**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">

        {{ 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*

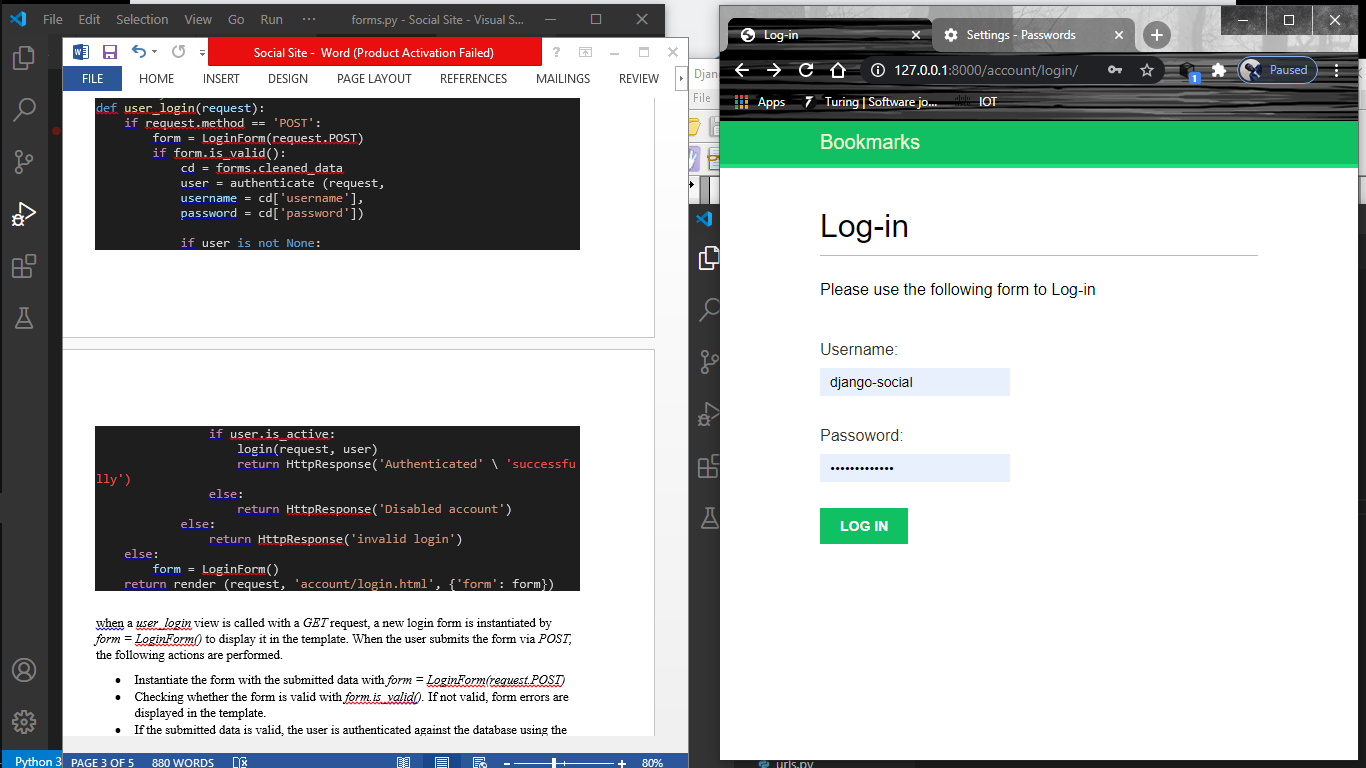


Figure 1: Log in form