# PLASMA DONOR APPLICATION

TEAM ID: PNT2022TMID31815


TEAM MEMBERS:

MATHAN RAJ A R

VITHYA M

AUSTIN S S

ARUN S

KARPAGAM INSTITUTE OF TECHNOLOGY, COIMBATORE

# Project Report Format

1. **INTRODUCTION**
   - 1.1 Project Overview
   - 1.2 Purpose
2. **LITERATURE SURVEY**
   - 2.1 Existing problem
   - 2.2 References
   - 2.3 Problem Statement Definition
3. **IDEATION & PROPOSED SOLUTION**
   - 3.1 Empathy Map Canvas
   - 3.2 Ideation & Brainstorming
   - 3.3 Proposed Solution
   - 3.4 Problem Solution fit
4. **REQUIREMENT ANALYSIS**
   - 4.1 Functional requirement
   - 4.2 Non-Functional requirements
5. **PROJECT DESIGN**
   - 5.1 Data Flow Diagrams
   - 5.2 Solution & Technical Architecture
   - 5.3 User Stories
6. **PROJECT PLANNING & SCHEDULING**
   - 6.1 Sprint Planning & Estimation
   - 6.2 Sprint Delivery Schedule
   - 6.3 Reports from JIRA
7. **CODING & SOLUTIONING (Explain the features added in the project along with code)**
   - 7.1 Feature 1
   - 7.2 Feature 2
   - 7.3 Database Schema (if Applicable)
8. **TESTING**
   - 8.1 Test Cases
   - 8.2 User Acceptance Testing
9. **RESULTS**
   - 9.1 Performance Metrics
10. **ADVANTAGES & DISADVANTAGES**
11. **CONCLUSION**
12. **FUTURE SCOPE**
13. **APPENDIX**

    Source Code

    GitHub & Project Demo Link

# 1.INTRODUCTION

## 1.1 Project Overview:

The Plasma Donor application is to create an e-information about the donor and organization that are related to donating the plasma. Through this application any person who is interested in donating the blood can register himself/herself in the same way if any organization wants to register itself with this site that can also register. Moreover, if any general consumer wants to make request blood online, he can also take the help of this site or web app. Admin is the main authority who can do addition, deletion, and modification if required.

This project is aimed to developing an “Online Blood Donation Information”. The entire project has been developed keeping in view of the distributed client server computing technology in mind.

The project has been planned to be having the view of distributed architecture with centralized storage of the database. The application for the storage of the data has been planned using the

Aim of this project is to provide a user friendly and interactive service via web interface, mobile application and emails. As soon as any updates occur in the blood database these changes are reflected in all the above-mentioned interfaces. So, by this manner this system provides a simple and quicker interaction among various groups connected with the blood bank.

Also, our goal is to develop a web-based system to manage blood requisition within the blood supply chain. The system is designed to overcome the drawbacks of existing system problem. The main objective is to improve the efficiency of data communication within the supply chain to reduce response time for each blood demand request. We also focused on managing blood inventory at blood bank effectively. The results have shown that the proposed system helps enhancing the communication among blood partners within the supply chain network. The recipient can get blood in emergency case also. We also provide SMS facility to donors so that they can reach to exact location.

## 1.2 Purpose:

There is no other redeeming act than to save a human life. People may feel afraid or selfish when it comes to donating blood. But if everybody thinks that way, then doctors may be unable save so many human lives. People who have never donated blood may themselves require blood at some point of their life. But think what will happen if everybody feels unwilling to donate blood. There will be no blood available in the blood banks. So many precious lives will be wasted. It may happen to anyone, even you. So don’t be afraid or selfish about donating blood. Blood is the fuel of life. In India, blood is required in every 2 seconds. More awareness should be created about blood donation so that more and more people come forward to donate blood. If human lives are wasted because of the dearth of blood in the blood banks it will be shame to the human society. So, donate blood and encourage people as well.

# 2.LITERATURE SURVEY

## 2.1 Existing Problems:

**A Web-based blood donor management information system for the Red Cross Society, Uganda (WBBDMI)**, describes about developing a blood management information system to assist in the management of blood donor records and control the distribution of blood in various parts of the Uganda based on the hospital demands.

The blood management information system offers functionalities to quick to donor records collected from various parts of the Uganda. It enables monitoring of the measurable objectives of the organization can be checked. It provides to management timely, confidential and secure medical reports that facilitates planning and decision making and hence improved medical service delivery.

