Project Report

1. INTRODUCTION

1.1 Project Overview

- This system is used if anyone needs a Plasma Donor. This system comprises of Admin and User where both can request for a Plasma.
- In this system there is something called an active user, which means the user is an Active member of the App and any sort of infection or disease he/she has recovered from.
- Both parties can Accept or Reject the request.
- The person who wants to donate his/her plasma needs to register in our application providing required information which are name, age, blood group, phone number, and location, etc.
- Patients who need plasma can also fill the form to request the plasma. Patients can directly call the donor by taking his/her contact number from the application.
- User can also search based on location they are living in using their phone's network to let them find and connect with people for plasma requirements.
- Just a single search allows anyone to reach maximum number of plasma donors in minimum possible time and that too within just 5kms from where the plasma is required.
- Also saves plasma donation history, to increase the possibility of saving lives.

1.2 Purpose

Plasma donation saves lives, and the communication between blood/plasma centers and donors plays a vital role in this. Smart apps are now considered an important communication tool, and could be best utilized in plasma donation if they are designed to fit the users' needs and preferences.

2. LITERATURE SURVEY

S NO	TITLE	Authors	Abstract	Drawbacks
1.	Plasma Donation	Neha Soni ,	The person who wants to donate his/her plasma needs	• Internet : It would

	Website using MERN stack	Software Engineering Intern at FICO | Technical Blogger	to register in our application providing required information which are name, age, blood group, phone number, and location, etc. Patients who need plasma can also fill the form to request the plasma. Patients can directly call the donor by taking his/her contact number from the application. The user can also view the total active cases, recovered cases, vaccine centres in their area, hospital location, and helpline number.	require an internet connection for the working of the website. • Auto-Verification : It cannot automatically verify the genuine

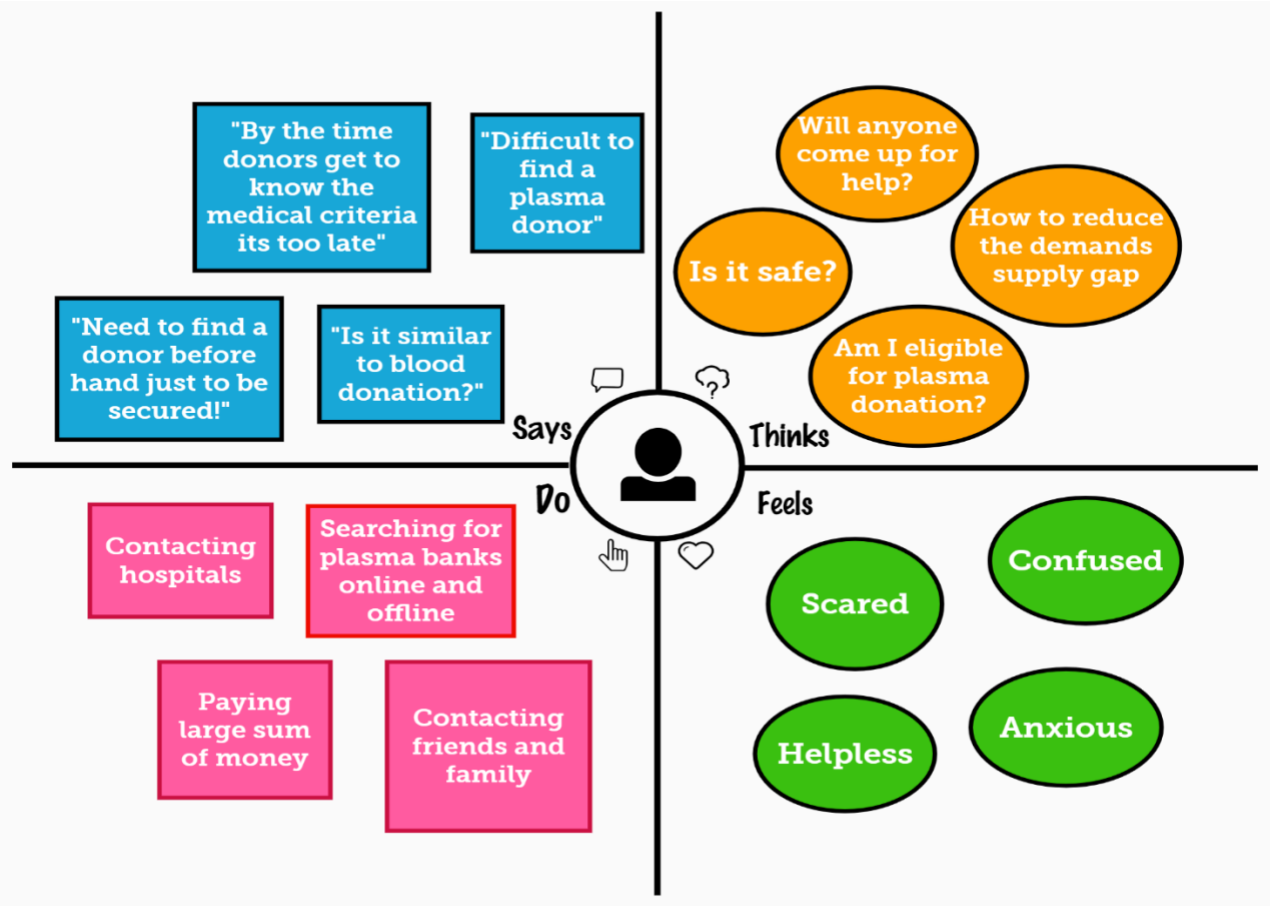
2.	Instant Plasma Donor Recipient Connector Web Application	Ripathi S Kumar V Prabhakar A	The world is suffering from COVID 19 crisis, and we haven't found any vaccine yet. But there is another scientific way from which we can help to lower the death ratio or help the COVID 19 affected person is by donating Plasma from recovered patients. With no approved antiviral treatment plan for the deadly COVID-19 infection, plasma therapy is an experimental approach to treat COVID positive patients and help them recover faster. The therapy considered to be safe and promising. If a particular person is fully recovered from COVID 19 he/she is applicable to donate their plasma. In the proposed system, donors who need to donate plasma can donate by uploading covid-19 certificate and blood bank can view donors and can raise requests to donors and the hospital can register/login and can search for plasma, they can raise requests to blood bank and can get the plasma.	• Tedious work. • Expensive. • Requires more man power. • Time Consuming
3.	Developing a plasma	Aishwarya R Gowri,	A plasma is a liquid portion of the blood, over 55% of	• It cannot auto

	donor application using Function-as-a-service in AWS	Jain University Department of MCA, computer science	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 antibodies 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, DynamoDB, 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. Situations like if the donor count is very low, it is 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.	verify user genuineness. • It requires an active internet connection.
4.	Plasma Donation App	Jenny Shersten	Motivation for further plasma collection from donors for recipients, as well as fast communication with	• Internet Connection is

			them. For both groups - always up-to-date information and the ability to follow statistics and data in the city and in the country	mandatory • Reports are not verified
5.	Plasma-Donor-App	Dheeraj Kotwani Pragathi Verma Sitam Sardar Vatsal Kesarwani Nakul Sharma Nuh Koca Harsh Rajgor	An Open-Source App which fills the gap between the patients and the Plasma Donors.	• No search filter available • Cannot login through Chrome • UI improvement in Login page

