Project Report Format

Team Id	PNT2022TMID06247
Project Name	Plasma Donor Application
Team Member	Aadityaa B S Raghul P Ramesh K Vijayakumar S

1) INTRODUCTION:

1.1 Project Overview:

Plasma is a critical part of the treatment for many serious health problems. Therefore, there are blood drives asking people to donate blood plasma. The main goal of our project is to make it easier for the COVID-19 patients to get a plasma donor easily as well as donate plasma if they have recovered. The system targets two types of users: the people who want to donate plasma and the people who need plasma. The user can also view the total active cases, nearby vaccine centres, hospitals address.

The main objective of developing the website is to make it easier for the COVID-19 patients to get a plasma donor easily and as soon as possible. Yet, the need for plasma-derived products has been strongly increasing for some years, and blood collection agencies have to adapt if they want to meet this demand. This article aims to review the main motivations and deterrents to whole blood donation, and to compare them with those that we already know concerning plasma donation. Current evidence shows similarities between both

behaviours, but also differences that indicate a need for further research regarding plasma donation.

1.2 Purpose :

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. Regarding the problem faced, an application is to be built which would take the donor details, store them and inform them upon a request.

2) LITERATURE SURVEY:

2.1 Existing problem:

2.2 Reference:

1]Javed Akhtar Khan and M. R. Alony, "A New Concept of Blood Bank Management System using Cloud Computing for Rural Area", International Journal of Electrical Electronics, vol. 4, no. 1, pp. 20-26, 2015 **.**

2]T. Hilda Jenipha and R. Backiyalakshmi, "Android Blood Donor Life Saving Application in Cloud Computing" in American Journal of Engineering Research (AJER), 2014.

3]Sagar Shrinivas, Vasaikar Vijay and Suresh Yennam, "Online Blood Bank Using Cloud Computing", International Journal of Advanced Research Ideas and Innovation In Technology, vol. 3, no. 1.

4]P. Priya, V. Saranya, S. Shabana and Kavitha Subramani, "The Optimization of Blood Donor Information and Management System by Technopedia", International Journal of Innovative Research in Science Engineering and Technology An ISO 3297: 2007 Certified Organization, vol. 3, no. 1, 2014.

5]Siva Shanmuga and N. Ch. S. N. Iyengar, "A Smart Application on Cloud-Based Blood Bank", Journal of Computer and Mathematical Sciences, vol. 7, no. 11, pp. 576-583, November 2016.

6]Almetwally M. Mostafa and Ahmed E. Youssef, . A Framework for a Smart Social Blood Donation System based on Mobile Cloud Computing.

7]Deepak Pandey, Achal Umare and R. S. Mangrulkar, "Requirement Based Blood Storage and Distribution System", International Journal of Research In Science & Engineering, vol. 3, no. 2, March-April 2017.

2.3 Problem Statement Definition:

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.

Plasma is the clear, straw-colored liquid portion of blood that remains after red blood cells, white blood cells, platelets and other cellular components are removed.

At least 18 years old. At least 110 pounds or 50 kilograms. Plasma is rich in nutrients and salts. This can result in dizziness, fainting, and ligtheadedness.

If the person Injected drugs or steroids not prescribed by a doctor within the last three months. Tested positive for HIV.

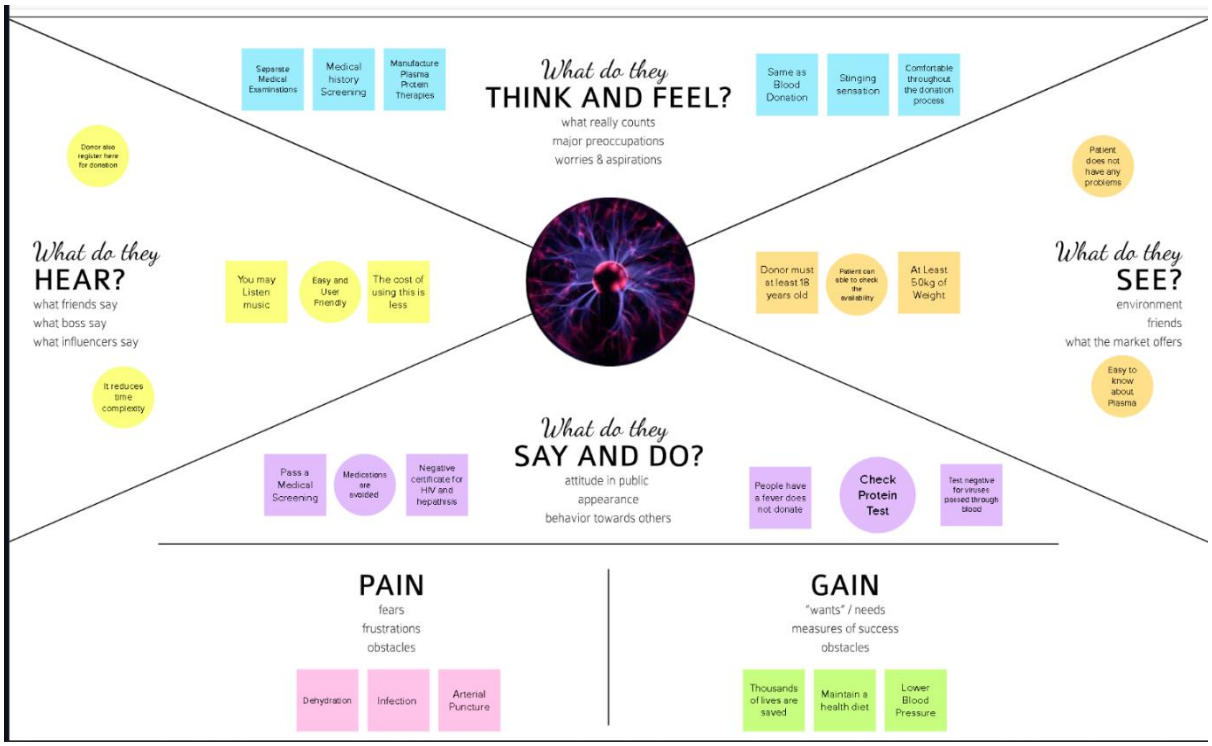
The importance of the problem is that the patient wants to know about the plasma availability and donor also wants to know about what are the requirements need to donate the plasma.