**Telemedicine, Telehealth, and Health Information Technology,** describes about providing a tool for ATA (American Telemedicine Association) members to identify opportunities for collaboration and to better understand the interaction between telemedicine/telehealth services and health information technology (HIT) applications on local, regional and national levels. Paper presents a framework for discussing the interdependency of both telemedicine and HIT.

**An Efficient Emergency, Healthcare and Medical Information System,** explains about present situation of Malaysian medical information and emergency systems and their information processing system, and also focuses on developing an integrated Emergency, Healthcare and Medical Information System (IEHMS) that can overcome many of the problems in the current systems in Malaysia.

The main goal of this system is to incorporate the real-time and mobility technologies with medical emergency systems, proposed system of this paper can offer. SMS, MMS, live audio and video coverage.

**Blood Bank Management Information System in India,** describes about the review of the main features, merits and demerits provided by the existing Web-based information system for blood banks and shows the comparison of various existing system and provide some more idea for the existing system.

And explains about some basic introduction about blood banks then try to provide comparative study of some existing web-based blood bank system. This system introduces some new idea for improving the existing techniques used in web-based blood bank system.

## 2.2 References:

1.https://nevonprojects.com/instant-plasma-donor-recipient-connector-android-app/

2. https://www.semanticscholar.org/paper/A-web-based-blood-donor-management-information-for-Kanobe/794239ceac555945c541d3c2e28ff11b2

The above-mentioned systems are referred from the online paper publishment sites.

## 2.3 Problem Statement Definition:

The Online Blood Donation Management System, the purpose of which is to act as a bridge between a person who needs blood, a patient, and a blood donor. The design of an automatic blood system has become an integral part for saving the human lives, who need the blood under different situations. Since, there are various drawbacks of the pre-existing system like privacy issues for the donors, which are getting reflected directly on the interface. Thus, we have designed a robust system that will create a connection between different hospitals, NGOs, and blood banks to help the patient in any difficult situation. Thus, HIPPA model provides a backbone for security breaches The interface designed will be easy-to-use and easy to access and will be a fast, efficient, and reliable way to get lifesaving blood, totally free of charge. Apart from this the visualization of the data is present along with the one extra COVID module, which will help covid and normal patients for plasma donation. The main aim of the paper is to reduce the complications of finding a blood donor during panic situations and provide a high level of security for the donors.

**Beneficiaries of blood bank management information system:**

There are three beneficiaries which can get benefits from the management information system of blood bank which are:

1. Donors: person who wants to donate the blood voluntarily at the blood donation camp. Information system also keeps the record of the donors who wants to register online.
2. Seekers: person who wants the blood from the blood bank due to various reasons like accidents, surgeries, delivery and many more.
3. Blood bank: staff people which are working in the blood bank which includes staff member, operator, blood bank in charge, head of pathological department.