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

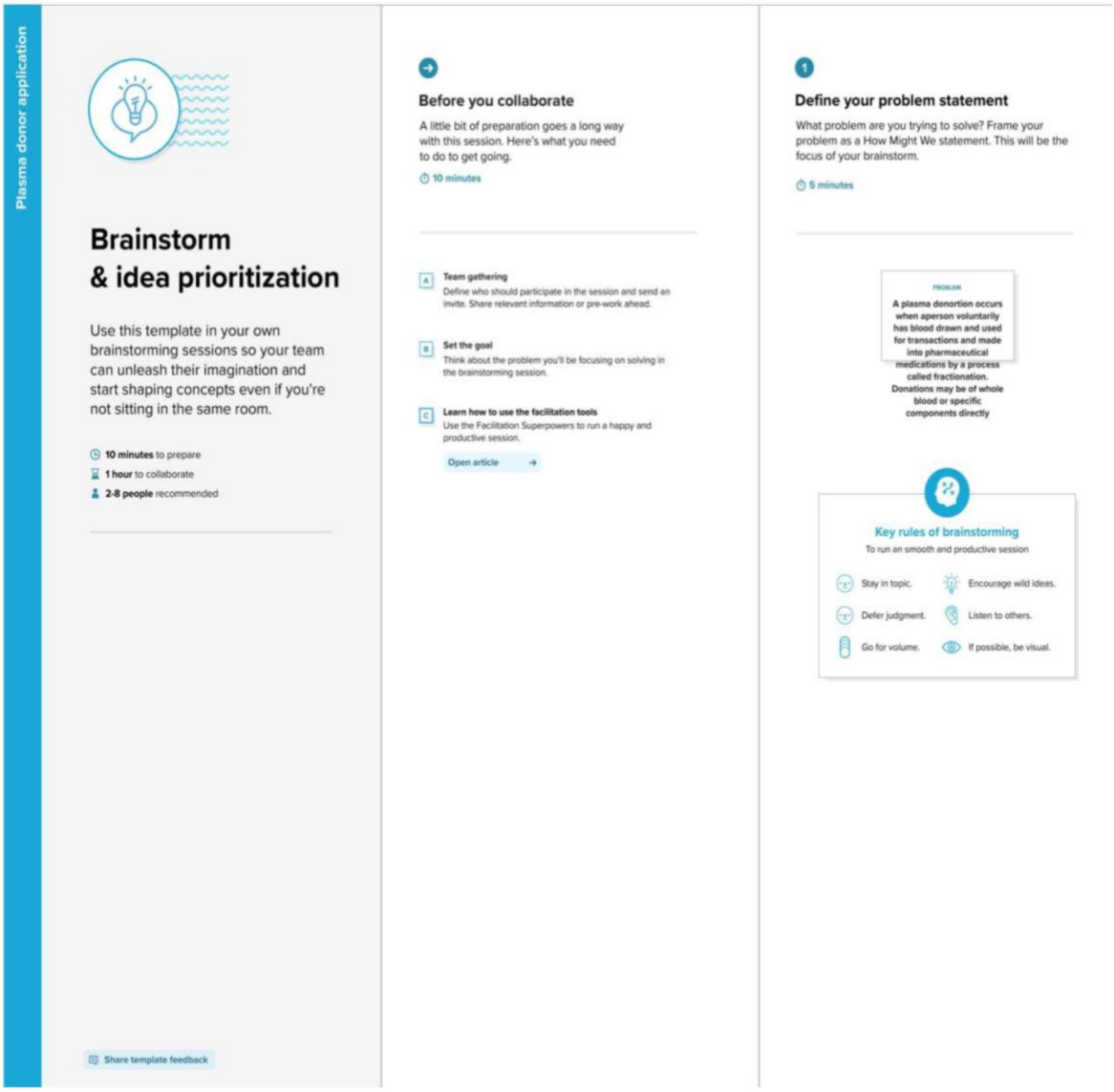


3.2 Ideation & Brainstorming

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions. Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you are not sitting in the same room.

Reference: https://www.mural.co/templates/empathy-map-canvas

Step-1: Team Gathering, Collaboration and Select the Problem Statement



Step-2: Brainstorm, Idea Listing and Grouping

Step-2: Brainstorm, Idea Listing and Grouping

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

Team lead- Harithaa S

- An application that includes all the present date available plasma donors along with their detail to communicate
- To have a track of donors nd to keep those donors during the demand
- Providing an easy end user friendly site for users
- Triggering the alert message with the patient needs blood immediately

Team Member -1 -Deepika A

- Centralized transportation system along with all the hospitals and branches
- Sending E-mail notification to the user regarding the blood needs
- Keep a record of the blood donors and send them regular message about the blood needs
- Plasma donor management system is web based system that can assist the information of blood bag during its handling in the blood bank

Team Member-2 - Narmadha D

- People who have diseases like anemia or people who have gotten into accidents
- During the COVID 19 crisis the requirement of plasma became a high of priority and the donor count has become low
- In regard to the problem faced an application is to be built whch would take the donor details, store them and inform them upon a request
- Triggering the alert messages when the patient needs blood immediately

Team Member-3 - Kamal Anand

- Plasma donor management system is a web based system that can assists the information of blood bag during its handling in the blood bank
- To have a track of donors and t0 keep those donors during the demand
- Centralized transportation system along with all the hospitals and branches
- Keep a record of the blood donors and send them regular message about the blood needs

Step-3: Idea Prioritization

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

Prediction and Analysis

- Predicting the sucess ratio of donated donors
- Predicting the ratio of new users to donate blood

Features

- E-mails and SMS alert to the users regarding the immediate blood needs
- 24*7 working of the website to help the patients

Services

- Online Website for patient who really needs a blood
- 24*7 working of the website to help the patients

