Password change by built in method

Always remember that password change is only done after successful login

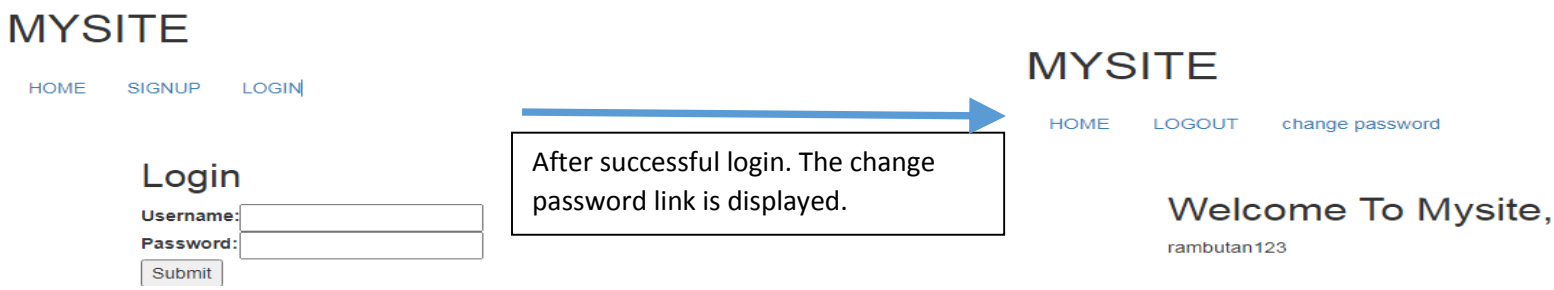
- **password_change** is the name of the u.r.l, when we use it in the link as shown below, django will automatically connect to the view which is defined as built in.
- Set the link inside the base.html page as shown below figure, in such a way that only after successful login the link to change password should be shown in the navigation bar.

Base.html

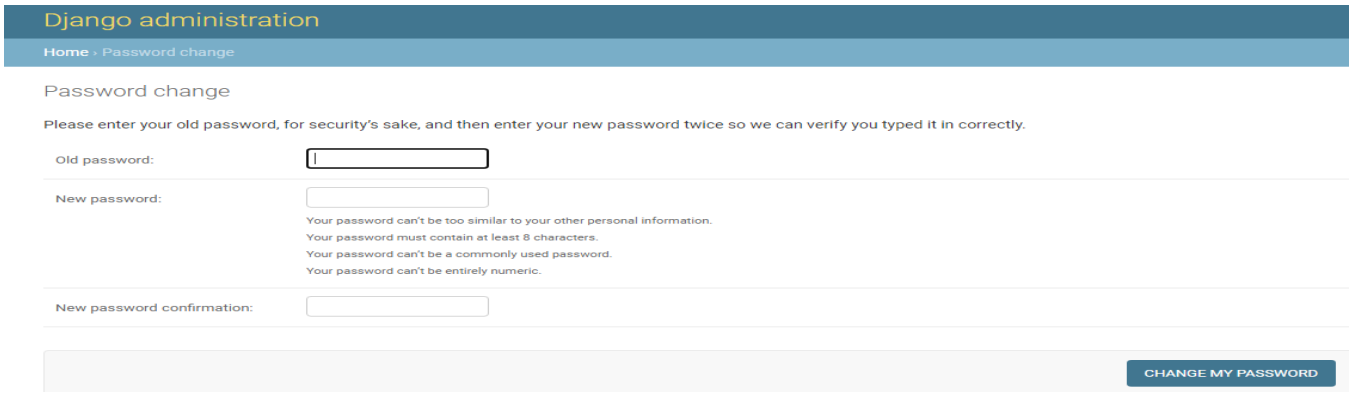
```
<body>
<div class="container-fluid">
    <h1>MYSITE</h1>
    <nav class="navbar navbar">
        <ul class="nav navbar-nav">
            <li><a href="{% url 'home' %}">HOME</a></li>
            {% if user.is_authenticated %}
             <li><a href="{% url 'logout' %}">LOGOUT</a></li>
             <li><a href="{% url 'password_change' %}">change password</a></li>
            {% else %}
               <li><a href="{% url 'signup1' %}">SIGNUP</a></li>
             <li><a href="{% url 'login' %}">LOGIN</a></li>
           {% endif %}
        </ul>
    </nav>
    {% block body_block %}
    {% endblock %}
</div>
</body>
```

Only after login the link of change password will be displayed on the navigation bar.

- After **python manage.py runserver**



- After clicking change password the page displayed is djangos own way i.e; not in the customized way as shown below.



- So we have to customize it. we create two html files inside registration folder (inside templates folder).we have to exactly use the same name of html files. django will inbuiltly connect to this html files.
  password_change_form.html:-Here coding can be done in two ways
  1.built in way:-here, only submit button is given,.rest will be provided by django.
  2.user-defined way:-here, we type all the boxes and give the same fieldname which is defined inside inbuilt class `PasswordChangeForm` (inside django.contrib.auth.forms) in the input tag.To be more clear, **old_password** for old password,**new_password1** for new password and **new_password2** for confirmation of new password.

  password_change_done.html:-to show the message that password changed successfully.

password_change_form.html by built in way

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Login{% endblock%}
{% block body_block %}
<div class="container">
<h2>Change Password</h2>
    <form method="POST">
        {% csrf_token %}
        <table>
            {{form.as_table}}
        </table>
        <input type="submit">
    </form>
</div>
{% endblock %}
```

password_change_form.html by user defined way

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Change password{% endblock%}
{% block body_block %}
<div class="container">
<h2>Password Change-User Defined</h2>
     <form class="form-horizontal" method="POST">
         {% csrf_token %}
    <div class="form-group">
      <label class="control-label col-sm-2" >Old Password:</label>
      <div class="col-sm-10">
        <input type="password" class="form-control"  placeholder="Enter old password" name="old_password">
      </div>
    </div>
    <div class="form-group">
      <label class="control-label col-sm-2" >New Password:</label>
      <div class="col-sm-10">
        <input type="password" class="form-control" placeholder="Enter new password" name="new_password1">
      </div>
    </div>
    <div class="form-group">
      <label class="control-label col-sm-2" >Confirmation Password:</label>
      <div class="col-sm-10">
        <input type="password" class="form-control" placeholder="Enter confirmation password" name="new_password2">
      </div>
    </div>
    <div class="form-group">
      <div class="col-sm-offset-2 col-sm-10">
        <button type="submit" class="btn btn-default">Submit</button>
      </div>
    </div>
  </form>
</div>
```

password_change_done.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Login{% endblock%}
{% block body_block %}
<div class="container">
<h2>Change Password</h2>
    <p>PASSWORD CHANGED SUCCESSFULLY</p>
</div>
{% endblock %}
```

As a conclusion, we created two html pages named **password_change_form.html** (either built in or user defined) and **password_change_done.html** inside **registration** folder, which is inside **templates** folder

To enter more details during sign up

- As per the method we discussed above in the procedure of sign up, we can only enter details such as username, password and confirmation of password.
- Here, in this topic we are going to study, how to enter more details in sign up such as date of birth, email id etc..
  There are two methods to achieve this:-1.customuser model
  2.one-to-one field.
- Here we are going to study about the first method i.e; 1.customuser model

1.customuser method

```
CustomUser

signup--->user model
          UserCreationForm
Custom User
1).models.py -model definition

class CustomUser(AbstractUser):
      email=models.EmailField()
      phno=models.IntegerField()
      address=models.CharField(max_length=20)

2).Forms.py-Form definition

class CustomUserCreationForm(UserCreationForm):
          class Meta:
                model=CustomUser
                fields=UserCreationForm.Meta.fields+('email','phno','address')
```

```
3)settings.py

AUTH_USER_MODEL="app1.C
ustomUser"

4)views.py

5)templates
```

- Earlier method of sign up:-only created **views.py**
  :-used built in table in django i.e; **usermodel**. So,no need of **models.py**
  :-used built in form in django i.e;**UserCreationForm**.So, no need of **form.py**
- New method(custom user method to add more details):-
  - In this method we are creating a table in **models.py** as a subclass of the built in table in django named **Usermodel**.
  - Also we are creating a form in form.py as a subclass of the built in form in django named **UserCreationForm**.
  - We create **views.py.**

Therefore, in the method of customuser model we creating extra details by extending the built in details in signup(i.e; username,password and confirmation of password)

models.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser
class CustomUser(AbstractUser):
    email=models.EmailField()
    phno=models.IntegerField()
    address=models.CharField(max_length=20)
```

1.we have to import models from django database.