**Benefits of blood bank management information system to donors**:

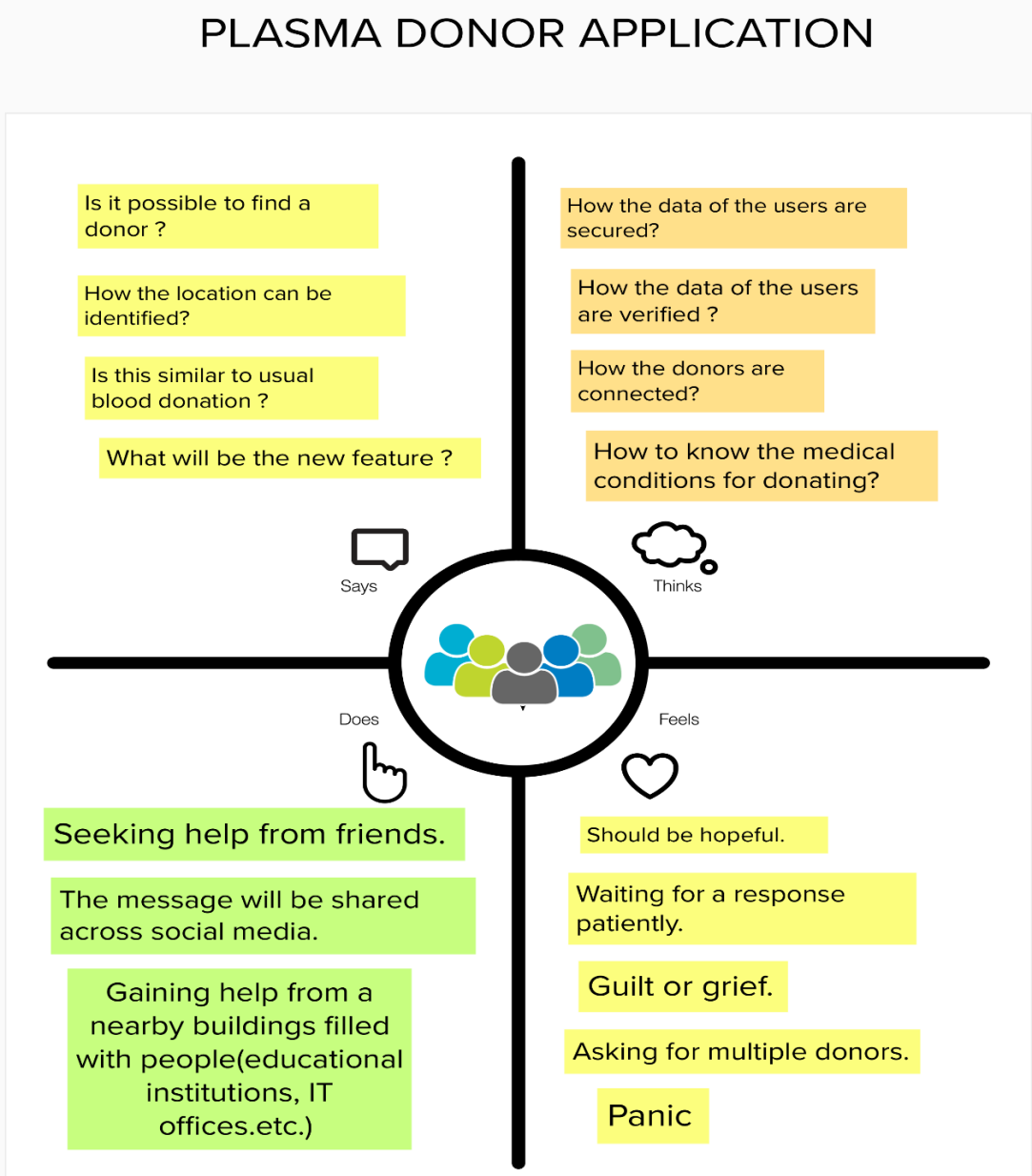
1. It provides the unique identification number at the time of blood donation camp which helps him for the future correspondence. MIS gives the unique user id and password for those donors who are applying online. They can edit their information time to time. This feature helps administrator to collect the information of all the donor's area wise and blood group wise.
2. Donors can view the blood donation camp organizing at the different places.
3. As it is a web-based application, its index page encourages the donor to donate the blood.
4. Donor can also check his blood group medical status whether it is healthy or unhealthy.
5. Donor can check the status of the particular blood group just on one click sitting at home.

**Benefits of blood bank management information system to seekers:**

1. Seeker can get the information of the desired blood group from the central inventory.
2. Seeker can get the list of donors' area wise, blood group wise if the desired blood group is not available in the central inventory.
3. Seeker can get the information of the particular blood group available in the blood bank.
4. Seeker can get the information of that blood group which is not fit for blood transfusion.
5. Seeker can get the blood units according to his requirement from the blood bank.

