

INTRODUCTION

The Blood Bank Management System is a comprehensive solution designed to streamline the management of blood banks, ensuring efficient tracking, handling, and distribution of blood and blood products. Blood is a vital resource in healthcare, often necessary in life-saving situations such as surgeries, trauma care, and treatment of specific health conditions. However, the availability and proper management of blood can be a challenge, as blood is a perishable commodity with a limited shelf life and specific storage requirements.

This system aims to simplify and automate key operations within a blood bank, including blood donation registration, blood collection, inventory management, blood testing, and blood transfusion requests. By creating a centralized and easily accessible database, the Blood Bank Management System facilitates accurate record-keeping, ensures that blood is readily available for patients in need, and helps minimize wastage.

With this system, blood banks can monitor their blood stocks in real-time, maintain updated records of donors and recipients, and optimize supply chain logistics. It is a critical tool that not only enhances operational efficiency but also contributes to improving healthcare delivery by ensuring a reliable supply of safe blood to those who need it.

This report provides an overview of the system's features, functionality, and its contribution to the overall efficiency and effectiveness of blood bank operations.

SCOPE

The Blood Bank Management System (BBMS) is designed to cover various essential aspects of blood bank operations, helping institutions maintain a reliable and efficient blood supply chain. Its scope includes:

1. Donor Registration and Management :

- Registration of new donors, including their personal information, blood type, health status, and eligibility.
- Record-keeping for regular and occasional donors to facilitate follow-ups and notifications for future donations.
- Tracking of donor history to ensure compliance with donation frequency limits and health criteria.

2. Inventory Management :

- Monitoring of blood stock levels, including blood types, quantities, and expiration dates.
- Alerts for low-stock blood types and upcoming expirations to reduce wastage.
- Categorizing blood components (e.g., red cells, plasma, platelets) for specific usage and ensuring proper storage.

3. Blood Collection and Testing :

- Recording details of blood collection, including volume, collection date, and handling information.
- Documenting and managing the results of blood tests for infectious diseases and compatibility checks.
- Ensuring that only safe, tested blood is added to the inventory.

4. Blood Request and Distribution :

- Facilitating requests from hospitals, clinics, and patients for specific blood types.
- Managing the distribution process, including tracking blood units and generating request status updates.
- Prioritizing urgent requests and coordinating with hospitals during emergencies.

5. Reporting and Analytics :

- Generating reports on inventory status, donor demographics, usage patterns, and request fulfillment.
- Providing analytics for data-driven decision-making, such as identifying high-demand blood types and peak donation periods.
- Offering insights for resource planning and effective donor engagement strategies.

6. Notification System :

- Sending notifications to donors regarding donation eligibility and blood drives.
- Alerting staff about low inventory levels or blood nearing expiration to optimize resource use.
- Notifying hospitals and clinics about blood availability or potential delays.

7. User Access and Security :

- Providing secure access to different user roles, including administrators, donors, and hospitals.
- Ensuring data privacy and compliance with relevant health data regulations.
- Enabling permissions to restrict access based on user roles, protecting sensitive donor and patient data.

8. Integration with External Systems :

- Enabling integration with hospital management systems and national health databases for better coordination.
- Supporting inter-blood bank collaboration to facilitate blood transfers during shortages or emergencies.

The Blood Bank Management System aims to enhance efficiency, improve resource utilization, and contribute to better healthcare outcomes by ensuring a safe, reliable, and well-coordinated blood supply system.

LITERATURE REVIEW

The Blood Bank Management System (BBMS) is a specialized healthcare management tool that has undergone significant advancements over recent decades. With the global rise in demand for safe and efficient blood transfusion services, numerous studies and innovations have contributed to the design and functionality of these systems. This review explores research on BBMS development, addressing core functionalities, challenges, technological innovations, and their implications for healthcare.

1. Core Functionality and Design of Blood Bank Systems :

Early blood bank systems were primarily manual, relying on paper records and manual inventory tracking, which often led to data inaccuracies and inefficiencies (Bisht et al., 2014). With the digitization of healthcare, computerized systems were developed to automate key tasks such as donor registration, blood collection tracking, and inventory management. Modern BBMSs, as studied by Rao et al. (2018), include advanced features like real-time inventory monitoring, which ensures blood availability and reduces wastage by alerting staff to expiring units. These systems enhance operational efficiency by streamlining workflows and reducing errors associated with manual processes.

2. Challenges in Blood Bank Management :

Blood banks face challenges in ensuring a safe, continuous blood supply due to factors like limited donor availability, blood storage constraints, and stringent regulatory compliance. According to a study by Sharma and Prasad (2017), BBMSs must address issues like traceability, as well as safety and compatibility testing, to minimize transfusion-related risks. In this context, significant attention has been given to enhancing the system's ability to perform comprehensive blood testing and ensure compliance with legal standards, such as data privacy and traceability mandates.

3. Technological Advances in BBMS :

The integration of new technologies like cloud computing, artificial intelligence (AI), and the Internet of Things (IoT) has been transformative in blood bank management. A study by Zhang et al. (2020) highlights the adoption of cloud-based BBMS, which facilitates centralized data management, supports remote access, and improves disaster recovery capabilities. Additionally, AI-driven analytics can predict blood demand patterns, enabling proactive donor engagement and inventory management. IoT integration allows for real-time monitoring of blood storage conditions, such as temperature and humidity, ensuring compliance with storage standards to maintain blood quality.

4. Data Analytics and Predictive Modeling :

Research shows that data analytics in BBMS has a positive impact on resource management and planning. According to Gupta and Singh (2019), data-driven insights from historical blood usage and donation patterns can help predict high-demand periods and optimize blood drives. Predictive modeling, integrated into many modern BBMS, enables hospitals to anticipate blood requirements, reducing the risk of shortages and ensuring timely access to needed blood types.

5. Impact on Healthcare Delivery and Patient Outcomes :

The implementation of a robust BBMS has shown to significantly improve healthcare outcomes by reducing wait times for blood transfusions, ensuring high standards of blood safety, and enhancing emergency response capabilities (Chaudhary et al., 2021). Furthermore, by digitizing records and streamlining the transfusion request process, BBMS contributes to better patient care and more efficient healthcare delivery.

6. Inter-Organization and National-Level Coordination :

Several studies emphasize the importance of inter-organizational collaboration in BBMS. According to Anderson et al. (2016), a national or regional networked BBMS can help hospitals coordinate during shortages or emergencies, ensuring equitable distribution of resources. These systems enable blood banks to exchange information in real-time, facilitating rapid transfers and enhancing the overall resilience of the healthcare system.

