**Building a Social website**

**Introduction**

In this application, users will be allowed to share images that they find on the internet in this way they will interacting with other users. The elements of the project include the following;

An authenticating system for users to register, log in, edit their profile and change or reset their password.

A follow system that allows users to follow each other.

A functionality to display shared images and implement a bookmarklet for users to share images from any website.

An activity stream that allows users to see the content uploaded by the people they follow.

**Starting the social site**

A directory was made with the following command and navigated to it:

*mkdir Social Site*

Inside the directory a virtual environment was created by:

*python -m venv virtual*

*source virtual/Scripts/activate*

In the virtual environment Django was installed by the following command:

*pip install django*

The following command was the executed to start a new project:

*django-admin startproject bookmarks*

Navigate to the project’s directory and create a new application:

*cd bookmarks*

*django-admin startapp account*

In the *settings.py file,* the application was added in the INSTALLED\_APPS section as follows

INSTALLED\_APPS = [

    'account.apps.AccountConfig',

#### #####

]

The application is placed in the first position in the installed apps to ensure that the authentication templates are used by default instead of any other authentication templates contained in other applications. Django looks for templates by order of application appearance in the installed apps settings.

In order to sync the database the following commend is executed

*python manage.py migrate*

**Using the Django Authentication Framework.**

**Creating a Log in View**

Django authentication framework will be used to allow users to log in to the site. The view should perform the following actions in order to log in a user.

Get the username and password posted by the user.

Authenticate the user against the data stored in the database.

Check whether the user is active.

Log the user into the site and start an authentication session.

In the account application directory a *forms.py file* is created and the following code added:

from django import forms

class LoginForm(forms.Form):

    username = forms.CharField

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

This form will be used to authenticate users against the database. The *PasswordInput* widget is used to render the *password* HTML element. This will include *type = “ password ”* in the HTML.

In the views.py file the following code is added

from django.shortcuts import render

from django.http import HttpResponse

from django.contrib.auth import authenticate, login

from .forms import LoginForm

# Create your views here.

def user\_login(request):

    if request.method == 'POST':

        form = LoginForm(request.POST)

        if form.is\_valid():

            cd = forms.cleaned\_data

            user = authenticate (request,

            username = cd['username'],

            password = cd['password'])

            if user is not None:

                if user.is\_active:

                    login(request, user)

                    return HttpResponse('Authenticated' \ 'successfully')

                else:

                    return HttpResponse('Disabled account')

            else:

                return HttpResponse('invalid login')

    else:

        form = LoginForm()

    return render (request, 'account/login.html', {'form': form})

when a *user\_login* view is called with a *GET* request, a new login form is instantiated by *form = LoginForm()* to display it in the template. When the user submits the form via *POST,* the following actions are performed.

* Instantiate the form with the submitted data with *form = LoginForm(request.POST)*
* Checking whether the form is valid with *form.is\_valid().* If not valid, form errors are displayed in the template.
* If the submitted data is valid, the user is authenticated against the database using the *authenticate()* method. This method takes the *request* object, *username, and the password* parameters and returns the *User* object if the user has been successfully authenticated, if not, a raw *HttpResponse* is returned displaying *Invalid Login message.*
* If the user is successfully authenticated, the user is checked if he/she is active by *is\_active* attribute. This is an attribute of Django’s user model. If the user is not active, a *HttpResponse* is returned and displays the *Disabled account*  message.
* If the user is active, he/she is logged into the website. The user is set in session by calling the *login()* method and return the *Authenticated successfully message.*

In the accounts’ application a new *url.py file* is created and the following code added in it:

from django.urls import path

from . import views

urlpatterns = [

    path('login/', views.user\_login, name = 'login'),

]

In the main *urls.py file the* following code is then added

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('account/', include('account.urls')),

]

**Creating Templates**

In the accounts application, create the following files and folders in this order.

*templates/*

*account/*

*login.html*

*base.html*

In the *base.html file the following code is added:*

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>{% block title %} {% endblock %}</title>

    <link rel="stylesheet" href="{% static 'css/base.css' %}">

</head>

<body>

    <div id="header">

        <span class="logo">Bookmarks</span>

    </div>

    <div id="content">

        {% block content %}

        {% endblock %}

    </div>

</body>

</html>

In the *login.html file* the following code is added:

{% extends "base.html" %}

{% block title %} Log-in {% endblock %}</title>

    <link rel="stylesheet" href="{% static 'css/base.css' %}">

{% block content %}

    <h1>Log-in</h1>

    <p>Please use the following form to Log-in</p>

    <form method="post" action = "{% url 'login' %}" >

        {{ form.as\_p }}

        {% csrf\_token %}

        <p><input type="submit" value="Log in"></p>

    </form>

{% endblock %}

**Creating a superuser**

To create a superuser, the following command is executed and the fields are filled by a desirable credentials.