2.we have to import the built in table named **Usermodel** also known as **Abstractuser** from **django.contrib.auth.models**.

3.Here we give the name of subclass as customuser, but remember that we can give any name for the subclass.

forms.py

```
from django.contrib.auth.forms import UserCreationForm
from app1.models import CustomUser
class CustomUserCreationForm(UserCreationForm):
    class Meta(UserCreationForm.Meta):
        model=CustomUser
        fields=UserCreationForm.Meta.fields+('email','phno','address')
```

We can give any name for the subclass.

Statement to add extra details with existing details

settings.py

```
# Application definition
AUTH_USER_MODEL="app1.CustomUser"
```

This statement means that we are merging our subclass named customuser which is created in models.py of app1 with the built in table named **Usermodel**.

views.py for built in

```
from app1.forms import CustomUserCreationForm
def signup(request):
    form=CustomUserCreationForm()
    if(request.method=="POST"):
        form=CustomUserCreationForm(request.POST)
        if(form.is_valid()):
            form.save()
            return home(request)
    return render(request,'signup.html',{'form':form})
```

The subclass name of form is used here

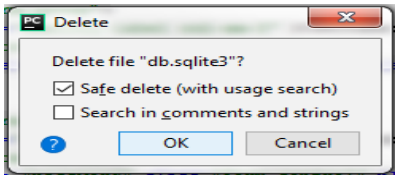
views.py for userdefined

```
from app1.forms import CustomUserCreationForm
def signup1(request):
    if (request.method == "POST"):
        form = CustomUserCreationForm(request.POST)
        if (form.is_valid()):
            form.save()
            return home(request)
    return render(request,'signup1.html')
```

Don't forget to delete **db.sqlite3** in the left side bar above manage.py to avoid unnecessary errors.

db.sqlite3
manage.py

After right click on db.sqlite3 following will come.



Only tick the first one,then click ok

- Apply **python manage.py makemigrations** and **python manage.py migrate**

Signup.html for built in

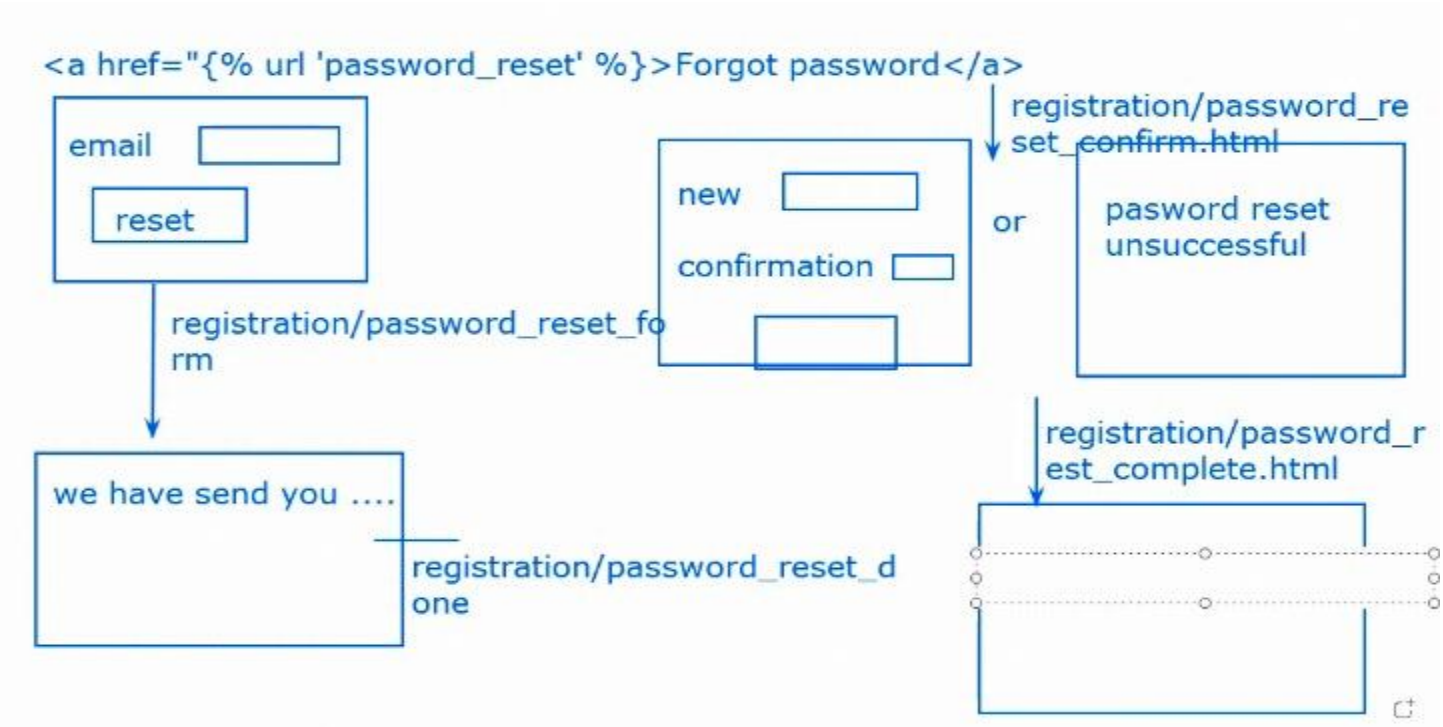
```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Signup{% endblock%}
{% block body_block %}
<div class="container">
<h2>SIGNUP-Built in(CUSTOMUSER)</h2>
    <form method="POST">
        {% csrf_token %}
        <table>
            {{form.as_table}}
        </table>
        <input type="submit">
    </form>
</div>
{% endblock %}
```

Signup1.html for userdefined

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Signup{% endblock%}
{% block body_block %}
<div class="container">
<h2>SIGNUP-User Defined(CustomUser)</h2>

     <form class="form-horizontal" method="POST">
        {% csrf_token %}
    <div class="form-group">
      <label class="control-label col-sm-2" >Username:</label>
      <div class="col-sm-10">
        <input type="text" class="form-control"  placeholder="Enter username" name="username">
      </div>
    </div>
    <div class="form-group">
      <label class="control-label col-sm-2" >Password:</label>
      <div class="col-sm-10">
        <input type="password" class="form-control" placeholder="Enter password" name="password1">
      </div>
    </div>
    <div class="form-group">
      <label class="control-label col-sm-2" >Password:</label>
      <div class="col-sm-10">
        <input type="password" class="form-control" placeholder="Enter confirmation password" name="password2">
      </div>
    </div>
        <div class="form-group">
      <label class="control-label col-sm-2" >email:</label>
      <div class="col-sm-10">
        <input type="text" class="form-control"  placeholder="Enter username" name="email">
      <label class="control-label col-sm-2" >phone:</label>
      <div class="col-sm-10">
        <input type="text" class="form-control"  placeholder="Enter username" name="phno">
      </div></div>
                <div class="form-group">
      <label class="control-label col-sm-2" >address:</label>
      <div class="col-sm-10">
        <input type="text" class="form-control"  placeholder="Enter username" name="address">
      </div></div>
    <div class="form-group">
      <div class="col-sm-offset-2 col-sm-10">
        <button type="submit" class="btn btn-default">Submit</button>
      </div>
    </div>
  </form>
</div>
{% endblock %}
```

Password reset (password forgot)



- **password_reset** is the name of the url for the forgot password. view will be inbuiltly called inside django ,so we do not want to define view.py.

Always remember that forgot password in the login page

- Create a link in the login.html(inside registration folder which is inside templates folder) for forgot password.

login.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Login{% endblock%}
{% block body_block %}
<div class="container">
<h2>Login</h2>
    <form method="POST">
        {% csrf_token %}
        <table>
            {{form.as_table}}
        </table>
        <input type="submit">
    </form>
    <a href="{% url 'password_reset' %}">Forgot password</a>
</div>
{% endblock %}
```

The link to the url for password reset is given as link here.

