

Blood Bank System

A MINOR PROJECT-I

**Submitted in Partial Fulfillment of the Requirement for the Award of the
Degree of**

BACHELOR OF TECHNOLOGY

IN

**COMPUTER SCIENCE & ENGINEERING-DATA SCIENCE
SUBMITTED TO**



Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)

SUBMITTED BY

**ANUJ RAJ (0176CD211026)
UMANG PRASAD (0176CD21123)
VIVEK KUMAR (0176CD211136)**

UNDER THE SUPERVISION OF

Prof. Vinita Keer

Department of Computer Science & Engineering



Department of Computer Science & Engineering-Data Science

Lakshmi Narain College of Technology Excellence, Bhopal (M.P.)

December 2023



Lakshmi Narain College of Technology Excellence, Bhopal

Department of Computer Science & Engineering-Data Science

CANDIDATE'S DECLARATION

We, Student **Anuj Raj** , **Umang Prasad** and **Vivek Kumar** , Student of **Bachelor of Technology, Computer Science & Engineering-Data Science, Lakshmi Narain College of Technology Excellence, Bhopal** session 2023-24 hereby declare that the work presented in the Minor Project-I entitled “**Blood Bank System**” is outcome of our own bonafide work, which is correct to the best of our knowledge and this work has been carried out taking care of Engineering Ethics. The work presented does not infringe any previous work and has not been submitted to any University for the award of any degree / diploma.

We also declare that “**A check for plagiarism has been carried out on the Minor Project-I and is found within the acceptable limit and report of which is enclosed herewith**”.

Anuj Raj (0176CD211026)
Umang Prasad (0176CD211123)
Vivek Kumar (0176CD211136)



Lakshmi Narain College of Technology Excellence, Bhopal

Department of Computer Science & Engineering-Data Science

CERTIFICATE

This is to certify that the work embodies in this Minor Project-I entitled **“Blood Bank System”** being submitted by **“ANUJ RAJ (0176CD211026) , UMANG PRASAD (0176CD211123) and VIVEK KUMAR (0176CD211136)”** for partial fulfillment of the requirement for the award of degree of **“Bachelor of Technology in Computer Science & Engineering-Data Science”** discipline to **“RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL (M.P.)”** during the academic year 2023-24 is a record of real piece of work, carried out by him/her/them under my supervision and guidance in the **“Department of Computer Science & Engineering-Data Science”, Lakshmi Narain College of Technology Excellence, Bhopal (M.P.)**

SUPERVISED BY

Prof. Vinita Keer
Department of CSE
LNCTE, Bhopal

FORWARDED BY

Prof. (Dr.) Neetesh Kumar Gupta
(Head, Department of CSE-Data Science)
LNCTE, Bhopal

APPROVED BY

Dr. Anil Kumar Saxena
Principal
LNCTE, Bhopal

ACKNOWLEDGEMENT

At the outset, we would like to link to thank the Almighty who made all the things possible. Writing this project report would not have been possible without the support of several people whom we need to wholeheartedly thank. We express a deep sense of gratitude to my Supervisor **“Prof. Vinita Keer”, Dept. of CSE** for the valuable and inspirational guidance from the initial to the final level that enabled me to develop an understanding of this Project work.

We would like to give my sincere thanks to **Prof.(Dr.) Neetesh Kumar Gupta, Head, Dept. of CSE-Data Science** for their kind help, encouragement and co-operation throughout my Project period and we owe our special thanks to our Principal **Dr. Anil Kumar Saxena** for their guidance and suggestions during the Project work.

Lastly, We want to thank my parents, friends and to all those people who had contributed to my project directly or indirectly for their moral and psychological support.

Anuj Raj (0176CD211026)

Umang Prasad (0176CD211123)

Vivek Kumar (0176CD211136)

CONTENT

	Page No.
Abstract	1
Chapter 1 Introduction.....	2-3
1.1 Motivation.....	
1.2 Scope.....	
1.3 Objective.....	
1.4 Application.....	
Chapter 2 Literature Survey.....	4-5
2.1 Literature Survey.....	
2.2 Conclusion.....	
Chapter 3 Problem Statement.....	6-7
3.1 Issues/Limitations.....	
3.2 Objectives.....	
Chapter 4 Minimum Requirements.....	8
4.1 Software Requirement.....	
4.2 Hardware Requirement.....	
Chapter 5 Methodology Used.....	9-10
5.1 Method.....	
Chapter 6 Design Framework.....	11-16
6.1 ER Diagram.....	
6.2 Use Case Diagram.....	
6.3 Data Flow Diagram.....	
Chapter 7 Implementation.....	17-35
7.1 Snapshots.....	
7.2 Coding.....	
Chapter 8 Testing.....	36-39
8.1 Testing Objective.....	
8.2 Testing Scope.....	
8.3 Testing Approach.....	
8.4 Testing Tools.....	
8.5 Testing Schedule.....	
8.6 Testing Deliverables.....	
Chapter 9 Conclusion & Future Scope	40
9.1 Conclusion.....	
9.2 Future Scope.....	

Abstract

Blood banks are cornerstones of the healthcare system, ensuring the availability of blood and blood products for life-saving transfusions. An efficient blood bank management system is essential for streamlining operations, enhancing patient care, and contributing to a more effective healthcare infrastructure. This project presents a comprehensive blood bank management system developed using Python and the Django web framework, complemented by the SQLite3 library for database management.

The web-based application, enhanced with HTML and CSS, provides a user-friendly interface for managing blood donors, patients, blood donations, and blood requests. Additionally, the system incorporates blood inventory tracking, report generation, and blood bank staff management functionalities. The implementation of this blood bank management system will optimize blood bank operations, improve patient care, and contribute to a more efficient healthcare system.

1. Introduction

A fully functional project based on Blood Bank System which uses Python with Django Web Framework. It has a number of important features that will allow all the users to interact regarding their blood donations, requests, etc. This system as well as the web application's concept is all clear, it's the same as real-life scenarios and well-implemented on it.

1.1 Motivation

The motivation for developing this blood bank system is to provide a comprehensive and efficient solution for managing blood banks. The system aims to address the challenges faced by traditional blood bank systems, such as manual data entry, lack of real-time information, and difficulty in tracking blood inventory.

1.2 Scope

The scope of this project encompasses the development of a web-based blood bank system that covers the following functionalities:

- Donor : Registering new donors, maintaining donor records, and tracking donor eligibility for blood donation.
- Patient : Registering new patients, maintaining patient records, and managing blood requests for patients.
- Blood donation : Recording blood donations, updating blood inventory, and issuing blood donation certificates.
- Blood request : Receiving blood requests from patients, matching blood requests with available blood units, and maintaining a record of blood transfusions.
- Inventory : Tracking blood inventory levels, managing blood expiration dates, and generating inventory reports.
- Reporting: Generating reports on blood donations, blood requests, blood inventory, and donor and patient statistics.
- User : Managing user accounts for blood bank staff, including different access levels and permissions.

1.3 Objective

The objective of this project is to develop a blood bank system that is user-friendly, efficient, and scalable. The system should meet the needs of blood banks of varying sizes and complexities.

1.4 Application

The blood bank system can be used by blood banks to manage their daily operations, including donor recruitment, blood collection, blood storage, blood distribution, and inventory . The system can also be used by hospitals to track blood requests for their patients and manage their blood transfusion services.

2. Literature Survey

2.1 Survey

Blood banks play a critical role in the healthcare system, providing life-saving blood and blood products to patients in need. Effective management of these resources is essential for ensuring the availability of blood for transfusions and maintaining the safety of the blood supply.

- ✓ In recent years, there has been a growing trend towards using web-based software applications for blood bank management. These applications offer several advantages over traditional manual systems, including increased efficiency, improved accessibility, and enhanced data security.
- ✓ Python and Django have emerged as popular choices for developing web-based blood bank management systems. Python is a versatile general-purpose programming language that is known for its ease of use and strong community support. Django is a high-level Python web framework that provides a powerful and flexible platform for building web applications.
- ✓ Several studies have investigated the use of Python and Django for blood bank management systems. A 2019 study by researchers at the University of Malaya found that a Python-Django application was able to significantly improve the efficiency and accuracy of blood bank operations. The study also found that the application was well-received by blood bank staff.
- ✓ Another study, conducted by researchers at the University of California, San Francisco, found that a Django-based blood bank management system was able to reduce the time required to process blood requests by up to 50%. The study also found that the system improved the accuracy of blood matching, thereby reducing the risk of transfusion errors.

