Blood\_bank\_app

**Introduction**

This project is a web application developed using the Django framework, designed to manage blood bank operations efficiently. It allows users to add and manage donor information, ensuring a streamlined process for blood donation management.

### Project Structure

The project consists of the following main components:

#### 1. Database

* **db.sqlite3**: This is the SQLite database file where all the information related to donors and blood bank operations is stored. SQLite is a lightweight, disk-based database that doesn't require a separate server process, making it ideal for small to medium-sized applications.

#### 2. Main Project Files

* **manage.py**: This is a command-line utility that lets you interact with this Django project in various ways. You can use it to run the development server, execute database migrations, create application-specific commands, and more.
* #!/usr/bin/env python
* """Django's command-line utility for administrative tasks."""
* import os
* import sys
* def main():
* """Run administrative tasks."""
* os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'blood\_bank.settings')
* try:
* from django.core.management import execute\_from\_command\_line
* except ImportError as exc:
* raise ImportError(
* "Couldn't import Django. Are you sure it's installed and "
* "available on your PYTHONPATH environment variable? Did you "
* "forget to activate a virtual environment?"
* ) from exc
* execute\_from\_command\_line(sys.argv)
* if \_\_name\_\_ == '\_\_main\_\_':
* main()
* **blood\_bank/**: This directory contains the core settings and configurations for the Django project.
  + **\_\_init\_\_.py**: This file makes the blood\_bank directory a Python package.
  + **asgi.py**: This file contains the ASGI configuration for the project, necessary for handling asynchronous requests.
  + **settings.py**: This file contains all the configurations such as database settings, installed apps, middleware, and other project-specific settings.
* """
* Django settings for blood\_bank project.
* Generated by 'django-admin startproject' using Django 5.0.7.
* For more information on this file, see
* https://docs.djangoproject.com/en/5.0/topics/settings/
* For the full list of settings and their values, see
* https://docs.djangoproject.com/en/5.0/ref/settings/
* """
* from pathlib import Path
* import os
* # Build paths inside the project like this: BASE\_DIR / 'subdir'.
* BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent
* # SECURITY WARNING: keep the secret key used in production secret!
* SECRET\_KEY = 'django-insecure-a(se@$6uk3u1ws!b$igt\*3=k4(#8q9t)xe)44=^q=k%%fq7p@o'
* # SECURITY WARNING: don't run with debug turned on in production!
* DEBUG = True
* ALLOWED\_HOSTS = []
* # Application definition
* INSTALLED\_APPS = [
* 'django.contrib.admin',
* 'django.contrib.auth',
* 'django.contrib.contenttypes',
* 'django.contrib.sessions',
* 'django.contrib.messages',
* 'django.contrib.staticfiles',
* 'blood\_bank\_app',  # Replace with your app name
* ]
* MIDDLEWARE = [
* 'django.middleware.security.SecurityMiddleware',
* 'django.contrib.sessions.middleware.SessionMiddleware',
* 'django.middleware.common.CommonMiddleware',
* 'django.middleware.csrf.CsrfViewMiddleware',
* 'django.contrib.auth.middleware.AuthenticationMiddleware',
* 'django.contrib.messages.middleware.MessageMiddleware',
* 'django.middleware.clickjacking.XFrameOptionsMiddleware',
* ]
* ROOT\_URLCONF = 'blood\_bank.urls'
* # Static files (CSS, JavaScript, Images)
* # https://docs.djangoproject.com/en/5.0/howto/static-files/
* STATIC\_URL = '/static/'
* # Add the static files directory path
* STATICFILES\_DIRS = [
* os.path.join(BASE\_DIR, 'static'),
* ]
* TEMPLATES = [
* {
* 'BACKEND': 'django.template.backends.django.DjangoTemplates',
* 'DIRS': [],
* 'APP\_DIRS': True,
* 'OPTIONS': {
* 'context\_processors': [
* 'django.template.context\_processors.debug',
* 'django.template.context\_processors.request',
* 'django.contrib.auth.context\_processors.auth',
* 'django.contrib.messages.context\_processors.messages',
* ],
* },
* },
* ]
* WSGI\_APPLICATION = 'blood\_bank.wsgi.application'
* # Database
* # https://docs.djangoproject.com/en/5.0/ref/settings/#databases
* DATABASES = {
* 'default': {
* 'ENGINE': 'django.db.backends.sqlite3',
* 'NAME': BASE\_DIR / 'db.sqlite3',
* }
* }
* # Password validation
* # https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators
* AUTH\_PASSWORD\_VALIDATORS = [
* {
* 'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',
* },
* {
* 'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',
* },
* {
* 'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',
* },
* {
* 'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',
* },
* ]
* # Internationalization
* # https://docs.djangoproject.com/en/5.0/topics/i18n/
* LANGUAGE\_CODE = 'en-us'
* TIME\_ZONE = 'UTC'
* USE\_I18N = True
* USE\_L10N = True
* USE\_TZ = True
* # Default primary keey field type
* # https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field
* DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'
  + **urls.py**: This file maps URLs to the respective views, enabling the routing of HTTP requests to appropriate parts of the application.
* # blood\_bank/urls.py
* from django.contrib import admin
* from django.urls import include, path
* urlpatterns = [
* path('admin/', admin.site.urls),
* path('', include('blood\_bank\_app.urls')),
* ]
  + **wsgi.py**: This file contains the WSGI configuration for the project, necessary for deploying the application on a web server.

