Team ID:PNT2022TMID17544

Project Name:Plasma Donar Application

CHAPTER 1
INTRODUCTION

1.1 Project Overview

A blood donation occurs when a person voluntarily has blood drawn and used for transfusions and/or made into pharmaceutical medications by a process called fractionation (separation of whole-blood components). Donation may be of whole blood, or of specific components directly (the latter called apheresis). Blood banks often participate in the collection process as well as the procedures that follow it. There are many reasons patients need blood. A common misunderstanding about blood usage is that accident victims are the patients who use the most blood. Actually, people needing the most blood include those: Being treated for cancer, undergoing orthopaedic surgeries, undergoing cardiovascular surgeries, being treated for inherited blood disorders. There are many private sectors as well as NGOs who organize Blood Donation Camps. One can donate the blood in Hospitals also. The safest blood donors are voluntary, nonremunerated blood donors from low-risk populations. The World Health Organization's goal is for all countries to obtain all their blood supplies through voluntary unpaid donors, in accordance with World Health Assembly resolution 28.72, which was adopted back in 1975.

1.2 Purpose

Donors can be individuals and blood banks. Donor users can register to the application to receive notification about blood donation requests when their blood type is required for an admitted patient to a clinic. In the online registration, users need to provide information about their blood type and address. Once the user login, he would be able to see the latest blood donation requests in their city/region. Each notification contains information about the required blood type and the clinic address together with a request

status as pending if the donation is not done yet. If someone has donated, then the request status is marked as success so that potential donors would receive an updated notification indicating that the blood donation has been made and there is no further donation is required for this particular request. Blood donation has a significant impact on iron stores in frequent donors, particularly females. Several measures are necessary to prevent, detect, and treat iron deficiency in donors. These include less frequent donation by donors most susceptible to iron deficiency, and better education of both donors and their physicians about iron needs associated with blood donation. Regular blood donors may require a course of iron supplements to replenish the iron lost in blood donation. These individuals can often return to blood donation, after an adequate course of iron supplementation. As a result, Donor may track his/her donation history details using “Donation History” to avoid such risky intensive donations before that the body can make up its lost red blood cells. Donors can invite friends to register to the application using “Invite Friends” to increase number of donors. When a Donor is notified about a blood request, he/she can book an appointment with the clinic.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing Problem

People who need blood are increasing day by day. People who have diseases like anemia or people who have gotten into accidents and run out of blood need constant supply of blood to sustain their life and there is not enough blood available for them. It is not that people do not want to donate blood, but because they have no idea where they can donate. It is important for the people who are excited to donate, but yet are very busy, to be sure where and when they can donate, and therefore We are designing a system which contains all the information regarding blood donation camps ongoing in a particular area so that people who want to donate blood will get information regarding these camps.

2.2 Reference

[1] https://en.wikipedia.org/wiki/Blood_donation

[2] Why Blood Is Needed, one blood: Share your power.

[3] WHO, 2010.Voluntary Donation, https://www.who.int/bloodsafety/voluntary_donation/en/

[4] http://lifesciencesite.com/lsj/life1108/008_24337life110814_60_65

[5] https://www.who.int/features/qa/61/en/

[6] TeenaC. A, Sankar. K and Kannan. S(2014).A Study on Blood Bank Management. https://www.idosi.org/mejsr/mejsr19(8)14/21.pdf

2.3 Problem Statement Definition

In today's rapid developing scientific world technology has become a very important aspect of life. Today's generation is more depended on advanced technology than any other aspect. Today, most of the people use advance technologies in their daily life like Internet, Smartphone. So, our idea will make the process of searching blood donor in less time consuming by gathering all information of donor and receiver.

In this application there will modules for donor, Receiver and Health and Welfare Department. Donor has to register himself to use this improved system. For Receiver, no need to call in every blood bank to check the blood availability, they can search in our app itself.

The basic solution is to create a centralized system to keep a track on the upcoming as well as past Blood Donation Events.