# 3. IDEATION PHASE

## 3.1 Empathy Canvas:

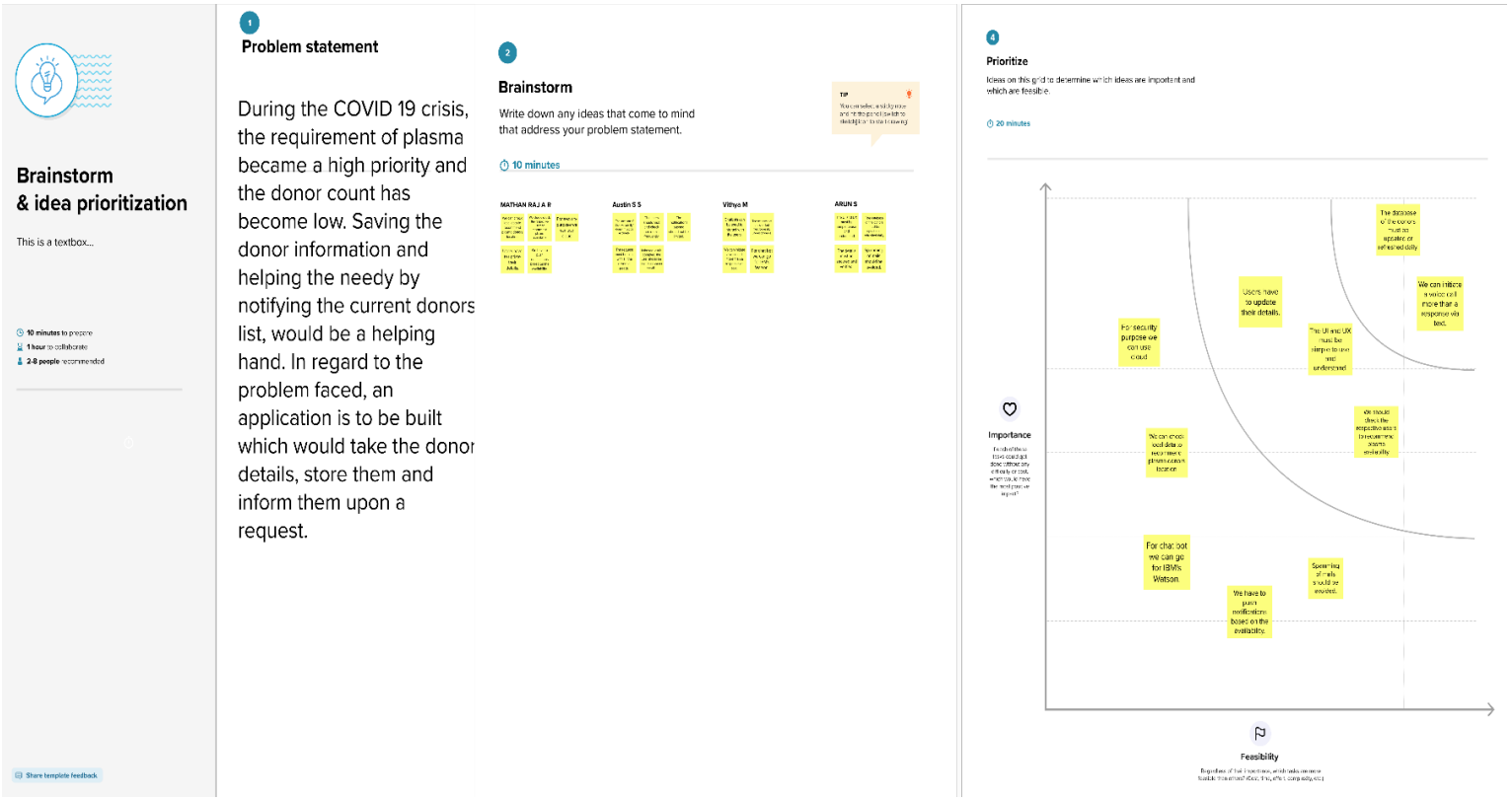


**EMPATHY MAP DESCRIPTION:**

Empathy map used to define the customers of how the product helps in their possessions or thoughts. It contains four sectors – Says, Thinks, Does, Feels.

Here we defined those four sectors by including the thoughts, it explained what the user's first thought while hearing about our project. It also includes what are the clarification to be done before initiating the usage of the app. This map also consists the initial measures and actions done by the seeker to contact the donor and also from other sources.

## 3.2 Ideation & Brainstorming:




We organized a brainstorming session. Through the session we defined the problems occuring nowadays due to the deaths caused by the lack of plasma donation. We came up with many ideas for solving those donor/seeker problems. From those ideas we prioritized some most neccesary ideas and concluded with the session.

