IBM NALAIYATHIRAN PROJECT

Project Title	**PLASMA DONAR APPLICATION**
Team ID	**PNT2022TMID11415**
Batch no.	**B11-5A1E**
Team members	**A.Niranjana(910619104053)** **N.Pradhiksha(910619104055)** **K.R.Priyadharshini(910619104060)** **K.G.Priyadharshini(910619104059)** **K.S.Sujitha(910619104090)**

TABLE OF CONTENTS

1. INTRODUCTION

1.1 PROJECT OVERVIEW

A 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 antibodies that fights the infection. In this project plasma donor application is being developed by using IBM services. With the help of these IBM 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 where 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. This helps to reduce the additional infrastructural cost and users can access technology services such as power, storage, compute, database, networking, analytics and also intelligence over the internet in order to offer flexible, innovation, and economies of scale.

1.2 PURPOSE

The main purpose 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.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

Conventionally, when a patient needs plasma, he/she has to contact a plasma donor bank or a donor in their circle, family, and friends. However, it is difficult to find suitable donor within a limited group of people in a given time. In addition, there is no guarantee that plasma banks will have compatible amount of plasma in stock.

2.2 REFERENCES

- Convalescent Plasma Therapy: Data driven approach for finding the Best Plasma Donors
  Published in**:** 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS).

- mHealth: Blood donation application using android smartphone
  Published in**:** 2016 Sixth International Conference on Digital Information and Communication Technology and its Applications (DICTAP).

- Nearest Blood & Plasma Donor Finding: A Machine Learning Approach

Published in:2020 23rd International Conference on Computer and Information Technology (ICCIT) .

- Automated online Blood bank database

  Published in: 2012 Annual IEEE India Conference (INDICON).

2.3 PROBLEM STATEMENT DEFINITION

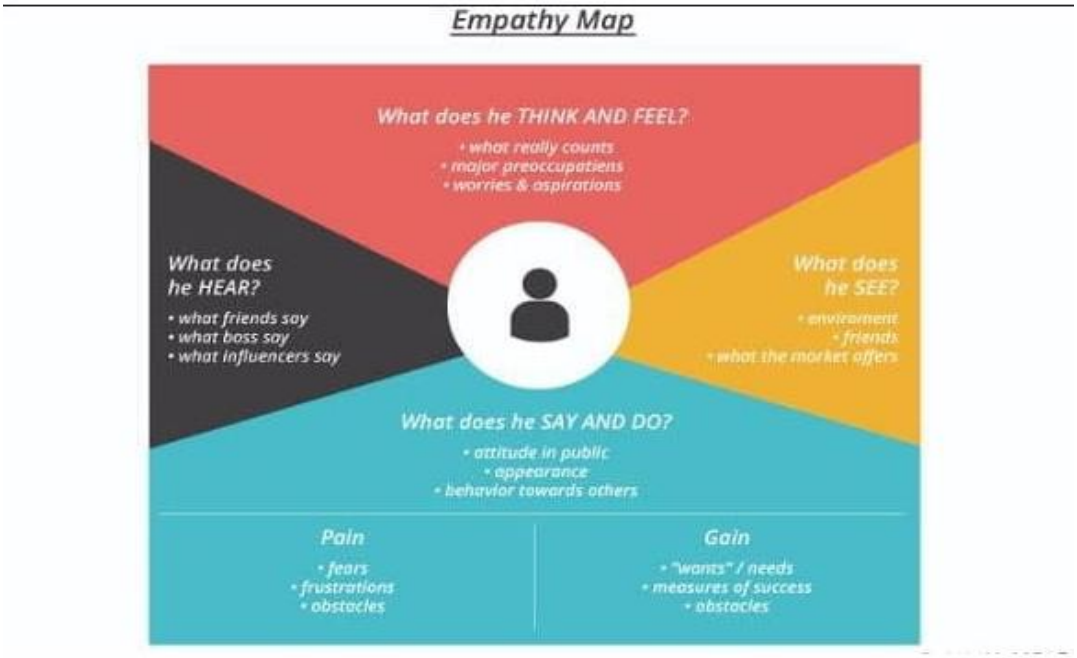
A problem statement is a concise description of the problem or issues a project seeks to address. The problem statement identifies the current state, the desired future state and any gaps between the two. A problem statement is an important communication tool that can help ensure everyone working on a project knows what the problem they need to address is and why the project is important.

3. IDEATION & PROPOSED SOLUTION

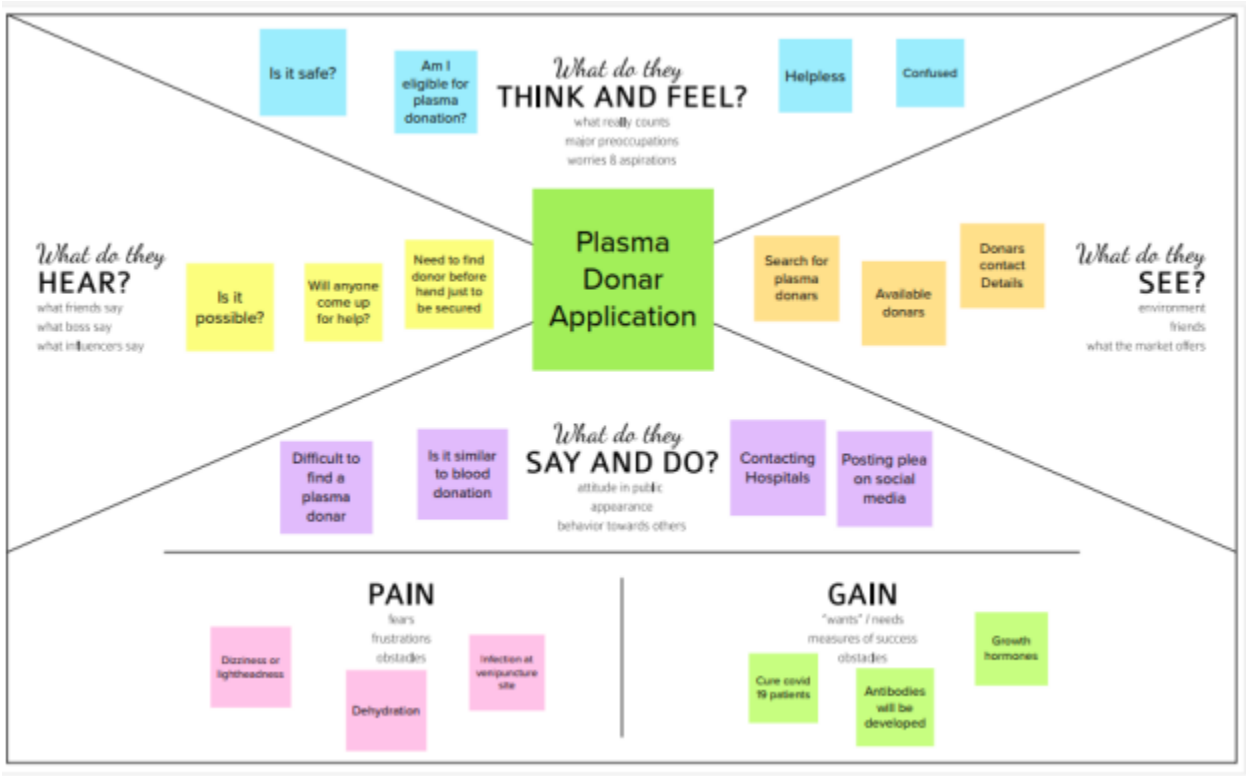
3.1 EMPATHY MAP CANVAS

An empathy map is a collaborative tool teams can use to gain a deeper insight into their customers. Much like a user persona, an empathy map can represent a group of users, such as a customer segment. The empathy map was originally created by Dave Gray and has gained much popularity within the agile community. An Empathy Map is just one tool that can help you empathise and synthesise your observations from the research phase, and draw out unexpected insights about your user's needs. An Empathy Map allows us to sum up our learning from engagements with people in the field of design research

Example:



