Overview

Friday, March 21, 2025 12:24 PM

This project is creating a social website project that uses login, logout, password change, password recovery views. It will also allow users to share images found on the internet.

I will create a dashboard view that users will have access to when they successfully authenticate

I will implement a user registration with the register view

I will extend the user model with a custom profile model and create the edit view to allow users to edit their profile

Goals:

- understand how to build social capabilities into a site

- Implement advanced functionalities with django and js

Elements:
- Authentication system for user to register, login, edit their profile, and change/reset password
- Social authentication to sign in with services such as google
- Functionality to display shared images and a system for users to share images from any website
- An activity stream that allows user to see the content uploaded by the people that they follow
- A follow system to allow users to follow each other

HTML Templates

Saturday, March 22, 2025 12:30 PM

Base:

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
  <link href="{% static 'css/bse.css' %}" rel="stylesheet">
</head>
<body>
  <div id="header">
    <span class="logo">Bookmarks</span>
  </div>
  <div id="content">
    {% block content %}
    {% endblock %}
  </div>
</body>
</html>
```

Login:

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

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

{% 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 %}
```

Initial Setup

Friday, March 21, 2025 12:28 PM

Requirements:
asgiref==3.7.2
Django~=5.0.4
sqlparse==0.5.0
Pillow==10.3.0

1. Setup venv
    a. Using pycharm it is done for me
2. Install django
    a. Create requirements.txt
        i. Pip install -r requirements.txt
3. Start project
    a. django-admin startproject bookmarks
    b. Cd bookmarks
    c. Django-admin startapp account
4. Add app to settings.py INSTALLED_APPS
    a. Place this one before everything else in order to ensure that it will be used rather than the ones used in .admin

```
INSTALLED_APPS = [
    'account.apps.AccountConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

5. Migrate
    a. Python manage.py migrate

Djangos Authentication Framework

Friday, March 21, 2025 12:40 PM

Django has a built in authentication framework that can handle user auth, sessions, permissions, and user groups.

Has views for logging in, out, pw change, and pw reset

Located in django.contrib.auth

The auth framework is included when running startproject.

Includes:

- Django.contrib.auth
- MIDDLEWARE:
  - AuthenticationMiddleware
    - Associates users with requests using sessions
  - SessionMiddleware
    - Handles the current sessions across requests
- User
  - A user model with basic fields
  - Main fields are username, password, email, first_name, last_name, is_active
- Group:
  - A group model to catergorize users
- Permission:
  - Flag for users or groups to perform certain actions
- Includes default authentication views and forms

Middleware: Classes with methods that are globally executed during the request or response phase

- Will use middleware classes on several occasions
- Will learn how to create them in chapter 17

Creating the login view

Friday, March 21, 2025 12:49 PM

This is a way to create a login view yourself, Django has its own login view (see Django Login View Page)

Create a forms.py and add

```
from django import forms

class LoginForm(forms.Form):
    username = forms.CharField()
    password = forms.CharField(widget=forms.PasswordInput)
```

Login View:

- Presents user with a login form
- Gets the user and password when they submit the form
- Authenticates the user against the data stored in the database
- Checks whether the user is active
- Log the user into the website and starts an authenticated session

```
from django.contrib.auth import authenticate, login
from django.http import HttpResponse
from django.shortcuts import render
from .forms import LoginForm

def user_login(request):
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            cd = form.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:
        form = LoginForm()
    return render(request, 'account/login.html', {'form': form})
```

1. User_login view is called with a GET request
2. New login form is instantiated with a form = LoginForm()
3. Submits Form
    a. POST

b. Instantiated with data submitted
    i. Form = LoginForm(request.POST)
c. Form is validated
    i. form.is_valid()
d. If valid: user gets authenticated against the database
    i. Authenticate() method
        1) Takes request object, username, password and returns the User object if authentication is successful otherwise returns None
    ii. If authenticate() fails
        1) Raw HttpResponse with invalid login message returned
    iii. If authenticate() succeeds
        1) User status is checked by accessing is_active.
            a) Is_active is an attribute of djangos user model
                i) If not active return HttpResponse with disabled account message
                ii) If active user is logged in by calling the login() method and auth successful message returned

Edit urls.py

Add urls.py to app

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

urlpatterns = [
    path('login/', views.user_login, name='login'),
]
```

Add to main urls.py

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

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

Create Login Template

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

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

{% 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 %}
```

We include csrf_token because the form will be submitted by POST

- Csrf tokens prevent against csrf attacks
    - Prevents unauthorized commands being transmitted
    - Generates a token and checks that it matches the user

Django login views

Django has its own class based views to deal with authentication