The name of the entire url is **password_reset** which is given here

- After clicking the forgot password, email id (which we entered during sign up) will be asked. Since we are working in our own local server, django will send the link to reset the password to the pycharm terminal. For this purpose we have to add a code in settings.py.

Settings.py

```
# Application definition
AUTH_USER_MODEL="app1.CustomUser"
EMAIL_BACKEND = "django.core.mail.backends.console.EmailBackend"
```

This is the code to be added in the settings.py

- To customize all operations we have to create 4 webpages:
  1. **password_reset_form.html:-**to type the email id.
  2. **password_reset_done.html:-**to show link send successfully.
  3. **password_reset_confirm.html:-**either return login page or unsuccessful message(invalid email id).
  4. **password_reset_complete.html:-**to successful completion and have a link to go the login page.
- The above pages is set inside registration folder inside templates folder. The pages will work automatically one by one after we click the forgot password button in login page.

password_reset_form.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Login{% endblock%}
{% block body_block %}
<div class="container">
<h2>Password Reset</h2>
    <form method="POST">
        {% csrf_token %}
        <table>
            {{form.as_table}}
        </table>
        <input type="submit">
    </form>

</div>
{% endblock %}
```

password_reset_done.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Login{% endblock%}
{% block body_block %}
<div class="container">
<h2>Password Reset Sent</h2>
<p>We've emailed you instructions for setting your password, if an account exists with the email you entered. You should receive them shortly.
If you don't receive an email, please make sure you've entered the address you registered with, and check your spam folder.
</p>
</div>
{% endblock %}
```

password_reset_confirm.html

The maid that we give should be exactly the same that we gave during sign up.

If validlink will check whether the email id we input is valid or not. if valid a page to enter the password will be shown. Otherwise, the page will print invalid email id

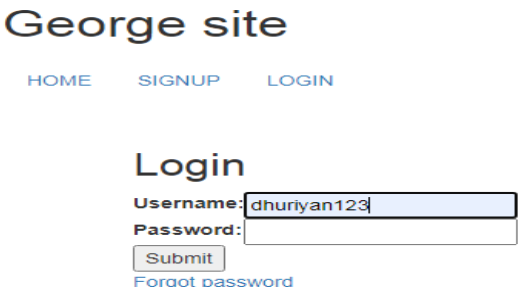
```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Login{% endblock%}
{% block body_block %}
<div class="container">
<h2>Password Reset</h2>
    {% if validlink %}
    <form method="POST">
        {% csrf_token %}
        <table>
            {{form.as_table}}
        </table>
        <input type="submit">
    </form>
    {% else %}
    <h2>Password reset unsuccessful</h2>
<p>The password reset link was invalid, possibly because it has already been used. Please request a new password reset.
</p>
    {% endif %}
</div>
{% endblock %}
```

password_reset_complete.html

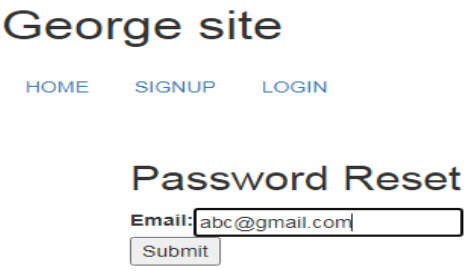
```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Login{% endblock%}
{% block body_block %}
<div class="container">
<h2>Password reset complete</h2>
    <p>Your password has been set. You may go ahead and log in now.</p>
<a href="{% url 'login' %}">LOGIN</a>
</div>
{% endblock %}
```

Link to go back to the login page is given

login.html



password_reset_form.html



password_reset_done.html

George site

HOME SIGNUP LOGIN

Password Reset Sent

We've emailed you instructions for setting your password, if an account exists with the email you entered. You should receive them shortly. If you don't receive an email, please make sure you've entered the address you registered with, and check your spam folder.

The link send to the terminal of pycharm

Terminal: Local ×

```
You're receiving this email because you requested a password reset for your user account at 127.0.0.1:8000.

Please go to the following page and choose a new password:

http://127.0.0.1:8000/accounts/reset/MQ/ar9r7u-24ea490e1f696c7b4ed7617f2c0b94d0/
```

password_reset_confirm.html

George site

HOME SIGNUP LOGIN

Password Reset

New password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

New password confirmation:

Submit

password_reset_complete.html

George site

HOME SIGNUP LOGIN

Password reset complete

Your password has been set. You may go ahead and log in now.

LOGIN

Login

2.user defined view.

- As compared to built in view of login, here in userdefined view of login we have to do everything manually.
- First create eg:-login1.html (we can use any name and can be anywhere inside template folder)

login1.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Signup{% endblock%}
{% block body_block %}
<div class="container">
<h2>Login(User Defined)</h2>
     <form class="form-horizontal" method="POST">
         {% csrf_token %}
    <div class="form-group">
      <label class="control-label col-sm-2" >Username:</label>
      <div class="col-sm-10">
        <input type="text" class="form-control"  placeholder="Enter username" name="username">
      </div>
    </div>
    <div class="form-group">
      <label class="control-label col-sm-2" >Password:</label>
      <div class="col-sm-10">
        <input type="password" class="form-control" placeholder="Enter password" name="password">
      </div>
    </div>
    <div class="form-group">
      <div class="col-sm-offset-2 col-sm-10">
        <button type="submit" class="btn btn-default">Submit</button>
      </div>
    </div>
  </form>
</div>
{% endblock %}
```

Since, we are doing userdefined way we can give any name

- Set url for login1.html

```
url(r'^login1/$', views.user_login, name="login1"),
```

- Use the 3rd method of form (without using form object) in **views.py**

```
from django.contrib.auth import authenticate,login,logout
def user_login(request):
    if(request.method=="POST"):
        u=request.POST['username']
        p=request.POST['password']
        user=authenticate(username=u,password=p)
        if user:
            login(request,user)
            return home(request)
        else:
            return HttpResponse("Invalid Details")
    return render(request,'login1.html')
```

login and logout are built in functions for login and logout operations.so, we have to import it.

Authenticate function which is used to check whether the entered login details is correct or not

- Make changes in **settings.py** as shown below:-
  - In built in view of login we wrote a code in **settimgs.py**, but to avoid errors we have to make that code a comment as shown below.

```
#LOGIN_REDIRECT_URL="home"
#LOGOUT_REDIRECT_URL="login"
```

  - After this we have to add a new code in **settings.py** as shown below. This is to inform the django that we are doing the login process in user defined way.

