

Django

Introduction:

- Django is a web application framework.
- Django is used to develop web applications.
- It written in python programming language.
- By using Django we can develop web applications in very less time.
- It is a high level and open source python based web framework.
- It has clear documentation and it follows the principle of DRY (Do not Repeat Yourself).
- It is based on MVT (Model View Template) design pattern.
- It provides RAD (Rapid Application Development) facilities.

History:

- **Django** was design and developed by Lawrence journal world in 2003 and publicly released under BSD (Berkeley Source Distribution) license in July 2005.
- The original authors of django are Adrian Holovaty and Simon Willison.
- Django was released on 21, July 2005 and its current version is 3.2.3
- It is maintained by DSF (Django Software Foundation).
- Django also pronounced as “jango”.
- The official website for Django documentation is www.djangoproject.com
- Django is a name of jazz guitarist called Django Reinhardt.

Features of Django:

- Rapid Development
- Secure
- Scalable
- Fully loaded
- Versatile
- Open Source
- Vast and Supported Community

Rapid Development:

- Django was designed with the intention to make a framework which takes less time to build web application.

Secure:

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

Scalable:

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

Fully loaded:

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

Versatile:

- Django is versatile in nature which allows it to build applications for different-different domains.
- Now a day, Companies are using Django to build various types of applications like: content management systems, social networks sites or scientific computing platforms etc.

Open Source:

- Django is an open source web application framework. It is publicly available without cost. It can be downloaded with source code from the public repository. Open source reduces the total cost of the application development.

Vast and supported community:

- Django is a one of the most popular web framework. It has widely supportive community and channels to share and connect.

What is a web application?

- The applications which will provide services over the web are called web applications.
- In web application development every programmer should understand the following terms.
 1. Webpage
 2. Website
 3. Web browser
 4. Web server

Webpage: information towards the end user or client is called webpage

- webpage can be of two types
 1. Static webpage
 2. Dynamic webpage
- Non Interaction webpage is called static webpage.
Ex: about us, contact us
- User Interaction webpage is called dynamic webpage.
Ex: login page, register page

Website: is a collection of webpages or group of webpages

Web browser: it is the software which is used to manage different types of webpages

Ex: Google chrome, IE, Opera etc.

Webserver: it is the software which is used to manage different types of websites.

Ex: Apache Tomcat,IIS.

- Every web application contains 2 main components
 1. Front end

2. Back end

Front end:

- It represents what user is seeing on the website.
- We can develop Front-End content by using the following technologies: HTML, CSS, Java script, query and Bootstrap.
- JQuery and Bootstrap are advanced front-end technologies, which are developed by using HTML, CSS and JavaScript only.
- **HTML:** Hyper Text Markup Language Every web application should contain HTML, i.e. HTML is the mandatory technology for web development, and it represents structure of web page.
- **CSS:** Cascading Style Sheets It is optional technology; still every web application contains CSS. The main objective of CSS is to add styles to the HTML Pages like colors, fonts, borders etc.
- **Java script:** It allows adding interactivity to the web application including programming logic.
- The main objective of Java Script is to add functionality to the HTML Pages. I.e. to add dynamic nature to the HTML Pages.
- HTML===>Meant for Static Responses
- HTML+JS==>Meant for Dynamic Responses
- Ex: To display "Welcome to DURGASOFT" response to the end user, only HTML is enough, because it is static response.
- Ex 2: To display current server date and time to the end user, only HTML is not enough we required to use some extra technology like Java script, JSP,ASP,PHP etc. as it is dynamic response.

Back end:

- It is the technology used to decide what to show to the end user on the Front-End, i.e. Backend is responsible to generate required response to the end user, which is displayed by the Front-End.
- Back-End has 3 important components.

1. The Language like Java, Python etc.

2. The Framework like DJango, Pyramid, Flask etc.
 3. The Database like SQLite, Oracle, MySQL etc.
- For the Backend language Python is the best choice because of the following reasons:
 1. Simple and easy to learn.
 2. Lot of libraries.
 3. Concise code.
 - For the Framework DJango is the best choice because it is Fast, Secure and Scalable.
 - Django is the most popular web application framework for Python.
 - Django provides inbuilt database which is nothing but SQLite, which is the best choice for database.
 - The following are various popular web applications which are developed by using Python and Django
 1. YouTube.
 2. Drop box.
 3. Quora.
 4. Instagram.
 5. The Washington times
 6. Disqus.
 7. Mozilla.
 8. Pinterest.
 9. Spotify.
 - 10.NASA
 - 11.National Geographic.

Steps to download and install python and django:

- To work with django framework python software is required.
- To install python software follow the below steps.

Go to Google

|

Type: python download for windows

|

Click on Python.org

|

Click on Download python 3.9.5

|

It will be download python 3.9.5.Exe

|

Open Python 3.9.5.Exe

|

Activate the Checkbox Add Python 3.9 to PATH

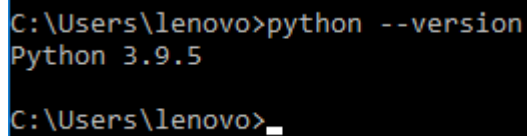
|

Click on Install now

To check python version: go to run----- type cmd---click on ok

Type in command prompt:

python -- version



```
C:\Users\lenovo>python --version
Python 3.9.5
C:\Users\lenovo>_
```

To check pip version:

pip --version

```
C:\Users\lenovo>pip --version
pip 21.1.1 from c:\users\lenovo\appdata\local\programs\python\python39\lib\site-packages\pip (python 3.9)
```

pip is a package-management system written in **Python** used to install and manage software packages.

To check django version:

django-admin --version

```
C:\Users\lenovo>django-admin --version
'django-admin' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\lenovo>_
```

Notice that we don't have django software.

To install django:

First we have to create virtual environment.

A virtual environment is a python environment such that the Python interpreter, libraries and scripts installed into it are isolated from those installed in other virtual environments.

To create virtual environment:

```
C:\Users\lenovo>pip install virtualenvwrapper-win
```

```
C:\Users\lenovo>mkvirtualenv sample
created virtual environment CPython3.9.5.final.0-64 in 14823ms
...
C:\Users\lenovo>
```

Now Install django:

pip install django

```
(sample) C:\Users\lenovo>pip install django
Collecting django
  Using cached Django-3.2.3-py3-none-any.whl (7.9 MB)
Collecting asgiref<4,>=3.3.2
  Using cached asgiref-3.3.4-py3-none-any.whl (22 kB)
Collecting pytz
  Using cached pytz-2021.1-py2.py3-none-any.whl (510 kB)
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.1-py3-none-any.whl (42 kB)
Installing collected packages: sqlparse, pytz, asgiref, django
Successfully installed asgiref-3.3.4 django-3.2.3 pytz-2021.1 sqlparse-0.4.1
```

To check django version:

django-admin --version

```
(sample) C:\Users\lenovo>django-admin --version
3.2.3
```

To create a directory:

mkdir myprojects

```
(sample) C:\Users\lenovo>mkdir myprojects
```

cd myprojects

```
(sample) C:\Users\lenovo\myprojects>
```

To create a project:

django-admin startproject sampleproject

```
(sample) C:\Users\lenovo\myprojects>django-admin startproject sampleproject
```

Now you can move to your project:

cd sampleproject

```
(sample) C:\Users\lenovo\myprojects>cd sampleproject
```

```
(sample) C:\Users\lenovo\myprojects\sampleproject>_
```

Then type dir


```
(sample) C:\Users\lenovo\myprojects\sampleproject>dir
Volume in drive C has no label.
Volume Serial Number is BE30-69DA

Directory of C:\Users\lenovo\myprojects\sampleproject

06/02/2021  05:20 PM    <DIR>          .
06/02/2021  05:20 PM    <DIR>          ..
06/02/2021  05:20 PM                691 manage.py
06/02/2021  05:20 PM    <DIR>          sampleproject
               1 File(s)                691 bytes
               3 Dir(s)  47,799,799,808 bytes free
```

To run server:

python manage.py runserver

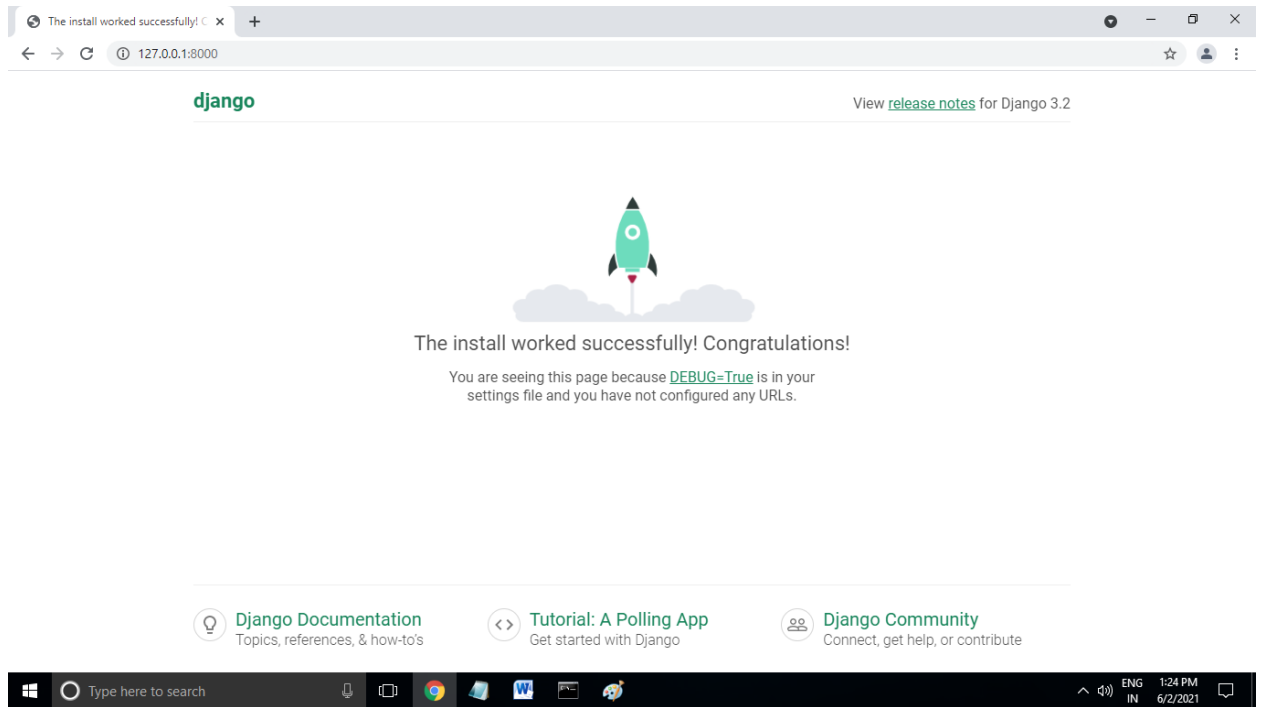
```
(sample) C:\Users\lenovo\myprojects\sampleproject>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
June 02, 2021 - 17:22:50
Django version 3.2.4, using settings 'sampleproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Now you have address: <http://127.0.0.1:8000/>

Go to any browser and type in URL: <http://127.0.0.1:8000/>



Now server is running successfully.

Steps to download and install Pycharm Professional Editor for Django Apps:

Go to Google

|

Type: download pycharm

|

Click on JetBrains.com



Version: 2021.1.1
Build: 211.7142.13
22 April 2021

[System requirements](#)

[Installation Instructions](#)

Download PyCharm

[Windows](#)

[macOS](#)

[Linux](#)

Professional

For both Scientific and Web Python development. With HTML, JS, and SQL support.

Download

Free trial

Community

For pure Python development

Download

Free, open-source

Click on Download professional

|

It will be download pycharm professional editor

|

Open pycharm professional editor from download folder

|

Double click on it

|

Click on Next

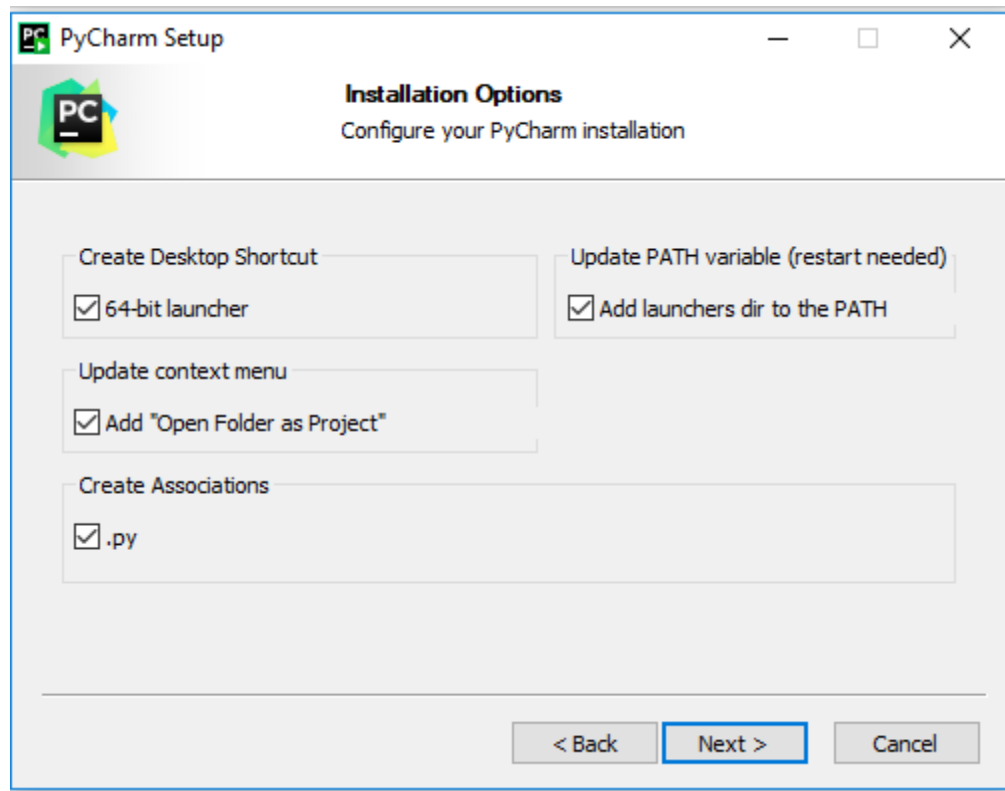
|

Click on Next

|

Activate all check boxes

|



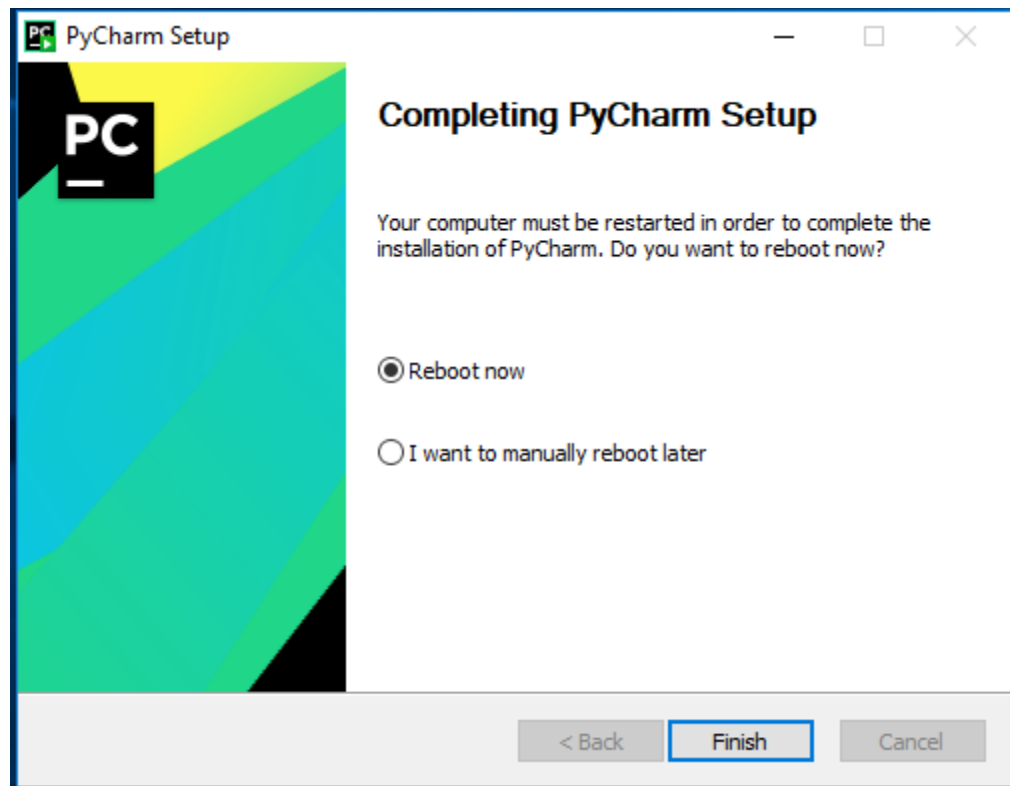
|

Click on Next

|

Click on Install

|

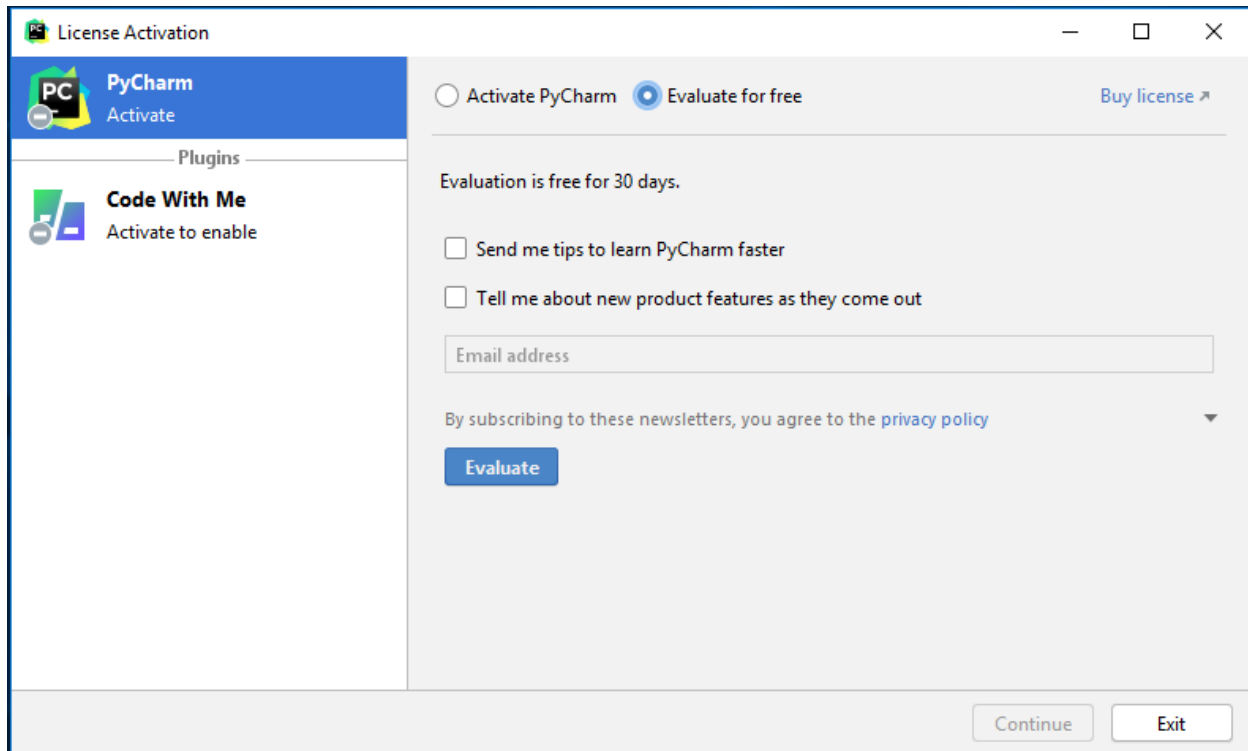


|

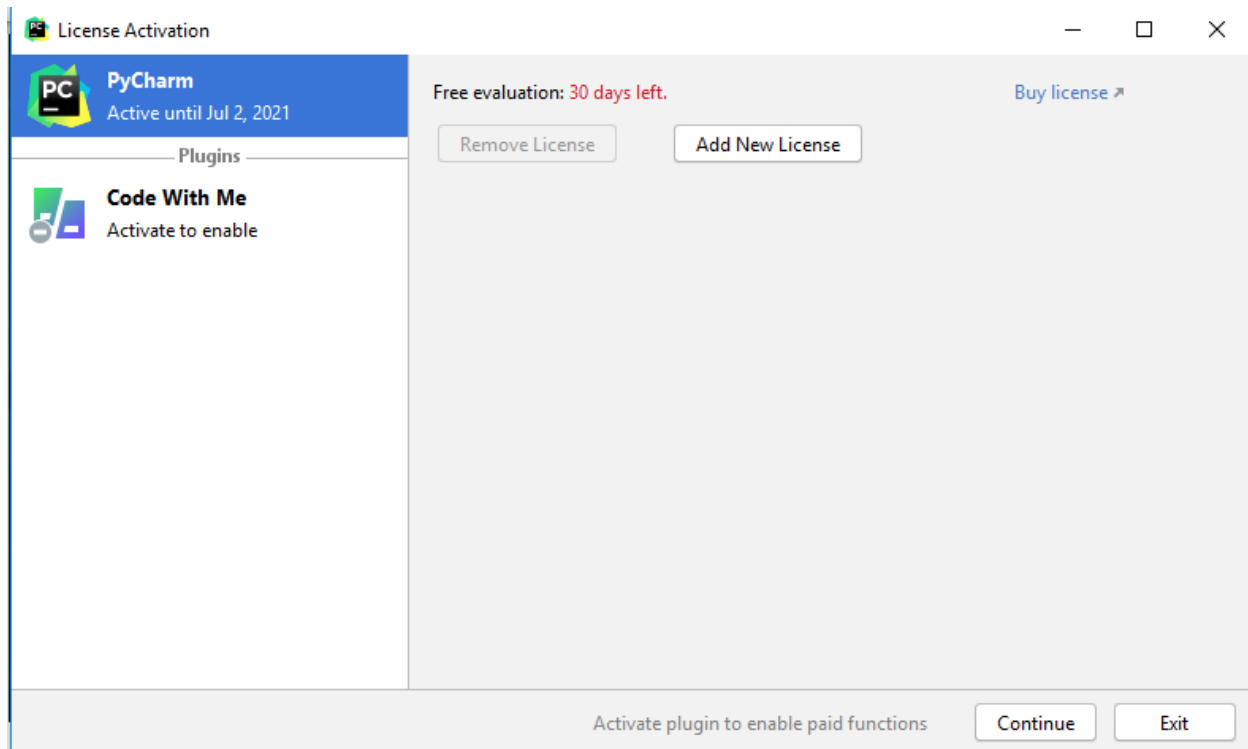
Click on Finish

Now your pycharm professional editor is ready.

Go to search -----type pycharm ----open pycharm professional editor



Click on Evaluate



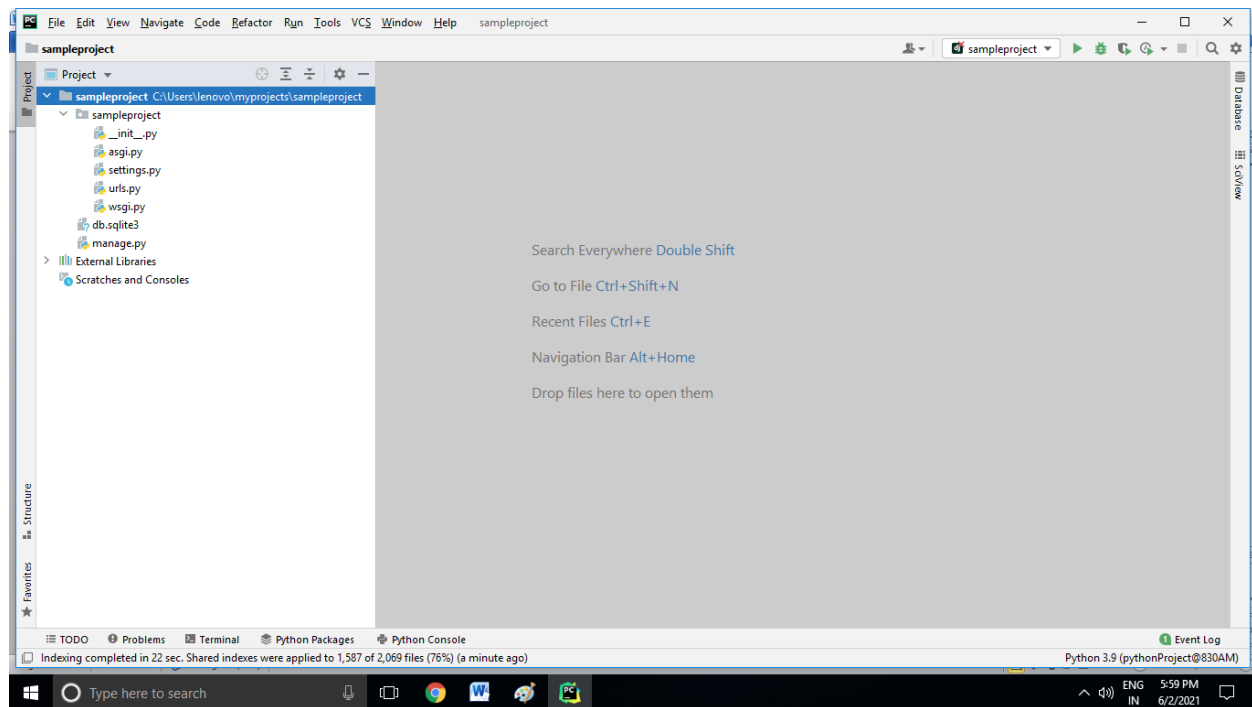
Click on continue

In pycharm editor click on file menu ----click on open and select your project

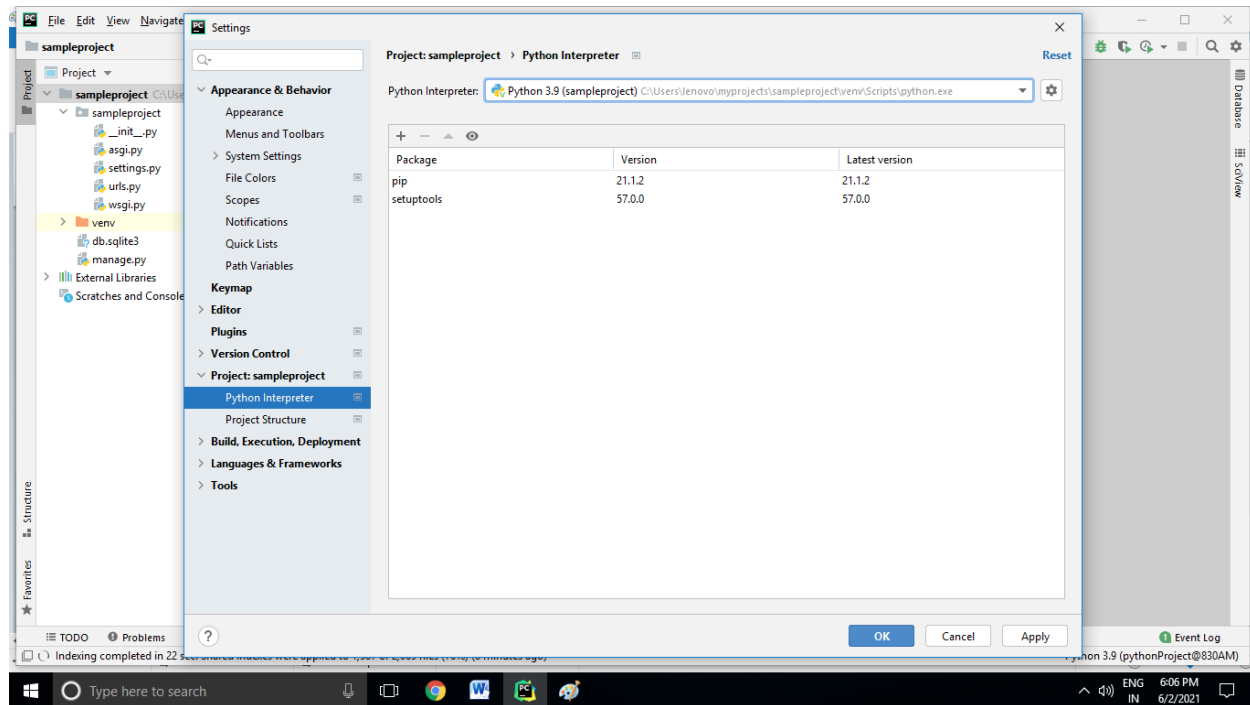
What we create through command prompt.