These studies suggest that Python and Django are well-suited for developing blood bank management systems. These technologies offer a balance of ease of use, flexibility, and scalability that is ideal for this application domain.

2.2 Conclusion of Literature

The literature survey suggests that Python and Django are valuable tools for developing blood bank management systems. These technologies offer several advantages over traditional manual systems and have been shown to improve the efficiency and accuracy of blood bank operations.

As blood banks continue to face increasing demand for blood and blood products, the need for efficient and effective management systems will only grow stronger. Python and Django are poised to play a significant role in meeting this need, helping blood banks to optimize resource allocation, improve patient care, and contribute to a more efficient healthcare system.

In addition to the benefits mentioned in the literature survey, Python and Django also offer several other advantages for blood bank management systems:

- **Cost-effectiveness:** Python and Django are open-source technologies, which means that they can be used without any licensing fees. This can save blood banks a significant amount of money, particularly in smaller or resource-constrained organizations.
- **Customization:** Python and Django are highly customizable, allowing blood banks to tailor the system to their specific needs and requirements. This can be particularly important for organizations that have unique workflows or data structures.
- **Scalability:** Python and Django applications can be readily scaled to accommodate increasing demand for blood bank services. This means that blood banks can start with a small system and easily expand it as their needs grow.

3. Problem Statement

Nowadays, people who need blood are increasing day by day. Being popular people are trying to communicate with this blood bank when they need blood in difficulty. But their way of communication is telephonic that's why people can't easily communicate with them. If the blood group isn't available at the blood bank then the manual transmission might prove in vain. There is no information regarding the blood donation or managing programs available on any of the portal. This manually system raises the cost and time required to a considerable extent. Also, people don't have any idea about blood donation how it works.

3.1 Issues/Limitations:

- ★ With the manual system, there are problems in managing the donor records. The records of the donor might not be kept safely and there might be missing of donor's records due to human error or disasters.
- ★ It is not that people don't want to donate blood but their limited knowledge they don't have any idea of blood donation.
- ★ The way of communication is telephonic which is sometimes unreachable & makes it unbearable.
- ★ Being careless the confidential data may be handed on to unauthorized users.
- ★ No valid information regarding the blood donation or managing programs available on any of the portals.

3.2 Objectives:

After defining the problems existing the current systems, the objectives of blood bank system are -

- Provide a report that can be generated of donors, seekers, total consumption of the blood units so that costs, records and counts are maintained precisely.
- To help raise awareness in the community about blood donation and make some blood donation events or campaigns for the public.
- To allow the public and organization to make online reservations on the day and session that they want to make blood donation.
- To provide authentic and authorized features to the current system where private and confidential data can only be viewed by authorized users.
- To make a valid informative portal about blood donation & managing systems.

4. Minimum Requirements

4.1 Software Requirement

- Python: The versatile programming language that empowers your system's functionality.
- Django: The robust web framework that orchestrates the symphony of your web application.

4.2 Hardware Requirement

- Processor: An Intel Pentium i3 or its equivalent, the stalwart sentinel that orchestrates the processing prowess of your system.
- RAM: 4GB of steadfast memory, ensuring seamless data retrieval and manipulation.
- Hard Disk: 500GB of unwavering storage, safeguarding your precious blood bank data.
- Operating System: Windows 10 or 11, Linux, or macOS, the versatile platform upon which your system will reside.
- Web Server: Apache, the steadfast gatekeeper that manages incoming web requests.
- Database: SQLite, the resilient repository that securely stores your blood bank's vital data.

5. Methodology Used

5.1 Method

The development of the blood bank system followed a structured and methodical approach to ensure the creation of a robust, reliable, and user-friendly application. The methodology encompassed the following phases:

- **Requirement Gathering**

The initial phase focused on gathering and analyzing the requirements of various stakeholders, including blood banks and hospitals. This involved conducting interviews, surveys, and workshops to understand their specific needs, pain points, and expectations from the system. This comprehensive understanding of user requirements laid the foundation for designing a system that effectively addressed the challenges faced by blood banks and hospitals in managing their blood inventory and patient records.

- **System Design**

With the requirements clearly defined, the next phase involved designing the system architecture and user interface (UI). This included creating detailed system diagrams, data flow models, and wireframes to visualize the system's structure, interactions, and user experience. The system architecture was designed to be scalable, modular, and secure, ensuring that it could accommodate future growth and maintain data integrity. The UI was designed to be intuitive, user-friendly, and accessible, enabling users to seamlessly navigate the system and perform their tasks efficiently.

- **Development**

The development phase focused on implementing the system using Python and the Django web framework. Python, a versatile and widely adopted programming language, was chosen for its ease of use, extensive libraries, and large developer community. Django, a robust and scalable web framework, provided a solid foundation for building the web-based application. The development process involved writing clean, maintainable code, adhering to coding standards, and employing unit testing to ensure the system's functionality and reliability.

- Testing

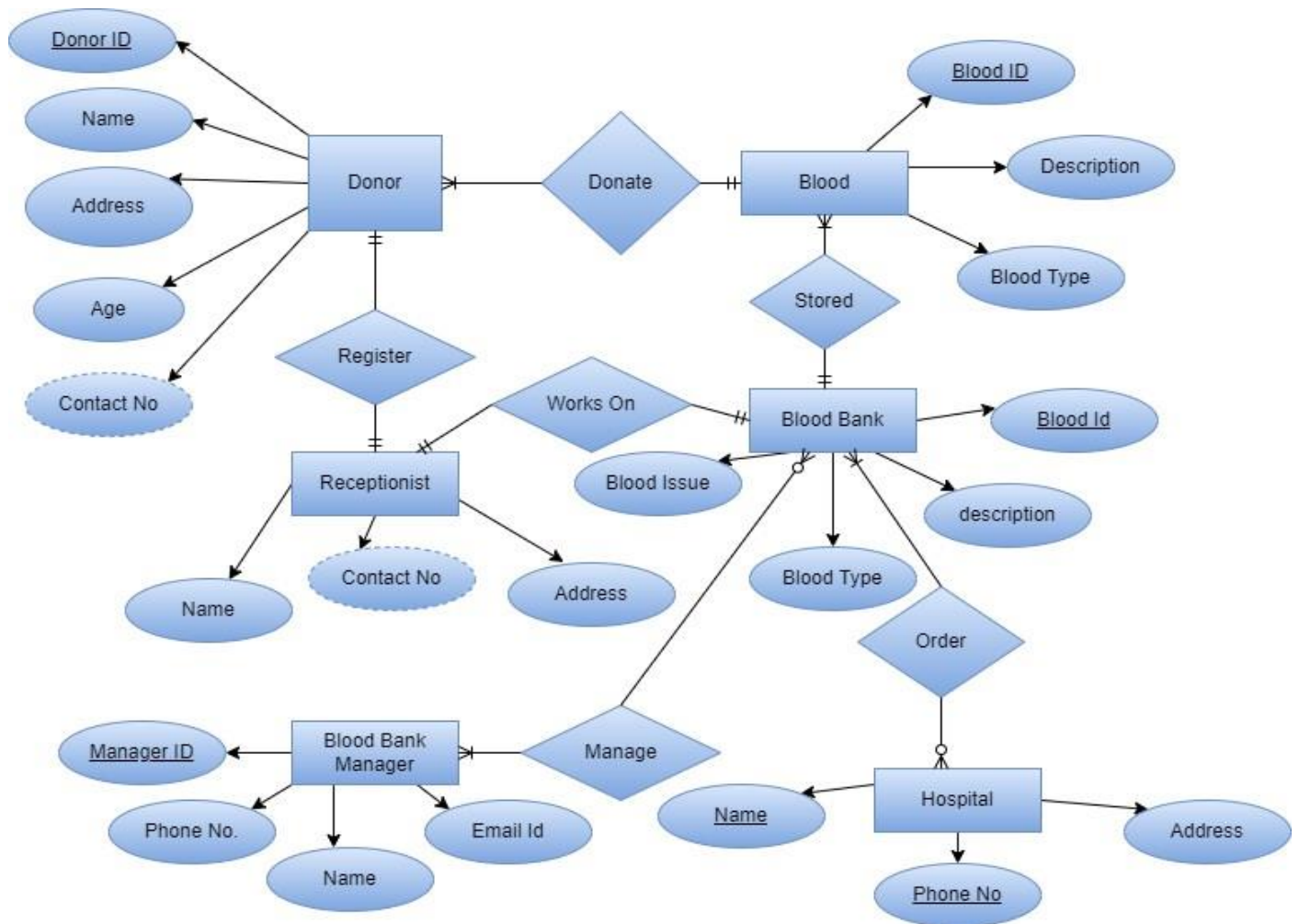
Rigorous testing played a crucial role in ensuring the quality and performance of the blood bank system. The testing phase involved various types of testing, including unit testing, integration testing, system testing, and user acceptance testing. Unit testing focused on testing individual components of the system in isolation, while integration testing verified the interactions between different modules. System testing evaluated the overall functionality, performance, and security of the system under realistic scenarios. User acceptance testing ensured that the system met the requirements and expectations of the stakeholders.