winpty *python manage.py createsuperuser*

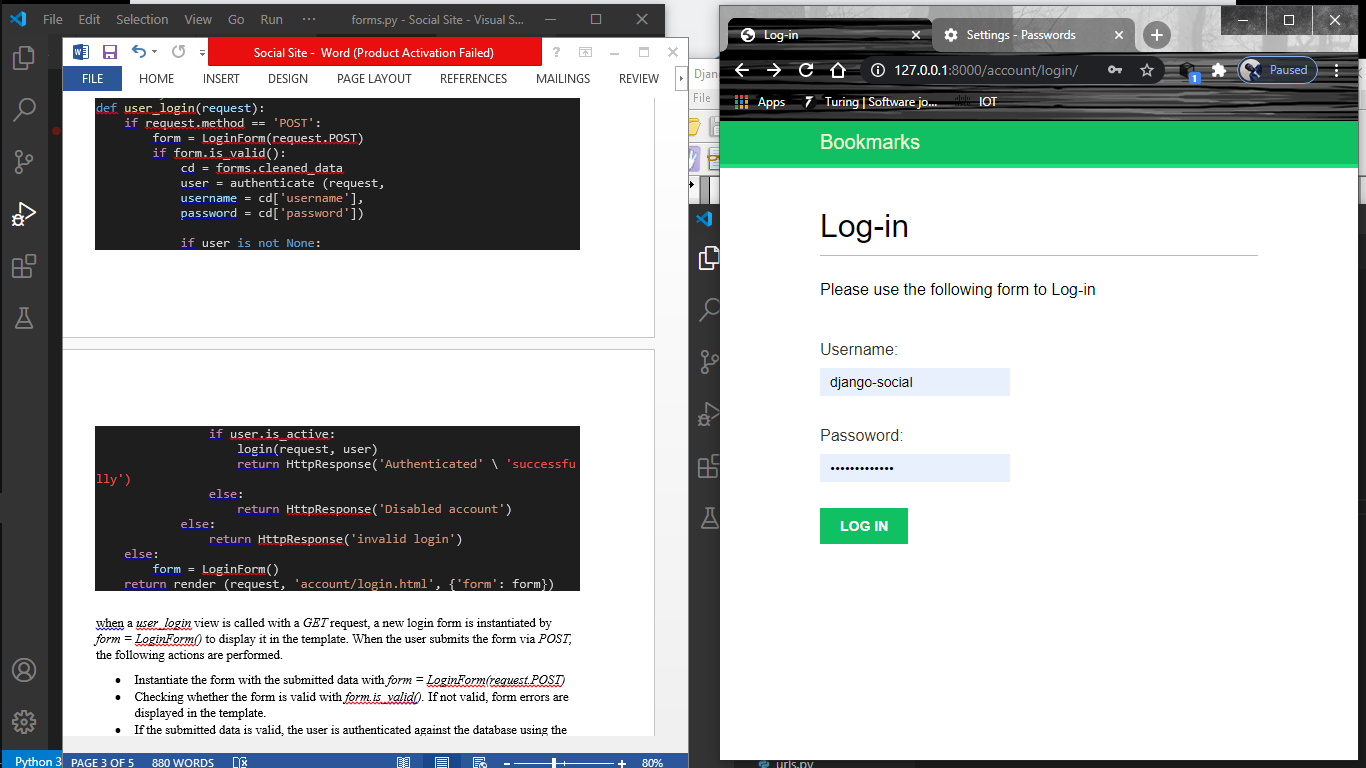


Figure 1: Log in form

**Using Django authentication views**

Django provides the following class-based views to deal with authentication.

*LoginView:* for handling the login form.

*LogoutView:* logs out a user.

Django provides the following views to handle password changes.

* *PasswordChangeView:* handling a form to change the user’s password.
* *PasswordChangeDoneView:* the success view that the user is redirected to after a successful password change.

Django also includes the following views to enable users to rest their password:

* *PasswordResetView:* allows users to resset their password. It generates a one-time-use link with a token and sends it to a user’s email.
* *PasswordRestDoneView:* tells users that an email-including a link to rest their password has been sent to them.
* *PasswordResetCompleteView:* the success view that the user is redirected to after successfully resetting their password.

**Login and logout views**

The *urls.py file is edited in the following way:*

from django.contrib.auth import views as auth\_views

urlpatterns = [

    # previous login

     # path('login/', views.user\_login, name = 'login'),

    path('login/', auth\_views.LoginView.as\_view(), name='login'),

    path('logout/', auth\_views.LogoutView.as\_view(), name='logout'),

    path('dashboard/', views.dashboard, name='dashboard'),

]

In the above code, the *user\_login* view is commented out and adding the LoginView and LogoutView url of the django’s authentication framework.

A new directory in the templates directory is created and named *registration.* This directory is the default path where the Django application expects the authentication templates to be.

In the registration directory, a new file named *login.html* is created and the following code added to it:

{% extends 'base.html' %}

{% block title %} Log-in {% endblock %}

{% block content %}

<h1>Log-in </h1>

{% if form.errors %}

    <p> your username and password didnt match. please try again.

    </p>

{% else %}

    <p>please use the following form to log in</p>

{% endif %}

<div class="login-form">

    <form action="{% url 'login' %}" method="post">

        {{ form.as\_p }}

        {% csrf\_token %}

        <input type="hidden" name="next" value="{{ next }}">

        <p><input type="submit" value="Log-in"></p>

    </form>

</div>

{% endblock %}

Django uses the *AuthenticationForm* form located at *Django.contrib.auth.forns* by default. This form tries to authenticate the user and raises a validation error if the login was unsuccessful. In this case errors are looked for using *{% if form.errors %}* in the template to check whether the credentials provided are wrong. A hidden html input element submits the value of a *next* parameter. This value is first set by the login view when you pass the *next* parameter in the request.

In the *registration* directory, a new file named *logged\_out.html* is created and the following code added to it.

{% extends 'base.html' %}

{% block title %} Logged out {% endblock %}

{% block content %}

<h1>Logged out</h1>

<p>

    you have been successfully logged out.

    you can <a href="{% url 'login' %}">log-in again</a>

</p>

{% endblock %}

This template is displayed when a user logs out.

The following code is added in the *views.py file* to create a view that is displayed when a user logs in. this is the dashboard view:

from django.contrib.auth.decorators import login\_required

@login\_required

def dashboard(request):

    return render(request,

                    'account/dashboard.html',

                    {'section': 'dashboard'})

The login view is decorated with *login\_required* decorator of the authentication framework. This decorator checks whether the current user is authenticated. If user is authenticated, it executes the decorated view, if not, the user is redirected to the login URL with the originally requested URL as a GET parameter named *next.*

By doing the above, the login view redirects the users to the URL that they were trying to access after they successfully log in. The hidden input in the form of the log in template is for this purpose.

