="{%url 'password_change'%}">change password</a>
```

```
about.html File:
<body>i am about</body>
</body>
```

Decorating in the Class

To decorate every instance of a class-based view, you need to decorate the class definition itself. To do this you apply the decorator to the dispatch() method of the class.

```
from django.utils.decorators import method_decorator
from django.views.generic import TemplateView
class ProfileTemplateView(TemplateView):
    template_name = 'registration/profile.html'

    @method_decorator(login_required)
    def dispatch(self, *args, **kwargs):
        return super().dispatch(*args, **kwargs)
```

classes mei decorators use krnay k 2 tarikay hai.
1 ye ouur dosra iss baad page wala.

Note: decorators ko import krna must hai.
login_required,is_staff_member wghaiira ko agr use krna hai.

Decorating in the Class

If you have a set of common decorators used in several places, you can define a list or tuple of decorators and use this instead of invoking method_decorator() multiple times.

```
@method_decorator(never_cache, name='dispatch')
@method_decorator(login_required, name='dispatch')
class ProfileTemplateView(TemplateView):
 template_name = 'registration/profile.html'
```

agr 1 se xyada decorators use krnay hai to iss trah list mei daal k neichei method decoreator mei likh lo.

```
decorators = [never_cache, login_required]
@method_decorator(decorators, name='dispatch')
class ProfileTemplateView(TemplateView):
 template_name = 'registration/profile.html'
```

The decorators will process a request in the order they are passed to the decorator. In the example, never_cache() will process the request before login_required().

method_decorator

The method_decorator decorator transforms a function decorator into a method decorator so that it can be used on an instance method.

A method on a class isn't quite the same as a standalone function, so you can't just apply a function decorator to the method you need to transform it into a method decorator first.

```
@method_decorator(*args, **kwargs)
```

Authentication Views

This is a list with all the views *django.contrib.auth* provides.

- LoginView
- LogoutView
- PasswordChangeView
- PasswordChangeDoneView
- PasswordResetView
- PasswordResetDoneView
- PasswordResetConfirmView
- PasswordResetCompleteView

jb hm authenticationView use krtay hai to
login/logout/resetpassword etc inn sb k backend pe
django ye Views use krtay hai.
Ab hm in views ko customix krna seekaingai.

```
A SMALL PROJECT JUST TO UNDERSTAND HOW TO CUSTOMIZE:
urls.py File:
from django.contrib import admin
from django.urls import path,include
from django.contrib.auth import views as authView
from myApp import views
from django.views.generic import TemplateView
urlpatterns = [                     #agr as_view(iss mei) kuch bhi nhi likhengai to ye default template
    path('admin/', admin.site.urls),    #path mangai ga.

    path('',authView.LoginView.as_view(template_name='registration/login.html'),name='login'),

    path('dashboard/',TemplateView.as_view(template_name='registration/dashboard.html'),name='dashboard'),

    path('logout/',authView.LogoutView.as_view(template_name='registration/myLogout.html'),name='logout'),

    path('passwordChange/',authView.PasswordChangeView.as_view(template_name='registration/changePassword.html')
    ,name='passwordChange'),

    path('passwordChangeDone/',authView.PasswordChangeDoneView.as_view(template_name='registration/dashboard.html')
    ,name='password_change_done') #by default ye iss name ka demand krta hai agr apna dena hai to iss se uper
    #wale path mei (.as_views(success_url='passwordChangeDone')) likhye ouur iss path mei apna name de dejiye.
]

#yha ki bjae agr views.py file mei ye attribute wghaira(template_name,authentication_form etc) likhne hai to:
# an example:
#from django.contrib.auth.views import LoginView
#from .forms import loginForm
#class myLoginView(LoginView):
#   template_name='registration/login.html'
#urls.py: from myApp import views as authView
# path('',authView.myLoginView.as_view(),name='login'),
#ISS TRAH BAQI(logout,passwordChange view ) wghaira k liye bhi kr sktay hai.
```

views.py File:

```
from django.views.generic import TemplateView
class profile(TemplateView):
    template_name='registration/profile.html'

class about(TemplateView):
    template_name='registration/about.html'
```

forms.py File:

```
#USE this code only if u want To give classess to ur login form for applying CSS:
from django import forms
from django.contrib.auth.forms import AuthenticationForm,UsernameField
from django.utils.translation import gettext,gettext_lazy as _
class loginForm(forms.Form):
    username=UsernameField(widget=forms.TextInput(attrs={'autofocus':True,'class':'myClass'}))
    password=forms.CharField(label=_("Password"),
    strip=False,
    widget=forms.PasswordInput(attrs={'autocomplete':'current-password','class':'myClass'}))