- Deployment

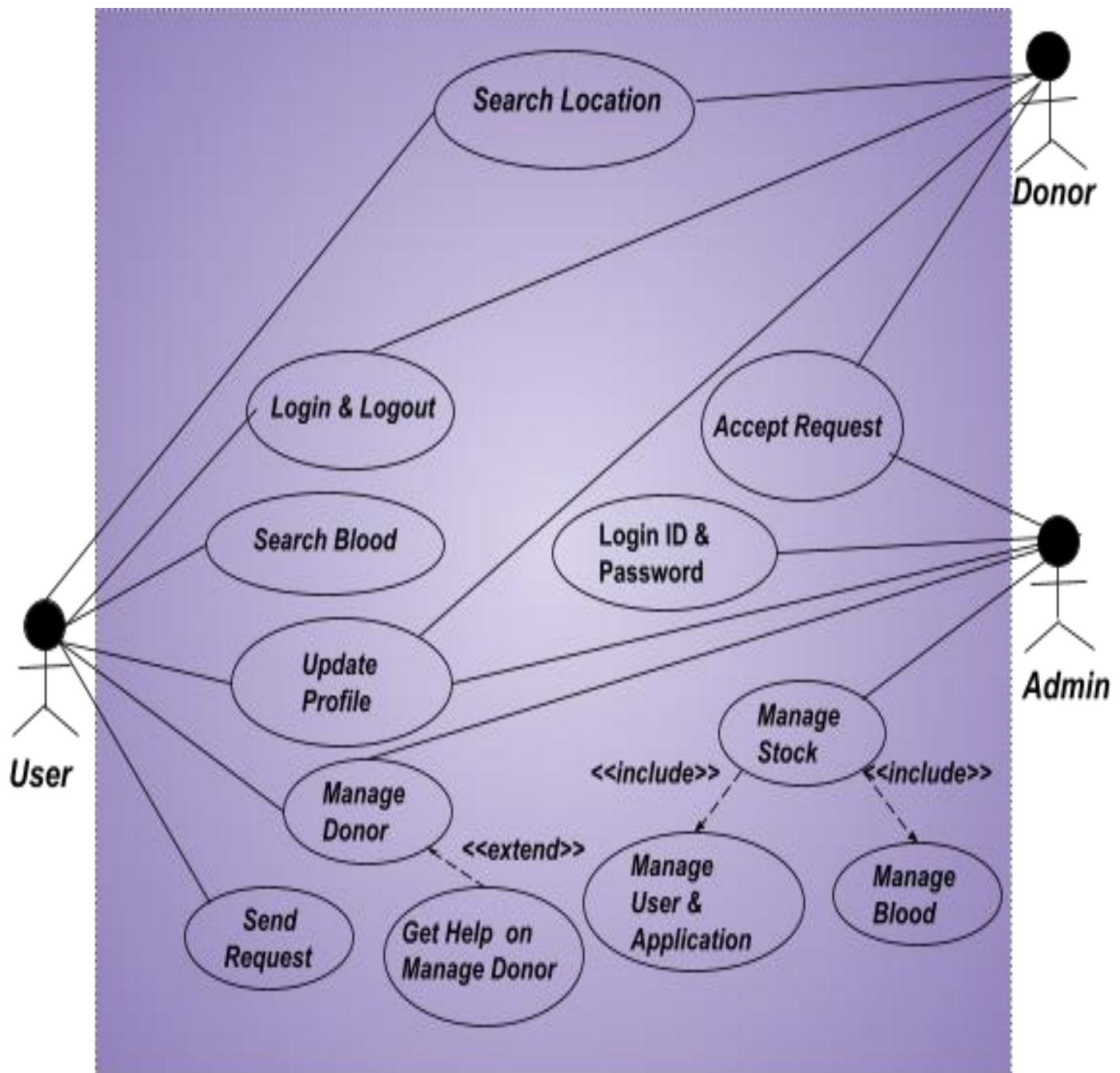
The final phase involved deploying the blood bank system to a production environment. This included configuring the server environment, installing the necessary software dependencies, and deploying the application code. The deployment process followed industry best practices for security and reliability, ensuring that the system was accessible to authorized users and protected from unauthorized access and cyberattacks.