Management

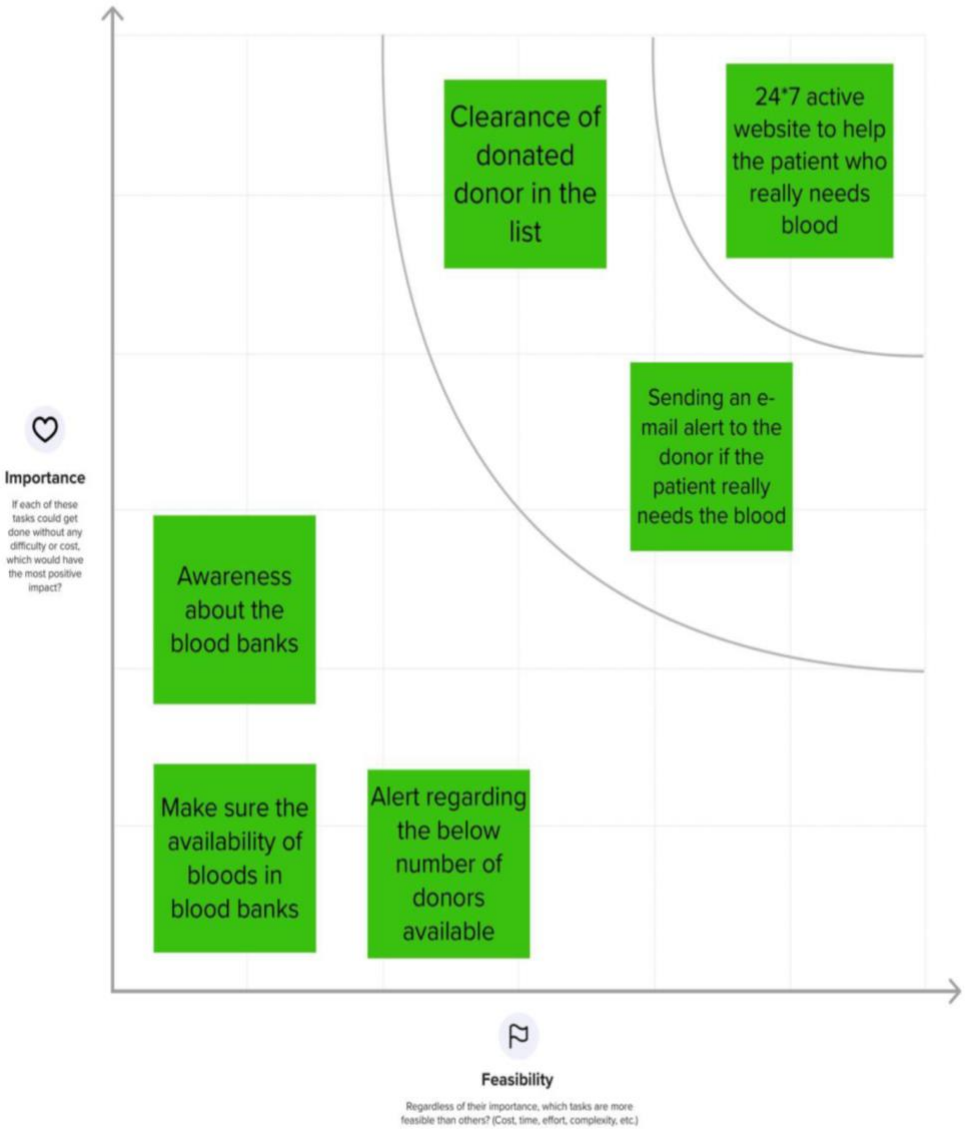
- Use feedback system management
- Managing the bloods in the blood bank shows to the users

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes



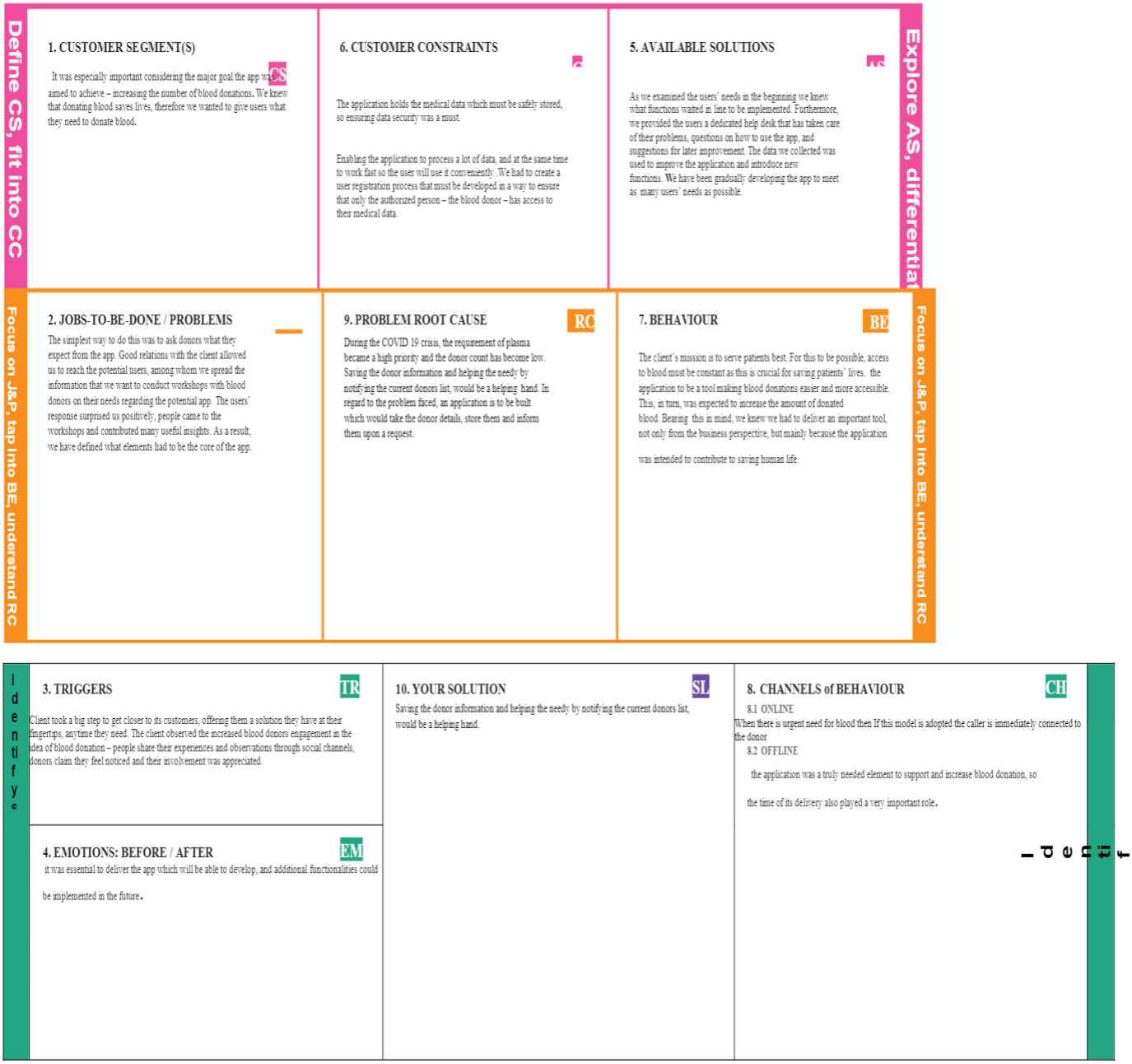
3.3 Proposed Solution

Project team shall fill the following information in proposed solution template.

S.No	Parameter	Description
1.	Problem Statement (Problem to be solved)	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.
2.	Idea / Solution description	The user interacts with the application. Registers by giving the details as a donor. The database will have all the details and if a user posts a request then the concerned blood group donors will get notified about it.
3.	Novelty / Uniqueness	The detailed objective of what donors needed from the application was predefined by the client. Our goal as an entity delivering the product was to confirm users' needs, design and develop a friendly, useful, and well-working mobile application.
4.	Social Impact / Customer Satisfaction I	It was especially important considering the major goal the app was aimed to achieve – increasing the number of blood donations. We knew that donating blood saves lives, therefore we wanted to give users what they need to donate blood.
5.	Business Model (Revenue Model)	This app will break the chain of business through blood and help the poor to find donor at free of cost. This project will help new blood banks improve their services and progress from traditional to user-friendly frameworks.