To solve this problem we are going to develop an applications to know the details about the plasma availability and to know the persons eligibility to donate the plasma.

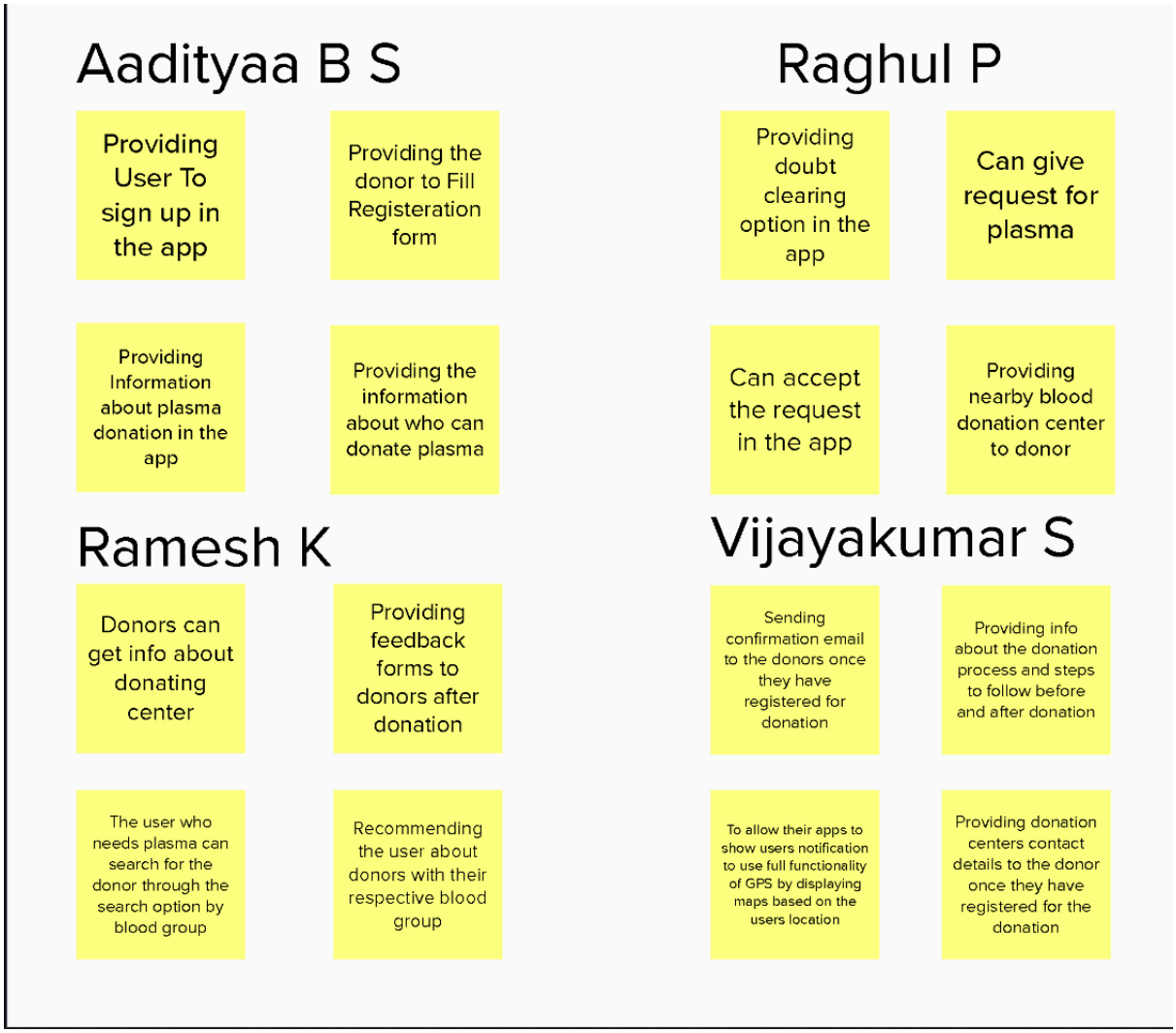
The methodology used to solve this problem is that create an application in the cloud and the user can use the application and can able see the details about the plasma donation.

3) IDEATION & PROPOSED SOLUTION:

3.1 Empathy Map Canvas:



3.2 Ideation and Brainstroming:



3.3 Proposed Solution:

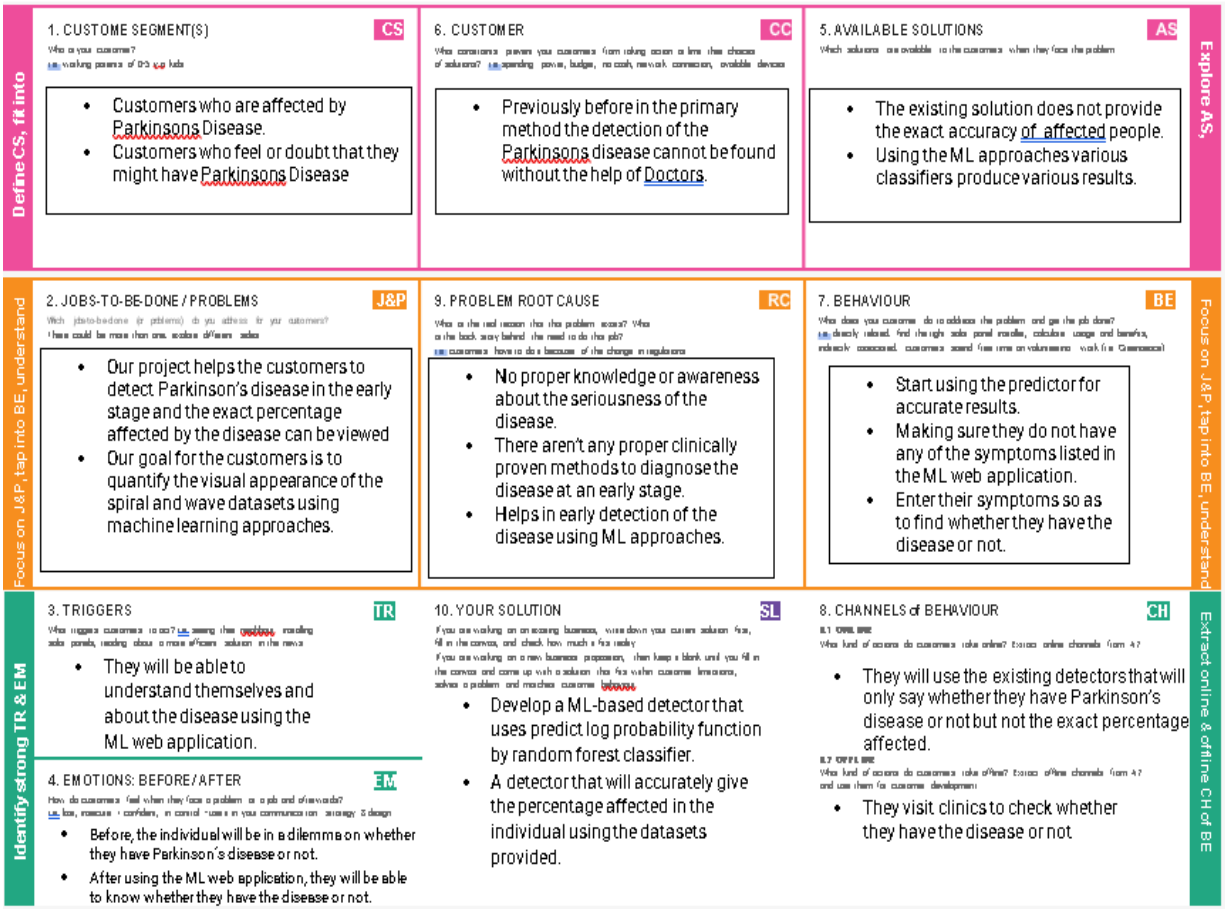
Plasma is the clear, straw-coloured liquid portion of blood that remains after red blood cells, white blood cells, platelets and other cellular components are removed. It is the single largest component of human blood. It helps to maintain blood pressure and volume. Plasma carries electrolytes such as sodium and potassium to our muscles. Plasma helps to maintain a proper pH balance in the body, which supports cell function. The plasma is separated from the

blood through the process of centrifugation and the rest of the blood is transferred back into your body thus avoiding any blood loss. This is a very significant way for you to help during Covid-19. Blood donated by people who have recovered from Covid-19 has antibodies to the virus that causes it. The donated blood is processed to remove blood cells, leaving behind liquid and antibodies. These can be given to people with Covid-19 to boost their ability to fight the virus. This is the brief introduction about the plasma. If the patient don't know about the plasma availability then it will leads to death. To avoid this situation the plasma donor application is here to helps the donor as well as the patient to check about the plasma availability. Saving the contributor data and telling about the ongoing givers would be some assistance as it can save time and assist the clients with finding the vital data about the contributor.

The proposed system gets a connection between the donor and the patient through an online application. The application helps the patient to raise an request for the plasma donation or other requirements.

A User interface is simple for all the users to understand. The user can use this application anytime and anywhere. If the user needs the plasma immediately then the user can raise an request via this application. The user can directly contact with the donor and ask him to donate the plasma. Hospitals can also raise the request against the donor to donate the plasma. This application is user friendly. This application will helps to save lives of many peoples. In now-adays all peoples have a mobile phones in their hand. So they can easily download and use the application.

3.3 Proposed 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 Registration through Gmail
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Location access	Share location access through link
FR-4	Share location access through link	Inform through notification Inform through mail
FR-5	Inform through notification Inform through mail	Accept through pop up Accept through mail
FR-6	End result	Define product features

4.2 NON-Functional requirement:

. FR No	Non-Functional Requirement	Description
NFR-1	Usability	The quality of experience when interacting with the application will be attained.

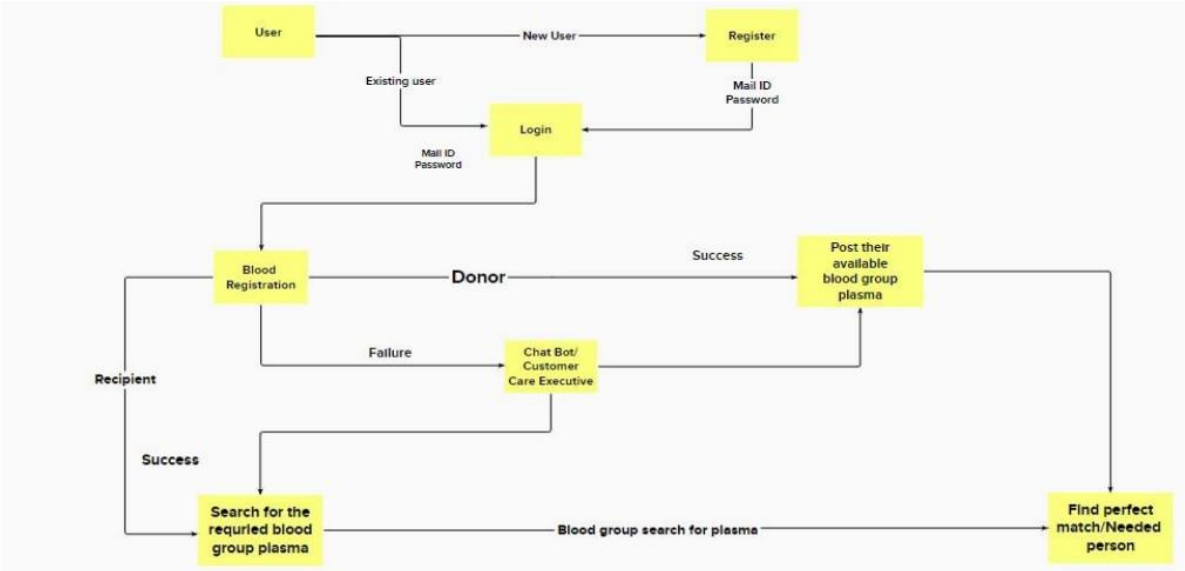
NFR-2	Security	A system's ability to prohibit unauthorized access , usage or behaviour modification while providing service to authorized users
NFR-3	Reliability	It gave the reliable information to the user , because the register donors are well reliable person .So reliability is high.
NFR-4	Performance	Performance Processing speed , response time ,resource consumption, throughput and efficiency
NFR-5	Availability	Made publicly available a new dataset formed by a set of plasma donors profiles and a set of patient collected from different search engine sites
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