#login path will be like:
#from .forms import loginForm
#  path('',authView.LoginView.as_view(template_name='registration/login.html',
# authentication_form=loginForm),name='login'),
```

settings.py File:

```
LOGIN_REDIRECT_URL='/dashboard/'
```

#TEMPLATE FILES:

login.html File:

```
<body>
    <form action="" method="post">{% csrf_token %}
        {{form.as_p}}
        <input type="submit" value="Login">
    </form>
</body>
```

dashboard.html File:

```
<body>
    <h2>Dashboard</h2>
    <a href="{%url 'logout'%}">Logout</a>
    <br>
    <a href="{%url 'passwordChange'%}">Change ur password</a>
</body>
```

logout.html File:

```
<body>
    Logged Out Successfully !
    <a href="{%url 'login'%}">Login Again</a>
</body>
```

changePassword.html File:

```
<body>
    <form action="" method="post">{% csrf_token %}
        {{form.as_p}}
        <input type="submit" value="Change Password">
    </form>
</body>
```

about.html File:

```
<body>   <h2>About page</h2>    </body>
```

Auth Settings

Settings for django.contrib.auth.

AUTHENTICATION_BACKENDS

A list of authentication backend classes (as strings) to use when attempting to authenticate a user.

Default: ['django.contrib.auth.backends.ModelBackend']

AUTH_USER_MODEL

The model to use to represent a User.

Default: 'auth.User'

LOGIN_REDIRECT_URL

The URL or named URL pattern where requests are redirected after login when the LoginView doesn't get a next GET parameter.

Default: '/accounts/profile/'

LOGIN_URL

The URL or named URL pattern where requests are redirected for login when using the login_required() decorator, LoginRequiredMixin, or AccessMixin.

Default: '/accounts/login/'

LOGOUT_REDIRECT_URL

The URL or named URL pattern where requests are redirected after logout if LogoutView doesn't have a next_page attribute.

If None, no redirect will be performed and the logout view will be rendered.

Default: None

PASSWORD_RESET_TIMEOUT_DAYS

The minimum number of days a password reset link is valid for. Depending on when the link is generated, it will be valid for up to a day longer.

Used by the PasswordResetConfirmView.

Default: 3

PASSWORD_HASHERS

Default: [

'django.contrib.auth.hashers.PBKDF2PasswordHasher',

'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',

'django.contrib.auth.hashers.Argon2PasswordHasher',

'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',

]

AUTH_PASSWORD_VALIDATORS

The list of validators that are used to check the strength of user's passwords.

Default: [ ] (Empty list)

SQLite3

Open settings.py

```
DATABASE = {
        'default':{
                'ENGINE': 'django.db.backends.sqlite3',
                'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
        }
```

MySQL Configuration

by default django sqlite3 db ko use krta hai jo iss se pehle pictue mei dikhaya gya hai. To agr aap ne mySql db use krna to ye tarika hai.

- Install mysql in your system.
- You have to create your own database and user to config MySQL with Django.
- Open settings.py

```
DATABASE = {
    'default':{
        'ENGINE': 'django.db.backends.mysql',
        'NAME':  'database_name'
        'USER':  'user_name',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': port_number
    }
}
```

Optional

example:

```
DATABASE = {
    'default':{
        'ENGINE': 'django.db.backends.mysql',
        'NAME':  'mydb'
        'USER':  'root',
        'PASSWORD': '',
        'HOST': 'localhost',
        'PORT': 5555
    }
}
```

Oracle SQL Configuration

- Install Oracle in your system.
- You have to create your own database and user to config Oracle with Django.
- Open settings.py

```
DATABASE = {
    'default':{
        'ENGINE': 'django.db.backends.oracle',
        'NAME':  'database_name'
        'USER':  'user_name',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': port_number
    }
}
```

Postgres Configuration

- Install Postgres in your system.
- You have to create your own database and user to config Postgres with Django.
- Open settings.py

```
DATABASE = {
    'default':{
        'ENGINE': 'django.db.backends.postgres',
        'NAME':  'database_name'
        'USER':  'user_name',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': port_number
    }
}
```

Pagination

Using pagination we can split data to several pages, with Previous/Next links.

Previous 1 2 3 Next

Django provides a few classes that help you manage paginated data:-

- Paginator Class
- Page Class

Using Pagination

- Pagination with Function Based View
- Pagination with Class Based View

Previous 1 2 3 Next

Paginator WITH FUNCTION BASED VIEW: (project:43)

urls.py File:

```
from django.contrib import admin
from django.urls import path
from blog import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.home,name='home')
]
```

models.py File:

```
from django.db import models
class myPost(models.Model):
    title=models.CharField(max_length=90)
    desc=models.TextField()
    pubDate=models.DateField()
```

admin.py File:

```
from django.contrib import admin
from .models import myPost
# Register your models here.
@admin.register(myPost)
class myPostAdmin(admin.ModelAdmin):
    list_display=['id','title','desc','pubDate']
```

views.py File:

```
from django.shortcuts import render
from .models import myPost
from django.core.paginator import Paginator
```

```
def home(request):
    posts=myPost.objects.all().order_by('id') #agr .order_by() method nhi bi likhengai to kaam krayga.
    myPaginator=Paginator(posts,3)  #mtlb 1 page pe 3 items dikhanei hai. agr 3 k bad ,orphan=1 likheingai
    page_number=request.GET.get('page')  #to last page jo 1 item bch jaega to wo second last page pe shift hojayega.
    pageObject=myPaginator.get_page(page_number)
    print('Page Objects : ' ,pageObject)
    print('PageNumber: ',page_number)
    return render(request,'blog/home.html',{'page_obj':pageObject})
#Note: context mei 'page_obj' key likhna issi name se must hai. page_number bhi issi name se likhna must hai.


templates/blog/home.html File:
<html>
<body>
    <h1>Home Page</h1>
    {%for post in page_obj%}
     <h2>{{post.title}}</h2>
     <p>{{post.desc}}</p>
     <small>{{post.pubDate}}</small>
    {% endfor %}
    <span>{{page_obj.number}}</span>

<div>
    <span>
        {% if page_obj.has_previous %}
            <a href="?page={{page_obj.previous_page_number}}">Previous</a>
        {% endif %}

        <span>{{page_obj.number}}</span>

        {% if page_obj.has_next %}
            <a href="?page={{page_obj.next_page_number}}">Next</a>
        {% endif %}
    </span>
</div>  </body>  </html>
```

Paginator Class

class Paginator(object_list, per_page, orphans=0, allow_empty_first_page=True)

Where,

object_list – It takes tuple, list, QuerySet or other sliceable object with a count() or __len__() method. It is required.

per_page – The maximum number of items to include on a page, not including orphans. It is required.

orphans – Use this when you don't want to have a last page with very few items. If the last page would normally have a number of items less than or equal to orphans, then those items will be added to the previous page (which becomes the last page) instead of leaving the items on a page by themselves. orphans defaults to zero, which means pages are never combined and the last page may have one item. It is optional.

allow_empty_first_page - Whether or not the first page is allowed to be empty. If False and object_list is empty, then an EmptyPage error will be raised. It is optional.

Pagination Class Attributes

- count - The total number of objects, across all pages.
- num_pages - The total number of pages.
- page_range - A 1-based range iterator of page numbers, e.g. yielding [1, 2, 3, 4].

Pagination Class Methods

- get_page(number) – This method returns a Page object with the given 1-based index, while also handling out of range and invalid page numbers.
  If the page isn't a number, it returns the first page.
  If the page number is negative or greater than the number of pages, it returns the last page.
  Raises an EmptyPage exception only if you specify Paginator(..., allow_empty_first_page=False) and the object_list is empty.

- page(number) – This method returns a Page object with the given 1-based index.
  Raises InvalidPage if the given page number doesn't exist.

Page Class

class Page(object_list, number, paginator)

A page acts like a sequence of Page.object_list when using len() or iterating it directly.

Page Class Attributes

- object_list - The list of objects on this page.
- number - The 1-based page number for this page.
- paginator - The associated Paginator object.

Page Class Methods

- has_next() - It returns True if there's a next page.
- has_previous() - It returns True if there's a previous page.
- has_other_pages() - It returns True if there's a next or previous page.
- next_page_number() - It returns the next page number. Raises InvalidPage if next page doesn't exist.
- previous_page_number() - It returns the previous page number. Raises InvalidPage if previous page doesn't exist.
- start_index() - It returns the 1-based index of the first object on the page, relative to all of the objects in the paginator's list. For example, when paginating a list of 5 objects with 2 objects per page, the second page's start_index() would return 3.
- end_index() - It returns the 1-based index of the last object on the page, relative to all of the objects in the paginator's list. For example, when paginating a list of 5 objects with 2 objects per page, the second page's end_index() would return 4.

Paginator WITH CLASS BASED VIEW(project:43 in class based view):

urls.py File:

```
from django.contrib import admin
from django.urls import path
from blog import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.home,name='home')
]
```

models.py File:

```
from django.db import models
class myPost(models.Model):
    title=models.CharField(max_length=90)
    desc=models.TextField()
    pubDate=models.DateField()
```

admin.py File:

```
from django.contrib import admin
from .models import myPost
# Register your models here.
@admin.register(myPost)
class myPostAdmin(admin.ModelAdmin):
    list_display=['id','title','desc','pubDate']
```

views.py File:

```
from django.http import Http404
from django.shortcuts import render
from .models import myPost
from django.core.paginator import Paginator
from django.views.generic import ListView
```