```
LOGIN_URL="login1"
```

login1 is the name of the html page where we are doing the login process in user defined way

- Next we have to set logout procedure.

views.py

```
def user_logout(request):
    logout(request)
    return user_login(request)
```

- Set url for logout

Urls.py

```
url(r'^logout1/$', views.user_logout, name="logout1"),
```

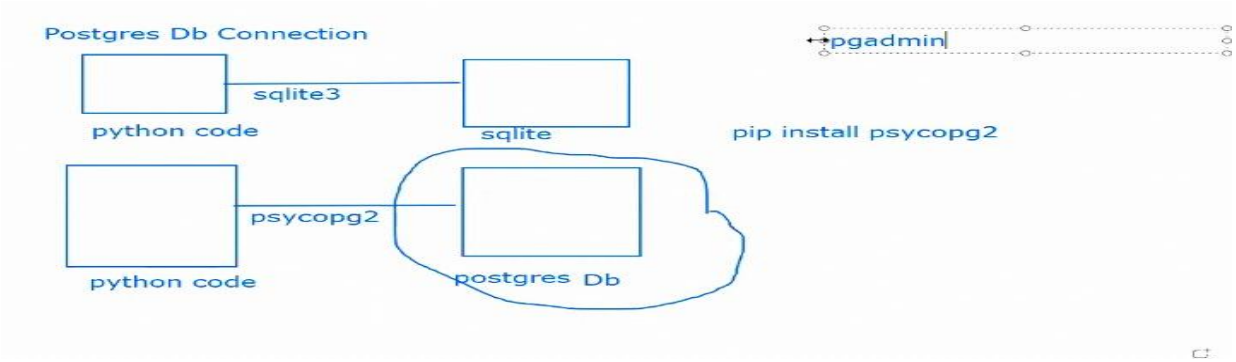
- Set link for login1 and logout1 in base.html

Base.html

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
</head>
<body>
<div class="container-fluid">
    <h1>George site</h1>
    <nav class="navbar navbar">
        <ul class="nav navbar-nav">
            <li><a href="{% url 'home' %}">HOME</a></li>
            {% if user.is_authenticated %}
             <li><a href="{% url 'logout1' %}">LOGOUT</a></li>
             <li><a href="{% url 'password_change' %}">change password</a></li>
            {% else %}
               <li><a href="{% url 'signup1' %}">SIGNUP</a></li>
             <li><a href="{% url 'login1' %}">LOGIN</a></li>
           {% endif %}
        </ul>
    </nav>
    {% block body_block %}
    {% endblock %}
</div>
</body>
</html>
```

Here, we have to set the link for login and logout which is named in the url as login1 and logout1 respectively.

Postgresql



- Sqlite database is already installed (built in) in django.To use sqlite we have the package sqlite3 inside it(by default).
- But to use postgresql as a database, we have to install the package for postgresql database named **psycopg2**
  To install psycopg2 we use the command:- **pip install psycopg2**

To remember,

- Sometimes we need to update pip package to install psycopg2. For this, we use the command:-

  python -m pip install --upgrade pip

- Sometimes, the older version will need not get updated. So, we have to forcefully update it. For this we use the command:-

  python -m pip install -U --force-reinstall pip

- Why we use postgresql?
  Ans:- Postgresql is a user interface database in which we can see the entire modifications in the databes.
  But in sqlite we do not have this user interface facility.

- To connect to the postgresql with the django, we have to change the default database i.e; sqlite to postgresql in the **settings.py**.

Settings.py

```
'''
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',

    }
}'''

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'Authentication',
        'USER': 'postgres',
        'PASSWORD': 'george123',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Here we made the the default database i.e;sqlite as a comment.

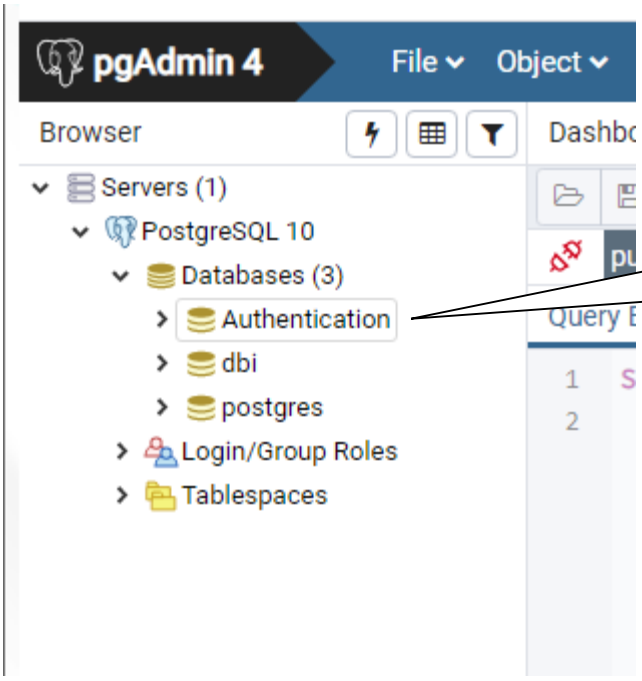
- This is code to include postgresql to django which can copy pasted from google.

  NAME

  Should be exactly same name as the name we created for the database in postgresql software

  PASSWORD and PORT

  Should be exactly the same password and port number which we give during installing postgresql software.

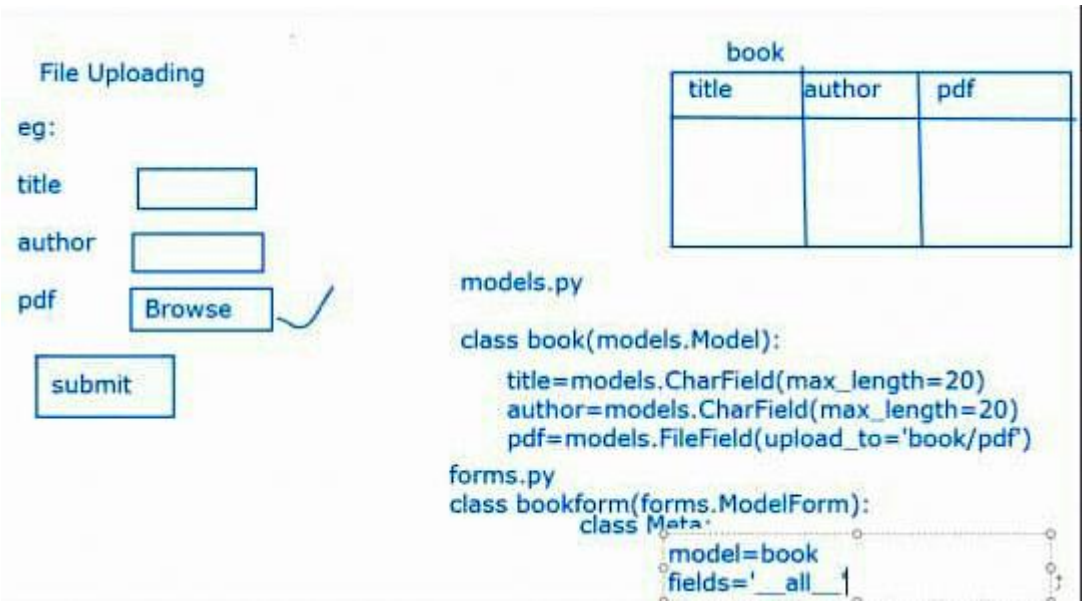


This is the name of the database we created (eg:- authentication) just by right clicking. The same name (here, eg:- authentication) should be given inside setting.py for new database.

Remember,

- If we already created table using the default database and done migrations, then we have to delete the migrations of default database from the migrations folder to avoid errors.
- After deleting the migrations of default database, we have to apply migrations for our new database which is postgresql using our usual command :-**python manage.py makemigrations**
  Python manage.py migrate

File uploading



- Create a table in models.py

models.py