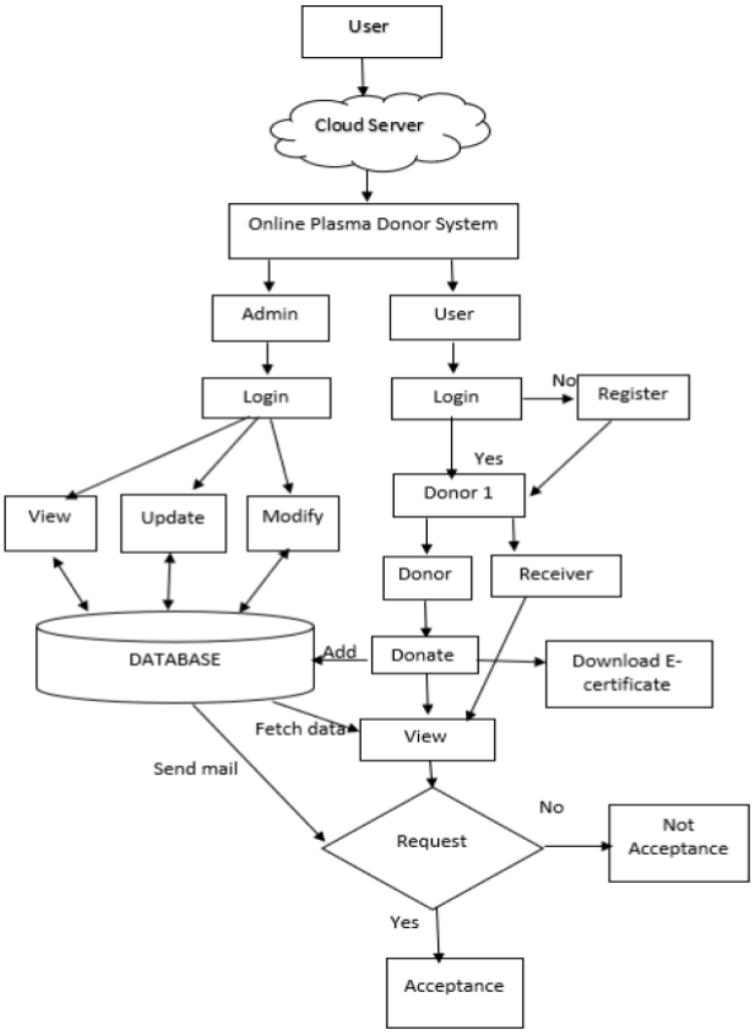
5) PROJECT DESIGN:

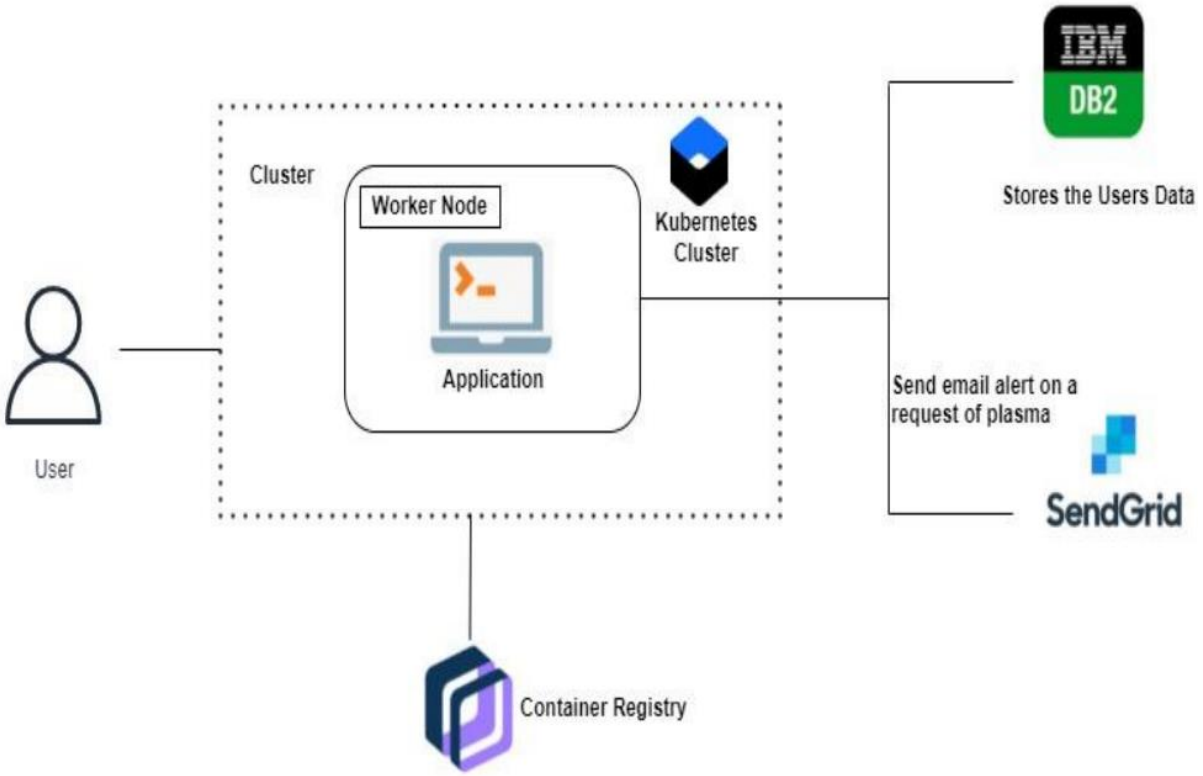
5.1 Data Flow Diagrams:

Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



5.2 Solution & Technical Architecture:





5.3 User Stories:

Use the below template to list all the user stories for the product.