Example: Plasma Donor Application

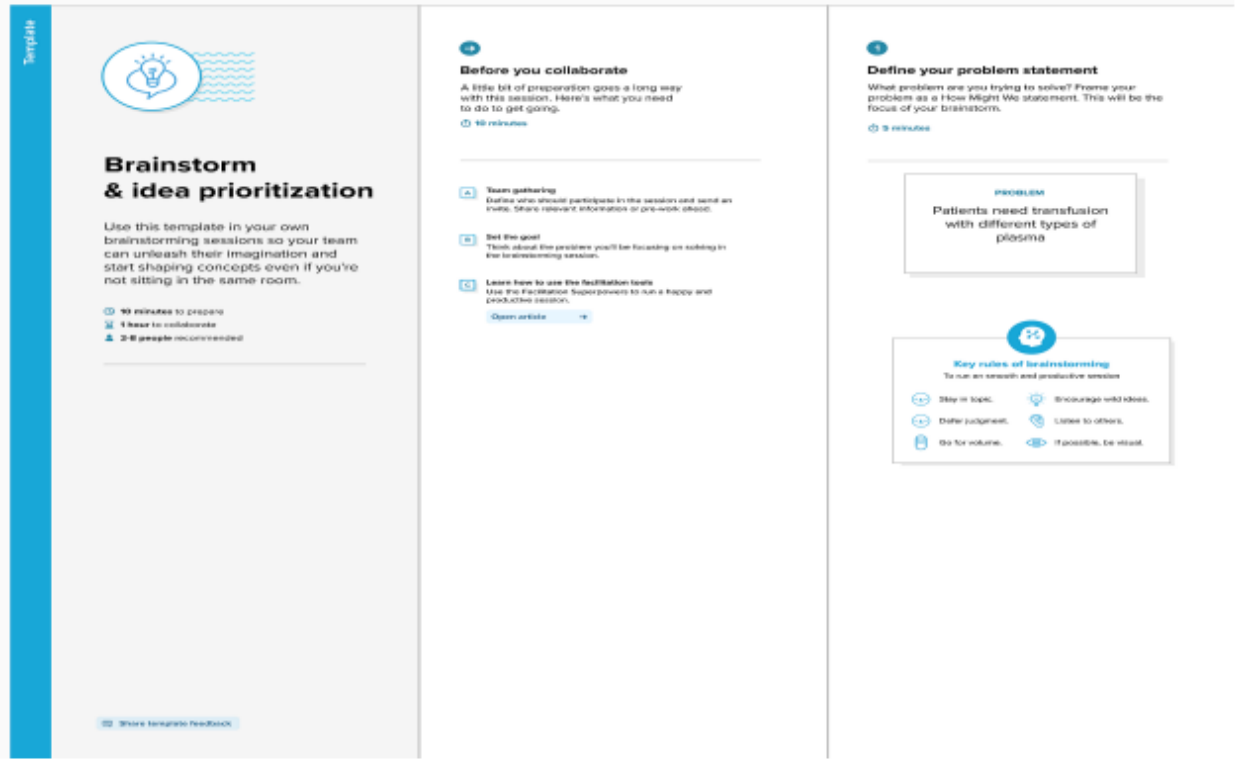


3.2 IDEATION & BRAINSTORMING

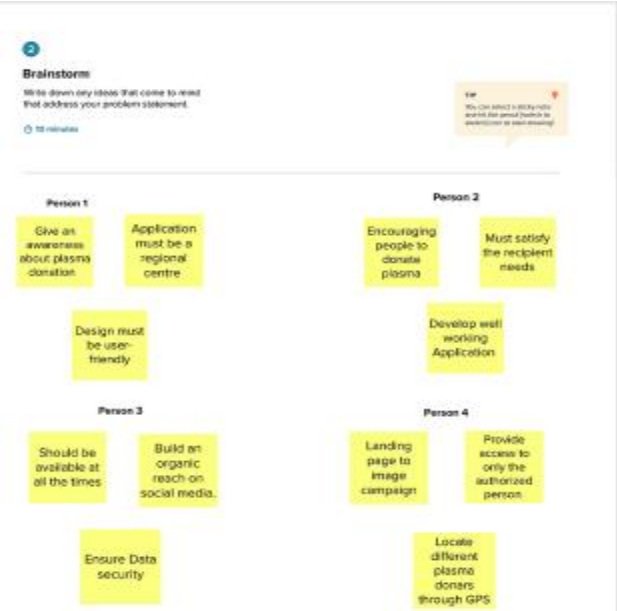
Brainstorming is a method design teams use to generate ideas to solve clearly defined design problems. In controlled conditions and a free-thinking environment, teams approach a problem by such means as “How Might We” questions. They produce a vast array of ideas and draw links between them to find potential solutions. 1. Set a time limit – Depending on the problem’s complexity, 15–60 minutes is normal. 2. Begin with a target problem/brief – Members should approach this sharply defined question, plan or goal and stay on topic. 3. Refrain from judgment/criticism – No-one should be negative about any idea. 4. Encourage weird and wacky ideas – Further to the ban on killer phrases like “too expensive”, keep the floodgates open so everyone feels free to blurt out ideas. 5. Aim for quantity – Remember, “quantity breeds quality”. The

sifting-and-sorting process comes later. 6. Build on others' ideas – It's a process of association where members expand on others' notions and reach new insights, allowing these ideas to trigger their own. 7. Stay visual – Diagrams and Post-Its help bring ideas to life and help others see things in different ways. 8. Allow one conversation at a time – To arrive at concrete results, it's essential to keep on track this way and show respect for everyone's ideas.

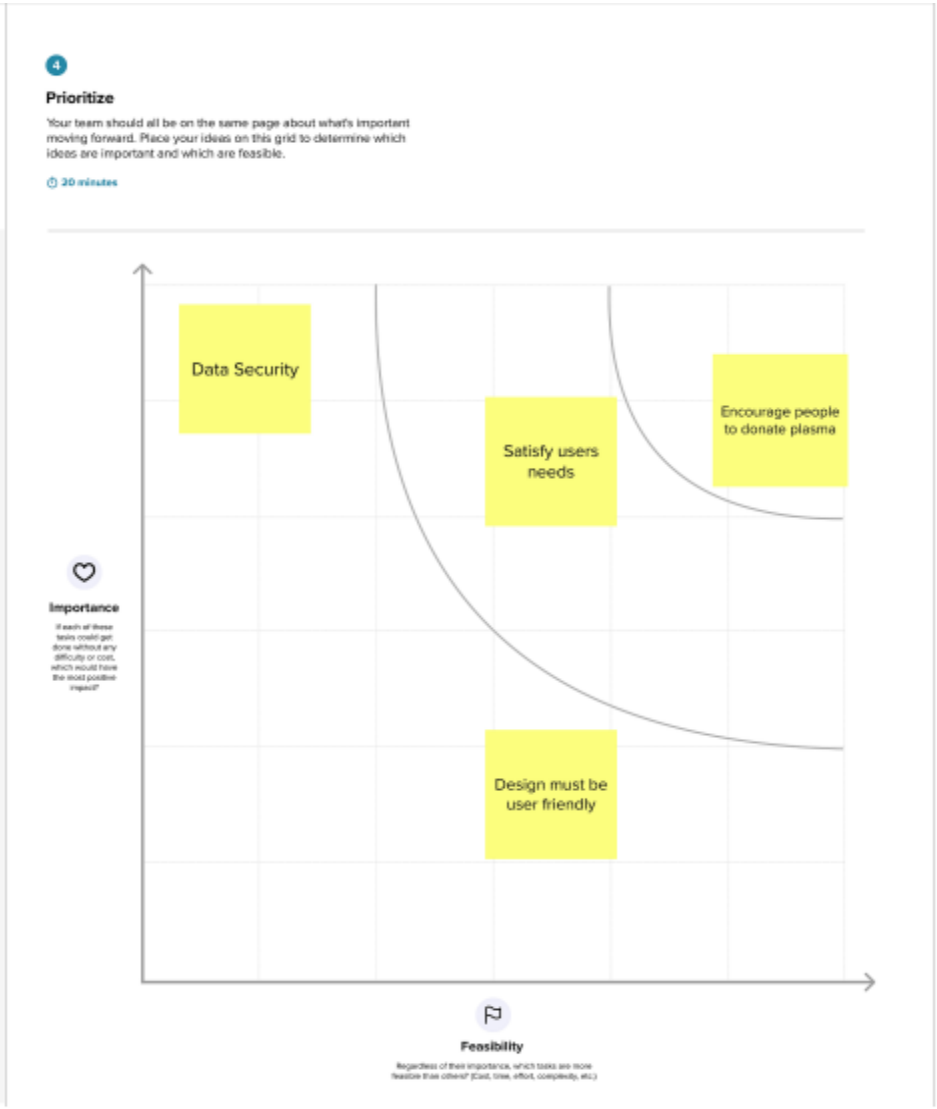
Step-1: Team Gathering, Collaboration and Select the Problem Statement