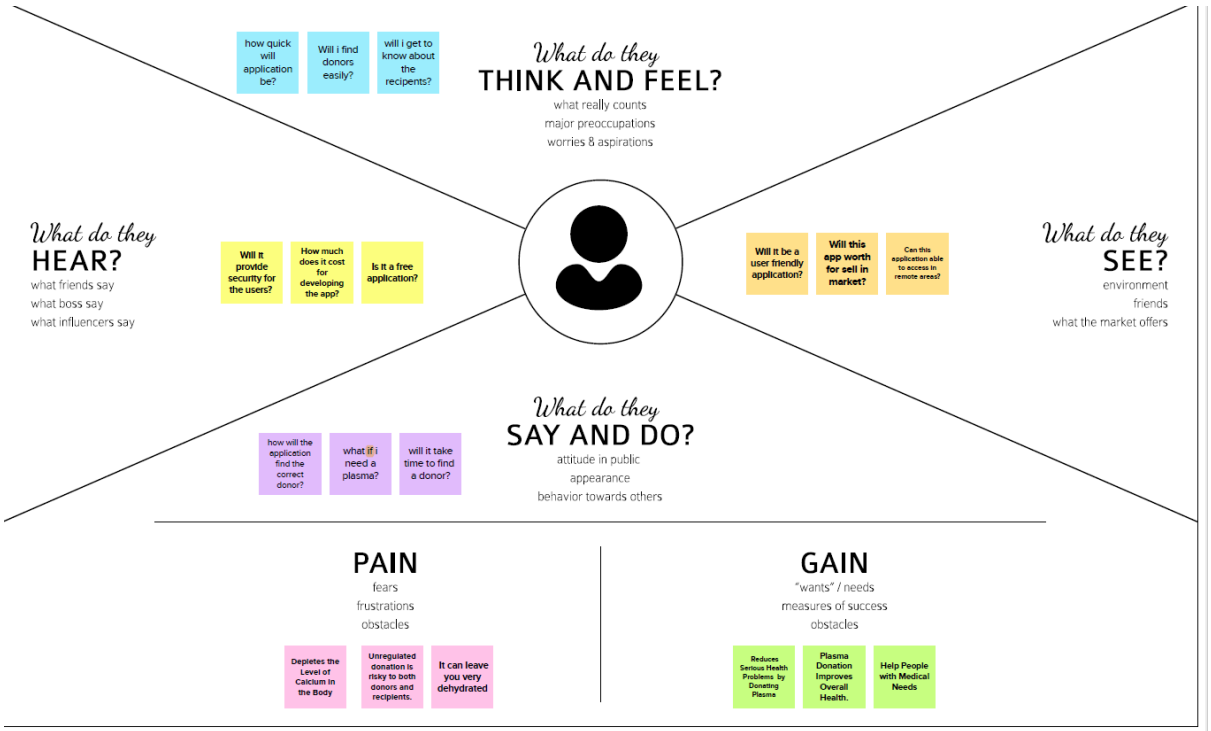
The recommendation solution:

- Donors will save the donor card digitally.
- The Programmer Officer will be able to predict the feasible date in order to get the maximum response.
- An emergency button is introduced in order to guarantee the safety of Donors.
- The call will be dialled automatically to the ambulance once the donor meets an accident.
- A centralized system is designed in order to keep a record of the past and upcoming events.