```
from django.contrib.auth import views as auth_views
from django.urls import path
from . import views

urlpatterns = [
    # old login path
    #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'),
]
```

Create Login.html templates/registration

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

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

{% block content %}
  <h1>Log-in</h1>
  {% if form.errors %}
    <p>
      Your username and password didn't 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 %}
```

We have created a login template and a logout template using django's authenticationForm in django.contrib.auth.forms

- {% if form.errors %} checks if credentials are wrong

Create logged_out.html templates/registration

```
{% 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 %}
```

- Displayed after logout

Inside view.py add
```
from django.contrib.auth.decorators import login_required

@login_required
def dashboard(request):
    return render(
        request,
        'account/dashboard.html',
        {'section': 'dashboard'}
    )
```

- Created the dashboard view
- Applied login_required decorater
    - Checks whether the user is authenticated
- If authenticated executes the decorated view
- If not authenticated directs to login url as a GET parameter named next
- By doing this the login view redirects to the URL that they were trying to access after the successfully logged in.
    - We added a hidden <input> named next in login template for this purpose

Add to account/urls.py
```
path('', views.dashboard, name='dashboard'),
```

Add login to setting.py
```
LOGIN_REDIRECT_URL = 'dashboard'
```
- Tells django which url to redirect user to after login
```
LOGIN_URL = 'login'
```
- Redirect the user to login
```
LOGOUT_URL = 'logout'
```
- The url to redirect the user to log out

Add a link to login URL and button to log out to the base template.

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <title>{% block title %}{% endblock %}</title>
  <link href="{% static 'css/base.css' %}" rel="stylesheet">
</head>
<body>
```

```
  <div id="header">
    <span class="logo">Bookmarks</span>
      {% 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|default:request.user.username }},
            <form action="{% url 'logout' %}" method="post">
                <button type="submit">Logout</button>
                {% csrf_token %}
            </form>
          {% else %}
            <a href="{% url 'login' %}">Log-in</a>
          {% endif %}
      </span>
  </div>
  <div id="content">
    {% block content %}
    {% endblock %}
  </div>
</body>
</html>
```

Change Password Views

Tuesday, March 25, 2025 2:30 PM

Users must be able to change their password after logging into the website

We will use django authentication views to change passwords

```
path(
    'password-change',
    auth_views.PasswordChangeView.as_view(),
    name='password_change'
),
path(
    'password-change/done/',
    auth_views.PasswordChangeDoneView.as_view(),
    name='password_change_done'
),
```

The PasswordChangeView will handle the form to change the password

PasswordChangeDoneView will display the success message after the user has changed their password

password_change_form.html

```
{% 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 %}
```

password_change_done.html

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

{% block title %}Password Changed{% endblock %}

{% block content %}
  <h1>Password Changed</h1>
  <p>Your password has been successfully changed.</p>
{% endblock %}
```

Reset Password Functionality

Inside account urls.py

```
# 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(
    'password-reset/<uidb64>/<token>/',
    auth_views.PasswordResetConfirmView.as_view(),
    name='password_reset_confirm'
),
path(
    'password=reset/complete/',
    auth_views.PasswordResetCompleteView.as_view(),
    name='password_reset_complete'
),
```

HTML

Password_reset_form.html

```
{% 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 %}
```

Password_reset_email.html

```
Someone asked for a password reset for email {{ email }} Follow the link below:
{{ protocol }}://{{ domain }}{% url "password_reset_confirm" uidb64=uid token=token %}
```

Password_reset_done.html

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

{% block title %}Reset your passsword{% 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 %}
```

Password_reset_confirm.html

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

{% block title %}Reset your passsword{% endblock %}

{% block content %}
  {% 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, possible because it has already
    been used. Please request a new password reset.</p>
  {% endif %}
{% endblock %}
```

Password_reset_complete.html

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

{% block title %}Reset your passsword{% 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 %}
```

Edit the registration/login.html template

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

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

{% block content %}
  <h1>Log-in</h1>
  {% if form.errors %}
    <p>
      Your username and password didn't 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>
    <p>
      <a href="{% url 'password_reset' %}">
        Forgotten your password?
      </a>
    </p>
  </div>
{% endblock %}
```

After this point we should add a smtp configuration in the setting like we did for the blog app
if
However django has a way to write emails to the console for testing

Write Emails To Console

Wednesday, March 26, 2025 11:22 AM

Settings.py
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend

Everything we just added can be complemented out and added as

from django.urls import include, path

path('', include('django.contrib.auth.urls')),

- This is telling django to add all of the default routes
- If you want to use something different however, you can uncomment and use the different path instead.

