

PROJECT REPORT

PLASMA DONOR APPLICATION

TEAM ID: PNT2022TMID16194

Submitted by,

TEAM LEADER	HARIPRIYA M	(927619BIT4034)
TEAM MEMBER 1	NANDHINI C	(927619BIT4062)
TEAM MEMBER 2	MADHUMITHA G	(927619BIT4054)
TEAM MEMBER 3	VIJAYA PRATHA E	(927619BIT4120)
TEAM MEMBER 4	RENUKA P	(927619BIT4082)

TABLE OF CONTENT

1. INTRODUCTION

1.1 Project Overview

1.2 Purpose

2. LITERATURE SURVEY

2.1 Existing problem

2.2 References

2.3 Problem Statement Definition

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

3.2 Ideation & Brainstorming

3.3 Proposed Solution

3.4 Problem Solution fit

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

4.2 Non-Functional requirements

5. PROJECT DESIGN

5.1 Data Flow Diagrams

5.2 Solution & Technical Architecture

5.3 User Stories

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

6.2 Sprint Delivery Schedule

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1 Feature 1

7.2 Feature 2

7.3 Database Schema (if Applicable)

8. TESTING

8.1 Test Cases

8.2 User Acceptance Testing

9. RESULTS

9.1 Performance Metrics

10. ADVANTAGES & DISADVANTAGES

11. CONCLUSION

12. FUTURE SCOPE

13. APPENDIX

Source Code

GitHub & Project Demo Link

PLASMA DONOR APPLICATION

ABSTRACT

During the COVID 19 crisis, the requirement of plasma became a high priority and the donor count has become low. Saving the donor information and helping the needy by notifying the current donors list, would be a helping hand. In regard to the problem faced, an application is to be built which would take the donor details, store them and inform them upon a request. This plasma therapy is considered to be safe & promising. This system proposed here aims at connecting the donors & the patients by an online application. By using this application, the users can either raise a request for plasma donation or requirement. This system is used if anyone needs a Plasma Donor.

1. INTRODUCTION

1.1 PROJECT OVERVIEW

Recently concern grows about the plasma donation for COVID-19 during the pandemic situation. This convalescent plasma was used to recover patients who are critically ill as it helps to grow antibodies on their body. Recent researches show that many people are willing to help someone in need through money, blood and plasma donation etc. but they find it difficult to identify and approach the needy people who are not aware of technological innovations, including the use of social media. Plasma is used to various infectious diseases and it is one of the oldest methods known as plasma therapy. Plasma therapy is a Process where blood is donated by recovered patients in order to establish anti bodies that fights the infection. This system comprises of Admin, user and donor where both can request for Plasma. The proposed method helps the users to check the availability of donors. A donor has to register to the website providing their details. The registered users can get the information about the donor count of each blood group. The database will have all the details such as name, email, phone number, infected status. Whenever a user requests for a particular blood group then the concerned blood group donors will receive the notification regarding the requirement. For instance, during COVID 19 crisis the requirement for plasma increased drastically as there were no vaccination found in order to treat the infected patients, with plasma therapy the recovery rates were high but the donor count was very low and in such situations it was very important to get the information about the plasma donors. Saving the donor information and notifying about the current donors would be a helping hand as it can save time and help the users to track down the necessary information about the donors.

1.2 PURPOSE

The main aim of developing this system is to provide blood to the people who are in need of plasma. The numbers of persons who are in need of plasma are increasing in large number day by day. Using this system user can search blood group available in the city and he can also get contact number of the donor who has the same blood group he/she needs for plasma. In order to help people who are in need of plasma, this plasma donor application can be used effectively for getting the details of available plasma and user can also get contact number of the plasma donors having the same blood group and within the same city.

2. LITERATURE REVIEW

2.1 EXISTING PROBLEM

2.1.1 TITLE : Instant Plasma Donor Recipient Connector web application

AUTHOR: Kalpana Devi Guntoju*1, Tejaswini Jalli*2

The world is suffering from the COVID 19 crisis and no vaccine has been found yet.. But there is another scientific way in which we can help reduce mortality or help people affected by COVID19 by donating plasma from recovered patients. In the absence of an approved antiviral treatment plan for a fatal COVID19 infection, plasma therapy is an experimental approach to treat COVID19-positive patients and help them faster recovery. Therapy is considered competent. In the recommendation system, the donor who wants to donate plasma can donate by uploading their COVID19 certificate and the blood bank can see the donors who have uploaded the certificate and they can make a request to the donor and the hospital can register/login and search for the necessary things. plasma from a blood bank and they can request a blood bank and obtain plasma from the blood bank. The main goal of our project is to design a user-friendly web application that is like a scientific vehicle from which we can help reduce mortality or help those affected by COVID19 by donating plasma from patients who have recovered without approved antiretroviral therapy planning for a deadly COVID19 infection, plasma therapy is an experimental approach to treat those COVID-positive patients and help them recover faster. Therapy, which is considered reliable and safe. If a particular person has fully recovered from COVID19, they are eligible to donate their plasma. As we all know, the traditional methods of finding plasma, one has to find out for oneself by looking at hospital records and contacting donors have been recovered, sometimes may not be available at home and move to other places. In this type of scenario, the health of those who are sick becomes disastrous. Therefore, it is not considered a rapid process to find plasma.

2.1.2 TITLE : A Web Application to Manage All Blood Donation and Transfusion Processes

AUTHOR: Rehab S. Ali,¹ Tamer F. Hafez,² Ali Badawey Ali

Many lives could be lost due to the difficulty in obtaining a proper blood bag, Therefore, this work aims to help citizens fulfill their needs for a safe and reliable blood group by searching for and locating a specific blood group. In this paper, we illustrate the problem of the blood bags shortage which is represented in the uncontrolled blood banks and parallel markets, lack of awareness and confidence, disappearance of the rare blood groups, and the difficulty in finding a specific blood group. Hence, we proposed the Blood Bag web-based application that is connected to a centralized database to gather and organize the data from all blood banks and blood donation campaigns. The proposed application organizes and controls the whole critical processes related to blood donation, testing and storage of blood bags, and delivering it to the patient. One blood bag can save a life during surgeries or road accidents, etc. Usually patients or their families look for a specific blood group they indeed need in the blood banks but they normally cannot find it due to the shortage of blood bags. This is because of the fear of donating blood and the misconception that donating blood is harmful and transmits diseases. This is one of the obstacles to provide the blood bags. The availability of the blood bags is critical because of the high proportion of patients with renal failure, some cases of birth, surgeries processes and incidents that need to get the blood as soon as possible to save these cases' lives. The blood bank is the pool of different blood groups where keeping a stockpile of blood to be distributed in case. The matched blood groups for a safe transfusion . Accidents (or any medical emergency and compensation of blood missing from the body), and keeping the blood in the freezer temperature.

2.1.3 TITLE : Developing a plasma donor application using Function-as-a-service in AWS

AUTHOR: Aishwarya R Gowri

Plasma is a liquid portion of the blood, over 55% of human blood is plasma. Plasma is used to treat various infectious diseases and it is one of the oldest methods known as plasma therapy. Plasma therapy is a process where blood is donated by recovered patients in order to establish an antibody that fights the infection. In this project plasma donor application is being developed by using AWS services. The services used are AWS Lambda, API gateway, Dynamo DB, AWS Elastic Compute Cloud with the help of these AWS services, it eliminates the need of configuring the servers and reduces the infrastructural costs associated with it and helps to achieve serverless computing. For instance, during COVID 19 crisis the requirement for plasma increased drastically as there were no vaccination found in order to treat the infected patients, with plasma therapy the recovery rates were high but the donor count was very low and in such situations it was very important to get the information about the plasma donors. Saving the donor information and notifying about the current donors would be a helping hand as it can save time and help the users to track down the necessary information about the donors. The proposed method helps the users to check the availability of donors. A donor has to register to the website providing their details. The registered users can get the information about the donor count of each blood group. The database will have all the details such as name, email, phone number, infected status. Whenever a user requests for a particular blood group then the concerned blood group donors will receive the notification regarding the requirement. A Json code is written to store the information, to fetch the requested information in lambda.

2.1.4 TITLE : Nearest Blood & Plasma Donor Finding: A Machine Learning Approach

AUTHOR: Nayan Das, Asif Iqbal.

The necessity of blood has become a significant concern in the present context all over the world. Due to a shortage of blood, people couldn't save themselves or their friends and family members. A bag of blood can save a precious life. Statistics show that a tremendous amount of blood is needed yearly because of major operations, road accidents, blood disorders, including Anemia, Hemophilia, and acute viral infections like Dengue, etc. Approximately 85 million people require single or multiple blood transfusions for treatment. Voluntary blood donors per 1,000 population of some countries are quite promising, such as Switzerland (113/1,000), Japan (70/1,000), while others have an unsatisfying result like India has 4/1,000, and Bangladesh has 5/1000. Recently a lifethreatening virus, COVID-19, spreading throughout the globe, which is more vulnerable for older people and those with pre-existing medical conditions. For them, plasma is needed to recover their illness. Our Purpose is to build a platform with clustering algorithms which will jointly help to provide the quickest solution to find blood or plasma donor. Closest blood or plasma donors of the same group in a particular area can be explored within less time and more efficiently. Keywords—Blood donation, Plasma donation, Kmeans clustering, Labeled Agglomerative clustering. Different methods have been used to solve this problem. This time, we have tried another way, a clustering approach, to solve the problem by grouping every user into small groups. This unsupervised machine learning approach is much faster and effective. In section II, we will discuss related work done previously to solve this problem. In section III, clustering algorithms relating to our project will explicitly be discussed. In section IV, our proposed method will be presented. In section V, we will analyze our experiment result.

2.1.5 TITLE : Securing Information on a Web Application System to Facilitate Online Blood Donation Booking

AUTHOR: Hrishitva Patel

Blood donation has saved many lives in the past. According to the American Red Cross statistics, a patient needs a blood transfusion every two seconds. Many benefits arise from blood donation to both the donor and the blood recipients. With blood donation, cancer patients, people involved in accidents, or those battling diseases that require blood donation have access to enough blood to sustain their survival. There is a need to digitize the blood donation booking to facilitate blood donation across the United States, and ensure patients in need of blood, receive their donation from eligible donors on time. This report demonstrates the security measures implemented to secure patient and blood donor data on a blood donation booking web application. Blood is donated for different reasons in hospitals and other blood banks. It is essential to help blood recipients survive surgeries, cancer treatment, and chronic illnesses, among other illnesses. The World Health Organization describes blood as the most precious gift a person can give to a person in need of it. Blood donated comprises four components: platelets, plasma, white blood cells, and red blood cells. Cancer patients require a blood transfusion to enhance platelets back into the body after radiation therapy. With a developed web application to book a blood donation session, it is easier for hospitals to know which blood component they need most and thus inform the system administrator to prompt for more blood donors.