```
from django.db import models
class book(models.Model):
    title=models.CharField(max_length=20)
    author=models.CharField(max_length=20)
    pdf=models.FileField(upload_to='book/pdf')
```

This means that the file is to be saved inside pdf folder of book folder of media folder

forms.py

```
from django import forms
from newbook.models import book
class bookform(forms.ModelForm):
    class Meta:
        model=book
        fields='__all__'
```

views.py

```
from django.shortcuts import render
from newbook.forms import bookform
from newbook.models import book
def home(request):
    return render(request,'home.html')
def upload(request):
    form=bookform()
    if(request.method=="POST"):
        form=bookform(request.POST,request.FILES)
        if(form.is_valid()):
            form.save()
            return home(request)
    return render(request,'upload.html',{'form':form})
def booklist(request):
    b=book.objects.all()
    return render(request,'list.html',{'b':b})
```

1st method of form creation:- built in way

Here, we have used two parameters request.post and request.files. The reason is that:- request.post will carry the value for column named title and author. But to save the path of the uploaded file in the column pdf we have to use request.files.

Data retrieval operation, which we have already discussed in the previous classes

Settings.py

```
MEDIA_ROOT=os.path.join(BASE_DIR,'media')
MEDIA_URL="/media/"
```

Urls.py of project url

```
from django.contrib import admin
from django.conf.urls import url
from newbook import views
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
   url('admin/', admin.site.urls),
    url(r'^$',views.home,name="home"),
    url(r'^upload/$', views.upload, name="upload"),
    url(r'^list/$', views.booklist, name="list"),
]
if(settings.DEBUG):
    urlpatterns+=static(settings.MEDIA_URL,document_root=settings.MEDIA_ROOT)
```

This is the important part

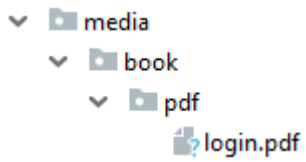
As we know that, after we click the download link in the web page the small sized file will be displayed in the webpage while the large sized file will be automatically downloaded. For this purpose we have to give the statement in this way.

Admin.py

```
from django.contrib import admin
from newbook.models import book
admin.site.register(book)
```

Since, here we are using the admin database of django

New folder media



- The media folder is used to store the files that we upload in the web page.
- The media folder will be automatically created in the outer container similar to template folder.
- The media folder will be automatically created when we upload a file in the web page. The sub folder named book and pdf will also be created by default as the media folder is created.
- Inside the pdf folder as shown in the figure, the files uploaded are stored.
- But for proper working of media folder we have to specify about the folder in settings.py.

Remember,

- Here, we created a table named book. Inside the table book we created columns named title,author and pdf.
- But in the pdf column only the reference (path) of the uploaded file will be saved. The reason we have already discussed that as we upload a file in the web page, it will be automatically save to the media folder. So, the path to the saved file in the media folder is shown in the column pdf.

Base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
</head>
<body>
<div class="container-fluid">
    <nav class="navbar navbar-inverse">
        <ul class="nav navbar-nav">
            <li><a href="{% url 'home' %}">HOME</a></li>
            <li><a href="{% url 'upload' %}">ADD BOOKS</a></li>
            <li><a href="{% url 'list' %}">VIEW BOOKS</a></li>
        </ul>
    </nav>
    {% block body_block %}
    {% endblock %}
</div>
</body>
</html>
```

Upload.html:-to add new books

List.html:-to view and download the uploaded books

home.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}HOME{% endblock %}
{% block body_block %}
<div class="container">
<h2>Welcome To HOME</h2>
    {% endblock %}
    </div>
```

upload.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}HOME{% endblock %}
{% block body_block %}
<div class="container">
<h2>ADD BOOK DETAILS</h2>
<form method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    <table>
        {{form.as_table}}
    </table>
    <input type="submit">
</form>
</div>
{% endblock %}
```

enctype="multipart/form-data" :-is used to avoid the manipulation of uploaded file.
Without this some errors may occur

List.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}HOME{% endblock %}
{% block body_block %}
<div class="container">
<h2>Book Details</h2>
    <table class="table table-bordered">
        <tr>
            <th>TITLE</th>
             <th>AUTHOR</th>
             <th>PDF</th>
        </tr>
        {% for i in b %}
        <tr>
            <td>{{i.title}}</td>
              <td>{{i.author}}</td>
              <td><a href="{{i.pdf.url}}">DOWNLOAD</a></td>
        </tr>
        {% endfor %}
    </table>
</div>
{% endblock %}
```

This is to create a link to download the file.

{{i.pdf.url}}:-to mention the the file to be downloaded.

How to upload image?

- **Just changes in program for adding image is mentioned here.**

models.py

The uploading image is saved in book/cover folder of media folder.

```
cover = models.ImageField(upload_to='book/cover', null=True, blank=True)
```

- To work image we have to install a package named pillow:-

pip install pillow

list.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}HOME{% endblock %}
{% block body_block %}
<div class="container">
<h2>Book Details</h2>
    <table class="table table-bordered">
        <tr>
            <th>TITLE</th>
             <th>AUTHOR</th>
             <th>PDF</th>
        </tr>
        {% for i in b %}
        <tr>
            {% if i.cover %}
            <td><img src="{{i.cover.url}}" height="100px" width="100px"></td>{% else %}
            <td>NO COVER</td>{% endif %}
            <td>{{i.title}}</td>
              <td>{{i.author}}</td>
              <td><a href="{{i.pdf.url}}">DOWNLOAD</a></td>
        </tr>
        {% endfor %}
    </table>
</div>
{% endblock %}
```

Update list.html with this statements inside if to display image.

- For proper working delete already existing migrations and give new migrations.

list.html will display like this after entering data

Book Details

TITLE	AUTHOR	PDF	
	hello	just	DOWNLOAD

How to DELETE, EDIT and VIEW?

1. Delete

list.html

- We have to add link for delete in list.html which means writing front end coding.

```
{% block title %}HOME{% endblock %}
{% block body_block %}
<div class="container">
<h2>Book Details</h2>
    <table class="table table-bordered">
        <tr>
            <th>COVER</th>
            <th>TITLE</th>
             <th>AUTHOR</th>
             <th>PDF</th>
<th>DELETE/th>
            <th>VIEW</th>
            <th>EDIT</th>
        </tr>
        {% for i in b %}
        <tr>
            {% if i.cover %}
            <td><img src="{{i.cover.url}}" height="1                    00px"></td>{% else %}
            <td>NO COVER</td>{% endif %}
            <td>{{i.title}}</td>
              <td>{{i.author}}</td>
              <td><a href="{{i.pdf.url}}">DOWNLO</a>
            <td><a href="{% url 'delete_book' i.pk %}" class="btn btn-danger">DELETE</a></td>
             <td><a href="" class="btn btn-danger">EDIT</a></td>
        </tr>
        {% endfor %}
    </table>
</div>
{% endblock %}
```

In delete, after clicking the delete button the entire row should be deleted. We know that if we do not set a column in the table as primary key, then django will automatically create a column named ID. The use this column ID is that we can uniquely identify each row (i.e; Id =1 for 1st row ,id=2 for 2nd row,etc..)

pk:- represents primary key. Here, primary key (pk) is ID.

i:-represents value of a particular column in each row (i.e; 1st row,2nd row etc...)

i.pk:-is similar to i.ID

views.py (here, we write back end coding for delete)

```
def delete_book(request,p):
    b=book.objects.get(pk=p)
    b.delete()
    return booklist(request)
```

urls.py

```
url(r'delete_book/(?P<p>[0-9]+)',views.delete_book,name="delete_book"),
```

<p> :- this p should be used as parameter in views.py. i.e; def delete_book(request,p):

: –we know that we already give {% url 'delete_book' i.pk %}, in the front end (list.html). therefore, when we click delete button for a particular row, the i.pk will identify the corresponding id value of the row. This id value is saved in the variable named p which is represented as <p> in the url address.