#### 3. Application Files (blood\_bank\_app/)

* **\_\_init\_\_.py**: This file makes the blood\_bank\_app directory a Python package.
* **admin.py**: This file configures the Django admin interface to manage the application’s data through a web-based interface.
* **apps.py**: This file contains the configuration for the blood\_bank\_app application.
* **models.py**: This file defines the data models, representing the structure of the data in the database. For example, it might define a Donor model with fields such as name, blood\_type, contact, etc.
* from django.db import models
* class Donor(models.Model):
* name = models.CharField(max\_length=100)
* blood\_group = models.CharField(max\_length=10)
* def \_\_str\_\_(self):
* return self.name
* **views.py**: This file contains the logic for handling HTTP requests and returning responses. For instance, it might include views for adding a new donor, listing all donors, and so on.
* from django.shortcuts import render, redirect
* from django.contrib.auth import authenticate, login
* from django.contrib.auth.decorators import login\_required
* from .models import Donor
* def login\_view(request):
* if request.method == 'POST':
* username = request.POST['username']
* password = request.POST['password']
* user = authenticate(request, username=username, password=password)
* if user is not None:
* login(request, user)
* return redirect('home')
* else:
* return render(request, 'blood\_bank\_app/login.html', {'error': 'Invalid credentials'})
* return render(request, 'blood\_bank\_app/login.html')
* @login\_required
* def home(request):
* return render(request, 'blood\_bank\_app/home.html')
* @login\_required
* def add\_donor(request):
* if request.method == 'POST':
* name = request.POST['name']
* blood\_group = request.POST['blood\_group']
* Donor.objects.create(name=name, blood\_group=blood\_group)
* return redirect('donor\_list')
* return render(request, 'blood\_bank\_app/add\_donor.html')
* @login\_required
* def donor\_list(request):
* donors = Donor.objects.all()
* return render(request, 'blood\_bank\_app/donor\_list.html', {'donors': donors})
* **urls.py**: This file defines URL patterns specific to this application, mapping URLs to the corresponding views.
* from django.urls import path
* from . import views
* urlpatterns = [
* path('', views.login\_view, name='login'),
* path('home/', views.home, name='home'),
* path('add\_donor/', views.add\_donor, name='add\_donor'),
* path('donor\_list/', views.donor\_list, name='donor\_list'),
* ]
* **tests.py**: This file contains tests for the application, ensuring that all parts of the application work as expected.
* **migrations/**: This directory contains migration files that handle changes to the database schema over time. For example, adding a new field to the Donor model would require a new migration file.
  + **0001\_initial.py**: Initial migration file for setting up the database schema.
  + **0002\_alter\_donor\_blood\_group.py**: Migration file for altering the blood\_group field in the Donor model.
  + **0003\_rename\_blood\_group\_donor\_blood\_type\_donor\_contact.py**: Migration file for renaming and altering fields in the Donor model.