```
C:\Users\lenovo\myprojects\sampleproject
```

Click on ok



Click on file menu----click on settings---expand project:sampleproject

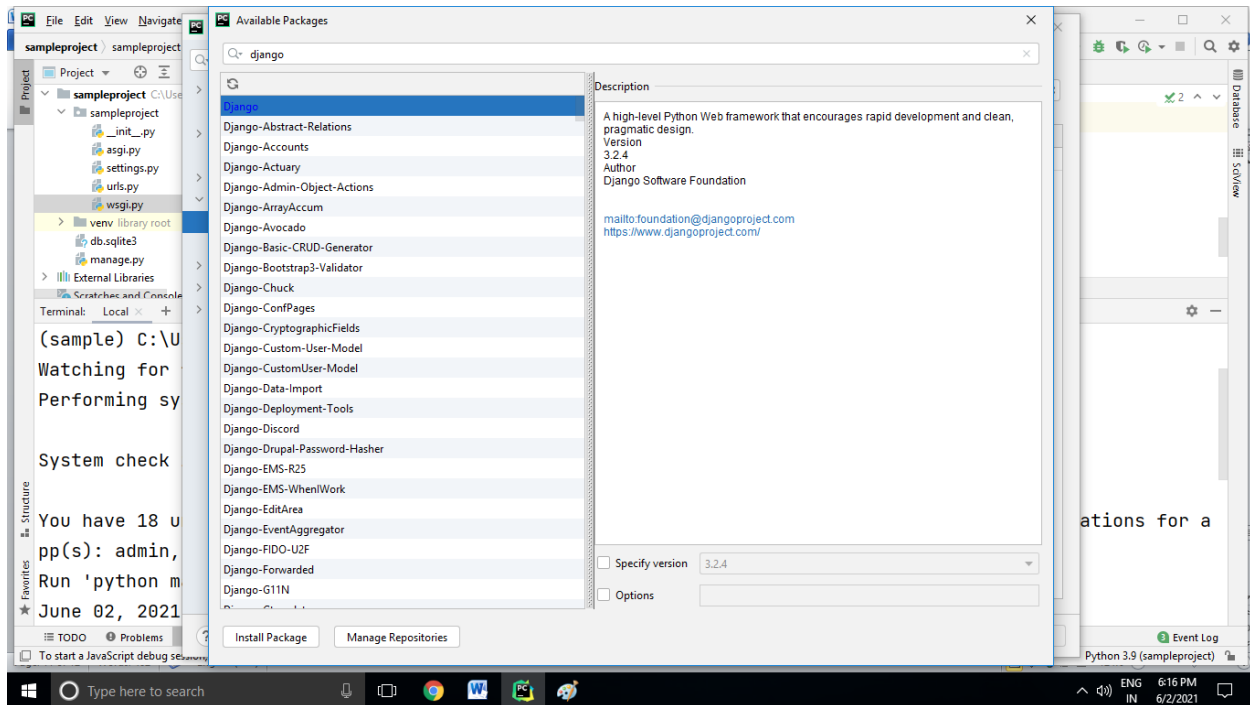


Select python interpreter like python 3.9(sampleproject)----click on ok

Now try to install django in pycharm Editor

Click on file -----settings----- select Project:sampleproject-----click on + symbol

Type django in search box----select Django ----click on install package.



Now go to terminal from bottom of pycharm editor

Change the virtual environment

Work on sample and then run server

```
Terminal: Local x +
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

(venv) C:\Users\lenovo\myprojects\sampleproject>workon sample
(sample) C:\Users\lenovo\myprojects\sampleproject>python manage.py runserver
```

Now start creating application under sample project.

For this go to terminal and type

Python manage.py startapp myapp

```
(sample) C:\Users\lenovo\myprojects\sampleproject>python manage.py startapp myapp
```

Now go to views.py file under myapp then write the following code

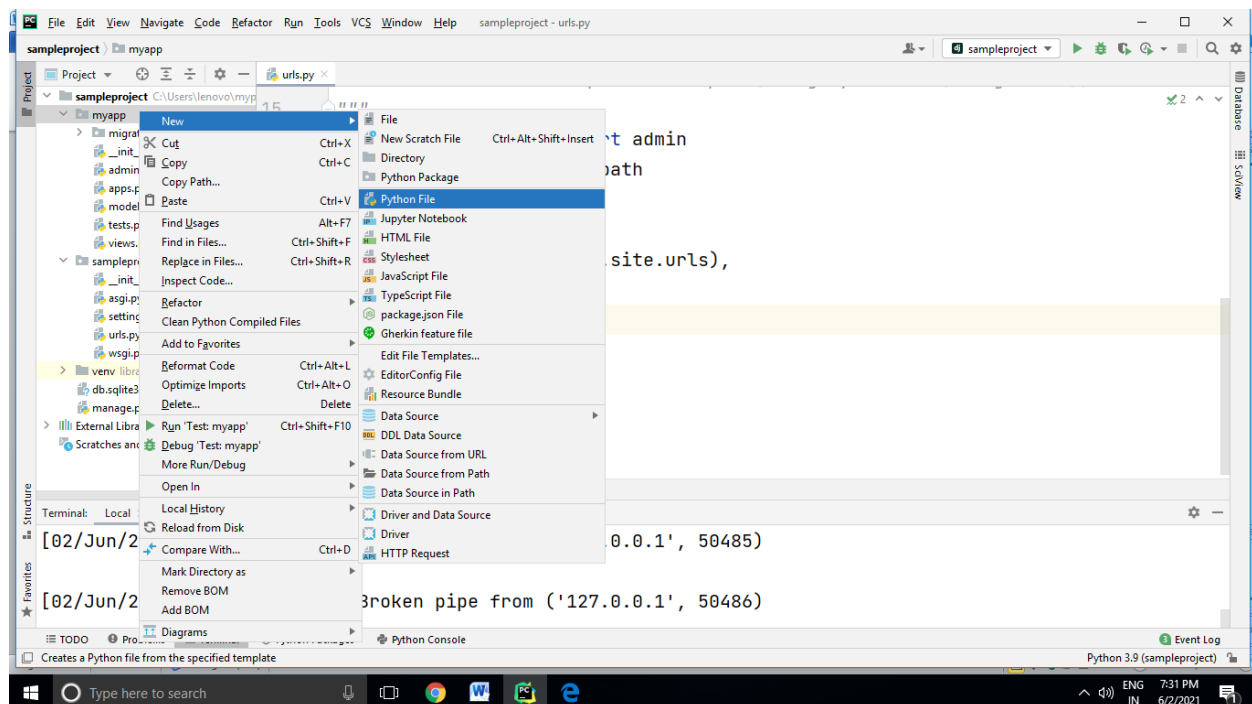
views.py:

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

# Create your views here.
def fl(request):
    return HttpResponse("This is my first webpage")
```

Now create an urls.py under myapp

Right click on myapp----go to new ----select python file----type filename urls.py---press enter.



This will create myapp\urls.py, in this file write the following code.

myapp\urls.py:

```
from django.urls import path
from . import views
```

```
urlpatterns=[  
    path('',views.f1,name="home")  
]
```

Now go to urls.py file under sampleproject then write the following code

sampleproject\urls.py:

```
from django.contrib import admin  
from django.urls import path,include  
  
urlpatterns = [  
    path('',include('myapp.urls')),  
    path('admin/', admin.site.urls),  
]
```

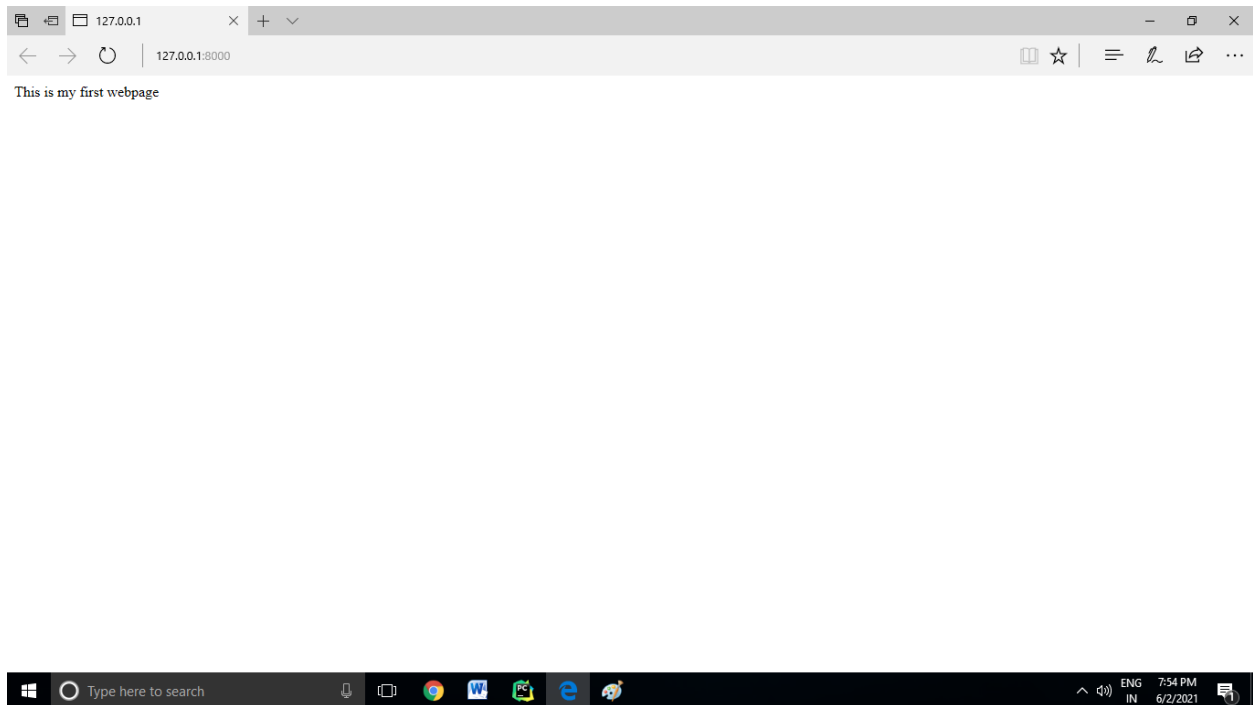
Now go to terminal and run server

```
(sample) C:\Users\lenovo\myprojects\sampleproject>python manage.py runserver
```

Click on the link

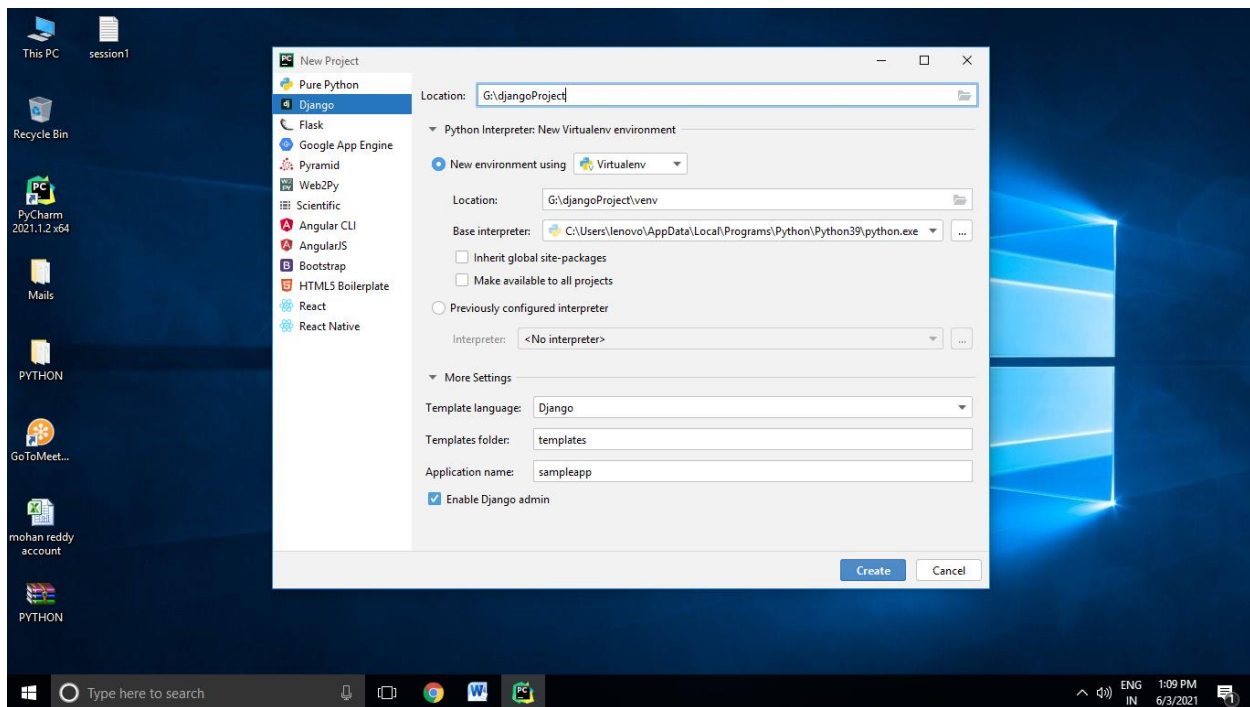
Starting development server at <http://127.0.0.1:8000/>

You will be navigating to browser then the output is:



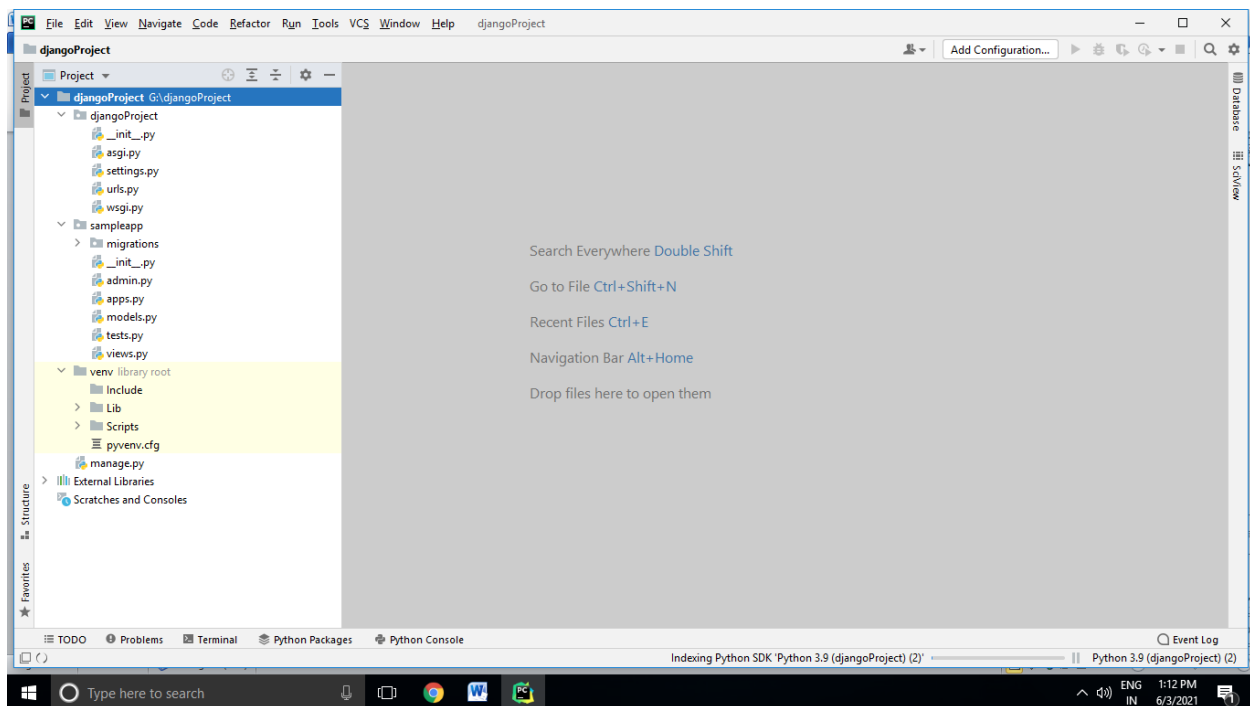
Now we can create a new project and application in pycharm editor without using command prompt.

Click on file menu-----click on New project----select Django-----Expand more settings -----give app name as sampleapp----click on create.

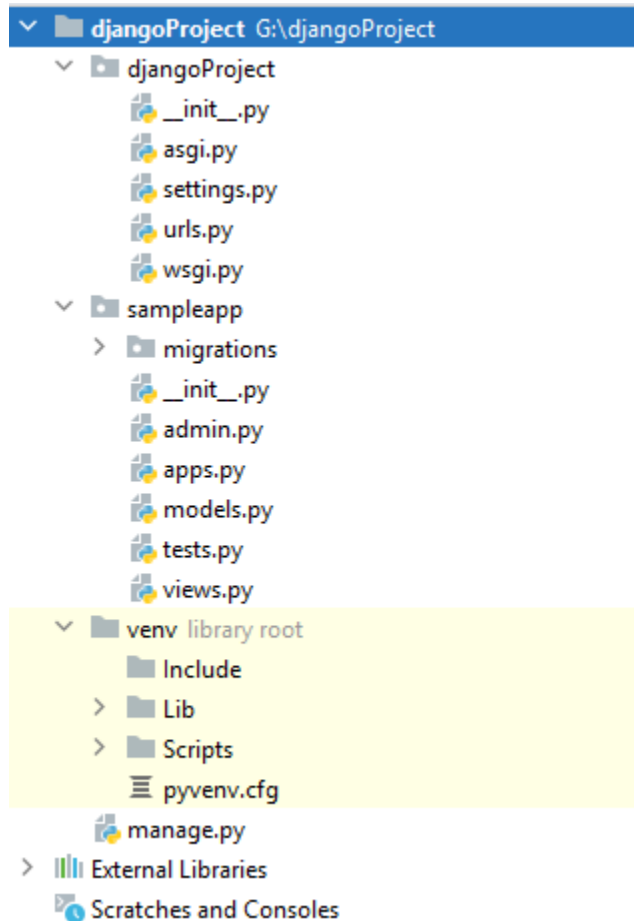


This will create a new project with the name djangoProject
Application name sampleapp.

Here by default virtual environment will create(venv)



Observe the project hierarchy:



Now go to views.py under sampleapp and write the following code.

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

# Create your views here.
def index(request):
    return HttpResponse("Welcome to durgasoft")
```

Now go to urls.py under djangoProject and write the following code

```
from django.contrib import admin
from django.urls import path
from sampleapp import views
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index, name="home")
]
```

Now go to terminal and run server

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.
(venv) G:\djangoProject>python manage.py runserver
```

Django version 3.2.4, using settings 'djangoProject.settings'
Starting development server at <http://127.0.0.1:8000/>
Quit the server with CTRL-BREAK.

Click on the link: <http://127.0.0.1:8000>

Here 8000 is a default port number.

If we don't have django software in pycharm:

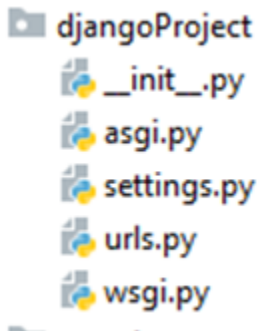
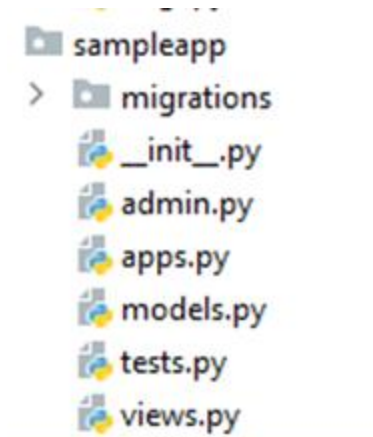
Steps to install django in pycharm editor:

Click on file menu-----click on settings-----expand project:djangoProject-----select python interpreter-----click on + symbol----type django in search---select django---

Click on install package.

[or]

Go to terminal---- type : pip install django

Django project structure:Django application structure:**manage.py :**

This file is used basically as a command-line utility and for deploying, debugging, or running our web application.

runserver:

This command is used to run the server for our web application.

Migration:

This is used for applying the changes done to our models into the database. That is if we make any changes to our database then we use migrate command. This is used the first time we create a database.

Makemigration:

this is done to apply new migrations that have been carried out due to the changes in the database.

`_init_.py` :

This file remains empty and is present there only to tell that this particular directory(in this case django project) is a package.

We won't be doing any changes to this file.

`setting.py` :

This file is present for adding all the applications and the middleware application present. Also, it has information about templates and databases. Overall, this is the main file of our django web application.

`urls.py` :

This file handles all the URLs of our web application. This file has the lists of all the endpoints that we will have for our website.

`wsgi.py` :

This file mainly concerns with the WSGI server and is used for deploying our applications on to servers like Apache etc.

WSGI, short for Web Server Gateway Interface can be thought of as a specification that describes how the servers interact with web applications.

Again we won't be doing any changes to this file.

`asgi.py` :

In the newer versions of django, you will also find a file named as `asgi.py` apart from `wsgi.py`. ASGI can be considered as a successor interface to the WSGI.

ASGI, short for Asynchronous Server Gateway interface also has the work similar to WSGI but this is better than the previous one as it gives better freedom in Django development. That's why WSGI is now being increasingly replaced by ASGI.

Again we won't be doing any changes to this file.

admin.py :

As the name suggests, this file is used for registering the models into the Django administration.

The models that are present have a superuser/admin who can control the information that is being stored.

apps.py :

This file deals with the application configuration of the apps. The default configuration is sufficient enough in most of the cases and hence we won't be doing anything here in the beginning.

models.py :

This file contains the models of our web applications (usually as classes).

Models are basically the blueprints of the database we are using and hence contain the information regarding attributes and the fields etc of the database.

views.py :

This file is a crucial one, it contains all the views(usually as classes). Views.py can be considered as a file that interacts with the client.

A view function, or view for short, is a Python function that takes a Web request and returns a Web response. For the sake of putting the code somewhere, the convention is to put views in a file called views.py.

urls.py :

Just like the project urls.py file, this file handles all the URLs of our web application. This file is just to link the Views in the app with the host web URL. The settings urls.py has the endpoints corresponding to the Views.

In url.py, the most important thing is the "urlpatterns" tuple. It's where you define the mapping between URLs and views.

tests.py :

This file contains the code that contains different test cases for the application. It is used to test the working of the application.

HTML Basics:

- HTML stands for Hypertext markup language. This is the most basic building block of every web application.
- Without using HTML we cannot build web applications. It is the mandatory technology.
- We can use CSS to style HTML Pages.
- We can use Java Script to add functionality to the HTML pages.
- In general we will add django template tags to HTML for generating dynamic content based on our requirement.

Structure of HTML Page:

Every HTML page contains 2 parts

1. Head

2. Body

Head contains Meta data like title of the page, keywords etc. CSS files and Java Script files information we have to specify in the Head Part only.

Body contains actual content.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
```

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

HTML Comment:

```
<!-- Anything here is considered as Comment -->
```

Heading Tags:

HTML supports 6 heading tags <h1>,<h2>,...<h6>

```
<h1>This is Heading1</h1>
```

Paragraph tag: <p>:

We can use this tag to represent paragraph of text.

```
<p> This is first paragraph </p>
```

Bold and Italic:

legacy tags:

```
<b> for bold
```

```
<i> for italic
```

These are old (legacy) html tags and not recommended to use.

```
Ex: <p><b><i>This is First Line</i></b></p>
```

Advanced tags:

We can use the following HTML 5 advanced tags for bold and italic

```
<strong> for strong text(bold)
```

```
<em> for emphasis (italic)
```

```
Ex: <p><strong><em>This is Second Line</em></strong></p>
```

Ex:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>HTML Basics</title>
</head>
<body>
<h1>Heading1</h1>
<p>This is first paragraph</p>
<p><b><i>This is First Line</i></b></p>
<p><strong><em>This is Second
Line</em></strong></p>
</body>

```

HTML Lists:

There are 2 types of lists

1. Ordered list
2. Un ordered list

1. Ordered list:

All list items will be displayed with numbers.

2. Un ordered list

Instead of numbers bullet symbol will come. Here order is not important.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>HTML Basics</title>
</head>
<body>
<ol>

```

```

    <li>C</li>
    <li>C++</li>
    <li>Java</li>
    <li>Python</li>
</ol>
<ul>
    <li>C</li>
    <li>C++</li>
    <li>Java</li>
    <li>Python</li>
</ul>
</body>
</html>

```

Nested Lists:

We can take list inside list, which are nothing but nested lists.

```

<ol>
    <li>C</li>
    <li>C++</li>
    <li>Java</li>
    <li>Python</li>
    <ul>
        <li>Core python</li>
        <li>Adv python</li>
        <li>Django</li>
    </ul>
</ol>

```

Div and Span Tags:

We can use div and span tags to group related tags into a single unit.

In general we can use these tags with CSS .

Note: <div> and tags are helpful only for styling html. Hence they will always work with css only.

Attributes:

HTML Attributes will provide extra information to HTML tags.

To insert image in the html page, src attribute specify location of the image to the tag.

Setting Image inside HTML:

```

```

src means source where we have to specify the image source(complete location). We can take image address from the google also.

alt means alternate. If image is missing then broken link image will display. In this case if we want to display some meaningful text information then we should go for alt attribute.

Note: We have to open the tag and we are not responsible to close the tag, such type of tags are called self-closing tags.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

</body>
</html>
```

Table Creation:

We can use <table> -----to create table

<thead> -----to specify head row

<th> -----to specify column data in head row(column name)

<tr>----- to insert row in the table

<td> -----to specify column data in the row/record

We can use border attribute inside <table> tag

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Basics of HTML for Django</title>
</head>
<body>
<table border="1">
    <thead>
        <th>Eno</th>
        <th>Ename</th>
        <th>Eaddress</th>
        <th>Esalary</th>
    </thead>
    <tr>
        <td>101</td>
        <td>sai</td>
        <td>hyd</td>
        <td>4000</td>
    </tr>
    <tr>
        <td>102</td>
        <td>sana</td>
        <td>hyd</td>
        <td>7800</td>
```



```

    </tr>
    <tr>
    <td>103</td>
    <td>mohan</td>
    <td>hyd</td>
    <td>7000</td>
  </tr>
  <tr>
  <td>104</td>
  <td>mani</td>
  <td>hyd</td>
  <td>9000</td>
</tr>
</table>
</body>
</html>

```

Creating Hyperlinks by using anchor tag: <a>:

```
<a href="home.html">Click Here to go home Page</a>
```

```
<a href="https://facebook.com">FaceBook</a>
```

Ex:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<a href="home.html">Click Here to go home
Page</a><br>
<a href="https://facebook.com">FaceBook</a>
</body>
</html>

```

Display image in table:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Basics of HTML for Django</title>
</head>
<body>
<table border="1">
  <thead>
    <th>sno</th>
    <th>countryname</th>
    <th>countryflag</th>
  </thead>
  <tr>
    <td>1</td>
    <td>INDIA</td>
    <td align="center"></td>
  </tr>
  <tr>
    <td>2</td>
    <td>USA</td>
    <td align="center"></td>
  </tr>
  <tr>
    <td>3</td>
    <td>UK</td>
    <td align="center"></td>
  </tr>
</table>
</body>
</html>
```

HTML Forms:

As the part of web application development, we have to develop several forms like login form, registration form etc.

We can create Html form by using <form> tag.

```
<form class="" action="" method=""> ..... </form>
```

Within the form to collect end user input, we have to use <input> tag. This <input> tag will play very important role in form creation.

syntax: <input type="" name="" value=""/>

type attribute can be used to specify the type of input end user has to provide.

The main important types are:

text

email

password

color

submit

checkbox

radio etc.

name attribute represents the name of input tag. By using this name, in the next target page we can access end user provided input value.

value attribute represents default value will be displayed in the form.

Ex:

```
<input type="text" name="username" value="Enter User Name"/>
```

```
<input type="email" name="mailid" value=""/>
```

```
<input type="password" name="pwd" value=""/>
```

```
<input type="checkbox" name="course" value=""/>
```

```
<input type="radio" name="married" value=""/>
```

To provide default value it is highly recommended to use placeholder attribute because end user is not required to delete default value while entering data.

Ex: `<input type="text" name="username" placeholder="Enter User Name"/>`

Creation of Labels for HTML Elements:

We can define Label Text for our HTML Elements like Radio Buttons, Text Box etc. by using `<label>` tag.

Syntax: `<label for="name">Enter Name:</label>`

Ex: `<p>Enter Name:</p>`

```
<input type="text" name="username" placeholder="Enter Name">
```

In this case there is no relation between text box and data.

To link data to text box, we have to use `<label>` tag.

```
<label for="name">Enter Name:</label>
```

```
<input id="name" type='text' name='username' placeholder='Name to Contact' >
```

Ex:

```
<input type="text" name="username" value=""
```

```

placeholder="Enter User Name"/><br>
<input type="password" name="pwd" value=""/><br>
<input type="email" name="mail" value=""/><br>
  <input type="checkbox" name="course"
value=""/><br>
<input type="radio" name="Django" value=""/><br>

  <p>Enter Name:</p>
  <input type="text" name="username"
placeholder="Enter Name"><br>

<label for="name">Enter Name:</label>
<input id="name" type='text' name='username'
placeholder='Name to Contact' >

```

required attribute:

If end user compulsory required providing input value then we should go for required attribute.

Ex:

```

<input id="name" type='text' name='username' placeholder='Name to Contact'
required>

```

action attribute:

once we fill the form and click submit, then to which page it will go is decided by action attribute. The value of action attribute can be either local resource or web url.

Ex:

```

<form action="target.html" >

<form action="https://facebook.com" >

```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h2>User contact form</h2>
<form action="home.html" >
  <label for="name">Enter name:</label>
  <input id="name" type="text" name='username'
placeholder='name to contact' required>
  <input type="submit" value="click to contact">
</form>
</body>
</html>
```

Marquee:

- An HTML marquee is a scrolling piece of text displayed either horizontally across or vertically down your webpage depending on the settings.
- This is created by using HTML <marquee> tag.
- <marquee attribute_name="attribute_value"....more attributes> Text here </marquee>

Attributes:

- width :This specifies the width of the marquee. This can be a value like 10 or 20% etc.
- height :This specifies the height of the marquee. This can be a value like 10 or 20% etc.
- direction: This specifies the direction in which marquee should scroll. This can be a value like up, down, left or right.
- Scrolldelay: This specifies how long to delay between each jump.

This will have a value like 10 etc.

- scrollamount : This specifies the speed of marquee text.
This can have a value like 10 etc.
- loop : This specifies how many times to loop.
The default value is INFINITE, which means that the marquee loops endlessly.
- bgcolor : This specifies background color in terms of color name .
- hspace: This specifies horizontal space around the marquee.
This can be a value like 10 or 20% etc.
- vspace : This specifies vertical space around the marquee.
This can be a value like 10 or 20% etc.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<marquee>Welcome to durgasoft</marquee>
  <marquee direction ="right">The direction of
text
    will be from left to right</marquee>
  <marquee scrollldelay="20">Using Scroll
Delay</marquee>
  <marquee scrollamount="2">Using Scroll
Amount</marquee>
  <marquee bgcolor="red">Using Background
Color</marquee>
```

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

HTML Formatting:

- HTML defines special elements for defining text with a special meaning.
- HTML uses elements like and <i> for formatting output, like bold or italic text.
- Formatting elements were designed to display special types of text.

```
bold-----<b>
italic-----<i>
underline-----<u>
subscript-----<sub>
superscript-----<sup>
emphasize-----<em>
small  -----<small>
big  -----<big>
strong -----<strong>
delete -----<del>
markedText-----<mark>
insertedText-----<ins>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
```



```

<b>Durgasoft</b>
<strong>Durgasoft</strong>
<i>Durgasoft</i>
<em>Durgasoft</em>
<h2>Durgasoft <small>Info</small> soultions</h2>
<h2>Durgasoft <big>Info</big> soultions</h2>
<h2>HTML <mark>Info</mark> Formatting</h2>
<p>My name is <del>durga</del>mohan</p>
<p>My name is mohan <ins>reddy</ins></p>
<p>This is <sub>subscripted</sub> text.</p>
<p>This is <sup>superscripted</sup> text.</p>

</body>
</html>

```

CSS (Cascading Style Sheets) :

- The main objective of CSS to add styles to HTML. CSS describes how HTML elements are displayed on a page.
- Styling includes colors, fonts, size, borders etc.

Syntax:

```
Selector {Property1: property1-value; Property2: property2-value; }
```

```

Selector {
    Property1: property1-value;
    Property2: property2-value;
}

```

Ex:

```

p { color: red; font-size: 22px;}

p {
    color: red;
    font-size: 22px;
}

```

```
}
```

We can define style in 3 ways:

1. Inline style sheet
2. Internal style sheet
3. External style sheet

Inline style sheet:

- We can define CSS styling inside HTML. But it is highly recommended to define styling inside a separate CSS file(.css extension) and link that file to HTML.
- Inline style is useful when we need to define specific style for individual elements present on a web page. The *style* attribute in a specific Tag or element, is used to create inline style. The style attribute can contain any CSS property between double quotes.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<p style="color: red;font-size: 50px"> It is first
paragraph</p>
<h1 style="color: blue;font-size: 30px"> It is
Heading</h1>

</body>
</html>
```

Internal style sheet:

- Internal Style sheet is a set of style that is created as a part of HTML document. An internal style sheet may be used if one single page has a unique style.
- Internal Style sheets are created using <style> element, which is added inside the <head> element of the HTML document.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style type="text/css">
    p { color: red; font-size: 24px;}
    h1{ color: blue; font-size: 24px;}
  </style>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>This is my first heading</h1>
<p>This is my first paragraph</p>
</body>
</html>
```

External style sheet:

- An external style sheet is a separate document that contains only CSS rules.
- An external style sheet helps to change the look of an entire website by changing just one css file.
- It should not contain any HTML Tags. It has .css extension.

Create a new css file with the name style1.css

Style1.css:--- this is external style sheet

```
h1{color: red}
p{color: orange;font-size: 22px}
h2{color: purple}
```

How to link Web Page to an External Style Sheet:

The href attribute with <link> element inside the <head> tag is used to link web page to an external style sheet.