A new template file is then created in the *account directory* and named *dashboard.html:*

{% extends 'base.html' %}  
  
{% block title %}Dashboard{% endblock %}  
{% block content %}  
  
<h1>Dashboard</h1>  
  
<p>Welcome to your Dashboard</p>  
  
{% endblock %}

A URL pattern for the above view is then added in the *urls.py file* as follows:

path('dashboard/', views.dashboard, name='dashboard'),

The following code is then added in the *settings.py file:*

LOGIN\_REDIRECT\_URL = 'dashboard'  
LOGIN\_URL = 'login'  
LOGOUT\_URL = 'logout'

*LOGIN\_REDIRECT\_URL* = tells Django which URL to redirect the user to after a successful login if no *next* parameter is present in the request.  
*LOGIN\_URL* = redirects the user to the login ( e.g. Views using the *login\_required* decorator )  
*LOGOUT\_URL* = redirects the user to logout.

Now, you will add log in and log out links to your base template to put everything together. In order to do this, you have to determine whether the current user is logged in or not in order to display the appropriate link for each case. The current user is set in the *HttpRequest* object by the authentication middleware. You can access it with *request.user.* You will find a User object in the request even if the user is not authenticated. A non-authenticated user is set in the request as an instance of *AnonymousUser*. The best way to check whether the current user is authenticated is by accessing the read-only attribute *is\_authenticated*.

The *base.html* template is modified and the following code added to it:

<div id="header">  
 {% if request.user.is\_authenticated %}  
 <ul class="menu">  
 <li {% if section == "dashboard" %}class="selected" {%endif%}>  
 <a href=" {% url 'dashboard' %}">My dashboard</a>  
 </li>  
  
 <li {% if section=="images" %} class="selected" {% endif %}>  
 <a href="#">Images</a>  
 </li>  
  
 <li {% if section == people %} class="selected" {% endif %}>  
 <a href="#">People</a>  
 </li>  
 </ul>  
 {% endif %}  
  
 <span class="user">  
 {% if request.user.is\_authenticated %}  
 Hello {{ request.user.first\_name }},  
 <a href="{% url 'logout' %}">Logout</a>  
 {% else %}  
 <a href="{% url 'login' %}">Log-in</a>  
 {% endif %}  
 </span>  
</div>

As you can see in the preceding code, you only display the site's menu to authenticated users. You also check the current section to add a selected class attribute to the corresponding <li> item in order to highlight the current section in the menu using CSS. You display the user's first name and a link to log out if the user is authenticated, or a link to log in otherwise.