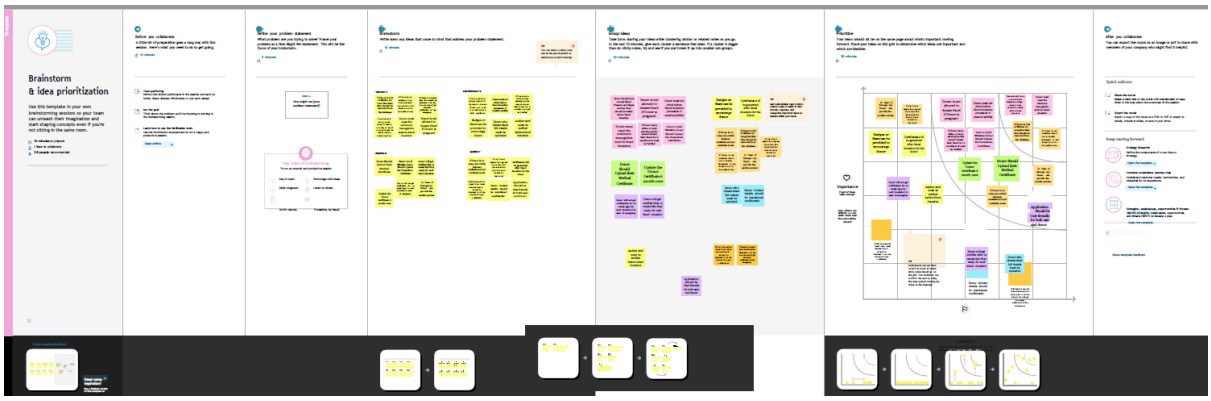
CHAPTER 3

IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



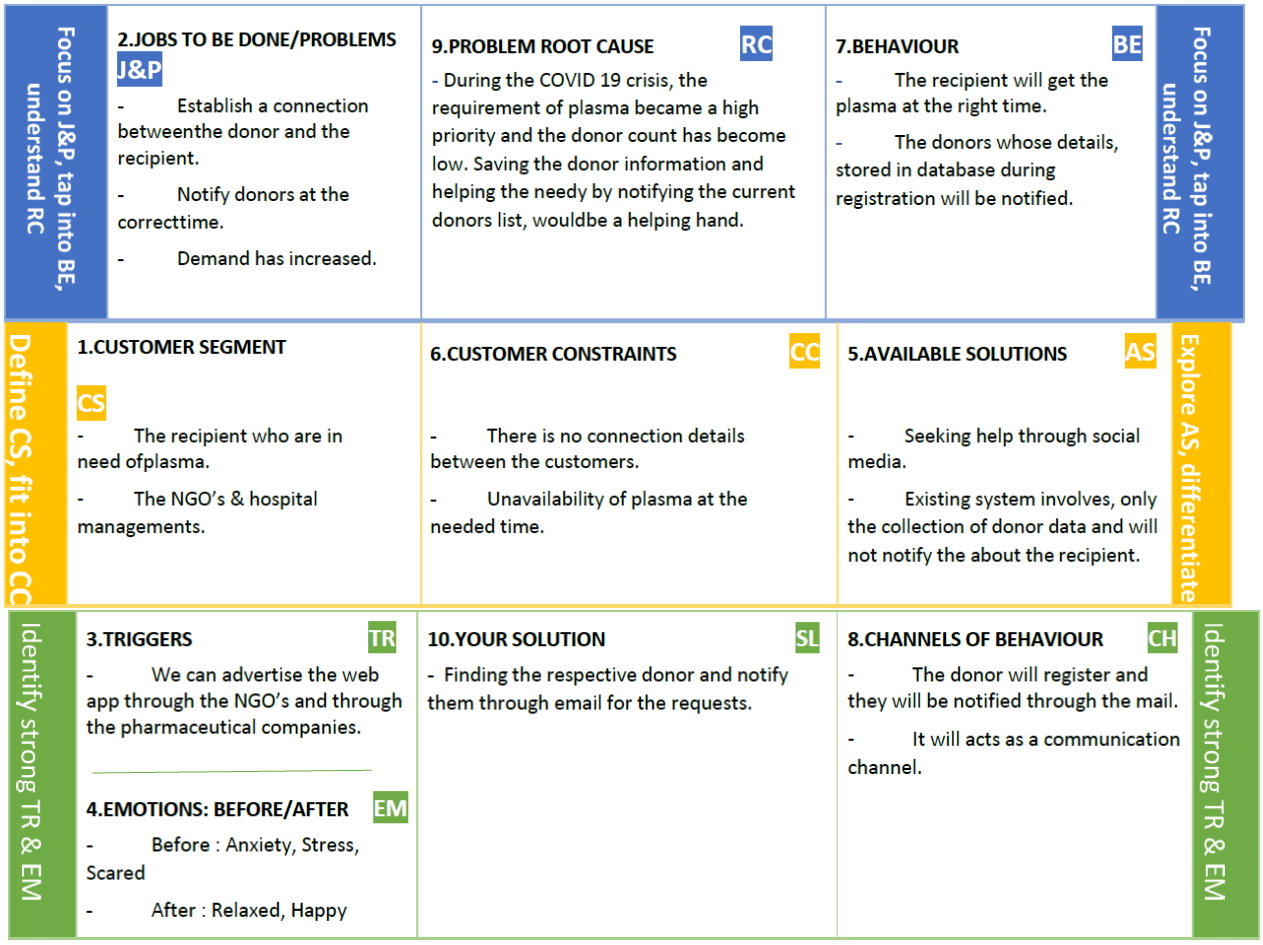
3.2 Ideation & Brainstorming



3.3 Proposed Solution

S. No	Parameter	Description
1.	Problem Statement (Problem to be solved)	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.
2.	Idea / Solution description	In regard to the problem, an application is to be built which would take the donor details, store them and inform them upon a request.
3.	Novelty / Uniqueness	The fresh & responsive interface along with the quick response on giving notification to recipient & donor has been the novelty of the model. A chatbot, which is fully dedicated to that application environment will interact with the users.
4.	Social Impact/ Customer Satisfaction	In society, it will create an awareness among the people about donation of plasma which will be done in an easy way of connecting the donor and the recipient. For sure, this application will satisfy the customers with its services.
5.	Business Model (Revenue Model)	The proposed model can be deployed in collaboration with the NGO's and in turn can yield revenue for each and every requests from the user.
6.	Scalability of the Solution	The model can be deployed in a large scale based on the requirement and usage by the users.

3.4 Problem Solution fit



CHAPTER 4

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

Admin

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

Users

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

Donors Registration

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

Donor Search

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

Notification

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

4.2 Non-Functional requirements

Usability

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

Availability

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

Scalability

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

Security

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

Performance

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

Reliability

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

CHAPTER 5
CONCLUSION