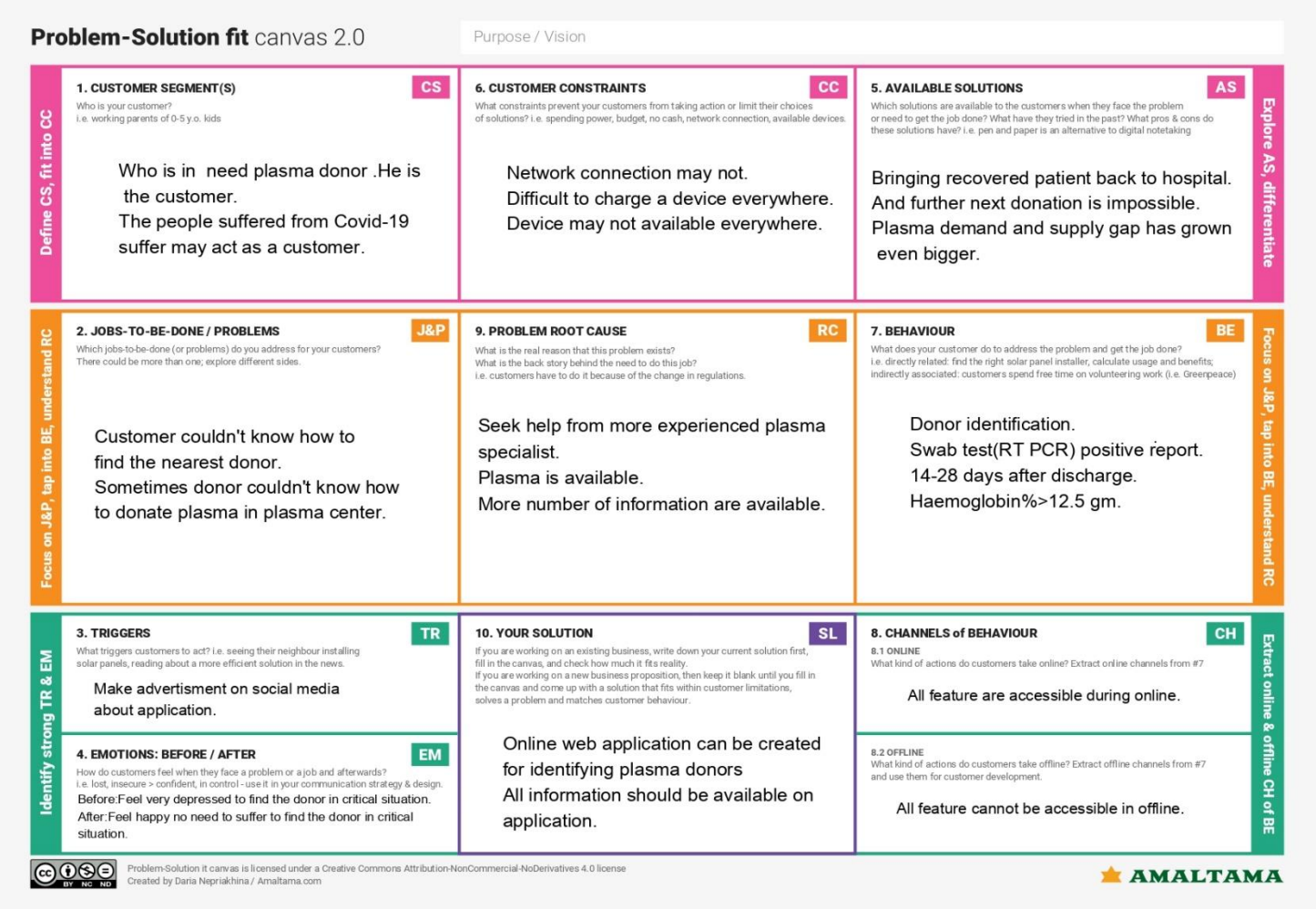
The main aim of the session is to give an user friendly and easily interactive site or app without lack of responses and also we have given importance for the communication between the donor and the seeker.

## 3.2 Proposed Solution:

| S.No. | PARAMETER | DESCRIPTION |
|---|---|---|
| 1. | Problem statement (Problem to be solved) | Nowadays, blood banks are the most essential inventories among all the other inventories. As there are chances of occurring new biological problems or new kind of health difficulties, there will be abundant need of blood for those infected persons. Our idea holds the information about the donors in a perfectly sorted manner. These details are updated regularly and they are completely verified. |
| 2. | Idea/Proposed Solution | For solving the problem, we came up with an idea of developing a cloud-based web application. This app contains the details of the donor and based on the request we receive a notification will be pushed to the requestors with the required details. |
| 3. | Novelty/Uniqueness | The requestors will receive all the essential details based upon their request. The location of the donor is also sent to the requestor. Before the donor is allowed to donate the COVID recovery certificate is verified. The user details are secured and saved by registering and can be logged in if needed. |
| 4. | Social Impact/Customer satisfaction | Even the users with a basic knowledge can access the website or web app easily. The application provides a direct connection between the donor and the requestor. |
| 5. | Business model (Revenue Model) | The web app requires finance source only for the cloud access and no other finance requirement is needed especially for users. |

| | | |
|---|---|---|
| 6. | Scalability of the solution | The app gets collaborated with the government organisations and the blood camping happening around certain region. Since cloud is used for storing data of the users a large number of data can be stored without any problem and can be retrieved without any difficulties. |

## Problem Solution Fit:



# 4. REQUIREMENT ANALYSIS

## 4.1 Functional Requirement:

**Functional Requirements:**

Following are the functional requirements of the proposed solution.