6.	Scalability of the Solution	This plasma therapy is an experimental approach to treat corona -positive patients and help them recover. This plasma therapy is considered to be safe & promising. A person who has recovered from Covid can donate his/her plasma to a person who is infected with the coronavirus.

3.4 Problem Solution fit



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form (WebApp)
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Certification	After the donor donates plasma, we will give them a certificate of appreciation and authentication.
FR-4	Statistical data	The availability of plasma is given in the page as stats, which will be helpful for the users.
FR-5	User Plasma Request	Users can request to donate plasma by filling out the request form on the page. Once the request is submitted, they will get an email
FR-6	Searching/reporting requirements	Users can use the search bar to look up information about camps and other topics.
FR-8	Virtual Assistants	A virtual assistant is a software agent that can carry out tasks or provide services on behalf of a person in response to commands or inquiries. When users enter their inquiries, the system will respond with pertinent information about plasma and details of plasma donation.

4.2 Non-Functional requirements

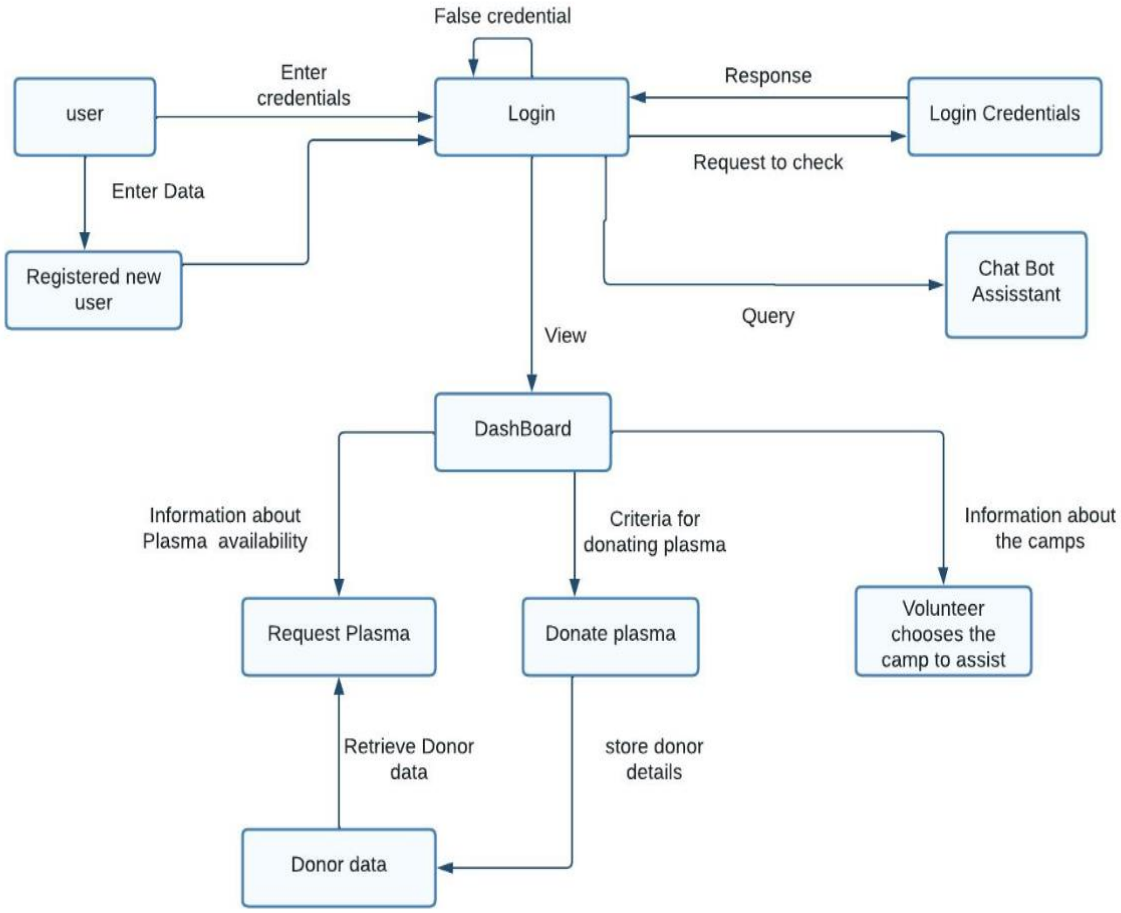
Following are the non-functional requirements of the proposed solution.

NFR No:	Non-Functional Requirement	Description
NFR-1	Usability	Must have a good looking User friendly interface.
NFR-2	Security	It must be secured with the proper username and password.
NFR-3	Reliability	The system should be made in such a way that it is reliable in its operations and for securing the sensitive details.
NFR-4	Performance	Users should have a proper Internet Connection

NFR-5	Availability	The system including the online and offline components should be available 24/7.
NFR-6	Scalability	The application has the ability to handle growing numbers of users and load without compromising on performance and causing disruptions to user experience.

5. PROJECT DESIGN

5.1 Data Flow Diagrams



5.2 Solution & Technical Architecture

Technical Architecture:

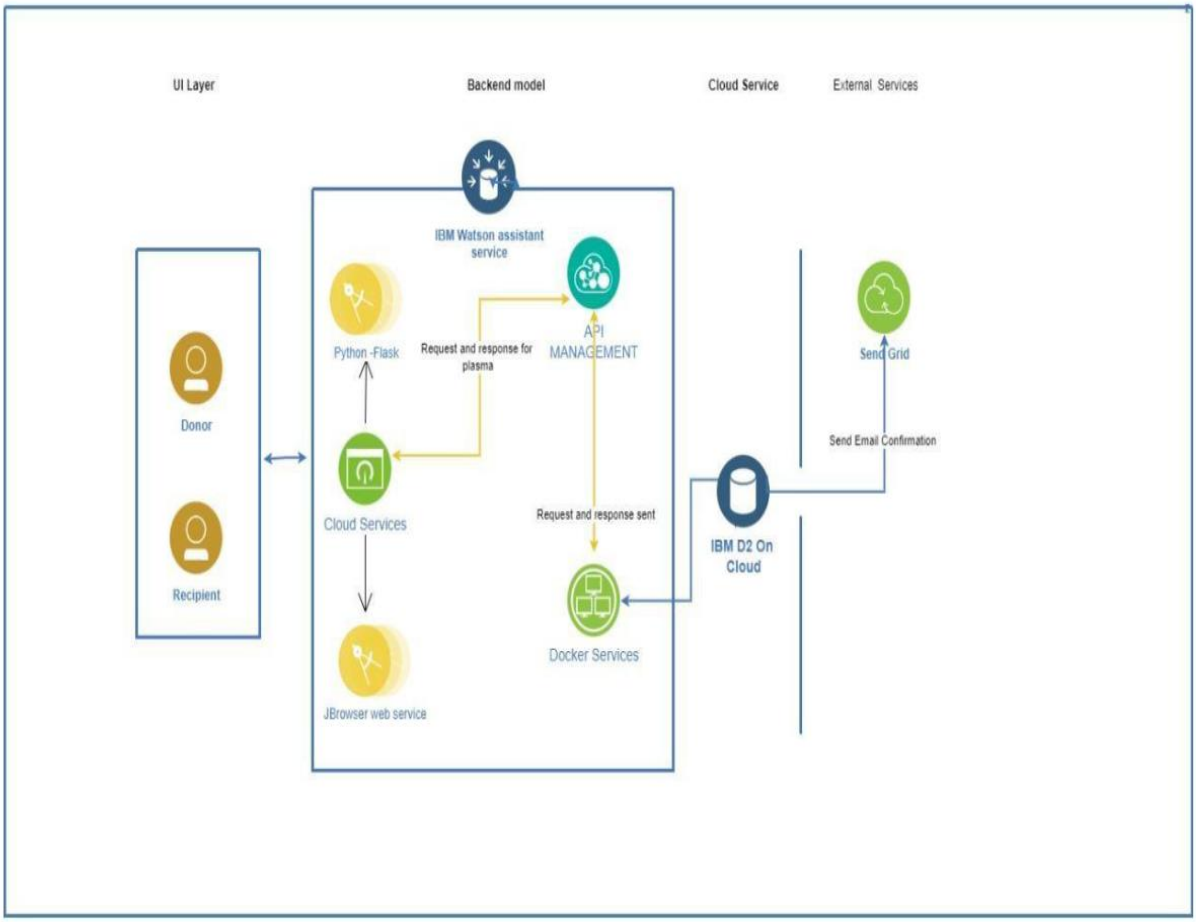


Table-1: Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	● The user creates an account or registers in the UI. ● Goes through the UI and view details	HTML, CSS, Python Flask
2.	Chatbot	● Used to clarify user queries	IBM Watson Assistant
3.	Data maintenance	For storing, maintaining, modifying and retrieving the user's details	MySQL

4.	Confirmation Email	Sending a confirmation email to users they have registered for donation and to check the availability of plasma	SendGrid
5.	Cloud Database	For storing the appointment ,donation details and user's details	IBM DB2
6.	File Storage	File storage requirements	IBM Block Storage
7.	Infrastructure (Server / Cloud)	To deploy an Application on Local System	Kubernetes

Table-2: Application Characteristics:

S. No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Python flask micro framework is used.	Python Flask
2.	Security Implementations	Mandatory Control (MAC) and Kubernetes is used.	SHA-256, Encryptions, IAM Controls, OWASP ,Kubernetes
3.	Scalable Architecture	3-Tier architecture is used.	Web Server-HTML ,CSS Application Server-Python Flask Database Server-IBM DB2
4.	Availability	Using Load Balancer to distribute network traffic across servers.	IBM Load Balancer
5.	Performance	Request and respond facility within a second. User-friendly API	IBM Content Delivery Network

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, and confirming my 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 confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Gmail	I can receive confirmation notifications through Gmail	Medium	Sprint-1
	Login	USN-4	As a user, I can log into the application by entering email & password	I can access into my User profile and view details in dashboard	High	Sprint-1
	Dashboard	USN-5	As a user,I can send the proper requests to donate and obtain plasma.	I can receive appropriate notifications through email	High	Sprint-1
Customer (Web user)	Login	USN-6	As a user,I can register and log into the application by entering email & password to view the profile	I can access into my User profile and view details in dashboard	High	Sprint-1
	Dashboard	USN-7	As a user,I can send the proper requests to donate and obtain plasma.	I can receive appropriate notifications through email	High	Sprint-1
Customer Care Executive	Application	USN-8	As a customer care executive,I can try to address user's concerns and questions	I can view and address their concerns and questions	Medium	Sprint-2
Administrator	Application	USN-9	As an administrator I can help with user-facing aspects of a website, like its appearance, navigation and use of media.	I can change the appearance and navigation in a user friendly manner	Medium	Sprint-3
		USN-10	As an administrator, I can involve working with the technical side of websites.	I can help with such as troubleshooting issues, setting up web hosts, ensuring users have access and programming servers	Medium	Sprint-1

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Chatbot	Dashboard	USN-11	In addition the Customer care executive,chatbot can try to address user's concerns and questions	I can reply to all the queries related to our application	Medium	Sprint-3

6. PROJECT PLANNING & SCHEDULING

Product Backlog, Sprint Schedule, and 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 (Harithaa S)
Sprint-1	Login	USN-2	As a admin, I can log into the application by entering email & password	9	High	Harithaa S
Sprint-1	Chatbot	USN-3	As a user I can ask query in chatbot	2	Medium	Harithaa S
Sprint-2	Confirmation	USN-4	As a user, I can receive confirmation mail.	4	Medium	Deepika A

Sprint-2	Dashboard	USN-5	As a user, I can view dashboard and select	5	Medium	Deepika A
Sprint-2	View Donor Lis	USN-6	As a user, I can view all the donor list and contact them directly	9	High	Deepika A
Sprint-2	Search Dono	USN-7	As a user, I can search for the donor	9	Medium	Narmadha D
Sprint-3	About us	USN-8	As a User, I can view the about us page which contains all contact information	5	Medium	Narmadha D
Sprint-3	Modify data	USN-9	As a admin, I can modify the User data.	9	High	Narmadha D
Sprint-3	Send mail	USN-10	As a user, I can send mail to donors using sendgrid.	9	High	Kamal anand
Sprint-3	Home page	USN-11	As a user I can view the home page and select the desired option.	9	Medium	Kamal anand
Sprint-4	Send Query	USN-12	As a user I can ask my query through email	9	Medium	Kamal anand
Sprint-4	Download data	USN-13	As a admin I can download the user data	9	High	Harithaa S

Project Tracker, Velocity & Burndown Chart:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	5 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

Velocity:

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

Velocity:

AV= 20/6= 3..333…

Sprint 1(AV)= 3.34

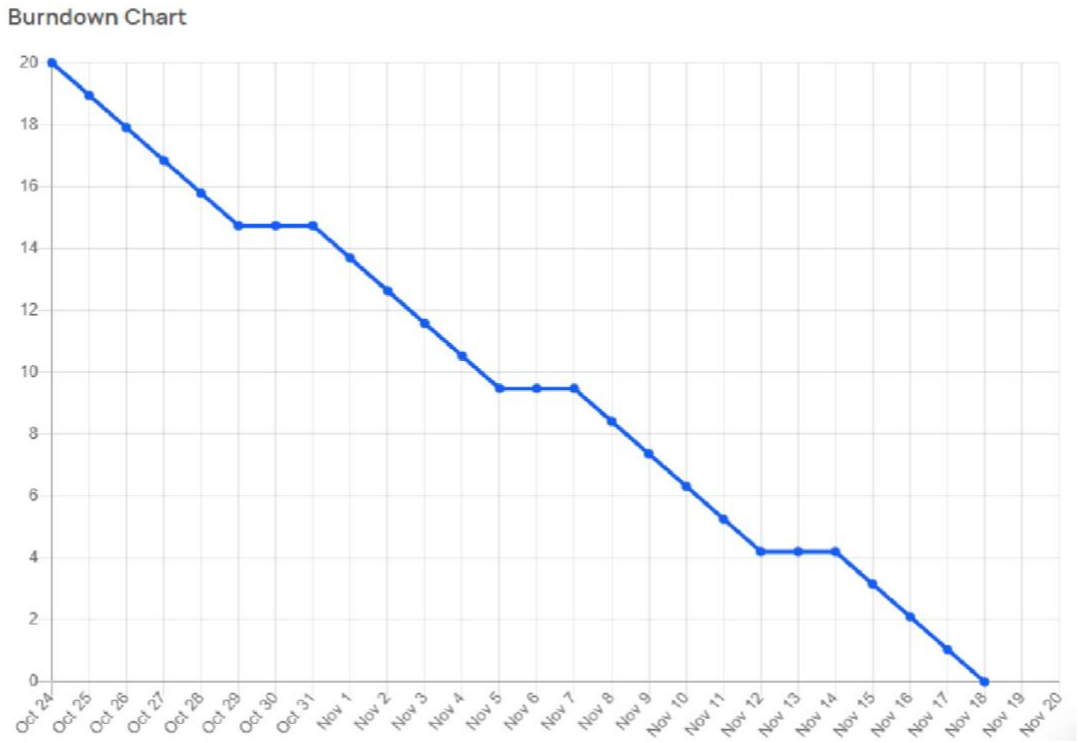
Sprint 2(AV)= 3.34

Sprint 3(AV)= 3.34

Sprint 4(AV)= 3.34

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.



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

7.1 Feature 1

#code for login and register

from distutils.log import debug

```
from flask import Flask, render_template, request, redirect, url_for, session
import ibm_db
import re

app = Flask(__name__)

app.secret_key = 'a'

conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=2f3279a5-73d1-4859-
88f0-a6c3e6b4b907.c3n41cmd0nqnrk39u98g.databases.
appdomain.cloud:30756;PORT=30756;
Security=SSL;SSLServerCertificate=Certificate.crt;UID=dfw40988;
PWD=6gng8NbeWBZg8jwW;","","")

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



@app.route('/loginpage',methods=['GET', 'POST'])
def loginpage():
    global userid
    msg = ''

    if request.method == 'POST' :
        username = request.form['username']
```

```
        password = request.form['password']
        sql = "SELECT * FROM donors WHERE username =? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,username)
        ibm_db.bind_param(stmt,2,password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print (account)
        if account:
            session['loggedin'] = True
            session['id'] = account['USERNAME']
            userid=  account['USERNAME']
            session['username'] = account['USERNAME']
            msg = 'Logged in successfully !'

            return redirect(url_for('dash'))
        else:
            msg = 'Incorrect username / password !'
    return render_template('login.html', msg = msg)

@app.route('/registration')
def home():
    return render_template('register.html')

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

```
def register():
    msg = ''
    if request.method == 'POST' :
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        phone = request.form['phone']
        city = request.form['city']
        infect = request.form['infect']
        blood = request.form['blood']
        sql = "SELECT * FROM donors WHERE username =?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,username)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = 'Account already exists !'
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = 'Invalid email address !'
        elif not re.match(r'[A-Za-z0-9]+', username):
            msg = 'name must contain only characters and numbers !'
        else:
            insert_sql = "INSERT INTO  donors VALUES (?, ?, ?, ?, ?, ?, ?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
```

```
        ibm_db.bind_param(prep_stmt, 1, username)
        ibm_db.bind_param(prep_stmt, 2, email)
        ibm_db.bind_param(prep_stmt, 3, password)
        ibm_db.bind_param(prep_stmt, 4, city)
        ibm_db.bind_param(prep_stmt, 5, infect)
        ibm_db.bind_param(prep_stmt, 6, blood)
        ibm_db.bind_param(prep_stmt, 7, phone)

        ibm_db.execute(prep_stmt)
        msg = 'You have successfully registered !'
    elif request.method == 'POST':
        msg = 'Please fill out the form !'
    return render_template('register.html', msg = msg)
```

7.2 Feature 2

```
* {
    font-family: 'Alegreya', serif !important;
}

/* // Small devices (landscape phones, 576px and up) */
@media (min-width: 576px) {

}
```

```
/* // Medium devices (tablets, 768px and up) */
@media (min-width: 768px) {

}

/* // Large devices (desktops, 992px and up) */
@media (min-width: 992px) {
    .navbar {
        padding-top:15px;
        padding-bottom:15px;
        background-color: white;
    }
    .navbar-brand{
        padding-left: 5px;
    }
    .navbar-nav {
        margin-left: 30px;
    }
    .nav-item {
        padding-left:5px;
    }
    .donate-sponsor{
        margin-right:10px;
    }
    #donate,#sponsor{
```

```
    margin:5px;
    padding: 5px 15px 5px 15px;
}
.homepage-header{
    background: url('../img/group.JPG');
    background-size: cover;
    background-position: center top;
    padding: 0;
    position: relative;
    width: 100%;
    overflow: hidden;
    display: -webkit-flex;
    display: -ms-flexbox;
    display: flex;
    height: 85vh;
}
.home-for-children {
    background-color: #ffeeba;
    margin-top:0px;
    padding-top:10px;
    padding-bottom:30px;
}
.home-for-children h2 {
    line-height: 2.5rem !important;
    letter-spacing: 3px;
```

```
    font-weight: 600;
}
.home-for-children h5 {
    line-height: 1.8rem;
}
.home-for-children .btn-success {
    padding: 8px 25px;
    font-size: large;
}
/* our focus */
.our-focus .container {
    margin-top:3rem !important;
}

.our-focus .row{
    margin-top:1.5rem !important;
}
.our-focus .card {
    border: none !important;
}
.our-focus #focus-first {
    margin-right:80px;
}
.our-focus #focus-second {
    margin-right:90px;
```

```
}
/* media */
.media .container{
    margin-top: 3rem !important;
}
.media .row{
    margin-top: 1.5rem !important;
}
/* footer */
.row-initiative {
    margin-top:10px
}
.site-footer {
    margin-top:30px;
    line-height:24px;
}
.footer-links {
    padding-left:0;
    list-style: none;
}
.footer-links li {
    display: block;
}
.footer-links.inline li {
    display: inline-block;
```

```
}
.footer-links li a{
    color: black;
}
.footer-logo img{
    width: 100px;
}
.social-icons {
    text-align: right;
    margin-left:50px;
}
.social-icons li{
    list-style: none;
    display: inline-block;
}
.social-icons li a {
    border-radius: 50%;
    margin-left:10px;
}
input {
    padding:10px 20px;
  }
  #button {
    border:none;
    padding:10px 20px;
```