:-Here, id (primary key) is integer type so we gave regular expression as [0-9] in the url address.

:-if the value passed is character type we use regular expression as [a-z] in the url address.

- before deleting

Book Details

COVER	TITLE	AUTHOR	PDF	DELETE
	hello	just	DOWNLOAD	DELETE

- After deleting

Book Details

COVER	TITLE	AUTHOR	PDF	DELETE

2. VIEW

- After clicking view button we can see the full details of the thing.

list.html

```
<td><a href="{% url 'view_book' i.pk %}" class="btn btn-danger">VIEW</a></td>
```

views.py

```
def view_book(request,p):
    b=book.objects.get(pk=p)
    return render(request,'book.html',{'b':b})
```

When we click the view button, the control goes to book.html page to show the entire details of the book

book.html

```
{% extends 'base.html' %}
{% load static %}
{% block body_block %}
<div class="container-fluid">
    <h2 style="..."><b>BOOK DETAILS</b></h2>
<table class="table table-boredered">
    <tr>
        <th>TITLE</th>
        <td>{{b.title}}</td>
    </tr>
     <tr>
        <th>AUTHOR</th>
        <td>{{b.author}}</td>
    </tr>
     <tr>
        <th>PDF</th>
        <td><a href="{{b.pdf.url}}">DOWNLOAD</a></td>
    </tr>
    <tr>
        <th>COVER</th>
        <td><img src="{{b.cover.url}}" width="100px" height="100px"></td>
    </tr>
</table>
</div>
{% endblock %}
```

Urls.py

```
url(r'view_book/(?P<p>[0-9]+)', views.view_book, name="view_book"),
```

- Before clicking view

Book Details

COVER	TITLE	AUTHOR	PDF	DELETE/th>	VIEW
	hello	just	DOWNLOAD	DELETE	VIEW

- After clicking view

BOOK DETAILS

TITLE	hello
AUTHOR	just
PDF	DOWNLOAD
COVER	

3.EDIT

list.html

```
<td><a href="{% url 'edit_book' i.pk %}" class="btn btn-danger">EDIT</a></td>
```

views.py

```
def edit_book(request,p):
    b=book.objects.get(pk=p)
    form = bookform(instance=b)
    if (request.method == "POST"):
        form = bookform(request.POST, request.FILES,instance=b)
        if (form.is_valid()):
            form.save()
            return home(request)
    return render(request, 'upload.html', {'form': form})
```

Here we used the keyword instance for editing

url.py

```
url(r'edit_book/(?P<p>[0-9]+)', views.edit_book, name="edit_book"),
```

- Upto here, we discussed the topics file **uploading** and **authentication.** Now, we are going to implement **authentication** in **file uploading** i.e; to do the operations of entire **file uploading** we have to first **sign up** in the website and then we to **login**. Only after successful **login** we can perform the entire process of **file uploading**. Here, we are going to discuss it.

Signup.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Signup{% endblock%}
{% block body_block %}
<div class="container">
<h2>SIGNUP</h2>
    <form method="POST">
        {% csrf_token %}
        <table>
            {{form.as_table}}
        </table>
        <input type="submit">
    </form>
</div>
{% endblock %}
```

login1.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Signup{% endblock%}
{% block body_block %}
<div class="container">
<h2>Login(User Defined)</h2>
    <form class="form-horizontal" method="POST">
        {% csrf_token %}
   <div class="form-group">
     <label class="control-label col-sm-2" >Username:</label>
     <div class="col-sm-10">
       <input type="text" class="form-control"  placeholder="Enter username" name="username">
     </div>
   </div>
   <div class="form-group">
     <label class="control-label col-sm-2" >Password:</label>
     <div class="col-sm-10">
       <input type="password" class="form-control" placeholder="Enter password" name="password">
     </div>
   </div>
   <div class="form-group">
     <div class="col-sm-offset-2 col-sm-10">
       <button type="submit" class="btn btn-default">Submit</button>
     </div>
   </div>
  </form>
</div>
{% endblock %}
```

Models.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser
class book(models.Model):
    title=models.CharField(max_length=20)
    author=models.CharField(max_length=20)
    pdf=models.FileField(upload_to='book/pdf')
    cover = models.ImageField(upload_to='book/cover', null=True, blank=True)
class CustomUser(AbstractUser):
    email=models.EmailField()
    phno=models.IntegerField()
    address=models.CharField(max_length=20)
```

Forms.py

```
from django import forms
from newbook.models import book,CustomUser
from django.contrib.auth.forms import UserCreationForm
class bookform(forms.ModelForm):
    class Meta:
        model=book
        fields='__all__'
class CustomUserCreationForm(UserCreationForm):
    class Meta(UserCreationForm.Meta):
        model=CustomUser
        fields=UserCreationForm.Meta.fields+('email','phno','address')
```

Base.html

- Edit base.html of file uploading in following way to implement authentication.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
</head>
<body>
<div class="container-fluid">
    <nav class="navbar navbar-inverse">
        <ul class="nav navbar-nav">
            <li><a href="{% url 'home' %}">HOME</a></li>
            {% if user.is_authenticated %}
            <li><a href="{% url 'upload' %}">ADD BOOKS</a></li>
            <li><a href="{% url 'list' %}">VIEW BOOKS</a></li>
            <li><a href="{% url 'logout1' %}">LOGOUT</a></li>
            {% else %}
            <li><a href="{% url 'signup1' %}">SIGNUP</a></li>
            <li><a href="{% url 'login1' %}">LOGIN</a></li>
{% endif %}
        </ul>
    </nav>
    {% block body_block %}
    {% endblock %}
</div>
</body>
</html>
```

Settings.py

```
LOGIN_URL="login1"
AUTH_USER_MODEL="newbook.CustomUser"
```

Views.py

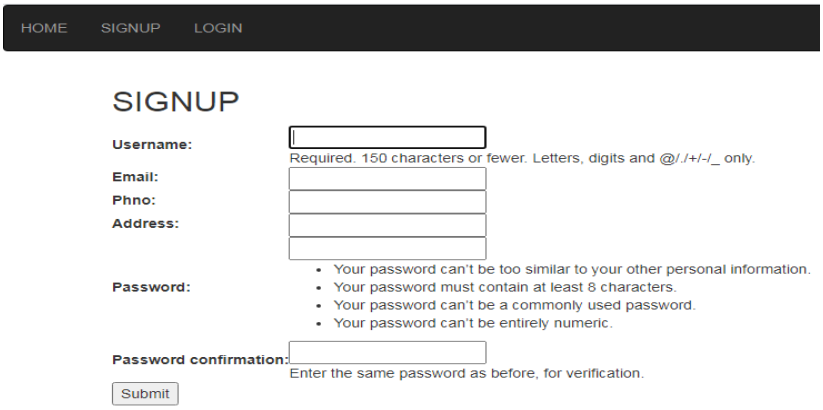
```
def signup(request):
    form=CustomUserCreationForm()
    if(request.method=="POST"):
        form=CustomUserCreationForm(request.POST)
        if(form.is_valid()):
            form.save()
            return home(request)
    return render(request,'signup.html',{'form':form})
def user_login(request):
    if(request.method=="POST"):
        u=request.POST['username']
        p=request.POST['password']
        user=authenticate(username=u,password=p)
        if user:
            login(request,user)
            return home(request)
        else:
            return HttpResponse("Invalid Details")
    return render(request,'login1.html')
def user_logout(request):
    logout(request)
    return user_login(request)
```