Step-2: Brainstorm, Idea Listing and Grouping



Step-3: Idea Prioritization

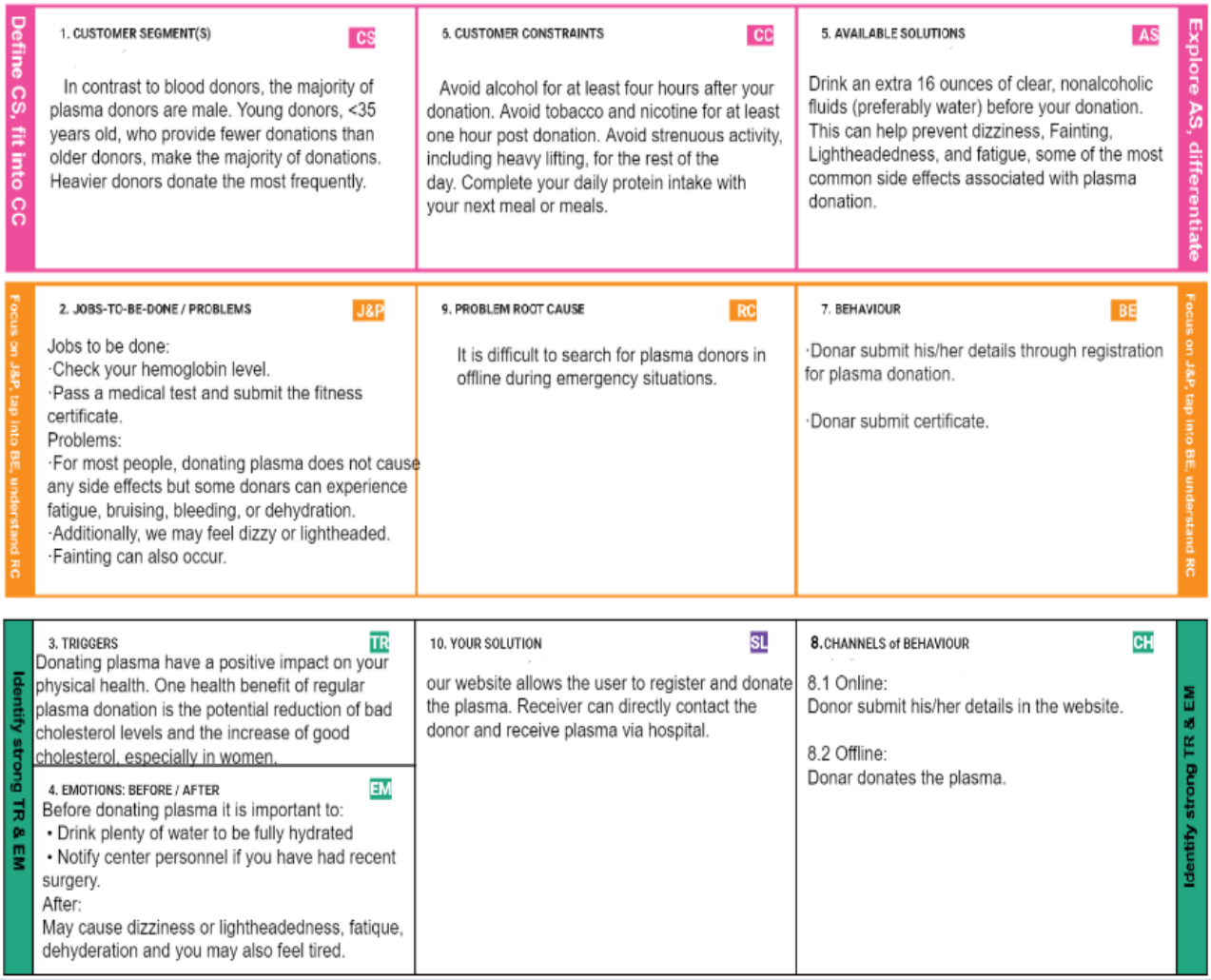


3.3 PROPSED SOLUTION

S.NO	PARAMETER	DESCRIPTION
1	Problem Statement (Problem to be solved)	Patients need transfusion with different types of plasma.
2	Idea / Solution description	International collaboration to supply different types of plasma.

3	Novelty / Uniqueness	● When we donate plasma, we are also donating unique antibodies. ● Our plasma can create a super army of lifesaving germ-fighters. This super weapon can help people with immune deficiencies.
4	Social Impact / Customer Satisfaction	Plasma donation is not simply an "individual fact" but a real "social action" characterized by systems of interpretation of reality based on individual-society relations.
5	Business Model (Revenue Model)	Plasma donar are paid for each donation. The plasma donar organization acts a bridge between the hospital and donar and get commission from the hospital for each donation.
6	Scalability of the Solution	User can locate different volunteer plasma donars and blood banks in their locality through GPS and then request for the blood in case of emergency.

3.4 PROBLEM SOLUTION FIT



4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

FR. No	Functional Requirement(Epic)	Sub Requirment(Story / Sub-Task)
FR-1	User Registration	This allows healthy public to register as volunteer donor
FR-2	Plasma Stock Management	The Plasma bank staffs can manage the plasma stock starting from the plasma collection to plasma screening, processing, storage through this system.
FR-3	Reporting	The system is able to generate predefined rep orts such as the list of donors, recipients, staffs in the bank.
FR-4	SendGrid	Sends notification to the donor according to the required blood group.
FR-5	New Releases	When a new/update version of the webapplication is released, the appearance will be automatically appears when the user access the web-application.
FR-6	Request Blood	User able to request for blood at emergency situation.

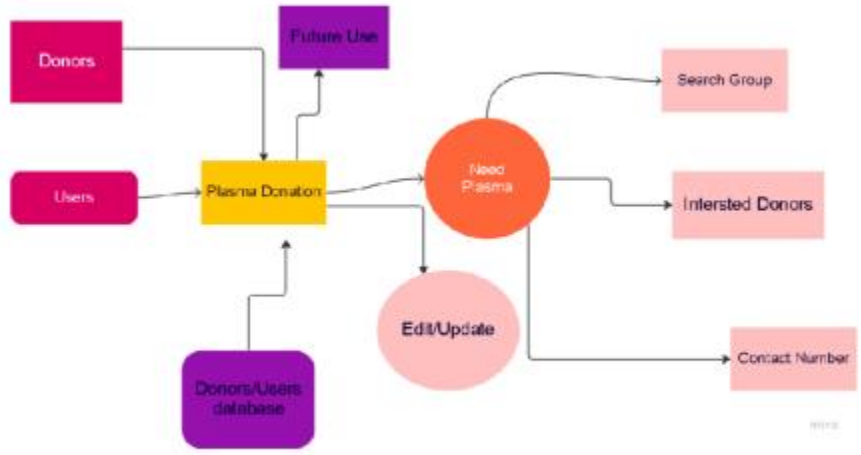
4.2 NON-FUNCTIONAL REQUIREMENTS

FR. No	Non-Functional Requirement	Description
NFR-1	Usability	The cost of the Plasma Units are standardized.
NFR-2	Security	The system automatically logout the customers after a period of inactivity.
NFR-3	Reliability	If there is extensive damage to a wide portion of the database, the recovery method restores a past copy of the database that was backed up to archival storage.
NFR-4	Maintainability	The Plasma Inventory Manager Maintain correct records of the Plasma Inventory Stock.
NFR-5	Availability	The system including the offline and online components should be available 24/7.
NFR-6	Scalability	Requirements of website extensibility in case there is a need to add new functional requirements.

5. PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS

A data flow diagram (DFD) is a graphical or visual representation using a standardized set of symbols and notations to describe a business's operations through data movement. They are often elements of a formal methodology such as Structured Systems Analysis and Design Method (SSADM).



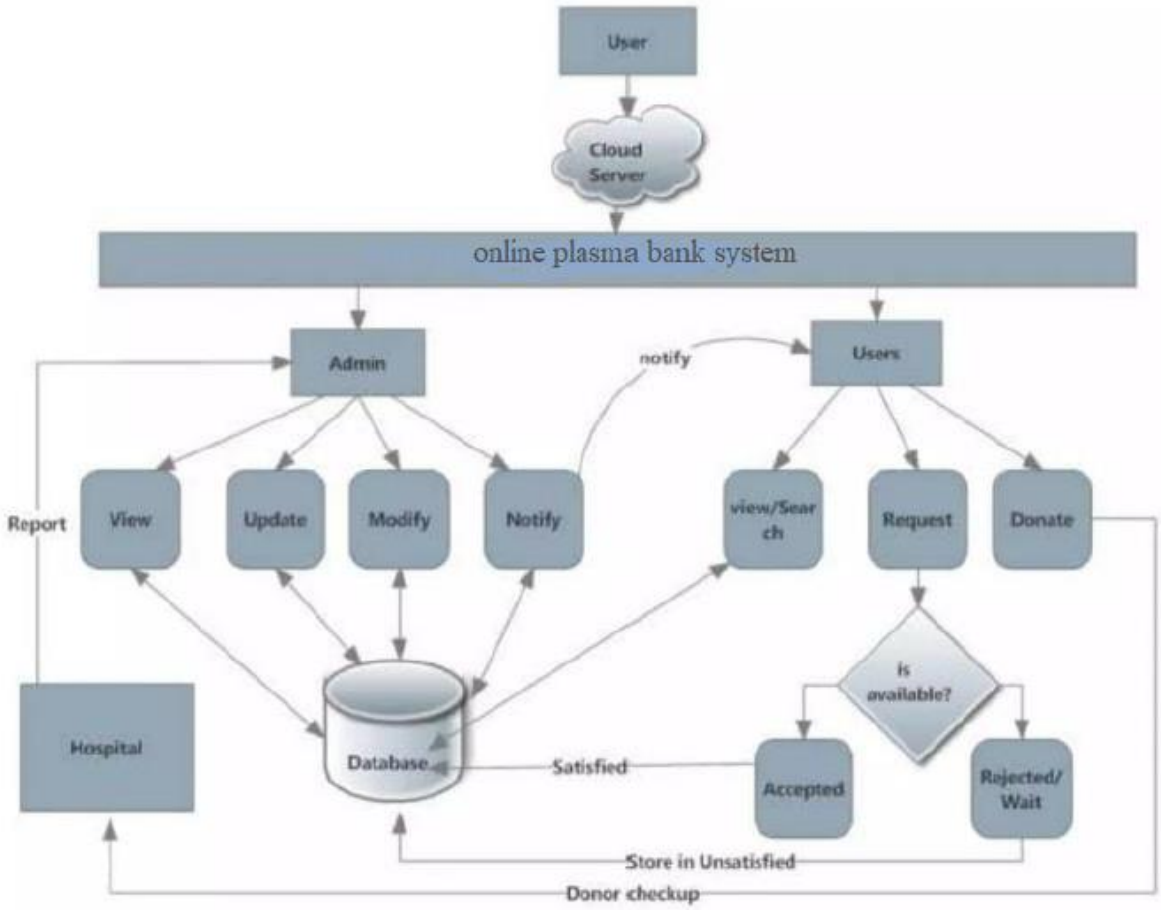
5.2 SOLUTION & TECHNICAL ARCHITECTURE

Solution architecture is the practice of designing, describing and managing solution engineering to match it with specific business problems. Solution architecture is one of the foundational elements of any project or organization. It is the instrumental backbone that holds things together. Solution architecture is aimed at the following overarching goals:

- streamlining of day-to-day activities;
- providing a more efficient production environment;

- lowering costs and gaining cost-effectiveness;
- providing a secure, stable, and supportable environment.