```
<head>
<title>Title</title>
<link rel="stylesheet" href="style1.css">
</head>
```

EX:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="style1.css">
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>This is Heading1</h1>
<h2>This is Heading2</h2>
<p>This is paragraph</p>
</body>
</html>
```

Priority of Style Sheets:

- Inline Styles
 - External or Internal Style Sheets
- If the internal style is defined after the link to the external style sheet then Internal Style has highest priority.

Ex:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" href="style1.css">
  <style type="text/css">
    p { color: red; font-size: 24px;}
    h1{ color: blue; font-size: 24px;}
  </style>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>This is my first heading</h1>
<p>This is my first paragraph</p>
</body>
</html>

```

- If the internal style is defined before the link to the external style sheet then External Style has highest priority.

EX:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <style type="text/css">
    p { color: red; font-size: 24px;}
    h1{ color: blue; font-size: 24px;}
  </style>
  <link rel="stylesheet" href="style1.css">

  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>This is my first heading</h1>
<p>This is my first paragraph</p>

```

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

Style1.css:

```
h1{color: red}  
p{color: orange;font-size: 22px}  
h2{color: purple}  
body{background-color: cyan}  
div{border-color: brown;  
border-width: thick;  
border-style: dotted;}
```

Ex:

Index.html :

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <link rel="stylesheet" href="style1.css">  
    <meta charset="UTF-8">  
    <title>Title</title>  
</head>  
<body>  
<div>  
    <h1>This is heading1</h1>  
    <p>This is paragraph</p>  
    <h2>This is heading2</h2>  
</div>  
</body>  
</html>
```

Note: CSS comment starts with /* and ends with */.

CSS Selectors:

- CSS selectors are used to select the HTML elements you want to style.
 1. Element selector
 2. Id selector
 3. Class selector
- The element selector selects HTML elements based on the element name.

Ex:

Style1.css:

```
p{
    text-align: center;
    color: red;
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="style1.css">
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<p>First paragraph</p>
<p>Second paragraph</p>
</body>
</html>
```

- The id selector uses the id attribute of an HTML element to select a specific element.
- The id of an element is unique within a page, so the id selector is used to select one unique element.
- To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
#para1 {
  text-align: center;
  color: red;
}
</style>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<p id="para1">First paragraph</p>
<p>Second paragraph</p>
</body>
</html>
```

- The class selector selects HTML elements with a specific class attribute.
- To select elements with a specific class, write a period (.) character, followed by the class name.

EX:

```
<!DOCTYPE html>
<html lang="en">
<head>
```



```

    <style>
.center {
    text-align: center;
    color: red;
}
</style>
}
</style>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<p class="center">First paragraph</p>
<p>Second paragraph</p>
</body>
</html>

```

- You can also specify that only specific HTML elements should be affected by a class.

Ex:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <style>
p.center {
    text-align: center;
    color: red;
}
</style>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1 class="center">First paragraph</h1>
<p class="center">Second paragraph</p>

```

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

- The universal selector (*) selects all HTML elements on the page.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    *{
      text-align: center;
      color: green;
    }
  </style>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>First paragraph</h1>
<p>Second paragraph</p>
</body>
</html>
```

Grouping selectors:

To group selectors, separate each selector with a comma.

```
h1, p, h2, h3 {
    color: red; font-size: 30px;
}
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```

        <style>
            h1, p, h2, h3 {
                color: red; font-size: 30px;
            }
        </style>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
    <h1>Heading1</h1>
    <h2>Heading2</h2>
    <h3>Heading3</h3>
    <p>Paragraph</p>
    </body>
</html>

```

Selector priority:

- **Id**
- **Class**
- **Element**

EX:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        #maindiv p{color: red;}
        #maindiv .cl { color: blue;}
        p{ color: green;}
    </style>
</head>
<body>
<div id="maindiv">
    <p class="cl"> This is para </p>
</div>

```

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

Pseudo element:

- This is used to style specified parts of an element.

Syntax:

```
selector::pseudo-element { property: value; }
```

Ex:

```
p::first-letter { color: red; }
```

```
<style>
  p::first-letter { color: blue;}
  p::first-line { color: red;}
</style>
```

LINK:

- a:link - a normal, unvisited link
- a:visited - a link the user has visited
- a:hover - a link when the user mouse over it
- a:active - a link the moment it is clicked

Ex:-

```
a:link {
  color: red;
}
```

- a:hover MUST come after a:link and a:visited
- a:active MUST come after a:hover

EX:

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
  <style>
    a:link { color: red;}
    a:visited { color: green;}
    a:hover { color: yellow;}
    a:active { color: blue;}

  </style>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<a href="###">Click Here</a>
</body>
</html>

```

Background Image:

Ex:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    body {background-image: url(iflag.png);
          background-repeat: no-repeat;
          background-position: right top;
          background-attachment: fixed;}
  </style>
  <!--<style>
    body {background: url(iflag.png) no-repeat
right top fixed;}
  </style> --->
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

```

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

Full Background Image:

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    html {
      background: url(iflag.png) center fixed
no-repeat;
      background-size: cover;
    }
  </style>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

</body>
</html>
```

div and span:

The div tag is used to group various other HTML elements. It is a block level element. We can say it also create a block

```
<div>
<h1> Heading </h1>
<p> Para </p>
</div>
```

The span tag is used to group inline elements. It is an inline element.

```
<p> This is an <span>example of span tag</span> </p>
```

EX:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <style>
    div{color: red; background-color:blue; }
    span{color: green; font-size: 20px; font-
family: Algerian}
  </style>
</head>
<body>
<div>
  <h1>Heading1</h1>
  <h2>Heading2</h2>
</div>
<p><b>Welcome to</b> <span>Durgasoft</span> </p>
</body>
</html>
```

Overflow:

Overflow : visible, auto, hidden, scroll

```
<style>
  p{
    height: 200px;
    width: 400px;
    border: 3px solid red;
    overflow: visible;
  }
</style>
```

Java script basics:

- JavaScript is a lightweight, interpreted programming language.
- JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages.
- JavaScript was first known as Live Script, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name Live Script.
- Java script is a **MUST** for students and working professionals to become a great Software Engineer especially when they are working in Web Development Domain.
- Java script is used for client side validations that is to verify any user input before submitting it to the server and Java script plays an important role in validating those inputs at front-end itself.

➤ JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

➤ You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

➤ **Syntax:**

```
<script>
```

```
Code here....
```

```
</script>
```

Comments:

```
//single line comment
```

```
/* multiline comments*/
```

Internal java script:

Ex:


```

<!DOCTYPE html>
<html lang="en">
<head>
    <!--Internal javascript file-->
    <script>
        document.write("Hello durgasoft")
    </script>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
</body>
</html>

```

External java script:

- Create a new java script file with any name.js

Sample.js:

```
document.write("Welcome to durgasoft")
```

Link java script file to html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <!--External javascript file-->
    <script type="text/javascript" src="sample.js">

    </script>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
</body>
</html>

```

Different ways to display result sing java script:

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <script>
    document.write("Hello world")
    window.alert("Hello world")
    alert("Hello world")
    console.log("Hello world")
  </script>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
</body>
</html>
```

To display within the web page:

```
document.write("Hello world")
```

To display alert:

```
window.alert("Hello world")
```

or

```
alert("Helloworld")
```

To display result in console window:

```
console.log("Hello world")
```

Note: After execution, right click on webpage, click on inspect and check in console.

Ex: with variables

```
<!DOCTYPE html>
<html lang="en">
<head>
  <script>
    //program1:
    //var name="mohan"
    //document.write("your name is:",name)

    //program2:
    //var name=prompt("Enter name")
    //document.write("name is:",name)

    //program3:
    //var a=10
    //var b=20
    //document.write("sum is:",a+b)

    //program4: accept values at runtime
    var a=Number(prompt("Enter Num1:"))
    var b=Number(prompt("Enter Num2:"))
    document.write("sum is:",a+b)
  </script>

  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
</body>
</html>
```

Ex:

Sample.js

```
var n=Number(prompt("Enter any number:"))
if(n%2==0)
{
    document.write(n+" "+"is Even")
}
else
{
    document.write(n+" "+"is Odd")
}
```

Ex:

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <script type="text/javascript" src="sample.js">

    </script>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
</body>
</html>
```

Ex: with function

```
<!DOCTYPE html>
<html lang="en">
<head>
    <script type="text/javascript">
        function f1()
        {
            alert("Hello durgasoft")
        }
    </script>
```

```

    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<input type="button" onclick="f1()" value="Click
Here">
</body>
</html>

```

Ex: to change html element by using getElementById.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>

<body>
<p id="demo">This is my first paragraph</p>
<p id="demo1">This is my second paragraph</p>
</body>
<script>

document.getElementById("demo").innerHTML="Welcome
to durgasoft"
    document.getElementById("demo1").innerHTML="<h1
style='color: blue'>Hello Hyderabad</h1>"
</script>
</html>

```

Java script form validation:

- It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

- JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.
- Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

Ex: to validate the name and password. The name can't be empty and password can't be less than 6 characters.

Sample.js

```
function validateform() {  
var name=document.myform.name.value;  
var password=document.myform.password.value;  
  
if (name==null || name=="") {  
    alert("Name can't be blank");  
    return false;  
}else if(password.length<6) {  
    alert("Password must be at least 6 characters  
long.");  
    return false;  
}  
}
```

index.html:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <script type="text/javascript" src="sample.js">  
  
    </script>
```

```

    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form name="myform" method="post"
action="https://flipkart.com" onsubmit="return
validateform()" >
Name:<input type="text" name="name"><br/>
Password:<input type="password"
name="password"><br/>
<input type="submit" value="Login">
</form>
</body>
</html>

```

Ex: retype password validation

Sample.js

```

function matchpwd(){
var firstpassword=document.f1.password1.value;
var secondpassword=document.f1.password2.value;

if(firstpassword==secondpassword){
return true;
}
else{
alert("password must be same!");
return false;
}
}

```

index.html:

```

<!DOCTYPE html>
<html lang="en">
<head>

```

```

<script type="text/javascript" src="sample.js">

</script>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>
<form name="f1" action="home.html" onsubmit="return
matchpwd()" ">
Password:<input type="password" name="password1"
/><br/>
Re-enter Password:<input type="password"
name="password2"/><br/>
<input type="submit">
</body>
</html>

```

JQuery basics:

- Official website: jquery.com
- jQuery is a fast and concise JavaScript library created by John Resig in 2006.
- jQuery is a fast, small, and rich JavaScript library.
- We can use jQuery Library to grab and manipulate html elements, to perform event handling and ajax.
- jQuery supports multiple browsers. i.e it provides support for cross browser compatibility.
- jQuery is a lightweight, "write less, do more", JavaScript library.
- The purpose of jQuery is to make it much easier to use JavaScript on your website.
- The main advantage of jQuery is it provides several methods and objects in the form of javascript file, so that we can use these directly.
- In Plain old java script, we have to write everything manually. But if we jQuery, we are not required to write much code and we can use its library directly.

How to connect with jQuery:

- We can make jQuery library available to our application in the following 2 ways
 1. By Locally
 2. By CDN

1. By Locally:

Download jQuery.js file from jquery.com

<https://code.jquery.com/jquery-3.6.0.min.js>

Download and place in application folder.

Inside head of html we have to write <script> tag as follows.

```
<script type="text/javascript" src="jquery.js">  
  
</script>
```

2. By using CDN: (Content Delivery Network)

```
<script src="https://code.jquery.com/jquery-  
3.6.0.js"  
integrity="sha256-  
H+K7U5CnXl1h5ywQfKtSj8PCmoN9aaq30gDh27Xc0jk="<br>crossorigin="anonymous"></script>">  
</script>
```

jQuery Syntax:

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: **\$(selector).action()**

- A \$ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action()* to be performed on the element(s)

Ex :

Document Ready Event:

```
$(document).ready(function(){

    // jQuery methods here...

});
```

This is to prevent any jQuery code from running before the document is finished loading (is ready).

It is good practice to wait for the document to be fully loaded and ready before working with it. This also allows you to have your JavaScript code before the body of your document, in the head section.

Ex1:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <script src="jquery-3.6.0.min.js"></script>
    <script>
$(document).ready(function() {
    $("button").click(function() {
        $("p").hide();
    });
});
</script>
    <meta charset="UTF-8">
    <title>Title</title>
</head>

<body>
```

```

<p>This is a first paragraph</p>
<p>This is a second paragraph</p>
<button>Click me to hide</button>
</body>
</html>

```

Ex2:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <script src="https://code.jquery.com/jquery-
3.6.0.js"
    integrity="sha256-
H+K7U5CnXl1h5ywQfKtSj8PCmoN9aaq30gDh27Xc0jk="
    crossorigin="anonymous"></script>

  <script>
$(document).ready(function() {
  $("button").click(function() {
    $("#test").hide();
  });
});
</script>
  <meta charset="UTF-8">
  <title>Title</title>
</head>

<body>
<p id="test">This is a first paragraph</p>
<p>This is a second paragraph</p>
<button>Click me to hide</button>
</body>
</html>

```

Ex3:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <script src="jquery-3.6.0.min.js"></script>
  <script>
$(document).ready(function() {
  $("#hide").click(function() {
    $("p").hide();
  });
  $("#show").click(function() {
    $("p").show();
  });
});
</script>
  <meta charset="UTF-8">
  <title>Title</title>
</head>

<body>
<p>This is a paragraph</p>
<button id="hide">Hine</button>
<button id="show">Show</button>
</body>
</html>

```

Ex4: jquery animation

```

<!DOCTYPE html>
<html lang="en">
<head>
  <script src="jquery-3.6.0.min.js"></script>
  <script>
$(document).ready(function() {
  $("#flip").click(function() {
    $("#panel").slideDown(5000);
  });
});

```

```

    });
    $("#stop").click(function() {
        $("#panel").stop();
    });
});
</script>
<style>
#panel, #flip {
    padding: 5px;
    font-size: 18px;
    text-align: center;
    background-color: orangered;
    color: white;
    border: solid 1px #666;
    border-radius: 3px;
}
#panel {
    padding: 50px;
    display: none;
}
</style>
<meta charset="UTF-8">
<title>Title</title>
</head>

<body>
<button id="stop">Stop sliding</button>
<div id="flip">Click to slide down panel</div>
<div id="panel">Welcome to durgasoft</div>
</body>
</html>

```

Bootstrap basics:

- Bootstrap is the most commonly used framework for Front-End Development. [Django is the most commonly used web framework for back-end development with Python]
- Bootstrap providing several pre-defined libraries for css and java script.
- Bootstrap is a free front-end framework for faster and easier web development.
- Bootstrap also gives you the ability to easily create responsive designs.

Advantages:

- supports responsive design
- saves lot of development time
- consistency
- customizable
- support

How to connect Bootstrap with HTML:

- We can connect Bootstrap with HTML by using the following 2 ways
 1. By using CDN
 2. Locally

By using CDN: (Content Delivery Network)

- just add the following in the <head> part of our html head section.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
```

```
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js
"></script>
```

By using Locally:

Download bootstrap.css file from the link [getbootstrap.com](https://getbootstrap.com/docs/5.0/getting-started/download/) and download button

<https://getbootstrap.com/docs/5.0/getting-started/download/>

zip file contains bootstrap.css file. copy this file in our application folder and add the following link in html

```
<link rel="stylesheet" href="Bootstrap.css">
```

Bootstrap table:

```
<table class="table">
```

```
<table class="table table-striped">
```

```
<table class="table table-bordered">
```

```
<table class="table table-hover">
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" href="bootstrap.css">
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<div class="container">
  <h2>Basic Table</h2>
  <table class="table">
    <thead>
      <tr>
```

```
        <th>Firstname</th>
        <th>Lastname</th>
        <th>Email</th>
    </tr>
</thead>
<tbody>
    <tr>
        <td>mohan</td>
        <td>reddy</td>
        <td>mohan@gmail.com</td>
    </tr>
    <tr>
        <td>shashi</td>
        <td>Nandan</td>
        <td>shashi@gmail.com</td>
    </tr>
    <tr>
        <td>shanvi</td>
        <td>reddy</td>
        <td>shanvi@gmail.com</td>
    </tr>
</tbody>
</table>
</div>
</body>
</html>
```

Responsive table:

```
<!DOCTYPE html>
<html lang="en">
```



```
<head>
  <link rel="stylesheet" href="bootstrap.css">
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <div class="table-responsive">
    <h2>Basic Table</h2>
    <table class="table table-hover">
      <thead>
        <tr>
          <th>Firstname</th>
          <th>Lastname</th>
          <th>Email</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>mohan</td>
          <td>reddy</td>
          <td>mohan@gmail.com</td>
        </tr>
        <tr>
          <td>shashi</td>
          <td>Nandan</td>
          <td>shashi@gmail.com</td>
        </tr>
        <tr>
          <td>shanvi</td>
          <td>reddy</td>
          <td>shanvi@gmail.com</td>
        </tr>
      </tbody>
    </table>
  </div>
</body>
</html>
```

Navigation bar:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-
width, initial-scale=1">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5
.2/css/bootstrap.min.css">
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3
.5.1/jquery.min.js"></script>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.
js/1.16.0/umd/popper.min.js"></script>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.
2/js/bootstrap.min.js"></script>
</head>
<body>

<nav class="navbar navbar-expand-md bg-dark navbar-
dark">
  <a class="navbar-brand" href="#">Navbar</a>
  <button class="navbar-toggler" type="button"
data-toggle="collapse" data-
target="#collapsibleNavbar">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse"
id="collapsibleNavbar">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>

```

```

    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
  </ul>
</div>
</nav>
<br>

<div class="container">
  <h3>Collapsible Navbar</h3>
  <p>In this example, the navigation bar is hidden
on small screens and replaced by a button in the
top right corner (try to re-size this window).</p>
  <p>Only when the button is clicked, the
navigation bar will be displayed.</p>
  <p>Tip: You can also remove the .navbar-expand-md
class to ALWAYS hide navbar links and display the
toggler button.</p>
</div>

</body>
</html>

```

Spinners:

Ex:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-

```

```

width, initial-scale=1">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5
.2/css/bootstrap.min.css">
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3
.5.1/jquery.min.js"></script>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.
js/1.16.0/umd/popper.min.js"></script>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.
2/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container">
  <h2>Spinners</h2>
  <div class="spinner-border"></div>
    <div class="spinner-border text-muted"></div>
    <div class="spinner-border text-primary"></div>
    <div class="spinner-border text-success"></div>
    <div class="spinner-border text-info"></div>
    <div class="spinner-border text-warning"></div>
    <div class="spinner-border text-danger"></div>
    <div class="spinner-border text-secondary"></div>
    <div class="spinner-border text-dark"></div>
    <div class="spinner-border text-light"></div>
</div>

</body>
</html>

```

Growing spinners:**Ex:**

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-
width, initial-scale=1">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5
.2/css/bootstrap.min.css">
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3
.5.1/jquery.min.js"></script>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.
js/1.16.0/umd/popper.min.js"></script>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.
2/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container">
  <h2>Spinners</h2>
  <div class="spinner-grow text-muted"></div>
  <div class="spinner-grow text-primary"></div>
  <div class="spinner-grow text-success"></div>
  <div class="spinner-grow text-info"></div>
  <div class="spinner-grow text-warning"></div>
  <div class="spinner-grow text-danger"></div>
  <div class="spinner-grow text-secondary"></div>
  <div class="spinner-grow text-dark"></div>
  <div class="spinner-grow text-light"></div>

```

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

Spinner buttons:

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-
width, initial-scale=1">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5
.2/css/bootstrap.min.css">
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3
.5.1/jquery.min.js"></script>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.
js/1.16.0/umd/popper.min.js"></script>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.
2/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container">
  <h2>Spinner Buttons</h2>
  <p>Add spinners to buttons:</p>

  <button class="btn btn-primary">
    <span class="spinner-border spinner-border-
sm"></span>
  </button>

  <button class="btn btn-primary">
```

```

    <span class="spinner-border spinner-border-
sm"></span>
    Loading..
</button>

<button class="btn btn-primary" disabled>
    <span class="spinner-border spinner-border-
sm"></span>
    Loading..
</button>

<button class="btn btn-primary" disabled>
    <span class="spinner-grow spinner-grow-
sm"></span>
    Loading..
</button>
</div>
</body>
</html>

```

Pagination:

Ex:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Bootstrap Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-
width, initial-scale=1">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5
.2/css/bootstrap.min.css">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3
.5.1/jquery.min.js"></script>
    <script

```

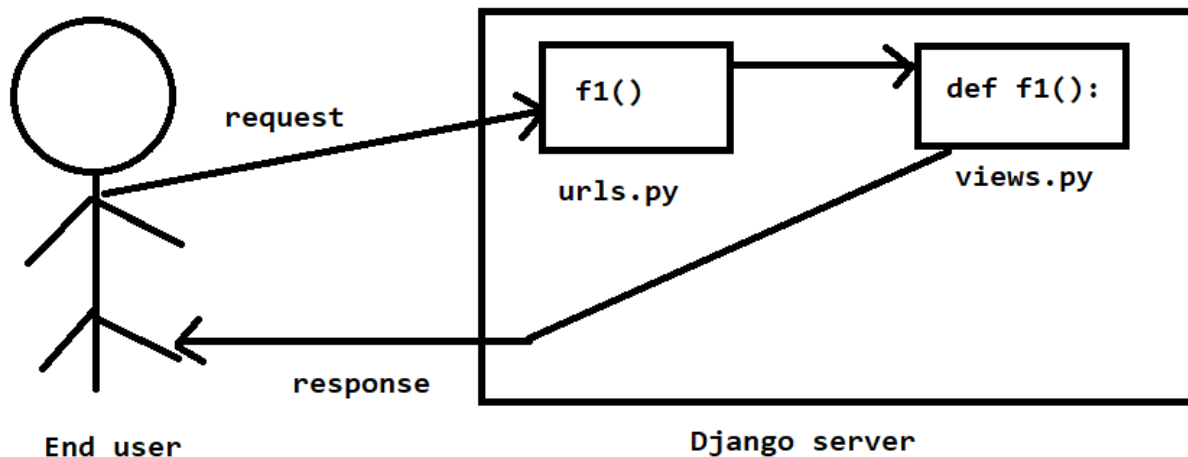
```

src="https://cdnjs.cloudflare.com/ajax/libs/popper.
js/1.16.0/umd/popper.min.js"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.
2/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container">
  <h2>Pagination</h2>
  <ul class="pagination">
    <li class="page-item"><a class="page-link"
href="#">Previous</a></li>
    <li class="page-item"><a class="page-link"
href="#">1</a></li>
    <li class="page-item"><a class="page-link"
href="#">2</a></li>
    <li class="page-item"><a class="page-link"
href="#">3</a></li>
    <li class="page-item"><a class="page-link"
href="#">Next</a></li>
  </ul>
</div>

</body>
</html>

```

Http request flow in Django Application:



1. Whenever end user sending the request, first django development server will get that request.
2. From that request, django will identify urlpattern and by using urls.py, the corresponding view will be identified.
3. The request will be forwarded to the view. The corresponding function will be executed and provide required response to the end user.

Activities related to django project:

1. Create a django project
Ex: `django-admin startproject sampleproject`
2. Create application in project
Ex: `python manage.py startapp sampleapp`
3. Add application to the project(inside settings.py)
4. Define view function inside views.py
5. Define url-pattern for our view inside urls.py
6. Start server---- `python manage.py runserver`
7. Run the app and check.

Working with multiple views in single Application using project level urls.py

Views.py:

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

# Create your views here.
def v1(request):
    return HttpResponse("<h1>This is my first
view</h1>")

def v2(request):
    return HttpResponse("<h1>This is my second
view</h1>")
```

urls.py: (project level)

```
from django.contrib import admin
from django.urls import path
from djapp import views
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('view1', views.v1),
    path('view2', views.v2),
]
```

Note: Here view1 and view2 are names which are used to run a particular view by providing in URL address.

Ex: <http://127.0.0.1:8000/view1>

Working with multiple views in single Application using application level urls.py

Views.py

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

# Create your views here.
def v1(request):
    return HttpResponse("<h1>This is my first
view</h1>")

def v2(request):
    return HttpResponse("<h1>This is my second
view</h1>")
```

urls.py: (application level)

```
from django.urls import path
from . import views

urlpatterns = [
    path('view1', views.v1),
    path('view2', views.v2),
]
```

Link to application level urls.py to project level urls.py

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

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

Note: Define URL patterns at application level instead of project level.

- Django project can contain multiple applications and each application can contain multiple views.
- Defining URL patterns for all views of all applications inside urls.py of project creates maintenance problems and reduces reusability of applications.
- So that it's better to define URL patterns at application level.
- For every application we have to create a separate urls.py and we have to link this application level urls.py file to project level urls.py file by using **include ()** method.

Single project with multiple applications:

Create a new app with the name djapp1

Go to terminal and then type the following command

```
py manage.py startapp djapp1
```

now we have 2 apps like djapp and djapp1.

djapp\views.py :

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

# Create your views here.
def v1(request):
    return HttpResponse("<h1>This is my first
view</h1>")

def v2(request):
    return HttpResponse("<h1>This is my second
view</h1>")
```

djapp1\views.py :

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

```
# Create your views here.
def date_timedisplay(request):
    time=datetime.datetime.now()
    t='<h1>Current date and time
is:'+str(time)+'</h1>'
    return HttpResponse(t)
```

djapp\urls.py :

```
from django.urls import path
from . import views

urlpatterns = [
    path('view1',views.v1),
    path('view2',views.v2),
]
```

djapp1\urls.py :

```
from django.urls import path
from . import views

urlpatterns = [
    path('date',views.date_timedisplay),
]
```

Link both application level urls.py to project level urls.py:

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

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

```
    path('', include('djapp1.urls')),
]
```

DTL(Django Template Language):

Django Templates:

- It is not recommended to write html code inside python script(views.py) file.
- Because it will reduces readability.
- Always we have to mix python code with html code.
- To overcome this, we have to separate html code into a separate html file is nothing but template.
- From python code(views.py) we can use these templates based on our requirement.
- We have to write templates at project level only, then we can use templates in multiple applications.

Ex: with template based application:

Create a new html page in Template folder of your project with the name home.html.

home.html :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1 style="color: red">Welcome to durgasoft</h1>
<h2>Welcome to {{ name }}</h2>
</body>
</html>
```

Views.py :

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

# Create your views here.

def home(request):
    return
render(request, 'home.html', {'name': "mohan"})
```

djapp\urls.py :

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

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

Project level urls.py:

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

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

GET and POST methods:

GET: In the GET method, after the submission of the form, the form values will be visible in the address bar of the new browser tab. It has a limited size of about 3000 characters. It is only useful for non-secure data not for sensitive information.

POST: In the post method, after the submission of the form, the form values will not be visible in the address bar of the new browser tab as it was visible in the GET method. It appends form data inside the body of the HTTP request. It has no size limitation.

Ex:

Home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

<form action="add" method="get">
    {% csrf_token %}
    Num1:<input type="text" name="num1"><br>
    Num2:<input type="text" name="num2"><br>
    <input type="submit" value="ADD">
</form>
</body>
</html>
```

Views.py:

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

# Create your views here.
```



```
def home(request):
    return render(request, "home.html")

def add(request):
    a=int(request.GET['num1'])
    b=int(request.GET['num2'])
    res=a+b
    return
render(request, "result.html", {'result':res})
```

urls.py :(application level):

```
from django.urls import path
from . import views
```

```
urlpatterns = [

    path('', views.home),
    path('add', views.add),
]
```

result.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>The result is:{{ result }}</h1>
</body>
</html>
```

Note: when we execute the above code for adding two numbers, the numbers are visible in URL to avoid this then we use POST method instead of GET method.

Render:

- **Render** Combines a given template **with a** given context dictionary and returns an **Http Response** object with that **rendered** text.

Cross site request forgery (CSRF):

- **CSRF tokens** can prevent **CSRF** attacks by making it impossible for an attacker to construct a fully valid HTTP request suitable for feeding to a victim user.

Django static files:

- In a web application, we need to handle and manage static resources like CSS, JavaScript, images etc.
- It is important to manage these resources so that it does not affect our application performance.
- Django deals with it very efficiently and provides a convenient manner to use resources.
- The `django.contrib.staticfiles` module helps to manage them.

Note: Include `django.contrib.staticfiles` in `settings.py` under installed apps, but in pycharm editor by default its included.

```
INSTALLED_APPS = [
    'django.contrib.staticfiles',
]
```

Steps to work with static files:

Step1: create a new folder or directory with the name static under project folder.

Step2: create 3 folders with the name css, javascript, images under static folder.

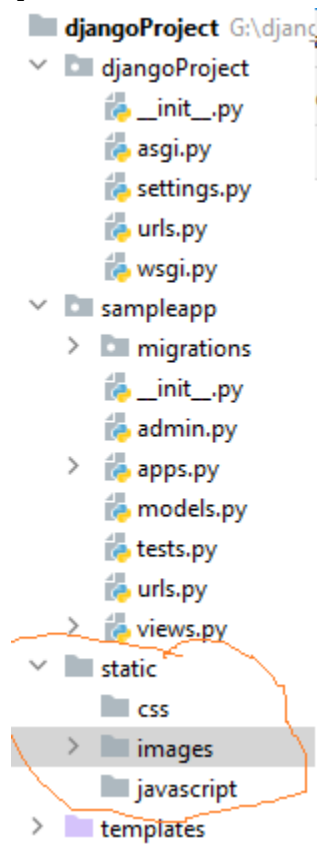
Step3: Go to `settings.py` and write the following code.