The literature reveals that advancements in BBMS technology have led to significant improvements in the management and distribution of blood resources, with a positive impact on patient outcomes and healthcare delivery efficiency. Nonetheless, challenges remain in ensuring data security, managing donor shortages, and optimizing blood distribution. Future research may focus on leveraging emerging technologies like AI and machine learning for even more refined predictive models and expanding inter-blood-bank networks for improved resilience in critical situations.

RELEVANT TECHNOLOGIES

1. Python:

Overview: Python is a versatile, high-level programming language known for its simplicity, readability, and ease of learning. It emphasizes code readability and expressiveness, making it an ideal choice for a wide range of applications, including web development.



Usage in the Project: In the context of the PG Portal project, Python serves as the primary programming language for implementing the backend logic. This includes handling user authentication, managing database interactions, implementing business logic, and orchestrating the overall functionality of the application. Python's extensive standard library and third-party packages provide developers with a wealth of tools and resources to streamline development tasks and build robust web applications efficiently.

Advantages: Python offers several advantages for web development projects:

- **Readability:** Python's clean and concise syntax promotes code readability and reduces the likelihood of errors, making it easier for developers to understand and maintain code.
- **Extensive Libraries and Frameworks:** Python boasts a rich ecosystem of libraries and frameworks tailored for web development, such as Django, Flask, and Pyramid. These frameworks provide pre-built components and abstractions that simplify common tasks, accelerate development, and promote best practices.
- **Scalability:** Python's versatility and scalability make it well-suited for projects of varying sizes and complexities. Whether developing a small-scale application or a large-scale enterprise system, Python offers the flexibility to adapt and scale according to project requirements.
- **Community Support:** Python enjoys a vibrant and active community of developers, enthusiasts, and contributors who continually enhance the language and ecosystem. This community-driven approach fosters collaboration, knowledge sharing, and innovation, providing developers with valuable resources and support.

2. Django:

Overview: Django is a high-level web framework written in Python, designed to facilitate rapid development of web applications with a clean and pragmatic design philosophy. It follows the "batteries-included" approach, providing developers with a comprehensive set of tools and features to build scalable, maintainable, and secure web applications.



Usage in the Project: In the PG Portal project, Django serves as the foundation of the web application framework. It provides a structured and modular architecture for organizing code, handling URL routing, rendering templates, managing database models, and implementing user

authentication. By leveraging Django's built-in components and conventions, developers can focus on implementing application-specific logic and features without getting bogged down by repetitive tasks.

Advantages: Django offers numerous advantages for web development:

- **Rapid Development:** Django's high-level abstractions and built-in components enable developers to prototype and build web applications quickly, reducing time-to-market and development costs.
- **DRY Principle:** Django embraces the "Don't Repeat Yourself" (DRY) principle, encouraging code reusability, modularity, and maintainability. By minimizing redundancy and promoting encapsulation, Django simplifies code maintenance and reduces the risk of errors.
- **ORM (Object-Relational Mapping):** Django's ORM abstracts database interactions, allowing developers to work with database entities using Python objects and methods. This abstraction simplifies database operations, enhances portability, and mitigates the risk of SQL injection attacks.
- **Admin Interface:** Django provides a powerful and customizable admin interface for managing application data, user accounts, and permissions. This built-in feature streamlines administrative tasks, enhances productivity, and improves user experience.
- **Security:** Django incorporates robust security features by default, including protection against common web vulnerabilities such as cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL injection. Additionally, Django's authentication system offers flexible user authentication and authorization mechanisms to safeguard application resources.

- **3. Web Development Tools:**

- **HTML (Hypertext Markup Language):**

- **Overview:** HTML is the standard mark-up language used to structure and present content on the web. It defines the structure of web pages using a hierarchical system of elements and attributes.
- **Usage in the Project:** HTML forms the backbone of the PG Portal's user interface, providing the structure and semantics for displaying content, forms, and interactive elements to users.
- **Advantages:** HTML's simplicity and universality make it an essential tool for web development. Its standardized syntax ensures cross-browser compatibility and accessibility, while its semantic markup enhances search engine optimization (SEO) and content discoverability.

HTML



CSS (Cascading Style Sheets):

- **Overview:** CSS is a style sheet language used for styling the presentation of HTML documents. It allows developers to control the layout, typography, colors, and visual aspects of web pages.
- **Usage in the Project:** CSS is employed in the PG Portal project to define the visual appearance and layout of HTML elements, ensuring a cohesive and aesthetically pleasing user interface.
- **Advantages:** CSS simplifies the process of styling web pages by providing a declarative syntax and powerful features such as selectors, inheritance, and cascading rules. It enables developers to create responsive and visually engaging designs that enhance the user experience across devices and screen sizes.



JavaScript:

- **Overview:** JavaScript is a versatile scripting language commonly used for adding interactivity and dynamic behavior to web pages. It enables client-side scripting, DOM manipulation, event handling, and asynchronous communication with the server.
- **Usage in the Project:** JavaScript enhances the functionality of the PG Portal by providing interactive features such as form validation, dynamic content loading, and user interface enhancements.
- **Advantages:** JavaScript empowers developers to create rich and immersive web experiences by leveraging its event-driven architecture and extensive ecosystem of libraries and frameworks. Its ability to execute code directly within the browser enhances performance and responsiveness, reducing server load and latency.

JavaScript



DATABASE SCHEMA

The database schema for the Blood Bank Management System provides the structure of the database, detailing tables, fields, relationships, and constraints. Below is a simplified representation of the schema for the project:

Donor Table:

Fields:

Donor-id (Primary Key): Unique identifier for each donor.

Name: Full name of the donor.

Blood Group: Blood type of the donor (e.g., A+, B-, O+).

Date of Birth: Donor's birth date.

Gender: Gender of the donor (e.g., "M" for male, "F" for female).

Contact Number: Phone number to contact the donor.

Email: Email address of the donor.

Last Donation Date: Date of the donor's most recent blood donation.

City: The city where the donor resides.

Is Available: Boolean indicating if the donor is currently available for donation.

Receiver Table:

Fields:

Receiver-id (Primary Key): Unique identifier for each receiver.

Name: Full name of the person or entity receiving blood.

Blood Group Needed: Blood type needed by the receiver.

Contact Number: Phone number to reach the receiver.

Email: Email address of the receiver.

Hospital Name: Name of the hospital or clinic where the blood is required.

City: The city where blood is needed.

Request Date: Date when the blood request was made.

Urgency Level: Indicates the urgency of the blood requirement (e.g., "Low", "Medium", "High").

Community Table:

Fields:

Community-id (Primary Key): Unique identifier for each community.

Name: Name of the community group.

City: City where the community operates.

Contact Number: Contact number for the community group.

Email: Email address for community correspondence.

Events: Information about any events organized by the community.

Total Members: Number of members in the community.

Created At: Date and time when the community group was created.