So comment it all out and just add this before dashboard…

User Registration

Tuesday, March 25, 2025 5:19 PM

Time to create a user registration for the website.

forms.py

```
from django import forms
from django.contrib.auth import get_user_model

class LoginForm(forms.Form):
    username = forms.CharField()
    password = forms.CharField(widget=forms.PasswordInput)

class UserRegistrationForm(forms.ModelForm):
    password = forms.CharField(
        label='Password',
        widget=forms.PasswordInput
    )
    password2 = forms.CharField(
        label='Repeat password',
        widget=forms.PasswordInput
    )

    class Meta:
        model = get_user_model()
        fields = ['username',
                  'first_name',
                  'email']
```

Created a model form for the user model

- Includes
    - Username
    - First name
    - Email
- Retrieves user model dynamically using the get_user_model() function
    - Retrieves the user model
    - Validated according to the validations of corresponding model fields
        - If user chooses a username that already exists they will get a validation error
- Added two additional fields
    - Password
    - Password2
        - Allows users to set and repeat a password

Now to add a field validation to check that both passwords are the same

Forms.py

Add after Meta inside UserRegistrationForm class

```
    def clean_password2(self):
        cd = self.cleaned_data
        if cd['password'] != cd['password2']:
            raise forms.ValidationError("Passwords don't match.")
        return cd['password2']
```

The clean_password2() method compares the second pw with the first and raise an error if they don't match

- Cd assigns the dictionary containing all data that has been validated and cleaned
- Compares password and password 2 fields
- If not same
    - Validation error
- Else
    - Return value of password2
        - Value is used as the cleaned data for the password2 field

Cleaned Data:

- A dictionary that contains the form data after it has been validated and processed through the cleaning methods.
- Safe to use because it has already passed all validations
    - For example if a user inputs a number into a text field django expects and int. Converts the input to an int in the cleaning process
- Ensures that you are working with validated and properly formatted data
    - Reduces risk of errors or security issues later in the application

Account views.py

Add UserRegistrationForm and register function

```
from .forms import LoginForm, UserRegistrationForm

def register(request):
    if request.method == 'POST':
        user_form = UserRegistrationForm(request.POST)
        if user_form.is_valid():
            # Create a new user object but avoid saving it yet
            new_user = user_form.save(commit=False)
            # Set the chosen password
            new_user.set_password(
                user_form.cleaned_data['password']
            )
            # Save the User object
            new_user.save()
            return render(
                request,
                'account/register_done.html',
                {'new_user': new_user}
            )
    else:
        user_form = UserRegistrationForm()
    return render(
        request,
        'account/register.html',
        {'user_form': user_form}
    )
```

- Instead of saving the raw password user entered
    - Use set_password() method of the user model
        - Handles password hashing before storing the password in the database
    - Django doesn't store clear text passwords

        - Stores hashed passwords instead
    -

Hashing	Process of transforming a given key to another value. A hash function is used to generate a fixed length value according to a mathematical algorithm.

        - By hashing passwords with secure algorithms django ensures that user passwords stored in the db require massive amounts of computing time to break
    - Django uses PBKDF2 hasing algorithm with a SHA256 hash to store al lpasswords
        - Django supports checking existing passwords hashed with PBKDF2
        - Supports checking stored passwords hashed with other algorithms too
            - PBKDF2SHA1
            - Argon2
            - Bcrypt
            - scrypt

Inside account/urls.py

```
path('', include('django.contrib.auth.urls')),
path('', views.dashboard, name='dashboard'),
path('register/', views.register, name='register'),
```

Add a new template in templates/account/

register.html

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

{% block title %}Create an account{% endblock %}

{% block content %}
  <h1>Create an account</h1>
  <p>Please, sign up using the following form:</p>
  <form method="post">
    {{ user.form.as_p }}
    {% csrf_token %}
    <p><input type="submit" value="Create my account"></p>
  </form>
{% endblock %}
```

register_done.html

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

{% block title %}Welcome{% endblock %}

{% block content %}
 <h1>Welcome {{ new_user.first_name }}!</h1>
 <p>
  Your account has been successfully created.
  Now you can <a href="{% url 'login' %}">log in</a>
 </p>
{% endblock %}
```

Extending the User Model

Wednesday, March 26, 2025 11:22 AM

Models.py

```
from django.db import models
from django.conf import settings


# Create your models here.
class Profile(models.Model):
    user = models.OneToOneField(
        settings.Auth_User_model,
        on_delete=models.CASCADE
    )
    date_of_birth = models.DateField(blank=True, null=True)
    photo = models.ImageField(
        upload_to='users/%Y/%m/%d/',
        blank=True
    )

    def __str__(self):
        return f'Profile of {self.user.username}'
```

Django provides a user model in the authentication framework and its great for most cases.

However if you want additional details you can extend the default user model

The way to do this is to create a profile model that contains one to one relationship with the django user model.