#### 4. Static Files

* **static/**: This directory includes static files such as CSS and images used across the application.
  + **css/styles.css**: CSS file for styling the application.
* body {
* background: url("static\blood\_bank\_app\images\istockphoto-1320162065-612x612.jpg") no-repeat center center fixed;
* background-size: cover;
* font-family: Arial, sans-serif;
* }
* h2 {
* color: white;
* }
* form, ul {
* background: rgba(255, 255, 255, 0.8);
* padding: 20px;
* border-radius: 10px;
* margin: 20px;
* }
  + **images/istockphoto-1320162065-612x612.jpg**: Image file used in the application.

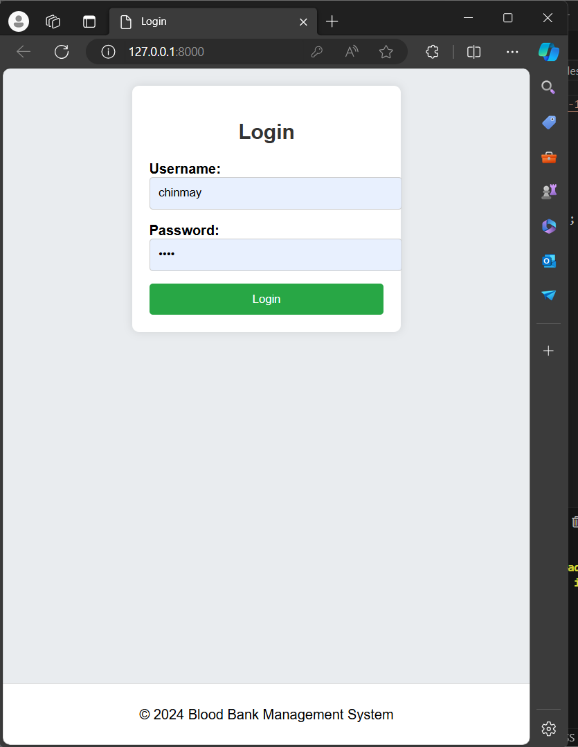
#### 5. Templates

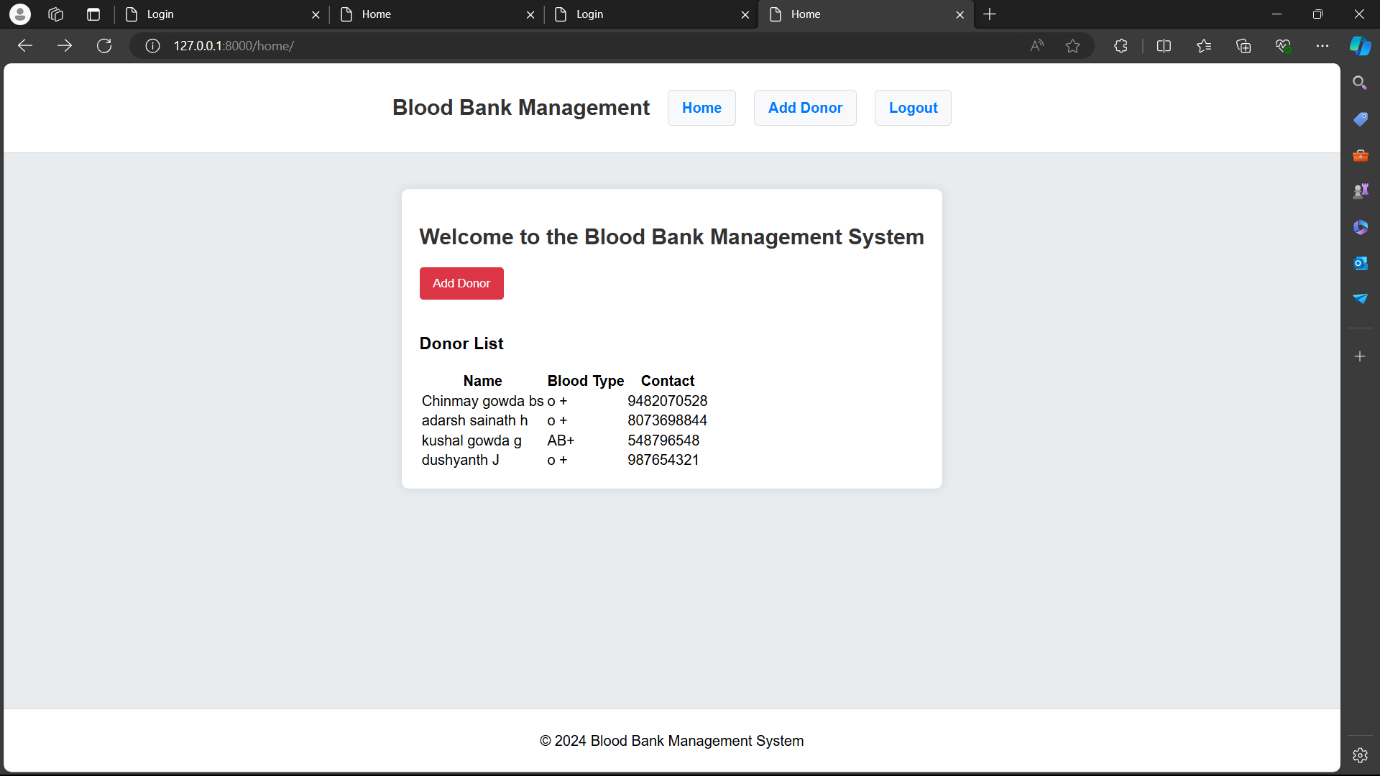
* **templates/**: This directory contains HTML templates for the application.
  + **add\_donor.html**: Template for adding a donor.
* <!DOCTYPE html>
* <html>
* <head>
* <title>Add Donor</title>
* {% load static %}
* <link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">
* </head>
* <body>
* <h2>Add Donor</h2>
* <form method="post">
* {% csrf\_token %}
* <label for="name">Name:</label>
* <input type="text" id="name" name="name"><br>
* <label for="blood\_group">Blood Group:</label>
* <input type="text" id="blood\_group" name="blood\_group"><br>
* <button type="submit">Add Donor</button>
* </form>
* </body>