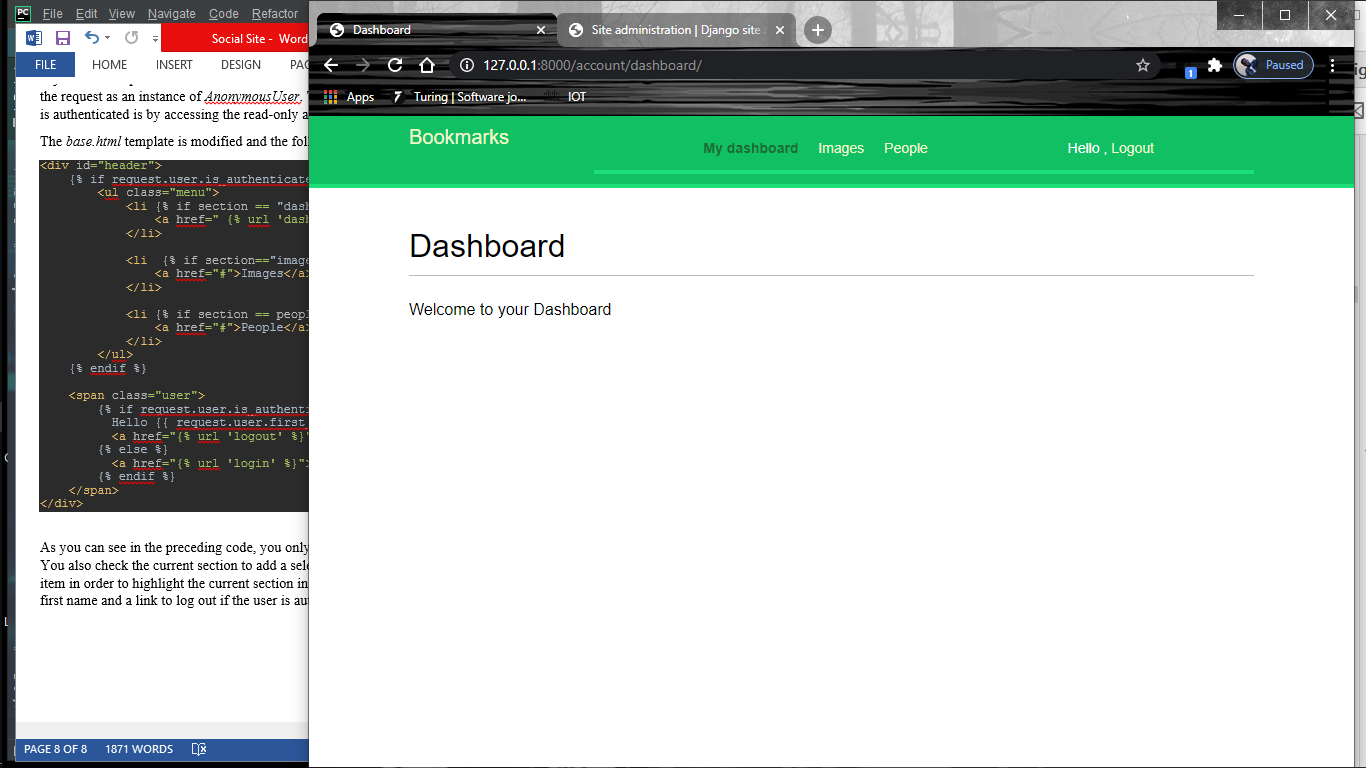


Figure 2: User redirected to the dashboard after login

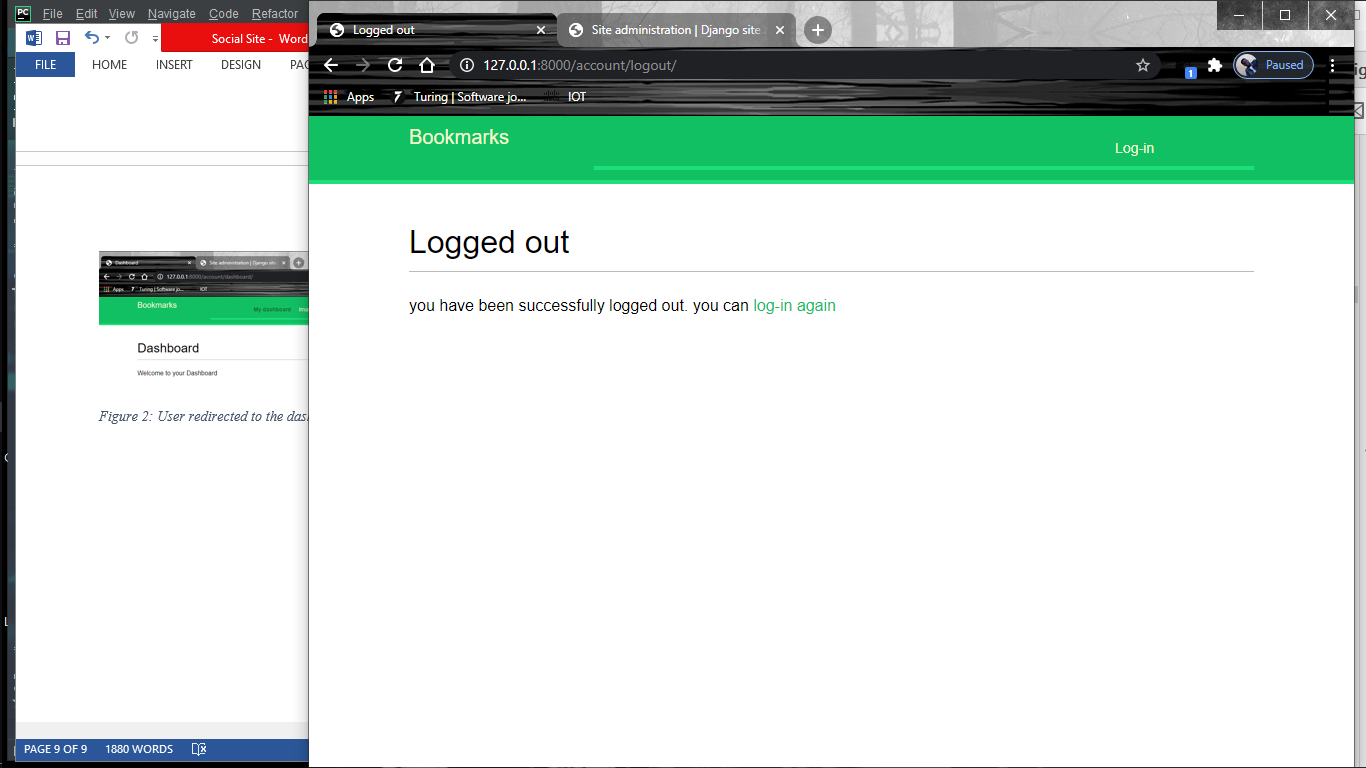


Figure 3: User logged out

**Changing password views**

The Django authentication views for a password change is used in this case. The following code is added in the app’s *urls.py file* to add the *change\_password and change\_password done* URL:

# change password urls  
 path('password\_change/', auth\_views.PasswordChangeView.as\_view(), name='password\_change'),  
 path('password\_change/done/', auth\_views.PasswordChangeDoneView.as\_view(), name='password\_change\_done'),

In the *registration directory, password\_change\_form.html* file is created and the following code added to it:

{% extends "base.html" %}  
{% block title %}Change your password{% endblock %}  
{% block content %}  
  
 <h1>Change your password</h1>  
 <p>Use the form below to change your password.</p>  
  
 <form method="post">  
 {{ form.as\_p }}  
 <p><input type="submit" value="Change"></p>  
 {% csrf\_token %}  
 </form>  
  
{% endblock %}

The above template includes a form to change the password.