In recent days, it is noticed the increase in blood request posts on social media such as Facebook, Twitter, and Instagram. Interestingly there are many people across the world interested in donating blood when there is a need, but those donors don't have an access to know about the blood donation requests in their local area. This is because that there is no platform to connect local blood donors with patients. Plasma Donor Application solves the problem and creates a communication channel through authorized clinics whenever a patient needs blood donation. It is a useful tool to find compatible blood donors who can receive blood request posts in their local area. Clinics can use this web application to maintain the blood donation activity. Collected data through this application can be used to analyse donations to requests rates in a local area to increase the awareness of people by conducting donations camps.

APPENDIX

```
//1
 from flask import
 Flask,redirect,url_for,render_template,request,make_response,jsonify
 import ibm_db
 conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=764264db-9824-
 4b7c-82df-
 40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=3253
 6;SECURITY=SSL;SSLServerCertificate=abc.crt;UID=gnq12618;PWD=0glS4
 tFaR2ciK8fB",",")
 print(conn)
 print("connection successful...")
 app = Flask(__name__)
 @app.route('/home')
 def home():
    return render_template("home.html")
 @app.route('/login',methods=['POST','GET'])
 def 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)
            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,
            role=role,
            donors = requests
        )
    else:
        return redirect(url_for('login'))
    return redirect(url_for('home'))
elif request.method=='GET':
    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=['PUT'])
def toggle_user():
```

```
    username = request.form['username']
    role = request.form['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('/requestPalsma',methods=['POST'])
def requestBloodPlasma():
    #fetch mail address of the donors
    username = request.form['username']
    name = request.form['name']
    age = request.form['age']
    sex = request.form['sex']
    blood_type = request.form['bloodtype']
    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)
    while dic!=False:
        print(dic['email'])
    #send mail
    #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():
    username = request.form['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(type(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
    )
if __name__=='__main__':
    app.run(debug = True)
```