SYSTEM DESIGN

Architecture Overview

The Blood Bank Management System is structured upon a robust and scalable micro services architecture, enabling modular development and deployment of its various components. Each

micro service is designed to encapsulate a specific functionality, such as donor management, blood inventory tracking, and request processing, ensuring a clear separation of concerns and facilitating flexibility in both development and maintenance.

Load balancing and containerization technologies, such as Docker and Kubernetes, are leveraged to optimize resource utilization, ensuring high availability and fault tolerance. Communication between microservices is facilitated through lightweight protocols like RESTful APIs or message queues, promoting loose coupling and interoperability. This architecture provides a solid foundation for the platform's growth and evolution, allowing it to adapt to changing requirements and handle increased demand efficiently, especially during critical situations or emergencies.

Database Design

In designing the database architecture for the Blood Bank Management System, careful consideration is given to ensuring data integrity, reliability, and performance. The database schema adheres to ACID (Atomicity, Consistency, Isolation, Durability) properties to maintain data consistency and transactional integrity, critical in managing sensitive medical and inventory data.

Entity-Relationship Diagrams (ERDs) are utilized to model relationships between core entities, including donors, blood inventory, requests, hospital partners, and administrative data. This provides a clear blueprint of the data structure and relationships, supporting both user needs and system requirements.

Normalization techniques are applied to eliminate data redundancy, reduce storage requirements, and minimize update anomalies, resulting in an efficient and well-optimized schema. Indexing and query optimization strategies are implemented to improve performance, especially for frequently accessed data (such as inventory levels and donor information) and complex queries, ensuring the system remains responsive and scalable as it expands. This architecture lays a strong foundation for secure, efficient data handling and enables the platform to support high availability, reliability, and responsiveness as demand grows.

User Interface Design

The user interface (UI) design of the Blood Bank Management System is crafted with a focus on delivering an intuitive, responsive, and visually appealing experience to users across various devices and screen sizes. HTML5, CSS3, and JavaScript libraries such as React.js or Vue.js are used to develop modular, reusable, and maintainable UI components, allowing for efficient development and customization to meet the needs of various user groups, including donors, hospitals, and administrators.

UI design patterns such as cards, grids, and modals are used to present information in a structured and visually engaging way, making it easy for users to view and interact with key information, such as blood availability, donation schedules, and request statuses. Accessibility features, including keyboard navigation support, screen reader compatibility, and semantic HTML markup, are integrated into the UI design to ensure compliance with Web Content Accessibility Guidelines (WCAG) and enable all users to access and interact with the platform seamlessly.

This approach to UI design aims to provide a frictionless, inclusive user experience that enhances usability, encouraging user engagement and the effective adoption of the Blood Bank Management System.

IMPLEMENTATION

LOGIN/REGISTRATION MODULE

- This module facilitates user authentication and registration processes.
- Users can securely create new accounts or log in with existing credentials.

- Authentication mechanisms such as username/password authentication or OAuth integration may be implemented.
- Registration forms capture essential user details and may include validation checks to ensure data integrity.
- User passwords are securely hashed and stored in the database to protect user privacy.
- Error handling mechanisms are implemented to provide informative feedback to users in case of login failures or registration errors.

SEARCH DONOR MODULE

- User registration allows new donors to register by providing personal details (name, age, contact information), health information, and blood type.
- Profile management enables registered donors to update their profiles, including personal details and donation preferences.
- It tracks each donor's donation history, including dates, locations, and blood type.

ABOUT BLOOD MODULE

- Request processing ensures compatibility by verifying donor blood types with requests from hospitals or patients.
- And it updates blood inventory in real-time after a donation, ensuring accurate tracking of blood availability.

PROFILE UPDATE MODULE

Donors can update personal details such as name, contact information (phone number, email), address, and emergency contact.

- Provides a summary of past donations, including dates, locations, and blood type, which donors can review to track their donation history.
- Users can change or reset their passwords for account security.
- Users can set their preferences for receiving notifications via email, SMS, or app alerts about upcoming donation events, reminders, and critical blood shortages.

COMMUNITY POST MODULE

- This facilitates communication between blood banks, donors, healthcare organizations, and the wider community. This module can help spread awareness, encourage donations, and share important updates about blood donation drives, emergency needs, and other related activities. Provides a summary of past donations, including dates, locations, and blood type, which donors can review to track their donation history.

CONTACT US MODULE

- The Contact Us module provides users with a means to communicate with platform administrators or support staff for inquiries, assistance, or feedback.

- Users can fill out a contact form with their name, email address, subject, and message detailing their query or concern.

- Captcha or anti-spam measures may be implemented to prevent misuse of the contact form.

- Upon form submission, the system sends an email notification to designated administrators or support personnel, who can then respond to the user's inquiry via email or through the platform.