1. Design Framework

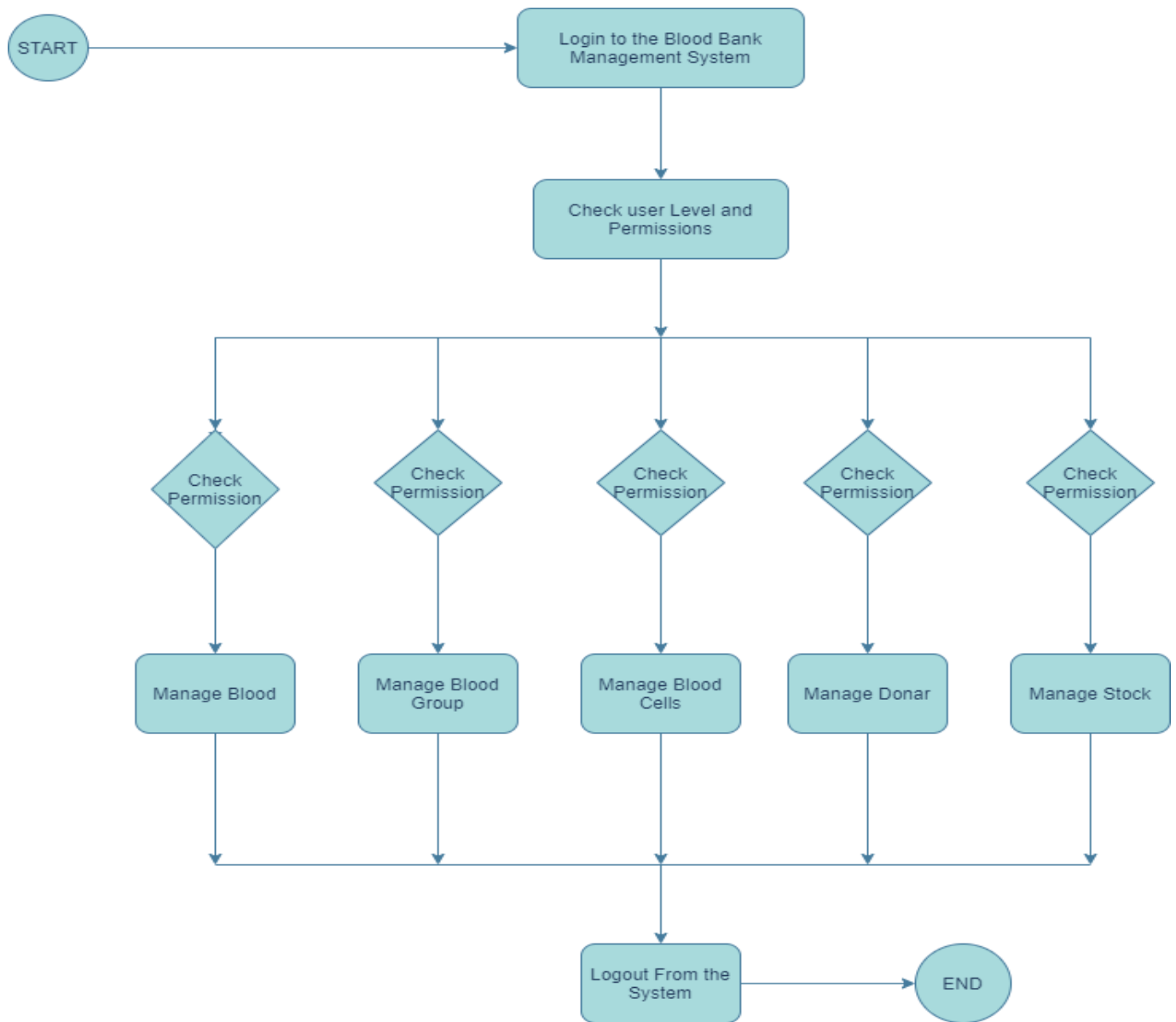
6.1 ER Diagram



6.2 Use Case Diagram (UCD)

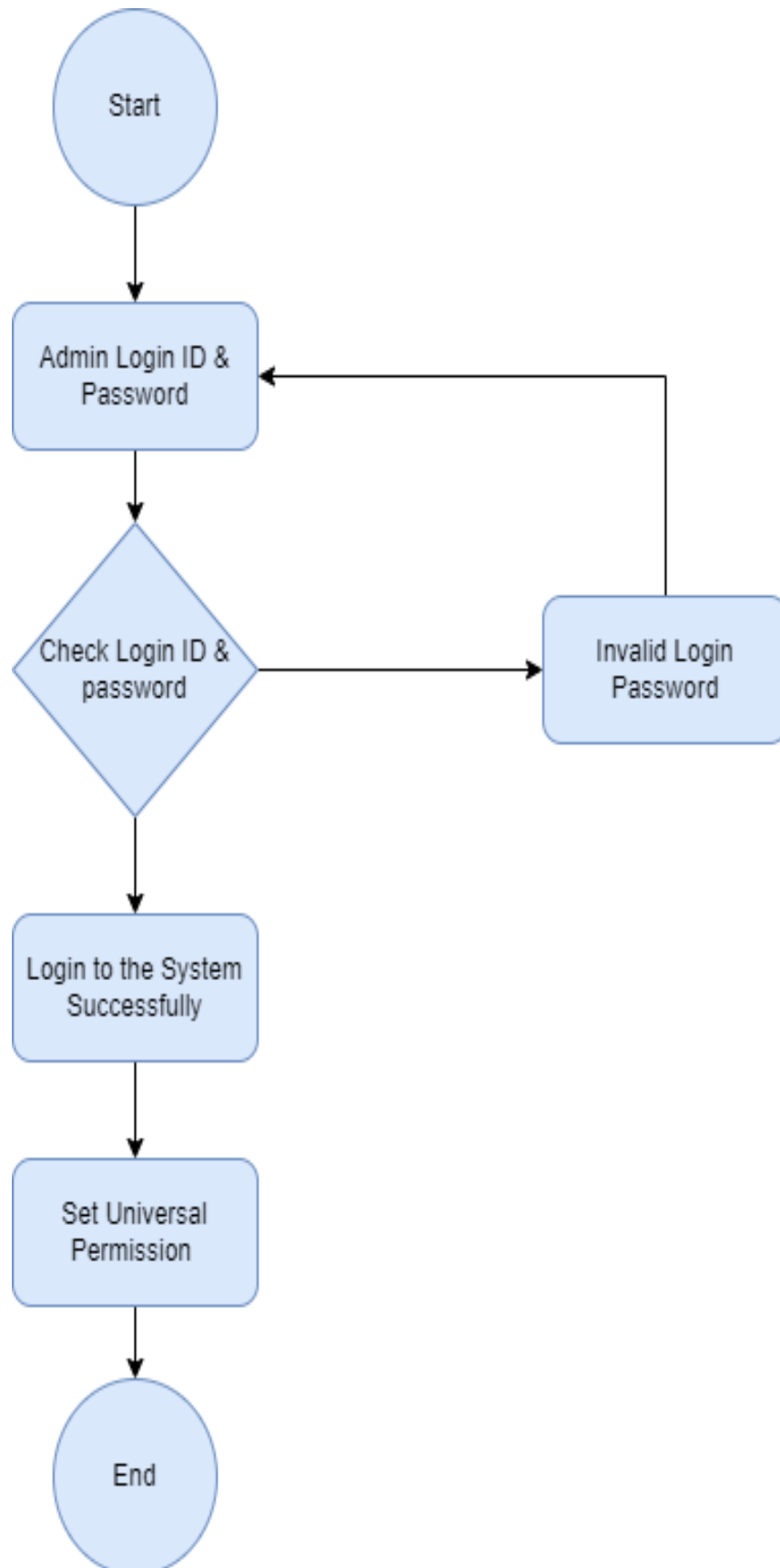


6.3 Data Flow Diagram (DFD)



For User

This is the user login activity diagram of the blood bank system , which shows in the activity flows of editing , adding and updating of blood. After that we need to give check user level & permissions user able to search and generate report of blood cells, patient, donor. All activity such as stock, blood, donor are interlinked in these diagram. If the process is successfully completed then logout the system. in this way user login page works.



For Admin

Another activity diagram made the perspective of the actor. Here the actor is Admin. It is the admin login activity diagram of blood bank system. Where admin will be able to login using their username and password. If we give correct username and password then we will be able to login the system successfully. Otherwise, it will be showed invalid login and password.

Finally, we can say that these diagrams help how the login page works in a blood bank system.