2.2 REFERENCES

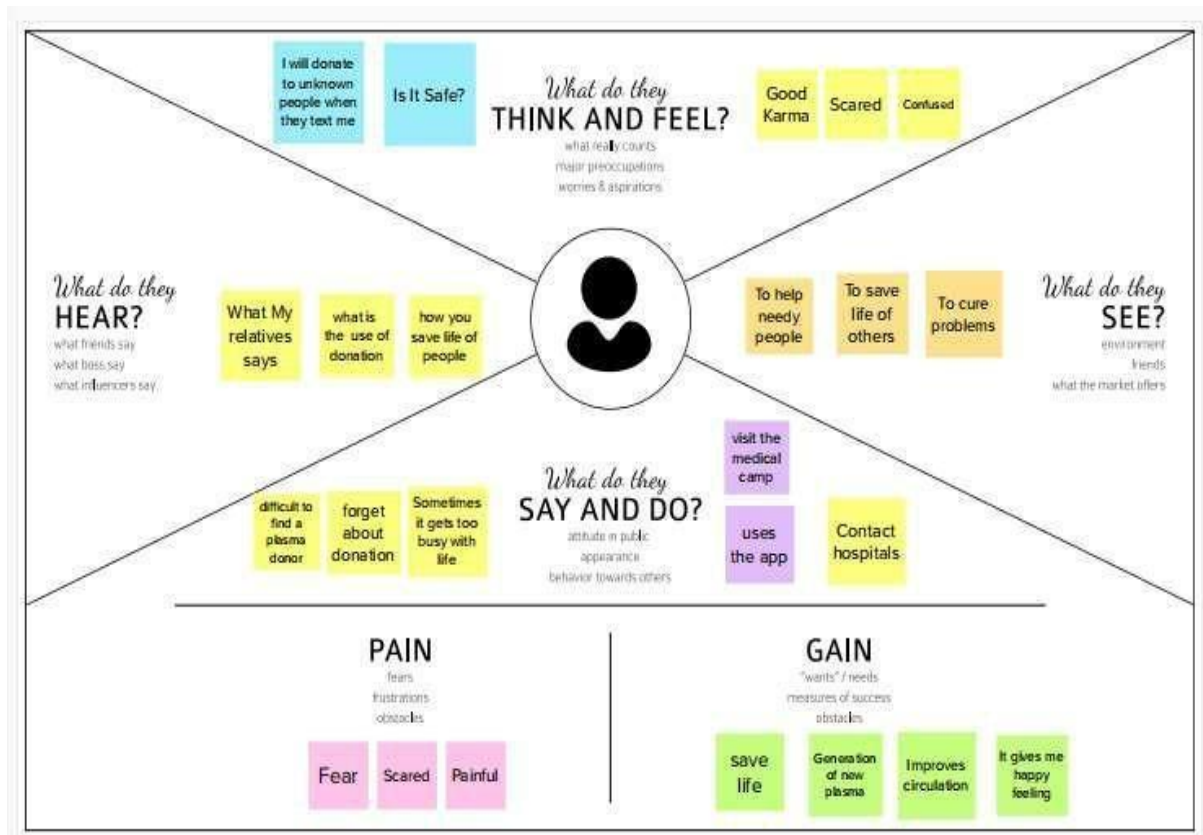
- 1.** Kalpana Devi Guntoju*1, Tejaswini Jalli*2, Instant Plasma Donor Recipient Connector web application, 2022.
- 2.** Rehab S. Ali,1 Tamer F. Hafez,2 Ali Badawey Ali, A Web Application to Manage All Blood Donation and Transfusion Processes, 2017.
- 3.** Aishwarya R Gowri, Developing a plasma donor application using Function-as-a-service in AWS, 2020.
- 4.** Nearest Blood & Plasma Donor Finding: A Machine Learning Approach, 2021
- 5.** Hrishitva Patel, Securing Information on a Web Application System to Facilitate Online Blood Donation Booking, 2022.

2.3 PROBLEM STATEMENT DEFINITION

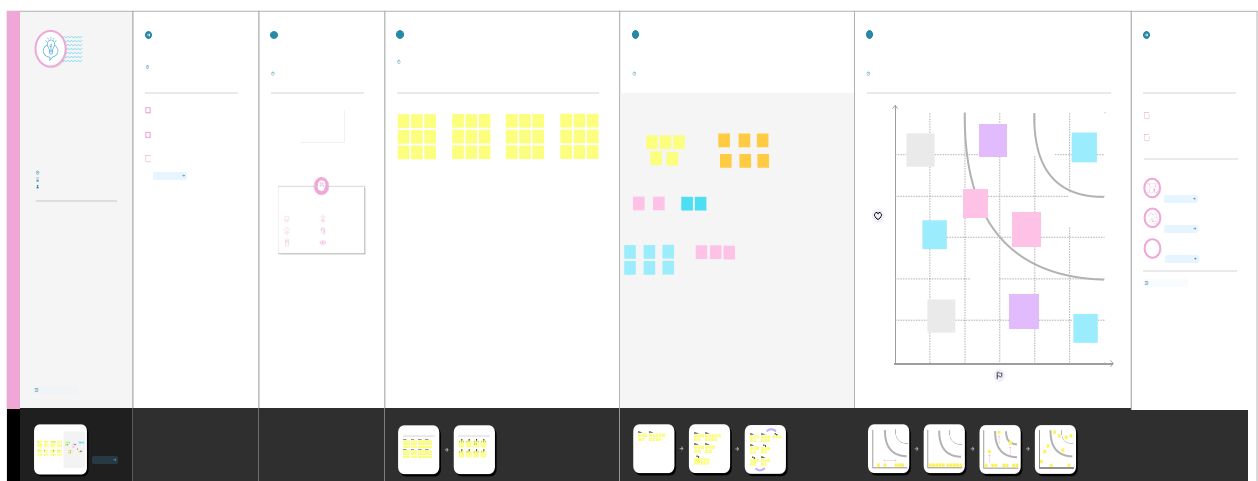
In critical or emergency situations where accident occurs or during on-going treatments and surgeries etc there is urgent need for specific blood group. It requires lot of time to make the blood available and it is inconvenient during emergency situation, some rare blood groups are time consuming and difficult to arrange which are O- , AB- etc. In our country there is less awareness of blood donation, near about 20% of Indian population donates blood. In existing system the blood bank management system exhibited at a lot of ineffectiveness and inefficiency that had fetched impact taken by management. The system which was manual that is based on paper card to collect blood donor data, keep record of blood donors and disseminate results to blood donors, had weakness that needed IT based solutions.

3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS



3.2 IDEATION & BRAINSTORMING



3.3 PROPOSED SOLUTION

The new idea will improve the existing system and it will move from conventional desktop system to mobile system. This paper introduces new features of improved system over existing system in many aspects. The proposed plasma donor application helps the people who are in need of plasma by giving them all details of plasma availability or regarding the donors with the same blood group. This is a web application allows you to access the whole information about plasma donor application, readily scalable and adaptable to meet the complex need of plasma Who are Key Facilitator for the Healthcare Sector, it also supports all the functionalities of plasma donor application.

3.4 PROBLEM SOLUTION FIT

In the emergency condition, sometimes it becomes very much difficult to look for the exact match of blood group of donor and acceptor. It may lead to delay in transaction of plasma within the specified amount of time. This application is providing each entity the facility to approach nearby blood donors so that it will become much easier to search rare blood groups in the hour of need.

4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

➤ **Admin**

Admin can manage both donors and users. Admin has the only responsibility maintain and stored the record.

➤ **Users**

From this module user can create their account, when user create his account the user get a user id and password which identifies him uniquely. From this module user can search donor for blood.

➤ **Donors Registration**

In this module, people who are interested in donating blood get registered in this site and give his overall details related to donor. User details contain name, address, city, gender, blood group, location, contact number etc.,

➤ **Donor Search**

The people who are in need of blood can search in our site for getting the details of donors having the same blood group and within the same city.

➤ **Notification**

In this module, notification sends to donors for emergency. SMS send to registered donors phone number.

4.2 NON FUNCTIONAL REQUIREMENTS

Usability

The system shall allow the users to access the system with pc using web application. The system uses a web application as an interface. The system is user friendly which makes the system easy

Availability

The system is available 100% for the user and is used 24 hrs a day and 365 days a year. The system shall be operational 24 hours a day and 7 days a week.

Scalability

Scalability is the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands.

Security

A security requirement is a statement of needed security functionality that ensures one of many different security properties of software is being satisfied.

Performance

The information is refreshed depending upon whether some updates have occurred or not in the application. The system shall respond to the member in not less than two seconds from the time of the request submittal. The system shall be allowed to take more time when doing large processing jobs. Responses to view information shall take no longer than 5 seconds to appear on the screen.

Reliability





The system has to be 100% reliable due to the importance of data and the damages that can be caused by incorrect or incomplete data. The system will run 7 days a week. 24 hours a day.

5. PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS

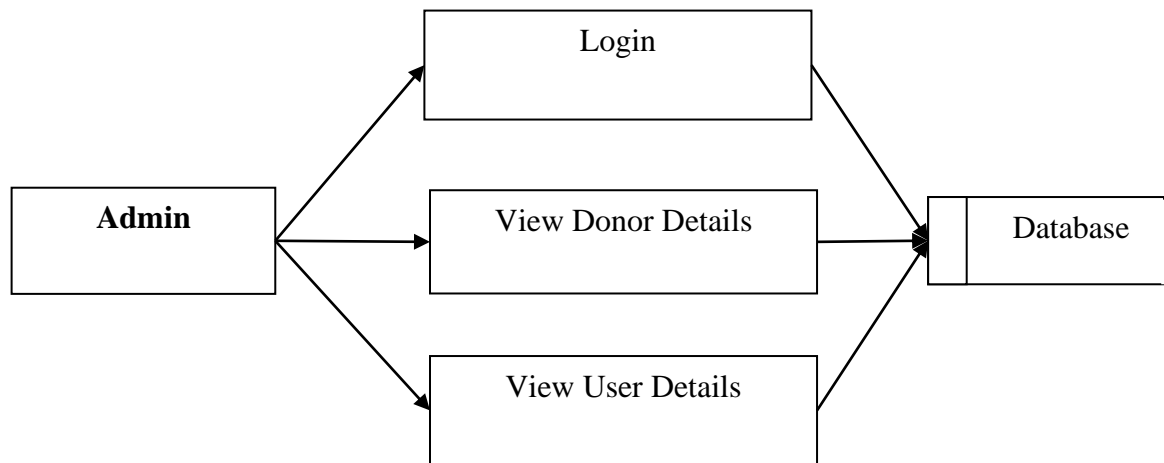
A data-flow diagram is a visual representation of how data moves through a system or a process (usually an information system). The DFD additionally gives details about each entity's inputs and outputs as well as the process itself. A data-flow diagram lacks control flow, loops, and decision-making processes. Using a flowchart, certain operations depending on the data may be depicted.

Data flow Symbols:

Symbol	Description
	An entity . A source of data or a destination for data.
	A process or task that is performed by the system.
	A data store , a place where data is held between processes.
	A data flow .

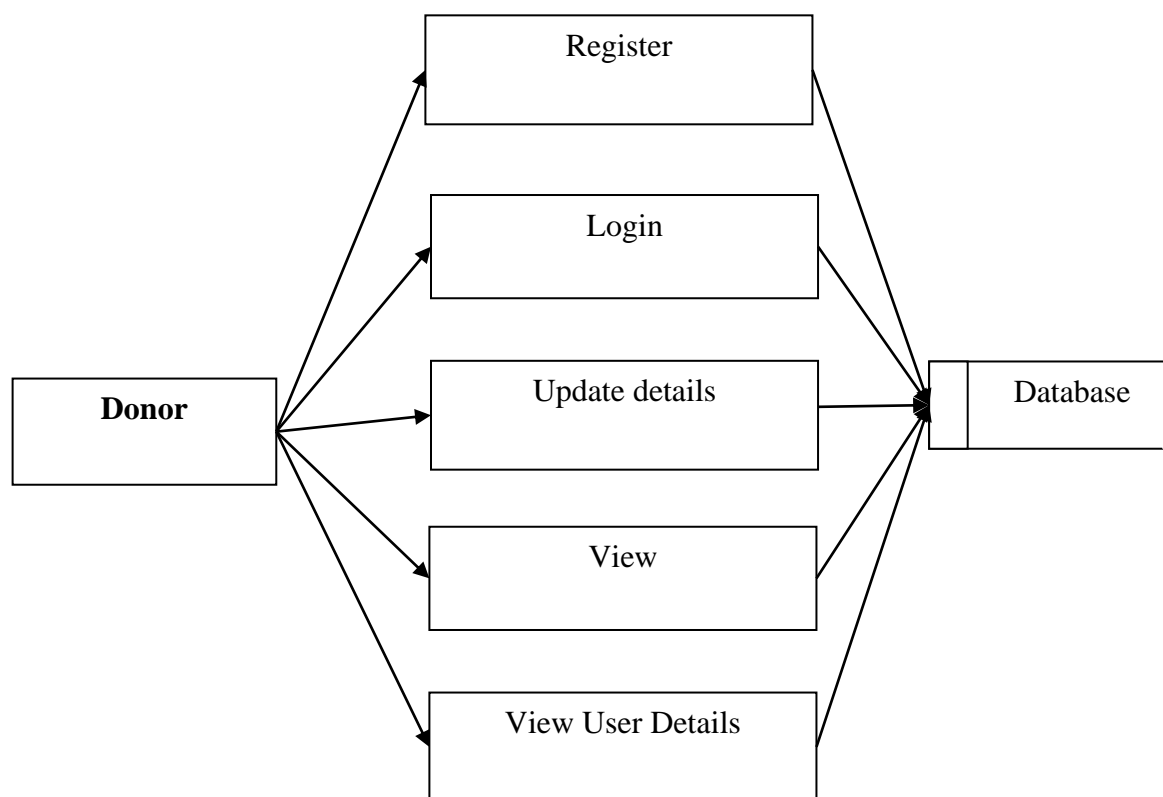
LEVEL 0

The Level 0 DFD shows how the system is divided into 'sub-systems' (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.