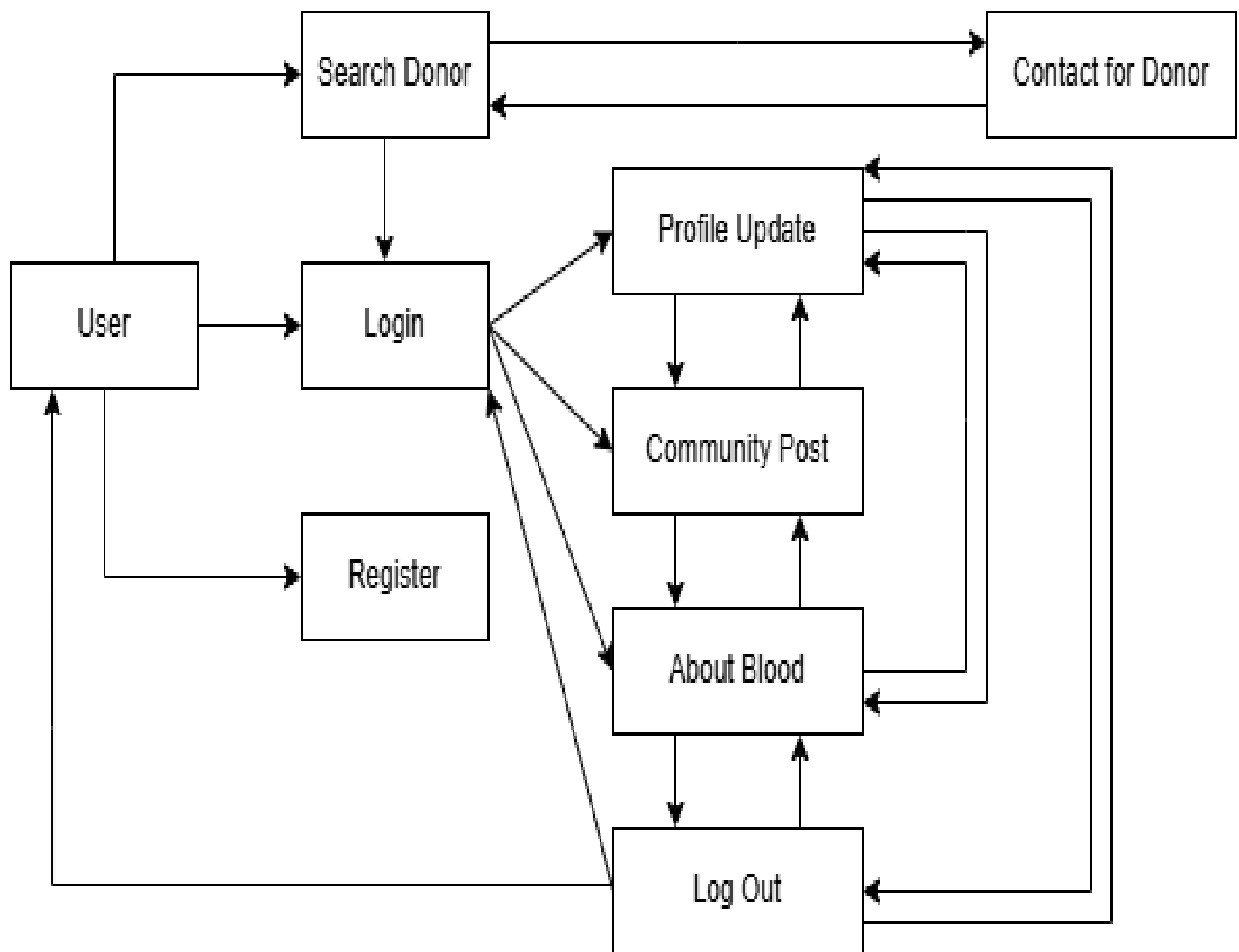
EMAILING FEATURE MODULE

- The Emailing Feature module enables the system to send automated email notifications to users for various events and actions.

- Notifications may include booking confirmations, payment receipts, booking reminders, and review requests.

- Email templates are designed to provide clear and informative messages, personalized with relevant user details and booking information.


BLOCK DIAGRAM:-



OUTPUT SNAPSHOT

Index Page


[bloodbank.stayhealthy@gmail.com](#) [+91 8260692236](#) [Registration](#)

 [Home](#) [About Blood](#) [Search Donor](#) [Login](#)

EVERYWHERE FOR EVERYONE


GIVE BLOOD GIVE LIFE

One blood donation can benefit many people.
A single donation can help at least three people in need.
It also makes the work of blood banks easier.



About Blood

[bloodbank.stayhealthy@gmail.com](#) [+91 8260692236](#) [Registration](#)

 [Home](#) [About Blood](#) [Search Donor](#) [Login](#)

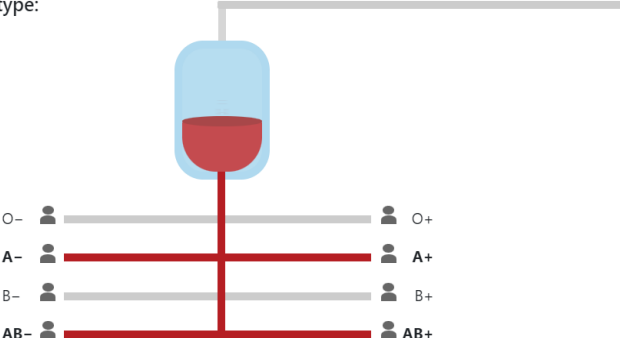
QUICK LINKS

- [Blood Types and Transfusion](#)
- [Lives are saved by blood transfusions](#)
- [A CRUCIAL NEED](#)
- [All child Parents Pairings](#)
- [Live Blood Campaign](#)

Blood Types and Transfusion

Select the donor blood type:

<input type="radio"/> O-	<input type="radio"/> O+	<input type="radio"/> A-	<input type="radio"/> A+
<input type="radio"/> B-	<input type="radio"/> B+	<input type="radio"/> AB-	<input type="radio"/> AB+



Each year 4.5 million lives are saved by blood transfusions.

Search Doner

Search Donors

Blood Group

Select

Location


Select City

Date of Blood Donation

dd-mm-yyyy

Search

Donor's List




Name: debasis

Phone: 06371599342

Blood Group: O+


Community Post


Create a new post


 debasis


Nov. 7, 2024, 5:35 p.m.

20 mins

 O+

 cuttack

 1 bag

 6371599342

2 comments Share

 Daniel Frozer

I like this alot! thanks alot

LikeReplyTranslate18 mins



Registration Form

New Account

Give blood, Take blood & Save life

<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>

** All data keeps secret except contacts info thats are not problem at all.

Login Form

Sign into your account

Give blood, Take blood & Save life

Login

[Forgot password?](#)

Don't have an account? [Register here](#)