```
import os
```

```
STATIC_DIR=os.path.join(BASE_DIR, 'static')
```

Step4: Configure static directory path under settings.py

```
STATIC_URL = '/static/'
STATICFILES_DIRS=[
    STATIC_DIR,
]
```



To load static files into html document then we use the following syntax:

```
{% load static %}
```

➤ **Steps to use images into templates (html page):**

- Place required images into images folder under static folder.
- Create new html page with the name index.html

Index.html:

```
<!DOCTYPE html>
<html lang="en">
{% load static %}
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    
</body>
</html>
```

Views.py :

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

def index(request):
    return render(request, "index.html")
```

urls.py : (application level)

```
from django.urls import path
from . import views

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

Note: Make sure that we have to map application level urls to project level urls.

urls.py : (projectlevel level)

```

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

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

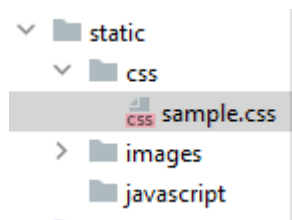
```

Now start server and type in address bar:

http://127.0.0.1:8000/index

➤ Steps to use CSS files into templates (html page):

Create a new CSS file under css folder of static folder with the name sample.css



sample.css :

```

h1{
    color:orange;font-size: 20px; font-family:
Calibri;
}
h2{
    color:maroon;font-size: 20px; font-
family:Arial;
}

```

index.html:

```

<!DOCTYPE html>
<html lang="en">
{% load static %}

```

```
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link href="{% static 'css/sample.css' %}"
rel="stylesheet">
</head>
<body>
<h1>Welcome to Durgasoft</h1>
<h2>Hello Hyderabad</h2>
</body>
</html>
```

Views.py :

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

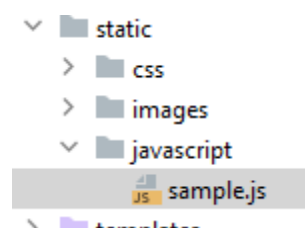
def index(request):
    return render(request, "index.html")
```

Now start server and type in address bar:

http://127.0.0.1:8000/index

➤ Steps to use java script files into templates (html page):

Create a new java script file under java script folder of static folder with the name sample.js



sample.js :

```
alert("Hello,Hyderabad")
```

index.html :

```
<!DOCTYPE html>
<html lang="en">
  {% load static %}
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script src="{% static 'javascript/sample.js'
  %}" type="text/javascript"></script>
</head>
<body>
</body>
</html>
```

Views.py :

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

def index(request):
    return render(request, "index.html")
```

Now start server and type in address bar:

http://127.0.0.1:8000/index

Note: we can also apply CSS, java script, images at a time in html page.

Ex:

```
<!DOCTYPE html>
<html lang="en">
  {% load static %}
<head>
```

```

<meta charset="UTF-8">
<title>Title</title>
    <link href="{% static 'css/sample.css' %}"
rel="stylesheet">
    <script src="{% static 'javascript/sample.js'
%}" type="text/javascript"></script>
</head>
<body>
    
<h1>Welcome to Durgasoft</h1>
<h2>Hello Hyderabad</h2>
</body>
</html>

```

DTL(Django Template Language):

- **Django** Template Language or **DTL** is a text-based Template language that provides a bridge between scripts like HTML, CSS, JS, etc. and programming languages like python.
- **DTL** is specifically built for developers to embed **Django** logic codes into HTML template files.
- **Jinja2** is a modern day templating language for Python developers.
- It was made after Django's template.
- It is used to create HTML, XML or other markup formats that are returned to the user via an HTTP request.
- It is a text based template language and it can be used to generate any markup as well as source code.

Basic syntax:

{%.....%}	For statements
{{.....}}	For expression to print output in template

{#.....#} For comments

- Variables look like this: **{{ variable }}** When the template engine encounters a variable, it evaluates that variable and replaces it with the result.

Rules:

- Variable names consist of any combination of alphanumeric characters and the underscore.
- Variable name should not start with underscore.
- Variable name cannot have spaces or punctuation characters.
 - Syntax: **{{ variable }}**
 - Example:- **{{ result }}**, **{{ name }}**, **{{ emp_name }}**

Filter:

- When we need to modify variable before displaying we can use filters.
- Pipe ‘|’ is used to apply filter.

Syntax: **{{ variable | filter }}**

Ex: **{{ name|upper }}**

Ex: **{{ value|lower }}**

Ex: **{{ value|length }}**

Ex: **{{ value|capfirst }}**

Features of jinja2:

- Sandboxed execution.
- Automatic HTML escaping to prevent cross-site scripting (XSS) attacks.
- Template inheritance.
- Easy to debug (for example, line numbers of exceptions directly point to the correct line in the template).

A sandbox is an isolated testing environment that enables users to run programs or execute files without affecting the application, system or platform on which they run.

Dynamic Template Data:

Ex:

views.py:

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

# Create your views here:
def emp_details(request):
    eid=1234
    ename="sai"
    eaddress="hyderabad"
    designation="MANAGER"
    salary=6000
    age=50

    details={"eid":eid, "ename":ename, "eaddress":eaddress, "designation":designation,
            "salary":salary, "age":age}
    return render(request, "emp.html", details)
    #or
    #return render(request, "emp.html", context=
    details)
```

emp.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
```

```

<!--to display data-->
<h2 style="color: red">Id: {{ eid }}<br>
Name: {{ ename|capfirst }}<br>
Address: {{ eaddress|upper }}<br>
Designation: {{ designation|lower }}<br>
Salary: {{ salary }}<br>
Age: {{ age }}<br></h2>
<!--to display in table-->
<table border="1" style="background-color: bisque">
<tr>
    <th>Id</th>
    <th>Name</th>
    <th>Address</th>
    <th>Designation</th>
    <th>Salary</th>
    <th>Age</th>
</tr>
<tr>
    <td>{{ eid }}</td>
    <td>{{ ename }}</td>
    <td>{{ eaddress }}</td>
    <td>{{ designation }}</td>
    <td>{{ salary }}</td>
    <td>{{ age }}</td>
</tr>
</table>
</body>
</html>

```

Template inheritance:

- Template Inheritance is a method to add all the elements of an HTML file into another without copy-pasting the entire code.
- 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.

Ex:

base.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Hello Hyderabad{%
endblock %}</title>

</head>
<body bgcolor="green">
<h1>welcome to durgasoft</h1>
{% block content %}

{% endblock %}
</body>
</html>
```

index.html:

```
<!DOCTYPE html>
{% extends 'base.html' %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title> Hello durgasoft </title>
</head>
<body>
```

```
{% block content %}
```

```
{% endblock %}
```

```
</body>
```

```
</html>
```

Note: in the above example base.html elements are used in index.html, means index.html is inherited from base.html.

Date and Time Filter:

Predefined Formats:

Format	Example
DATE_FORMAT	June 6 2021
DATETIME_FORMAT	June 6 2021 10 a.m.
SHORT_DATE_FORMAT	21/06/2021
SHORT_DATETIME_FORMAT	21/06/2021 10 a.m.
TIME_FORMAT	10:30

Ex: {{ value|date:" DATETIME_FORMAT" }}

Ex: {{ value|time:"TIME_FORMAT" }}

Views.py:

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

```
# Create your views here:
```

```
def date_time(request):
    dt=datetime.now()
    return render(request, 'index.html', {'dt':dt})
```

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Date and time formats</title>
</head>
<body>
<h1>{{ dt }}</h1>
<h1>{{ dt|date:'DATE_FORMAT' }}</h1>
<h1>{{ dt|date:'DATETIME_FORMAT' }}</h1>
<h1>{{ dt|date:'SHORT_DATE_FORMAT' }}</h1>
<h1>{{ dt|date:'SHORT_DATETIME_FORMAT' }}</h1>
<h1>{{ dt|time:'TIME_FORMAT' }}</h1>
</body>
</html>
```

Template Inheritance using static files:

Sample.css of static folder:

```
body
{
    background-color: red;
}
```

base.html:

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```

        <title></title>
        <link href="{% static 'css/sample.css' %}"
rel="stylesheet">
</head>
<body>
{% block content %}

{% endblock %}
</body>
</html>

```

Index.html:

```

<!DOCTYPE html>
{% extends 'base.html' %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
{% block content %}
<h1>{{ dt }}</h1>
<h1>{{ dt|date:'DATE_FORMAT' }}</h1>
<h1>{{ dt|date:'DATETIME_FORMAT' }}</h1>
<h1>{{ dt|date:'SHORT_DATE_FORMAT' }}</h1>
<h1>{{ dt|date:'SHORT_DATETIME_FORMAT' }}</h1>
<h1>{{ dt|time:'TIME_FORMAT' }}</h1>
{% endblock %}
</body>
</html>

```

If:

Syntax: {% if variable %}

Code....

```
{% endif %}
```

Ex1:

Views.py :

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

# Create your views here:
def v1(request):
    return
render(request, 'index.html', {'value': 'mohan'})
```

urls.py :

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.v1),
]
```

index.html :

```
<!DOCTYPE html>

<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
{% if value %}
    <h1>Hello:{{ value }}</h1>
{% endif %}
</body>
</html>
```


Ex2:

Views.py :

```
def v1(request):
    return
render(request, 'index.html', {'name': 'mohan', 'age': 37})
```

index.html:

```
{% if name and age %}
    <h1>{{ name }} age is:{{ age }}</h1>
{% endif %}
```

Ex3:

Index.html:

```
{% if name or age %}
    <h1>{{ name }} age is:{{ age }}</h1>
{% endif %}
```

Ex4:

Views.py :

```
def v1(request):
    return render(request, 'index.html')
```

Index.html:

```
{% if not name %}
    <h1>Name is not available</h1>
{% endif %}
```

Ex5:

Views.py:

```
def v1(request):
    return
render(request, 'index.html', {'name': 'mohan'})
```

index.html:

```
{% if name == 'mohan' %}
    <h1>Hello:{{ name }}</h1>
{% endif %}
```

Ex6:

Views.py:

```
def v1(request):
    return
render(request, 'index.html', {'name': 'mohan', 'age': 37})
```

index.html:

```
{% if name == 'mohan' and age == 37 %}
    <h1>{{ name }} age is {{ age }}</h1>
{% endif %}
```

If else:

Syntax: {% if variable %}

Code....

{% else %}

Code...

```
{% endif %}
```

Ex7:

Views.py:

```
def v1(request):  
    return  
render(request, 'index.html', {'name': 'mohan'})
```

index.html:

```
{% if name %}  
    <h1>Hello:{{ name }} </h1>  
    {% else %}  
    <h1>Name is not present</h1>  
{% endif %}
```

Ex8:

Views.py :

```
def v1(request):  
    return render(request, 'index.html', {'num': 10})
```

index.html:

```
{% if num == 1 %}  
    <h1>One </h1>  
    {% elif num == 2 %}  
    <h1>Two</h1>
```

```

    {% else %}
    <h1>Invalid num</h1>
{% endif %}

```

For loop:

Syntax:

```

{% for variable in variables %}

    {{ variable }}

{% endfor %}

```

Ex:

Views.py:

```

def v1(request):
    names={'names': ["sai", "mohan", "durga", "ram"]}
    return
render(request, 'index.html', context=names)

```

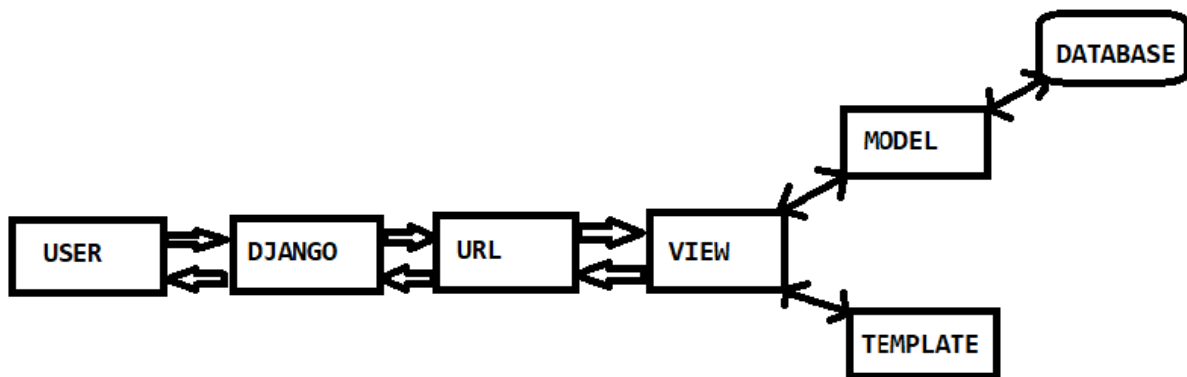
index.html:

```

<ul>
{% for name in names %}
    <li>{{ name }}</li>
{% endfor %}
</ul>
</body>

```

MVT Architecture :(Model View Template):



- The MVT (Model View Template) is a software design pattern. It is a collection of three important components Model View and Template.
- The Model helps to handle database. It is a data access layer which handles the data.
- The Template is a presentation layer which handles User Interface part completely.
- The View is used to execute the business logic and interact with a model to carry data and renders a template.
- There is no separate controller and complete application is based on Model View and Template. Control is handled by the Django framework itself.
- Here user requests for a resource to the Django, Django works as a controller and check to the available resource in URL.
- If URL maps, a view is called that interact with model and template, it renders a template.
- Django responds back to the user and sends a template as a response.

Working with models and databases:

- As the part of web application development, compulsory we required to interact with database to store our data and retrieve the data.
- Django provides a built-in support for database operations.
- Django will provide in built database sqlite3.
- For small and medium level applications this database is enough.

- Django provide support for other databases like my sql, oracle ,ms sql.

ORM(Object Relational Mapper):

- It enables the application to interact with database such as sqlite3,my sql, oracle.
- ORM will create automatically database schema from defined model classes.
- It will generate sql statements from python code(class).

Model:

- A model is a python class which contains database information.
- A model is a single, definitive source of information about our data.
- A model contains fields.
- Each model maps to one database table.
- Every model is a python class which is the child class of `django.db.models.Model`
- Each attribute of a model represents a database field.
- We have to write all model classes inside `models.py` file.

Creating a model:

- `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)

Ex:

Models.py :

```
from django.db import models
```

```
# Create your models here.
```

```
class Employee(models.Model):
    eno=models.IntegerField()
    ename=models.CharField(max_length=20)
    esal=models.IntegerField()
    eaddress=models.CharField(max_length=30)
```

- This class will create a table with columns and their data types
- Table Name will be ApplicationName_ClassName, in this case it will be sampleapp_Employee
- Field name will become table's Column Name, in this case it will be eno, ename, esal, eaddress with their data type.
- If we have not mentioned primary key in any of these columns so that it will automatically create a new column named 'id' Data Type Integer with primary key and auto increment.

Migrations:

- Migrations are Django's way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema.
- **makemigrations**: it is responsible for creating new migrations based on your models.
- **migrate**: this is responsible for applying migrations.
- **sqlmigrate**: This is going to display sql statements for migrations.
- **showmigrations**: this will show list of migrations and their status.

Note: Once you have defined your models, you need to tell Django you're going to use those models.

- Go to settings.py file
- Include application name in settings.py file under installed apps.
- Go to Terminal
- **Run python manage.py makemigrations**

This will create sql statements with migration file name like 0001_intial.py.

- **Run python manage.py migrate**

This will create a table in database.

To display sql statements:

Syntax:

```
python manage.py sqlmigrate application_name dbfile_name
```

Go to terminal and type: `python manage.py sqlmigrate sampleapp 0001`

How to check created table in Django admin panel:

Go to admin.py file under the sampleapp

admin.py:

```
from django.contrib import admin
from sampleapp.models import Employee

# Register your models here.
admin.site.register(Employee)
```

Creating super user for login to admin panel:

- Go to terminal and type the following command
- `Python manage.py createsuperuser`
- Provide username ,mailid, password
- Start server : `python manage.py runserver`
- Open browser and paste : <http://127.0.0.1:8000/admin/>
- Now enter username and password.

Steps to work with model:

- Create Project: `django-admin startproject sampleproject`

- Create Application: `python manage.py startapp samplapp`
- Add Application to Project using `settings.py` `INSTALLED_APPS`
- Open `models.py` file from application
- Create Model Class
- Go to terminal and type: `python manage.py makemigrations`
- migration file will be generate automatically inside **migrations** folder
- Type : `python manage.py migrate`
- Database table will be created automatically
- Create View Function inside `views.py` file
- Define URL for view using `urls.py` file under application
- Write Template files code
- Then run server : `python manage.py runserver`

How to display table data to user in template:

- Go to Admin panel: <http://127.0.0.1:8000/admin/> by entering username and password.
- Add few records and then save it.
- We have to write code in `views.py` for getting data from database.
- We have to display data in template by using render function.
- **all ()** : It is a method and it will return a copy of current query set.
- Syntax : `Modelclassname.objects.all()`

Note: Instead of admin panel we can also use a tool DB Browser for SQLite for inserting and manage the data.

Use this link for download and install DB Browser for SQLite

<https://sqlitebrowser.org/>

Go to download and click on below option

- **DB Browser for SQLite - Standard installer for 64-bit Windows**

views.py:

```
from django.shortcuts import render
from django.http import HttpResponse
from sampleapp.models import Employee

# Create your views here.
def displayemp(request):
    emp = Employee.objects.all()
    return render(request, 'home.html', {'emp':emp})
```

home.html :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Employee Data</h1>
    {% if emp %}
        <h1>Display Data</h1>
        {% for e in emp %}
            <h3>{{e.eno}}</h3>
            <h3>{{e.ename}}</h3>
            <h3>{{e.esal}}</h3>
            <h3>{{e.eaddress}}</h3>
        {% endfor %}
    {% else %}
        <h1>No Data</h1>
    {% endif %}
```

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

urls.py : (application level)

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

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

To display data in table format:

home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Employee Details</h1>
{% if emp %}
<h1>Display Data</h1>
    <table border="1">
        <thead>
            <th>Eno</th>
            <th>Ename</th>
            <th>Eaddress</th>
            <th>Esalary</th>
        </thead>
        {% for e in emp %}
            <tr>
                <td>{{ e.eno }}</td>
```

```

        <td>{{ e.ename }}</td>
        <td>{{ e.esal }}</td>
        <td>{{ e.eaddress }}</td>
{% endfor %}
    </table>
    {% else %}
    <h1>No Data</h1>
{% endif %}
</body>
</html>

```

How to display particular record:

views.py:

```

from django.shortcuts import render
from django.http import HttpResponse
from sampleapp.models import Employee

# Create your views here.
def displayemp(request):
    emp=Employee.objects.get(pk=1)
    return render(request, 'home.html', {'emp':emp})

```

home.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Employee Details</h1>
{% if emp %}
    {{ emp.eno }}
    {{ emp.ename }}

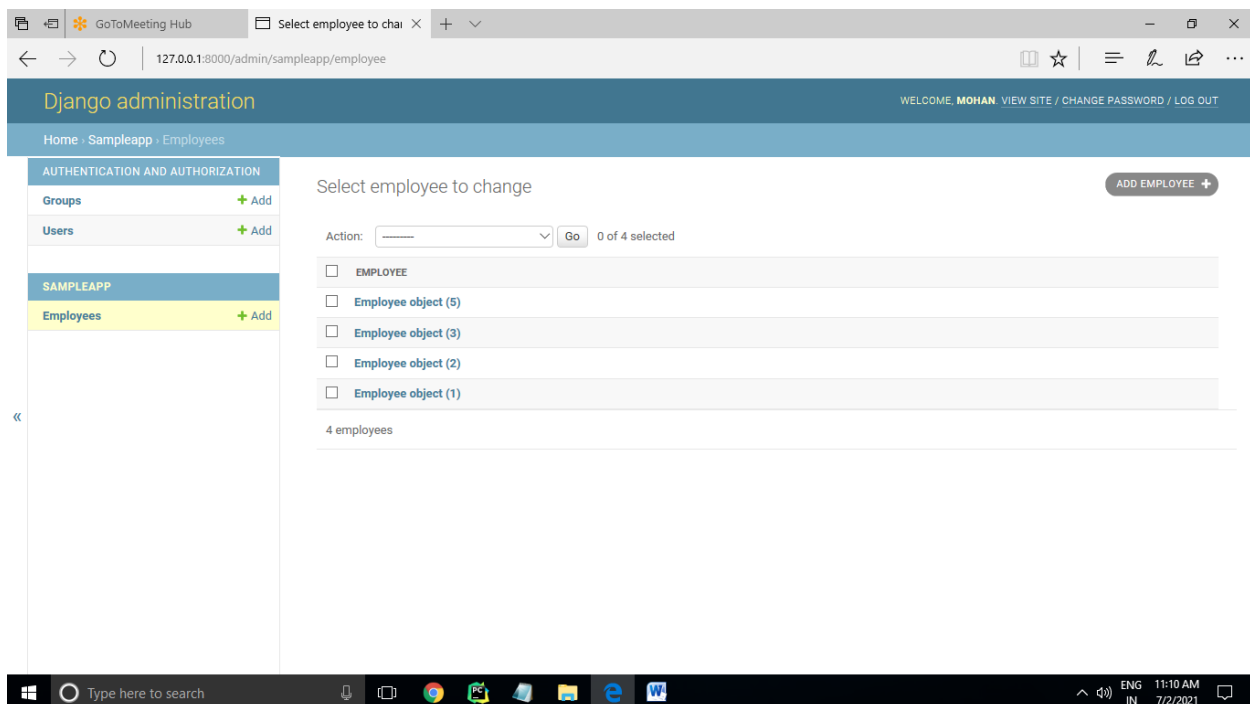
```

```

{{ emp.esal }}
{{ emp.eaddress }}
{% else %}
<h1>No Data</h1>
{% endif %}
</body>
</html>

```

Go to admin panel : <http://127.0.0.1:8000/admin/>



Note: notice that in admin panel we have 4 records, as we know that every record is considered as one object, in this case we are not able to recognize employee records with their id or name, to achieve this then we use `__str__(self)` method.

- **`__str__(self)`** : It is a python method which is called when we use `print/str` to convert object into a string. It is predefined.
- **`str`** function in a django model returns a string that is exactly rendered as the display name of instances for that model.

Syntax:

```
def __str__(self):
    return self.fieldname
```

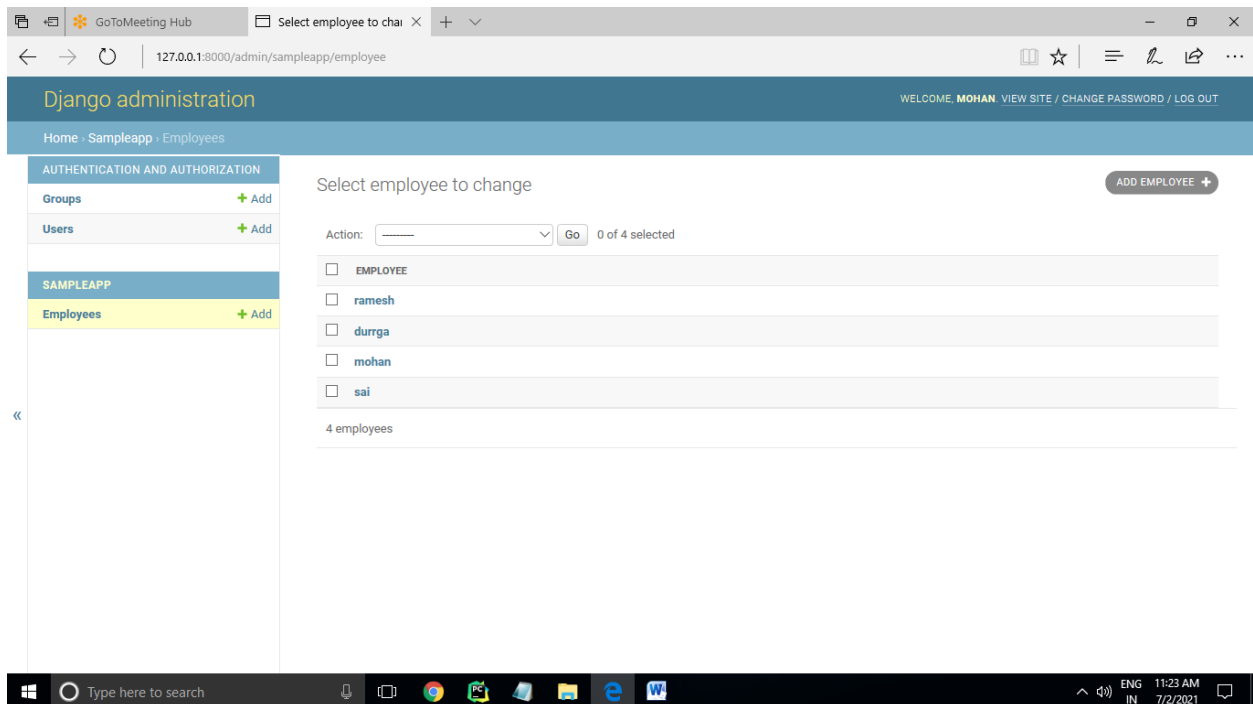
model.py :

```
from django.db import models

# Create your models here.
class Employee(models.Model):
    eno=models.IntegerField()
    ename=models.CharField(max_length=20)
    esal=models.IntegerField()
    eaddress=models.CharField(max_length=30)

    def __str__(self):
        return self.ename
```

Now we can see in admin panel, every record with their name.



To display with their eno :

```
def __str__(self):
    return str(self.eno)
```

How to display records as table structure in admin panel:

ModelAdmin :

- The ModelAdmin class is the representation of a model in the admin interface.
- To show table's all data in admin interface we have to create a ModelAdmin class in admin.py file of Application folder.

Syntax :

Class ModelAdminClassName(admin.ModelAdmin):

list_display=('field1', 'field2', 'field3'.....)

Register a Created Class :

admin.site.register(Modelclassname, ModeladminClassname)

- list_display is used to display all fields in admin panel as table format.

admin.py :

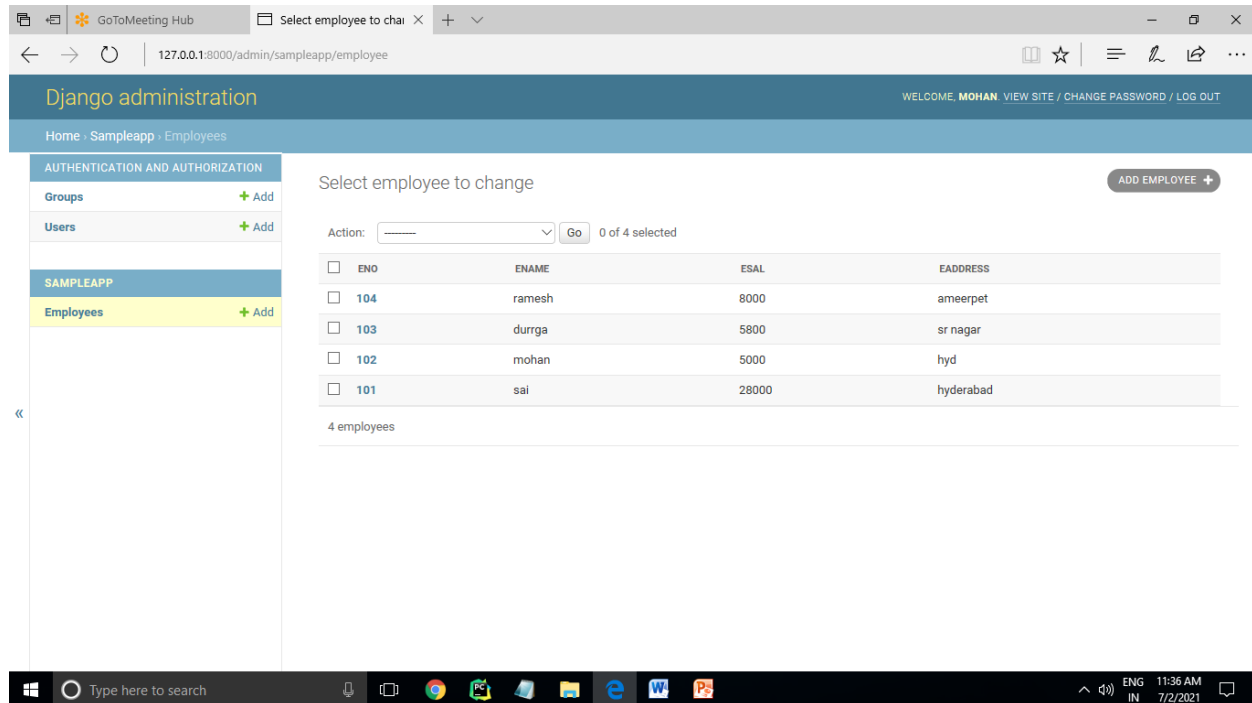
```
from django.contrib import admin
from sampleapp.models import Employee
```

```
class EmployeeAdmin(admin.ModelAdmin):
    list_display =
    ['eno', 'ename', 'esal', 'eaddress']
```

```
# Register your models here.
admin.site.register(Employee, EmployeeAdmin)
```

Now we can see in admin panel, records will display in table format.

<http://127.0.0.1:8000/admin/>



Register a model using decorator:

admin.py :

```
from django.contrib import admin
from sampleapp.models import Employee

@admin.register(Employee)
class EmployeeAdmin(admin.ModelAdmin):
    list_display =
    ['eno', 'ename', 'esal', 'eaddress']
# Register your models here.
```



```
#admin.site.register(Employee,EmployeeAdmin)
```

Django Forms:

- Django forms are very important in web development.
- The main purpose of Django forms is to take user inputs.
Ex : registration form, login form etc.
- From the forms we can read end user inputs and we can use the end user inputs based on requirement.
- We can store end user inputs in the form and store in database for further usage.
- we can also use form data for validation and authentication.
- Django provides Form class which is used to create HTML forms.
- Django Form class describes a form and how it works and appear.
- Each field of Form class map to the HTML form<input> element.

Advantages of Django Forms over HTML forms:

- We can develop forms very easily with python code
- We can generate HTML widgets like text area, email, password field etc. very quickly.
- Validation data with Django Forms is very easy.
- Creation of Models based on forms will be easy.

How to create Django Forms:

step1:

- To create Django Forms first we need to create new file inside the application folder with name forms.py .

forms.py:

```
from django import forms

class StudentForm(forms.Form):
```

```
name=forms.CharField()
marks=forms.IntegerField()
```

Note: Here name and marks are the field names which will be available in HTML form

step2: Use forms.py inside views.py

- views.py file is responsible to send this form to the html template file

views.py:

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from . import forms

# Create your views here.

def studentview(request):
    form=forms.StudentForm()
    myform = {'form': form}
    return
render(request, 'home.html', context=myform)
```

home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    {{ form }}
</body>
</html>
```

Now run server in chrome browser, right click on page, view page source, the following code is generated.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>
<tr><th><label
for="id_name">Name:</label></th><td><input
type="text" name="name" required
id="id_name"></td></tr>
<tr><th><label
for="id_marks">Marks:</label></th><td><input
type="number" name="marks" required
id="id_marks"></td></tr>
</body>
</html>
```

Note: it will add only form fields; it will not add <form> tag and button so we have to add manually.

home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<form method="get">
  <table>
    {{ form }}
```

```

        </table>
        <input type="submit" value="submit">
</form>
</body>
</html>

```

Form rendering options:

1. {{ form }}
2. {{ form.as_table }}
3. {{ form.as_p }}
4. {{ form.as_ul }}
5. {{ form.name }}
{{ form.marks }}

Ex:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    {{ form }}
    <hr>
    {{ form.as_table }}
    <hr>
    {{ form.as_p }}
    <hr>
    {{ form.as_ul }}
    <hr>
    {{ form.name }}
    {{ form.marks }}
</body>
</html>

```

Order_fields ():

- It is used to display the form fields in order what the way you want.
- It is used to re-arrange the fields.

Syntax: formobject.order_fields (field_order=['field1','field2','field3'])

Ex:

forms.py :

```
from django import forms

class StudentForm(forms.Form):
    name=forms.CharField()
    marks=forms.IntegerField()
    mailid=forms.EmailField()
    address=forms.CharField()
```

views.py :

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

# Create your views here.
def studentview(request):
    form=forms.StudentForm()

    form.order_fields(field_order=['name', 'address', 'mailid', 'marks'])
    myform = {'form': form}
    return render(request, 'home.html', context=myform)
```

home.html :

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form method="get">
    <table>
        {{ form.as_p }}
    </table>
    <input type="submit" value="submit">
</form>
</body>
</html>

```

Form fields arguments:

Try these all arguments with form fields in forms.py file

Ex1: `class StudentForm(forms.Form):`
`name=forms.CharField(initial='mohan')`

Ex2: `class StudentForm(forms.Form):`
`name=forms.CharField(label_suffix=" ")`

Ex3: `class StudentForm(forms.Form):`
`name=forms.CharField(required=False)`

Ex4: `class StudentForm(forms.Form):`
`name=forms.CharField(disabled=True)`

Ex5: `class StudentForm(forms.Form) :`

```
name=forms.CharField(widget=forms.Textarea,help_text="only 150 character")
```

Ex6: `class StudentForm(forms.Form) :`

```
    name=forms.CharField(widget=forms.Textarea)
    name =
forms.CharField(widget=forms.PasswordInput)
    name = forms.CharField(widget=forms.FileInput)
    name =
forms.CharField(widget=forms.CheckboxInput)
```

Working with Form using POST method:

- 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 and POST** are the only HTTP methods to use when dealing with forms.
- **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.