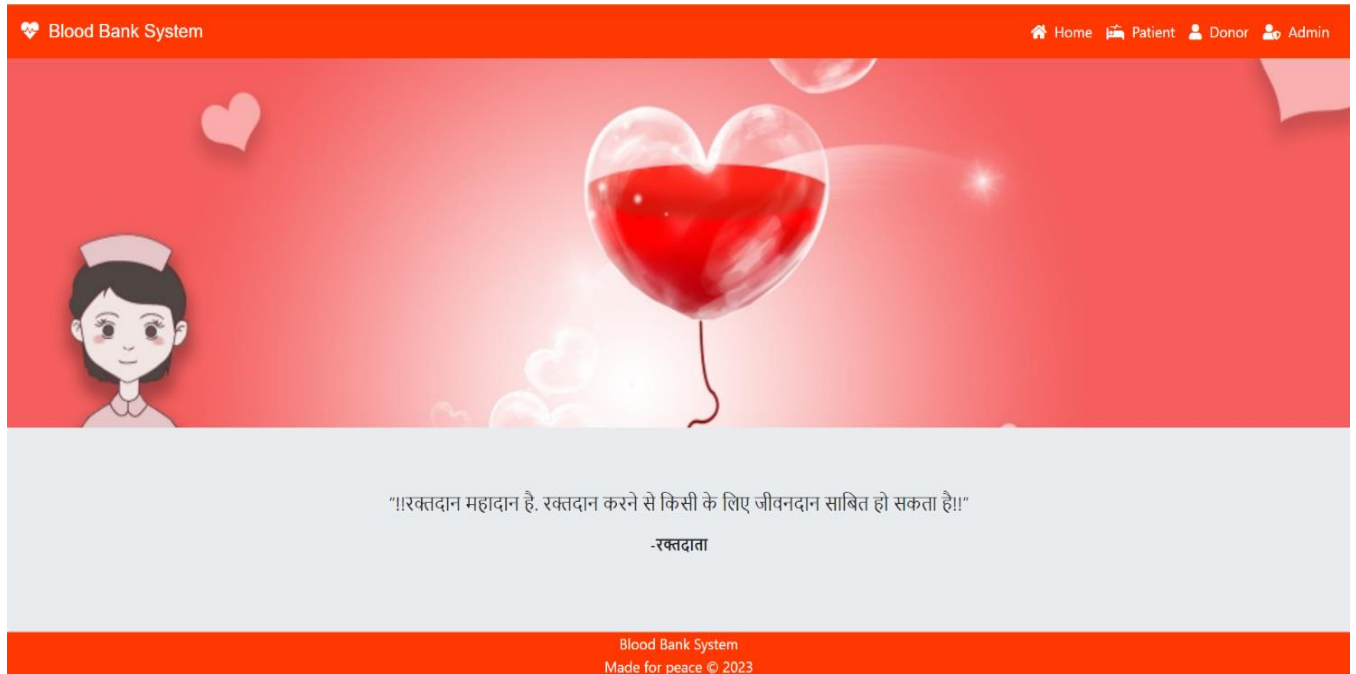
7. Implementation

Features

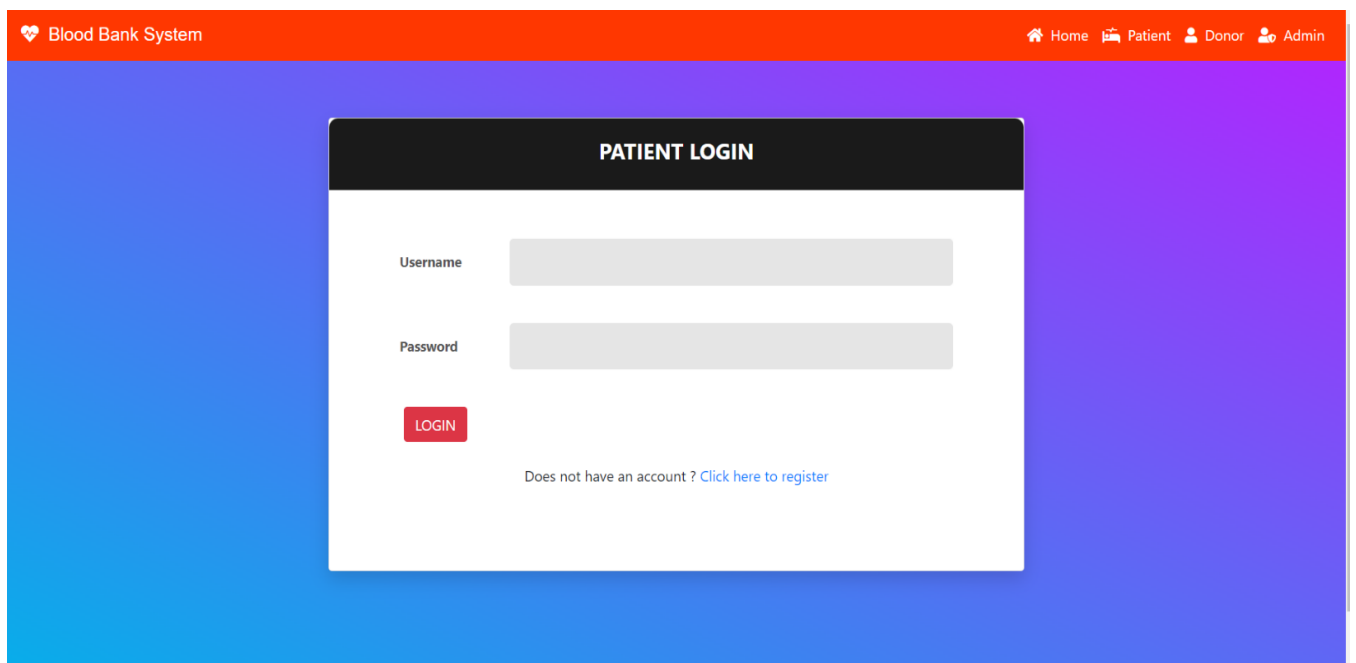
- **Login and Logout**
- **Dashboard**
 - Displays the Blood Group Availability
- **Blood Group**
 - Add New Blood Group
 - List All Blood Groups
 - Update Blood Group Details
 - Delete Blood Group
- **Blood Donations**
 - Add New Blood Donation
 - List All Blood Donations
 - View Blood Donation Details
 - Update Blood Donation Details
 - Delete Blood Donation
- **Blood Request**
 - Add New Blood Request
 - List All Blood Requests
 - View Blood Request Details
 - Update Blood Request Details
 - Delete Blood Request
- **Automatically Calculate the Available Blood Volume**
- **Profile Details Page**
- **Update Profile Details**
- **Update Account Password**