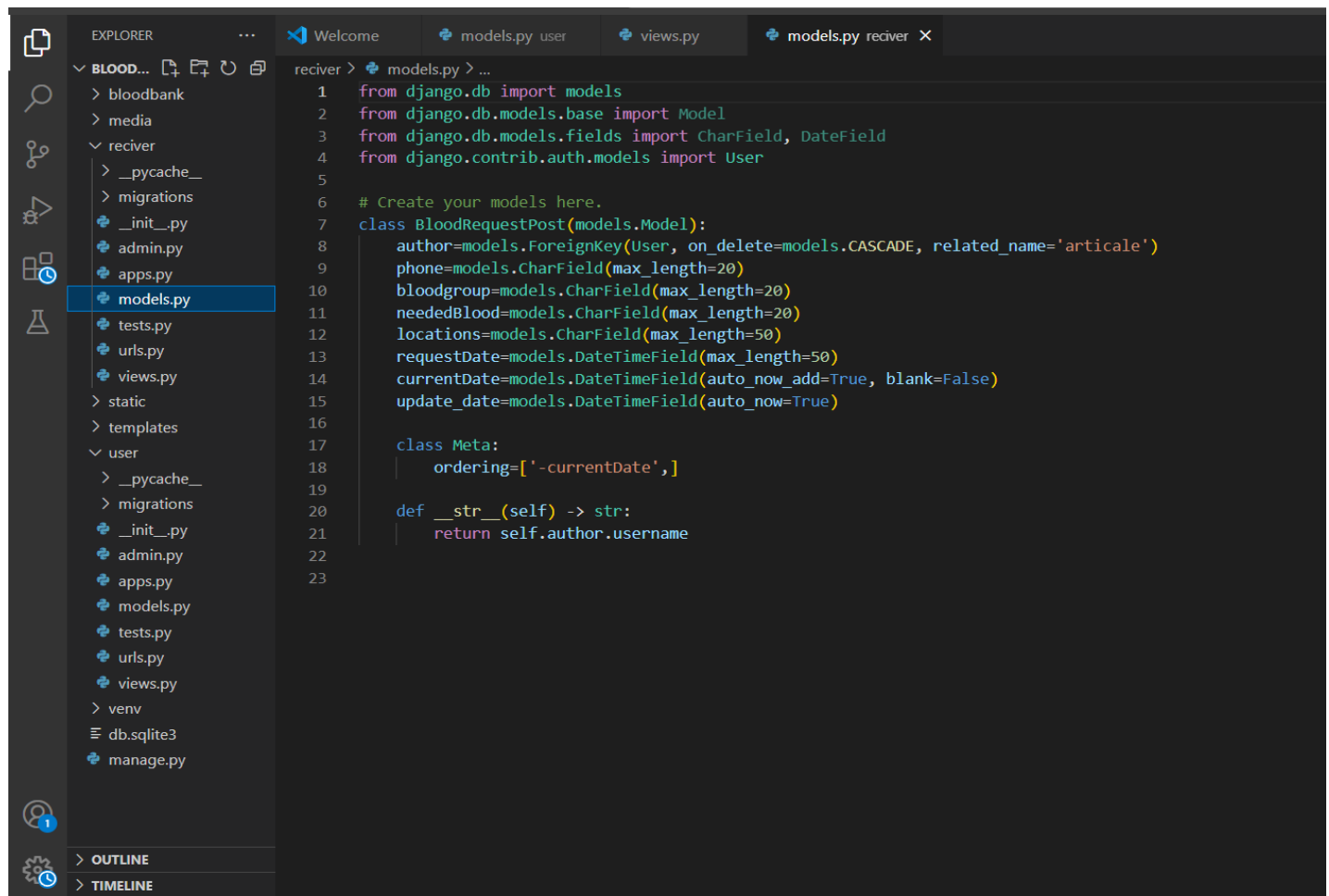
Emergency Call

Email For Contact

Contact us Social Media

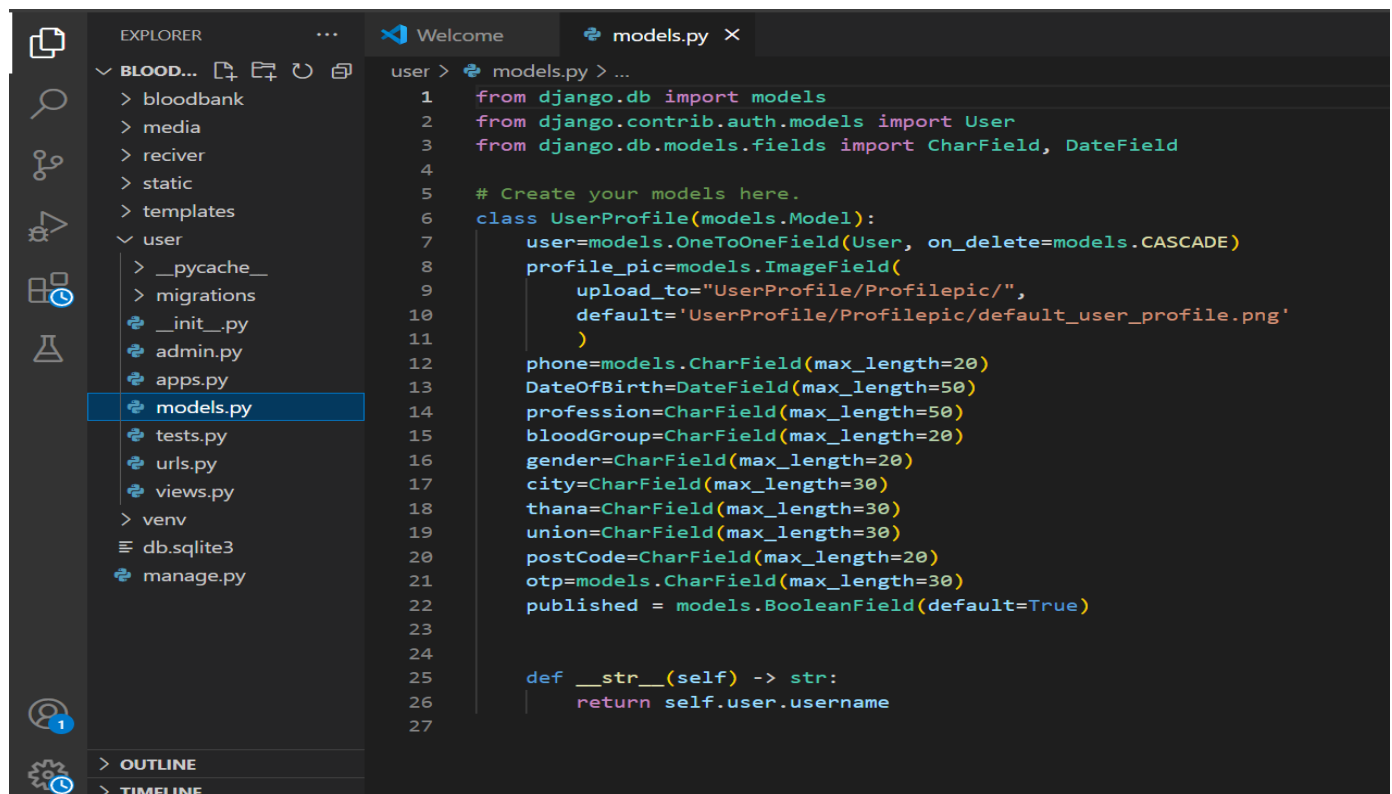
Here is Some Implementation Code Snippet:

Models.py



This screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows a project structure with folders like 'bloodbank', 'media', and 'reciver'. The 'reciver' folder is expanded, showing files like '__pycache__', 'migrations', '__init__.py', 'admin.py', 'apps.py', 'models.py', 'tests.py', 'urls.py', and 'views.py'. The 'models.py' file is selected and its content is displayed in the Editor. The code defines a Django model named 'BloodRequestPost' with various fields and a Meta class.

```
1 from django.db import models
2 from django.db.models.base import Model
3 from django.db.models.fields import CharField, DateField
4 from django.contrib.auth.models import User
5
6 # Create your models here.
7 class BloodRequestPost(models.Model):
8     author=models.ForeignKey(User, on_delete=models.CASCADE, related_name='artical')
9     phone=models.CharField(max_length=20)
10    bloodgroup=models.CharField(max_length=20)
11    neededBlood=models.CharField(max_length=20)
12    locations=models.CharField(max_length=50)
13    requestDate=models.DateTimeField(max_length=50)
14    currentDate=models.DateTimeField(auto_now_add=True, blank=False)
15    update_date=models.DateTimeField(auto_now=True)
16
17    class Meta:
18        ordering=['-currentDate',]
19
20    def __str__(self) -> str:
21        return self.author.username
22
23
```



This screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows a project structure with folders like 'bloodbank', 'media', 'reciver', 'static', 'templates', and 'user'. The 'user' folder is expanded, showing files like '__pycache__', 'migrations', '__init__.py', 'admin.py', 'apps.py', 'models.py', 'tests.py', 'urls.py', and 'views.py'. The 'models.py' file is selected and its content is displayed in the Editor. The code defines a Django model named 'UserProfile' with various fields and a Meta class.

```
1 from django.db import models
2 from django.contrib.auth.models import User
3 from django.db.models.fields import CharField, DateField
4
5 # Create your models here.
6 class UserProfile(models.Model):
7     user=models.OneToOneField(User, on_delete=models.CASCADE)
8     profile_pic=models.ImageField(
9         upload_to="UserProfile/Profilepic/",
10         default='UserProfile/Profilepic/default_user_profile.png'
11     )
12     phone=models.CharField(max_length=20)
13     DateOfBirth=DateField(max_length=50)
14     profession=CharField(max_length=50)
15     bloodGroup=CharField(max_length=20)
16     gender=CharField(max_length=20)
17     city=CharField(max_length=30)
18     thana=CharField(max_length=30)
19     union=CharField(max_length=30)
20     postCode=CharField(max_length=20)
21     otp=models.CharField(max_length=30)
22     published = models.BooleanField(default=True)
23
24
25    def __str__(self) -> str:
26        return self.user.username
27
```

Views.py

The image shows two screenshots of a VS Code editor working on a Django project named 'bloodbank'.

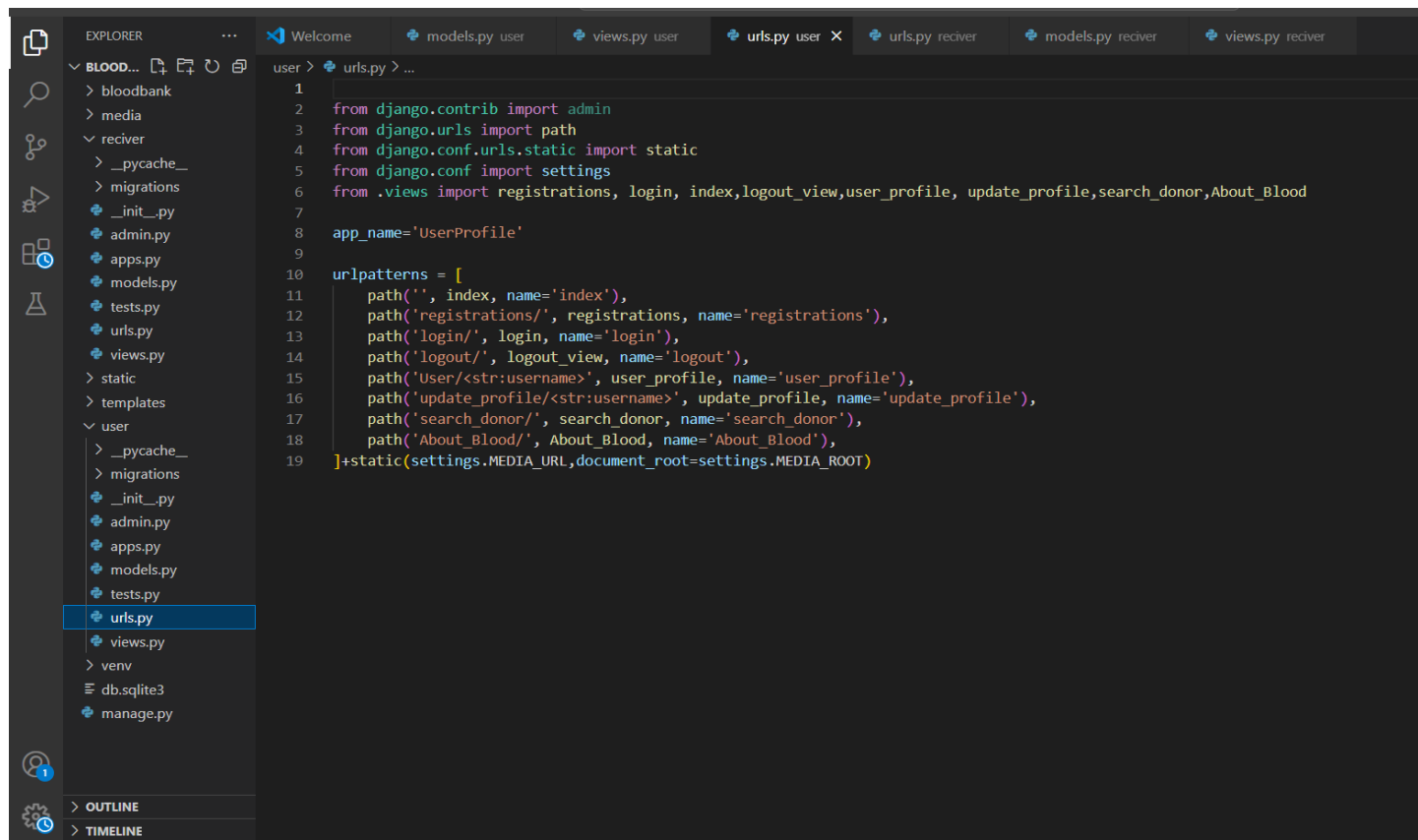
Top Screenshot: The Explorer sidebar shows the project structure with folders 'bloodbank', 'media', 'reciver', 'static', 'templates', and 'user'. The 'views.py' file in the 'user' folder is selected. The main editor shows the content of 'views.py' with the following code:

```
1 from django.contrib.auth.models import User
2 from django.http import HttpResponse
3 from django.shortcuts import redirect, render
4 from django.urls import reverse_lazy
5 from .models import BloodRequestPost
6
7 # Create your views here.
8 def articals_index(request):
9     post=BloodRequestPost.objects.all()
10    context={"allPost":post}
11    return render(request, 'articals/communityPost.html',context)
12
13
14 def request_blood_post(request):
15     if request.user.is_authenticated:
16         current_user=request.user
17
18     if request.method=='POST':
19         bloodGroup=request.POST.get('bloodGroup')
20         quantity=request.POST.get('quantity')
21         location=request.POST.get('location')
22         date=request.POST.get('date')
23         phone=request.POST.get('phone')
24
25         bloodPost=BloodRequestPost(
26             author=current_user,
27             phone=phone,
28             bloodgroup=bloodGroup,
29             locations=location,
30             neededBlood=quantity,
31             requestDate=date,
32         )
33         bloodPost.save()
34         return redirect(reverse_lazy('articals:articals_index'))
35
36
```