Ex:

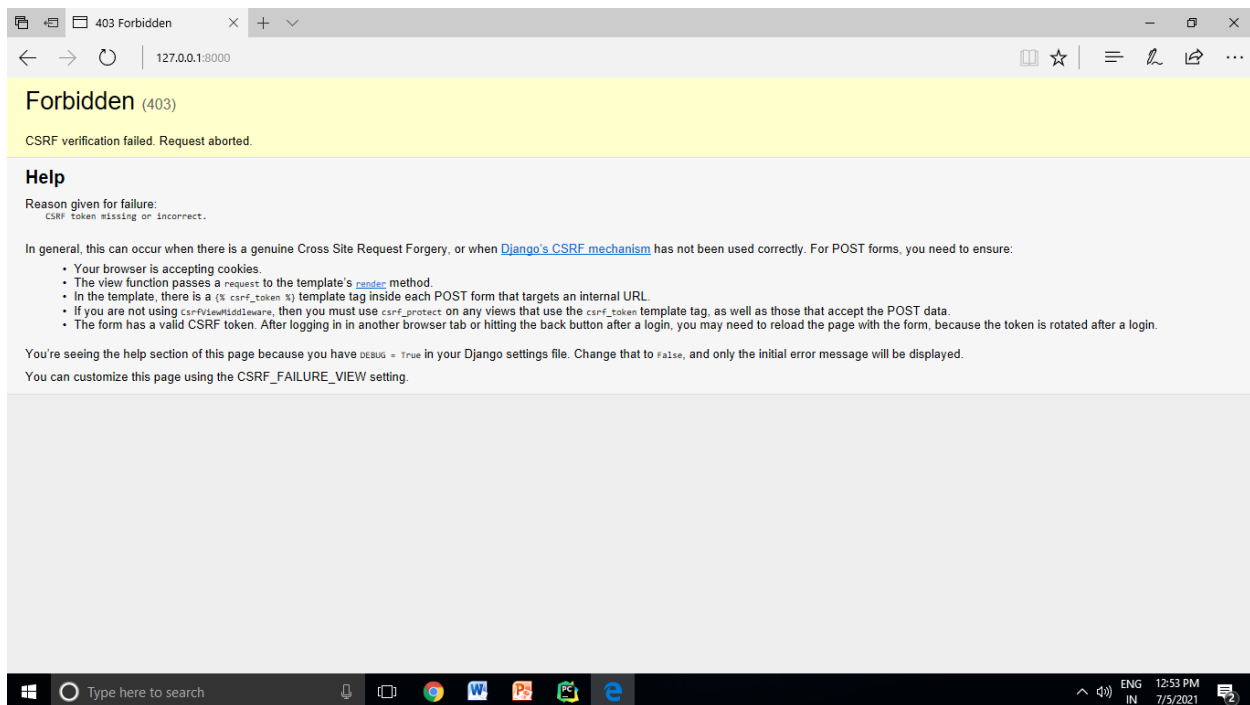
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
```

```

<form method="post">
  <table>
    {{ form.as_p }}
  </table>
  <input type="submit" value="submit">
</form>
</body>
</html>

```

Run server and enter the data in textboxes then click on submit, you will get 403 error.



CSRF (Cross Site Request Forgery):

- Every form should satisfy CSRF verification; otherwise Django will not accept our form.
- It means for website security, being a programmer we need not to do anything for this, django will takes care.

- But we have to add csrf_token in our form.

home.html :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form method="post">
    <table>
        {{ form.as_p }}
        {% csrf_token %}
    </table>
    <input type="submit" value="submit">
</form>
</body>
</html>
```

Note: if we add csrf_token then it will add hidden field which makes our post request is secure.

- Run server and enter the data in textboxes then click on submit.
- Right click on the page ,click on view page source.

```
<input type="hidden" name="csrfmiddlewaretoken"
value="Unfkm9eZ1ylCVDTaMSrx1F138QblNGHvJnVtAfJla9Qlmq
r3I4d1m8j4QHe2SEbL">
```

- The value of hidden field is keep on changing from request to request so that it is impossible to forgery of our request.

Process input data from the form inside views.py file:

- First we have to validate the data and then get cleaned data.

cleaned_data:

- This is used to access clean data, we can access the clean data using cleaned_data attribute.
- cleaned_data is containing always a key for fields defined in the Form.
- Once the Form is valid, cleaned_data will include a key and value for all its fields.

is_valid() :

- This method is used to run validation and return a Boolean, like True means valid and false means invalid.

Ex:

forms.py :

```
from django import forms

class StudentForm(forms.Form):
    name=forms.CharField()
    marks=forms.IntegerField()
```

views.py:

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

# Create your views here.
def studentinputview(request):
    if request.method == 'POST':
        form=StudentForm(request.POST)
        if form.is_valid():
            print("Form validation is success and
printing the data")
            print("Student
name:", form.cleaned_data['name'])
```

```

        print("Student
marks:", form.cleaned_data['marks'])
    else:
        form=StudentForm()
    return render(request, 'home.html', {'form':
form})

```

home.html :

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form method="POST">
<table border="1" style="background-color:
darkcyan">
{{ form.as_table }}
{% csrf_token %}

    </table>
<input type="submit" value="submit">

</form>
</body>
</html>

```

How to check password and Retype password same or not:

forms.py:

```

from django import forms

class StudentForm(forms.Form):
    name=forms.CharField()
    mailid=forms.EmailField()

password=forms.CharField(widget=forms.PasswordInput

```

```
)
```

```
retypepassword=forms.CharField(widget=forms.PasswordInput)
```

```
def clean(self):
    super().clean()
    pwd=self.cleaned_data['password']
    rpwd=self.cleaned_data['retypepassword']
    if pwd!=rpwd:
        raise forms.ValidationError("password
mismatch")
```

views.py:

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

# Create your views here.
def studentinputview(request):
    if request.method == 'POST':
        form=StudentForm(request.POST)
        if form.is_valid():
            print("Form validation is success and
printing the data")

print("name:",form.cleaned_data['name'])

print("mailid:",form.cleaned_data['mailid'])
            print("password:",
form.cleaned_data['password'])
            print("retypepassword:",
form.cleaned_data['retypepassword'])
        else:
            form=StudentForm()
            return render(request, 'home.html', {'form':
form})
```

home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

<form method="POST">
<table border="1" style="background-color:
darkcyan">
{{ form.as_table }}
{% csrf_token %}

    </table>
<input type="submit" value="submit">

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

How to validating a specific field:

forms.py:

```
from django import forms

class StudentForm(forms.Form):
    name=forms.CharField()
    mailid=forms.EmailField()

password=forms.CharField(widget=forms.PasswordInput
)

    def clean_name(self):
        name = self.cleaned_data['name']
        #name = self.cleaned_data.get('name')
```

```

        if len(name) < 6:
            raise forms.ValidationError("Enter more
than or equal 6")
        return name

```

views.py:

```

from django.shortcuts import render
from sampleapp.forms import StudentForm

# Create your views here.
def studentinputview(request):
    if request.method == 'POST':
        form=StudentForm(request.POST)
        if form.is_valid():
            print("Form validation is success and
printing the data")

print("name:", form.cleaned_data['name'])

print("mailid:", form.cleaned_data['mailid'])
            print("password:",
form.cleaned_data['password'])
        else:
            form=StudentForm()
            return render(request, 'home.html', {'form':
form})

```

home.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form method="POST">
    <table border="1" style="background-color:

```

```

darkcyan">
{{ form.as_table }}
{% csrf_token %}
    </table>
<input type="submit" value="submit">
</form>
</body>
</html>

```

Built-in Validators:

- We can use Built-in Validators which are available in django.core module.

forms.py:

```

from django.core import validators
from django import forms
class StudentForm(forms.Form):

name=forms.CharField(validators=[validators.MaxLengthValidator(10)])
    mailid=forms.EmailField()

```

views.py:

```

from django.shortcuts import render
from sampleapp.forms import StudentForm

# Create your views here.
def studentinputview(request):
    if request.method == 'POST':
        form=StudentForm(request.POST)
        if form.is_valid():
            print("Form validation is success and
printing the data")

print("name:", form.cleaned_data['name'])

```

```

print("mailid:", form.cleaned_data['mailid'])
    else:
        form=StudentForm()
    return render(request, 'home.html', {'form':
form})

```

home.html:

```

<!DOCTYPE html>
<html lang="en">
<head>

    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form method="POST">
    <table border="1" style="background-
color:lightseagreen">
{{ form.as_table }}
{% csrf_token %}
    </table>
<input type="submit" value="submit">
</form>
</body>
</html>

```

Custom Form Validators:

forms.py:

```

from django.core import validators
from django import forms

```



```
def starts_with_D(value):
    if value[0] != 'D':
        raise forms.ValidationError('Name should
start with D')

class StudentForm(forms.Form):
    name=forms.CharField(validators=[starts_with_D])
    mailid=forms.EmailField()
```

views.py:

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

# Create your views here.
def studentinputview(request):
    if request.method == 'POST':
        form=StudentForm(request.POST)
        if form.is_valid():
            print("Form validation is success and
printing the data")

print("name:", form.cleaned_data['name'])

print("mailid:", form.cleaned_data['mailid'])
    else:
        form=StudentForm()
        return render(request, 'home.html', {'form':
form})
```

home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```

        <meta charset="UTF-8">
        <title>Title</title>
</head>
<body>
<form method="POST">
    <table border="1" style="background-
color:lightseagreen">
{{ form.as_table }}
{% csrf_token %}
    </table>
<input type="submit" value="submit">
</form>
</body>
</html>

```

How to save form data into data base (sqlite3):

Step1: go to forms.py file and write the following code

```

from django import forms

class StudentForm(forms.Form):

name=forms.CharField(error_messages={'required': 'En
ter name'})

mailid=forms.EmailField(error_messages={'required':
'Enter mailid'})

address=forms.CharField(error_messages={'required':
'Enter address'})

password=forms.CharField(widget=forms.PasswordInput
,error_messages={'required': 'Enter password'})

```

Step2: go to models.py file and write the following code

```
from django.db import models

# Create your models here.
class Student(models.Model):
    name=models.CharField(max_length=20)
    mailid=models.EmailField(max_length=30)
    address=models.CharField(max_length=20)
    password=models.CharField(max_length=10)
```

Step3: Register model class in admin.py file

```
from django.contrib import admin
from sampleapp.models import Student

@admin.register(Student)
class StudentAdmin(admin.ModelAdmin):
    list_display =
    ['name', 'mailid', 'address', 'password']
# Register your models here.
#admin.site.register(Employee,EmployeeAdmin)
```

Step4: create html template with the name home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form method="POST" novalidate>
    {% csrf_token %}
```

```

        <table border="1" style="background-color:
darkcyan">
            {{ form.as_table }}
        </table>
        <input type="submit" value="submit">
</form>
</body>
</html>

```

Step5: go to terminal and execute the following commands

- python manage.py makemigrations
- python manage.py migrate

Step6: go to views.py file and write the following code

```

from django.shortcuts import render
from sampleapp.forms import StudentForm
from sampleapp.models import Student

# Create your views here.
def studentinputview(request):
    if request.method == 'POST':
        form=StudentForm(request.POST)
        if form.is_valid():
            nm=form.cleaned_data['name']
            ml=form.cleaned_data['mailid']
            ad=form.cleaned_data['address']
            pw=form.cleaned_data['password']
            #for insert
            s = Student(name=nm, mailid=ml,
address=ad, password=pw)
            s.save()
            #for update

#s=Student(id=1,name=nm,mailid=ml,address=ad,passwo

```

```

rd=pw)
        #s.save()
        #for delete
        #s=Student(id=1)
        #s.delete()
    else:
        form=StudentForm()
    return render(request, 'home.html', {'form':
form})

```

Step7: go to terminal and create super user

- python manage.py createsuperuser
- Enter username, mailid and password.
- Run server : python manage.py runserver
- Enter the data into html form and submit
- Go to admin panel and check : <http://127.0.0.1:8000/admin/>

How to save form data into data base (sqlite3) using model Form:

Model Forms (Forms based on model):

- Sometimes we can create form, based on model, such type of forms is called model based forms or model forms.
- The main advantage of model forms is we can collect end user inputs and we can save that inputs very easily into database.
- Django will provide inbuilt support to develop model based forms very easily.
- Django provides a helper class that will allow you to create a form class from django model and that helper class is called as Model Form.

Syntax:

```
class ModelFormsclassname(forms.ModelForm):
```

```
    Class Meta:
```

```
        model=ModelClassname()
```

```
        fields=["field1","field2","field3"]
```

Step1: go to models.py file and write the following code

```
from django.db import models

# Create your models here.
class User(models.Model):
    name=models.CharField(max_length=20)
    password=models.CharField(max_length=30)
    address=models.CharField(max_length=20)
```

Step2: go to forms.py file and write the following code

```
from django import forms
from sampleapp.models import User

class UserForm(forms.ModelForm):
    class Meta:
        model=User
        fields='__all__'
        #fields=['name','password','address']
        #exclude=['address']
        labels={'name':'Enter name','password':'Enter
password','address':'Enter address'}
        widgets={'password':forms.PasswordInput}
        error_messages={'name':{'required':'name is
mandatory'}},
```

```
'password': {'required': 'password must'}}
```

Step3: Register model class in admin.py file

```
from django.contrib import admin
from sampleapp.models import User

@admin.register(User)
class UserAdmin(admin.ModelAdmin):
    list_display = ['name', 'password', 'address']
# Register your models here.
#admin.site.register(Employee, EmployeeAdmin)
```

Step4: create html template with the name home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form method="POST" novalidate>

    <table border="1" style="background-color:
darkcyan">
        {{ form.as_table }}
    </table>
    {% csrf_token %}
    <input type="submit" value="submit">
</form>
</body>
</html>
```

Step5: go to terminal and execute the following commands

- python manage.py makemigrations
- python manage.py migrate

Step6: go to views.py file and write the following code

```
from django.shortcuts import render
from sampleapp.forms import UserForm
from sampleapp.models import User

# Create your views here.
def studentinputview(request):
    if request.method == 'POST':
        form=UserForm(request.POST)
        if form.is_valid():
            nm=form.cleaned_data['name']
            pw = form.cleaned_data['password']
            ad=form.cleaned_data['address']
            s =
User (name=nm,password=pw,address=ad)
            s.save()
        else:
            form=UserForm()
    return render(request, 'home.html', {'form':
form})
```

Step7: go to terminal and create super user

- python manage.py createsuperuser
- Enter username, mailid and password.
- Run server : python manage.py runserver

- Enter the data into html form and submit
- Go to admin panel and check : <http://127.0.0.1:8000/admin/>

Dynamic URL:

- A **dynamic URL** is the address - or Uniform Resource Locator (**URL**) - of a Web page with content that depends on variable parameters that are provided to the server that delivers it. The parameters may be already present in the **URL** itself or they may be the result of user input.
- **Django** offers a way to **name urls** so it's easy to reference them in view methods and templates.
- The most basic technique to **name Django urls** is to add the **name** attribute to **url** definitions in **urls.py**.

Ex1:

Urls.py (project level):

```
from django.contrib import admin
from django.urls import path
from myapp import views

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

    path('employee/<emp_id>', views.emp_details, name
    = 'details'),
]
```

Views.py:

```
def emp_details(request, emp_id):
    emp = {'id': emp_id}
    return
render(request, 'display.html', context=emp)
```

display.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Display Employee Template</h1>
<h1>{{ id }}</h1>>
</body>
</html>

```

Ex2:

Urls.py (project level):

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),

    path('employee/<emp_id>', views.emp_details, name='de
tails')
]

```

views.py:

```

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

def emp_details(request, emp_id):
    emp={'id':emp_id}
    return
render(request, 'display.html', context=emp)

```

display.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Display Employee Template</h1>
<h1>{{ id }}</h1>
<a href="{% url 'home' %}">Back to home</a>
</body>
</html>
```

Home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Home Page</h1>
<a href="{% url 'details' 1 %}">Employee 1</a><br>
<a href="{% url 'details' 2 %}">Employee 2</a><br>
<a href="{% url 'details' 3 %}">Employee 3</a><br>
<a href="{% url 'details' 4 %}">Employee 4</a><br>
</body>
</html>
```

Ex3:

Urls.py (project level):

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),

    path('employee/<int:emp_id>', views.emp_details, name
    ='details')
]

```

views.py:

```

from django.shortcuts import render

# Create your views here.

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

def emp_details(request, emp_id):
    if emp_id == 1:
        emp={'id':emp_id, 'name':"mohan"}
    if emp_id == 2:
        emp = {id: emp_id, 'name':"durga"}
    if emp_id == 3:
        emp = {id: emp_id, 'name':"sai"}
    if emp_id == 4:
        emp = {'id': emp_id, 'name':"manoj"}
    return render(request, 'display.html', emp)

```

display.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

```

```

        <title>Title</title>
</head>
<body>
<h1>Display Employee Template</h1>
<h1>{{ id }}</h1>
<h1>{{ name }}</h1>
<a href="{% url 'home' %}">Back to home</a>
</body>
</html>

```

Home.html :

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Home Page</h1>
<a href="{% url 'details' 1 %}">Employee 1</a><br>
<a href="{% url 'details' 2 %}">Employee 2</a><br>
<a href="{% url 'details' 3 %}">Employee 3</a><br>
<a href="{% url 'details' 4 %}">Employee 4</a><br>
</body>
</html>

```

Django Messages:

- The Django messages allow you to store messages in one request and retrieve them for display in a subsequent request.
- To work with message framework we need to use `django.contrib.messages`
- **Syntax:** `add_message(request, level, message)`

Ex:

models.py :

```

from django.db import models

# Create your models here.

class customer(models.Model):
    name=models.CharField(max_length=20)
    address = models.CharField(max_length=20)
    mailid = models.EmailField(max_length=20)

```

forms.py:

```

from django import forms
from .models import customer

class customerModel(forms.ModelForm):
    class Meta:
        model=customer
        fields='__all__'

```

admin.py:

```

from django.contrib import admin
from .models import customer

# Register your models here.
@admin.register(customer)
class customerAdmin(admin.ModelAdmin):
    list_display = ['name', 'address', 'mailid']

```

views.py:

```

from django.shortcuts import render
from .forms import customerModel
from django.contrib import messages

# Create your views here.

```

```

def msg(request):
    if request.method == 'POST':
        form=customerModel(request.POST)
        if form.is_valid():
            form.save()

messages.add_message(request,messages.SUCCESS,"Record inserted sucessfully...")
            messages.info(request,"Inserted,Now you can check in database")
        else:
            form=customerModel()
    return
render(request,'display.html',{'form':form})

```

display.html :

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="SUBMIT">
</form>
{% if messages %}
    {% for message in messages %}
        {{ message }}
    {% endfor %}
{% endif %}
</body>
</html>

```

urls.py (project level) :

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.msg),
]

```

➤ Go to terminal and execute the following commands

- Python manage.py makemigrations
- Python manage.py migrate
- Python manage.py createsuperuser
- Python manage.py runserver.

State Management:

In general when we send request for the page to server, server will process and send response to client, once response is delivered then server will destroy the entire page information.

- State management is the process of maintaining the state of user or webpage.
- User state is nothing but user entered values like username, password.
- Web page state is nothing but complete web page information can hold for particular period of time.
- State management is the process of maintaining the state of values between multiple requests of the webpage or webpages.
- To maintain state of values, we have different types of options where the values can be maintain either client side or server side.
- Client and server can communicate with some language is called HTTP.
- The basic limitation of HTTP is, it is a stateless protocol.
- Http is unable to remember client information for future purpose across multiple requests, every request to the server is a new request.

- Some mechanism is required to remember client information that is state management techniques.

State Management Techniques:

- Cookies
- Session
- URL Rewriting
- Hidden Form Fields
- Cache

Cookies:

- Cookie is a client side state management technique.
- Cookie is a small amount of information created by server and maintained by client.
- Using cookie we can transfer the data from one webpage to another.
- It is also used to maintain state of values.
- Cookie can store max of 4096 bytes of information or 4 kb information.
- Cookie will store the data in plain text format i.e. string format.
- Once cookie is create, it will travel always in between client to server.
- Cookie is domain based; every cookie will know for which website it has been create.
- For single website, single bowser will allow maximum 20 cookies.
- If 21st cookie is creating then oldest cookie will delete automatically.
- For multiple websites, multiple browsers will allow maximum 200 cookies.
- If 201st cookie created then oldest cookie will be destroy.

Types of cookies:

1. In-memory cookie
2. Persistent cookie

In memory cookie:

- These cookies will store in browser memory, as soon as we close browser Window of the website these cookies will be destroy.

- These cookies don't have any expiration date and time.

Persistent cookie:

- These cookies are having expiration date and time.
- Even when we close the browser window of the website, these cookies will remain stored in browser memory.
- Once date and time expire, these cookies will be delete automatically.

Drawback of cookies:

- It will store the data in text format so that no security.
- Anyone can see the cookie data and they can be able to delete cookies.
- Limited capacity.

Creating cookies:

- **set_cookie ():** This method is used to set/create/sent cookies.
- To create cookies we use Httpcookie class.
- **syntax:** Httpcookie.set_cookie(key, value)
- **key:** This is the name of the cookie.
- **value:** This will sets the value of cookie, This value is stored on the clients computer.
- Name and value are required to set cookie.

Reading cookies:

- **Syntax :** request.COOKIES['key'];

Ex:

Views.py:

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

# Create your views here.

def setcookie(request):
```

```

response=HttpResponse("Cookie is set")
response.set_cookie("name", "mohan")
return response

def getcookie(request):
    nm=request.COOKIES["name"]
    return HttpResponse("your name is:"+nm)

```

urls.py :(project level):

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('set', views.setcookie),
    path('get', views.getcookie),
]

```

Steps to check cookies in browser:

Go to browser settings (Google chrome) ---- click on privacy and security ---- click on site settings---click on cookies and site data---- click on see all cookies and site data---click on 127.0.0.1 cookies and expand.

Ex with persistent cookie:

Views.py:

```

from django.shortcuts import render
from django.http import HttpResponse
from datetime import datetime, timedelta

# Create your views here.

def setcookie(request):
    response=render(request, 'set.html')

```

```

        #response.set_cookie("name","mohan",max_age=60)

response.set_cookie("name","mohan",expires=datetime
.utcnow()+timedelta(days=3))
return response

def getcookie(request):
    #nm=request.COOKIES["name"]
    nm=request.COOKIES.get("name","cookie deleted")
    return render(request,'get.html',{'name':nm})

def deletecookie(request):
    response=render(request,'del.html')
    response.delete_cookie('name')
    return respons

```

urls.py: (project level):

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('set',views.setcookie),
    path('get',views.getcookie),
    path('del',views.deletecookie),

]

```

set.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>

```

```
</head>
<body>
<h1>Cookie is set</h1>
</body>
</html>
```

get.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>Your name is:{{ name }}</h1>
</body>
</html>
```

del.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>Cookie is deleted</h1>
</body>
</html>
```

Note: when we change the cookie value, it will be override, when we change the cookie key then it will create a new cookie.

Ex with page count using cookie:

Views.py:

```

from django.shortcuts import render
from django.http import HttpResponseRedirect

def count_view(request):
    if 'count' in request.COOKIES:
        newcount=int(request.COOKIES['count'])+1
    else:
        newcount=1

    response=render(request, 'count.html', {'count':newcount})
    response.set_cookie('count', newcount)
    return response

```

urls.py: (project level):

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('count', views.count_view),
]

```

count.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>page count is:{{ count }}</h1>

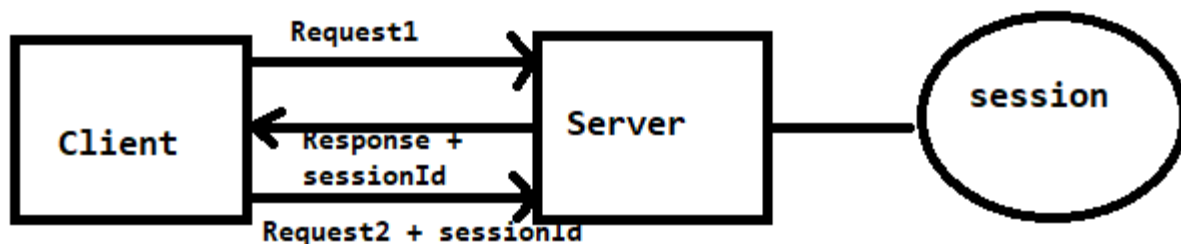
```

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

Note: Every time count will increment when we refresh the webpage.

Session:

- Session is a server side state management technique.
- It stores data on the server side and sends session id to cookies.
- Cookies contain a session ID not the session data.
- By default, Django stores sessions in database.
- As it stores sessions in database so it is mandatory to run makemigrations and migrate to use session. It will create required tables.
- The Django sessions are completely cookie-based.
- **Note: The Default time period of session is 14 days(2 weeks)**



- Once client sends request to the server, if server wants to remember client information for the future purpose then server will create a session object and store required information in that object.
- For every session object is having unique identifier which is nothing but session ID.
- Server sends the session ID to the client as response.
- Client retrieves the session ID from the response and saves.
- By accessing that session ID server can remember client request.

Session Methods:

- **To add data to the session**

```
request.session["key"]=value
```

- **To get data from session**
value=request.session["key"]
- **To set the expiry time for session**
request.session.set_expiry(seconds)
- **To get the session expiry age in seconds**
request.session.get_expiry_age()
- **To get the session expiry date**
request.session.get_expiry_date()

Ex:

Views.py:

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

# Create your views here.
def setsession(request):
    request.session['name']='durgasoft'
    return render(request, 'set.html')

def getsession(request):
    #name=request.session['name']
    name=request.session.get('name', default="no
name")
    return render(request, 'get.html', {'name':name})

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


urls.py:

```
from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('set', views.setsession),
    path('get', views.getsession),
    path('del', views.deletesession),
]
```

set.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Session is set</h1>
</body>
</html>
```

get.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
```

```
<h1>Your name is:{{ name }}</h1>
</body>
</html>
```

del.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Session is deleted</h1>
</body>
</html>
```

Session Methods:

- **flush ():** It deletes the current session data from the session and deletes the session cookie.
- **get_expire_at_browser_close ():** It returns either True or False, depending on whether the user's session cookie will expire when the user's Web browser is closed.
- **clear_expired ():** It removes expired sessions from the session store.
- **set_test_cookie ():** It sets a test cookie to determine whether the user's browser supports cookies.
- **test_cookie_worked ():** It returns either True or False, depending on whether the user's browser accepted the test cookie.
- **delete_test_cookie ():** It deletes the test cookie.

Session settings:

- **SESSION_COOKIE_AGE:** The age of session cookies, in seconds.
Default: 1209600 (2 weeks, in seconds)

- **SESSION_COOKIE_NAME:** The name of the cookie to use for sessions.
Default is: sessionid
- **SESSION_COOKIE_PATH:** The path set on the session cookie.

Ex 1:**Views.py:**

```

from django.shortcuts import render
from django.http import HttpResponseRedirect

# Create your views here.
def setsession(request):
    request.session['name']='durgasoft'
    return render(request, 'set.html')

def getsession(request):
    name=request.session['name']

    #print(request.session.get_session_cookie_age())
    #print(request.session.get_expiry_date())
    #print(request.session.get_expiry_age())

    #print(request.session.get_expire_at_browser_close(
    ))

    return render(request, 'get.html', {'name':name})

def deletesession(request):
    request.session.flush()
    request.session.clear_expired()
    return render(request, 'del.html')

```

settings.py:

```

STATIC_URL = '/static/'
SESSION_COOKIE_AGE=500

```

```
SESSION_COOKIE_NAME="durgasession"
SESSION_COOKIE_PATH="/login"
```

Ex 2: How to check browser supports cookie or not

Views.py:

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

# Create your views here.
def setsession(request):
    request.session.set_test_cookie()
    return render(request, 'set.html')

def checksession(request):
    if request.session.test_cookie_worked():
        print("Cookies are working properly")
    return render(request, 'check.html')

def deletesession(request):
    request.session.delete_test_cookie()
    return render(request, 'del.html')
```

Ex 3: page count with session

Views.py:

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

# Create your views here.
def page_count_view(request):
    count=request.session.get('count',0)
    newcount=count+1
    request.session['count']=newcount
    return
render(request, 'pagecount.html', {'count':count})
```

pagecount.html :

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        span{font-size: 300px}
    </style>
</head>
<body>
<h1>The page count is:<span>{{ count }}</span></h1>
</body>
</html>

```

Session Expired:**Views.py:**

```

from django.shortcuts import render,HttpResponse

# Create your views here.
def setsession(request):
    request.session["name"]="mohan"
    return render(request, 'set.html')

def getsession(request):
    if 'name' in request.session:
        name=request.session['name']
        request.session.modified=True
        return
    render(request, 'get.html', {'name':name})
    else:
        return HttpResponse("Session has
expired...")

```

```
def delsession(request):
    request.session.flush()
    request.session.clear_expired()
    return render(request, 'del.html')
```

set.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Session is set</h1>
</body>
</html>
```

get.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Session is Get</h1>
{{ name }}<br>
{{ request.session.get_session_cookie_age }}<br>
{{ request.session.get_expiry_age }}<br>
{{ request.session.get_expiry_date }}<br>
</body>
</html>
```

del.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Session is deleted</h1>
</body>
</html>
```

settings.py:

```
SESSION_COOKIE_AGE=30
```

urls.py: (project level)

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('set',views.setsession),
    path('get',views.getsession),
    path('del',views.delsession),
]
```

File based session:**Views.py:**

```
from django.shortcuts import render,HttpResponse

# Create your views here.
def setsession(request):
```

```

    request.session["name"]="mohan"
    return render(request, 'set.html')

def getsession(request):
    name=request.session['name']
    return
render(request, 'get.html', {'name':name})

def delsession(request):
    request.session.flush()
    request.session.clear_expired()
    return render(request, 'del.html')

```

settings.py:

```

STATIC_URL = '/static/'
SESSION_COOKIE_AGE=30
SESSION_ENGINE='django.contrib.sessions.backends.file'
SESSION_FILE_PATH=os.path.join(BASE_DIR, 'session')

```

Note: create a new folder with the name session under your project.

Cache:

- Cache is a server side state management technique.
- Cache is a temporary memory for the client at server side.
- Cache will improve the website performance.
- When we send request to server, server will process client request and deliver response to client, after deliver response , server will not maintain the data related to client.
- Every time when we send request to server , server will process as new request so that we will get delay response from the server, to avoid this we use cache mechanism.

- Once the webpage is having cache, for the first time server will process client request and store data related to client in cache memory.
- When we send request again, request will not process rather server will redirect your request to cache memory.

Types of cache:

- per-site cache
- per-view cache
- Template fragment cache

Options for cache:

- Database Caching
- File System Caching
- Local Memory Caching

Ex with per-site cache:

Database Caching:

- In this cache data will store in database.

settings.py :

```
STATIC_URL = '/static/'
CACHE_MIDDLEWARE_SECONDS=40
CACHES={
    'default': {

'BACKEND': 'django.core.cache.backends.db.DatabaseCa
che',
'LOCATION': 'myapp_cache',

    }
}
```

```

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddlewa
re',

    'django.middleware.cache.UpdateCacheMiddleware',
    'django.middleware.common.CommonMiddleware',

    'django.middleware.cache.FetchFromCacheMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddl
eware',

    'django.contrib.messages.middleware.MessageMiddlewa
re',

    'django.middleware.clickjacking.XFrameOptionsMiddle
ware',
]

```

home.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Durgasoftware solutions</h1>
<h1>Hyderabad</h1>
<h1>Maitrivanam</h1>
<h1>TS</h1>

```

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

Views.py:

```
from django.shortcuts import render

# Create your views here.

def v1(request):
    return render(request, 'home.html')
```

urls.py (application level) :

```
from django.urls import path
from MyApp import views

urlpatterns = [
    path('v1', views.v1),
]
```

urls.py (project level) :

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

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

- **Note:** before using this database cache we must and should create a cache table
- **python manage.py createcachetable**
- **python manage.py migrate**

File System Caching:

Settings.py:

```

STATIC_URL = '/static/'
CACHE_MIDDLEWARE_SECONDS=40
CACHES={
    'default': {

        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': 'G:\MydjangoProject\cache',

    }
}

```

Note: create a new folder with the name cache under project

Local Memory Caching:**settings.py:**

```

STATIC_URL = '/static/'
CACHE_MIDDLEWARE_SECONDS=40
CACHES={
    'default': {

        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': '',

    }
}

```

Note: Default time period of cache is 300 seconds or 5 minutes.