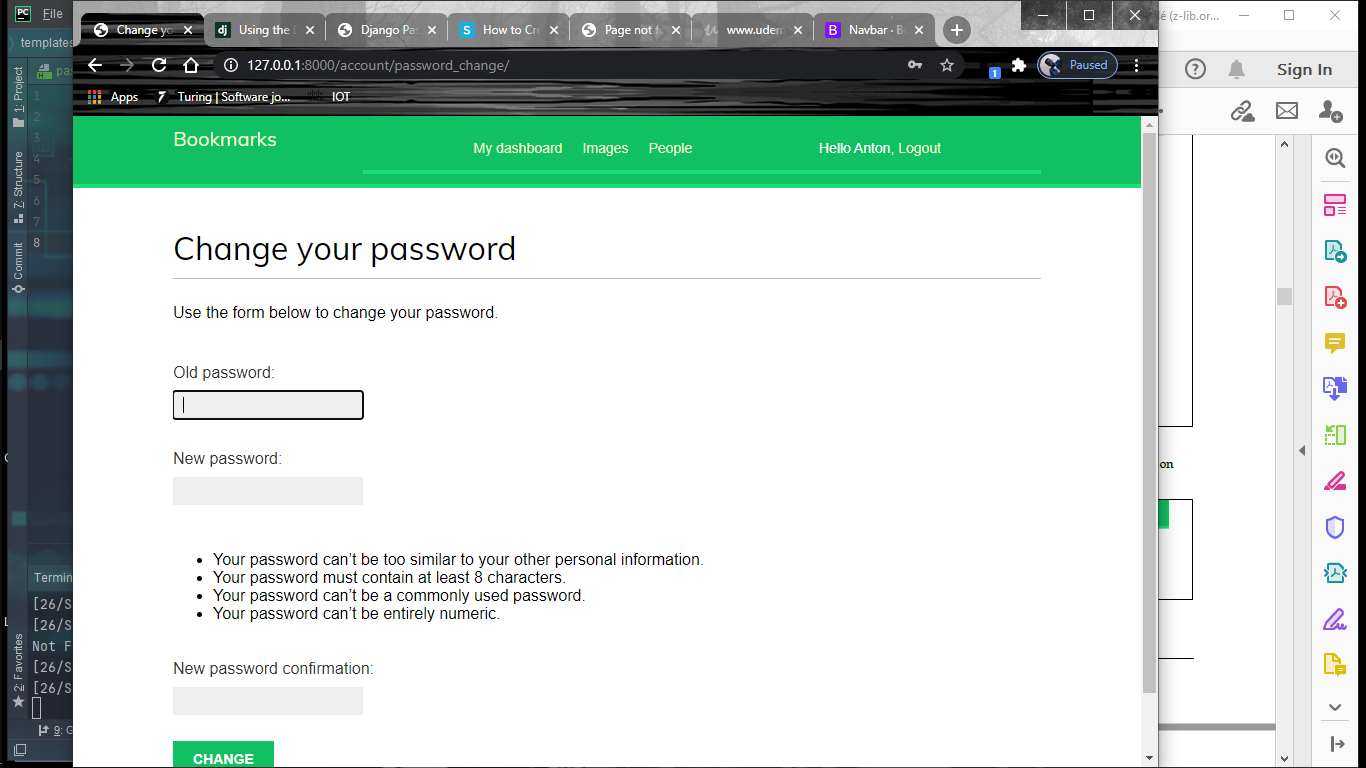


Figure 4: Password Change Form

Another file named *password\_change\_done.html* is created and the following code added to it:

{% extends "base.html" %}  
{% block title %}Password changed{% endblock %}  
{% block content %}  
  
 <h1>Password changed</h1>  
 <p>Your password has been successfully changed.</p>  
  
{% endblock %}

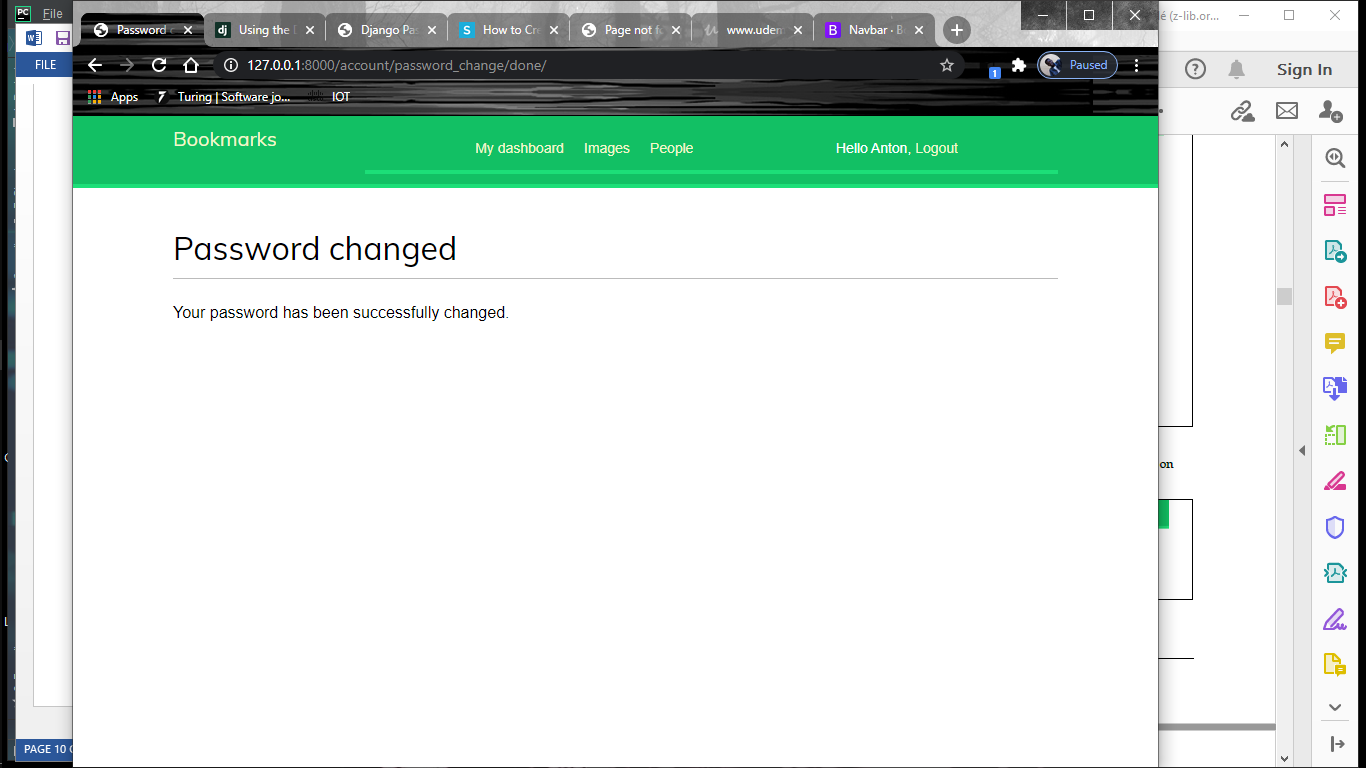


Figure 5: Password change done

**Resetting password views**

The following URL patterns are added in the *urls.py* file for password restoration:

# reset password urls  
path('password\_reset/', auth\_views.PasswordResetView.as\_view(),  
 name='password\_reset'),  
path('password\_reset/done', auth\_views.PasswordResetDoneView.as\_view(),  
 name='password\_reset\_done'),  
path('reset/<uidb64>/<token>/', auth\_views.PasswordResetConfirmView.as\_view(),  
 name='password\_reset\_confirm'),  
path('reset/done', auth\_views.PasswordResetCompleteView.as\_view(),  
 name='password\_reset\_complete')

In the *registration folder* the *password\_reset\_form.htm* template is created and the following code added to it:

{% extends "base.html" %}  
{% block title %}Reset your password{% endblock %}  
{% block content %}  
  
 <h1>Forgotten your password?</h1>  
 <p>Enter your e-mail address to obtain a new password.</p>  
  
 <form method="post">  
 {{ form.as\_p }}  
 <p><input type="submit" value="Send e-mail"></p>  
 {% csrf\_token %}  
 </form>  
  
{% endblock %}

Under the same directory, *password\_reset\_email.html* template is created and the following code added to it:

Someone asked for password reset for email {{ email }}. Follow the  
link below:  
{{ protocol }}://{{ domain }}{% url "password\_reset\_confirm"  
uidb64=uid token=token %}  
Your username, in case you've forgotten: {{ user.get\_username }}

The above template is used to render the email sent to users to reset their password. It includes a reset token that is generated by the view.