Bottom Screenshot: The Explorer sidebar shows the project structure with folders 'bloodbank', 'media', 'reciver', 'static', 'templates', and 'user'. The 'views.py' file in the 'user' folder is selected. The main editor shows the content of 'views.py' with the following code:

```
62 return compatibility.get(blood_group, [])
63
64 def registrations(request):
65     if request.method == 'POST':
66         fname = request.POST.get('fname')
67         lname = request.POST.get('lname')
68         username = request.POST.get('username')
69         phone = request.POST.get('phone')
70         email = request.POST.get('email')
71         Password = request.POST.get('password')
72         Password2 = request.POST.get('password2')
73         DateOfBirth = request.POST.get('DateOfBirth')
74         profession = request.POST.get('profession')
75         bloodGroup = request.POST.get('bloodGroup')
76         gender = request.POST.get('gender')
77         city = request.POST.get('city')
78         thana = request.POST.get('thana')
79         union = request.POST.get('union')
80         postCode = request.POST.get('postCode')
81
82         check_user = User.objects.filter(email=email).first()
83         check_profile = UserProfile.objects.filter(phone=phone).first()
84
85         if Password != Password2:
86             context1 = {"message1": "Password mismatch", "class1": "danger"}
87             return render(request, 'UserProfile/registration.html', context1)
88
89         if check_user or check_profile:
90             context = {"message": "User already exists", "class": "danger"}
91             return render(request, 'UserProfile/registration.html', context)
92
93         user = User.objects.create_user(
94             first_name=fname, last_name=lname, username=username, email=email, password=Password
95         )
96         user.save()
97
98         profile = UserProfile(
```