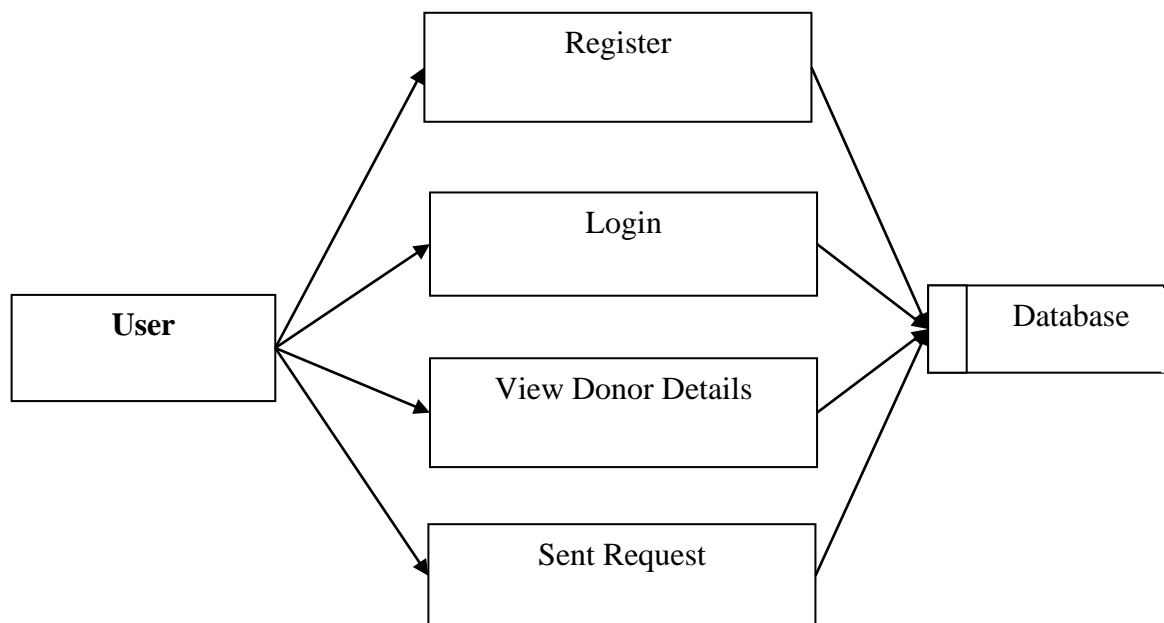
LEVEL 1

The next stage is to create the Level 1 Data Flow Diagram. This highlights the main functions carried out by the system. As a rule, to describe the system was using between two and seven functions - two being a simple system and seven being a complicated system. This enables us to keep the model manageable on screen or paper.

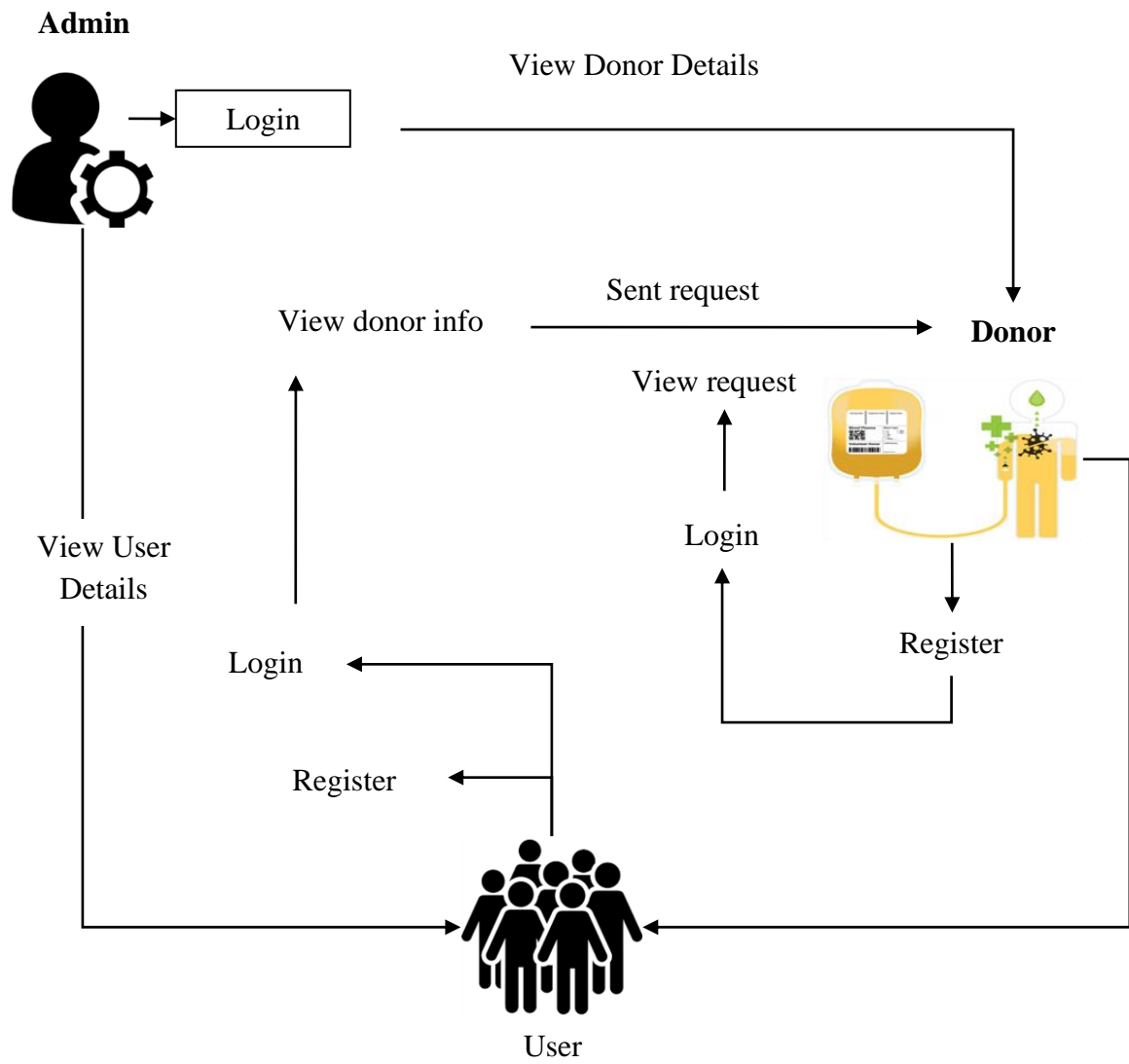


LEVEL 2

A Data Flow Diagram (DFD) tracks processes and their data paths within the business or system boundary under investigation. A DFD defines each domain boundary and illustrates the logical movement and transformation of data within the defined boundary. The diagram shows 'what' input data enters the domain, 'what' logical processes the domain applies to that data, and 'what' output data leaves the domain. Essentially, a DFD is a tool for process modeling and one of the oldest.



5.2 SOLUTION & TECHNICAL ARCHITECTURE



5.3 USER STORIES

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password.	I can access my account dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive successful message	High	Sprint-1
	Login	USN-3	As a user, I can log into the application by entering email & password	I can access into my Profile and view my dashboard	High	Sprint-1
	Dashboard	USN-4	As a user, I can login using my credentials and it will direct it to my dashboard	I can view and access what are the features are provided in dashboard	High	Sprint-1
Customer (Web user)		USN-5	As a user, I can login using my credentials and it will direct it to my dashboard	I can view and access what are the features are provided in dashboard	High	Sprint -1
Customer Care Executive	Query	USN-6	As a user had an any query about the given requirements	I can view a query and rectify the given query	medium	Sprint-2
Administrator	Login	USN-7	As a admin ,have credentials using that they can login	They can view and modify the data in database	medium	Sprint-2
	View	USN-8	As a admin I can view plasma information	View and modify	High	Sprint-1
	Modify	USN-9	As a admin I can modify the plasma information.	Modify only if there is a false information/	Medium	Sprint-1

6. PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING & ESTIMATION

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Donor Registration	USN-1	As a user, I can register in the donor application by entering my name, phone_no, Email id, blood group ,aadhar no	9	High	Team Lead (Haripriya M)
Sprint-1	Login	USN-2	As a admin, I can log into the application by entering email & password	9	High	Team Lead
Sprint-1	Chatbot	USN-3	As a user I can ask query in chatbot.	2	Medium	Team Lead
Sprint -2	Confirmation	USN-4	As a user, I can receive confirmation mail.	4	Medium	Team Lead
Sprint – 2	Dashboard	USN-5	As a user, I can view dashboard and select	5	Medium	Team Member 1
Sprint-2	View Donor List	USN-6	As a user, I can view all the donor list and contact them directly	9	High	Team Lead
Sprint-2	Search Donor	USN-7	As a user, I can search for the donor	9	Medium	Team Lead
Sprint-3	About us	USN-8	As a User, I can view the about us page which contains all contact information	5	Medium	Team Member 2
Sprint-3	Modify data	USN-9	As a admin, I can modify the User data.	9	High	Team Lead
Sprint-3	Send mail	USN-10	As a user, I can send mail to donors using sendgrid.	9	High	Team Lead Team Member 3 Team Member 1
Sprint-3	Home page	USN-11	As a user I can view the home page and select the desired option.	9	Medium	Team Lead Team Member 1 Team Member 2 Team Member 3
Sprint -4	Send Query	USN-12	As a user I can ask my query through email.	9	Medium	Team Lead Team Member 3
Sprint-4	Download data	USN-13	As a admin I can download the user data	9	High	Team Lead

6.2 SPRINT DELIVERY SCHEDULE

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Donor Registration	USN-1	As a user, I can register in the donor application by entering my name, phone_no, Email id, blood group ,aadhar no	9	High	Team Lead (Hari priya M)
Sprint-1	Login	USN-2	As a admin, I can log into the application by entering email & password	9	High	Team Lead
Sprint-1	Chatbot	USN-3	As a user I can ask query in chatbot.	2	Medium	Team Lead
Sprint -2	Confirmation	USN-4	As a user, I can receive confirmation mail.	4	Medium	Team Lead
Sprint – 2	Dashboard	USN-5	As a user, I can view dashboard and select	5	Medium	Team Member 1
Sprint-2	View Donor List	USN-6	As a user, I can view all the donor list and contact them directly	9	High	Team Lead
Sprint-2	Search Donor	USN-7	As a user, I can search for the donor	9	Medium	Team Lead
Sprint-3	About us	USN-8	As a User, I can view the about us page which contains all contact information	5	Medium	Team Member 2
Sprint-3	Modify data	USN-9	As a admin, I can modify the User data.	9	High	Team Lead
Sprint-3	Send mail	USN-10	As a user, I can send mail to donors using sendgrid.	9	High	Team Lead Team Member 3 Team Member 1
Sprint-3	Home page	USN-11	As a user I can view the home page and select the desired option.	9	Medium	Team Lead Team Member 1 Team Member 2 Team Member 3
Sprint -4	Send Query	USN-12	As a user I can ask my query through email.	9	Medium	Team Lead Team Member 3
Sprint-4	Download data	USN-13	As a admin I can download the user data	9	High	Team Lead
Sprint-4	Download data	USN-13	As a admin I can download the user data	9	High	Team Lead

7. CODING & SOLUTIONING

7.1 FEATURE 1

```
<!DOCTYPE html>
<html>
<head>
<title>Plasma Donor</title>

    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>

*,
*:before,
*:after{
    padding: 0;
    margin: 0;
    box-sizing: border-box;
}

nav{

    width: 100%;
    min-width: 600px;
    display: flex;
    background-color: #0a0a0a;
    padding: 20px;
```



```
    justify-content: space-around;

}

a{
    position: relative;
    text-decoration: none;
    font-family: 'Poppins',sans-serif;
    color: #a0a0a0;
    font-size: 18px;
    letter-spacing: 0.5px;
    padding: 0 10px;
}

a:after{
    content: "";
    position: absolute;
    background-color: #ff3c78;
    height: 3px;
    width: 0;
    left: 0;
    bottom: -10px;
    transition: 0.3s;
}

a:hover{
    color: #ffffff;
}

a:hover:after{
    width: 100%;
}
```