Ex with per-view cache:

- Using this we can cache particular view.

settings.py:

```

STATIC_URL = '/static/'
CACHES={
    'default':{

        'BACKEND': 'django.core.cache.backends.db.DatabaseCa
che',
        'LOCATION': 'view_cache',

    }
}

```

views.py:

```

from django.shortcuts import render
from django.views.decorators.cache import
cache_page

# Create your views here.
@cache_page(30)
def v1(request):
    return render(request, 'home.html')

def v2(request):
    return render(request, 'home1.html')

```

urls.py: (application level)

```

from django.urls import path
from MyApp import views

urlpatterns = [
    path('v1', views.v1),
    path('v2', views.v2),
]

```

urls.py: (project level)

```

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('myapp/',include('MyApp.urls')),
]

```

home.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Durgasoftware solutions</h1>
<h1>Hyderabad</h1>
<h1>maitrivanam</h1>
<h1>TS</h1>
</body>
</html>

```

home1.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Contact Details</h1>

```

```
<h1>Conatct:040-36547354</h1>
</body>
</html>
```

Ex with specifying per view cache in the URL configuration:

views.py:

```
from django.shortcuts import render

# Create your views here.
def v1(request):
    return render(request, 'home.html')

def v2(request):
    return render(request, 'home1.html')
```

urls.py: (application level)

```
from django.urls import path
from MyApp import views
from django.views.decorators.cache import
cache_page

urlpatterns = [
    path('v1', cache_page(30)(views.v1)),
    path('v2', views.v2),
]
```

Ex with Template fragment cache:

- Using this we can cache particular html elements.
- To implement this, we should include {% load cache %} in html document.

views.py:

```

from django.shortcuts import render

# Create your views here.
def v1(request):
    return render(request, 'home.html')

```

home.html:

```

<!DOCTYPE html>
{% load cache %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Durgasoftware solutions</h1>
{% cache 40 addr %}
<h1>Hyderabad</h1>
<h1>maitrivanam</h1>
<h1>TS</h1>
{% endcache addr %}
</body>
</html>

```

urls.py: (application level) :

```

from django.urls import path
from MyApp import views

urlpatterns = [
    path('v1', views.v1),
]

```

Django Security:

Authentication and Authorization:

- **Authentication** is the process of checking whether user credentials (username and password) are correct or not.
- **Authorization** is the process of checking whether authenticated user is having permission to access website content or not.
- Authentication and authorization is under comes into django authentication system.
- Django provides the following built-in applications for authentication
 1. django.contrib.auth
 2. django.contrib.contenttypes
- auth application internally uses contenttypes application to track models installed in our database.

Physical location of django authentication files:

C:\Users\lenovo\AppData\Local\Programs\Python\Python39\Lib\site-packages\django\contrib\auth

Note: if you don't find AppData, click on view menu and activate hidden items checkbox.

Creating signup form or Registration form:

Ex1:

views.py:

```
from django.shortcuts import render
from django.contrib.auth.forms import
UserCreationForm

# Create your views here.
def sign_up(request):
    fm=UserCreationForm()
    return
render(request, 'signup.html', { 'form': fm})
```

signup.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    {{ form.as_p }}
</body>
</html>
```

urls.py:

```
from django.contrib import admin
from django.urls import path
from secureapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('signup/', views.sign_up),
]
```

Note: by default UserCreationForm will provide username, password, confirm password fields, if you want more fields then we have to create our own form class.

Ex2: save the signup form data in to database:**forms.py:**

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

```

class signupForm(UserCreationForm):
    class Meta:
        model=User

fields=['username','first_name','last_name','email']

        labels={'email':'Email'}

```

views.py:

```

from django.shortcuts import render
#from django.contrib.auth.forms import
UserCreationForm
from .forms import signupForm
from django.contrib import messages

# Create your views here.
def sign_up(request):
    if request.method=='POST':
        fm=signupForm(request.POST)
        if fm.is_valid():
            messages.success(request,'user creation
successfully')
            fm.save()
        else:
            fm=signupForm()
        return
render(request,'signup.html', {'form':fm})

```

signup.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>

```

```

<body>
<form action="" method="post" novalidate>
    {% csrf_token %}
    {% for fm in form %}
        {{ fm.label_tag }} {{ fm }} {{
fm.errors|striptags }}<br><br>
    {% endfor %}
<input type="submit" value="submit">
</form>
{% if messages %}
    {% for message in messages %}
        {{ message }}
    {% endfor %}
{% endif %}
</body>
</html>

```

Creating login form and profile form:

views.py:

```

from django.shortcuts import render
#from django.contrib.auth.forms import
UserCreationForm
from .forms import signupForm
from django.contrib import messages
from django.contrib.auth.forms import
AuthenticationForm
from django.contrib.auth import
authenticate, login, logout
from django.http import HttpResponseRedirect

# Create your views here.
def sign_up(request):
    if request.method=='POST':
        fm=signupForm(request.POST)
        if fm.is_valid():
            messages.success(request, 'user creation

```

```

successfully')
        fm.save()
    else:
        fm=signupForm()
    return
render(request, 'signup.html', {'form':fm})

def user_login(request):
    if request.method=='POST':

fm=AuthenticationForm(request=request,data=request.
POST)
    if fm.is_valid():
        uname=fm.cleaned_data['username']
        pwd=fm.cleaned_data['password']

user=authenticate(username=uname,password=pwd)
        if user is not None:
            login(request,user)
            return
HttpResponseRedirect('/profile/')
    else:
        fm=AuthenticationForm()
        return render(request, 'login.html', {'form':fm})

def profile(request):
    if request.user.is_authenticated:
        return
render(request, 'profile.html', {'name':request.user}
)
    else:
        return HttpResponseRedirect('/login/')

def user_logout(request):
    logout(request)
    return HttpResponseRedirect('/login/')

```

forms.py:

```

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

class signupForm(UserCreationForm):
    class Meta:
        model=User

fields=[ 'username', 'first_name', 'last_name', 'email'
]

labels={ 'email': 'Email' }

```

signup.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post" novalidate>
    {% csrf_token %}
    {% for fm in form %}
        {{ fm.label_tag }}{{ fm }}{{
fm.errors|striptags }}<br><br>
    {% endfor %}
<input type="submit" value="submit">
</form>
{% if messages %}
    {% for message in messages %}
        {{ message }}
    {% endfor %}
{% endif %}
<a href="{% url 'login' %}">Login</a>

```

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

login.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<style>
  .error{color: red}
</style>
<body>

<form action="" method="post" novalidate>
  {% csrf_token %}
  {% if form.non_field_errors %}
    {% for error in form.non_field_errors %}
      <p class="error">{{ error }}</p>
    {% endfor %}
  {% endif %}

  {% for fm in form %}
    {{ fm.label_tag }}{{ fm }}{{
fm.errors|striptags }} <br><br>
  {% endfor %}
  <input type="submit" value="Login">
<a href="{% url 'signup' %}">signup</a>
</form>
</body>
</html>
```

profile.html:

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Profile page</h1>
<h1>welcome {{ name }}</h1>
<a href="{% url 'logout' %}">Logout</a>
</body>
</html>

```

urls.py: (project level)

```

from django.contrib import admin
from django.urls import path
from secureapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('signup/', views.sign_up, name='signup'),
    path('login/', views.user_login, name='login'),
    path('profile/', views.profile, name='profile'),
    path('logout/', views.user_logout, name='logout')
]

```

Creating change password form:

views.py:

```

from django.shortcuts import render
#from django.contrib.auth.forms import
UserCreationForm
from .forms import signupForm
from django.contrib import messages

```



```

from django.contrib.auth.forms import
AuthenticationForm, PasswordChangeForm, SetPasswordFo
rm
from django.contrib.auth import
authenticate, login, logout, update_session_auth_hash
from django.http import HttpResponseRedirect

# Create your views here.
def sign_up(request):
    if request.method=='POST':
        fm=signupForm(request.POST)
        if fm.is_valid():
            messages.success(request, 'user creation
successfully')
            fm.save()
        else:
            fm=signupForm()
    return
render(request, 'signup.html', {'form':fm})

def user_login(request):
    if not request.user.is_authenticated:
        if request.method == 'POST':
            fm =
AuthenticationForm(request=request,
data=request.POST)
            if fm.is_valid():
                uname = fm.cleaned_data['username']
                pwd = fm.cleaned_data['password']
                user = authenticate(username=uname,
password=pwd)
                if user is not None:
                    login(request, user)
                    return
            HttpResponseRedirect('/profile/')
        else:
            fm = AuthenticationForm()
            return render(request, 'login.html',

```

```

{'form': fm})
    else:
        return HttpResponseRedirect('/profile/')

def profile(request):
    if request.user.is_authenticated:
        messages.success(request, 'Login successfull')
        return
    render(request, 'profile.html', {'name': request.user})
    else:
        return HttpResponseRedirect('/login/')

def user_logout(request):
    logout(request)
    return HttpResponseRedirect('/login/')

#change password with old password
def user_changepassword(request):
    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, 'changepassword.html', {'form': fm})

#change password without old password
def user_changepassword1(request):
    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, 'changepassword1.html', {'form': fm})

```

forms.py:

```

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

class signupForm(UserCreationForm):
    class Meta:
        model=User

fields=['username','first_name','last_name','email']
labels={'email':'Email'}

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from djangoapp import views

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

```

```

    path('signup/', views.sign_up, name='signup'),
    path('login/', views.user_login, name='login'),
    path('profile/', views.profile,
name='profile'),
    path('logout/', views.user_logout,
name='logout'),

path('changepassword/', views.user_changepassword, na
me='changepassword'),
    path('changepassword1/',
views.user_changepassword1,
name='changepassword1'),

]

```

signup.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post" novalidate>
    {% csrf_token %}
    {% for fm in form %}
        {{ fm.label_tag }} {{ fm }} {{
fm.errors|striptags }}<br><br>
    {% endfor %}
<input type="submit" value="submit">
</form>
{% if messages %}
    {% for message in messages %}
        {{ message }}
    {% endfor %}
{% endif %}

```

```
<a href="{% url 'login' %}">Login</a>
</body>
</html>
```

login.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<style>
    .error{color: red}
</style>
<body>

<form action="" method="post" novalidate>
    {% csrf_token %}
    {% if form.non_field_errors %}
        {% for error in form.non_field_errors %}
            <p class="error">{{ error }}</p>
        {% endfor %}
    {% endif %}

    {% for fm in form %}
        {{ fm.label_tag }}{{ fm }}{{
fm.errors|striptags }} <br><br>
    {% endfor %}
    <input type="submit" value="Login">
<a href="{% url 'signup' %}">signup</a>

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

profile.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Profile page</h1>
<h1>welcome {{ name }}</h1>
{% if messages %}
    {% for message in messages %}
        {{ message }}
    {% endfor %}
{% endif %}
<a href="{% url 'logout' %}">Logout</a>
<a href="{% url 'changepassword' %}">changepassword</a>
<a href="{% url 'changepassword1' %}">changepassword1</a>
</body>
</html>

```

changepassword.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post" novalidate>
    {% csrf_token %}
    {% if form.non_field_errors %}

```

```

    {% for error in form.non_field_errors %}
        <p class="error">{{ error }}</p>
    {% endfor %}
{% endif %}

{% for fm in form %}
    {{ fm.label_tag }}{{ fm }}{{
fm.errors|striptags }} <br><br>
{% endfor %}
<input type="submit" value="save">
</form>
<a href="{% url 'profile' %}">profile</a>
<a href="{% url 'logout' %}">logout</a>
</body>
</html>

```

changepassword1.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post" novalidate>
    {% csrf_token %}
    {% if form.non_field_errors %}
    {% for error in form.non_field_errors %}
        <p class="error">{{ error }}</p>
    {% endfor %}
    {% endif %}

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

```

```

    <input type="submit" value="save">
</form>
<a href="{% url 'profile' %}">profile</a>
<a href="{% url 'logout' %}">logout</a>
</body>
</html>

```

Display profile data, modify and save into database:

views.py:

```

from django.shortcuts import render
#from django.contrib.auth.forms import
UserCreationForm
from .forms import signupForm, EditUserProfileForm
from django.contrib import messages
from django.contrib.auth.forms import
AuthenticationForm, PasswordChangeForm
from django.contrib.auth import
authenticate, login, logout, update_session_auth_hash
from django.http import HttpResponseRedirect

# Create your views here.
def sign_up(request):
    if request.method=='POST':
        fm=signupForm(request.POST)
        if fm.is_valid():
            messages.success(request, 'user creation
successfully')
            fm.save()
        else:
            fm=signupForm()
    return
render(request, 'signup.html', {'form':fm})

def user_login(request):
    if not request.user.is_authenticated:
        if request.method == 'POST':

```



```

        fm =
AuthenticationForm(request=request,
data=request.POST)
        if fm.is_valid():
            uname = fm.cleaned_data['username']
            pwd = fm.cleaned_data['password']
            user = authenticate(username=uname,
password=pwd)
            if user is not None:
                login(request, user)
                return
HttpResponseRedirect('/profile/')
        else:
            fm = AuthenticationForm()
            return render(request, 'login.html',
{'form': fm})
        else:
            return HttpResponseRedirect('/profile/')

def profile(request):
    if request.user.is_authenticated:
        if request.method=='POST':

fm=EditUserProfileForm(request.POST,instance=request
t.user)
            if fm.is_valid():
                fm.save()
                messages.success(request, 'profile
is updated')
            else:

fm=EditUserProfileForm(instance=request.user)
            return
render(request, 'profile.html', {'name': request.user,
'form': fm})
        else:
            return HttpResponseRedirect('/login/')

```

```

def user_logout(request):
    logout(request)
    return HttpResponseRedirect('/login/')

#change password with old password
def user_changepassword(request):
    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,'changepassword.html',{'form':fm})

```

forms.py:

```

from django.contrib.auth.models import User
from django.contrib.auth.forms import
UserCreationForm,UserChangeForm

class signupForm(UserCreationForm):
    class Meta:
        model=User

fields=['username','first_name','last_name','email'
]
        labels={'email':'Email'}

```

```
class EditUserProfileForm(UserChangeForm):
    password = None
    class Meta:
        model=User

fields=['username','first_name','last_name','email',
        'date_joined','last_login']
```

urls.py:

```
from django.contrib import admin
from django.urls import path
from djangoapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('signup/', views.sign_up, name='signup'),
    path('login/', views.user_login, name='login'),
    path('profile/', views.profile,
name='profile'),
    path('logout/', views.user_logout,
name='logout'),

    path('changepassword/', views.user_changepassword, na
me='changepassword'),

]
```

signup.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
```

```

<body>
<form action="" method="post" novalidate>
    {% csrf_token %}
    {% for fm in form %}
        {{ fm.label_tag }} {{ fm }} {{
fm.errors|striptags }}<br><br>
    {% endfor %}
<input type="submit" value="submit">
</form>
{% if messages %}
    {% for message in messages %}
        {{ message }}
    {% endfor %}
{% endif %}
<a href="{% url 'login' %}">Login</a>
</body>
</html>

```

login.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<style>
    .error{color: red}
</style>
<body>

<form action="" method="post" novalidate>
    {% csrf_token %}
    {% if form.non_field_errors %}
        {% for error in form.non_field_errors %}
            <p class="error">{{ error }}</p>
        {% endfor %}

```

```

    {% endif %}

    {% for fm in form %}
        {{ fm.label_tag }} {{ fm }} {{
fm.errors|striptags }} <br><br>
    {% endfor %}
    <input type="submit" value="Login">
<a href="{% url 'signup' %}">signup</a>

</form>
</body>
</html>

```

profile.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Profile page</h1>
<h1>welcome {{ name }}</h1>
{% if messages %}
    {% for message in messages %}
        {{ message }}
    {% endfor %}
{% endif %}

<form action="" method="post">
    {% csrf_token %}
    {% if form.non_field_errors %}
        {% for error in form.non_field_errors %}
            <p class="error">{{ error }}</p>
        {% endfor %}
    {% endif %}

```

```

        {% for fm in form %}
            {{ fm.label_tag }} {{ fm }} {{
fm.errors|striptags }} <br><br>
        {% endfor %}
        <input type="submit" value="save">
</form>

<a href="{% url 'logout' %}">Logout</a>
<a href="{% url 'changepassword'
%}">changepassword</a>
</body>
</html>

```

changepassword.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post" novalidate>
    {% csrf_token %}
    {% if form.non_field_errors %}
    {% for error in form.non_field_errors %}
        <p class="error">{{ error }}</p>
    {% endfor %}
    {% endif %}

    {% for fm in form %}
        {{ fm.label_tag }} {{ fm }} {{
fm.errors|striptags }} <br><br>
    {% endfor %}
    <input type="submit" value="save">
</form>
<a href="{% url 'profile' %}">profile</a>

```

```
<a href="{% url 'logout' %}">logout</a>  
</body>  
</html>
```

Types of views:

There are 2 types of views

- function based view
- class based view

Class based view:

- Class based views are introduced in django 1.3 version to implement generic views.
- When compared with function based views, class based views are very easy to use.
- Class based views are most frequently used views in real time.
- Internally class based views will be convert into function based views.
- Class based views simply act as wrappers to the function based views.
- Function based views are most powerful when compared with class based views.
- For simple operations like listing of all records or display details of a particular record then we should go for class based views.
- For complex operations like handling multiple forms simultaneously then we should go for function based views.
- While defining class based views we have to extend View class.
- To provide response to GET request, Django will always calls get() method so that we have to override this method to respond to the GET request.
- While defining url pattern we have to use as_view() method.

Ex:

Views.py:

```

from django.shortcuts import render
from django.views.generic import View
from django.http import HttpResponse

# Create your views here.
class ClassBasedView(View):
    def get(self, request):
        return HttpResponse('<h1>This is my first
classbasedview')

```

urls.py :

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.ClassBasedView.as_view()),
]

```

- Class based views will provide alternative way to create views as python objects instead of functions.
- Class Based views are two types:
 - Base class-based view or Base view
 - Generic class-based view or Generic view

Base class-based view or Base view:

- View
- Template View
- Redirect View

Ex1:

Views.py:


```

from django.shortcuts import render
from django.views import View
from django.http import HttpResponse

# Create your views here.
#function based view
def fun_view(request):
    return HttpResponse("<h1>Function based
view</h1>")

#class based view
class cls_view(View):
    name=""
    def get(self,request):
        #return HttpResponse("<h1>Class based
view</h1>")
        return HttpResponse(self.name)

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('fun/', views.fun_view),

    path('cls/', views.cls_view.as_view(name="mohan")),
]

```

Ex2:**Views.py:**

```

from django.shortcuts import render
from django.views import View

```

```

from django.http import HttpResponseRedirect

# Create your views here.
#function based view
def fun_view1(request):
    return render(request, 'home.html')

#class based view
class cls_view1(View):
    def get(self, request):
        return render(request, 'home.html')

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('fun/', views.fun_view1),
    path('cls/', views.cls_view1.as_view()),
]

```

home.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Welcome to Home</h1>
</body>
</html>

```

Ex3:**Views.py:**

```

from django.shortcuts import render
from django.views import View
from django.http import HttpResponseRedirect

# Create your views here.
#function based view
def fun_view2(request):
    context={'msg':"welcome to durgasoft"}
    return render(request, 'home.html', context)

#class based view
class cls_view2(View):
    def get(self, request):
        context = {'msg': "welcome to durgasoft"}
        return render(request, 'home.html', context)

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('fun/', views.fun_view2),
    path('cls/', views.cls_view2.as_view()),
]

```

home.html:

```

<!DOCTYPE html>
<html lang="en">

```

```

<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Welcome to Home</h1>
{{ msg }}
</body>
</html>

```

Ex4:**Forms.py:**

```

from django import forms

class MyForm(forms.Form):
    name=forms.CharField(max_length=20)
    address=forms.CharField(max_length=30)

```

views.py:

```

from django.shortcuts import render
from django.views import View
from django.http import HttpResponse
from myapp.forms import MyForm

# Create your views here.
#function based view
def fun_view3(request):
    if request.method=='POST':
        form=MyForm(request.POST)
        if form.is_valid():
            print(form.cleaned_data['name'])
            print(form.cleaned_data['address'])
            return HttpResponse("Form is
submitted")

```

```

        else:
            form=MyForm()
        return
    render(request, 'home.html', {'form':form})

#class based view
class cls_view3(View):
    def get(self,request):
        form=MyForm()
        return
    render(request, 'home.html', {'form':form})

    def post(self,request):
        form=MyForm(request.POST)
        if form.is_valid():
            print(form.cleaned_data['name'])
            print(form.cleaned_data['address'])
        return HttpResponseRedirect("Form is submitted")

```

home.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="submit">
</form>
</body>
</html>

```

Urls.py:

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('fun/', views.fun_view3),
    path('cls/', views.cls_view3.as_view()),
]

```

Ex5: Rendering multiple Templates using same view.**Views.py:**

```

from django.shortcuts import render
from django.views import View
from django.http import HttpResponse

# Create your views here.
#function based view
def fun_view4(request, template_name):
    template_name=template_name
    context={'msg':"Durgasoft Django Sessions"}
    return render(request, template_name, context)

#class based view
class cls_view4(View):
    template_name=""
    def get(self, request):
        context={'msg':"Durgasoft python sessions"}
        return
render(request, self.template_name, context)

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from myapp import views

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

    path('fun/', views.fun_view4, {'template_name': 'home.html'}),
    path('fun1/', views.fun_view4,
{'template_name': 'home1.html'}),

    path('cls/', views.cls_view4.as_view(template_name='home.html')),
    path('cls1/',
views.cls_view4.as_view(template_name='home1.html')
),
]

```

home.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Information:{{ msg }}</h1>
</body>
</html>

```

home1.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Message:{{ msg }}</h1>
</body>
</html>
```

Template view:

```
from django.shortcuts import render
from django.views.generic.base import TemplateView
# Create your views here.

class MyTemplateView(TemplateView):
    template_name = 'home.html'

    def get_context_data(self, **kwargs):
        context=super().get_context_data(**kwargs)
        context['name']="mohan"
        context['mail']="mohan@gmail.com"
        return context
```

urls.py:

```
from django.contrib import admin
from django.urls import path
from djangoapp import views

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

    path('home/', views.MyTemplateView.as_view(extra_con
```



```
text={ 'age':37})),
]
```

home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Home Page</h1>
<h1>Name is:{{ name }}</h1>
<h1>Mail is:{{ mail }}</h1>
<h1>Age is:{{ age }}</h1>
</body>
</html>
```

Redirect view:

Views.py:

```
from django.shortcuts import render
from django.views.generic.base import
TemplateView,RedirectView
# Create your views here.

class FlipKartView(RedirectView):
    url='https://flipkart.com'
```

urls.py:

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

```

from djangoapp import views

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

    path('', views.TemplateView.as_view(template_name='home1.html')),

    path('home1/', views.RedirectView.as_view(url='/')),

    #path('flipkart/', views.RedirectView.as_view(url='https://flipkart.com')),
    path('flipkart/', views.FlipKartView.as_view()),

]

```

home1.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Welcome to home page</h1>
</body>
</html>

```

Generic class-based view or Generic view:

1. **Display view** : List view, Details View
2. **Editing View**: Form View, Create View, Update View, Delete View
3. **Date views**

List View:

- We can use List view class to list out all records from database table (model).

Ex:

models.py:

```
from django.db import models

# Create your models here.
class Student(models.Model):
    name=models.CharField(max_length=20)
    mail=models.EmailField(max_length=30)
    address=models.CharField(max_length=20)
```

admin.py:

```
from django.contrib import admin
from myapp.models import Student

# Register your models here.

@admin.register(Student)
class StudentAdmin(admin.ModelAdmin):
    list_display = ['name', 'mail', 'address']
```

- **Run these commands in terminal**
- Python manage.py makemigrations
- Python manage.py migrate
- Python manage.py createsuperuser
- **Login to admin panel and insert few records**

views.py:

```
from django.shortcuts import render
from django.views.generic.list import ListView
from .models import Student
```

```
# Create your views here.
class StudentListView(ListView):
    model = Student
```

urls.py:

```
from django.contrib import admin
from django.urls import path
from myapp import views

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

    path('', views.StudentListView.as_view(template_name
    = "student_list.html")),
]
```

- **create student_list.html in template folder**

student_list.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
{% for student in student_list %}
    {{ student.name }}
    {{ student.mail }}
    {{ student.address }}<br>
{% endfor %}
</body>
</html>
```

More options with List view:**Views.py:**

```

from django.shortcuts import render
from django.views.generic.list import ListView
from .models import Student

# Create your views here.
class StudentListView(ListView):
    model = Student
    template_name = 'student_list.html'
    #template_name = "student_details.html"
    #template_name_suffix = '_details'
    #ordering = ['name']
    #template_name="student.html"
    context_object_name = 'students'

    def get_queryset(self):
        return
Student.objects.filter(address='hyd')

```

Details view:

- It is used to display particular record to the user.

Ex:**models.py:**

```

from django.db import models

# Create your models here.
class Student1(models.Model):
    name=models.CharField(max_length=20)
    mail=models.EmailField(max_length=30)
    address=models.CharField(max_length=20)

```

admin.py:

```

from django.contrib import admin
from detailsapp.models import Student1
# Register your models here.
@admin.register(Student1)
class StudentAdmin1(admin.ModelAdmin):
    list_display = ['name', 'mail', 'address']

```

views.py:

```

from django.shortcuts import render
from detailsapp.models import Student1
from django.views.generic.detail import DetailView

```

```

# Create your views here.
class StudentDetailsView(DetailView):
    model = Student1
    template_name = 'student_detail.html'
    pk_url_kwarg = 'stid'

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from detailsapp import views

```

```

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

    #path('stu/<int:pk>', views.StudentDetailsView.as_view()),

    path('stu/<int:stid>', views.StudentDetailsView.as_view()),
]

```

student_detail.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h3>Student Details</h3>
{{ student1.name }}
{{ student1.mail }}
{{ student1.address }}
</body>
</html>

```

Ex with List view with details view:

models.py:

```

from django.db import models

# Create your models here.
class Student1(models.Model):
    name=models.CharField(max_length=20)
    mail=models.EmailField(max_length=30)
    address=models.CharField(max_length=20)

```

admin.py:

```

from django.contrib import admin
from detailsapp.models import Student1
# Register your models here.
@admin.register(Student1)
class StudentAdmin1(admin.ModelAdmin):
    list_display = ['name', 'mail', 'address']

```

views.py

```

from django.shortcuts import render
from detailsapp.models import Student1
from django.views.generic.detail import DetailView
from django.views.generic.list import ListView

# Create your views here.
class StudentDetailsView(DetailView):
    model = Student1
    template_name = 'student_detail.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 = Student1
    template_name = 'student_list.html'

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from detailsapp import views

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

    path('student/', views.StudentListView.as_view(), name='studentlist'),

    path('student/<int:pk>', views.StudentDetailsView.as_view(), name='studentdetail'),

]

```


student_list.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h3>Student list</h3>

{% for stu1 in student1_list %}
    <li><a href="{% url 'studentdetail' stu1.id
%}">{{ stu1.name }}</a></li>
{% endfor %}

</body>
</html>

```

Student_detail.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h3>Student Details</h3>
{{ student1.name }}
{{ student1.mail }}
{{ student1.address }}
<hr>
{% for stu1 in all_student %}
    <li><a href="{% url 'studentdetail' stu1.id

```

```

    %}">{{ stu1.name }}</a></li>
    {% endfor %}
</body>
</html>

```

Working with editing views:

- Form View
- Create View
- Update View
- Delete View

Form view:

- Form view refers to a view to display and verify a django form.
- Form view present in `django.views.generic.edit.FormView`.
- **Attributes:**
- **initial:** initial values for the form.
- **form_class:** form class to instantiate.
- **success_url :** URL in which you redirect when the form successfully processed.
- **form_valid ():** this method is called when the valid form data has been posted.

Ex:

forms.py:

```

from django import forms

class StudentForm(forms.Form):
    name=forms.CharField()
    mail=forms.EmailField()
    msg=forms.CharField(widget=forms.Textarea)

```

views.py:

```

from django.shortcuts import render
from formsapp.forms import StudentForm
from django.views.generic.edit import FormView
from django.views.generic.base import TemplateView

```

```

# Create your views here.

```

```

class StudentFormView(FormView):
    template_name = 'student.html'
    form_class = StudentForm
    success_url = '/success/'

    def form_valid(self, form):
        print(form.cleaned_data['name'])
        print(form.cleaned_data['mail'])
        print(form.cleaned_data['msg'])
        return super().form_valid(form)

class SuccessTemplateView(TemplateView):
    template_name = 'success.html'

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from formsapp import views

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

    path('student/', views.StudentFormView.as_view()),

    path('success/', views.SuccessTemplateView.as_view(
        , name='success'),

```

]

student.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post" novalidate>
    {% csrf_token %}
    {{form.as_p}}
    <input type="submit" value="submit">
</form>
</body>
</html>

```

success.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Form submitted successfully</h1>
</body>
</html>

```

Create view:

- Create view refers to a view to create an instance of a table in the database.

Ex:

models.py:

```
from django.db import models

# Create your models here.
class JobModel(models.Model):
    jobname=models.CharField(max_length=30)
    jobtitle=models.CharField(max_length=40)
    jobdescription=models.TextField()

    def __str__(self):
        return self.jobtitle
```

views.py:

```
from django.shortcuts import render
from createapp.models import JobModel
from django.views.generic.edit import CreateView
from django.views.generic.base import TemplateView

# Create your views here.
class JobCreateView(CreateView):
    model = JobModel
    template_name = 'jobmodel_form.html'
    fields = ['jobname',
, 'jobtitle', 'jobdescription']
    success_url = '/success/'

class JobTemplateView(TemplateView):
    template_name = 'success.html'
```

urls.py:

```
from django.contrib import admin
from django.urls import path
from createapp import views
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.JobCreateView.as_view()),

    path('success/', views.JobTemplateView.as_view(), name='success'),
]
```

jobmodel_form.html :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="submit">
</form>
</body>
</html>
```

success.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Form Data stored in database</h1>
</body>
</html>
```

Update view:

- Update view refers to a view to update a particular instance of a table from the database.

Ex:**models.py:**