```
        border-radius:10px;
        animation:pulse 3s infinite ease-out;
      }
}

/* // X-Large devices (large desktops, 1200px and up) */
@media (min-width: 1200px) {
    .navbar-brand{
        padding-left: 10px;
    }
}

/* // XX-Large devices (larger desktops, 1400px and up) */
@media (min-width: 1400px) {
    .navbar-brand{
        padding-left: 15px;
    }
}
```

7.3 Database Schema

```
from flask import Flask, render_template, request, redirect, url_for, session
import ibm_db
import re

app = Flask(_name_)
```

```
hostname = '2f3279a5-73d1-4859-88f0-a6c3e6b4b907.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud'
uid = 'dfw40988'
pwd = '6gng8NbeWBZg8jwW'
driver = "{IBM DB2 ODBC DRIVER}"
db_name = 'Bludb'
port = '30756'
protocol = 'TCPIP'
cert = "Certificate.crt"
dsn = (
    "DATABASE ={0};"
    "HOSTNAME ={1};"
    "PORT ={2};"
    "UID ={3};"
    "SECURITY=SSL;"
    "PROTOCOL={4};"
    "PWD ={6};"
).format(db_name, hostname, port, uid, protocol, cert, pwd)
connection = ibm_db.connect(dsn, "", "")
print()
# query = "SELECT username FROM USER1 WHERE username=?"
# stmt = ibm_db.prepare(connection, query)
# ibm_db.bind_param(stmt, 1, username)
# ibm_db.execute(stmt)
# username = ibm_db.fetch_assoc(stmt)
# print(username)
```

```
app.secret_key = 'a'



@app.route('/', methods=['GET', 'POST'])
@app.route('/register', methods=['GET', 'POST'])
def register():
    msg = " "
    if request.method == 'POST':
        username = request.form['username']
        email_id = request.form['email_id']
        phone_no = request.form['phone_no']
        password = request.form['password']
        query = "SELECT * FROM USER1 WHERE username=?;"
        stmt = ibm_db.prepare(connection, query)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        if (account):

            msg = "Account already exists!"
            return render_template('register.html', msg=msg)
        # elif not re.match(r'[^@]+@[^@]+\.[^@]+', email_id):
        #     msg = "Invalid email addres"
        # elif not re.match(r'[A-Za-z0-9+', username):
        #     msg = "Name must contain only characters and numbers"
```

```
        else:
            query = "INSERT INTO USER1 values(?,?,?,?)"
            stmt = ibm_db.prepare(connection, query)
            ibm_db.bind_param(stmt, 1, username)
            ibm_db.bind_param(stmt, 2, email_id)
            ibm_db.bind_param(stmt, 3, phone_no)
            ibm_db.bind_param(stmt, 4, password)
            ibm_db.execute(stmt)
            msg = 'You have successfully Logged In!!'
            return render_template('login.html', msg=msg)
    else:
        msg = 'PLEASE FILL OUT OF THE FORM'
        return render_template('register.html', msg=msg)



@app.route('/login', methods=['GET', 'POST'])
def login():
    global userid
    msg = ' '
    if request.method == "POST":
        username = request.form['username']
        password = request.form['password']
        query = "select * from user1 where username=? and password=?"
        stmt = ibm_db.prepare(connection, query)
        ibm_db.bind_param(stmt, 1, username)
```

```
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['Loggedin'] = True
            session['id'] = account['USERNAME']
            session['username'] = account['USERNAME']
            msg = 'Logged in Successfully'
            return render_template('welcome.html', msg=msg, username=str.upper(username))
        else:
            msg = 'Incorrect Username or Password'
            return render_template('login.html', msg=msg)
    else:
        msg = 'PLEASE FILL OUT OF THE FORM'
        return render_template('login.html', msg=msg)



@app.route('/welcome', methods=['GET', 'POST'])
def welcome():
    if request.method == 'POST':
        username = request.form['username']
        print(username)
        return render_template('welcome.html', username=username)
    else:
```

```
    return render_template('welcome.html', username=username)


if _name_ == "_main_":
    app.run(debug=True)
    app.run(host='0.0.0.0')
```

8. TESTING

8.1 Test Cases

Test case ID	Feature Type	Component	Test Scenario	Pre-Requisite	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Comments	TC for Automation(Y/N)	BUG ID	Executed By
LoginPage_TC_001	Functional	Home Page	Verify user is able to see the Login/Signup page whenever get into the application		1.Enter URL and click go 2.Click on the login/signup page 3.Verify login/Signup by entering the details	-	Login/Signup page should display	Working as expected	Pass				kosalaraman
LoginPage_TC_002	UI	Home Page	Verify the UI elements in Login/Signup page		1.Enter URL and click go 2.Click on Login/signup and get into next respective page. 3.Verify login/Signup page with below UI elements: a.email text box b.password text box c.Login button d.New User? Create account link	-	Application should show below UI elements: a.email text box b.password text box c.Login button with orange colour d.New customer? Create account link	Working as expected	Pass				Kishore kumar
LoginPage_TC_003	Functional	Home page	Verify user is able to log into application with Valid credentials		1.Enter URL and click go 2.Click on login button 3.Enter Valid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: demo@gmail.com password: 12345678	User should navigate to Donar/Recipient requesting page	Working as expected	pass				Madhankumar
LoginPage_TC_004	Functional	Login page	Verify user is able to log into application with InValid credentials		1.Enter URL and click go 2.Click on login button 3.Enter Valid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: demo@gmail password: Testing123	Application should show 'Incorrect email or password' validation message	Working as expected	pass				Bharath
LoginPage_TC_005	Functional	Login page	Verify Admin is able to log into application with Valid credentials		1.Enter URL and click go 2.Click on login button 3.Enter Valid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: admin@gmail.com password: admin@m	Admin should navigate to Donar/Recipient requesting page	Working as expected	pass				Kishore kumar
LoginPage_TC_006	Functional	Login page	Verify Admin is able to log into application with InValid credentials		1.Enter URL(http://shopenaw.com/) and click go 2.Click on My Account dropdown button 3.Enter InValid username/email in Email text box 4.Enter Invalid password in password text box 5.Click on login button	Username: admin@gmail.com password: Admin@	Application should show 'Incorrect email or password' validation message	Working as expected	pass				kosalaraman

8.2 User Acceptance Testing

1. Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the [ProductName] project at the time of the release to User Acceptance Testing (UAT).

2. Defect Analysis

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity1	Severity2	Severity3	Severity4	Subtotal
By Design	10	4	2	4	20
Duplicate	1	0	1	0	2
External	2	2	1	1	6
Fixed	4	1	1	10	16
Not Reproduced	0	0	0	0	0
Skipped	1	1	0	1	3
Won't Fix	0	2	2	0	4
Totals	18	10	7	16	51

3. Test Case Analysis

This report shows the number of test cases that have passed, failed, and untested.

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	9	0	0	9
Client Application	10	0	0	10
Security	1	0	0	1
Outsource Shipping	0	0	0	0
Exception Reporting	9	0	0	9
Final Report Output	9	0	0	9
Version Control	1	0	0	1

9. RESULTS

9.1 Performance Metrics

• **Formal code metrics** - Such as Lines of Code (LOC), code complexity, Instruction Path Length, etc. In modern development environments, these are considered less useful.

• **Developer productivity metrics -** Such as active days, assignment scope, efficiency and code churn. These metrics can help you understand how much time and work developers are investing in a software project.

• **Agile process metrics -** Such as lead time, cycle time and velocity. They measure the progress of a dev team in producing working, shipping-quality software features.

• **Operational metrics -** Such as Mean Time Between Failures (MTBF) and Mean Time to Recover (MTTR). This checks how software is running in production and how effective operations staff are at maintaining it.

• **Test metrics -** Such as code coverage, percent of automated tests, and defects in production. This measures how comprehensively a system is tested, which should be correlated with software quality.

• **Customer satisfaction -** Such as Net Promoter Score (NPS), Customer Effort Score (CES) and Customer Satisfaction Score (CSAT). The ultimate measurement of how customers experience the software and their interaction with the software vendor.

10. ADVANTAGES & DISADVANTAGES

Advantages

• **Speed:** This website is fast and offers great accuracy as compared to manual registered keeping.

• **Maintenance:** Less maintenance is required

• **User Friendly:** It is very easy to use and understand. It is easily workable and accessible for everyone.

• **Fast Results:** It would help you to provide plasma donors easily depending upon the availability of it.

Disadvantages

• **Internet:** It would require an internet connection for the working of the website.

• **Auto-Verification:** It cannot automatically verify the genuine users.

11. CONCLUSION

The efficient way of finding plasma donor for the infected people is implemented using the plasma donor Application.

12. FUTURE SCOPE

The sole purpose of this project is to develop a computer system that will link all donors, control a plasma transfusion service and create a database to hold data on stocks of plasma in each area. Furthermore, people will be able to see which patients need plasma supplies via the android application.

13. APPENDIX

Source Code

Aim: AS a user, I can register and make request for plasma donation.