| FR No. | Functional Requirement (Epic) | Sub Requirement (Story / Sub-Task) |
|---|---|---|
| FR-1 | **Access Website** | Software operator should be capable to access web-application through either an application browser or similar on the pc. |
| FR-2 | **Software operator Registration** | The software operator should be able to register through the web-application.The donor software operator must provide user name,gender,blood/plasma group,location,contact. |
| FR-3 | **Login/logout/update details** | The login information will be stored on the database for future use. |
| FR-4 | **Search for donor** | Search result can be viewed in a list.Each element in the list represents a specific donor with the donor details. |
| FR-5 | **View request** | The plasma bank should be able to view received request and then respond to them. |
| FR-6 | **View distribution details** | The plasma bank should be able to view the status of the distribution details. |

## 4.2 Non-functional Requirements:

**Non-functional Requirements:**

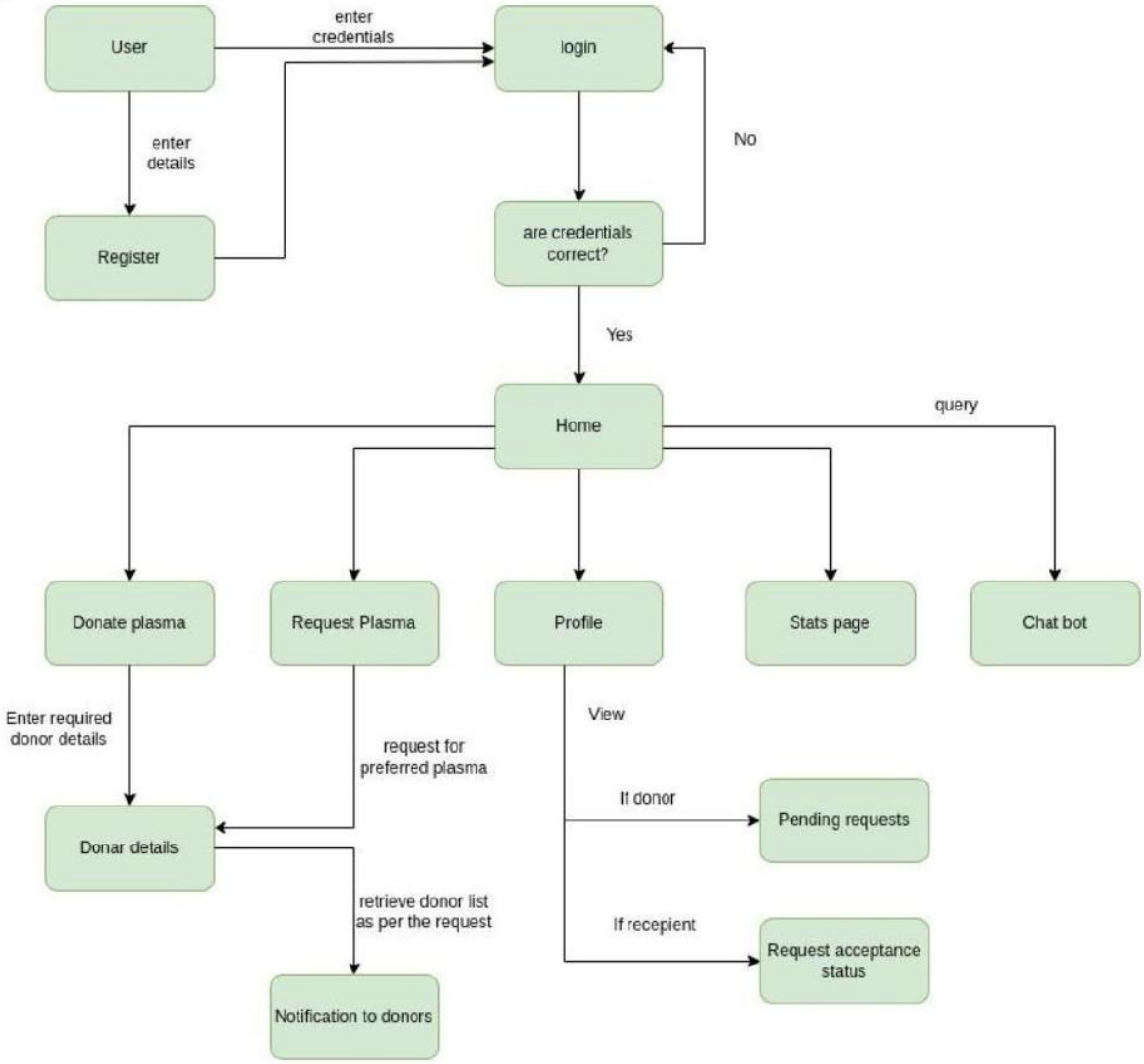
Following are the non-functional requirements of the proposed solution.