</style>

```

</head>
<body>

<nav>
    <a href="/">Home</a>
        <a href="/AdminLogin" >AdminLogin</a>
        <a href="/DonorLogin">DonorLogin</a>
        <a href="/NewDonor">NewDonor</a>
        <a href="/UserLogin">UserLogin</a>
        <a href="/NewUser">NewUser</a>
    <div class="animation start-home"></div>
</nav>
<!-- header -->

<!-- header -->
<!-- banner -->

<!-- //banner -->
<div class="about all_pad">
    <div class="container">
        <h3 class="title" align="center">Admin Login Here..!</h3>

        <form id="form1" name="form1" method="post" action="/adminlogin">
        <table width="100%" border="0" align="center">
            <tr>
                <th scope="col">&nbsp;</th>
                <th scope="col">&nbsp;</th>
                <th colspan="2" scope="col"><span class="style4"> </span></th>
                <th scope="col">&nbsp;</th>
                <th scope="col">&nbsp;</th>
            </tr>

```

```

<tr>
  <th width="9%" scope="col">&nbsp;</th>
  <th width="20%" scope="col">&nbsp;</th>
  <th width="20%" scope="col">&nbsp;</th>
  <th width="22%" scope="col">&nbsp;</th>
  <th width="24%" scope="col">&nbsp;</th>
  <th width="5%" scope="col">&nbsp;</th>
</tr>
<tr>
  <td height="35">&nbsp;</td>
  <td>&nbsp;</td>
  <td><span class="style3">User Name </span></td>
  <td><input name="uname" type="text" id="uname" /></td>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
</tr>
<tr>
  <td height="38">&nbsp;</td>
  <td>&nbsp;</td>
  <td><span class="style3">Password</span></td>
  <td><input name="password" type="password" id="password" /></td>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
</tr>
<tr>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
</tr>
<tr>
  <td>&nbsp;</td>

```

```

        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td><input name="bnt" type="submit" id="bnt" value="Login" />

        <input type="reset" name="Reset" value="Reset">
        </td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
    </tr>
    </table>
</form>

</div>
</div>

<footer style="padding:20px;background-color: black;">

```

<p style="text-align: center;color:white;">Copyright to plasma donation hub</p>

</footer>

</body>

</html>

7.2 FEATURE 2

```
8. from flask import Flask, render_template, flash, request, session, send_file
   from flask import render_template, redirect, url_for, request

import ibm_db
import pandas
import ibm_db_dbi
from sqlalchemy import create_engine

engine = create_engine('sqlite://',
                       echo = False)

dsn_hostname = "b70af05b-76e4-4bca-a1f5-
23dbb4c6a74e.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud"
dsn_uid = "gqk63124"
dsn_pwd = "sHBf4pu1Js4iA5iz"

dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "BLUDB"
dsn_port = "32716"
dsn_protocol = "TCPIP"
dsn_security = "SSL"

dsn = (
    "DRIVER={0};"
    "DATABASE={1};"
    "HOSTNAME={2};"
    "PORT={3};"
    "PROTOCOL={4};"
    "UID={5};"
    "PWD={6};"
    "SECURITY={7};").format(dsn_driver, dsn_database, dsn_hostname,
dsn_port, dsn_protocol, dsn_uid, dsn_pwd,dsn_security)

try:
    conn = ibm_db.connect(dsn, "", "")
    print ("Connected to database: ", dsn_database, "as user: ", dsn_uid,
"on host: ", dsn_hostname)

except:
    print ("Unable to connect: ", ibm_db.conn_errormsg() )

app = Flask(__name__)
app.config['DEBUG']
app.config['SECRET_KEY'] = '7d441f27d441f27567d441f2b6176a'
```

```

@app.route("/")
def homepage():

    return render_template('index.html')

@app.route("/AdminLogin")
def AdminLogin():

    return render_template('AdminLogin.html')

@app.route("/DonorLogin")
def DonorLogin():
    return render_template('DonorLogin.html')

@app.route("/NewDonor")
def NewDonor():
    return render_template('NewDonor.html')

@app.route("/UserLogin")
def UserLogin():
    return render_template('UserLogin.html')

@app.route("/PersonalInfo")
def PersonalInfo():
    return render_template('DonorPersonal.html')

@app.route("/NewUser")
def NewUser():
    return render_template('NewUser.html')

@app.route("/AdminHome")
def AdminHome():

    conn = ibm_db.connect(dsn, "", "")
    pd_conn = ibm_db_dbi.Connection(conn)
    selectQuery = "SELECT * from regtb "
    dataframe = pandas.read_sql(selectQuery, pd_conn)

    dataframe.to_sql('Employee_Data', con=engine, if_exists='append')
    data = engine.execute("SELECT * FROM Employee_Data").fetchall()
    return render_template('AdminHome.html',data=data)

@app.route("/AdminDonorInfo")
def AdminDonorInfo():

    conn = ibm_db.connect(dsn, "", "")
    pd_conn = ibm_db_dbi.Connection(conn)
    selectQuery = "SELECT * from personl1tb "
    dataframe = pandas.read_sql(selectQuery, pd_conn)

    dataframe.to_sql('Employee_Data', con=engine, if_exists='append')

```

```

        data = engine.execute("SELECT * FROM Employee_Data").fetchall()

    return render_template('AdminDonorInfo.html', data=data)

@app.route("/UserHome")
def UserHome():
    user = session['uname']

    conn = ibm_db.connect(dsn, "", "")
    pd_conn = ibm_db_dbi.Connection(conn)
    selectQuery = "SELECT * FROM regtb where username='" + user + "'"
    dataframe = pandas.read_sql(selectQuery, pd_conn)

    dataframe.to_sql('Employee_Data', con=engine, if_exists='append')
    data = engine.execute("SELECT * FROM Employee_Data").fetchall()
    return render_template('UserHome.html', data=data)

@app.route("/DonorHome")
def DonorHome():
    cuname = session['cname']

    conn = ibm_db.connect(dsn, "", "")
    pd_conn = ibm_db_dbi.Connection(conn)
    selectQuery = "SELECT * FROM donortb where username='" + cuname + "'"
    dataframe = pandas.read_sql(selectQuery, pd_conn)

    dataframe.to_sql('Employee_Data', con=engine, if_exists='append')
    data = engine.execute("SELECT * FROM Employee_Data").fetchall()

    return render_template('DonorHome.html', data=data)

@app.route("/adminlogin", methods=['GET', 'POST'])
def adminlogin():
    error = None
    if request.method == 'POST':
        if request.form['uname'] == 'admin' or request.form['password'] == 'admin':

```

```

        conn = ibm_db.connect(dsn, "", "")
        pd_conn = ibm_db_dbi.Connection(conn)
        selectQuery = "SELECT * FROM regtb"
        dataframe = pandas.read_sql(selectQuery, pd_conn)

        dataframe.to_sql('Employee_Data', con=engine,
if_exists='append')
        data = engine.execute("SELECT * FROM Employee_Data").fetchall()
        return render_template('AdminHome.html' , data=data)

    else:
        return render_template('index.html', error=error)

@app.route("/donorlogin", methods=['GET', 'POST'])
def donorlogin():
    error = None
    if request.method == 'POST':
        username = request.form['uname']
        password = request.form['password']
        session['dname'] = request.form['uname']

        conn = ibm_db.connect(dsn, "", "")
        pd_conn = ibm_db_dbi.Connection(conn)

        selectQuery = "SELECT * from donortb where username='" + username +
        "'" and Password='" + password + "'"
        dataframe = pandas.read_sql(selectQuery, pd_conn)

        if dataframe.empty:
            data1 = 'Username or Password is wrong'
            return render_template('goback.html', data=data1)
        else:
            print("Login")
            selectQuery = "SELECT * from donortb where username='" +
            username + "'" and Password='" + password + "'"
            dataframe = pandas.read_sql(selectQuery, pd_conn)

            dataframe.to_sql('Employee_Data',
                            con=engine,
                            if_exists='append')

            # run a sql query
            print(engine.execute("SELECT * FROM Employee_Data").fetchall())

            return render_template('DonorHome.html',
            data=engine.execute("SELECT * FROM Employee_Data").fetchall())

@app.route("/userlogin", methods=['GET', 'POST'])
def userlogin():

```



```

if request.method == 'POST':
    username = request.form['uname']
    password = request.form['password']
    session['uname'] = request.form['uname']

    conn = ibm_db.connect(dsn, "", "")
    pd_conn = ibm_db_dbi.Connection(conn)

    selectQuery = "SELECT * from regtb where UserName='" + username +
    "' and password='" + password + "'"
    dataframe = pandas.read_sql(selectQuery, pd_conn)

    if dataframe.empty:
        data1 = 'Username or Password is wrong'
        return render_template('goback.html', data=data1)
    else:
        print("Login")
        selectQuery = "SELECT * from regtb where UserName='" + username
        + "' and password='" + password + "'"
        dataframe = pandas.read_sql(selectQuery, pd_conn)

        dataframe.to_sql('Employee_Data',
                        con=engine,
                        if_exists='append')

        # run a sql query
        print(engine.execute("SELECT * FROM Employee_Data").fetchall())

        return render_template('UserHome.html',
                                data=engine.execute("SELECT * FROM Employee_Data").fetchall())

@app.route("/newuser", methods=['GET', 'POST'])
def newuser():
    if request.method == 'POST':

        name1 = request.form['name']
        gender1 = request.form['gender']
        Age = request.form['age']
        email = request.form['email']
        pnumber = request.form['phone']
        address = request.form['address']

        uname = request.form['uname']
        password = request.form['psw']

        conn = ibm_db.connect(dsn, "", "")

        insertQuery = "INSERT INTO regtb VALUES ('" + name1 + "','" +
        gender1 + "','" + Age + "','" + email + "','" + pnumber + "','" + address +
        "','" + uname + "','" + password + "'"
        insert_table = ibm_db.exec_immediate(conn, insertQuery)
        print(insert_table)
        # return 'file register successfully'

```

```

        return render_template('UserLogin.html')

@app.route("/personal", methods=['GET', 'POST'])
def personal():
    if request.method == 'POST':

        name1 = request.form['name']
        gender1 = request.form['gender']
        Age = request.form['age']
        email = request.form['email']
        pnumber = request.form['phone']
        address = request.form['address']

        blood = request.form['blood']
        health = request.form['health']
        dname = session['dname']

        conn = ibm_db.connect(dsn, "", "")

        insertQuery = "INSERT INTO personl1tb VALUES ('" + name1 + "','" + gender1 + "','" + Age + "','" + email + "','" + pnumber + "','" + address + "','" + blood + "','" + health + "','" + dname + "'"")
        insert_table = ibm_db.exec_immediate(conn, insertQuery)
        print(insert_table)

        alert = 'Record Saved'

        return render_template('goback.html', data=alert)

@app.route("/appr")
def appr():

    cid = request.args.get('cid')
    dname = session['dname']

    conn = ibm_db.connect(dsn, "", "")

    insertQuery = "delete from personl1tb where Name='" + str(cid) + "' and UserName='" + dname + "'"
    insert_table = ibm_db.exec_immediate(conn, insertQuery)
    print(insert_table)

    conn = ibm_db.connect(dsn, "", "")
    pd_conn = ibm_db_dbi.Connection(conn)
    selectQuery = "SELECT * FROM personl1tb where Username='" + dname + "'"
    dataframe = pandas.read_sql(selectQuery, pd_conn)

```

```

dataframe.to_sql('Employee_Data', con=engine, if_exists='append')
data = engine.execute("SELECT * FROM Employee_Data").fetchall()

return render_template('DonorPersonalInfo.html', data=data)

@app.route("/DonorPersonalInfo")
def DonorPersonalInfo():

    dname = session['dname']

    conn = ibm_db.connect(dsn, "", "")
    pd_conn = ibm_db_dbi.Connection(conn)
    selectQuery = "SELECT * FROM personl1tb where Username='" + dname + "'"
    dataframe = pandas.read_sql(selectQuery, pd_conn)

    dataframe.to_sql('Employee_Data', con=engine, if_exists='append')
    data = engine.execute("SELECT * FROM Employee_Data").fetchall()

    return render_template('DonorPersonalInfo.html', data=data)

@app.route("/newdonor", methods=['GET', 'POST'])
def newdonor():
    if request.method == 'POST':

        name1 = request.form['name']

        phone = request.form['phone']

        email = request.form['email']

        uname = request.form['uname']
        password = request.form['psw']

        conn = ibm_db.connect(dsn, "", "")

        insertQuery = "INSERT INTO donortb VALUES ('" + name1 + "', '" +
phone + "', '" + email + "', '" + uname + "', '" + password + "')"
        insert_table = ibm_db.exec_immediate(conn, insertQuery)
        print(insert_table)

    return render_template('DonorLogin.html')

@app.route("/Search")
def Search():

    return render_template('Search.html')

@app.route("/dsearch", methods=['GET', 'POST'])