* </html>
  + **donor\_list.html**: Template for displaying a list of donors.
* <!DOCTYPE html>
* <html>
* <head>
* <title>Donor List</title>
* {% load static %}
* <link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">
* </head>
* <body>
* <h2>Donor List</h2>
* <ul>
* {% for donor in donors %}
* <li>{{ donor.name }} - {{ donor.blood\_group }}</li>
* {% endfor %}
* </ul>
* </body>
* </html>
  + **home.html**: Home page template.
* <!DOCTYPE html>
* <html>
* <head>
* <title>Home</title>
* {% load static %}
* <link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">
* </head>
* <body>
* <h2>Home</h2>
* <a href="{% url 'add\_donor' %}">Add a new donor</a><br>
* <a href="{% url 'donor\_list' %}">View donor list</a>
* </body>
* </html>
  + **login.html**: Login page template.
* <!DOCTYPE html>
* <html>
* <head>
* <title>Login</title>
* {% load static %}
* <link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">
* </head>
* <body>
* <h2>Login</h2>
* <form method="post">
* {% csrf\_token %}
* <label for="username">Username:</label>
* <input type="text" id="username" name="username"><br>
* <label for="password">Password:</label>
* <input type="password" id="password" name="password"><br>
* <button type="submit">Login</button>
* </form>
* {% if error %}
* <p>{{ error }}</p>
* {% endif %}
* </body>
* </html>