```
from django.db import models

# Create your models here.
class Trainer(models.Model):
    tname=models.CharField(max_length=20)
    tpassword=models.CharField(max_length=10)
    taddress=models.CharField(max_length=20)
    tage=models.CharField(max_length=20)
```

admin.py:

```
from django.contrib import admin
from updateapp.models import Trainer
# Register your models here.
@admin.register(Trainer)
class TrainerAdmin(admin.ModelAdmin):
    list_display =
    ['tname', 'tpassword', 'taddress', 'tage']
```

Execute the following commands in terminal

- Python manage.py makemigrations
- Python manage.py migrate
- Python manage.py createsuperuser

Views.py:

```

from django.shortcuts import render
from django.views.generic.edit import UpdateView
from django.views.generic.base import TemplateView
from updateapp.models import Trainer

# Create your views here.

class TrainerUpdateView(UpdateView):
    model = Trainer
    template_name = 'trainer_form.html'
    fields =
['tname','tpassword','taddress','tage']
    success_url = '/success/'

class SuccessTemplateView(TemplateView):
    template_name = 'success.html'

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from updateapp import views

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

    path('trainer/<int:pk>', views.TrainerUpdateView.as_
view()),

    path('success/', views.SuccessTemplateView.as_view()
, name='success'),

]

```


trainer_form.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Update">
</form>
</body>
</html>
```

success.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Data Updated successfully</h1>
</body>
</html>
```

Delete view:

- Delete view refers to a view to delete a particular instance of a table from the database.

Ex:

models.py:

```
from django.db import models

# Create your models here.
class Actor(models.Model):
    name=models.CharField(max_length=20)
    password=models.CharField(max_length=10)
    address=models.CharField(max_length=20)
    age=models.CharField(max_length=20)
```

admin.py:

```
from django.contrib import admin
from deleteapp.models import Actor
# Register your models here.
@admin.register(Actor)
class ActorAdmin(admin.ModelAdmin):
    list_display =
    ['name', 'password', 'address', 'age']
```

Execute the following commands in terminal

- Python manage.py makemigrations
- Python manage.py migrate
- Python manage.py createsuperuser

views.py:

```
from django.shortcuts import render
from django.views.generic.edit import DeleteView
from django.views.generic.base import TemplateView
from deleteapp.models import Actor

# Create your views here.

class ActorDeleteView(DeleteView):
```

```

    model = Actor
    template_name = 'actor_confirm_delete.html'
    fields = ['name', 'password', 'address', 'age']
    success_url = '/success/'

class SuccessTemplateView(TemplateView):
    template_name = 'success.html'

class NoDeleteTemplateView(TemplateView):
    template_name = 'nodelete.html'

urls.py:

from django.contrib import admin
from django.urls import path
from deleteapp import views

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

    path('actor/<int:pk>', views.ActorDeleteView.as_view
    ()),

    path('success/', views.SuccessTemplateView.as_view()
    , name='success'),

    path('nodelete/', views.NoDeleteTemplateView.as_view
    (), name='nodelete'),

]

```

actor_confirm_delete.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

```

```

    <title>Title</title>
</head>
<body>
<h2>Do you want to delete?</h2>
<form action="" method="post">
    {% csrf_token %}
    <input type="submit" value="Delete">
    <a href="{% url 'nodelete' %}">Cancel</a>
</form>
</body>
</html>

```

success.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Data Deleted successfully</h1>
</body>
</html>

```

nodelete.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h2>You have not confirm to delete</h2>

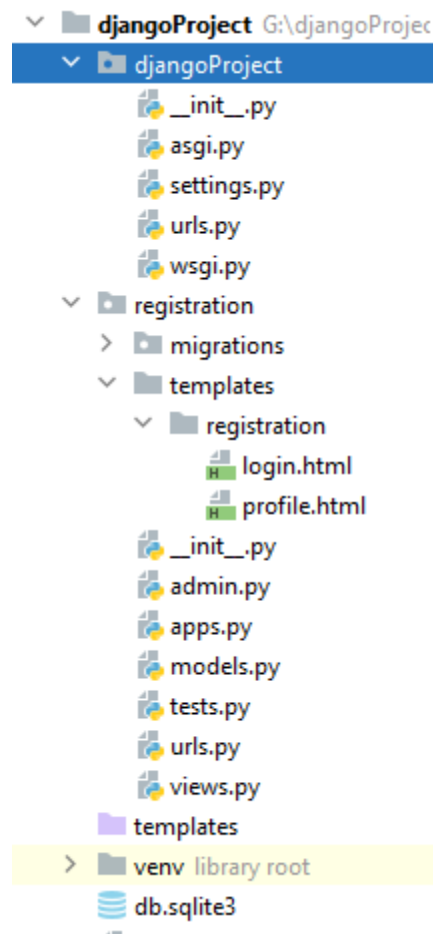
```

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

Authentication views:

- Django will provide several views that you can use for handling login, logout, and change password and reset password.
- Django will not provide default template for the authentication views.
- We should create our own templates for the views which we want to use.
- Physical location of authentication views and urls.
- C:\Users\lenovo\AppData\Local\Programs\Python\Python39\Lib\site-packages\django\contrib\auth

Must follow the bellow project hierarchy.



Ex:

- Create a new project with the name.djangoproject
- Create a new application with the name registration
- python manage.py startapp registration
- Register your app under installed apps of settings.py file
- Python manage.py makemigrations
- Python manage.py migrate
- Python manage.py createsuperuser
- Login into admin panel and create a user.

url.py (project level):

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

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

    path('accounts/',include('django.contrib.auth.urls'
)),
    path('accounts/',include('registration.urls')),
]
```

urls.py :(application level) :

```
from django.urls import path
from registration import views

urlpatterns = [
```

```
    path('profile/', views.profile, name='profile'),
]
```

views.py:

```
from django.shortcuts import render

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

login.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="" method="post" novalidate>
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Login">
    <a href="{% url 'password_reset' %}">Reset
Password</a>
</form>
</body>
</html>
```

Profile.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```

        <title>Title</title>
</head>
<body>
<h1>Profile page</h1>
<a href="{% url 'logout' %}">Logout</a>
<a href="{% url 'password_change' %}">Change
Password</a>
</body>
</html>

```

Settings.py:

```

STATIC_URL = '/static/'

EMAIL_BACKEND =
'django.core.mail.backends.console.EmailBackend'

```

Ex with login_required decorator and staff_member_required decorator:**Views.py :**

```

from django.shortcuts import render
from django.contrib.auth.decorators import
login_required
from django.contrib.admin.views.decorators import
staff_member_required

# Create your views here.
@login_required
def profile(request):
    return
render(request, 'registration/profile.html')

@staff_member_required()
def contact(request):

```



```

    return
render(request, 'registration/contact.html')

```

urls.py: (application level)

```

from django.urls import path
from registration import views

urlpatterns = [
    path('profile/', views.profile, name='profile'),
    path('contact/', views.contact, name='contact'),
]

```

contact.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>contact details</h1>
<h2>DUrga software solutions</h2>
<h3>Hyderabad</h3>
</body>
</html>

```

Pagination:

- Pagination is the process of display data in to different pages.
- Pagination allows us to split data into multiple pages.
- Django will provide classes to manage page data.
 - Paginator class
 - Page class

Ex with Pagination using function based views

- Create a new application with the name pageapp
 - Python manage.py startapp pageapp
 - Install pageapp into settings.py

models.py:

```
from django.db import models

# Create your models here.
class Book(models.Model):
    title=models.CharField(max_length=30)
    author=models.CharField(max_length=20)
    description=models.TextField(max_length=300)
```

admin.py:

```
from django.contrib import admin
from pageapp.models import Book

# Register your models here.
@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    list_display =
    ['id', 'title', 'author', 'description']
```

- Go to terminal and execute the following commands
 - Python manage.py makemigrations
 - Python manage.py migrate
 - Python manage.py createsuperuser
 - Login into admin panel and insert few records records

views.py:

```
from django.shortcuts import render
from pageapp.models import Book
from django.core.paginator import Paginator
```

```

# Create your views here.
def page_view(request):
    all_pages=Book.objects.all().order_by('id')
    paginator=Paginator(all_pages,3,orphans=1)
    page_number=request.GET.get('page')
    page_obj=paginator.get_page(page_number)
    return
render(request,'pages.html',{'page_obj':page_obj})

```

pages.html:

```

<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Book Details</h1>
{% for page in page_obj %}
    <h2>{{ page.title }}</h2>
    <h3>{{ page.author }}</h3>
    <p>{{ page.description }}</p>
{% endfor %}

<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>
```

urls.py: (application level)

```
from django.urls import path
from pageapp import views

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

urls.py: (project level)

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

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

Ex with Pagination using class based views

- Create a new application with the name pageapp
 - Python manage.py startapp pageapp
 - Install pageapp into settings.py

models.py:

```
from django.db import models

# Create your models here.
class Book(models.Model):
```

```

title=models.CharField(max_length=30)
author=models.CharField(max_length=20)
description=models.TextField(max_length=300)

```

admin.py:

```

from django.contrib import admin
from pageapp.models import Book

# Register your models here.
@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    list_display =
['id', 'title', 'author', 'description']

```

- **Go to terminal and execute the following commands**
 - Python manage.py makemigrations
 - Python manage.py migrate
 - Python manage.py createsuperuser
 - Login into admin panel and insert few records records

views.py:

```

from django.shortcuts import render
from pageapp.models import Book
from django.views.generic import
ListView,DetailView
from django.http import Http404

# Create your views here.
class PageListView(ListView):
    model = Book
    template_name = 'pages.html'

```

```

ordering = ['id']
paginate_by = 3
paginate_orphans = 1

def get_context_data(self, *args, **kwargs):
    try:
        return
    super(PageListView, self).get_context_data(*args, **kwargs)
    except Http404:
        self.kwargs['page']=1
        return super(PageListView,
self).get_context_data(*args, **kwargs)

class PageDetailView(DetailView):
    model = Book
    template_name = 'detail.html'

```

urls.py :(application level)

```

from django.urls import path
from pageapp import views

urlpatterns = [
    path('', views.PageListView.as_view()),

    path('detail/<int:pk>', views.PageDetailView.as_view(), name="page"),

]

```

detail.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

```

```

        <title>Title</title>
</head>
<body>

        <h2>{{ book.title }}</h2>
        <h3>{{ book.author }}</h3>
        <p>{{ book.description }}</p>

</body>
</html>

```

urls.py :(project level)

```

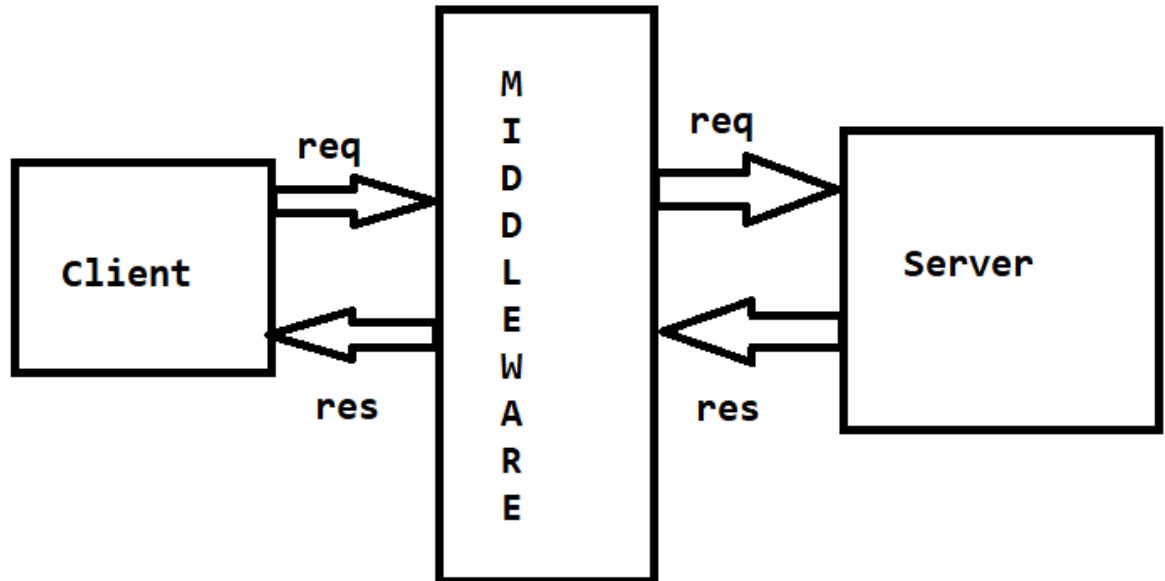
from django.contrib import admin
from django.urls import path,include

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

```

Middleware:

- Middleware is a framework of hooks into django's request/response processing.
- It is a light weight and low level plugin system for globally altering django's input or output.
- If we want to perform any activity at the time of preprocessing of the request or post processing of the request then we should go for middleware.



Types of middleware:

- Built-in middleware
- Custom middleware

Note: Django contains several inbuilt middleware's which are configured inside settings.py

```
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.XFrameOptionsMiddle
ware'
]
```

- All these middleware's will be executed before and after processing of every request.

Ex with custom middleware using Function based:

- Create a new application with the name middlewareapp
 - Python manage.py startapp middlewareapp
 - Install middlewareapp into settings.py
 - Create new file inside the application with the name middleware.py

middleware.py:

```
def demo_middleware(get_response):
    print("One time initialization")

    def fun_middleware(request):
        print("This is before view calling")
        response=get_response(request)
        print("This is after view calling")
        return response
    return fun_middleware
```

Activate middleware:

- Go to settings .py file and add your middleware
- MIDDLEWARE = [

```
'django.middleware.security.SecurityMiddleware'
,

'django.contrib.sessions.middleware.SessionMidd
leware',
```

```

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationM
iddleware',

'django.contrib.messages.middleware.MessageMidd
leware',

'django.middleware.clickjacking.XFrameOptionsMi
ddleware',
    'middlewareapp.middleware.demo_middleware'
]

```

views.py:

```

from django.shortcuts import render,HttpResponse

# Create your views here.

def v1(request):
    print("This is a view v1")
    return HttpResponse("This is my first
middleware program")

```

urls.py: (application level):

```

from django.urls import path
from middlewareapp import views

urlpatterns=[
    path('',views.v1),
]

```

urls.py: (project level):

```

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

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

```

Ex with custom middleware using Class based:

- Create a new application with the name middlewareapp
 - Python manage.py startapp middlewareapp
 - Install middlewareapp into settings.py
 - Create new file inside the application with the name middleware.py

middleware.py:

```

class ClassBasedMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        print("This line is added at pre-processing request")
        response = self.get_response(request)
        print("This line is added post-processing request")
        return response

```

views.py:

```

from django.http import HttpResponse

def welcome_view(request):
    print("This is added by view function")

```

```

        return HttpResponse("<h1>Class Based
Middleware</h1>")

```

urls.py: (application level)

```

from django.urls import path
from middlewareapp import views

urlpatterns=[
    path('',views.welcome_view),
]

```

urls.py: (project level)

```

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

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

```

settings.py:

```

MIDDLEWARE = [

    'middlewareapp.middleware.ClassBasedMiddleware',
]

```

Ex with multiple middleware's

middleware.py:

```

from django.http import HttpResponse

class FirstMiddleWare:

```

```

def __init__(self, get_response):
    self.get_response = get_response
    print("One time firstmiddleware
initialization")

def __call__(self, request):
    print("First middleware before view")
    #response=self.get_response(request)
    response=HttpResponse("Response come from
first middleware")
    print("First middleware after view")
    return response

class SecondMiddleWare:
    def __init__(self, get_response):
        self.get_response = get_response
        print("One time SecondMiddleWare
initialization")

    def __call__(self, request):
        print("Second MiddleWare before view")
        response = self.get_response(request)
        print("Second MiddleWare after view")
        return response

class ThirdMiddleWare:
    def __init__(self, get_response):
        self.get_response = get_response
        print("One time ThirdMiddleWare
initialization")

    def __call__(self, request):
        print("Third middleware before view")
        response = self.get_response(request)
        print("Third middleware after view")
        return response

```

views.py:

```
from django.http import HttpResponse

def myview(request):
    print("My view")
    return HttpResponse("This is my view")
```

urls.py: (application level)

```
from django.urls import path
from middlewareapp import views

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

settings.py:

```
MIDDLEWARE = [
    'middlewareapp.middleware.FirstMiddleWare',
    'middlewareapp.middleware.SecondMiddleWare',
    'middlewareapp.middleware.ThirdMiddleWare',

]
```

Ex with site under construction middleware:**middleware.py:**

```
from django.http import HttpResponse

class SiteUnderConstructionMiddleware:
    def __init__(self,get_response):
        self.get_response=get_response
```

```

def __call__(self, request):
    return HttpResponse("<h1>Currently site is
under construction please visit after few
days</h1>")

```

views.py:

```

from django.http import HttpResponse

def home_view(request):
    return HttpResponse("<h1>This is home
page</h1>")

```

urls.py: (application level)

```

from django.urls import path
from middlewareapp import views

urlpatterns=[
    path('', views.home_view),
]

```

settings.py:

```

MIDDLEWARE = [

'middlewareapp.middleware.SiteUnderConstructionMidd
leware',

]

```

Ex to show meaningful information when exception raised:**middleware.py:**

```

from django.http import HttpResponseRedirect

class ErrorMessageMiddleware:
    def __init__(self, get_response):
        self.get_response=get_response

    def __call__(self, request):
        return self.get_response(request)

    def process_exception(self, request, exception):
        return HttpResponseRedirect("<h1>Currently we are
facing some technical issues</h1>")

```

views.py :

```

from django.http import HttpResponseRedirect

def home_view(request):
    print(10/0)
    return HttpResponseRedirect("<h1>This is home
page</h1>")

```

urls.py: (application level)

```

from django.urls import path
from middlewareapp import views

urlpatterns=[
    path('',views.home_view),
]

```

settings.py:


```
MIDDLEWARE = [  
  
    'middlewareapp.middleware.ErrorMessageMiddleware',  
  
]
```

Signals:

- Signals allow certain senders to notify a set of receivers that some action has taken place.
- Django includes a “signal dispatcher” which helps decoupled applications get notified when actions occur elsewhere in the framework.
- The signals are utilities that help us to connect events with actions.
- We can develop a function that will run when a signal calls it.
- Signals are used to perform some action on modification/creation of a particular entry in Database.
- 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.

Types of signals:

1. Login and Logout signal
2. Model signals
3. Request/Response signals
4. Management signals

Login and Logout signal:

- `user_logged_in(sender,request,user)`
- `user_logged_out(sender,request,user)`
- `user_login_failed(sender,credentials,request)`

Ex:

- Create a new application with the name appsignal
- Install appsignal into settings.py inside installed apps.

- Create a new python file with the name **signals.py** inside the application.

signals.py:

```
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

#decorator connect
@receiver(user_logged_in,sender=User)
def login_success(sender,request,user,**kwargs):
    print("Logged in signal")
    print("Sender:",sender)
    print("Request:",request)
    print("User:",user)
    print(f'kwargs:{kwargs}')
```

#manual connect
#user_logged_in.connect(login_success,sender=User)

```
@receiver(user_logged_out, sender=User)
def log_out(sender, request, user, **kwargs):
    print("Logout signal")
    print("Sender:", sender)
    print("Request:", request)
    print("User:", user)
    print(f'kwargs:{kwargs}')
```

user_logged_out.connect(log_out,sender=User)

```
@receiver(user_login_failed)
def login_failed(sender,credentials, request,
**kwargs):
    print("Login failed signal")
    print("Sender:", sender)
    print("Request:", request)
    print("Credentials:", credentials)
```

```

        print(f'kwargs:{kwargs}')

# user_login_faield.connect(login_failed)

```

apps.py:

```

from django.apps import AppConfig

class AppsignalConfig(AppConfig):
    default_auto_field =
'django.db.models.BigAutoField'
    name = 'appsignal'

    def ready(self):
        import appsignal.signals

```

__init__.py:

```

default_app_config='appsignal.apps.AppsignalConfig'

```

Note: Execute the following commands in terminal window before start server.

- Python manage.py makemigrations
- Python manage.py migrate
- Python manage.py createsuperuser

Model signals:

Model signals are present in django.db.models.signals

- pre_init(sender,args,kwargs)
- post_init(sender,instance)
- pre_save(sender,instance,using,update_fields)
- post_save(sender,instance,created,using,update_fields)

- `pre_delete(sender,instance,using)`
- `post_delete(sender,instance,using)`

Ex:

signals.py:

```
from django.contrib.auth.models import User
from django.dispatch import receiver
from django.db.models.signals import
pre_init,post_init,pre_save,post_save,pre_delete,po
st_delete
```

```
@receiver(pre_save,sender=User)
def before_save(sender,instance, **kwargs):
    print("Pre save signal")
    print("Sender:", sender)
    print("Instance:", instance)
    print(f'kwargs:{kwargs}')
```

```
# pre_save.connect(before_save,sender=User)
```

```
@receiver(post_save,sender=User)
def after_save(sender,instance,created, **kwargs):
    if created:
        print("Post save signal")
        print("Sender:", sender)
        print("Instance:", instance)
        print("Created:",created)
        print(f'kwargs:{kwargs}')
```

```
    else:
        print("Post save signal")
        print("update")
        print("Sender:", sender)
        print("Instance:", instance)
        print("Created:", created)
        print(f'kwargs:{kwargs}')
```

```

# post_save.connect(after_save,sender=User)

@receiver(pre_delete,sender=User)
def before_delete(sender,instance, **kwargs):
    print("Pre delete signal")
    print("Sender:", sender)
    print("Instance:", instance)
    print(f'kwargs:{kwargs} ')

# pre_delete.connect(before_delete,sender=User)

@receiver(post_delete,sender=User)
def after_delete(sender,instance, **kwargs):
    print("Post delete signal")
    print("Sender:", sender)
    print("Instance:", instance)
    print(f'kwargs:{kwargs} ')

# post_delete.connect(after_delete,sender=User)

@receiver(pre_init,sender=User)
def before_init(sender,*args , **kwargs):
    print("Pre init signal")
    print("Sender:", sender)
    print(f'args:{args} ')
    print(f'kwargs:{kwargs} ')

# pre_init.connect(before_init,sender=User)

@receiver(post_init,sender=User)
def after_init(sender,*args , **kwargs):
    print("Post init signal")
    print("Sender:", sender)
    print(f'args:{args} ')
    print(f'kwargs:{kwargs} ')

# post_init.connect(after_init,sender=User)

```

Request/Response signals:**Request/Response signals are present in `django.core.signals`**

- `request_started()`
- `request_finished()`
- `got_request_exception()`

Ex:**signals.py:**

```

from django.contrib.auth.models import User
from django.dispatch import receiver

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

@receiver(request_started)
def before_request(sender, environ, **kwargs):
    print("Request started signal")
    print("Sender:", sender)
    print("Environ:", environ)
    print(f'kwargs:{kwargs}')

# request_started.connect(before_request)

@receiver(request_finished)
def after_request(sender, **kwargs):
    print("Request finished signal")
    print("Sender:", sender)
    print(f'kwargs:{kwargs}')

# request_finished.connect(after_request)

@receiver(got_request_exception)

```

```
def request_exception(sender ,request, **kwargs):
    print("Request Exception signal")
    print("Sender:", sender)
    print("Request:", request)
    print(f'kwargs:{kwargs}')

# got_request_exception.connect(request_exception)
```

views.py:

```
from django.shortcuts import render,HttpResponse

# Create your views here.
def v1(request):
    result=10/0
    return HttpResponse("welcome")
```

urls.py: (project level)

```
from django.contrib import admin
from django.urls import path
from appsignal import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('v1', views.v1),
]
```

Model Inheritance:

- Model inheritance in Django works almost identically to the way normal class inheritance works in Python, but the basics at the beginning of the page should still be followed. That means the base class should subclass `django.db.models.Model`.

Types of model inheritance:

1. Abstract base class model inheritance
2. Multi table inheritance
3. Proxy model inheritance

Abstract base class model inheritance:

- If several model classes having common fields, then it is not recommended to write these fields separately in every model class, it increases the length of the code and reduces reusability.
- We can separate the common fields into another model class, which is known as Base class.
- If we extend base class automatically common fields will be inherited to the child classes.

Ex:

models.py:

```
from django.db import models

# Create your models here.
class Commoninfo(models.Model):
    name = models.CharField(max_length=40)
    age = models.IntegerField()
    date=models.DateField()

    class Meta:
        abstract = True

class Trainer(Commoninfo):
    address = models.CharField(max_length=30)
    date =None

class Teacher(Commoninfo):
    date=models.DateTimeField()
    salary=models.IntegerField()
```


admin.py:

```

from .models import Teacher,Trainer
# Register your models here.

@admin.register(Trainer)
class TrainerAdmin(admin.ModelAdmin):
    list_display = ['id','name','age','address']

@admin.register(Teacher)
class TeacherAdmin(admin.ModelAdmin):
    list_display = ['id','name','date','salary']

```

- **Go to terminal and execute the following commands**
 - Python manage.py makemigrations
 - Python manage.py migrate
 - Python manage.py createsuperuser
 - Login into admin panel and insert few records records

To display records to the user:**Views.py:**

```

from django.shortcuts import render
from .models import Teacher,Trainer

# Create your views here.

def display(request):
    trainer_data=Trainer.objects.all()
    teacher_data=Teacher.objects.all()
    return render(request,'display.html',

{'trainers':trainer_data,'teachers':teacher_data})

```

urls.py: (project level) :

```

from django.contrib import admin
from django.urls import path
from modelinheritanceapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.display),
]

```

display.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        table, th, tr, td {
            border: 1px solid black
        }
    </style>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
width, initial-scale=1.0">
    <title>Title</title>
</head>
<body>

<table>
<h3>Trainer Information</h3>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>Address</th>
{% for trainer in trainers %}
    <tr>
        <td>{{trainer.id}}</td>
        <td>{{trainer.name}}</td>

```

```

        <td>{{trainer.age}}</td>
        <td>{{trainer.address}}</td>
    </tr>
    {% endfor %}
</table>

<table>
<h3>Teacher Information</h3>
    <th>ID</th>
    <th>Name</th>
    <th>Age</th>
    <th>Date</th>
    <th>Salary</th>
    {% for teacher in teachers %}
        <tr>
            <td>{{teacher.id}}</td>
            <td>{{teacher.name}}</td>
            <td>{{teacher.age}}</td>
            <td>{{teacher.date}}</td>
            <td>{{teacher.salary}}</td>
        </tr>
    {% endfor %}
</table>

</body>
</html>

```

Multi table inheritance:

- In this inheritance each model has their own database table, which means base class and child class will have their own table.
- In multi-table inheritance, each model corresponds to a database table.
- Django creates a OneToOneField field for the relationship in the child's model to its parent.

- Django would include an automatically generated OneToOneField field in the Text model and create a database table for each model.

Ex:

models.py:

```
from django.db import models

# Create your models here.

class Bank(models.Model):
    bname=models.CharField(max_length=20)
    baddress=models.CharField(max_length=20)

class BankManager(Bank):
    name=models.CharField(max_length=20)
    age=models.IntegerField()
```

admin.py:

```
from django.contrib import admin
from .models import Bank,BankManager

# Register your models here.
@admin.register(Bank)
class BankAdmin(admin.ModelAdmin):
    list_display = ['id','bname','baddress']

@admin.register(BankManager)
class BankManagerAdmin(admin.ModelAdmin):
    list_display =
    ['id','bname','baddress','name','age']
```

- **Go to terminal and execute the following commands**
 - Python manage.py makemigrations
 - Python manage.py migrate

- Python manage.py createsuperuser
- Login into admin panel and insert few records records

views.py:

```
from django.shortcuts import render
from .models import Bank,BankManager

# Create your views here.
def bankdetails(request):
    bank_data=Bank.objects.all()
    manager_data=BankManager.objects.all()
    return
render(request,'display.html',{'bank':bank_data,'bankmanager':manager_data})
```

urls.py: (project level)

```
from django.contrib import admin
from django.urls import path
from multitableinheritance import views

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

]
```

display.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
    table, th, tr, td {
        border: 1px solid black
    }
```

```

</style>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-
width, initial-scale=1.0">
  <title>Title</title>
</head>
<body>

<table>
<h3>Bank Information</h3>
  <th>ID</th>
  <th>BName</th>
  <th>BAddress</th>
  {% for b in bank %}
    <tr>
      <td>{{b.id}}</td>
      <td>{{b.bname}}</td>
      <td>{{b.baddress}}</td>
    </tr>
  {% endfor %}
</table>

<table>
<h3>Bank Manager Information</h3>
  <th>ID</th>
  <th>BName</th>
  <th>BAddress</th>
  <th>Name</th>
  <th>Age</th>
  {% for bm in bankmanager %}
    <tr>
      <td>{{bm.id}}</td>
      <td>{{bm.bname}}</td>
      <td>{{bm.baddress}}</td>
      <td>{{bm.name}}</td>
      <td>{{bm.age}}</td>
    </tr>
  {% endfor %}

```

```
</table>

</body>
</html>
```

Multi table inheritance:

- In this inheritance each model has their own database table, which means base class and child class will have their own table.
- In multi-table inheritance, each model corresponds to a database table.
- Django creates a OneToOneField field for the relationship in the child's model to its parent.
- Django would include an automatically generated OneToOneField field in the Text model and create a database table for each model.

Ex:

models.py:

```
from django.db import models

# Create your models here.

class Bank(models.Model):
    bname=models.CharField(max_length=20)
    baddress=models.CharField(max_length=20)

class BankManager(Bank):
    name=models.CharField(max_length=20)
    age=models.IntegerField()
```

admin.py:

```
from django.contrib import admin
from .models import Bank,BankManager

# Register your models here.
```

```

@admin.register(Bank)
class BankAdmin(admin.ModelAdmin):
    list_display = ['id', 'bname', 'baddress']

@admin.register(BankManager)
class BankManagerAdmin(admin.ModelAdmin):
    list_display =
    ['id', 'bname', 'baddress', 'name', 'age']

```

- **Go to terminal and execute the following commands**
 - Python manage.py makemigrations
 - Python manage.py migrate
 - Python manage.py createsuperuser
 - Login into admin panel and insert few records records

views.py:

```

from django.shortcuts import render
from .models import Bank, BankManager

# Create your views here.
def bankdetails(request):
    bank_data=Bank.objects.all()
    manager_data=BankManager.objects.all()
    return
render(request, 'display.html', {'bank':bank_data, 'bankmanager':manager_data})

```

urls.py: (project level)

```

from django.contrib import admin
from django.urls import path
from multitableinheritance import views

urlpatterns = [

```



```

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

]

```

display.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        table, th, tr, td {
            border: 1px solid black
        }
    </style>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
width, initial-scale=1.0">
    <title>Title</title>
</head>
<body>

<table>
<h3>Bank Information</h3>
    <th>ID</th>
    <th>BName</th>
    <th>BAddress</th>
    {% for b in bank %}
        <tr>
            <td>{{b.id}}</td>
            <td>{{b.bname}}</td>
            <td>{{b.baddress}}</td>
        </tr>
    {% endfor %}
</table>

<table>
<h3>Bank Manager Information</h3>
    <th>ID</th>

```

```

<th>BName</th>
<th>BAddress</th>
<th>Name</th>
<th>Age</th>
{% for bm in bankmanager %}
    <tr>
        <td>{{bm.id}}</td>
        <td>{{bm.bname}}</td>
        <td>{{bm.baddress}}</td>
        <td>{{bm.name}}</td>
        <td>{{bm.age}}</td>
    </tr>
{% endfor %}
</table>

</body>
</html>

```

Proxy model inheritance:

- The main Usage of a proxy model is to override the main functionality of existing Model.
- It is a type of model inheritance without creating a new table in Database. It always query on original model with overridden methods or managers.