Here,we used 3rd method of form input for login

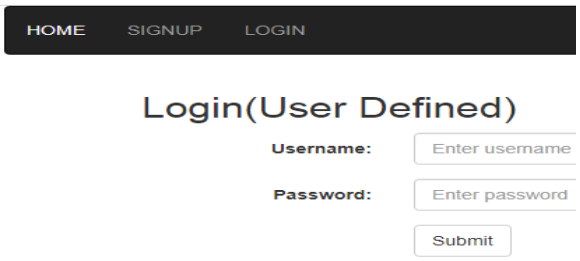
Urls.py

```
url(r'^sign1/$', views.signup, name="signup1"),
url(r'^login1/$', views.user_login, name="login1"),
url(r'^logout1/$', views.user_logout, name="logout1"),
```

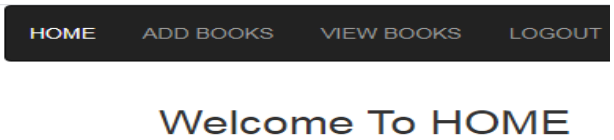
Before sign up look navigation bar



Before login



After login navigation bar becomes



Welcome To HOME

- Thus only after successful login the entire process of file uploading can be done.

- We discussed the way in which only after successful login the entire process of file uploading can be done. But, here we can access the file uploading page by just typing the url address without login. To rectify the problem, the corresponding functions for pages to be displayed after login which is given in **views.py** is started with the statement **@login_required()** .

Views.py

```
from django.shortcuts import render
from newbook.forms import bookform,CustomUserCreationForm
from newbook.models import book
from django.contrib.auth import authenticate,login,logout
from django.contrib.auth.decorators import login_required
def home(request):
    return render(request,'home.html')
@login_required()
def upload(request):
    form=bookform()
    if(request.method=="POST"):
        form=bookform(request.POST,request.FILES)
        if(form.is_valid()):
            form.save()
            return home(request)
    return render(request,'upload.html',{'form':form})
@login_required()
def booklist(request):
    b=book.objects.all()
    return render(request,'list.html',{'b':b})
@login_required()
def delete_book(request,p):
    b=book.objects.get(pk=p)
    b.delete()
    return booklist(request)
@login_required()
def view_book(request,p):
    b=book.objects.get(pk=p)
    return render(request,'book.html',{'b':b})
@login_required()
def edit_book(request,p):
    b=book.objects.get(pk=p)
    form = bookform(instance=b)
    if (request.method == "POST"):
        form = bookform(request.POST, request.FILES,instance=b)
        if (form.is_valid()):
            form.save()
            return home(request)
    return render(request, 'upload.html', {'form': form})
def signup(request):
    form=CustomUserCreationForm()
    if(request.method=="POST"):
        form=CustomUserCreationForm(request.POST)
        if(form.is_valid()):
            form.save()
            return home(request)
    return render(request,'signup.html',{'form':form})
def user_login(request):
    if(request.method=="POST"):
        u=request.POST['username']
        p=request.POST['password']
        user=authenticate(username=u,password=p)
        if user:
            login(request,user)
            return home(request)
        else:
            return HttpResponse("Invalid Details")
    return render(request,'login1.html')
@login_required()
def user_logout(request):
    logout(request)
    return user_login(request)
```

Import @login_required() in this way

REMEMBER

b=book.objects.all():-To select entire records in a table
b=book.objects.get():-To select a particular record or row in a table.
b=book.objects.filter():-To select all matching records.
b=book.objects.create():-To create a particular row in the table.

Here we used the statement @login_required() before the functions related to the html pages which should be displayed only after successful login.

SEARCH

- Add link for search page in **base.html**

```
<li><a href="{% url 'search' %}">SEARCH BOOKS</a></li>
```

The link should be given in such a way that the link should be displayed only after successful login

Search.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Signup{% endblock%}
{% block body_block %}
<div class="container">
<form method="POST">
    {% csrf_token %}
<div class="form-group">
<div class="col-lg-5">
<input type="text" name="srh" class="form-control" placeholder="Enter name">
</div>
<label class="col-lg-2">
<button type="submit" class="btn btn-danger">Search</button>
</label>
</div>
</form>
    {% if b %}
    <br>
    <br>
    <h2>Book Details</h2>
    <table class="table table-bordered">
        <tr>
            <th>COVER</th>
            <th>TITLE</th>
             <th>AUTHOR</th>
             <th>PDF</th>
        </tr>
        {% for i in b %}
        <tr>
            {% if i.cover %}
            <td><img src="{{i.cover.url}}" height="100px" width="100px"></td>{% else %}
            <td>NO COVER</td>{% endif %}
            <td>{{i.title}}</td>
                <td>{{i.author}}</td>
                <td><a href="{{i.pdf.url}}">DOWNLOAD</a>
        </tr>
        {% endfor %}
    </table>
    {% endif %}
    {% if messages %}
<ul class="messages">
    {% for message in messages %}
    <br>
    <br>
    <li>{{ message }}</li>
    {% endfor %}
</ul>
{% endif %}
</div>
{% endblock %}
```


 tag is to provide blank line

Views.py

```
from django.contrib.auth.decorators import login_required
from django.db.models import Q
from django.contrib import messages
@login_required()
def search_book(request):
    if(request.method=="POST"):
        m=request.POST['srh']
        match=book.objects.filter(Q(title__icontains=m)|Q(author__icontains=m))
        if(match):
            return render(request,'search.html',{'b':match})
        else:
         messages.error(request,"NO results Found")
    return render(request,'search.html')
```

We have to import Q and messages

Urls.py

```
    url(r'^search/$', views.search_book, name="search"),
```

SEARCH OPTION IN NAVIGATION BAR

BASE.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
</head>
<body>
<div class="container-fluid">
    <nav class="navbar navbar">
        <ul class="nav navbar-nav">
            <li><a href="{% url 'home' %}">HOME</a></li>
            {% if user.is_authenticated %}
            <li><a href="{% url 'upload' %}">ADD BOOKS</a></li>
            <li><a href="{% url 'list' %}">VIEW BOOKS</a></li>
            <li><a href="{% url 'logout1' %}">LOGOUT</a></li>
            <li><form method="POST" action="{% url 'search' %}">
                {% csrf_token %}
                <div class="form-group" >
                <div class="col-lg-10">
                <input type="text" name="srh" class="form-control" placeholder="Enter name">
                </div>
                    <label class="col-lg-2">
                    <button type="submit" class="btn btn-danger">Search</button>
                    </label>
                    </div>
            </form>
            </li>
            {% else %}
            <li><a href="{% url 'signup1' %}">SIGNUP</a></li>
            <li><a href="{% url 'login1' %}">LOGIN</a></li>
{% endif %}
        </ul>

    </nav>
    {% block body_block %}
    {% endblock %}
</div>
</body>
</html>
```

action be given in this way to go to another page.
Here, **action="{% url 'search' %}"**

SEARCH.HTML

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}Signup{% endblock%}
{% block body_block %}
<div class="container">
    {% if b %}
    <h2>Book Details</h2>
    <table class="table table-bordered">
        <tr>
            <th>COVER</th>
            <th>TITLE</th>
            <th>AUTHOR</th>
            <th>PDF</th>
        </tr>
        {% for i in b %}
        <tr>
            {% if i.cover %}
            <td><img src="{{i.cover.url}}" height="100px" width="100px"></td>{% else %}
            <td>NO COVER</td>{% endif %}
            <td>{{i.title}}</td>
            <td>{{i.author}}</td>
            <td><a href="{{i.pdf.url}}">DOWNLOAD</a>
        </tr>
        {% endfor %}
    </table>
    {% endif %}
    {% if messages %}
<ul class="messages">
    {% for message in messages %}
    <li>{{ message }}</li>
    {% endfor %}
</ul>
{% endif %}
</div>
{% endblock %}
```

VIEWS.PY