```
from distutils.log import debug

from sendgridmail import sendmail

from flask import Flask, render_template, request, redirect, url_for, session

import ibm_db
```

```
import re
import os
from dotenv import load_dotenv

load_dotenv()



app = Flask(__name__)

app.secret_key = 'a'

conn=ibm_db.connect(os.getenv('DB_KEY'),"","")

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



@app.route('/loginpage',methods=['GET', 'POST'])
def loginpage():
    global userid
    msg = ''

    if request.method == 'POST' :
```

```
        username = request.form['username']
        password = request.form['password']
        sql = "SELECT * FROM donors WHERE username =? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,username)
        ibm_db.bind_param(stmt,2,password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print (account)
        if account:
            session['loggedin'] = True
            session['id'] = account['USERNAME']
            userid=  account['USERNAME']
            session['username'] = account['USERNAME']
            msg = 'Logged in successfully !'
            sendmail(account['EMAIL'],'Plasma donor App login','You are successfully
logged in!')
            return redirect(url_for('dash'))
        else:
            msg = 'Incorrect username / password !'
    return render_template('login.html', msg = msg)


@app.route('/registration')
def home():
    return render_template('register.html')
```

```
@app.route('/register',methods=['GET', 'POST'])
def register():
    msg = ''
    if request.method == 'POST' :
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        phone = request.form['phone']
        city = request.form['city']
        infect = request.form['infect']
        blood = request.form['blood']
        sql = "SELECT * FROM donors WHERE username =?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,username)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = 'Account already exists !'
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = 'Invalid email address !'
        elif not re.match(r'[A-Za-z0-9]+', username):
            msg = 'name must contain only characters and numbers !'
        else:
            insert_sql = "INSERT INTO  donors VALUES (?, ?, ?, ?, ?, ?, ?)"
```

```
        prep_stmt = ibm_db.prepare(conn, insert_sql)
        ibm_db.bind_param(prep_stmt, 1, username)
        ibm_db.bind_param(prep_stmt, 2, email)
        ibm_db.bind_param(prep_stmt, 3, password)
        ibm_db.bind_param(prep_stmt, 4, city)
        ibm_db.bind_param(prep_stmt, 5, infect)
        ibm_db.bind_param(prep_stmt, 6, blood)
        ibm_db.bind_param(prep_stmt, 7, phone)

        ibm_db.execute(prep_stmt)
        msg = 'You have successfully registered !'
        sendmail(email,'Plasma donor App Registration','You are successfully Registered {}!'.format(username))

    elif request.method == 'POST':
        msg = 'Please fill out the form !'
    return render_template('register.html', msg = msg)

@app.route('/dashboard')
def dash():
    if session['loggedin'] == True:
        sql = "SELECT COUNT(*), (SELECT COUNT(*) FROM DONORS WHERE blood= 'O Positive'), (SELECT COUNT(*) FROM DONORS WHERE blood='A Positive'), (SELECT COUNT(*) FROM DONORS WHERE blood='B Positive'), (SELECT COUNT(*) FROM DONORS WHERE blood='AB Positive'), (SELECT COUNT(*) FROM DONORS WHERE blood='O Negative'), (SELECT COUNT(*) FROM DONORS WHERE blood='A Negative'), (SELECT COUNT(*) FROM
```

```
DONORS WHERE blood='B Negative'), (SELECT COUNT(*) FROM DONORS
WHERE blood='AB Negative') FROM donors"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        return render_template('dashboard.html',b=account)
    else:
        msg = 'Please login!'
        return render_template('login.html', msg = msg)

@app.route('/requester')
def requester():
    if session['loggedin'] == True:
        return render_template('request.html')
    else:
        msg = 'Please login!'
        return render_template('login.html', msg = msg)

@app.route('/requested',methods=['POST'])
def requested():
    bloodgrp = request.form['bloodgrp']
    address = request.form['address']
    name=  request.form['name']
    email=  request.form['email']
    phone= request.form['phone']
```

```
    insert_sql = "INSERT INTO  requested VALUES (?, ?, ?, ?, ?)"
    prep_stmt = ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prep_stmt, 1, bloodgrp)
    ibm_db.bind_param(prep_stmt, 2, address)
    ibm_db.bind_param(prep_stmt, 3, name)
    ibm_db.bind_param(prep_stmt, 4, email)
    ibm_db.bind_param(prep_stmt, 5, phone)
    ibm_db.execute(prep_stmt)
    sendmail(email,'Plasma donor App plasma request','Your request for plasma is recieved.')
    return render_template('request.html', pred="Your request is sent to the concerned people.")



@app.route('/logout')


def logout():
   session.pop('loggedin', None)
   session.pop('id', None)
   session.pop('username', None)
   return render_template('login.html')


if __name__ == '__main__':
   app.run(host='0.0.0.0',debug='TRUE')
```

GitHub & Project Demo Link

GitHub Link

https://github.com/IBM-EPBL/IBM-EPBL-IBM-Project-21309-1659777422.git

Project Demo Link

https://clipchamp.com/watch/P6MdF12mEHg