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 register & access thedashboard with Gmail Login	Low	Sprint-2
		USN-4	As a patient, I can directly access the application and find the plasma available bank	I can access my account /dashboard	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password	I can register & access thedashboard with Gmail Login	High	Sprint-1
	Dashboard	USN-6	As a user, I can search the blood group for which I need plasma.	I can get perfectly-matched plasma through filters.	High	Sprint-2
Customer (Web user)	Dashboard	USN-7	As a user, I can see login page and registration page for which the user logins and searches for the required blood group plasma.	I can login through Gmail and register for my required blood group plasma.	Medium	Sprint-2
Customer Care Executive	Dashboard	USN-8	As a user, I can see the availability of donor information	I can update the donor information	High	Sprint-3

6) PROJECT PLANNING & SCHEDULING:

6.1 Sprint Planning & Estimation:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Login	USN-1	Create a registration form a web page for user log in	2	High	Aadityaa B S Raghul P Ramesh K Vijayakumar S
Sprint-1	Login	USN-2	Validating form fields in javascript and linking backend flask	1	High	Aadityaa B S Raghul P Ramesh K Vijayakumar S
Sprint-1	Login	USN-3	Gmail and Facebook external api used to authenticate user	2	Low	Aadityaa B S Raghul P Ramesh K Vijayakumar S

Sprint-1	Database	USN-4	Designing database schema and storing simple user information	2	Medium	Aadityaa B S Raghul P Ramesh K Vijayakumar S
Sprint-1	Login	USN-5	Form to collect donor details created and linked to backend	1	High	Aadityaa B S Raghul P Ramesh K Vijayakumar S
Sprint-2	Chatbot	USN-6	Creating a chatbot using watson assistant	2	High	Aadityaa B S Raghul P Ramesh K Vijayakumar S
Sprint-2	Chatbot	USN-7	Connecting chatbot with IBM DB2 database	1	Medium	Aadityaa B S Raghul P Ramesh K Vijayakumar S
Sprint-2	Dashboard	USN-8	CSS and javascript should be used to create an admin page	1	Low	Aadityaa B S Raghul P Ramesh K Vijayakumar S
Sprint-2	Database	USN-9	Basic dashboard interface for users and admin will be completed	1	Medium	Aadityaa B S Raghul P Ramesh K Vijayakumar S
Sprint-3	Database	USN-10	Storage format of donor information should be designed and implemented	2	High	Aadityaa B S Raghul P Ramesh K Vijayakumar S
Sprint-3	Chatbot	USN-11	Chat should be able to retrieve donor information from database	2	High	Aadityaa B S Raghul P Ramesh K Vijayakumar S

6.2 Sprint Delivery Schedule:

Phase 1 :

- Registration form created and validated
- User database schema designed
- Gmail and email user authentication is done
- Form for donor is created

Phase 2 :

- A chat bot to assist users built
- Admin interface and donor page implementation
- Connecting chatbot to IBM database

Phase 3 :

- UI enhancement for dashboard and user view improved
- Integrating the application with sendgrid
- Donor information retrieved using chatbot

Phase 4 :

- Implementing and testing notification triggers from webpage
- Adding password change features
- Creating docker container and hosting the app

6.3 Reports from JIRA:

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

7.1 Feature 1 :

URL for sprint 1:
https://github.com/IBM-EPBL/IBM-Project-561-1658306943/tree/main/Project%20Development%20Phase/Sprint%201

7.2 Feature 2 :

URL for sprint 2:
https://github.com/IBM-EPBL/IBM-Project-561-1658306943/tree/main/Project%20Development%20Phase/Sprint%202

7.3 Feature 3:

URL for sprint 1:
https://github.com/IBM-EPBL/IBM-Project-561-1658306943/tree/main/Project%20Development%20Phase/Sprint%203

7.4 Feature 4:

URL for sprint 1:
https://github.com/IBM-EPBL/IBM-Project-561-1658306943/tree/main/Project%20Development%20Phase/Sprint%204

8) TESTING :

8.1 Test Cases :

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	Comments	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 : 20190112 80	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 :

Test case ID	Feature Type	Component	Test Scenario	Pre-Requisite	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Commnets
LoginPage_TC_001	Functional	Home Page	Verify user is able to see the Login/Signup popup when user clicked on Login/Signup button		1.Enter URL and click go 2.Click on Login/Signup button 3.Verify login/Singup popup displayed or not		Login/Signup page popup should display	Working as expected	Pass	
LoginPage_TC_002	UI	Home Page	Verify the UI elements in Login/Signup popup		1.Enter URL and click go 2.Click on Login/Signup button 3.Verify login/Singup popup with below UI elements: a.email text box b.password text box c.Login button d.New customer? Create account link		Application should show below UI elements: a.email text box b.password text box c.Login button. d.New customer? Create account link	Working as expected	Pass	Recover Password Feature not yet added
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/Signup button 3.Enter Valid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: charan@gmail.com password: Testing123	User should navigate to user account homepage	Working as expected	Pass	