Example - Solution Architecture Diagram:



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
	Confirmation	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
	Access application	USN-3	As a user, I can register for the application through facebook.	I can register & access the dashboard with Facebook login.	Low	Sprint-2
	Registration Gmail	USN-4	As a user, I can register	User can receive successful	Medium	Sprint-1

			for the application through Gmail	registration Gmail		
	Login	USN-5	As a user, I can log into the applicati on by entering email & password	User can enter into application successfully by applying valid email and password	High	Sprint-1
Customer (Web user)	Access Website	USN-6	Capable to access web application through an browser	User can access web application	Medium	Sprint-1
	Search for donor	USN-7	Search result can be viewed in a list	User can view list represents a specific donor with donor detail	High	Sprint-1
Customer Care Executive	Software Operator	USN-8	User should be able to register through application. Donor must provide username, gender, blood group, location,	The user's response surprised us positively	High	Sprint-1

			contact.			
	View request	USN-9	The customer care executive should be able to view received request and then respond to them	User can receive the request response immediately	High	Sprint-1
Maintenance	Maintenance	USN-10	Admin can access, view, modify, update all details of the plasma donor application	Admin is the authorized person of the overall application	High	Sprint-1

6. PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING & ESTIMATION:

Sprint	Functional Requirement	User Story Number	User Story/Task	Story Points	Priority	Team Members
sprint1	Registration	USN-1	As a user, I can register for the application by using necessary details	5	High	A.Niranjana N.pradhiksha
sprint1	Login	USN-2	As a user, I can log into the application by entering user id & password	4	High	K.R.Priyadharshini K.G.Priyadharshini
sprint1	Profile	USN-3	As a user, I am able to update and view my profile page.	4	High	A.Niranjana K.S.Sujitha
Sprint-2	Plasma Request	USN-4	As a user, I can place a plasma request or donate plasma.	4	High	N.Pradhiksha K.R.Priyadharshini
sprint2	Verifying Request	USN-5	As a user, I will wait until my request is verified.	5	HIgh	A.Niranjana K.G.Priyadharshini
Sprint-3	Accept the request	USN-6	AS a user, I will wait until the Request is accepted.	4	Medium	K.S.Sujitha K.G.Priyadharshini

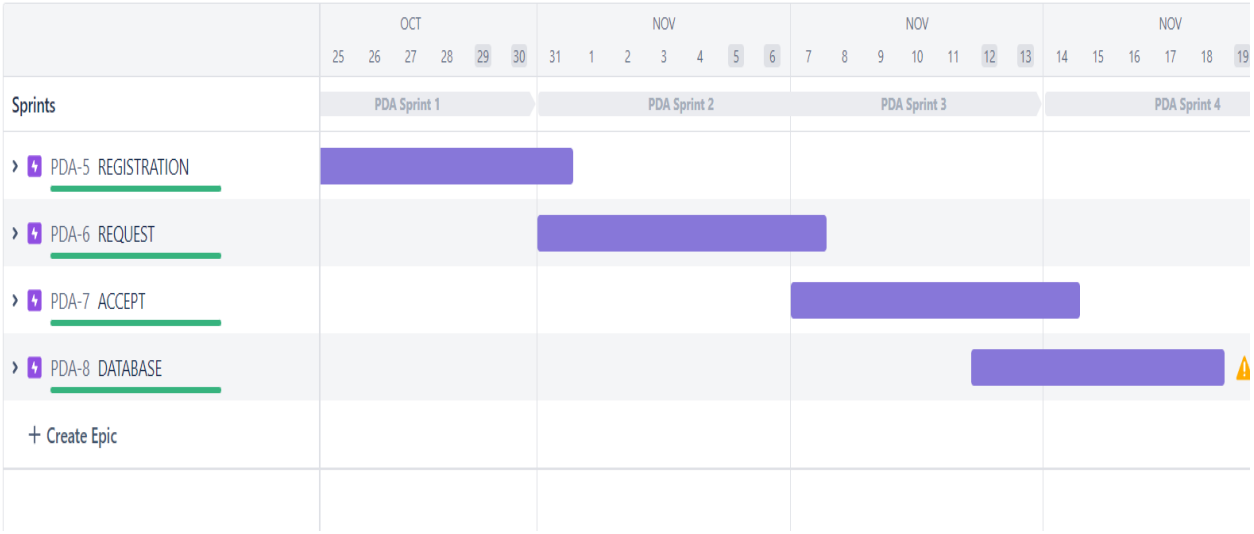
sprint-3	Donor Details	USN-7	The details of the volunteered donor are stored in thedatabase.	4	Medium	N.Pradhiksha A.Niranjana
sprint-4	Support	USN-8	As a user, I can chat with a chatbot regarding my queries and doubts.	3	Medium	K.R.Priyadharshini K.S.Sujitha
sprint-4	Certificate and Rewards	USN-9	As a donor, I will receive an e-certificate after donations.Virtual rewards are also provided to the donor.	3	Low	A.Niranjana N.Pradhiksha
Sprint-4	About	USN-10	As a user, I will find about the importance of plasma donation in this section of the application.	3	Medium	K.G.Priyadharshini K.S.Sujitha
Sprint-4	Administrator		We admins will approve all the plasma transaction in the application after proper verification.	3	High	K.R.Priyadharshini K.G.Priyadharshini
Sprint-4	We admins will update the plasma availability and donor count periodically.			3	Medium	

6.2 SPRINT DELIVERY SCHEDULE:

Project Tracker:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date	Story Points Completed(as on planned End Date)	Sprint Release Date(Autual)
Sprint - 1	13	6 Days	24 Oct 2022	29 Oct 2022	13	29 Oct 2022
Sprint - 2	9	6 Days	31 Oct 2022	05 Nov 2022	9	05 Nov 2022
Sprint - 3	8	6 Days	07 Nov 2022	12 Nov 2022	8	12 Nov 2022
Sprint - 4	18	6 Days	14 Nov 2022	19 Nov 2022	18	19 Nov 2022

6.3 REPORTS FROM JIRA:



7. CODING & SOLUTIONING

7.1 FEATURE 1: Flask

- Flask is a web application framework, it's a Python module that lets you develop web applications easily.
- It does have many cool features like url routing, template engine. It is a WSGI web app framework.
- Flask is often referred to as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself.
- Flask is one of the most popular web frameworks, meaning it's up-to-date and modern. We can easily extend it's functionality. We can scale it up for complex applications.
- It is designed to keep the core of the application simple and scalable. Applications that use the Flask framework include Pinterest and LinkedIn.

7.2 FEATURE 2: Docker

- Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers.
- The software that hosts the containers is called Docker Engine. The service has both free and premium tiers.
- Docker can package an application and its dependencies in a virtual container that can run on any Linux, Windows, or macOS computer. This enables the application to run in a variety of locations, such as on-premises,

in public (see decentralized computing, distributed computing, and cloud computing) or private cloud.

- When running on Linux, Docker uses the resource isolation features of the Linux kernel (such as cgroups and kernel namespaces) and a union-capable file system (such as OverlayFS) to allow containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.
- Docker implements a high-level API to provide lightweight containers that run processes in isolation. Docker containers are standard processes, so it is possible to use kernel features to monitor their execution—including for example the use of tools like strace to observe and intercede with system calls.

7.3 DATABASE SCHEME:

IBM DB2

- Db2 is a family of data management products, including database servers, developed by IBM.
- It initially supported the relational model, but was extended to support object–relational features and non-relational structures like JSON and XML.
- Db2 (Formerly Db2 for LUW) is a relational database that delivers advanced data management and analytics capabilities for transactional workloads. This operational database is designed to deliver high performance, actionable insights, data availability and reliability, and it is supported across Linux, Unix and Windows operating systems.

- The Db2 database software includes advanced features such as in-memory technology (IBM BLU Acceleration), advanced management and development tools, storage optimization, workload management, actionable compression and continuous data availability (IBM pureScale).
- Db2 on Cloud complies with data protection laws and includes at-rest database encryption and SSL connections. The Db2 on Cloud high availability plans offer rolling security updates and all database instances include daily backups. Security patching and maintenance is managed by the database administrator.

KUBERNETES

- Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management.
- Kubernetes automates operational tasks of container management and includes built-in commands for deploying applications, rolling out changes to your applications, scaling your applications up and down to fit changing needs, monitoring your applications, and more—making it easier to manage applications.
- Kubernetes services provide load balancing and simplify container management on multiple hosts.
- It's used for bundling and managing clusters of containerized applications — a process known as 'orchestration' in the computing world.
- The Kubernetes platform is all about optimization — automating many of the DevOps processes that were previously handled manually and simplifying the work of software developers.

8. TESTING

8.1 TEST CASES:

Testing

- It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectation and does not fail in an unacceptable manner.
- There are various types of test. Each test type addresses a specific testing requirement.

Test case ID	Feature Type	Component	Test Scenario	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Commnets	TC for Automation(Y/N)	BUG ID	Executed By
LoginPage_TC_OO1	UI	Admin Login Page	Verify user is able to see the Login/Signup popup when user clicked on My account button	1.Enter URL http://127.0.0.1:8000/ and click go 2.Click on My Account dropdown button 3.Verify login/Singup popup displayed or not	Username: rit password : rit123	Login/Signup popup should display and navigate to Admin dashboard	Working as expected	Pass		Y		Admin
LoginPage_TC_OO2	Functional	Patient Login page	Verify user is able to log into application with InValid credentials	1.Enter URL http://127.0.0.1:8000/ and click go 2.Click on 3.Verify login/Singup popup with below Patient elements: a.username text box b.password text box c.Login button	Username: shriram password : 2019011280	Application should show 'Incorrect Username or password ' validation message.	Working as expected	Fail	Steps are not clear to follow	N	BUG-1234	Patient

LoginPage_TC_OO3	Functional	Donor Login Page	Verify user is able to log into application with Valid credentials	1.Enter URL http://127.0.0.1:8000/and click go 2.Click on 3.Enter Valid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: sathish password: 201901120	User should navigate to user Donor Home Page	Working as expected	Pass		Y		Donor
LoginPage_TC_OO4	Functional	Patient Login page	Verify user is able to log into application with InValid credentials	1.Enter URL http://127.0.0.1:8000/and click go 2.Click on 3.Enter Valid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: shriram password: 201901128	User should navigate to user Donor Home Page	Working as expected	Pass		Y		Patient

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 Plasma Donation Application project at the time of the release to User Acceptance Testing (UAT).