```

```

def dsearch():
    if request.form["submit"] == "Search":
        blood = request.form['blood']

        conn = ibm_db.connect(dsn, "", "")
        pd_conn = ibm_db_dbi.Connection(conn)
        selectQuery = "SELECT * FROM personlrb where blood='" + blood +
        ""

        dataframe = pandas.read_sql(selectQuery, pd_conn)

        dataframe.to_sql('Employee_Data', con=engine, if_exists='append')
        data = engine.execute("SELECT * FROM Employee_Data").fetchall()

        return render_template('Search.html', data=data)

    elif request.form["submit"] == "SendMail":
        blood = request.form['blood']
        info = request.form['info']

        conn = ibm_db.connect(dsn, "", "")
        pd_conn = ibm_db_dbi.Connection(conn)
        selectQuery = "SELECT * FROM personlrb where Blood like '%" +
        blood + "%'"
        dataframe = pandas.read_sql(selectQuery, pd_conn)

        dataframe.to_sql('Employee_Data', con=engine, if_exists='append')
        data = engine.execute("SELECT * FROM Employee_Data").fetchall()

        for item in data:
            sendmsg(item[4], info)
            print(item[4])
            alert = 'Send Notication'

        return render_template('goback.html', data=alert)

@app.route("/SendRequest")
def SendRequest():

    session['cid'] = request.args.get('cid')

    return render_template('Notification.html')

```

```

def sendmsg(Mailid,message):
    import smtplib
    from email.mime.multipart import MIMEMultipart
    from email.mime.text import MIMEText
    from email.mime.base import MIMEBase
    from email import encoders

    fromaddr = "riyairah2002@gmail.com"
    toaddr = Mailid

    # instance of MIMEMultipart
    msg = MIMEMultipart()

    # storing the senders email address
    msg['From'] = fromaddr

    # storing the receivers email address
    msg['To'] = toaddr

    # storing the subject
    msg['Subject'] = "Alert"

    # string to store the body of the mail
    body = message

    # attach the body with the msg instance
    msg.attach(MIMEText(body, 'plain'))

    # creates SMTP session
    s = smtplib.SMTP('smtp.gmail.com', 587)

    # start TLS for security
    s.starttls()

    # Authentication
    s.login(fromaddr, "otfsaxtbjkywthoj")

    # Converts the Multipart msg into a string
    text = msg.as_string()

    # sending the mail
    s.sendmail(fromaddr, toaddr, text)

    # terminating the session
    s.quit()

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug = True, port = 5000)

```

8.1 DATABASE SCHEMA

```
CREATE TABLE donortb (  
    Name varchar(250) NOT NULL,  
    Mobile varchar(250) NOT NULL,  
    Email varchar(250) NOT NULL,  
    UserName varchar(250) NOT NULL,  
    Password varchar(250) NOT NULL  
)  
  
CREATE TABLE personl1tb (  
    Name varchar(250) NOT NULL,  
    Gender varchar(250) NOT NULL,  
    Age varchar(250) NOT NULL,  
    Email varchar(250) NOT NULL,  
    Phone varchar(250) NOT NULL,  
    Address varchar(500) NOT NULL,  
    blood varchar(250) NOT NULL,  
    Health varchar(250) NOT NULL,  
    UserName varchar(50) NOT NULL  
)  
  
CREATE TABLE regtb (  
    Name varchar(250) NOT NULL,  
    Gender varchar(250) NOT NULL,  
    Age varchar(250) NOT NULL,  
    Email varchar(250) NOT NULL,  
    Mobile varchar(250) NOT NULL,  
    Address varchar(250) NOT NULL,  
    UserName varchar(250) NOT NULL,  
    Password varchar(250) NOT NULL  
)
```

```
-- phpMyAdmin SQL Dump  
-- version 2.11.6  
-- http://www.phpmyadmin.net  
--  
-- Host: localhost  
-- Generation Time: Nov 05, 2022 at 04:58 AM  
-- Server version: 5.0.51  
-- PHP Version: 5.2.6  
  
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";  
  
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;  
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
```

```

/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Database: `1plasmadb`
--

--
-- -----
--
-- Table structure for table `admintb`
--

CREATE TABLE `admintb` (
  `UserName` varchar(250) NOT NULL,
  `Password` varchar(250) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `admintb`
--

INSERT INTO `admintb` (`UserName`, `Password`) VALUES
('admin', 'admin');

--
-- -----
--
-- Table structure for table `donortb`
--

CREATE TABLE `donortb` (
  `id` bigint(10) NOT NULL auto_increment,
  `Name` varchar(250) NOT NULL,
  `Mobile` varchar(250) NOT NULL,
  `Email` varchar(250) NOT NULL,
  `UserName` varchar(250) NOT NULL,
  `Password` varchar(250) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=2 ;

--
-- Dumping data for table `donortb`
--

INSERT INTO `donortb` (`id`, `Name`, `Mobile`, `Email`, `UserName`, `Password`)
VALUES
(1, 'Sangeeth kumar', '9087556035', 'san@gmail.com', 'san', 'san');

--
-- -----
--
-- Table structure for table `personl1tb`
--

CREATE TABLE `personl1tb` (
  `id` bigint(10) NOT NULL auto_increment,
  `Name` varchar(250) NOT NULL,
  `Gender` varchar(250) NOT NULL,

```

```

`Age` varchar(250) NOT NULL,
`Email` varchar(250) NOT NULL,
`Phone` varchar(250) NOT NULL,
`Address` varchar(500) NOT NULL,
`blood` varchar(250) NOT NULL,
`Health` varchar(250) NOT NULL,
`UserName` varchar(50) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=2 ;

--
-- Dumping data for table `personl1tb`
--

INSERT INTO `personl1tb` (`id`, `Name`, `Gender`, `Age`, `Email`, `Phone`,
`Address`, `blood`, `Health`, `UserName`) VALUES
(1, 'san', 'male', '20', 'sangeeth5535@gmail.com', '9486365535', 'No 16, Samnath
plazza, Melapudur', 'A+', 'nill', 'san');

-----

--
-- Table structure for table `regtb`
--

CREATE TABLE `regtb` (
  `Name` varchar(250) NOT NULL,
  `Gender` varchar(250) NOT NULL,
  `Age` varchar(250) NOT NULL,
  `Email` varchar(250) NOT NULL,
  `Mobile` varchar(250) NOT NULL,
  `Address` varchar(250) NOT NULL,
  `UserName` varchar(250) NOT NULL,
  `Password` varchar(250) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `regtb`
--

INSERT INTO `regtb` (`Name`, `Gender`, `Age`, `Email`, `Mobile`, `Address`,
`UserName`, `Password`) VALUES
('san', 'male', '20', 'sangeeth5535@gmail.com', '9486365535', 'no 6 trichy',
'san', 'san'),
('sanNew', 'male', '20', 'sangeeth5535@gmail.com', '9486365535', 'no ', 'sanNew',
'sanNew'),
('mani', 'male', '33', 'ishu@gmail.com', '9486365535', 'dgh', 'mani', 'mani');

```


9. TESTING

9.1 TEST CASES

A test case has components that describe input, action and an expected response, in order to determine if a feature of an application is working correctly. A test case is a set of instructions on “HOW” to validate a particular test objective/target, which when followed will tell us if the expected behavior of the system is satisfied or not.

Characteristics of a good test case:

- Accurate: Exacts the purpose.
- Economical: No unnecessary steps or words.
- Traceable: Capable of being traced to requirements.
- Repeatable: Can be used to perform the test over and over.
- Reusable: Can be reused if necessary.

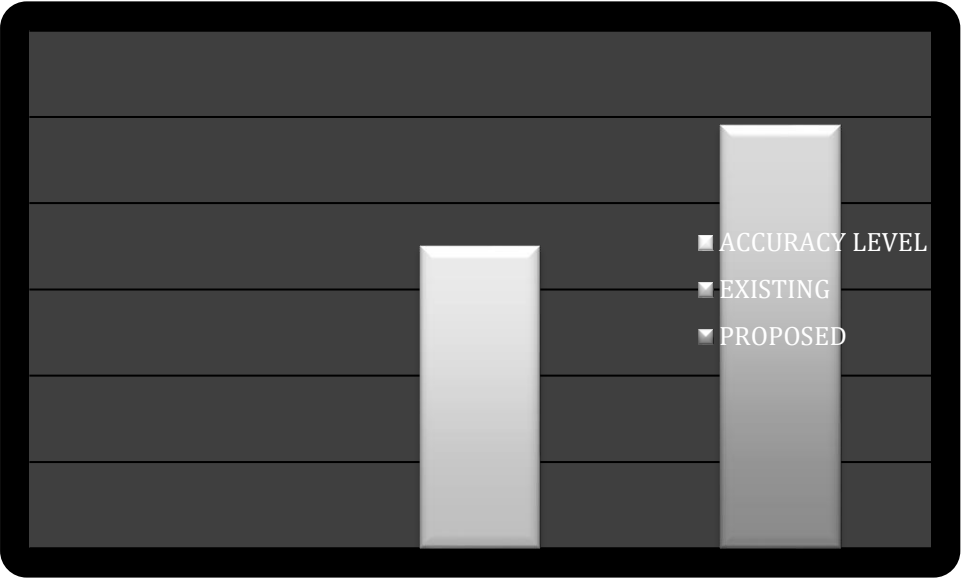
S.NO	Scenario	Input	Excepted output	Actual output
1	Admin Login Form	User name and password	Login	Login success.
2	Donor Registration Form	Donor basic details	Registration	Donor registration details stored in database.
3	User Registration Form	User basic details	Registration	User registration details stored in database.
4	User Login Form	User name and password	Login	Login success.

9.2 USER ACCEPTANCE TESTING

This is a type of testing done by users, customers, or other authorised entities to determine application/software needs and business processes. Acceptance testing is the most important phase of testing as this decides whether the client approves the application/software or not. It may involve functionality, usability, performance, and U.I of the application. It is also known as user acceptance testing (UAT), operational acceptance testing (OAT), and end-user testing.

10. RESULTS

10.1PERFORMANCE METRICS



11. ADVANTAGES & DISADVANTAGES

ADVANTAGES

- It is a user-friendly application.
- The people in need of plasma can search for the donors by giving their blood group and city name.
- It saves time as he can search donors online without going anywhere.
- Using this system user can get plasma in time and can save and here our system work, whenever a person needs plasma user get information of the person who has the same blood group needs.

DISADVANTAGES

- It cannot auto verify user genuineness.
- It is time consuming
- It leads to error prone results
- It consumes lot of manpower to better results
- It lacks of data security
- Retrieval of data takes lot of time

12. CONCLUSION

This project is designed for successful completion of project on Plasma Donor Application system. The basic building aim is to provide plasma donation service to the city recently. Plasma Donor Application System is a Web based application that is designed to store, process, retrieve and analyze information concerned with the administrative and inventory management within a plasma. This project aims at maintaining all the information pertaining to plasma donors, different blood groups available in each plasma bank and helps them manage in a better way plasma donation system can collect plasma from many donators in short from various sources and distribute that plasma to needy people who require plasma. To do all this we require high quality Web Application to manage those jobs. Plasma application provides a reliable platform to connect local plasma donors with patients.

13. FUTURE SCOPE

This system is developed such a way that additional enhancement can be done without much difficulty. The renovation of the project would increase the flexibility of the system. In future, we can develop this project in android platform. We will add extra features like donor location tracking system (GPS), Feedback form, and enable call option etc.