| FR No. | Non-Functional Requirement | Description |
|---|---|---|
| NFR-1 | **Usability** | The plasma donor application must have a good looking user friendly interface. |
| NFR-2 | **Security** | The plasma donor application must be secured with proper user name and passwords. |
| NFR-3 | **Reliability** | The plasma donor application should work properly,even when faults occur. |
| NFR-4 | **Performance** | The plasma donor application must perform well in different scenarios. |
| NFR-5 | **Availability** | The plasma donor application must available 24 hours a day with no bandwidth issues. |
| NFR-6 | **Scalability** | The plasma donor application should able to increase or decrease in performance and cost in response to changes in application and system processing demands. |

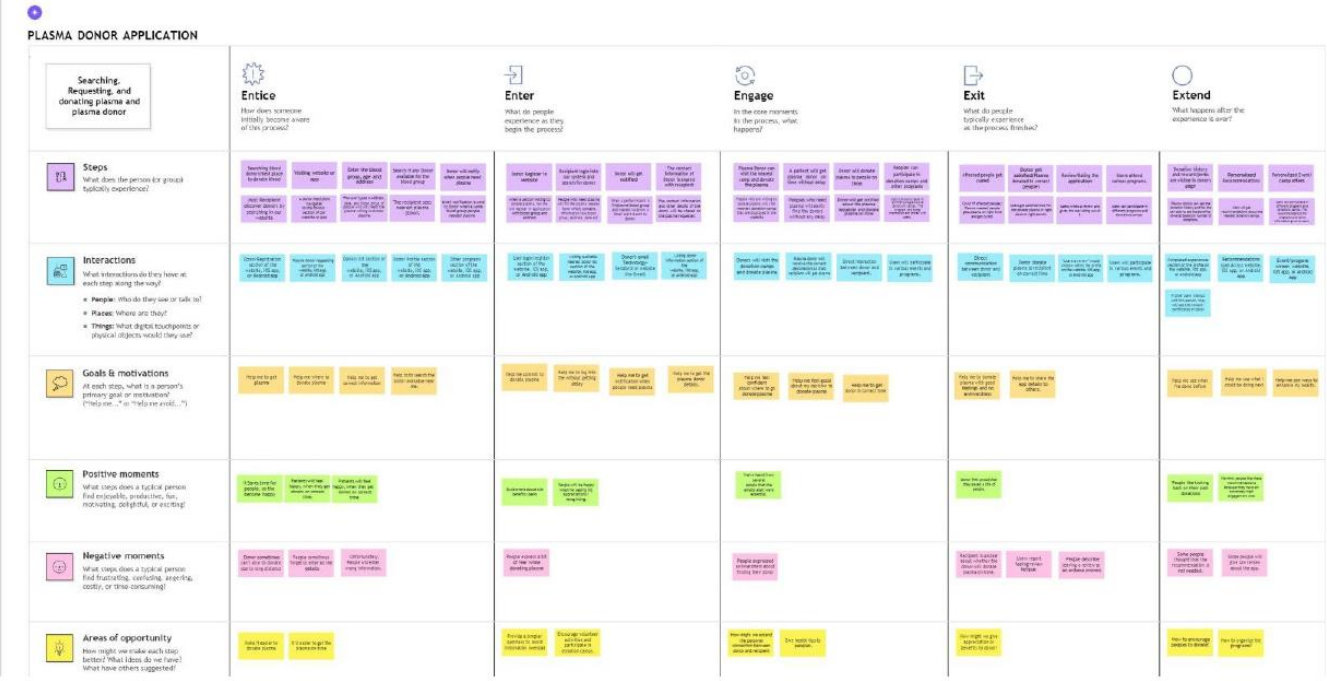
# 5. PROJECT DESIGN

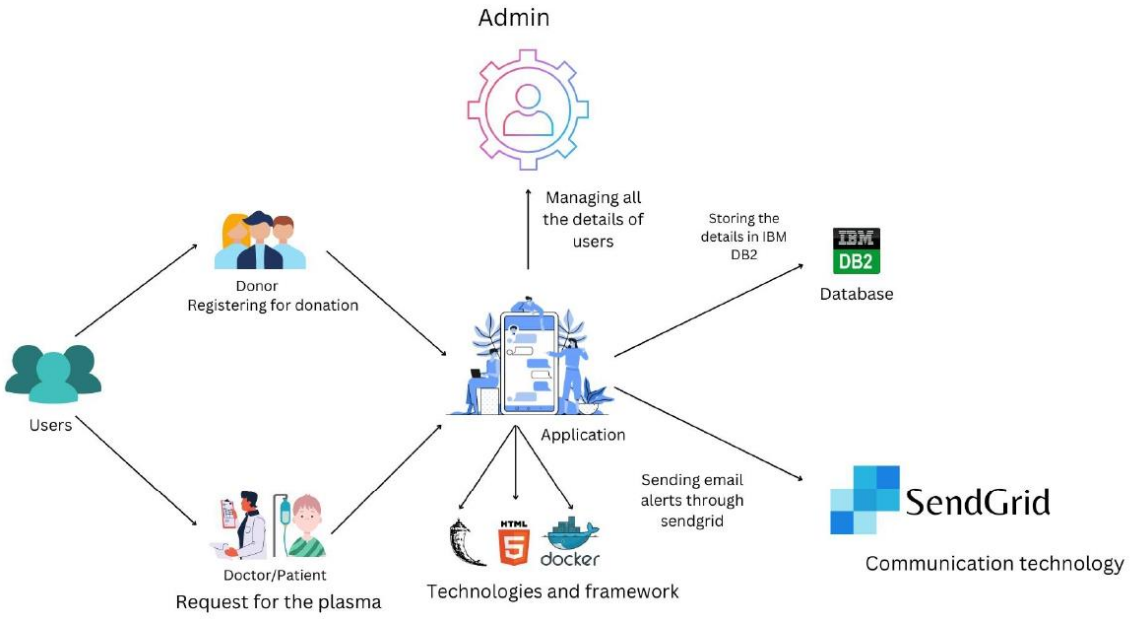
## 5.1 Data Flow Diagrams:

Data Flow Diagrams:



## 5.2 Solution & Technical Architecture:



## 5.3 User Stories:

**User Stories**

| User Type | Functional Requirement (Epic) | User Story Number | User Story / Task | Acceptance criteria | Priority | Release |
|---|---|---|---|---|---|---|
| Customer (Mobile user) | Registration | USN-1 | As a user, I can register for the application by entering my email, password, and confirming my password. | I can access my account / dashboard | High | Sprint-1 |
| | | USN-2 | As a user, I will receive confirmation email once I have registered for the application | I can receive confirmation email & click confirm | High | Sprint-1 |
| | | USN-3 | As a user, I can register for the application through Gmail | I can receive confirmation notifications through Gmail | Medium | Sprint-1 |
| | Login | USN-4 | As a user, I can log into the application by entering email & password | I can access into my User profile and view details in dashboard | High | Sprint-1 |