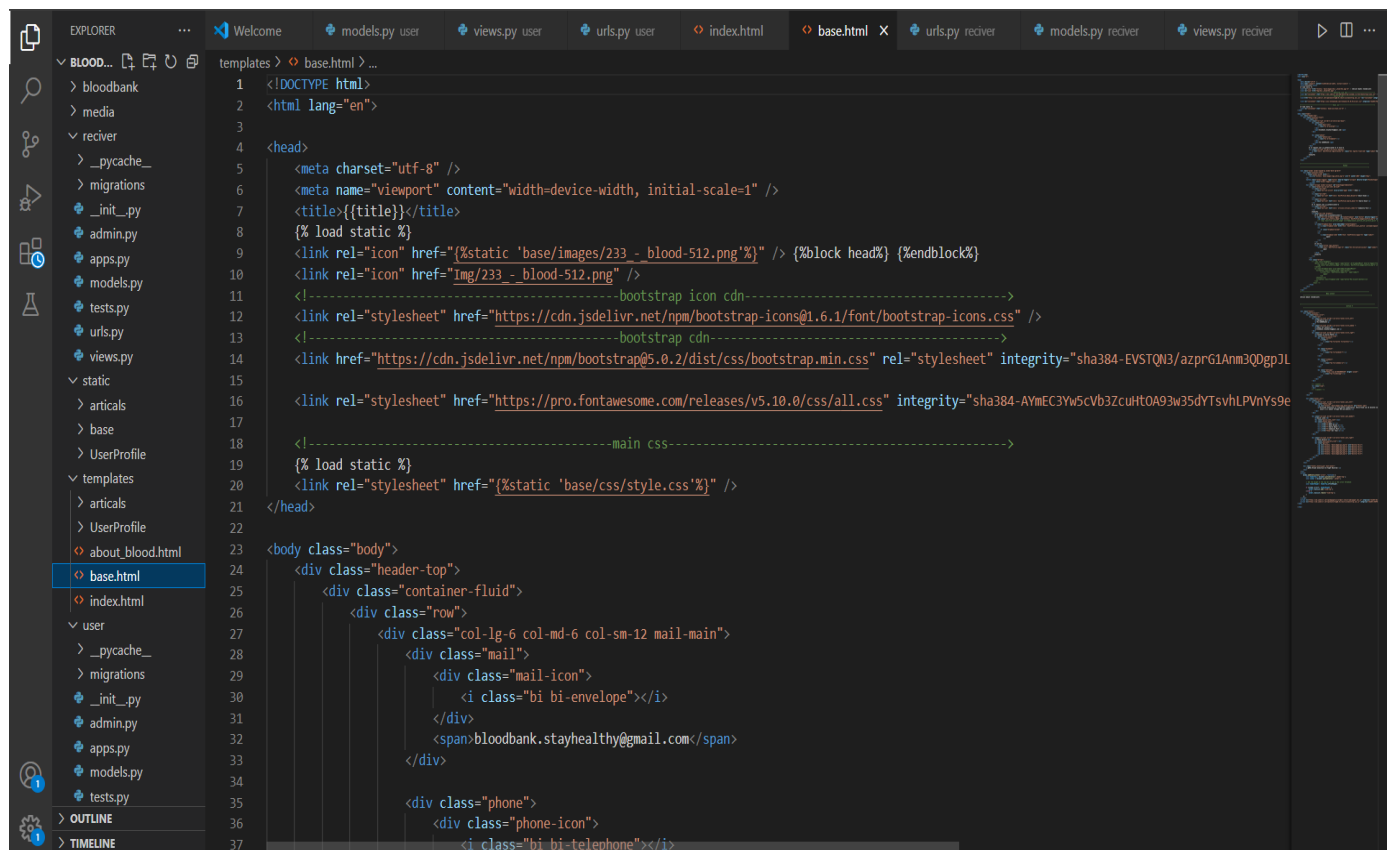

URLS.py



The screenshot shows the Visual Studio Code editor with a Django project open. The Explorer sidebar on the left shows the project structure, including folders like 'bloodbank', 'media', 'reciver', 'static', 'templates', and 'user'. The 'urls.py' file is selected under the 'user' folder. The main editor displays the content of 'urls.py', which imports necessary Django modules and defines URL patterns for the application.

```
1
2 from django.contrib import admin
3 from django.urls import path
4 from django.conf.urls.static import static
5 from django.conf import settings
6 from .views import registrations, login, index, logout_view, user_profile, update_profile, search_donor, About_Blood
7
8 app_name = 'UserProfile'
9
10 urlpatterns = [
11     path('', index, name='index'),
12     path('registrations/', registrations, name='registrations'),
13     path('login/', login, name='login'),
14     path('logout/', logout_view, name='logout'),
15     path('User/<str:username>', user_profile, name='user_profile'),
16     path('update_profile/<str:username>', update_profile, name='update_profile'),
17     path('search_donor/', search_donor, name='search_donor'),
18     path('About_Blood/', About_Blood, name='About_Blood'),
19 ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

HTML Templates:



The screenshot shows the Visual Studio Code editor with the same Django project open. The Explorer sidebar shows the 'templates' folder selected. The 'base.html' file is selected under the 'templates' folder. The main editor displays the content of 'base.html', which is a Django template that defines the base structure of the application's HTML pages, including the head, body, and various CSS and JS links.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="utf-8" />
6     <meta name="viewport" content="width=device-width, initial-scale=1" />
7     <title>{{title}}</title>
8     {% load static %}
9     <link rel="icon" href="{%static 'base/images/233 - blood-512.png%}" /> {%block head%} {%endblock%}
10    <link rel="icon" href="img/233 - blood-512.png" />
11    <!-------bootstrap icon cdn----->
12    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.6.1/font/bootstrap-icons.css" />
13    <!-------bootstrap cdn----->
14    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgp1L"
15
16    <link rel="stylesheet" href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css" integrity="sha384-AYmEC3YwScVb3zcuHtOA93w35dYTsVhLPVnYs9e
17
18    <!-------main css----->
19    {% load static %}
20    <link rel="stylesheet" href="{%static 'base/css/style.css%}" />
21 </head>
22
23 <body class="body">
24     <div class="header-top">
25         <div class="container-fluid">
26             <div class="row">
27                 <div class="col-lg-6 col-md-6 col-sm-12 mail-main">
28                     <div class="mail">
29                         <div class="mail-icon">
30                             <i class="bi bi-envelope"></i>
31                         </div>
32                         <span>bloodbank.stayhealthy@gmail.com</span>
33                     </div>
34
35                     <div class="phone">
36                         <div class="phone-icon">
37                             <i class="bi bi-telephone"></i>
```

```
EXPLORER Welcome models.py user views.py user urls.py user index.html base.html about_blood.html communityPost.html X urls.py reciver
```

```
templates > articles > communityPost.html > ...
```

```
5 {%extends 'base.html'%} {%block head%} {% load static %}
6 <!-- Main css -->
7 <link rel="stylesheet" href="{%static 'UserProfile/css/search.css'%}" />
8 <link rel="stylesheet" href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css" integrity="sha384-AymEC3Yw5cVb3ZcuHT0A93w35dYTsVhLPVnYs9eStHf" />
9
10 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.7.2/font/bootstrap-icons.css">
11 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-alpha1/dist/css/bootstrap.min.css">
12 {%endblock%} {%block body%}
13 <!-- The Modal -->
14 <div class="modal fade" id="myModal">
15   <div class="modal-dialog">
16     <div class="modal-content">
17       <!-- Modal Header -->
18       <div class="modal-header">
19         <h4 class="modal-title">Create a new post</h4>
20         <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
21       </div>
22
23       <!-- Modal body -->
24       <div class="modal-body">
25         <div class="registration_form">
26           <form action="{%url 'articles:request_blood_post'%}" method="post" class="row g-3">
27             {% csrf token %}
28             <div class="col-md-6">
29               <select name="bloodGroup" class="form-select" aria-label="Default select example" required>
30                 <option value="">Select Blood Group</option>
31                 <option value="A+">A+</option>
32                 <option value="AB+">AB+</option>
33                 <option value="A-">A-</option>
34                 <option value="AB-">AB-</option>
35                 <option value="B+">B+</option>
36                 <option value="B-">B-</option>
37                 <option value="O+">O+</option>
38                 <option value="O-">O-</option>
39               </select>
40             </div>
41             <div class="col-md-6">
```

```
File Edit Selection View Go Run Terminal Help bloodbank
```

```
EXPLORER views.py user urls.py user index.html base.html about_blood.html communityPost.html login.html Profile.html registration.html X
```

```
templates > UserProfile > registration.html > ...
```

```
1 {%extends 'base.html'%} {%block head%} {% load static %}
2 <link rel="stylesheet" href="{%static 'UserProfile/css/style.css'%}" />
3 {%endblock%} {%block body%}
4 <div class="registration_form">
5   <h3>New Account</h3>
6   <h5 class="pb-3">Give blood, Take blood & Save life</h5>
7   <form action="{%url 'UserProfile:registrations'%}" method="post" class="row g-3">
8     {% csrf token %}
9     <div class="col-lg-12 col-md-6 col-sm-6">
10       <center><span class="text-{{class}}">{{message}}</span></center>
11     </div>
12     <div class="col-md-6">
13       <input type="text" name="fname" placeholder="First Name" class="form-control" id="inputfname" autofocus/>
14     </div>
15     <div class="col-md-6">
16       <input type="text" name="lname" placeholder="Last Name" class="form-control" id="inputlname" />
17     </div>
18     <div class="col-md-6">
19       <input type="text" name="username" placeholder="User Name" class="form-control" id="inputfname" required/>
20     </div>
21     <div class="col-md-6">
22       <input type="number" name="phone" placeholder="Phone" class="form-control" id="inputPhone" />
23     </div>
24     <div class="col-md-6">
25       <input type="email" name="email" placeholder="Email" class="form-control" id="inputEmail4" />
26     </div>
27     <div class="col-md-6">
28       <input type="password" name="password" placeholder="Password" class="form-control" id="inputEmail4" />
29     </div>
30     <div class="col-md-6">
31       <input
32         type="password"
33         name="password2"
34         placeholder="Confirm Password"
35         class="form-control"
36         id="inputPassword4"
37       />
38     </div>
39   </form>
40 </div>
```

Conclusion:

The blood bank management system outlined by the data flow diagram aims to streamline the processes involved in donor and recipient interactions, enhancing the availability and accessibility of blood donations. The system allows users to search for blood donors, update their profiles, and make community posts. It also provides information about blood types and donation criteria, helping users understand and engage with the donation process better. By integrating these functionalities, the system improves the efficiency of locating donors and communicating within the community, ultimately contributing to a more responsive and organized blood donation network.

Future Work:

- **Real-time Donor Availability:** Implementing real-time status updates for donor availability would allow the system to provide accurate information to users seeking donors.
- **Notifications and Alerts:** Adding notifications for donation opportunities, blood shortages, and reminders for eligible donors to donate would enhance engagement and responsiveness.
- **Geolocation Integration:** Using geolocation services to help users find nearby donors could make the search process quicker and more relevant.
- **Enhanced Security and Privacy:** Strengthening security measures to protect user data, including contact information, to ensure privacy and trust in the system.
- **Data Analytics for Predictive Insights:** Integrating analytics to predict demand based on historical data could help manage inventory and improve preparedness for emergencies.
- **Mobile Application Development:** Developing a mobile app version of the system would increase accessibility, especially in urgent situations, allowing users to access features on-the-go.
- **Multi-language Support:** Adding support for multiple languages would make the system more inclusive, catering to a diverse user base.
- **Integration with Health Systems:** Linking with hospitals and health management systems could improve coordination and ensure timely blood supply to patients in need.



Thank
You!

From:

Nishika Mishra

Pratik Swain

Preetam Kumar Patra

Partha Sarathi Sahoo