14. APPENDIX

SOURCE CODE

```
from flask import Flask, render_template, flash, request, session, send_file
from flask import render_template, redirect, url_for, request
#from wtforms import Form, TextField, TextAreaField, validators, StringField, SubmitField
from werkzeug.utils import secure_filename
import datetime

from flask_mail import Mail, Message

import mysql.connector
import sys
app = Flask(__name__)
app.config['DEBUG']
app.config['SECRET_KEY'] = '7d441f27d441f27567d441f2b6176a'

@app.route("/")
def homepage():

    return render_template('index.html')

@app.route("/AdminLogin")
def AdminLogin():

    return render_template('AdminLogin.html')

@app.route("/DonorLogin")
```

```

def DonorLogin():
    return render_template('DonorLogin.html')

@app.route("/NewDonor")
def NewDonor():
    return render_template('NewDonor.html')
@app.route("/UserLogin")
def UserLogin():
    return render_template('UserLogin.html')

@app.route("/PersonalInfo")
def PersonalInfo():
    return render_template('DonorPersonal.html')

@app.route("/NewUser")
def NewUser():
    return render_template('NewUser.html')

@app.route("/AdminHome")
def AdminHome():
    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM regtb ")
    data = cur.fetchall()
    return render_template('AdminHome.html',data=data)

@app.route("/AdminDonorInfo")
def AdminDonorInfo():

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
    cur = conn.cursor()

```



```
cur.execute("SELECT * FROM personltb ")
data = cur.fetchall()
```

```
return render_template('AdminDonorInfo.html', data=data)
```

```
@app.route("/UserHome")
```

```
def UserHome():
```

```
    user = session['uname']
```

```
    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
```

```
    # cursor = conn.cursor()
```

```
    cur = conn.cursor()
```

```
    cur.execute("SELECT * FROM regtb where username='" + user + "'")
```

```
    data = cur.fetchall()
```

```
    return render_template('UserHome.html',data=data)
```

```
@app.route("/DonorHome")
```

```
def DonorHome():
```

```
    cuname = session['cname']
```

```
    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
```

```
    # cursor = conn.cursor()
```

```
    cur = conn.cursor()
```

```
    cur.execute("SELECT * FROM companytb where username='" + cuname + "'")
```

```
    data = cur.fetchall()
```

```
    return render_template('DonorHome.html', data=data)
```

```
@app.route("/adminlogin", methods=['GET', 'POST'])
```

```
def adminlogin():
```

```
    error = None
```

```

if request.method == 'POST':
    if request.form['uname'] == 'admin' or request.form['password'] == 'admin':

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
        # cursor = conn.cursor()
        cur = conn.cursor()
        cur.execute("SELECT * FROM regtb ")
        data = cur.fetchall()
        return render_template('AdminHome.html' , data=data)

    else:
        return render_template('index.html', error=error)

@app.route("/donorlogin", methods=['GET', 'POST'])
def donorlogin():
    error = None
    if request.method == 'POST':
        username = request.form['uname']
        password = request.form['password']
        session['dname'] = request.form['uname']

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
        cursor = conn.cursor()
        cursor.execute("SELECT * from donortb where username='" + username + "' and
Password='" + password + "'")
        data = cursor.fetchone()
        if data is None:
            alert = 'Username or Password is wrong'
            return render_template('goback.html', data=alert)
        else:
            print(data[0])

```

```

        session['uid'] = data[0]

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')

        # cursor = conn.cursor()

        cur = conn.cursor()

        cur.execute("SELECT * FROM donortb where username='" + username + "' and
Password='" + password + "'")

        data = cur.fetchall()


        return render_template('DonorHome.html', data=data)


@app.route("/userlogin", methods=['GET', 'POST'])
def userlogin():

    if request.method == 'POST':

        username = request.form['uname']
        password = request.form['password']
        session['uname'] = request.form['uname']


        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')

        cursor = conn.cursor()

        cursor.execute("SELECT * from regtb where username='" + username + "' and
Password='" + password + "'")

        data = cursor.fetchone()

        if data is None:

            alert = 'Username or Password is wrong'
            return render_template('goback.html', data=alert)
        else:

            print(data[0])
            session['uid'] = data[0]

            conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')

```

```

        # cursor = conn.cursor()

        cur = conn.cursor()

        cur.execute("SELECT * FROM regtb where username='" + username + "' and
Password='" + password + "'")

        data = cur.fetchall()

        return render_template('UserHome.html', data=data )

@app.route("/newuser", methods=['GET', 'POST'])
def newuser():
    if request.method == 'POST':

        name1 = request.form['name']
        gender1 = request.form['gender']
        Age = request.form['age']
        email = request.form['email']
        pnumber = request.form['phone']
        address = request.form['address']

        uname = request.form['uname']
        password = request.form['psw']

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO regtb VALUES ('" + name1 + "'," + gender1 + "'," + Age + "'," +
email + "'," + pnumber + "'," + address + "'," + uname + "'," + password + "'")
        conn.commit()
        conn.close()

        # return 'file register successfully'

    return render_template('UserLogin.html')

```

```

@app.route("/personal", methods=['GET', 'POST'])
def personal():
    if request.method == 'POST':

        name1 = request.form['name']
        gender1 = request.form['gender']
        Age = request.form['age']
        email = request.form['email']
        pnumber = request.form['phone']
        address = request.form['address']

        blood = request.form['blood']
        health = request.form['health']
        dname = session['dname']

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO personltb VALUES ('" + name1 + "','" + gender1 + "','" + Age +
            "','" + email + "','" + pnumber + "','" + address + "','" + blood + "','" + health + "','" +
            dname+"'))")
        conn.commit()
        conn.close()

        alert = 'Record Saved'

        return render_template('goback.html', data=alert)

@app.route("/appr")
def appr():

```

```

cid = request.args.get('cid')
dname = session['dname']

conn = mysql.connector.connect(user='root', password="", host='localhost',
database='IPlasmadb')
cursor = conn.cursor()
cursor.execute(
    "delete from personltb where id=" + str(cid) + " ")
conn.commit()
conn.close()

conn = mysql.connector.connect(user='root', password="", host='localhost',
database='IPlasmadb')
cur = conn.cursor()
cur.execute("SELECT * FROM personltb where Username=" + dname + " ")
data = cur.fetchall()

return render_template('DonorPersonalInfo.html', data=data)

@app.route("/DonorPersonalInfo")
def DonorPersonalInfo():

    dname = session['dname']

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='IPlasmadb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM personltb where Username=" + dname + " ")
    data = cur.fetchall()

    return render_template('DonorPersonalInfo.html', data=data)

```

```

@app.route("/newdonor", methods=['GET', 'POST'])
def newdonor():
    if request.method == 'POST':

        name1 = request.form['name']

        phone = request.form['phone']

        email = request.form['email']

        uname = request.form['uname']
        password = request.form['psw']


        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO donortb VALUES ('" + name1 + "','" + phone + "','" + email + "','"
+ uname + "','" + password + "')"
        conn.commit()
        conn.close()

    return render_template('DonorLogin.html')


@app.route("/Search")
def Search():

    return render_template('Search.html')


@app.route("/dsearch", methods=['GET', 'POST'])
def dsearch():

```

```

if request.form["submit"] == "Search":
    blood = request.form['blood']

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM personltb where blood =" + blood + "")
    data = cur.fetchall()
    return render_template('Search.html', data=data)

elif request.form["submit"] == "SendMail":
    blood = request.form['blood']
    info = request.form['info']

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM personltb where Blood like '%" + blood + "%")
    data = cursor.fetchall()
    for item in data:
        sendmsg(item[4], info)
        print(item[4])
    alert = 'Send Notication'

    return render_template('goback.html', data=alert)

@app.route("/SendRequest")
def SendRequest():

    session['cid'] = request.args.get('cid')

    return render_template('Notification.html')

```



```

@app.route("/noti", methods=['GET', 'POST'])
def noti():

    info = request.form['info']

    did = session['cid']

    print(did)

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM personltb where id='" + did + "'")
    data = cursor.fetchone()
    if data:
        bloo =data[7]
        print(bloo)
    else:
        return 'Incorrect username / password !'

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='lPlasmadb')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM personltb where Blood like '%" + bloo + "%'")
    data = cursor.fetchall()
    for item in data:
        sendmsg(item[4], info)
        print(item[4])

    alert = 'Send Notication'

    return render_template('goback.html', data=alert)

```

```
def sendmsg(Mailid,message):  
    import smtplib  
    from email.mime.multipart import MIMEMultipart  
    from email.mime.text import MIMEText  
    from email.mime.base import MIMEBase  
    from email import encoders  
  
    fromaddr = "sampletest685@gmail.com"  
    toaddr = Mailid  
  
    # instance of MIMEMultipart  
    msg = MIMEMultipart()  
  
    # storing the senders email address  
    msg['From'] = fromaddr  
  
    # storing the receivers email address  
    msg['To'] = toaddr  
  
    # storing the subject  
    msg['Subject'] = "Alert"  
  
    # string to store the body of the mail  
    body = message  
  
    # attach the body with the msg instance  
    msg.attach(MIMEText(body, 'plain'))  
  
    # creates SMTP session  
    s = smtplib.SMTP('smtp.gmail.com', 587)  
  
    # start TLS for security  
    s.starttls()
```

```
# Authentication
```

```
s.login(fromaddr, "hneucvnontsuwgpj")
```

```
# Converts the Multipart msg into a string
```

```
text = msg.as_string()
```

```
# sending the mail
```

```
s.sendmail(fromaddr, toaddr, text)
```

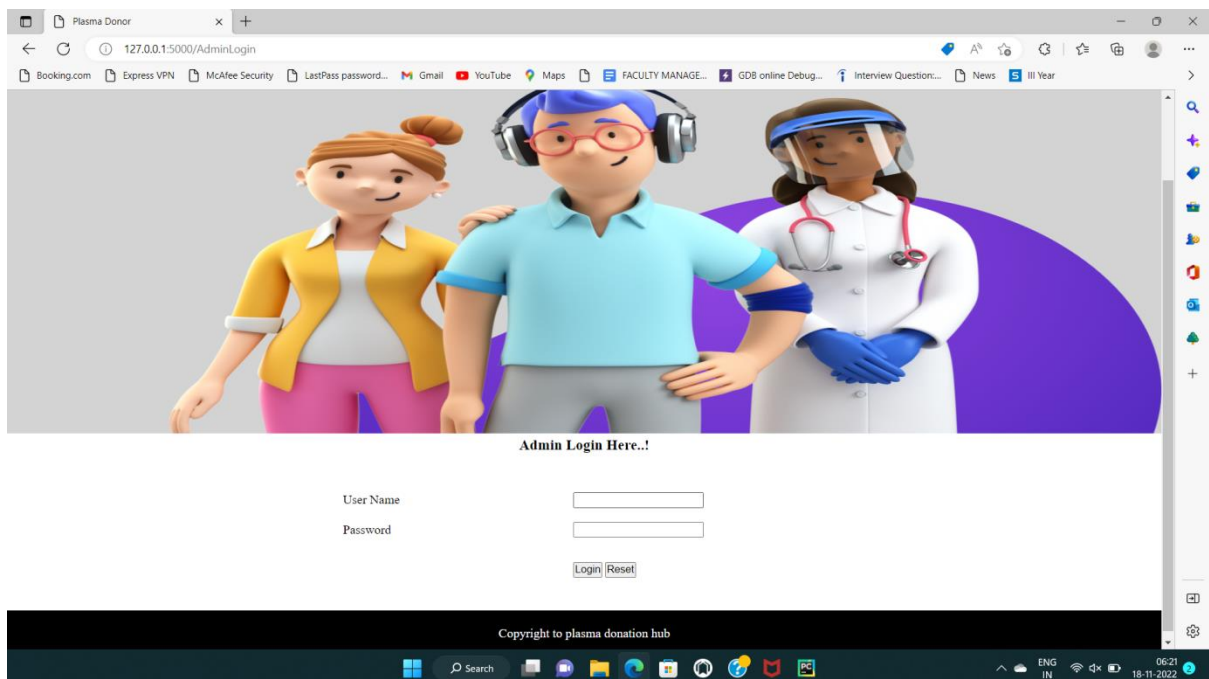
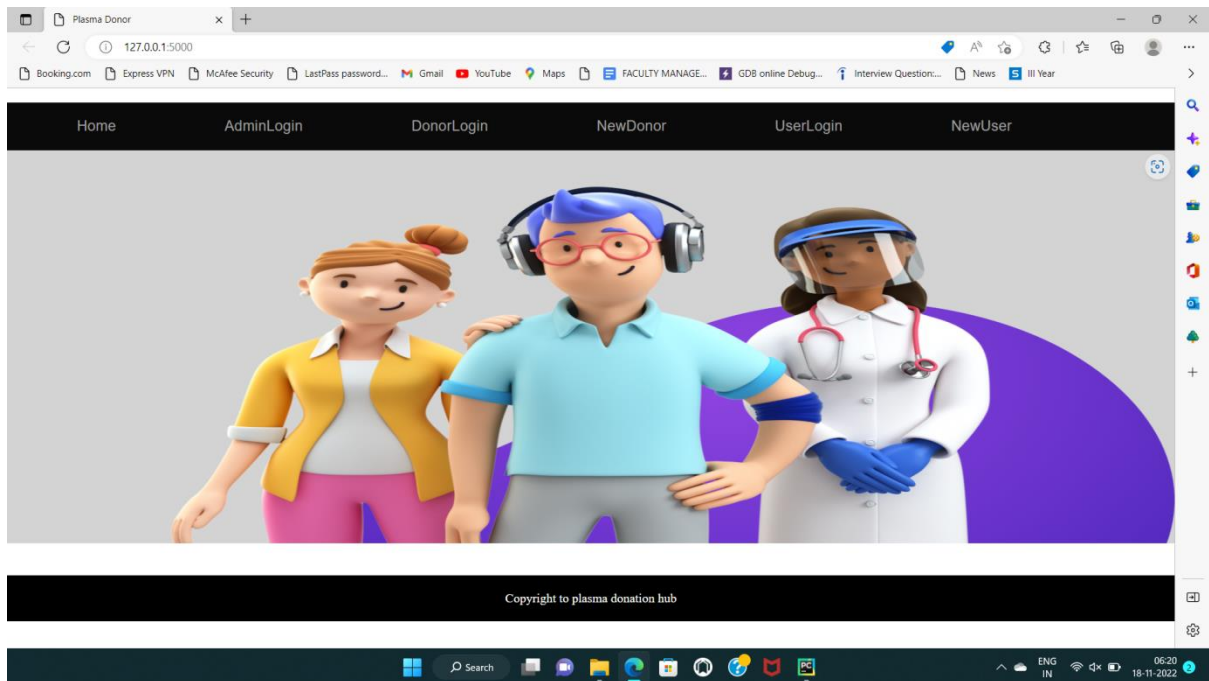
```
# terminating the session
```