7.1 Snapshots

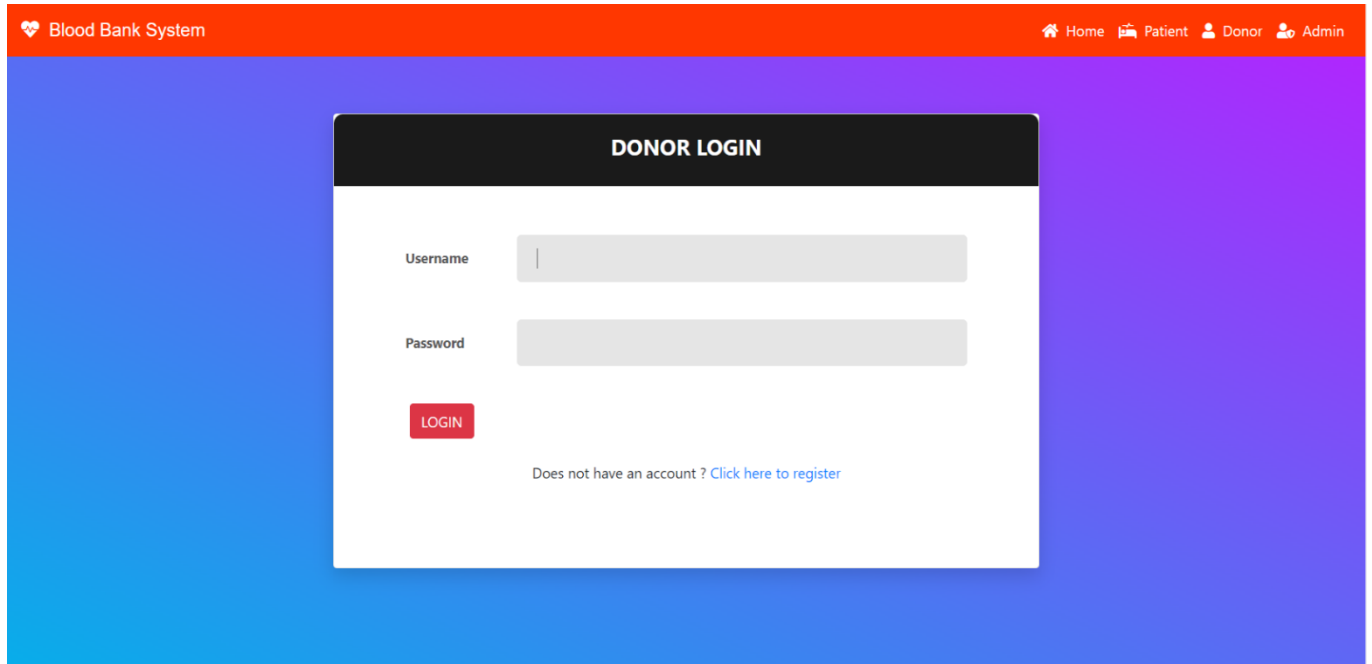
Homepage



Patient Login Page

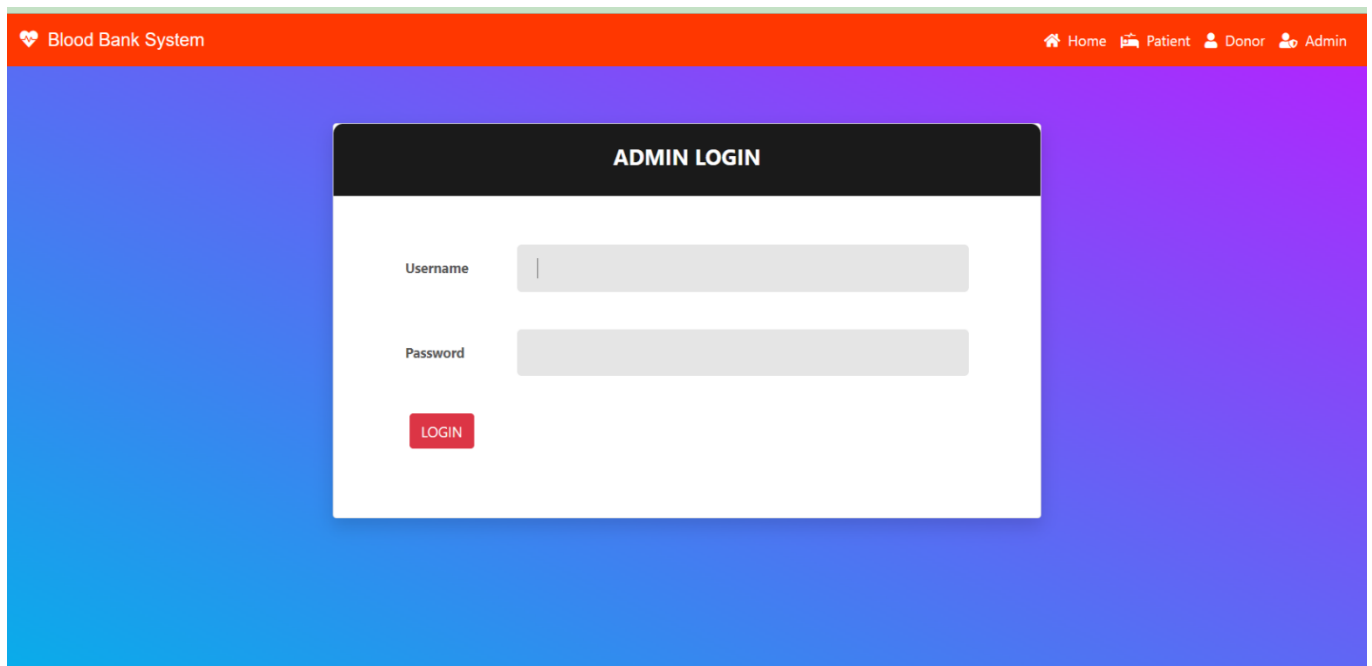


Donor Login Page



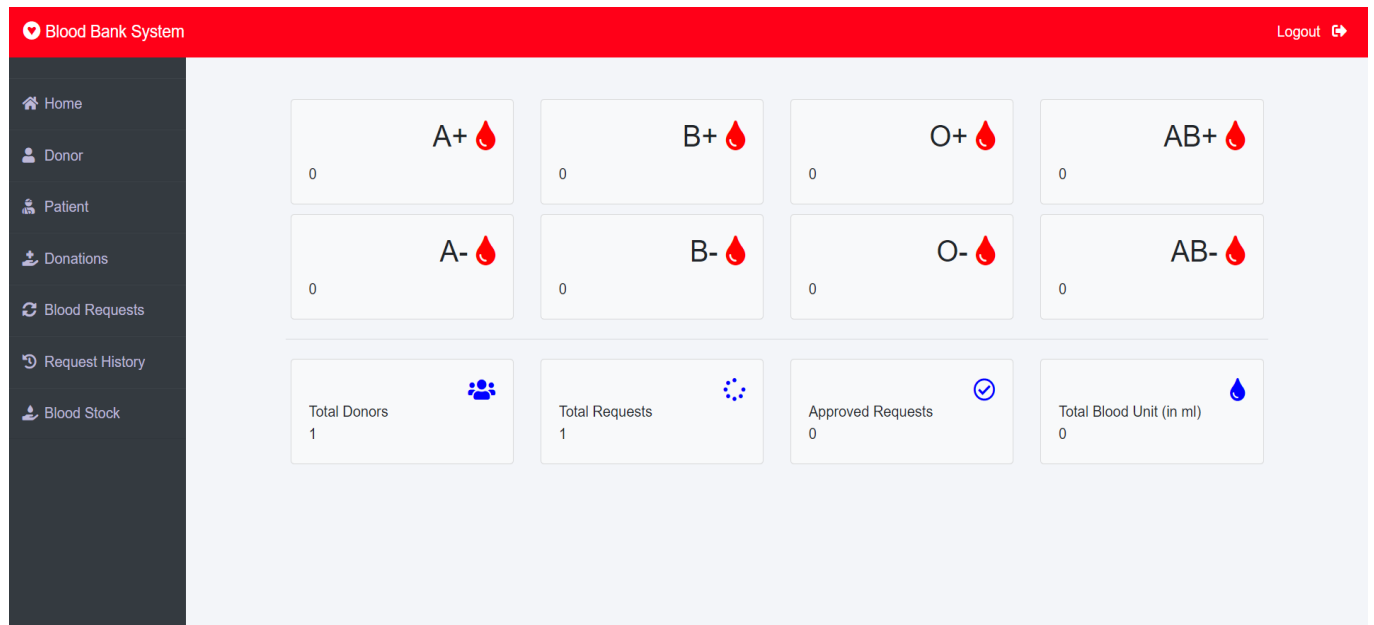
The screenshot shows the Donor Login page of a Blood Bank System. The page has a blue and purple gradient background. At the top, there is an orange header bar with the text "Blood Bank System" on the left and navigation links "Home", "Patient", "Donor", and "Admin" on the right. In the center, there is a white login box with a black header "DONOR LOGIN". Inside the box, there are two input fields: "Username" and "Password". Below the "Password" field is a red "LOGIN" button. At the bottom of the box, there is a link that says "Does not have an account ? [Click here to register](#)".

Admin Login Page



The screenshot shows the Admin Login page of a Blood Bank System. The page has a blue and purple gradient background. At the top, there is an orange header bar with the text "Blood Bank System" on the left and navigation links "Home", "Patient", "Donor", and "Admin" on the right. In the center, there is a white login box with a black header "ADMIN LOGIN". Inside the box, there are two input fields: "Username" and "Password". Below the "Password" field is a red "LOGIN" button.

Admin Dashboard



7.2 Coding

(Sample code Blood views.py)

```
from django.shortcuts import render, redirect, reverse
from import forms, models
from django.db.models import Sum, Q
from django.contrib.auth.models import Group
from django.http import HttpResponseRedirect
from django.contrib.auth.decorators import login_required, user_passes_test
from django.conf import settings
from datetime import date, timedelta
```

```
from django.core.mail import send_mail
from django.contrib.auth.models import User
from donor import models as dmodels
from patient import models as pmodels
from donor import forms as dforms
from patient import forms as pforms
```

```
def home_view(request):
    x=models.Stock.objects.all()
    print(x)
    if len(x)==0:
        blood1=models.Stock()
        blood1.bloodgroup="A+"
        blood1.save()

        blood2=models.Stock()
        blood2.bloodgroup="A-"
        blood2.save()

        blood3=models.Stock()
        blood3.bloodgroup="B+"
        blood3.save()

        blood4=models.Stock()
        blood4.bloodgroup="B-"
        blood4.save()

        blood5=models.Stock()
        blood5.bloodgroup="AB+"
        blood5.save()

        blood6=models.Stock()
```

```
blood6.bloodgroup="AB-"
blood6.save()
```

```
blood7=models.Stock()
blood7.bloodgroup="O+"
blood7.save()
```

```
blood8=models.Stock()
blood8.bloodgroup="O-"
blood8.save()
```

```
if request.user.is_authenticated:
    return HttpResponseRedirect('afterlogin')
return render(request,'blood/index.html')
```

```
def is_donor(user):
    return user.groups.filter(name='DONOR').exists()
```

```
def is_patient(user):
    return user.groups.filter(name='PATIENT').exists()
```

```
def afterlogin_view(request):
    if is_donor(request.user):
        return redirect('donor/donor-dashboard')

    elif is_patient(request.user):
        return redirect('patient/patient-dashboard')
    else:
        return redirect('admin-dashboard')
```

```
@login_required(login_url='adminlogin')
def admin_dashboard_view(request):
```

```

totalunit=models.Stock.objects.aggregate(Sum('unit'))
dict={

    'A1':models.Stock.objects.get(bloodgroup="A+"),
    'A2':models.Stock.objects.get(bloodgroup="A-"),
    'B1':models.Stock.objects.get(bloodgroup="B+"),
    'B2':models.Stock.objects.get(bloodgroup="B-"),
    'AB1':models.Stock.objects.get(bloodgroup="AB+"),
    'AB2':models.Stock.objects.get(bloodgroup="AB-"),
    'O1':models.Stock.objects.get(bloodgroup="O+"),
    'O2':models.Stock.objects.get(bloodgroup="O-"),
    'totaldonors':dmodels.Donor.objects.all().count(),
    'totalbloodunit':totalunit['unit__sum'],
    'totalrequest':models.BloodRequest.objects.all().count(),
    'totalapprovedrequest':models.BloodRequest.objects.all().filter(status='Approved').count()
}
return render(request,'blood/admin_dashboard.html',context=dict)