2. Defect Analysis

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

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Sub total
By Design	8	4	2	3	17
Duplicate	1	0	2	1	4
External	2	3	0	1	6
Fixed	10	2	5	18	35
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	3	2	1	6
Totals	21	12	13	25	71

3. Test Case Analysis

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

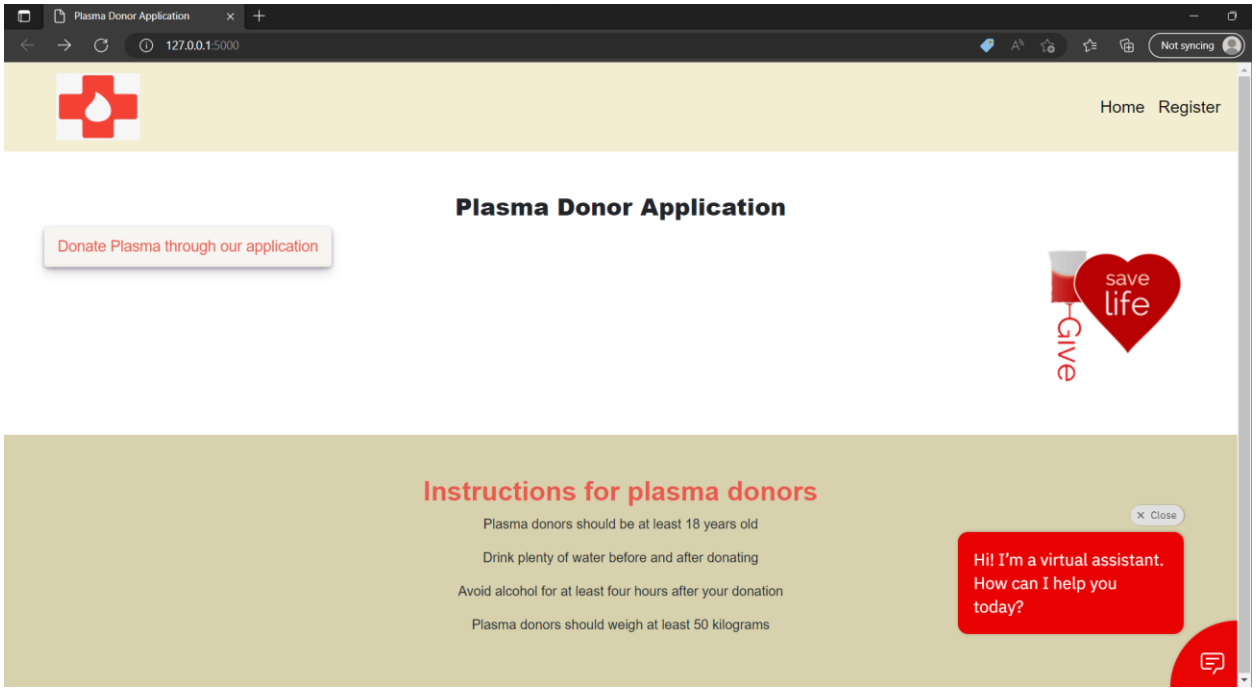
Section	Total Cases	Not Tested	Fail	Pass
Print Engine	8	0	0	8
Client Application	50	0	0	50
Security	2	0	0	2
Outsource Shipping	3	0	0	3
Exception Reporting	10	0	0	10
Final Report Output	6	0	0	6
Version Control	3	0	0	3

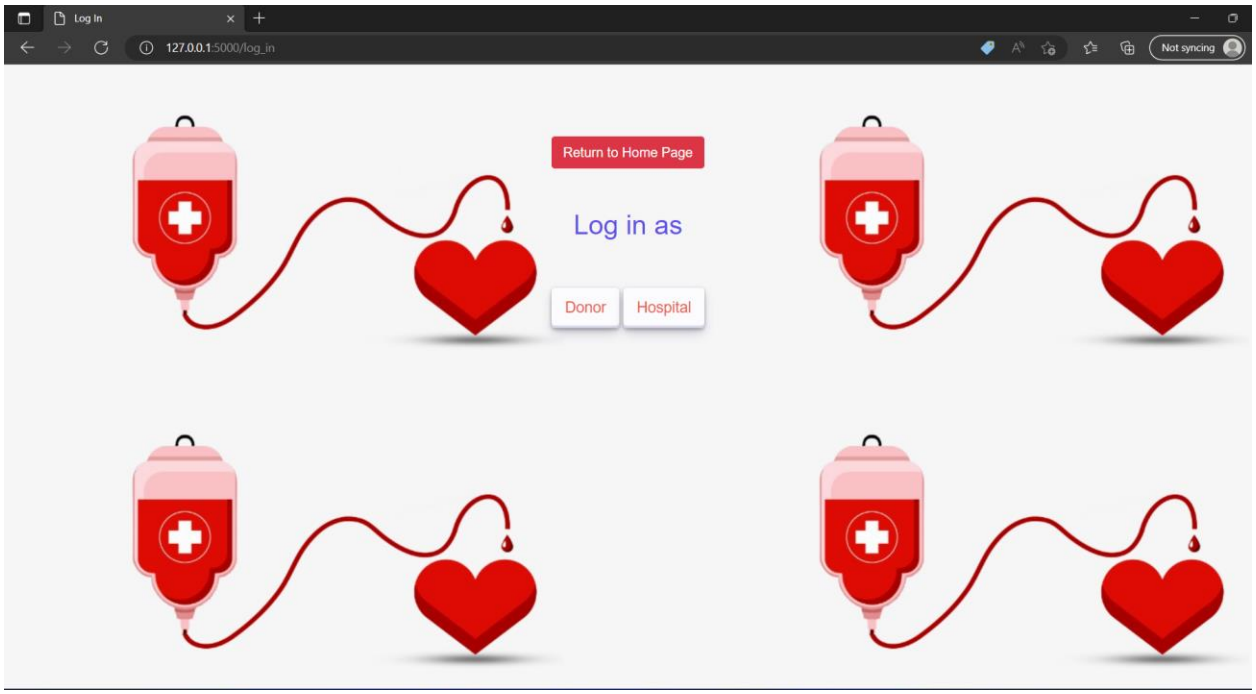
9. RESULTS

9.1 PERFORMANCE METRICS:

- Performance metrics are used to track the progress and performance of a project.
- Monitoring parts of a project like productivity, scheduling, and scope make it easier for team leaders to see what's on track.
- As a project evolves, managers need access to changing deadlines or budgets to meet their client's expectations.
- Performance metrics can give team leaders a 10,000-foot view of any project in their pipeline while it's in progress.

OUTPUT:





Return to Home Page

Log In

Donor

Username

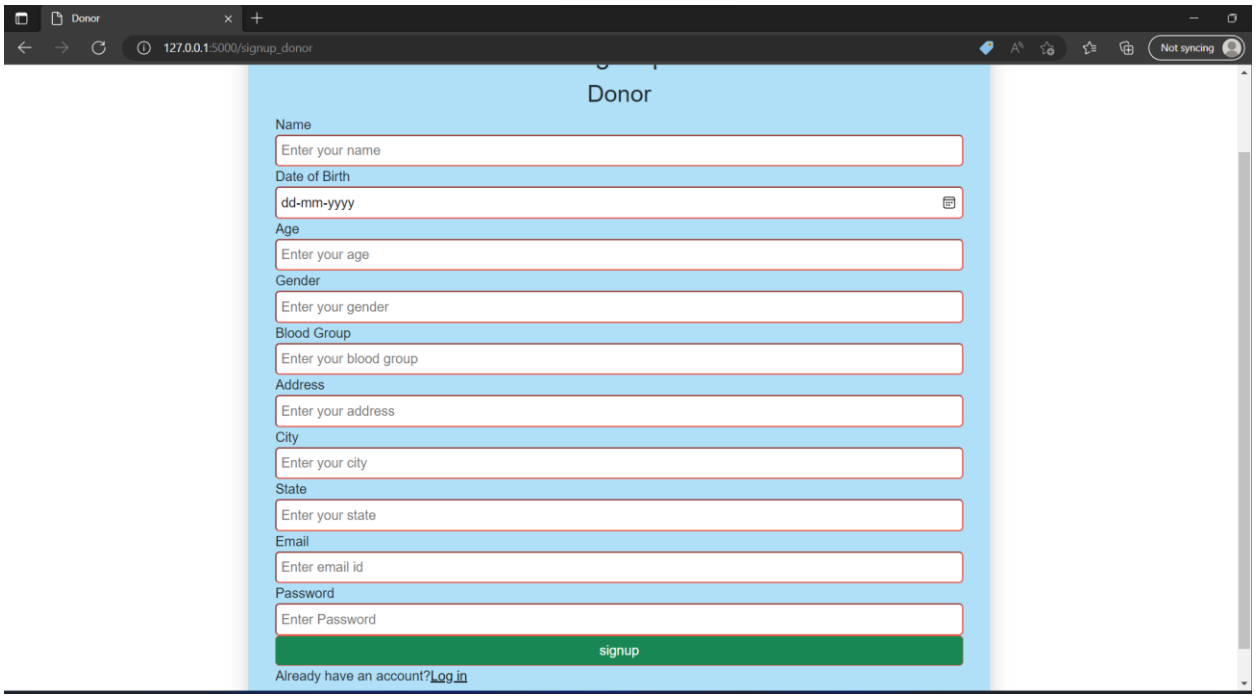
Enter username

Password

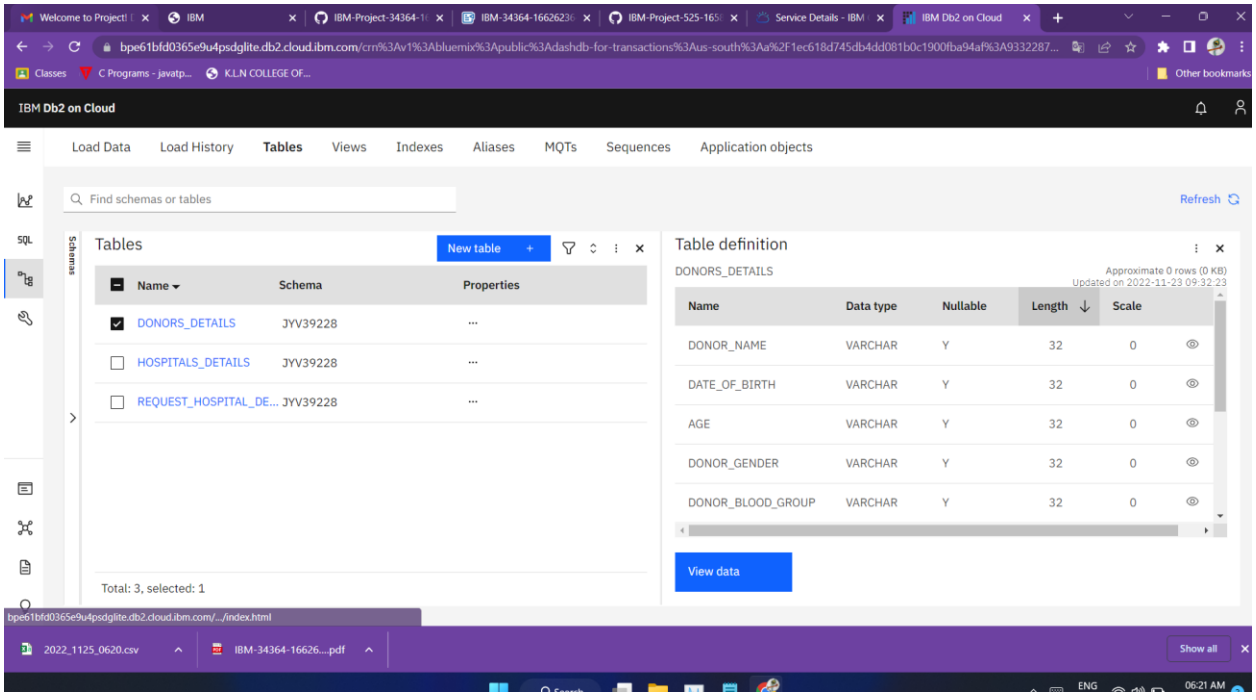
Enter Password

Log in

Don't have an account?Sign up


Donor
127.0.0.1:5000/signup_donor
Not syncing
Donor
Name
Enter your name
Date of Birth
dd-mm-yyyy
Age
Enter your age
Gender
Enter your gender
Blood Group
Enter your blood group
Address
Enter your address
City
Enter your city
State
Enter your state
Email
Enter email id
Password
Enter Password
signup
Already have an account?Log in

IBM DB2:


IBM Db2 on Cloud
Load Data
Load History
Tables
Views
Indexes
Aliases
MQTs
Sequences
Application objects
Find schemas or tables
Refresh
Tables
New table
Name
Schema
Properties
DONORS_DETAILS
JYV39228
HOSPITALS_DETAILS
JYV39228
REQUEST_HOSPITAL_DE...
JYV39228
Total: 3, selected: 1
Table definition
DONORS_DETAILS
Approximate 0 rows (0 KB)
Updated on 2022-11-23 09:32:23
Name
Data type
Nullable
Length
Scale
DONOR_NAME
VARCHAR
Y
32
0
DATE_OF_BIRTH
VARCHAR
Y
32
0
AGE
VARCHAR
Y
32
0
DONOR_GENDER
VARCHAR
Y
32
0
DONOR_BLOOD_GROUP
VARCHAR
Y
32
0
View data
2022_1125_0620.csv
IBM-34364-16626....pdf
Show all

10. ADVANTAGES & DISADVANTAGES

ADVANTAGES:

- The plasma donor website is easy to use and accessible for everyone.
- User can search for donor easily during emergency situation.
- Saves time.
- Stores donors information in database and easy to access it.

DISADVANTAGES:

- This website requires internet connection to access it.
- This website needs maintenance.
- This site cannot automatically verify the genuine users.

11. CONCLUSION

- The efficient way of finding plasma door for the infected people is implemented using the plasma donor website that is hosted on IBM Cloud platform.
- To ensure the smooth functioning of the web site operation. I have hosted the website in IBM Db2 & Kubernetes Cluster to make sure the operations are running successfully Cloud lambda function is used and to deploy the application IBM Db2 service is used.

12. FUTURE SCOPE

- Upgrading the UI that is more user-friendly which will help many users to access the website and also ensures that many plasma donors can be added into the community.
- Using elastic load balancer, it helps to handle multiple requests at the same time which will maintain the uptime of the website with negligible downtime.