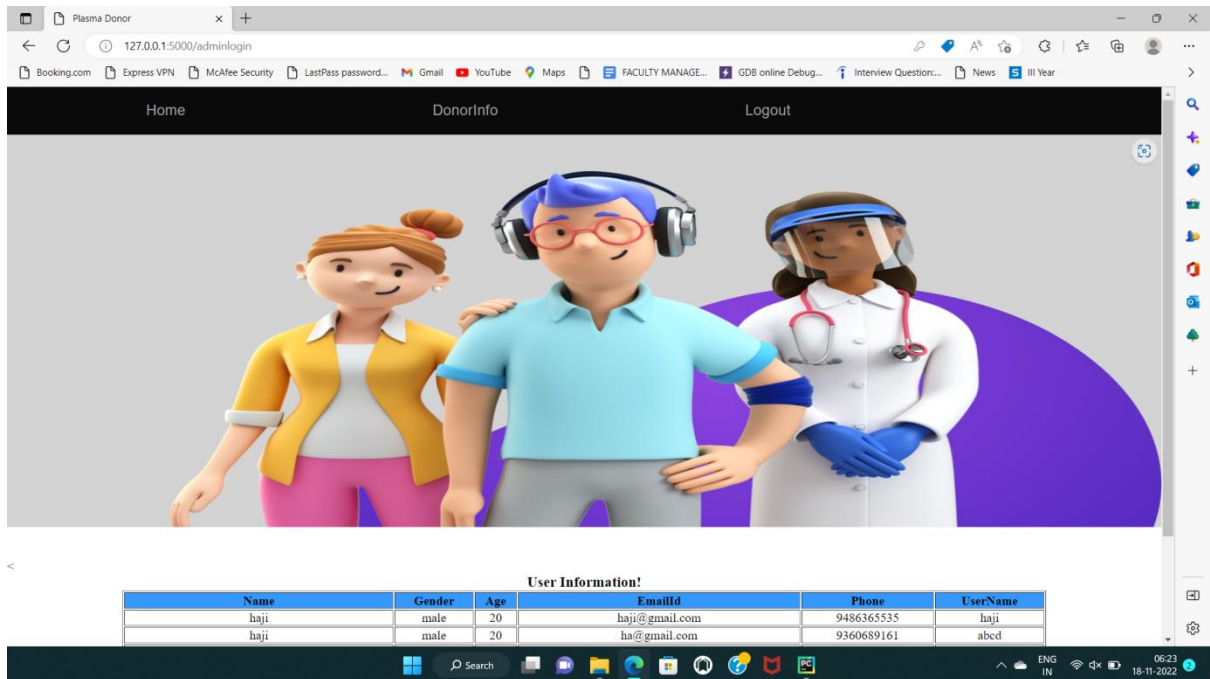
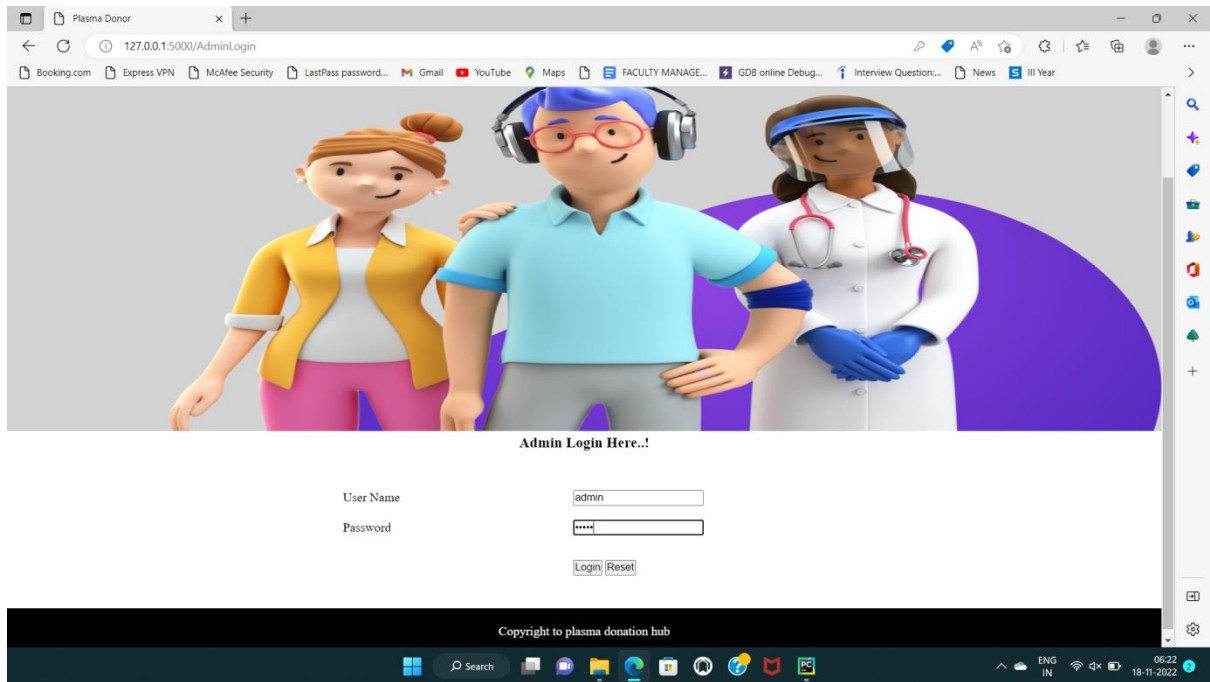
```
s.quit()
```

```
if __name__ == '__main__':
```

```
    app.run(debug=True, use_reloader=True)
```

SCREENSHOTS






Plasma Donor

127.0.0.1:5000/AdminDonorInfo

Home DonorInfo Logout




Donor Information!

Name	Gender	Age	Email	Phone	Address	BloodGroup	HealthIssues
san	male	20	sangeeth5535@gmail.com	9486365535	No 16, Samnath Plaza, Madurai Main Road, Melapudhur	A+	nil
Haji ali meeran M	male	20	hajaslan111apat@gmail.com	9360689161	9/40, Middle street, Aravakurichi, Karur	A+	hg
Haji ali meeran M	male	20	hajaslan111apat@gmail.com	9360689161	9/40, Middle street, Aravakurichi, Karur	A+	j

127.0.0.1:5000/AdminDonorInfo

Plasma Donor

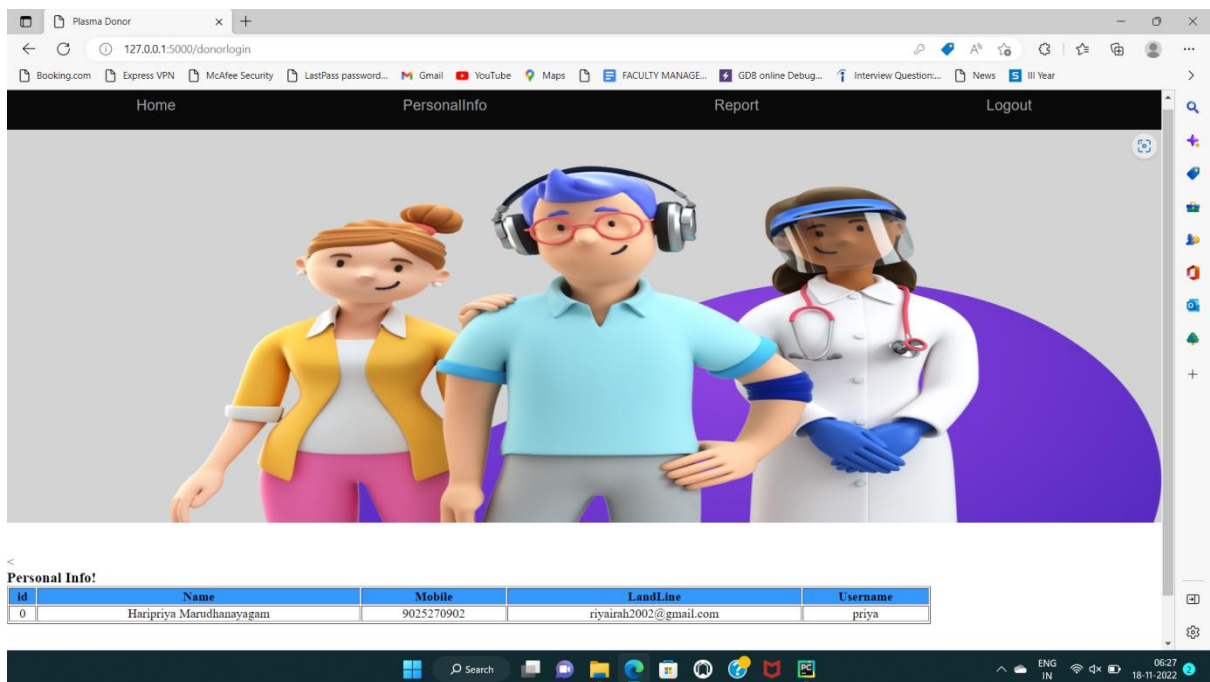
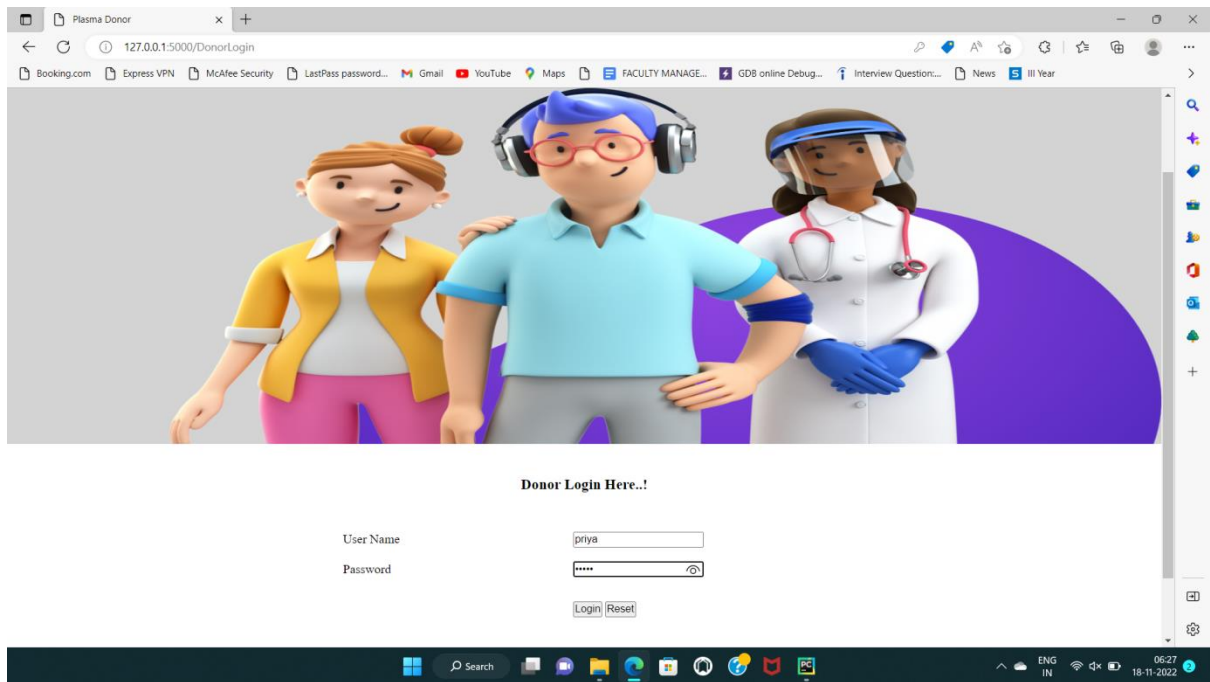
127.0.0.1:5000/DonorLogin