- **user = models.OneToOneField()**
    - **settings.AUTH_USER_MODEL**
    - **on_delete=models.CASCADE**

- **setting.AUTH_USER_MODEL**
    - Creates a one to one relationship
        - Each instance of model is linked to exactly one user
            - Used for creating a user profile model that extrends the built in user mode
    - **Settings.AUTH_USER_MODEL**
        - Instead of hardcoding a regerence to Djangos default user model (django.contrib.auth.models.User) use the setting to refer to the user model
        - Makes code more flexible
            - If you define a custom user model by updating AUTH_USER_MODEL this relationship will automatically refer to the correct model
    - **on_delete=models.CASCADE**
        - When a user is deleted the instance of the model will also be deleted
            - Cascading delete
        - Helps maintain data integrity
        - Ensures don't end up with a profile or related record that no longer has an associated user

Serving Media Files

Wednesday, March 26, 2025 11:36 AM

Python -m pip install Pillow==10.3.0
- Already installed if requirements.txt installed

Pillow Library:
- Standard library for image processing in Python
- Supports multiple image formats and provides powerful image processing functions
- **Required by Django to handle images with ImageField**

Settings.py

```
MEDIA_URL = 'media/'
MEDIA_ROOT = BASE_DIR / 'media'
```

Bookmarks/urls.py
```
from django.conf import settings
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import include, path

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

if settings.DEBUG:
    urlpatterns += static(
        settings.MEDIA_URL,
        document_root=settings.MEDIA_ROOT
    )
```

Added the static() helper function to serve media files with the Django dev server during development
- (when DEBUG=True)

- Serves static files which is not good for production
    - Django inefficient at serving static files

Migrate the profile model
- Python manage.py makemigrations
- Python manage.py migrate

Account/admin.py
```
from django.contrib import admin
from .models import Profile

# Register your models here.
@admin.register(Profile)
class ProfileAdmin(admin.ModelAdmin):
    list_display = ['user','date_of_birth', 'photo']
    raw_id_fields = ['user']
```

Forms.py

```
from .models import Profile


class UserEditForm(forms.ModelForm):
    class Meta:
        model = get_user_model()
        fields = ['first_name', 'last_name', 'email']

class ProfileEditForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = ['date_of_birth', 'photo']
```

Views.py

```
from .models import Profile

def register(request):
    if request.method == 'POST':
        user_form = UserRegistrationForm(request.POST)
        if user_form.is_valid():
            # Create a new user object but avoid saving it yet
            new_user = user_form.save(commit=False)
            # Set the chosen password
            new_user.set_password(
                user_form.cleaned_data['password']
            )
            # Save the User object
            new_user.save()
            # Create the user profile
            Profile.objects.create(user=new_user)
            return render(
                request,
                'account/register_done.html',
                {'new_user': new_user}
            )
    else:
        user_form = UserRegistrationForm()
    return render(
        request,
        'account/register.html',
        {'user_form': user_form}

    )
```

After registering a Profile object will be created with the associated user object created

- Users created through the admin site wont automatically get a profile object
- Users with and without a profile (staff) can co-exist

Allowing users to edit their profiles

Wednesday, March 26, 2025 11:57 AM

Account/views.py

```
from .forms import (
    LoginForm,
    UserRegistrationForm,
    UserEditForm,
    ProfileEditForm
)


@login_required

def edit(request):
    if request.method == 'POST':
        user_form = UserEditForm(
            instance=request.user,
            data=request.POST
        )
        profile_form = ProfileEditForm(
            instance=request.user.profile,
            data=request.POST,
            files=request.FILES
        )
        if user_form.is_valid() and profile_form.is_valid():
            user_form.save()
            profile_form.save()
        else:
            user_form = UserEditForm(instance=request.user)
            profile_form = ProfileEditForm(instance=request.user.profile)
        return render(
            request,
            'account/edit.html',
            {
                'user_form': user_form,
                'profile_form': profile_form
            }
        )
```

Added a new edit view
- Allows users to edit their personal info
- Login_requried decorator
  - Only want authenticated users to be able to edit their profiles
- Use two model forms
  - UserEditForm
    - Stores data of the build in user model
  - ProfileEditForm
    - Store additional personal data in the custom Profile model
- To validate data submitted use the is_valid() method
  - If contain valid data
    - Save both forms by calling save()
    - Update objects in the database

Account/urls.py

Urlpatterns add

- path('edit/', views.edit, name='edit'),

Create a template for the view templates/account/

Edit.html

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

{% block title %}Edit your account{% endblock %}

{% block content %}
  <h1>Edit your account</h1>
  <p>You can edit your account using the following form:</p>
  <form method="post" enctype="multipart/form-data">
    {{ user_form.as_p }}
    {{ profile_form.as_p }}
    {% csrf_token %}
    <p><input type="submit" value="Save Changes"></p>
  </form>
{% endblock %}
```

enctype="multipart/form_data"

- Enables file uploads