13. APPENDIX

13.1 SOURCE CODE:

App.py

```
import os
os.add_dll_directory("C:\\Program Files\\IBM\\IBM DATA SERVER
DRIVER\\bin")
from flask import Flask,render_template,url_for,session,redirect,request
import re,sendgrid
from sendgrid.helpers.mail import *
import ibm_db
app = Flask(__name__)
app.secret_key="1123"
try:
    conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b1bc1829-6f45-
4cd4-bef4-
10cf081900bf.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32304;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=j
yv39228;PWD=yCEXohXeyfdjTyug",",")
   print("connected")
except:
    print("not connected")
# index page starts----------------------------
@app.route("/")
def hello_world():
    return render_template("index.html")
```

```
@app.route("/log_in")
def login():
    return render_template("log_in.html")
#index page ends--------------------------------
#donor login starts-----------------------------------
@app.route("/login_donor",methods=['GET','POST'])
def login_donor():
    global userid
    msg = ''
    if request.method == 'POST' :
        donor_name = request.form['donor_name']
        donor_password = request.form['donor_password']
        sql = "SELECT * FROM donors_details WHERE donor_name =? AND
donor_password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,donor_name)
        ibm_db.bind_param(stmt,2,donor_password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['loggedin'] = True
            session['id'] = account['DONOR_NAME']
            userid=  account['DONOR_NAME']
            session['donor_name'] = account['DONOR_NAME']
            msg = 'Logged in successfully as donor!'
           return render_template('homepage_donor.html', msg = msg)
```

```
        else:
            msg = 'Incorrect email / password !'
    return render_template("login_donor.html")
#donor login ends----------------------------------------
#sign up donor starts----------------------------------@app.route("/signup_donor")
def signup_donor():
    return render_template("signup_donor.html")
@app.route("/donor_form", methods=['GET','POST'])
def donor_form():
    msg = ''
    if request.method == 'POST' :
        donor_name = request.form['donor_name']
        date_of_birth = request.form['date_of_birth']
        age = request.form['age']
        donor_gender = request.form['donor_gender']
        donor_blood_group = request.form['donor_blood_group']
        donor_address = request.form['donor_address']
        donor_city = request.form['donor_city']
        donor_state = request.form['donor_state']
        donor_email = request.form['donor_email']
        donor_password = request.form['donor_password']
        sql = "SELECT * FROM donors_details WHERE donor_name =?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,donor_name)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
```

```
        if account:
            msg = 'Account already exists !'
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', donor_email):
            msg = 'Invalid email address !'
        elif not re.match(r'[A-Za-z0-9]+', donor_name):
            msg = 'name must contain only characters and numbers !'
        else:
            mailtest_registration(donor_email)
            insert_sql = "INSERT INTO donors_details VALUES (?, ?, ?,
?,?,?,?,?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, donor_name)
            ibm_db.bind_param(prep_stmt, 2, date_of_birth)
            ibm_db.bind_param(prep_stmt, 3, age)
            ibm_db.bind_param(prep_stmt, 4, donor_gender)
            ibm_db.bind_param(prep_stmt, 5, donor_blood_group)
            ibm_db.bind_param(prep_stmt, 6, donor_address)
            ibm_db.bind_param(prep_stmt, 7, donor_city)
            ibm_db.bind_param(prep_stmt, 8, donor_state)
            ibm_db.bind_param(prep_stmt, 9, donor_email)
            ibm_db.bind_param(prep_stmt, 10, donor_password)
            ibm_db.execute(prep_stmt)
            msg = 'You have successfully registered as donor!'
           return render_template('login_donor.html', msg = msg)
        #sign up donor ends----------------------------------------


# login hospital----------------------------------------
```

```
@app.route("/login_hospital", methods=['GET','POST'] )
def login_hospital():
    msg = ''
    if request.method == 'POST' :
        hospital_name = request.form['hospital_name']
        hospital_password = request.form['hospital_password']
        sql = "SELECT * FROM hospitals_details WHERE hospital_name =? AND
hospital_password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,hospital_name)
        ibm_db.bind_param(stmt,2,hospital_password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['loggedin'] = True
            session['id'] = account['HOSPITAL_NAME']
            userid=  account['HOSPITAL_NAME']
            session['hospital_name'] = account['HOSPITAL_NAME']
            msg = 'Logged in successfully as recipient!'
            return render_template('donors.html', msg = msg)
        else:
            msg = 'Incorrect email / password !'
    return render_template('login_hospital.html', msg = msg)
#sign up hospital starts----------------------------------------------
        @app.route("/signup_hospital",methods=['GET','POST'])
def signup_hospital():
```

```
msg = ''
if request.method == 'POST' :
    hospital_name = request.form['hospital_name']
    hospital_date = request.form['hospital_date']
    hospital_contact_number = request.form['hospital_contact_number']
    hospital_address = request.form['hospital_address']
    hospital_city = request.form['hospital_city']
    hospital_state = request.form['hospital_state']
    hospital_email = request.form['hospital_email']
    hospital_password = request.form['hospital_password']
    sql = "SELECT * FROM hospitals_details WHERE hospital_name =?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,hospital_name)
    ibm_db.execute(stmt)
    account = ibm_db.fetch_assoc(stmt)
    print(account)
    if account:
        msg = 'Account already exists !'
    elif not re.match(r'[^@]+@[^@]+\.[^@]+', hospital_email):
        msg = 'Invalid email address !'
    elif not re.match(r'[A-Za-z0-9]+', hospital_name):
        msg = 'name must contain only characters and numbers !'
    else:
        mailtest_hosp_registration(hospital_email)
        insert_sql = "INSERT INTO hospitals_details VALUES (?, ?, ?, ?,?,?,?,?)"
        prep_stmt = ibm_db.prepare(conn, insert_sql)
        ibm_db.bind_param(prep_stmt, 1, hospital_name)
```

```
            ibm_db.bind_param(prep_stmt, 2, hospital_date)
            ibm_db.bind_param(prep_stmt, 3, hospital_contact_number)
            ibm_db.bind_param(prep_stmt, 4, hospital_address)
            ibm_db.bind_param(prep_stmt, 5, hospital_city)
            ibm_db.bind_param(prep_stmt, 6, hospital_state)
            ibm_db.bind_param(prep_stmt, 7, hospital_email)
            ibm_db.bind_param(prep_stmt, 8, hospital_password)
           ibm_db.execute(prep_stmt)
            msg = 'You have successfully registered as recipient!'
            # mailtest(usermail)
        return render_template('login_hospital.html', msg = msg)
    elif request.method == 'POST':
        msg = 'Please fill out the form !'
    return render_template("signup_hospital.html")
#sign up hospital ends----------------------------------------------------
#donor list table starts--------------------------------------------------------
@app.route('/donorlist')
def donorlist():
    donors_details = []
    sql = "SELECT * FROM DONORS_DETAILS"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_both(stmt)
    while dictionary != False:
        donors_details.append(dictionary)
        dictionary = ibm_db.fetch_both(stmt)
    if donors_details:
```

```
        return render_template("homepage_hospital.html", donors_details =
donors_details)
#donor list table endss----------------------------------------------------------
#request plasma-----------------------------------------------------------------
@app.route("/request_plasma",methods=['GET','POST'])
def request_plasma():
    msg = ''
    if request.method == 'POST' :
        hospital_name = request.form['hospital_name']
        hospital_contact_number = request.form['hospital_contact_number']
        patient_name = request.form['patient_name']
        patient_age = request.form['patient_age']
        patient_blood_group = request.form['patient_blood_group']
        cause_of_plasma = request.form['cause_of_plasma']
        hospital_address = request.form['hospital_address']
        hospital_city = request.form['hospital_city']
        hospital_state = request.form['hospital_state']
        hospital_email = request.form['hospital_email']
        sql = "SELECT * FROM request_hospital_details WHERE hospital_name
=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,hospital_name)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = 'Account already exists !'
```

```
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', hospital_email):
            msg = 'Invalid email address !'
        elif not re.match(r'[A-Za-z0-9]+', hospital_name):
            msg = 'name must contain only characters and numbers !'
        else:
            mailtest_request_plasma(hospital_email)
            insert_sql = "INSERT INTO request_hospital_details VALUES (?, ?, ?,
?,?,?,?,?,?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, hospital_name)
            ibm_db.bind_param(prep_stmt, 2, hospital_contact_number)
            ibm_db.bind_param(prep_stmt, 3, patient_name)
            ibm_db.bind_param(prep_stmt, 4, patient_age)
            ibm_db.bind_param(prep_stmt, 5, patient_blood_group)
            ibm_db.bind_param(prep_stmt, 6, cause_of_plasma)
            ibm_db.bind_param(prep_stmt, 7, hospital_address)
            ibm_db.bind_param(prep_stmt, 8, hospital_city)
            ibm_db.bind_param(prep_stmt, 9, hospital_state)
            ibm_db.bind_param(prep_stmt, 10, hospital_email)
            ibm_db.execute(prep_stmt)
            msg = 'You have successfully registered to request plasma. our admin will
connect you shortly with donor details!'
            # mailtest(usermail)
            return render_template('success.html', msg = msg)
    elif request.method == 'POST':
        msg = 'Please fill out the form !'
    return render_template("request_plasma.html")
```

```
# sendgrid integration
#donor registration starts----------------------------------------
def mailtest_registration(to_email):
    sg = sendgrid.SendGridAPIClient(api_key='')
    from_email = Email("niranjanaasokan25@gmail.com")
    subject = "Registered Successfull as a DONOR!"
    content = Content("text/plain", "You have successfully registered as donor.
Please Login using your Username and Password to donate Plasma.")
    mail = Mail(from_email, to_email, subject, content)
    response = sg.client.mail.send.post(request_body=mail.get())
    print(response.status_code)
    print(response.body)
    print(response.headers)
#donor registration ends----------------------------------------
#hospital registration starts----------------------------------------
def mailtest_hosp_registration(to_email):
    sg = sendgrid.SendGridAPIClient(api_key= ' )
    from_email = Email("niranjanaasokan25@gmail.com")
    subject = "Registered Successfull as a Hospital!"
    content = Content("text/plain", "You have successfully registered as recipient.
Please Login using your Username and Password to request Plasma.")
    mail = Mail(from_email, to_email, subject, content)
    response = sg.client.mail.send.post(request_body=mail.get())
    print(response.status_code)
    print(response.body)
    print(response.headers)
#hospital registration ends----------------------------------------
```

```
#request plasma confirmation mail starts----------------------------------
def mailtest_request_plasma(to_email):
    sg = sendgrid.SendGridAPIClient(api_key= " )
    from_email = Email("niranjanaasokan25@gmail.com")
    subject = "Successfully registered to request plasma!"
    content = Content("text/plain", "You have successfully registered to request
plasma. our admin will connect you shortly with donor details")
    mail = Mail(from_email, to_email, subject, content)
    response = sg.client.mail.send.post(request_body=mail.get())
    print(response.status_code)
    print(response.body)
    print(response.headers)
#request plasma confirmation mail ends----------------------------------
@app.route('/index')
def index():
    session.clear()
    return render_template("index.html")
if __name__ == '__main__':
   app.run(host='0.0.0.0', port=5000, debug=True)
```

donors.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <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>Logged In Successfully</title>
     <!--bootstrap files starts-->
     <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet">
     <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js">
</script>
     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css">
    <!--bootstrap code ends-->
<style>
    .button{
       display: grid;
       grid-template-columns: 1fr;
       align-items: center;
       justify-items: center;
    }
</style>
</head>
<body>
    <h2 style="text-align: center;">Plasma Donor Application</h2>
<h4 style="text-align: center;">{{ msg }}</h4>
<h3 class="text-danger" style="margin-top: 50px; text-align: center;">Welcome
{{session["hospital_name"]}}!!</h3>
<h5 style="margin-top: 50px; text-align: center;">Please, Click on this below
button to navigate to your dashborad to request plasma! </h5>
```

```
<div class="button">
<button class="btn btn-danger"><a href="{{url_for('donorlist')}}" style="text-
decoration: none; color: white;"> Request</a></button>
</div>
</body>
</html>
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <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>Plasma Donor Application</title>
    <link rel="stylesheet" href="../static/style.css">
    <!--bootstrap files starts-->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js">
</script>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css">
   <!--bootstrap code ends-->
   <style>
```

```
  .custom-toggler.navbar-toggler {
    border:transparent;
       }
     .custom-toggler .navbar-toggler-icon {
           background-image: ("img.png");
       }
        .navbar-nav {
           margin-left: auto;
        }
        .nav-item{
          text-decoration: none;
          color:#070909;
        }
        .nav-item:hover{
          text-decoration: none;
         color:black;
        }
        .nav-item:visited{
          text-decoration: none;
        }
  </style>
<script>
  window.watsonAssistantChatOptions = {
    integrationID: "b5e67d39-9862-4f40-9dc5-020482814f3a", // The ID of this
integration.
    region: "au-syd", // The region your integration is hosted in.
```