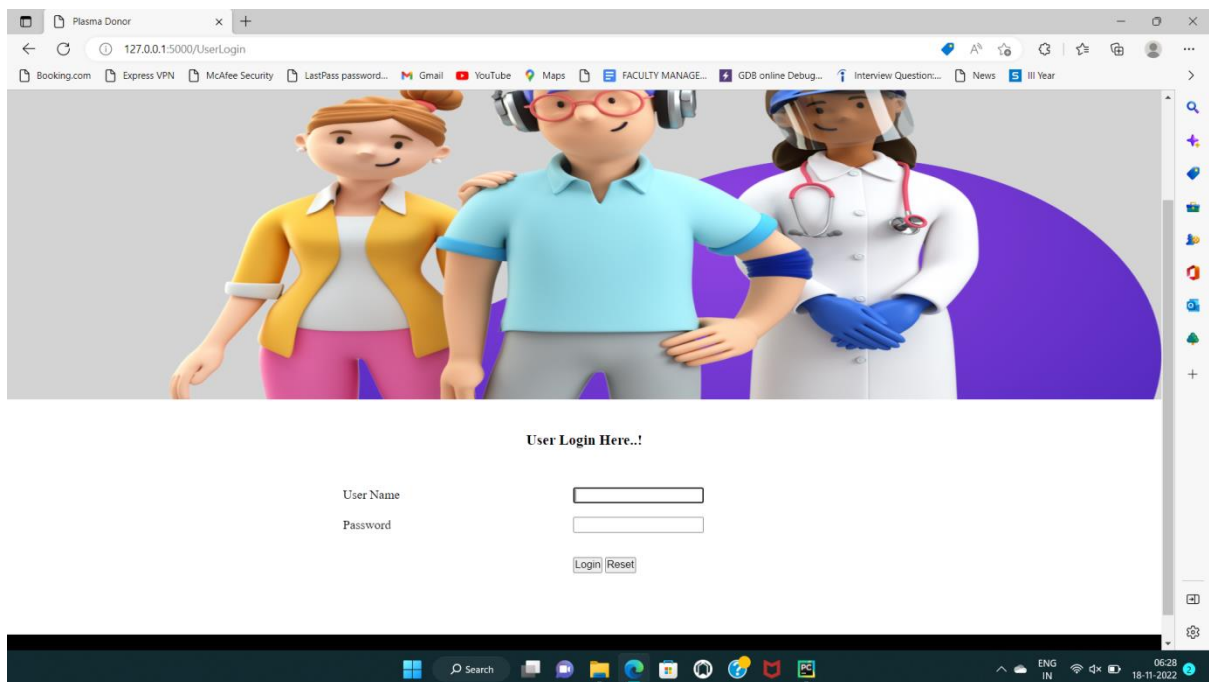
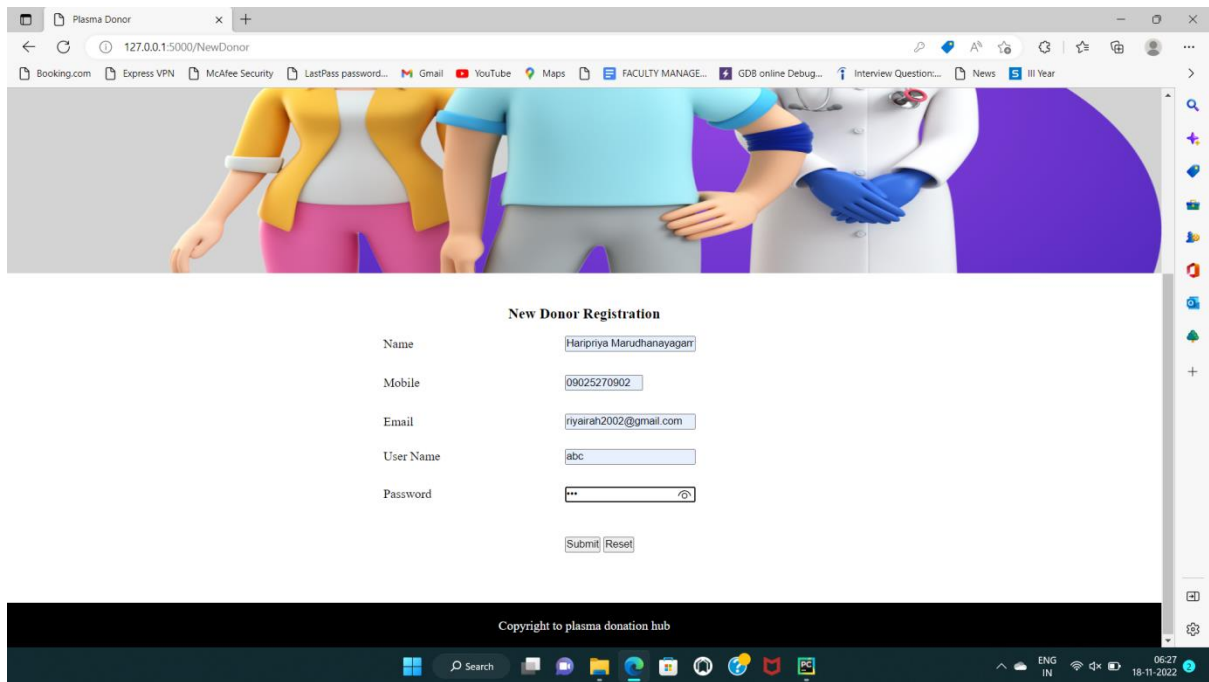


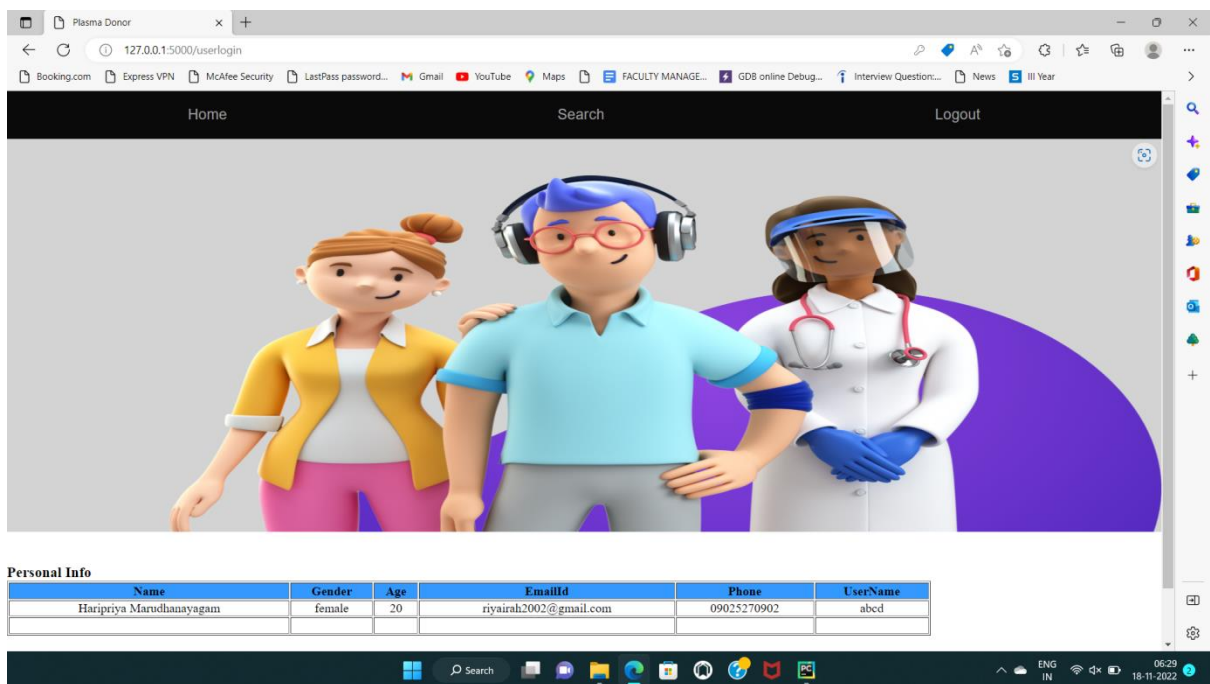
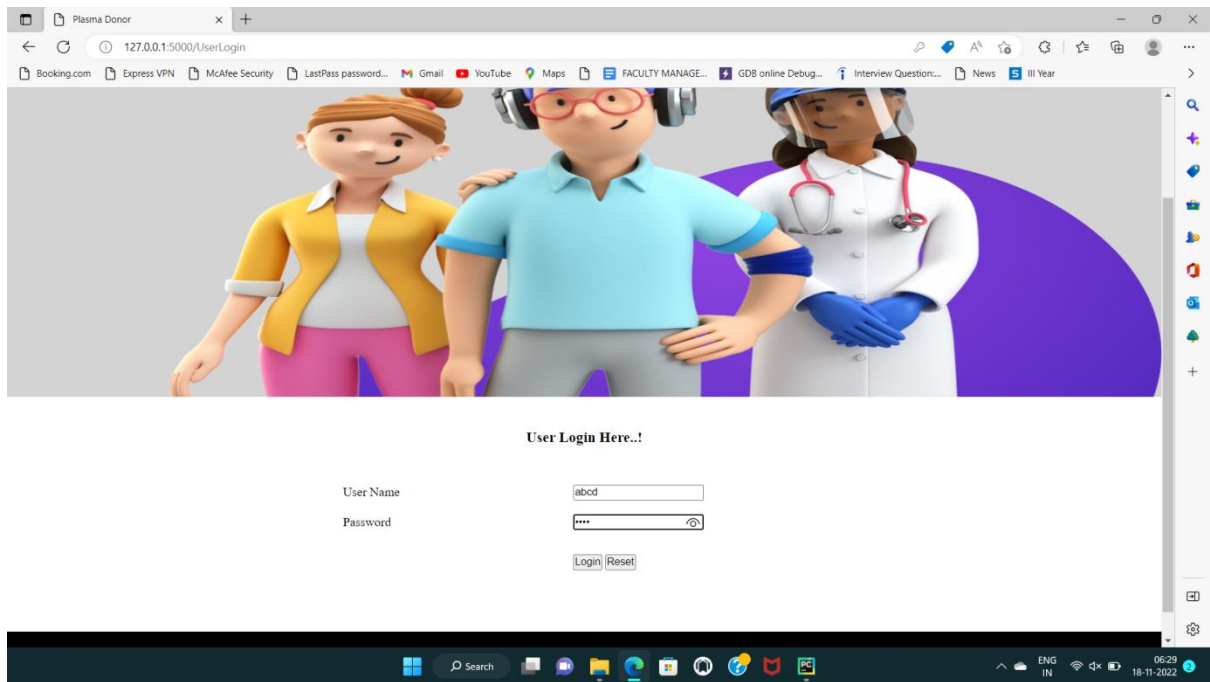
Donor Login Here..!

User Name

Password








Plasma Donor x +

127.0.0.1:5000/NewUser

Booking.com Express VPN McAfee Security LastPass password... Gmail YouTube Maps FACULTY MANAGE... GDB online Debug... Interview Question... News III Year



New User Registration

Name

Gender ☐ Male ☐ Female

Age

Email Id

Phone Number

Address

User Name

Passwrod

Search

ENG IN 06:29 18-11-2022

GITHUB & PROJECT DEMO LINK

[https://github.com/IBM-EPBL/IBM-Project-20146-1659713370/blob/bf4af04a5f813fd6c211fe882228290c5d349730/Final%20Deliverables/Plasma%20Donor%20and%20%20more%20pages%20-%20Profile%201%20-%20Microsoft_%20Edge%202022-11-19%2015-13-04%20\(1\)%20\(1\).mp4](https://github.com/IBM-EPBL/IBM-Project-20146-1659713370/blob/bf4af04a5f813fd6c211fe882228290c5d349730/Final%20Deliverables/Plasma%20Donor%20and%20%20more%20pages%20-%20Profile%201%20-%20Microsoft_%20Edge%202022-11-19%2015-13-04%20(1)%20(1).mp4)