A *password\_reset\_done.html* is added under the same directory and the following code added to it:

{% extends "base.html" %}  
{% block title %}Reset your password{% endblock %}  
{% block content %}  
  
 <h1>Reset your password</h1>  
 <p>We've emailed you instructions for setting your password.</p>  
 <p>If you don't receive an email, please make sure you've entered  
 the address you registered with.</p>  
  
{% endblock %}

Another template named*password\_reset\_confirm.html* is added with the following code:

{% extends "base.html" %}  
{% block title %}Reset your password{% endblock %}  
{% block content %}  
 <h1>Reset your password</h1>  
 {% if validlink %}  
 <p>Please enter your new password twice:</p>  
 <form method="post">  
 {{ form.as\_p }}  
 {% csrf\_token %}  
 <p><input type="submit" value="Change my password" /></p>  
 </form>  
 {% else %}  
 <p>The password reset link was invalid, possibly because it has  
 already been used. Please request a new password reset.</p>  
 {% endif %}  
{% endblock %}

In the above template, the reset link is checked whether it is valid by checking the *validlink* variable. The *PasswordResetConfirmView* checks the validity of the token provided in the url and passes the *validlink* variable to the template

Still in the same directory, a *password\_reset\_complete.html* template is created and the following code added to it:

{% extends "base.html" %}  
{% block title %}Password reset{% endblock %}  
{% block content %}  
  
 <h1>Password set</h1>  
 <p>Your password has been set. You can  
 <a href="{% url 'login' %}">log in now</a></p>  
  
{% endblock %}

In the *registration/login.html* the following code is added after the *<form>* element.

<p><a href="{% url 'password\_reset' %}">Forgotten your password?</a></p>

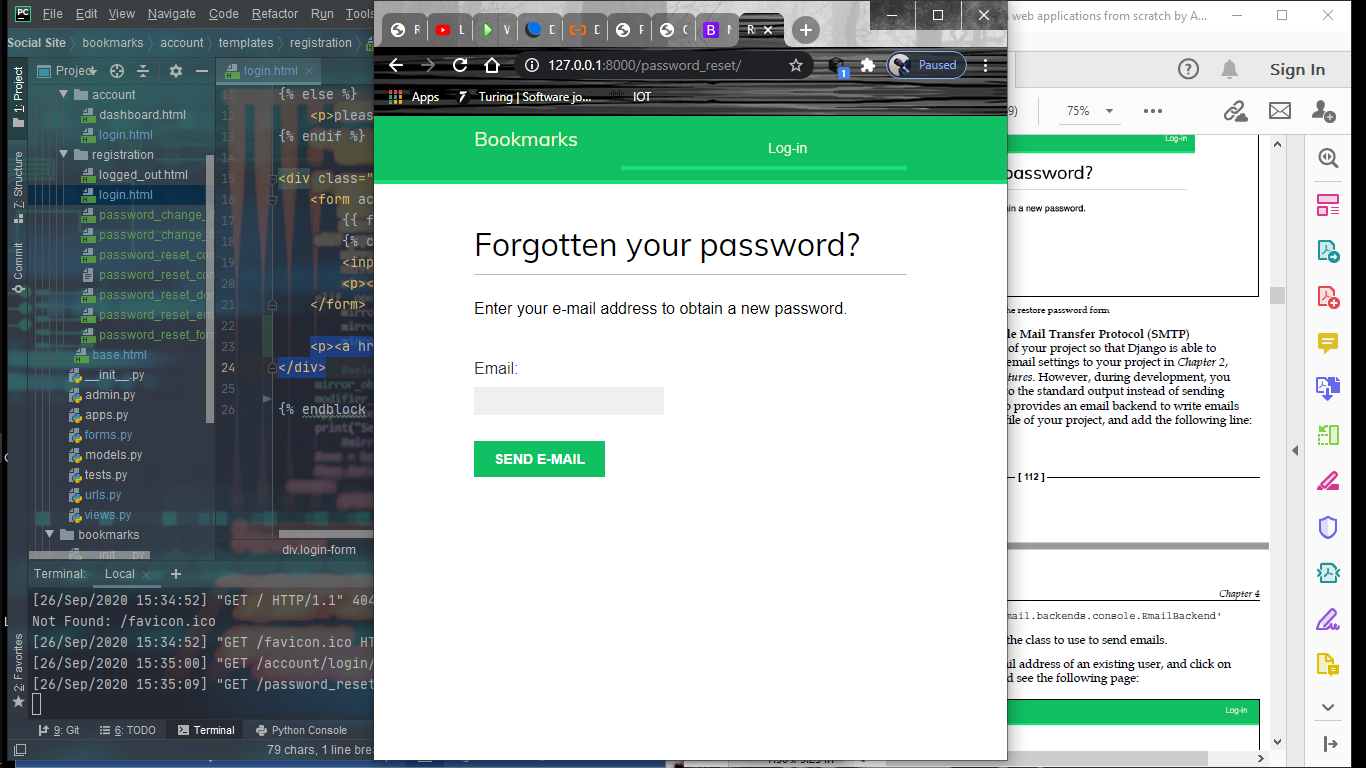


Figure 6: Password Reset Form

For Django to send an email, SMTP configuration is added to the *settings.py* as follows in order for it to write an email to the standard output to the console:

EMAIL\_BACKEND = 'django.core.mail.backends.console.EmailBackend'

When the email of an existing user is filled and the SEND MAIL button clicked, the following page is produced:

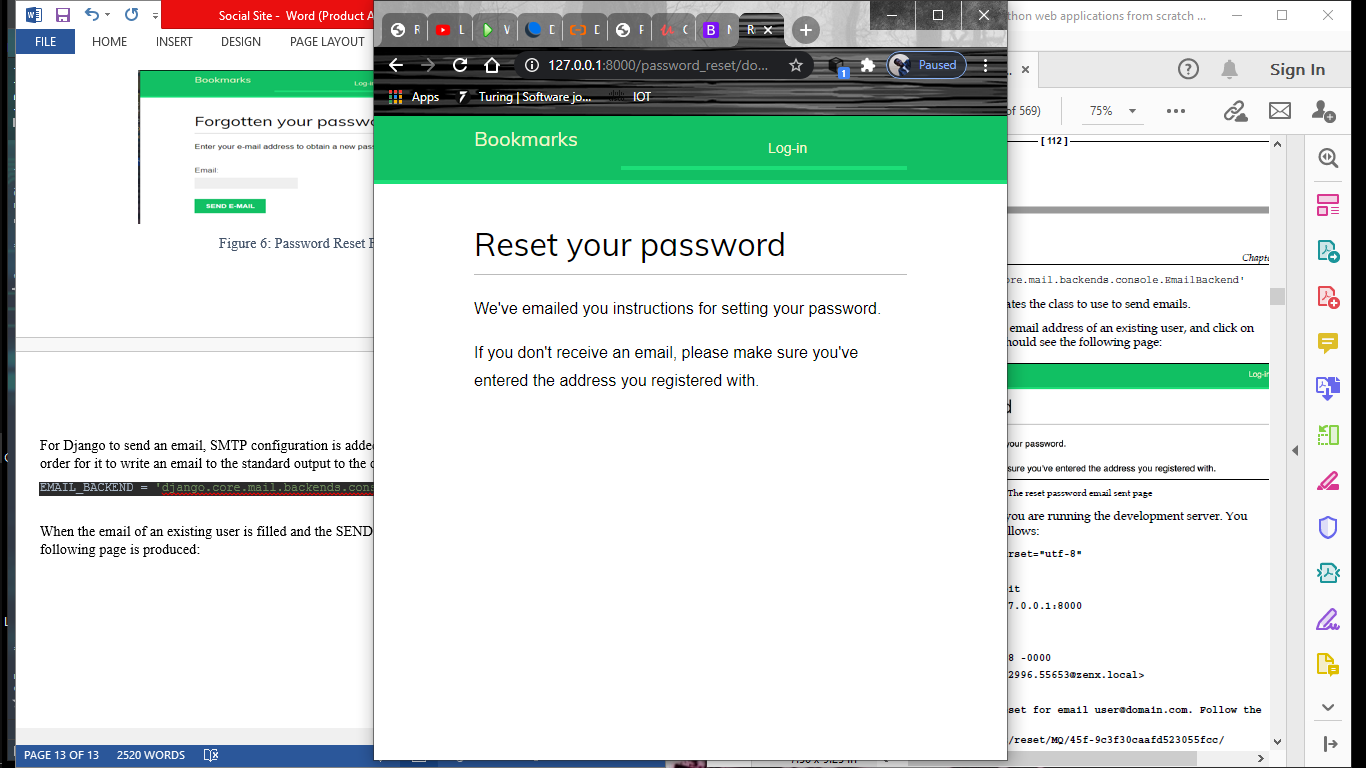


Figure 7: Password Reset email sent page

In the console where the server is running, the following output is produced

Content-Type: text/plain; charset="utf-8"

MIME-Version: 1.0

Content-Transfer-Encoding: 7bit

Subject: Password reset on 127.0.0.1:8000

From: webmaster@localhost

To: jumaantony@outlook.com

Date: Sat, 26 Sep 2020 23:08:13 -0000

Message-ID: <160116169345.16420.1052638202664218794@WICKED>

Someone asked for password reset for email jumaantony@outlook.com. Follow the

link below:

http://127.0.0.1:8000 /reset/NA/aate5p-13ed8005653cebb3edfae443ff4a8868/

Your username, in case you've forgotten: jumaantony

The above email is rendered using the *password\_reset\_email.html* template. The URL for password reset contains a token that was generated dynamically by Django.

When the URL is opened in the browser, the following screenshot is produced:

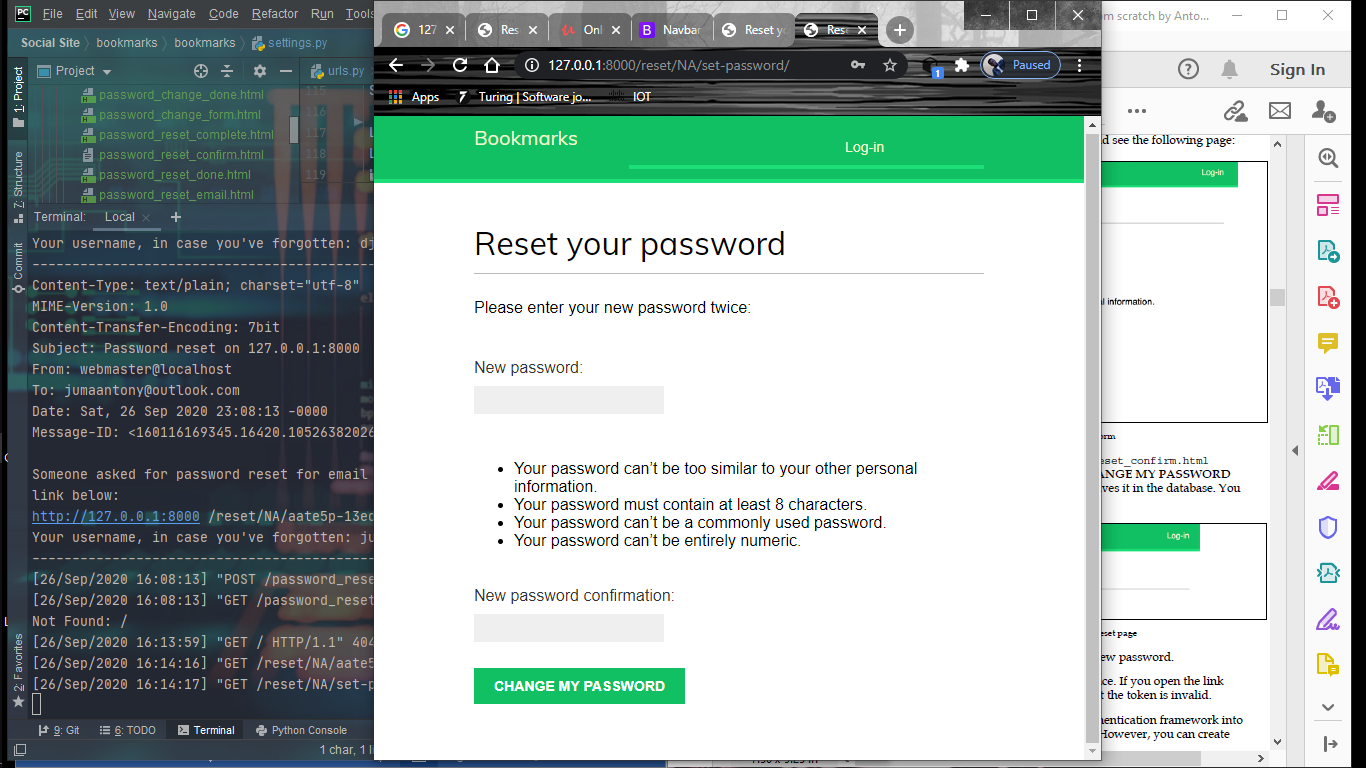


Figure 8: Password Reset Form

The above screenshot uses the *password\_reset\_confirm.html* template. When the above form has been filled and submitted, Django will create a new hashed password and save it in the database. The following success message is shown:

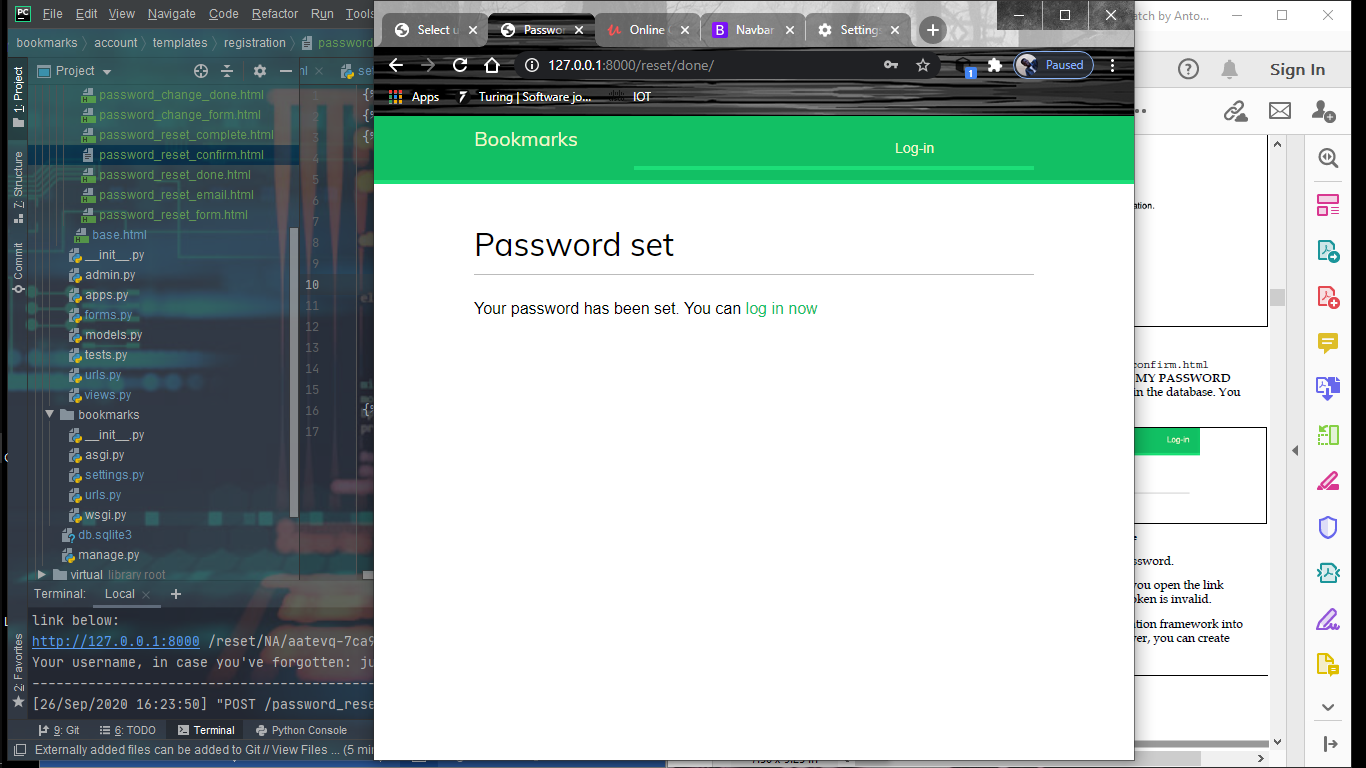


Figure 9: Successful Password Reset

Each token to set a new password can only be used once.

Django also provides an authentication URL patterns for the patterns that we’ve just created and this pattern can be added in the account *urls.py* as

# Django authentication URL  
path('', include('django.contrib.auth.urls')),