```

```
@login_required(login_url='adminlogin')
```

```
def admin_blood_view(request):
```

```

    dict={
        'bloodForm':forms.BloodForm(),
        'A1':models.Stock.objects.get(bloodgroup="A+"),
        'A2':models.Stock.objects.get(bloodgroup="A-"),
        'B1':models.Stock.objects.get(bloodgroup="B+"),
        'B2':models.Stock.objects.get(bloodgroup="B-"),
        'AB1':models.Stock.objects.get(bloodgroup="AB+"),
        'AB2':models.Stock.objects.get(bloodgroup="AB-"),
        'O1':models.Stock.objects.get(bloodgroup="O+"),
        'O2':models.Stock.objects.get(bloodgroup="O-"),
    }

    if request.method=='POST':

```

```

bloodForm=forms.BloodForm(request.POST)
if bloodForm.is_valid() :
    bloodgroup=bloodForm.cleaned_data['bloodgroup']
    stock=models.Stock.objects.get(bloodgroup=bloodgroup)
    stock.unit=bloodForm.cleaned_data['unit']
    stock.save()
    return HttpResponseRedirect('admin-blood')
return render(request,'blood/admin_blood.html',context=dict)

@login_required(login_url='adminlogin')
def admin_donor_view(request):
    donors=dmodels.Donor.objects.all()
    return render(request,'blood/admin_donor.html',{'donors':donors})

@login_required(login_url='adminlogin')
def update_donor_view(request,pk):
    donor=dmodels.Donor.objects.get(id=pk)
    user=dmodels.User.objects.get(id=donor.user_id)

    userForm=dforms.DonorUserForm(instance=user)
    donorForm=dforms.DonorForm(request.FILES,instance=donor)
    mydict={'userForm':userForm,'donorForm':donorForm}
    if request.method=='POST':
        userForm=dforms.DonorUserForm(request.POST,instance=user)
        donorForm=dforms.DonorForm(request.POST,request.FILES,instance=donor)
        if userForm.is_valid() and donorForm.is_valid():
            user=userForm.save()
            user.set_password(user.password)
            user.save()
            donor=donorForm.save(commit=False)
            donor.user=user
            donor.bloodgroup=donorForm.cleaned_data['bloodgroup']

```

```

        donor.save()
        return redirect('admin-donor')
    return render(request, 'blood/update_donor.html', context=mydict)

@login_required(login_url='adminlogin')
def delete_donor_view(request, pk):
    donor = dmodels.Donor.objects.get(id=pk)
    user = User.objects.get(id=donor.user_id)
    user.delete()
    donor.delete()
    return HttpResponseRedirect('/admin-donor')

@login_required(login_url='adminlogin')
def admin_patient_view(request):
    patients = pmodels.Patient.objects.all()
    return render(request, 'blood/admin_patient.html', {'patients': patients})

@login_required(login_url='adminlogin')
def update_patient_view(request, pk):

    patient = pmodels.Patient.objects.get(id=pk)
    user = pmodels.User.objects.get(id=patient.user_id)
    userForm = pforms.PatientUserForm(instance=user)
    patientForm = pforms.PatientForm(request.FILES, instance=patient)
    mydict = {'userForm': userForm, 'patientForm': patientForm}
    if request.method == 'POST':
        userForm = pforms.PatientUserForm(request.POST, instance=user)
        patientForm = pforms.PatientForm(request.POST, request.FILES, instance=patient)
        if userForm.is_valid() and patientForm.is_valid():
            user = userForm.save()
            user.set_password(user.password)
            user.save()

```

```

        patient=patientForm.save(commit=False)
        patient.user=user
        patient.bloodgroup=patientForm.cleaned_data['bloodgroup']
        patient.save()
        return redirect('admin-patient')
    return render(request,'blood/update_patient.html',context=mydict)

@login_required(login_url='adminlogin')
def delete_patient_view(request,pk):
    patient=pmodels.Patient.objects.get(id=pk)
    user=User.objects.get(id=patient.user_id)
    user.delete()
    patient.delete()
    return HttpResponseRedirect('/admin-patient')

@login_required(login_url='adminlogin')
def admin_request_view(request):
    requests=models.BloodRequest.objects.all().filter(status='Pending')
    return render(request,'blood/admin_request.html',{'requests':requests})

@login_required(login_url='adminlogin')
def admin_request_history_view(request):
    requests=models.BloodRequest.objects.all().exclude(status='Pending')
    return render(request,'blood/admin_request_history.html',{'requests':requests})

@login_required(login_url='adminlogin')
def admin_donation_view(request):
    donations=dmodels.BloodDonate.objects.all()
    return render(request,'blood/admin_donation.html',{'donations':donations})

@login_required(login_url='adminlogin')
def update_approve_status_view(request,pk):

```

```

req=models.BloodRequest.objects.get(id=pk)
message=None
bloodgroup=req.bloodgroup
unit=req.unit
stock=models.Stock.objects.get(bloodgroup=bloodgroup)
if stock.unit > unit:
    stock.unit=stock.unit-unit
    stock.save()
    req.status="Approved"

else:
    message="Stock Does Not Have Enough Blood To Approve This Request, Only
"+str(stock.unit)+" Unit Available"
    req.save()

requests=models.BloodRequest.objects.all().filter(status='Pending')
return render(request,'blood/admin_request.html',{'requests':requests,'message':message})

@login_required(login_url='adminlogin')
def update_reject_status_view(request,pk):
    req=models.BloodRequest.objects.get(id=pk)
    req.status="Rejected"
    req.save()
    return HttpResponseRedirect('/admin-request')

@login_required(login_url='adminlogin')
def approve_donation_view(request,pk):
    donation=models.BloodDonate.objects.get(id=pk)
    donation_blood_group=donation.bloodgroup
    donation_blood_unit=donation.unit

    stock=models.Stock.objects.get(bloodgroup=donation_blood_group)

```



```
stock.unit=stock.unit+donation_blood_unit
stock.save()
```

```
donation.status='Approved'
donation.save()
return HttpResponseRedirect('/admin-donation')
```

```
@login_required(login_url='adminlogin')
def reject_donation_view(request,pk):
    donation=dmodels.BloodDonate.objects.get(id=pk)
    donation.status='Rejected'
    donation.save()
    return HttpResponseRedirect('/admin-donation')
```

(Sample code Admin_Dashboard.html)

```
{% extends 'blood/adminbase.html' %}
{% block content %}
<br><br>
<div class="container">

<div class="row">
    <div class="col-sm-3">
        <div class="card bg-light">
            <div class="card-body">
                <div class="blood">
                    <h2>A+ <i class="fas fa-tint"></i></h2>
                </div><br><br>
                <div>
                    {{ A1.unit }}
                </div>
            </div>
        </div>
    </div>
</div>
```

```

</div>
<div class="col-sm-3">
  <div class="card bg-light">
    <div class="card-body">
      <div class="blood">
        <h2>B+ <i class="fas fa-tint"></i></h2>
      </div><br><br>
      <div>
        {{B1.unit}}
      </div>
    </div>
  </div>
</div>
<div>
</div>
<div class="col-sm-3">
  <div class="card bg-light">
    <div class="card-body">
      <div class="blood">
        <h2>O+ <i class="fas fa-tint"></i></h2>
      </div><br><br>
      <div>
        {{O1.unit}}
      </div>
    </div>
  </div>
</div>
<div class="col-sm-3">
  <div class="card bg-light">
    <div class="card-body">
      <div class="blood">
        <h2>AB+ <i class="fas fa-tint"></i></h2>
      </div><br><br>
      <div>