```
    serviceInstanceID: "c084d960-061d-4eb3-bad3-3745a05f6c2a", // The ID of
your service instance.
    onLoad: function(instance) { instance.render(); }
  };
  setTimeout(function(){
    const t=document.createElement('script');
    t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +
(window.watsonAssistantChatOptions.clientVersion || 'latest') +
"/WatsonAssistantChatEntry.js";
    document.head.appendChild(t);
  });
 </script>
</head>
<body>
<!--navbar section starts-->
<div class="m-0" style=" background-color: rgba(226, 211, 139, 0.4);">
    <nav class="navbar navbar-expand-lg">
        <div class="container-fluid">
            <a href="#" class="navbar-brand" style="margin-left:50px;">
                <img src={{url_for('static',filename="/images/logo.png")}} width="100"
height="80" alt="plasma">
     </a>
            <button type="button" class="navbar-toggler custom-toggler" data-bs-
toggle="collapse" data-bs-target="#navbarCollapse"></button>
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarCollapse">
```

```
            <div class="navbar-nav" style="font-weight: 400;">
                <a href="{{url_for('hello_world')}}" class="nav-item nav-link active" style="font-size: 20px;">Home</a>
                <a href="{{url_for('login')}}" class="nav-item nav-link" style="font-size: 20px;">Register</a>
            </div>
        </div>
    </div>
  </nav>
</div>
<!--navbar section ends-->
<!--What is plasma donation section starts-->
<div class="what_is_plasma_donation_section" >
  <div class="what_is_plasma_donation">
    <h3 style="text-align: center; font-weight: 800;">Plasma Donor Application</h3>
    <button class="button-30"><a href="{{url_for('login')}}">Donate Plasma through our application</a></button>
    <img align ="right" src={{url_for('static',filename="/images/pic.png")}} alt="picture">
  </div>
</div>
<!--What is plasma donation section ends-->
<!--why there is a need of plasma section starts-->
<div class="what_is_plasma_donation_section" style=" background-color: rgba(156, 139, 53, 0.4);">
  <div class="what_is_plasma_donation">
```

```
        <h2 style="text-align: center;font-weight: 600;">Instructions for plasma donors</h2>
        <p style="text-align: center; font-weight:500">Plasma donors should be at least 18 years old</p>
        <p style="text-align: center; font-weight:500">Drink plenty of water before and after donating</p>
        <p style="text-align: center; font-weight:500">Avoid alcohol for at least four hours after your donation</p>
        <p style="text-align: center; font-weight:500"> Plasma donors should weigh at least 50 kilograms</p>
    </div>
</div>
<!--why there is a need of plasma section ends-->
</body>
</html>
```

Log_in.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <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>Log In</title>
    <link rel="stylesheet" href="../static/log_in.css">
     <!--bootstrap files starts-->
```

```
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js">
</script>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css">
   <!--bootstrap code ends--> </head>
<body>
   <!--log in section starts-->
<div class="log_in_section">
   <div class="log_in">
      <button class="btn btn-danger text-center" style="text-align: center;"><a
href="{{url_for('hello_world')}}" style="text-decoration: none;
color:white">Return to Home Page</a></button>
      <h2>Log in as</h2>
      <div class="log_in_buttons">
        <button class="button-30"><a
href="{{url_for('login_donor')}}">Donor</a></button>
          <button class="button-30"><a
href="{{url_for('login_hospital')}}">Hospital</a></button>
      </div>
   </div>
</div>
<!--log in section ends-->
</body>  </html>
```

Request_plasma.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <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>Hospital</title>
    <link rel="stylesheet" href="../static/login.css">
</head>  <body>
    <div class="log_in_section">
        <div class="log_in_container">
            <h2>Donor Request form</h2>
            <h3>(After filling the request form, we will connect you with the right
donor)</h3>
            <div class="log_in_form">
                <form action="/request_plasma" method="POST">
                    <label>Hospital Name</label>
                    <input type="text" name="hospital_name" placeholder="Enter your
Hospital name">
 <label>Hospital Phone Number</label>
                    <input type="text" name="hospital_contact_number"
placeholder="Enter Hospital Phone Number">
 <label>Patient Name</label>
                    <input type="text" name="patient_name" placeholder="Enter the
patient Name">
     <label>Patient Age</label>
```

```
            <input type="text" name="patient_age" placeholder="Enter the patient
Age">
<label>Patient Blood Group</label>
            <input type="text" name="patient_blood_group" placeholder="Enter
the patient Blood Group">
 <label>Cause of requesting plasma</label>
            <input type="text" name="cause_of_plasma" placeholder="Cause of
requesting plasma">
<label>Hospital Address</label>
            <input type="text" name="hospital_address" placeholder="Enter your
Hospital address">
 <label>Hospital City</label>
            <input type="text" name="hospital_city" placeholder="Enter your
Hospital city">
<label> Hospital State</label>
            <input type="text" name="hospital_state" placeholder="Enter your
Hospital state">
<label>Hospital Email</label>
            <input type="email" name="hospital_email" placeholder="Enter
Hospital email id">
<input type="submit" value="Request" class="btn btn-success">
 </form>
</div>
      </div>
   </div>
</body>
</html>
```

Signup_donor.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <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>Donor</title>
    <link rel="stylesheet" href="../static/login.css">
     <!--bootstrap files starts-->
     <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet">
     <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js">
</script>
     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css">
    <!--bootstrap code ends-->
</head>
<body>
    <div class="log_in_section">
        <button class="btn btn-danger text-center" style="text-align: center;margin-
bottom:10px;"><a href="{{url_for('hello_world')}}" style="text-decoration: none;
color:white">Return to Home Page</a></button>
        <div class="log_in_container">
            <h2>Sign Up</h2>
```

```
        <h3>Donor</h3>
        <div class="log_in_form">
           <form action="/donor_form" method="POST">
              <label>Name</label>
              <input type="text" name="donor_name" placeholder="Enter your
name">
  <label>Date of Birth</label>
              <input type="date" name="date_of_birth" placeholder="Enter your
date of birth">
<label>Age</label>
              <input type="number" name="age" placeholder="Enter your age">
<label>Gender</label>
              <input list="Gender" name="donor_gender" placeholder="Enter your
gender">
              <datalist id="Gender">
                 <option value="Male">
                 <option value="Female">
              </datalist>
<label>Blood Group</label>
              <input type="text" name="donor_blood_group" placeholder="Enter
your blood group">
<label>Address</label>
              <input type="text" name="donor_address" placeholder="Enter your
address">
<label>City</label>
       <input type="text" name="donor_city" placeholder="Enter your city">
<label>State</label>
```

```
                <input type="text" name="donor_state" placeholder="Enter your
state">
 <label>Email</label>
                <input type="email" name="donor_email" placeholder="Enter email
id">
<label>Password</label>
                <input type="password" name="donor_password" placeholder="Enter
Password">
<input type="submit" value="signup" class="btn btn-success">
     <p>Already have an account?<a href="{{url_for('login_donor')}}">Log
in</a></p>
              </form>
            </div>
        </div>
    </div>
</body>
</html>
```

Signup_hospital.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <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>Hospital</title>
    <link rel="stylesheet" href="../static/login.css">
```

```
    <!--bootstrap files starts-->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js">
</script>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css">
   <!--bootstrap code ends-->
</head>
<body>
    <div class="log_in_section">
        <button class="btn btn-danger text-center" style="text-align: center;margin-
bottom:10px;"><a href="{{url_for('hello_world')}}" style="text-decoration: none;
color:white">Return to Home Page</a></button>
        <div class="log_in_container">
            <h2>Sign Up</h2>
            <h3>Hospital</h3>
            <div class="log_in_form">
                <form action="/signup_hospital" method="POST">
                    <label>Hospital Name</label>
                    <input type="text" name="hospital_name" placeholder="Enter your
Hospital name">
<label>Hospital Established Date</label>
                    <input type="date" name="hospital_date" placeholder="Enter your
Hospital Established Date">
```

```
                <label>Hospital Phone Number</label>
                <input type="text" name="hospital_contact_number"
placeholder="Enter Hospital Phone Number">
 <label>Hospital Address</label>
                <input type="text" name="hospital_address" placeholder="Enter your
Hospital address">
 <label>Hospital City</label>
                <input type="text" name="hospital_city" placeholder="Enter your
Hospital city">
<label> Hospital State</label>
                <input type="text" name="hospital_state" placeholder="Enter your
Hospital state">
<label>Hospital Email</label>
                <input type="email" name="hospital_email" placeholder="Enter
Hospital email id">
<label>Password</label>
                <input type="password" name="hospital_password"
placeholder="Enter Password">
<input type="submit" value="signup" class="btn btn-success">
        <p>Already have an account?<a href="{{url_for('login_hospital')}}">Log
in</a></p>
              </form> </div>
       </div>
    </div>
</body>
</html>
```

Success.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <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>Successfully requested plasma</title>
     <!--bootstrap files starts-->
     <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet">
     <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js">
</script>
     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css">
    <!--bootstrap code ends-->
    <style>
        .button{
            display: grid;
            grid-template-columns: 1fr;
            align-items: center;
            justify-items: center;
        }
    </style>
```

```
</head>
<body>
    <h2 style="text-align: center;">Plasma Donor Application</h2>
    <h4 style="text-align: center; color:#5783b6;">{{ msg }}</h4>
    <h5 style="margin-top: 50px; text-align: center;">Please, Click on this below button to navigate to your dashboard! </h3>
    <div class="button">
    <button class="btn btn-danger"><a href="{{url_for('donorlist')}}" style="text-decoration: none; color: white;"> Return to dashboard</a></button>
    </div>
</body>
</html>
```

13.2 GITHUB & PROJECT DEMO LINK:

Github link:

https://github.com/IBM-EPBL/IBM-Project-34364-1660234586

Project demo link:

https://drive.google.com/file/d/1A0T8oO1Wtry_D3TgXcpGgk-tUgx3oElS/view?usp=sharing