Test case ID	Feature Type	Component	Test Scenario	Pre-Requisite	Steps To Execute	Test Data	Expected Result	Actual Result	Status
HomePage_TC_006	Functional	Home page	Verify User is able to Sign in With his Details		1.Enter URL and click go 2.Click on Sign in button 3.Redirected to Sign in page 4.Enter valid password and username 5.Click on login button	Username: charan@gmail.c	Application must redirect to proper webpage without delay	Working as expected	Pass
HomePage_TC_007	Functional	Home page	Verify User is able to Register With his Details		1.Enter URL and click go 2.Click on Login/Signup button 3.Enter Valid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: charan@gmail.com password: Testing123 Email : abc@gmail.com PhoneNo:123456789 Sex:-M Blood:B+ Address:123 street ,abc	Application must redirect to proper webpage after verifying the details	Working as expected	Pass
Register_TC_008	UI	Register Page	Verify the UI elements in Login/Signup popup		1.Enter URL and click go 2.Click on Login/Signup button 3.Verify login/Singup popup with below UI elements: a.Name b.email text box c.password text box d.Phone No e.Sex f:Age g:Blood	Username: charan@gmail.com password: Testing123 Email : abc@gmail.com PhoneNo:123456789 Sex:-M Blood:B+ Address:123 street ,abc nagar,india	Application should show below UI elements: a.Name b.email text box c.password text box d.Phone No e.Sex f:Age g:Blood h:Address	Working as expected	Pass

9) RESULTS :

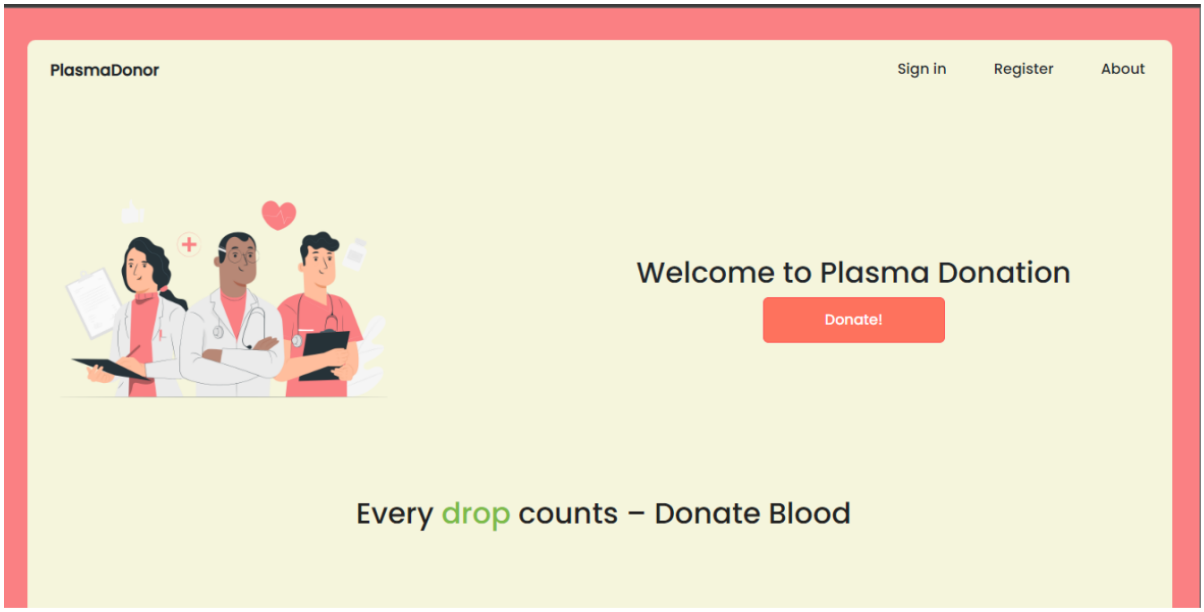
9.1 Performance Metrics :

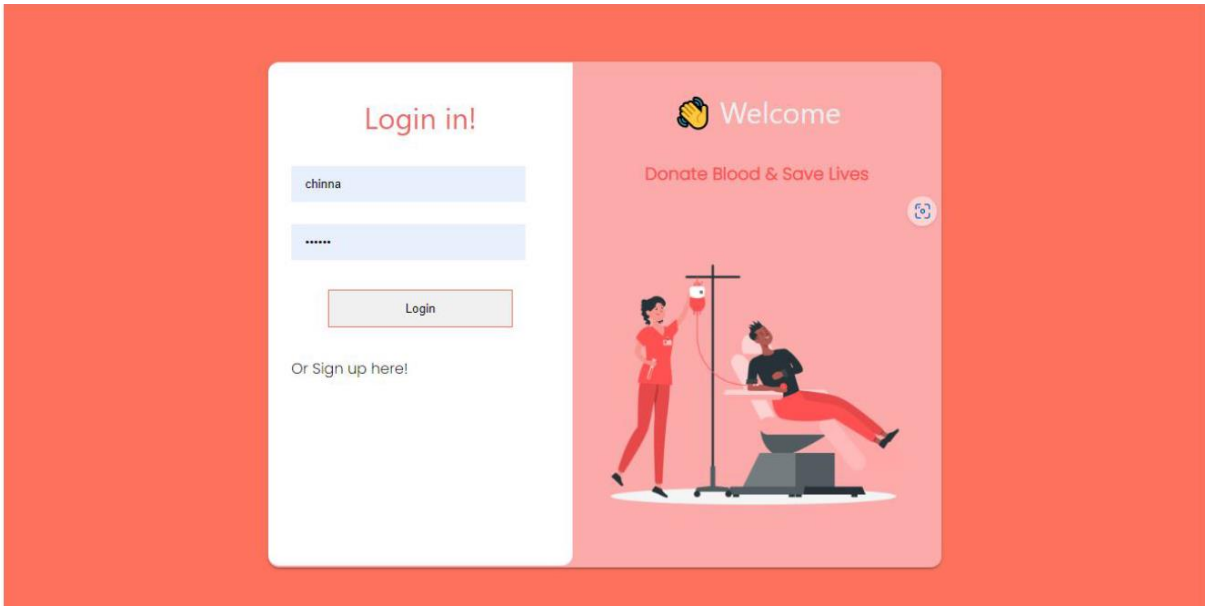
Project metrics are used to track the progress and performance of a project.