//2

```
from flask import
Flask,redirect,url_for,render_template,request,make_response,jsonify,request
import ibm_db
from flask import request
import json
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=764264db-9824-
4b7c-82df-
40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=3253
6;SECURITY=SSL;SSLServerCertificate=abc.crt;UID=gnq12618;PWD=0glS4
tFaR2ciK8fB",",")
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():

    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)

        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
    username = request.form['username']
    name = request.form['name']
    age = request.form['age']
    sex = request.form['sex']
    blood_type = request.form['bloodtype']
    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)
    while dic!=False:
        print(dic['email'])
    #send mail
    #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
    )
if __name__=='__main__':
    app.run(host="0.0.0.0",debug = True)
```

//3

```
from flask import Flask, redirect, url_for, render_template, request,
make_response, jsonify, request
import ibm_db
from flask import request
import json

conn = ibm_db.connect(
    "DATABASE=bludb;HOSTNAME=764264db-9824-4b7c-82df-
40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=3253
6;SECURITY=SSL;SSLServerCertificate=abc.crt;UID=gnq12618;PWD=0glS4
tFaR2ciK8fB",
    ", ")
print(conn)
print("connection successful...")
app = Flask(__name__)
import os
from sendgrid import SendGridAPIClient
```

```
from sendgrid.helpers.mail import Mail

@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():
    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 redirect(url_for('login'))
        return redirect(url_for('home'))
    elif request.method == 'GET':
        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('/requestPalsma', 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']
    blood_type = data['bloodtype']
    phone_number = data['phone_num']
    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)
    email_list = []
    while dic != False:
        email_list.append(dic['EMAIL'])
        print(dic['EMAIL'])
        dic = ibm_db.fetch_assoc(stmt)
    # send mail

    message = Mail(
        from_email='eshwaran.s.2019.cse@rajalakshmi.edu.in',
        to_emails=email_list,
        subject='Sending with Twilio SendGrid is Fun',
```

```
        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)
        print(response.status_code)
        print(response.body)
        print(response.headers)
    except Exception as e:
        print(e.message)
    # 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(type(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
    )


if __name__ == '__main__':
    app.run(host="0.0.0.0", debug=True)
//4
from flask import Flask, redirect, url_for, render_template, request,
make_response, jsonify, request
import ibm_db
from flask import request
import json

conn = ibm_db.connect(
    "DATABASE=bludb;HOSTNAME=764264db-9824-4b7c-82df-
40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=3253
6;SECURITY=SSL;SSLServerCertificate=abc.crt;UID=gnq12618;PWD=0glS4
tFaR2ciK8fB",
    '', '')
print(conn)
print("connection successful...")
app = Flask(__name__)
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail


@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():
    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 redirect(url_for('login'))
        return redirect(url_for('home'))
    elif request.method == 'GET':
        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('/requestPalsma', 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']
    blood_type = data['bloodtype']
    phone_number = data['phone_num']
    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)
    email_list = []
    while dic != False:
        email_list.append(dic['EMAIL'])
        print(dic['EMAIL'])
        dic = ibm_db.fetch_assoc(stmt)
    # send mail

    message = Mail(
        from_email='eshwaran.s.2019.cse@rajalakshmi.edu.in',
        to_emails=email_list,
        subject='Sending with Twilio SendGrid is Fun',
        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)
        print(response.status_code)
        print(response.body)
        print(response.headers)
    except Exception as e:
        print(e.message)
    # 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(type(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
    )


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