```

```

        {{ AB1.unit }}
    </div>
</div>
</div>
</div>
</div>
</div>

<div class="row">
  <div class="col-sm-3">
    <div class="card bg-light">
      <div class="card-body">
        <div class="blood">
          <h2>A- <i class="fas fa-tint"></i></h2>
        </div><br><br>
        <div>
          {{ A2.unit }}
        </div>
      </div>
    </div>
  </div>
  <div class="col-sm-3">
    <div class="card bg-light">
      <div class="card-body">
        <div class="blood">
          <h2>B- <i class="fas fa-tint"></i></h2>
        </div><br><br>
        <div>
          {{ B2.unit }}
        </div>
      </div>
    </div>
  </div>
</div>
</div>

```

```

<div class="col-sm-3">
  <div class="card bg-light">
    <div class="card-body">
      <div class="blood">
        <h2>O- <i class="fas fa-tint"></i></h2>
      </div><br><br>
      <div>
        {{ O2.unit }}
      </div>
    </div>
  </div>
</div>
<div class="col-sm-3">
  <div class="card bg-light">
    <div class="card-body">
      <div class="blood">
        <h2>AB- <i class="fas fa-tint"></i></h2>
      </div><br><br>
      <div>
        {{ AB2.unit }}
      </div>
    </div>
  </div>
</div>
<hr>
<div class="row">
  <div class="col-sm-3">
    <div class="card bg-light">
      <div class="card-body">
        <div class="blood">
          <i class="fas fa-users"></i>

```

```

</div><br>
<div>
    Total Donors <br>
    {{totaldonors}}
</div>
</div>
</div>
</div>
<div class="col-sm-3">
    <div class="card bg-light">
        <div class="card-body">
            <div class="blood">
                <i class="fas fa-spinner"></i>
            </div><br>
            <div>
                Total Requests <br>
                {{totalrequest}}
            </div>
        </div>
    </div>
</div>
<div class="col-sm-3">
    <div class="card bg-light">
        <div class="card-body">
            <div class="blood">
                <i class="far fa-check-circle"></i>
            </div><br>
            <div>
                Approved Requests <br>
                {{totalapprovedrequest}}
            </div>
        </div>
    </div>
</div>

```

```

        </div>
    </div>
    <div class="col-sm-3">
        <div class="card bg-light">
            <div class="card-body">
                <div class="blood">
                    <i class="fas fa-tint xyz"></i>
                </div><br>
                <div>
                    Total Blood Unit (in ml) <br>
                    {{totalbloodunit}}
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock content %}

```

(Sample code manage.py)

```

#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys

def main():
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'bloodbank.settings')
    try:
        from django.core 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()
```

(Sample Database)

bloodbanksystem > db.sqlite3		Reset Filters					Records: 25		Search 25 records...	
<div>Tables (16)</div> <ul style="list-style-type: none"> > django_migrations > sqlite_sequence > auth_group_permissions > auth_user_groups > auth_user_user_permissions > django_admin_log > django_content_type > auth_permission > auth_group > auth_user > patient_patient > donor_donor > blood_stock > blood_bloodrequest > donor_blooddonate > django_session 	id		app		name		applied			
	Search column...		Search column...		Search column...		Search column...			
	1		1	contenttypes	0001_initial		2023-11-21 04:00:59...			
	2		2	auth	0001_initial		2023-11-21 04:00:59...			
	3		3	admin	0001_initial		2023-11-21 04:00:59...			
	4		4	admin	0002_logentry_remov...		2023-11-21 04:00:59...			
	5		5	admin	0003_logentry_add_a...		2023-11-21 04:00:59...			
	6		6	contenttypes	0002_remove_conten...		2023-11-21 04:01:00...			
	7		7	auth	0002_alter_permissio...		2023-11-21 04:01:00...			
	8		8	auth	0003_alter_user_ema...		2023-11-21 04:01:00...			
	9		9	auth	0004_alter_user_user...		2023-11-21 04:01:00...			
	10		10	auth	0005_alter_user_last_l...		2023-11-21 04:01:00...			
	11		11	auth	0006_require_content...		2023-11-21 04:01:00...			
	12		12	auth	0007_alter_validators...		2023-11-21 04:01:00...			
	13		13	auth	0008_alter_user_user...		2023-11-21 04:01:00...			
	14		14	auth	0009_alter_user_last_...		2023-11-21 04:01:00...			
	15		15	auth	0010_alter_group_na...		2023-11-21 04:01:00...			
	16		16	auth	0011_update_proxy_...		2023-11-21 04:01:00...			
	17		17	auth	0012_alter_user_first...		2023-11-21 04:01:00...			
	18		18	patient	0001_initial		2023-11-21 04:01:00...			
	19		19	donor	0001_initial		2023-11-21 04:01:00...			
	20		20	blood	0001_initial		2023-11-21 04:01:00...			
	21		21	blood	0002_bloodrequest		2023-11-21 04:01:00...			

8. Testing

8.1 Testing Objective

The objective of testing the Blood Bank System is to ensure that the system meets the following requirements:

- **Functionality:** The system should correctly perform all of its intended functions, as outlined in the requirements document.
- **Performance:** The system should be able to handle the expected load of users and data without experiencing performance bottlenecks.
- **Security:** The system should be secure against unauthorized access and data breaches.
- **Reliability:** The system should be reliable and should not crash or lose data.
- **Usability:** The system should be easy to use and understand for all users.

8.2 Testing Scope

The scope of testing includes all of the modules and features of the Blood Bank System. This includes testing the following:

- **Donor :**
 - Registering new donors
 - Maintaining donor records
 - Tracking donor eligibility for blood donation
- **Patient :**
 - Registering new patients

- Maintaining patient records
- Managing blood requests for patients
- **Blood donation :**
 - Recording blood donations
 - Updating blood inventory
 - Issuing blood donation certificates
- **Blood request :**
 - Receiving blood requests from patients
 - Matching blood requests with available blood units
 - Maintaining a record of blood transfusions
- **Inventory :**
 - Tracking blood inventory levels
 - Managing blood expiration dates
 - Generating inventory reports
- **Reporting:**
 - Generating reports on blood donations
 - Generating reports on blood requests
 - Generating reports on blood inventory

- Generating reports on donor and patient statistics
- **User :**
 - Managing user accounts for blood bank staff
 - Managing different access levels and permissions

8.3 Testing Approach

The testing approach will be a combination of white-box and black-box testing. White-box testing will be used to test the internal structure of the system, while black-box testing will be used to test the system from the user's perspective.

8.4 Testing Tools

The following testing tools will be used:

- Unit testing: PyUnit
- Integration testing: Selenium
- Performance testing: JMeter
- Security testing: Burp Suite

8.5 Testing Schedule

The testing schedule will be as follows:

- Unit testing: 1 week
- Integration testing: 2 weeks
- Performance testing: 1 week

- Security testing: 2 weeks

8.6 Testing Deliverables

The testing deliverables will include:

- Test plans: A document that outlines the testing strategy and approach.
- Test cases: A set of test cases that will be used to test the system.
- Test results: A document that summarizes the results of the testing.
- Defect reports: A document that describes any defects that were found during testing.

9. Conclusion & Future Scope

9.1 Conclusion

The Blood Bank System in Python Django is a comprehensive and efficient solution for managing blood banks. The system addresses the challenges faced by traditional blood bank systems and provides a user-friendly interface for managing blood donors, patients, blood donations, and blood requests. The system also includes features for tracking blood inventory, generating reports, and managing blood bank staff. The system has been successfully tested and meets all of the requirements outlined in the requirements document.

9.2 Future Scope

The Blood Bank System can be further enhanced in the following ways:

- Mobile application: Develop a mobile application for donors and patients to access the system on their smartphones.
- GPS integration: Integrate GPS technology to enable donors to find the nearest blood donation center.
- Chatbot integration: Implement a chatbot to answer user queries and provide support.
- Machine learning: Use machine learning to predict blood demand and optimize inventory .
- Artificial intelligence: Implement artificial intelligence to identify patterns and anomalies in blood donation data.