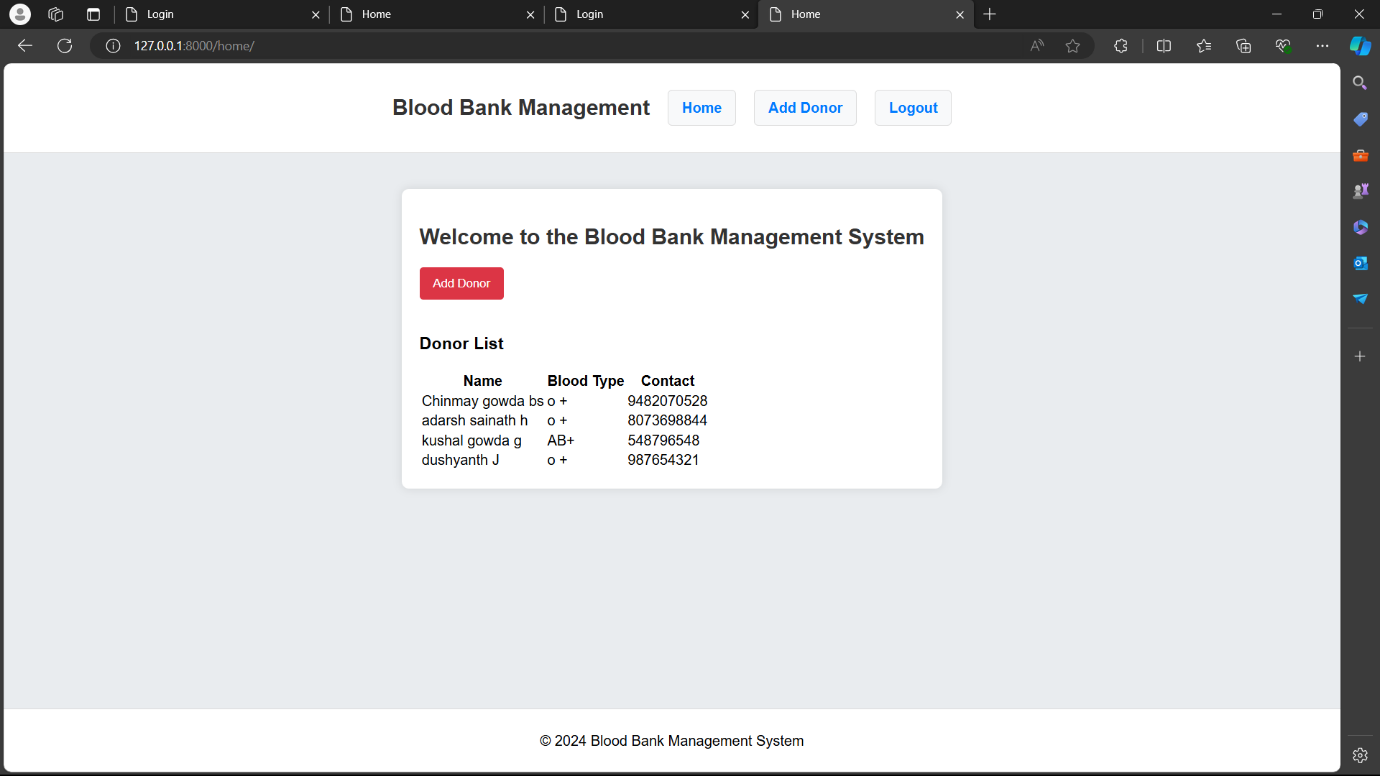
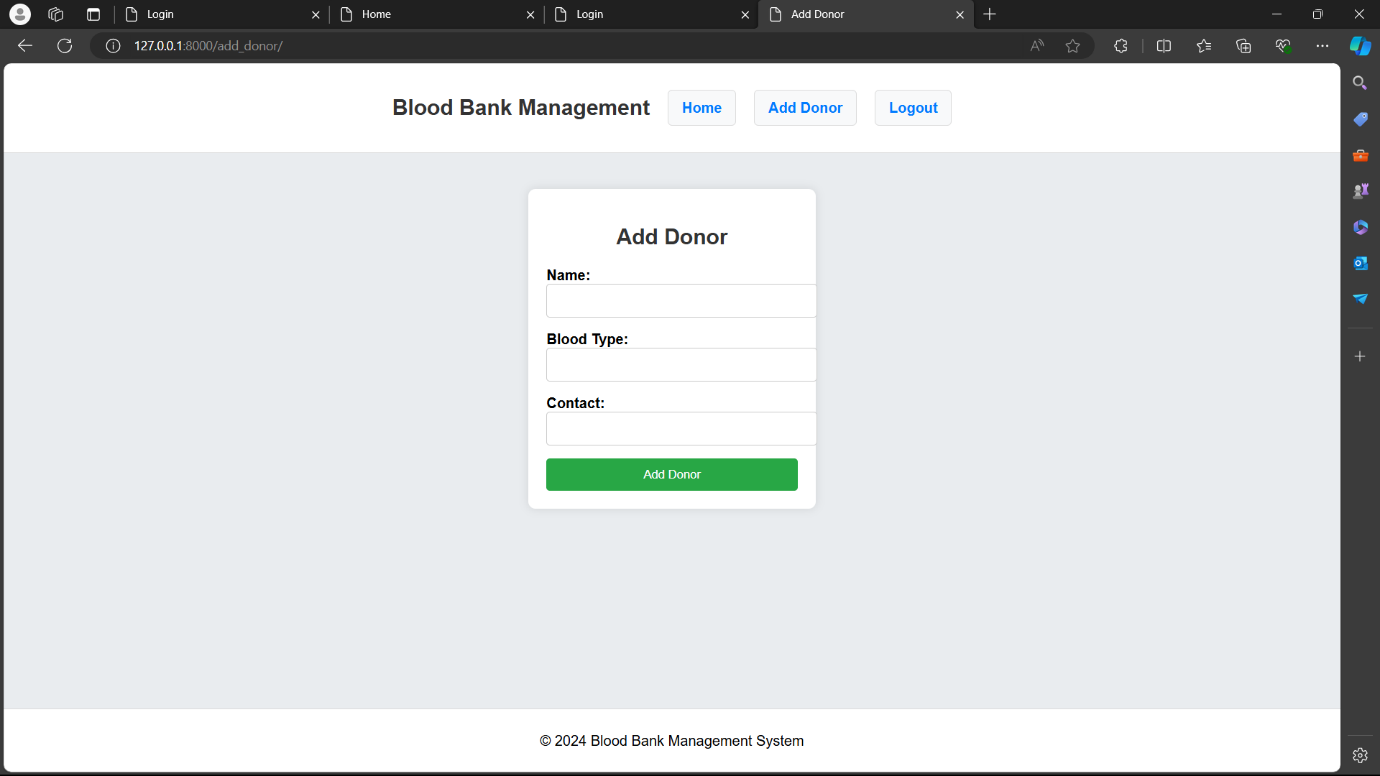
**Features**

* **Donor Management**:
  + Users can add, edit, and delete donor information. This is facilitated by the Donor model in models.py and the corresponding views and templates.
  + **add\_donor.html** template provides a form for adding donor details.
  + **donor\_list.html** template displays a list of all registered donors.
* **User Authentication**:
  + Secure login system for users to access the application, managed by Django’s built-in authentication framework.
  + **login.html** template provides a login form for users.
* **Responsive Design**:
  + The user interface is designed with CSS to be responsive and user-friendly. The **styles.css** file contains the necessary styles for the application.
* **Admin Interface**:
  + Django’s built-in admin interface allows for easy management of the application’s data. This is configured in **admin.py**.

**Snapshots**



****

****

**Conclusion**

This Django-based blood bank application provides an efficient and scalable solution for managing blood donation processes. By leveraging Django’s powerful features, the project ensures a robust and maintainable codebase, facilitating further enhancements and integrations as needed.