```
@login_required()
def search_book(request):
    if(request.method=="POST"):
        m=request.POST['srh']
        match=book.objects.filter(Q(title__icontains=m)|Q(author__icontains=m))
        if(match):
            return render(request,'search.html',{'b':match})
        else:
            messages.error(request, "NO results Found")
    return render(request,'search.html')
```

URLS.PY

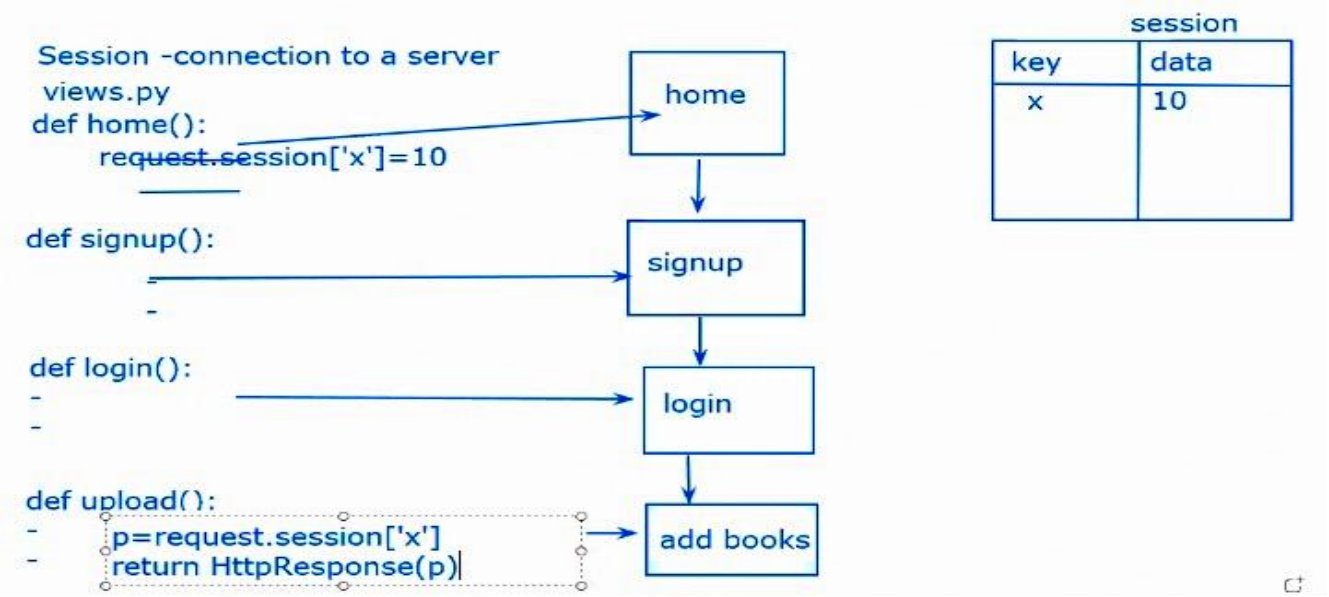
```
        url(r'^search/$', views.search_book, name="search"),
```

HOME ADD BOOKS VIEW BOOKS LOGOUT Enter name Search

SESSION

- Session means connection to a server.
- When we execute python manage.py runserver and click http://127.0.0.1:8000/ in the terminal, we start a session. We travel to different web pages as a part of each session. A session will be ended at the time we close that particular window.
- Session means connection to a server, after connection we can communicate with the server.

Usage of session



- From above figure, as an example if we declare x=10 in one function named home() in views.py, we cannot use the same variable x in another function named upload() directly. That means we cannot simply pass values between different functions in views.py.
- Therefore, to use same variable in different functions, we have to first store it in a default table in the data base named django_session. The data is stored in session table as shown above figure i.e; column named key and data. From that session table , the variable can be used inside other functions in views.py.
- The figure above demonstrate an example, i.e; The variable x=10 , which given in home() function can be used in other functions in views.py in the following ways;-
  1. The variable x assigned with 10 inside home() function in views.py is stored in the django_session table , by the following statement:-
     request.session['x']=10
  2. The variable x is used in the other function named upload() in views.py in the following way:-
     p=request.session['x']

ORDER BY

- We already studied delete, edit and view things in a particular table. But to perform order by task we have to use the concept of session.

Steps to perform order by

- The concept of order by is that when we click the order by button , the corresponding **book details** and the **details of the user** who ordered is stored in the table that we define in models.py.
- For the above purpose we create a table in models.py

Models.py

```
class order(models.Model):
    btitle = models.CharField(max_length=20)
    bauthor = models.CharField(max_length=20)
    uname=models.CharField(max_length=20)
    uphone=models.IntegerField()
    uemail=models.EmailField()
```

Views.py

```
def user_login(request):
    if(request.method=="POST"):
        u=request.POST['username']
        p=request.POST['password']
        user=authenticate(username=u,password=p)
        if user:
            login(request,user)
            request.session['name'] = user.username
            return home(request)
        else:
            return HttpResponse("Invalid Details")
    return render(request,'login1.html')
def order_book(request,p):
    b=book.objects.get(pk=p)
    n=request.session['name']
    u=CustomUser.objects.get(username=n)
    o=order.objects.create(btitle=b.title,bauthor=b.author,uname=u.username,uphone=u.phno,uemail=u.email)
    o.save()
    return order_status(request)
def order_status(request):
    return render(request,'status.html')
```

To fetch user details

To fetch book details

Status.html

```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}HOME{% endblock %}
{% block body_block %}
<div class="container">
<h2>ORDERED SUCCESSFULLY</h2>
</div>
{% endblock %}
```

List.html

Add link for order in list.html

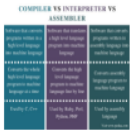
```
<!DOCTYPE html>
{% extends 'base.html' %}
{% block title %}HOME{% endblock %}
{% block body_block %}
<div class="container">
<h2>Book Details</h2>
    <table class="table table-bordered">
        <tr>
            <th>COVER</th>
            <th>TITLE</th>
            <th>AUTHOR</th>
            <th>PDF</th>
            <th>DELETE</th>
            <th>VIEW</th>
            <th>EDIT</th>
            <th>ORDER</th>
        </tr>
        {% for i in b %}
        <tr>
            {% if i.cover %}
            <td><img src="{{i.cover.url}}" height="100px" width="100px"></td>{% else %}
            <td>NO COVER</td>{% endif %}
            <td>{{i.title}}</td>
              <td>{{i.author}}</td>
              <td><a href="{{i.pdf.url}}">DOWNLOAD</a>
             <td><a href="{% url 'delete_book' i.pk %}" class="btn btn-danger">DELETE</a></td>
             <td><a href="{% url 'view_book' i.pk %}" class="btn btn-danger">VIEW</a></td>
             <td><a href="{% url 'edit_book' i.pk %}" class="btn btn-danger">EDIT</a></td>
             <td><a href="{% url 'order_book' i.pk %}" class="btn btn-danger">ORDER BOOK</a></td>
        </tr>
        {% endfor %}
    </table>
</div>

{% endblock %}
```

Urls.py

```
url(r'order_book/(?P<p>[0-9]+)', views.order_book, name="order_book"),
```

Book Details

COVER	TITLE	AUTHOR	PDF	DELETE	VIEW	EDIT	ORDER
	hello	just	DOWNLOAD	DELETE	VIEW	EDIT	ORDER BOOK