```

class University(models.Model):
    fields...

```

```

class College(University):
    class Meta:
        proxy=True

```

- In admin panel if we add a new record to either university or college, then automatically those changes will be reflected to other model view.

Ex:

models.py:

```

from django.db import models

# Create your models here.
class University(models.Model):
    uname=models.CharField(max_length=20)
    ulocation=models.CharField(max_length=20)

class College(University):
    class Meta:
        proxy=True
        #ordering=['id']
        ordering=['ulocation']

```

admin.py :

```

from .models import University,College

# Register your models here.
@admin.register(University)
class UnivercityAdmin(admin.ModelAdmin):
    list_display = ['id','uname','ulocation']

@admin.register(College)
class CollegeAdmin(admin.ModelAdmin):
    list_display = ['id','uname','ulocation']

```

- **Go to terminal and execute the following commands**
 - Python manage.py makemigrations
 - Python manage.py migrate
 - Python manage.py createsuperuser
 - Login into admin panel and insert few records records

views.py:

```

from django.shortcuts import render
from .models import University, College

# Create your views here.
def home(request):
    university_data=University.objects.all()
    College_data=College.objects.all()
    return
render(request, 'display.html', {'university':university_data, 'college':College_data})

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from proxymodelapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home),
]

```

display.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        table, th, tr, td {
            border: 1px solid black
        }
    </style>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Title</title>

```

```

</head>
<body>

<table>
<h3>Univercity Information</h3>
  <th>ID</th>
  <th>UName</th>
  <th>ULocation</th>
  {% for u in univercity %}
    <tr>
      <td>{{u.id}}</td>
      <td>{{u.uname}}</td>
      <td>{{u.ulocation}}</td>
    </tr>
  {% endfor %}
</table>

<table>
<h3>College Information</h3>
  <th>ID</th>
  <th>UName</th>
  <th>ULocation</th>
  {% for c in college %}
    <tr>
      <td>{{c.id}}</td>
      <td>{{c.uname}}</td>
      <td>{{c.ulocation}}</td>
    </tr>
  {% endfor %}
</table>

</body>
</html>

```

Model Relationship:

Types of database relationships

- One to One Relationship
- Many to One Relationship
- Many to Many Relationships

One to One Relationship:

- To define a one-to-one relationship, use `OneToOneField`.
- A One-to-One relationship is a type of Relationship where both tables can have only one record on either side.
- A one-to-one relationship is like a relationship between a husband and a wife.

Ex:

Models.py :

```
from django.db import models
from django.contrib.auth.models import User
# Create your models here.

class Book(models.Model):

    #user=models.OneToOneField(User,on_delete=models.CASCADE,primary_key=True)
    #user = models.OneToOneField(User,
    on_delete=models.PROTECT, primary_key=True)
    user = models.OneToOneField(User,
on_delete=models.CASCADE, primary_key=True,

    limit_choices_to={'is_staff':True})
    book_name=models.CharField(max_length=20)
    book_author=models.CharField(max_length=20)
    book_publishdate=models.DateField()
```

admin.py :

```
from django.contrib import admin
from .models import Book
# Register your models here.
@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    list_display =
    ['book_name', 'book_author', 'book_publishdate', 'user
    ']
```

Many to One Relationship:

- A Many to One relationship is a type of relationship where the first model can have one or many records in the second table and the second table can be related only to one record in the first table. This is one of the most commonly used relationships.
- A Many to One relationship is like a relationship between you and your parents. Your parents can have any number of children where you can have only one parent.
- To perform many to one relationship, we will be using the Foreign Key method in Django.

EX:

models.py:

```
from django.db import models
from django.contrib.auth.models import User
# Create your models here.

class Post(models.Model):

    #user=models.ForeignKey(User,on_delete=models.CASCADE)
    #user = models.ForeignKey(User,
    on_delete=models.PROTECT)
    user = models.ForeignKey(User,
```

```

on_delete=models.SET_NULL,null=True)
    post_title=models.CharField(max_length=20)
    post_category=models.CharField(max_length=30)
    post_publish_date=models.DateField()

```

admin.py:

```

from django.contrib import admin
from .models import Post
# Register your models here.
@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display =
    ['post_title', 'post_category', 'post_publish_date', '
    user']

```

Many to Many Relationships:

- In this type of relationship, each record of the first table is related to many records of the second table and also each record of the second table is related to many records of the first table.
- To perform many to many relationships, we will be using the ManyToManyField in Django.

models.py:

```

from django.db import models
from django.contrib.auth.models import User
# Create your models here.

class Dance(models.Model):

```



```

user=models.ManyToManyField(User)
dance_name=models.CharField(max_length=20)
dance_duration=models.IntegerField()

def dance_by(self):
    return ",".join([str(d) for d in
self.user.all()])

```

admin.py:

```

from django.contrib import admin
from .models import Dance

# Register your models here.
@admin.register(Dance)
class DanceAdmin(admin.ModelAdmin):
    list_display =
['dance_name', 'dance_duration', 'dance_by']

```

Query set API:

- Internally, a Query Set can be constructed, filtered, sliced, and generally passed around without actually hitting the database. No database activity actually occurs until you do something to evaluate the query set
1. **Query property:** this property is used to get sql query of queryset.
Syntax: queryset.query
 2. **Retrieving all objects:**
 - **all():** This method is used to retrieve all objects, it will return a copy of current query set
Syntax: Modelname.objects.all()
Ex: Employee.objects.all()

3. Retrieving specific objects:

- Filter (**kwargs): it will return a new queryset containing objects that match the given lookup parameters.
- Filter () will always give a queryset even if only a single object matches the query.
- **syntax:** modelname.objects.filter(eaddress='hyderabad')

4. exclude(**kwargs):

- It will return a new query set containing objects that do not match the given lookup paramaters.
- **syntax:** modelname.objects.exclude(eaddress='hyderabad')

5. order_by(*fields): It will order the fields

'field' : ascending order

'-field': descending order

'?' : randomly

reverse(): this will work only when there is ordering in queryset.

Ex:

models.py:

```
from django.db import models

# Create your models here.
class Employee(models.Model):
    name=models.CharField(max_length=20)
    eid=models.IntegerField(unique=True,null=False)
    eaddress=models.CharField(max_length=30)
    age=models.IntegerField()
    dob=models.DateField()
```

admin.py :

```

from django.contrib import admin
from myapp.models import Employee

# Register your models here.
@admin.register(Employee)
class EmployeeAdmin(admin.ModelAdmin):
    list_display =
    ['id', 'name', 'eid', 'eaddress', 'age', 'dob']

```

- **Go to terminal and execute the following commands**

- Python manage.py makemigrations
- Python manage.py migrate
- Python manage.py createsuperuser
- Login into admin panel and insert few records records

views.py:

```

from django.shortcuts import render
from myapp.models import Employee

# Create your views here.

def v1(request):
    #employee_data=Employee.objects.all()

    #employee_data=Employee.objects.filter(eaddress='hy
    derabad')
    #employee_data =
    Employee.objects.exclude(eaddress='hyderabad')
    #employee_data =
    Employee.objects.order_by('eid')
    #employee_data = Employee.objects.order_by('-
    eid')
    #employee_data = Employee.objects.order_by('?')
    employee_data =
    Employee.objects.order_by('eid').reverse()[1:4]

```

```

        print("Return type:",employee_data)
        print("SQL query:",employee_data.query)
        return
render(request,'display.html',{'Employees':employee
_data})

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.v1),
]

```

display.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        table, th, tr, td {
            border: 1px solid black
        }
    </style>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
width, initial-scale=1.0">
    <title>Title</title>
</head>
<body>

<table>
<h3>Employee Details</h3>
    <th>ID</th>

```

```

<th>Name</th>
<th>Eid</th>
<th>EAddress</th>
<th>Age</th>
<th>DOB</th>
{% for emp in Employees %}
  <tr>
    <td>{{emp.id}}</td>
    <td>{{emp.name}}</td>
    <td>{{emp.eid}}</td>
    <td>{{emp.eaddress}}</td>
    <td>{{emp.age}}</td>
    <td>{{emp.dob}}</td>
  </tr>
{% endfor %}
</table>

</body>
</html>

```

Query set API:

- **values()** : It returns a QuerySet that returns dictionaries, rather than model instances.
- Each of those dictionaries represents an object, with the keys corresponding to the attribute names of model objects.
- **values_list()** : This is similar to values() except that instead of returning dictionaries, it returns tuples when iterated over.
- **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.
- **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.
- **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.

Ex:

Views.py:

```
from django.shortcuts import render
from myapp.models import Employee

def v1(request):
    employee_data=Employee.objects.values()
    employee_data =
    Employee.objects.values('name','eaddress')
    employee_data = Employee.objects.values_list()
    employee_data =
    Employee.objects.values_list("id","name")
    employee_data = Employee.objects.values_list("id",
    "name",named=True)
    employee_data = Employee.objects.using('default')
    employee_data =
    Employee.objects.dates('dob','month')

    print("Return type:",employee_data)
    print("SQL query:",employee_data.query)
    return
    render(request,'display.html',{'Employees':employee
    _data})
```

Query set API:

- **none()** : It will create a queryset that never returns any objects and no query will be executed when accessing the results.

- **union()** : It is used 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.
- **intersection()** : INTERSECT operator to return the shared elements of two or more QuerySets.
- **difference()** : EXCEPT operator to keep only elements present in the QuerySet but not in some other QuerySets.

Models.py:

```
from django.db import models

# Create your models here.
class Employee(models.Model):
    name=models.CharField(max_length=20)
    eid=models.IntegerField(unique=True,null=False)
    eaddress=models.CharField(max_length=30)
    age=models.IntegerField()
    dob=models.DateField()

class Manager(models.Model):
    mname=models.CharField(max_length=20)
    eaddress=models.CharField(max_length=30)
    salary=models.IntegerField()
    dob=models.DateField()
```

admin.py:

```
from django.contrib import admin
from myapp.models import Employee,Manager

# Register your models here.
@admin.register(Employee)
class EmployeeAdmin(admin.ModelAdmin):
    list_display =
    ['id','name','eid','eaddress','age','dob']
```

```
@admin.register(Manager)
class ManagerAdmin(admin.ModelAdmin):
    list_display =
    ['id', 'mname', 'eaddress', 'salary', 'dob']
```

views.py:

```
from django.shortcuts import render
from myapp.models import Employee, Manager
from django.db.models import Q

# Create your views here.

def v1(request):

    #employee_data = Employee.objects.none()

    #qs1=Employee.objects.values_list('id','name',named
    =True)

    #qs2=Manager.objects.values_list('id','mname',named
    =True)
    #employee_data=qs2.union(qs1,all=True)
    #employee_data = qs2.intersection(qs1)
    #employee_data=qs1.difference(qs2)

    #employee_data=Employee.objects.filter(id=6) &
    Employee.objects.filter(age=35)

    #employee_data=Employee.objects.filter(id=6,age=35)
    #employee_data=Employee.objects.filter(Q(id=6)
    & Q(age=35))

    #employee_data =
    Employee.objects.filter(eaddress='sr nagar') |
    Employee.objects.filter(age=45)
```



```

employee_data=Employee.objects.filter(Q(address='sr nagar') | Q(age=45))

    print("Return type:",employee_data)
    print("SQL query:",employee_data.query)
    return
render(request, 'display.html', {'Employees':employee_data})

```

Query set API Fields Lookups:

- exact
- iexact
- contains
- icontains
- in
- gt(greater than)
- gte(greater than or equal to)
- lte(less than or equal to)
- lt(less than)
- range
- date
- year
- month
- day
- week day
- minute
- quarter

Ex:

Models.py:

```
from django.db import models
```

```
# Create your models here.
class Student(models.Model):
    name=models.CharField(max_length=30)
    sid=models.IntegerField(unique=True,null=False)
    address=models.CharField(max_length=30)
    marks=models.IntegerField()
    passdate=models.DateField()
    admitdate=models.DateTimeField()
```

admin.py:

```
from django.contrib import admin
from querysetapp.models import Student
# Register your models here.

@admin.register(Student)
class StudentAdmin(admin.ModelAdmin):
    list_display =
    ['id', 'name', 'sid', 'address', 'marks', 'passdate', 'admitdate']
```

- **Go to terminal and execute the following commands**
 - Python manage.py makemigrations
 - Python manage.py migrate
 - Python manage.py createsuperuser
 - Login into admin panel and insert few records records

views.py:

```
from django.shortcuts import render
from querysetapp.models import Student
from datetime import date,time

# Create your views here.
def home(request):
    #student_data=Student.objects.all()
```

```

#student_data=Student.objects.filter(name__exact='m
ohan')

#student_data=Student.objects.filter(name__iexact='
MOHAN')
#student_data =
Student.objects.filter(name__contains='n')
#student_data =
Student.objects.filter(name__icontains='hi')
#student_data =
Student.objects.filter(id__in=[2,5,7])
#student_data =
Student.objects.filter(marks__in=[90])
#student_data =
Student.objects.filter(marks__gt=80)
#student_data =
Student.objects.filter(marks__lt=80)
#student_data =
Student.objects.filter(marks__gte=75)
#student_data =
Student.objects.filter(marks__lte=75)
#student_data =
Student.objects.filter(passdate__range=('2021-09-
01','2021-09-20'))
#student_data =
Student.objects.filter(admitdate__date=date(2019,8,
6))
#student_data =
Student.objects.filter(passdate__year=2019)
#student_data =
Student.objects.filter(passdate__year__gt=2019)
#student_data =
Student.objects.filter(passdate__month__gt=9)
#student_data =
Student.objects.filter(passdate__week=35)
#student_data =
Student.objects.filter(passdate__week__day=3)

```

```

    #student_data =
    Student.objects.filter(passdate__quarter=4)
    student_data =
    Student.objects.filter(admitdate__minute__gt=20)

    print("Return:", student_data)
    print("SQL Query:", student_data.query)
    return
render(request, 'home.html', {'students': student_data
})

```

home.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        table,th,tr,td{
            border: 1px solid black;
        }
    </style>
</head>
<body>
<table>
    <h2>Student Information</h2>
    <th>Id</th>
    <th>Name</th>
    <th>Sid</th>
    <th>Address</th>
    <th>Marks</th>
    <th>Passdate</th>
    <th>Admitdate</th>
    {%for student in students %}
        <tr>

```

```

        <td>{{ student.id }}</td>
        <td>{{ student.name }}</td>
        <td>{{ student.sid }}</td>
        <td>{{ student.address }}</td>
        <td>{{ student.marks }}</td>
        <td>{{ student.passdate }}</td>
        <td>{{ student.admitdate }}</td>
    </tr>
    {% endfor %}
</table>

</body>
</html>

```

urls.py (project level) :

```

from django.contrib import admin
from django.urls import path
from querysetapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home),
]

```

Query set API Aggregate functions:

- Avg()
- Max()
- Min()
- Sum()
- Count()

Ex:

models.py:

```

from django.db import models

# Create your models here.
class Student(models.Model):
    name=models.CharField(max_length=30)
    sid=models.IntegerField(unique=True,null=False)
    address=models.CharField(max_length=30)
    marks=models.IntegerField()
    passdate=models.DateField()
    admitdate=models.DateTimeField()

```

admin.py:

```

from django.contrib import admin
from querysetapp.models import Student
# Register your models here.

@admin.register(Student)
class StudentAdmin(admin.ModelAdmin):
    list_display =
    ['id','name','sid','address','marks','passdate','admitdate']

```

views.py:

```

from django.shortcuts import render
from django.http import HttpResponse
from querysetapp.models import Student
from django.db.models import Avg,Sum,Min,Max,Count

# Create your views here.
def home(request):
    student_data=Student.objects.all()
    average=student_data.aggregate(Avg('marks'))
    total = student_data.aggregate(Sum('marks'))
    minimum = student_data.aggregate(Min('marks'))

```

```

        maximum = student_data.aggregate(Max('marks'))

totalcount=student_data.aggregate(Count('marks'))

context={'students':student_data,'average':average,
'total':total,

'minimum':minimum,'maximum':maximum,'totalcount':to
talcount}

print(average)
print("Return:",student_data)
print("SQL Query:",student_data.query)
return render(request,'home.html',context)

def add(request):
    #to insert one record

    #s=Student(name='raz',sid=110,address='hyd',marks=5
    7,passdate='2020-9-12',admitdate='2019-4-23')
    #s.save()

    #to insert multiple records

    #Student.objects.bulk_create([Student(name='sana',s
    id=111,address='hyd',marks=67,passdate='2020-9-
    12',admitdate='2019-4-23'),

    #Student(name='sam',sid=112,address='sr
    nagar',marks=87,passdate='2020-9-
    12',admitdate='2019-4-23')])

    #to delete a record
    s=Student.objects.get(sid=109)
    s.delete()

```

```
c=Student.objects.all()
return HttpResponse(c)
```

home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        table,th,tr,td{
            border: 1px solid black;
        }
    </style>
</head>
<body>
<table>
    <h2>Student Information</h2>
    <th>Id</th>
    <th>Name</th>
    <th>Sid</th>
    <th>Address</th>
    <th>Marks</th>
    <th>Passdate</th>
    <th>Admitdate</th>
    {%for student in students %}
        <tr>
            <td>{{ student.id }}</td>
            <td>{{ student.name }}</td>
            <td>{{ student.sid }}</td>
            <td>{{ student.address }}</td>
            <td>{{ student.marks }}</td>
            <td>{{ student.passdate }}</td>
            <td>{{ student.admitdate }}</td>
        </tr>
```



```

        {% endfor %}
</table>
<h1>Average marks:{{ average }}</h1>
<h1>Average marks:{{ average.marks__avg }}</h1>

<h1>Sum:{{ total }}</h1>
<h1>Sum:{{ total.marks__sum }}</h1>

<h1>Minimum:{{ minimum }}</h1>
<h1>Minimum:{{ minimum.marks__min }}</h1>

<h1>Maximum:{{ maximum }}</h1>
<h1>Maximum:{{ maximum.marks__max }}</h1>

<h1>Count:{{ totalcount }}</h1>
<h1>Count:{{ totalcount.marks__count }}</h1>

</body>
</html>

```

urls.py :

```

from django.contrib import admin
from django.urls import path
from querysetapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('home', views.home),
    path('add', views.add)
]

```

CRUD operations using MySQL:

- Go to command prompt and create a django project
Ex: django-admin startproject mycrudproject
- Create application under mycrudproject

Ex: cd mycrudproject

Ex: python manage.py startapp mycrudapp

- Open mycrudproject in pycharm editor
- Install mycrudapp under settings.py
- Go to mysql database and create a database with the name django10am

Settings.py :

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django10am',
        'USER': 'root',
        'PASSWORD': 'Shanvi@123',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

Models.py:

```
from django.db import models

# Create your models here.
class User(models.Model):
    uname=models.CharField(max_length=50)
    uemail=models.EmailField(max_length=50)
    upassword=models.CharField(max_length=50)

    class Meta:
        db_table='users'
```

- **Go to terminal and execute the following commands**
 - Python manage.py makemigrations
 - Python manage.py migrate
 - This will creates a user's table in django10am database

forms.py:

```

from django import forms
from mycrudapp.models import User

class UserForm(forms.ModelForm):
    class Meta:
        model=User
        fields='__all__'
        widgets={

'uname':forms.TextInput(attrs={'class':'form-
control'}),
        'uemail':
forms.EmailInput(attrs={'class': 'form-control'}),
        'upassword':
forms.TextInput(attrs={'class': 'form-control'}),
        }

```

views.py :

```

from django.shortcuts import render
from mycrudapp.forms import UserForm

# Create your views here.

def insert(request):
    form=UserForm()
    return
render(request, 'index.html', {'form':form})

```

- **Create a templates folder under project**
- **Go to settings.py and include template folder name**

```

TEMPLATES = [
    {
        'BACKEND':

```

```
'django.template.backends.django.DjangoTemplate
s',
    'DIRS': ['templates'],
```

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- CSS only -->
<link
href="https://cdn.jsdelivrivr.net/npm/bootstrap@5.1.1/
dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
F3w7mX95PdgyTmZZMECANGseQB83DfGTowi0iMjiWaeVhAn4FJk
qJByhZMI3AhiU" crossorigin="anonymous">
</head>
<body>
<div class="container">
    <form>
        <div class="form-group">
            <label><b>User Name:</b></label>
            {{ form.uname }}

        </div>
        <div class="form-group">
            <label><b>User Email:</b></label>
            {{ form.uemail }}

        </div>
        <div class="form-group">
            <label><b>User Passowrd:</b></label>
            {{ form.upassword }}

        </div>
```

```

        <input type="submit" name="insert"
value="insert" class="btn btn-primary">

    </form>
</div>
</body>
</html>

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from mycrudapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.insert),
]

```

CRUD operations using MySQL:

Models.py:

```

from django.db import models

# Create your models here.
class User(models.Model):
    username=models.CharField(max_length=50)
    uemail=models.EmailField(max_length=50)
    upassword=models.CharField(max_length=50)

    class Meta:
        db_table='users'

```

forms.py :

```

from django import forms
from mycrudapp.models import User

class UserForm(forms.ModelForm):
    class Meta:
        model=User
        fields='__all__'
        widgets={

'uname':forms.TextInput(attrs={'class':'form-
control'}),
        'uemail':
forms.EmailInput(attrs={'class': 'form-control'}),
        'upassword':
forms.TextInput(attrs={'class': 'form-control'}),
        }

```

views.py :

```

from django.shortcuts import render,redirect
from mycrudapp.forms import UserForm
from django.http import HttpResponseRedirect
from mycrudapp.models import User

# Create your views here.

def insert(request):
    if request.method=='POST':
        form=UserForm(request.POST)
        if form.is_valid():
            try:
                form.save()
                return HttpResponseRedirect("<h2>Data
inserted into Database</h2>")
            except:
                pass

```

```

        form=UserForm()
        return
    render(request, 'index.html', {'form':form})

def show(request):
    users=User.objects.all()
    return
    render(request, 'show.html', {'users':users})

def delete(request,id):
    user=User.objects.get(id=id)
    user.delete()
    return redirect('/show')

def edit(request,id):
    user=User.objects.get(id=id)
    return
    render(request, 'edit.html', {'user':user})

def update(request,id):
    user=User.objects.get(id=id)
    form=UserForm(request.POST,instance=user)
    if form.is_valid():
        form.save()
        return redirect('/show')
    return
    render(request, 'edit.html', {'user':user})

```

index.html :

```

<!DOCTYPE html>
<html lang="en">
<head>

```

```

    <meta charset="UTF-8">
    <title>Title</title>
    <!-- CSS only -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/
dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
F3w7mX95PdgyTmZZMECAngseQB83DfGTowi0iMjiWaeVhAn4FJk
qJByhZMI3AhiU" crossorigin="anonymous">
</head>
<body>
<div class="container">
<form method="POST" action="/">
    {% csrf_token %}
    <div class="form-group">
        <label><b>User Name:</b></label>
        {{ form.uname }}
    </div>
    <div class="form-group">
        <label><b>User Email:</b></label>
        {{ form.uemail }}
    </div>
    <div class="form-group">
        <label><b>User password:</b></label>
        {{ form.upassword }}
    </div>
    <input type="submit" name="insert"
value="insert" class="btn btn-primary">
</form>
</div>
</body>
</html>

```

Show.html :

```

<!DOCTYPE html>
<html lang="en">

```



```

<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <!-- CSS only -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/
dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
F3w7mX95PdgyTmZZMECANGseQB83DfGTowi0iMjiWaeVhAn4FJk
qJByhZMI3AhiU" crossorigin="anonymous">
</head>
<body>
<div class="container">
  <table class="table table-border table-bordered
text-center shadow">
    <tr>
      <th>User Id</th>
      <th>User Name</th>
      <th>User Email</th>
      <th>User Password</th>
      <th>Delete User</th>
      <th>Edit User</th>
    </tr>
    {% for user in users %}
      <tr>
        <td>{{ user.id }}</td>
        <td>{{ user.username }}</td>
        <td>{{ user.email }}</td>
        <td>{{ user.password }}</td>
        <td><a href="/delete/{{ user.id }}"
class="btn btn-danger">Delete</a></td>
        <td><a href="/edit/{{ user.id }}"
class="btn btn-success">Edit</a></td>
      </tr>
    {% endfor %}
  </table>

</div>

```

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

edit.html :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <!-- CSS only -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/
dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
F3w7mX95PdgyTmZZMECAngseQB83DfGTowi0iMjiWaeVhAn4FJk
qJByhZMI3AhiU" crossorigin="anonymous">
</head>
<body>
<div class="container">
<form method="POST" action="/update/{{ user.id }}">
  {% csrf_token %}
  <div class="form-group">
    <label><b>User Name:</b></label>
    <input type="text" name="uname" class="form-
control" value="{{ user.uname }}">
  </div>
  <div class="form-group">
    <label><b>User Email:</b></label>
    <input type="text" name="uemail" class="form-
control" value="{{ user.uemail }}">
  </div>
  <div class="form-group">
    <label><b>User password:</b></label>
    <input type="text" name="upassword"
class="form-control" value="{{ user.upassword }}">
  </div>
```

```

        <input type="submit" name="update"
value="update" class="btn btn-success">
</form>
</div>
</body>
</html>

```

Urls.py :

```

from django.contrib import admin
from django.urls import path
from crudapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.insert),
    path('show', views.show),
    path('delete/<int:id>', views.delete),
    path('edit/<int:id>', views.edit),
    path('update/<int:id>', views.update),
]

```

Uploading image and display on webpage:

- Go to command prompt and create a django project
Ex: django-admin startproject imageuploadproject
- Create application under imageuploadproject
Ex: cd imageuploadproject
Ex: python manage.py startapp imageuploadapp
- Open imageuploadproject in pycharm editor
- Install imageuploadapp under settings.py

models.py:

```

from django.db import models

# Create your models here.

class ImageUploadModel(models.Model):
    title=models.CharField(max_length=20)
    image=models.ImageField(upload_to='uploads')

    class Meta:
        db_table='users'

```

forms.py:

```

from django import forms
from imageuploadapp.models import ImageUploadModel

class ImageUploadForm(forms.ModelForm):
    class Meta:
        model=ImageUploadModel
        fields='__all__'

```

- **Go to terminal and execute the following commands**

- Python manage.py makemigrations
- Python manage.py migrate
- This will creates a user's table in sqlite3 database

Views.py:

```

from django.shortcuts import render
from django.http import HttpResponse
from imageuploadapp.forms import ImageUploadForm
from imageuploadapp.models import ImageUploadModel

# Create your views here.
def upload_file(request):

```

```

if request.method=='POST':

form=ImageUploadForm(request.POST,request.FILES)
    if form.is_valid():
        form.save()
        return HttpResponseRedirect("<h2>Image upload
successfully..</h2>")
    else:
        form=ImageUploadForm()
        images=ImageUploadModel.objects.all()
        return
render(request,'index.html',{'form':form,'images':i
mages})

```

- **Create a templates folder under project**
- **Go to settings.py and include template folder name**
- ```
TEMPLATES = [
 {
 'BACKEND':
'django.template.backends.django.DjangoTemp
lates',
 'DIRS': ['templates'],
```

#### **settings.py:**

```
MEDIA_ROOT='C:\\Users\\lenovo\\Desktop\\imageupload
project\\media'
```

#### **index.html:**

```

<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Title</title>

```

```

 <!-- CSS only -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/
dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
F3w7mX95PdgyTmZZMECAngseQB83DfGTowi0iMjiWaeVhAn4FJk
qJByhZMI3AhiU" crossorigin="anonymous">

</head>

<body>
<form method="post" enctype="multipart/form-data"
action="/">
{% csrf_token %}
 {{ form.as_p }}
 <button type="submit">upload</button>
</form>

<div class="container">
 <div class="row">
 {% for i in images %}
 <div class="col-sm-4">
 <div class="card m-2">

 </div>
 </div>
 {% endfor %}
 </div>
</div>
</body>
</html>

```

**urls.py:**

```

from django.contrib import admin
from django.urls import path

```

```

from imageuploadapp import views
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
 path('admin/', admin.site.urls),
 path('', views.upload_file)
]+static(settings.MEDIA_URL, document_root=settings.
MEDIA_ROOT)

```

### Django-crispy-forms:

- Django-crispy-forms is **an application that helps to manage Django forms**.
- It allows adjusting forms' properties (such as method, send button or CSS classes) on the backend without having to re-write them in the template.
- It will let you control the rendering behavior of your Django forms in a very elegant and DRY way.

### EX:

- **Create a new project and application**

### Working with crispy forms:

- **Installing django-crispy-forms:**
- **Go to terminal and type the following command**
- `pip install django-crispy-forms`
- **settings.py:**
- `INSTALLED_APPS = [
 'django.contrib.admin',
 'django.contrib.auth',
 'django.contrib.contenttypes',
 'django.contrib.sessions',
 'django.contrib.messages',
 'django.contrib.staticfiles',`

```

 'crispyformapp',
 'crispy_forms'
]
 CRISPY_TEMPLATE_PACK = 'bootstrap4'

```

- 
- **For loading django-crispy-forms tags, add**
- {%load crispy\_forms\_tags %} On the top of your django template
- Now to style any form with django-crispy-forms, replace
- 
- {{ form }}
- with
- {{ form|crispy }}
- 

#### models.py :

```

from django.db import models

GENDER_OPTIONS=(('male','Male'), ('female','Female')
)
Create your models here.
class Customer(models.Model):
 name=models.CharField(max_length=50)

 email=models.EmailField(max_length=50,blank=True)

 gender=models.CharField(max_length=50,choices=GENDE
R_OPTIONS,null=True)
 city=models.CharField(max_length=50)

 class Meta:
 db_table='customers'

```

#### forms.py:



```

from django import forms
from crispyformapp.models import
Customer, GENDER_OPTIONS

class CustomerForm(forms.ModelForm):

gender=forms.ChoiceField(choices=GENDER_OPTIONS,

widget=forms.RadioSelect, initial='male')
 class Meta:
 model=Customer
 fields='__all__'

```

**views.py:**

```

from django.shortcuts import render
from crispyformapp.forms import CustomerForm
Create your views here.

def customerview(request):
 form=CustomerForm()
 return
render(request, 'home.html', {'form':form})

```

**base.html:**

```

<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
 <!-- CSS only -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/
dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-

```

```

F3w7mX95PdgyTmZZMECAngseQB83DfGTowi0iMjiWaeVhAn4FJk
qJByhZMI3AhiU" crossorigin="anonymous">
</head>
<body>

<div class="container">
 <div class="row justify-content-center">
 <div class="col-7">
 <h1>Customer Details</h1>
 {% block content %}
 {% endblock %}
 </div>
 </div>
</div>
</body>
</html>

```

### home.html:

```

<!DOCTYPE html>
{% extends 'base.html' %}
{% load crispy_forms_filters %}
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
 {% block content %}
 <form method="post" novalidate>
 {% csrf_token %}
 {{ form|crispy }}
 <button type="submit" class="btn btn-
primary">Save Customer</button>
 <button type="submit" class="btn btn-
danger">Cancel</button>
 </form>

```

```
{% endblock content%}
</body>
</html>
```

- **Go to terminal and execute the following commands**
- Python manage.py makemigrations
- Python manage.py migrate
- Python manage.py runserver

**THANK YOU**