```
class postListView(ListView):
    model=myPost
    template_name='blog/home.html'
    ordering=['id']
    paginate_by=3
    paginate_orphans=1

#funtion based view mei:
#let suppose hmare yha total 3 pages hia ouur hm url mei 4 lik dei ya koi string
# wghaira to wo khud bakhud handle krta
#tha yani k last page pe le jata ta lekin class based view mei aeisa nhi hai.
#yha sirf agr url mei 'last' likho gay page_number ki jga pe to last page dikhaiga
#nhi to page not found wala error dikhayega: ab iss problem ko solve krnay k liye 2
#trikai hai:
    def get_context_data(self,*args,**kwargs):
        try:
            return super(postListView,self).get_context_data(*args,**kwargs)
        except Http404:
            self.kwargs['page']=1 #ab koi bi agr ghalat page_number likh de to page 1 pe return hoga.
            return super(postListView,self).get_context_data(*args,**kwargs)

#2sra trika ye hai: dono same kaam krtay hai.
    # def paginate_queryset(self,queryset,page_size):
    #     try:
    #         return super(postListView,self).paginate_queryset(queryset,page_size)
    #     except Http404:
    #         self.kwargs['page']=1 #ab koi bi agr ghalat page_number likh de to page 1 pe return hoga.
    #         return super(postListView,self).paginate_queryset(queryset,page_size)
```

templates/blog/home.html File: #same as function based view i.e:project 43

```
<html>
<body>
    <h1>Home Page</h1>
    {%for post in page_obj%}
     <h2>{{post.title}}</h2>
     <p>{{post.desc}}</p>
     <small>{{post.pubDate}}</small>
    {% endfor %}
    <span>{{page_obj.number}}</span>

<div>
    <span>
        {% if page_obj.has_previous %}
            <a href="?page={{page_obj.previous_page_number}}">Previous</a>
        {% endif %}

        <span>{{page_obj.number}}</span>

        {% if page_obj.has_next %}
            <a href="?page={{page_obj.next_page_number}}">Next</a>
        {% endif %}
    </span>
</div>  </body>  </html>
```

#Note: this project with extra feature is available i.e project 44.

PREVIOUS PROJECT WITH EXTRA FEATURES(project:44)

views.py File:

```
from django.http import Http404
from .models import myPost
from django.core.paginator import Paginator
from django.views.generic import ListView,DetailView

class postListView(ListView):
    model=myPost
    template_name='blog/home.html'
    ordering=['id']
    paginate_by=3   #ab agr ye line htaya jae to hme sara data 1 page pe dikhaya jae ga
    paginate_orphans=1  #q k hm ne if is_paginated ... else wali condition lgai hai home.html mei.
                        #agr wo condition ouur ye line dono htai jae to phir kuch bhi nhi dikaiga.
    def get_context_data(self,*args,**kwargs):
        try:
            return super(postListView,self).get_context_data(*args,**kwargs)
        except Http404:
            self.kwargs['page']=1
            return super(postListView,self).get_context_data(*args,**kwargs)


class postDetailView(DetailView):
    model=myPost
    template_name='blog/post.html'  #yha context model name ko bheja jata hai detail view mei but in small
                                    #alphabets.
```

urls.py File:

```
from django.contrib import admin
from django.urls import path
from blog import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.postListView.as_view(),name='home'),
    path('post/<int:pk>',views.postDetailView.as_view(),name='post')            ]
```

home.html:

```
<html lang="en">
<body>
    <h1>Home Page</h1>
{%if is_paginated%}
    {%for post in page_obj%}
     <h2>{{post.title}}</h2>
     <p>{{post.desc}}</p>
     <small>{{post.pubDate|truncatewords:20}} <a href="{%url 'post' post.id%}">Read More</a></small>
    {% endfor %}


<div>
    <span>
        {% if page_obj.has_previous %}
            <a href="?page={{page_obj.previous_page_number}}">Previous</a>
        {% endif %}

        <span>Page {{page_obj.number}} of {{page_obj.paginator.num_pages}}</span><br>

        {% if page_obj.has_next %}
            <a href="?page={{page_obj.next_page_number}}">Next</a>
        {% endif %}
    </span>
</div>
{% else %}
    {% for post in object_list %}
    <h2>{{post.title}}</h2>
    <p>{{post.desc}}</p>
    <small>{{post.pubDate}}</small>
    {% endfor %}
{% endif %}

</body>
</html>
```

post.html File:

```
<body>
<h2>Post Detail</h2>
{{mypost.title}}  <br>
{{mypost.desc}}   <br>
{{mypost.pubDate}}    <br>
<a href="{%url 'home'%}">Back to Home</a> </body>
```