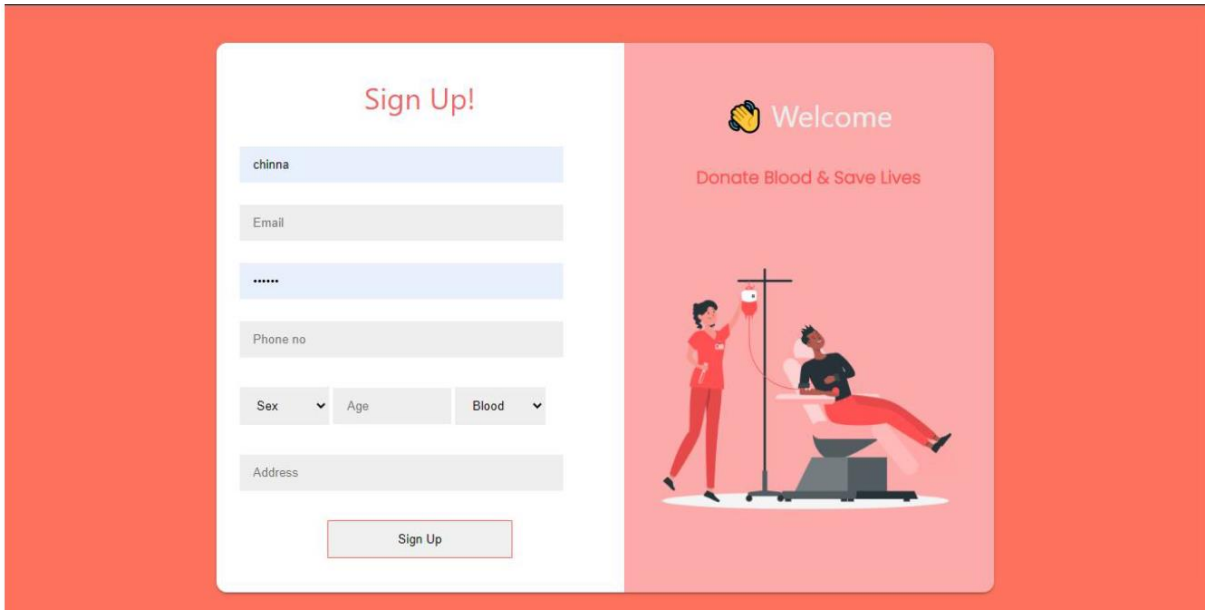
Monitoring parts of a project like productivity, scheduling, and scope make it easier for team leaders to see what's on track.

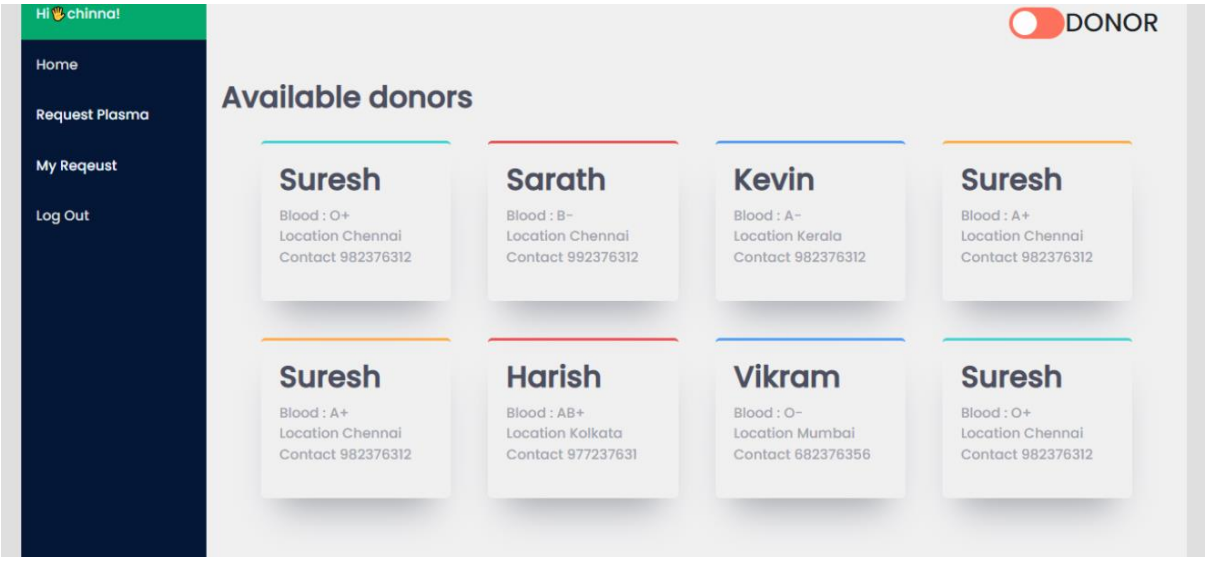
As a project evolves, managers need access to changing deadlines or budgets to meet their client's expectations

OUTPUT SCREEN




Login in!
chinna
Login
Or Sign up here!
Welcome
Donate Blood & Save Lives


Sign Up!
chinna
Email
Phone no
Sex
Age
Blood
Address
Sign Up
Welcome
Donate Blood & Save Lives



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 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 & Kubernates 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.

13) APPENDIX:

13.1) CODE:

```
import json
import os

import ibm_db
from flask import (Flask, jsonify, make_response, redirect, render_template,
                   request, url_for)
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail

from flask import (Flask, jsonify, make_response, redirect, render_template,
                   request, url_for)
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=764264db-9824-4b7c-82df-40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=32536;SECURITY=SSL;SSLServerCertificate=abc.crt;UID=gnq12618;PWD=0glS4tFaR2ciK8fB","","")
print(conn)
print("connection successful...")
app = Flask(__name__,template_folder='template')



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


@app.route('/home')
def dash():
```

```
    return render_template("dashboard.html")


@app.route('/login', methods=['POST', 'GET'])
def login():
    print("login")
    if request.method=='POST':
        username = request.form['username']
        password = request.form['password']
        sql = "select * from user 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)
        dic = ibm_db.fetch_assoc(stmt)
        print(dic)
        role = str()
        requests = []
        if dic:
            role = dic['ROLE']
            # sql = "select * from user where blood_group=?"
            # stmt = ibm_db.prepare(conn, sql)
            # ibm_db.bind_param(stmt, 1, username)
            # ibm_db.execute(stmt)
            # dic = ibm_db.fetch_assoc(stmt)

            # while dic != False:
            #     single_request = {
            #         'name': dic['NAME'],
            #         'age': dic['AGE'],
            #         'sex': dic['SEX'],
            #         'blood_type': dic['BLOOD_TYPE']
            #     }
            #     print(single_request)
            #     requests.append(single_request)
            #     dic = ibm_db.fetch_assoc(stmt)
```

```
            return  render_template('dashboard.html',  username=username,
role=role)


        else:

            return render_template('login.html')
        return redirect(url_for('home'))
    else:
       print("else")
       return render_template('login.html')


@app.route('/signup', methods=['POST', 'GET'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        roll_no = request.form['roll_no']
        sex = request.form['sex']
        age = request.form['age']
        address = request.form['address']
        blood_group = request.form['blood_group']
        sql = "insert into user values(?,?,?,?,?,?,?,?,?)"
        prep_stmt = ibm_db.prepare(conn, 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, roll_no)
        ibm_db.bind_param(prep_stmt, 5, sex)
        ibm_db.bind_param(prep_stmt, 6, age)
        ibm_db.bind_param(prep_stmt, 7, "USER")
        ibm_db.bind_param(prep_stmt, 8, address)
        ibm_db.bind_param(prep_stmt, 9, blood_group)
        ibm_db.execute(prep_stmt)
        # db post operation
```

```
        return redirect(url_for('login'))
    elif request.method == 'GET':
        return render_template('signup.html')


@app.route('/toggle', methods=['POST'])
def toggle_user():
    data = request.get_json(force=True)

    username = data['username']
    role = data['role']
    print(username)
    print(role)
    sql = "update user set role=? where username=?"
    prep_stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(prep_stmt, 1, role)
    ibm_db.bind_param(prep_stmt, 2, username)
    ibm_db.execute(prep_stmt)
    return jsonify(
        status="success",
        role=role
    )


@app.route('/requestPlasma', methods=['POST'])
def requestBloodPlasma():
    #fetch mail address of the donors
    data =  request.get_json(force=True)
    username = data['username']
    name = data['name']
    age = data['age']
    sex = data['sex']
    phone_number = data['phno']
    blood_type = data['blood']
    sql = "select email from user where blood_group=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, blood_type)
```

```
    ibm_db.execute(stmt)
    dic = ibm_db.fetch_assoc(stmt)

    #send mail
    email_list = []
    while dic != False:
        email_list.append(dic['EMAIL'])
        print(dic['EMAIL'])
        dic = ibm_db.fetch_assoc(stmt)
    # send mail
    print(email_list)
    message = Mail(
        from_email='eshwaran.s.2019.cse@rajalakshmi.edu.in',
        to_emails=email_list,
        subject='Blood Need',
        html_content='<h1>Need Of
Blood</h1><table><tr><th>Name</th><th>' + name +
'</th></tr><tr><th>Age</th><th>' + age +
'</th></tr><tr><th>Sex</th><th>' + sex + '</th></tr><tr><th>Blood
Group</th><th>' + blood_type + '</th></tr><tr><th>Phone
Number</th><th>' + phone_number + '</th></tr></table>'
    )
    try:
        sg = SendGridAPIClient("SG.3iBLSgAYTEuVbfSHu9dCPA.-
nrnikWJvaRlNLMONA04_CuKAyPeV69c46vPAh3vUX0")
        response = sg.send(message=message)
        print(response.status_code)
        print(response.body)
        print(response.headers)
    except Exception as e:
        print(e)
    # insert data into requests table
    #insert data into requests table
    sql = "insert into bloodrequests(username,name,age,sex,blood_type)
values (?,?,?,?,?)"
    prep_stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(prep_stmt, 1, username)
```

```
    ibm_db.bind_param(prep_stmt, 2, name)
    ibm_db.bind_param(prep_stmt, 3, age)
    ibm_db.bind_param(prep_stmt, 4, sex)
    ibm_db.bind_param(prep_stmt, 5, blood_type)
    ibm_db.execute(prep_stmt)

    return jsonify(
        name=name,
        age=age,
        sex=sex,
        bloodtype=blood_type,
        status="yes"
    )


@app.route('/getrequests', methods=['POST'])
def getBloodRequests():
    data =  request.get_json(force=True)

    username =data['username']
    sql = "select * from bloodrequests where username=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, username)
    ibm_db.execute(stmt)
    dic = ibm_db.fetch_assoc(stmt)
    requests = []
    print(dic)
    while dic != False:
        single_request = {
            'name': dic['NAME'],
            'age': dic['AGE'],
            'sex': dic['SEX'],
            'blood_type': dic['BLOOD_TYPE']
        }
        print(single_request)
        requests.append(single_request)
        dic = ibm_db.fetch_assoc(stmt)
```

```
    return jsonify(
        username=username,
        requests=requests
    )
@app.route('/form')
def form():
    return render_template("form.html")


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

13.2) GITHUB LINK:

https://github.com/IBM-EPBL/IBM-Project-561-1658306943

13.3) MODULE VEDIO LINK:

https://drive.google.com/drive/folders/1nofzI_G5Na7jFN3SgJuDufYH0eWcKLAX