| | Dashboard | USN-5 | As a user,I can donate and request plasma. | I can receive appropriate notifications through email | High | Sprint-1 |
| Customer (Web user) | Login | USN-6 | As a user, I can register and log into the application by entering email & password to view the profile | I can access into my user profile and view details in dashboard | High | Sprint-1 |
| | Dashboard | USN-7 | As a user,I can donate and request plasma. | I can receive appropriate notifications through email | High | Sprint-1 |
| Customer Care Executive | Application | USN-8 | As a customer care executive, I can try to address user's concerns and questions | I can view and address their concerns and questions | Medium | Sprint-2 |
| Administrator | Application | USN-9 | As an administrator, I can listen to feedbacks and make the user interface more friendly and make complex process simple. | I can change the appearance and navigation in a user friendly manner | Medium | Sprint-3 |
| | | USN-10 | As an administrator, I can involve working with the technical side of websites. | I can help with troubleshooting issues, fixing bugs and provide a seamless experience. | Medium | Sprint-1 |
| Chatbot | Dashboard | USN-11 | In addition the Customer care executive,chatbot can try to address user's concerns and questions | It can reply to all the queries related to our application | Medium | Sprint-3 |

# 6. PROJECT PLANNING & SCHEDULING

## 6.1 Sprint Planning & Estimation:

Product Backlog, Sprint Schedule, and Estimation (4 Marks):

| Sprint | Functional Requirement (Epic) | User Story Number | User Story / Task | Story Points | Priority | Team Members |
|---|---|---|---|---|---|---|
| **Sprint-1** | Registration | USN-1 | As a user, I can register for the web app by entering my email, password, and confirming mypassword. | 20 | High | 1.Mathan Raj A R<br>2.Austin S S<br>3.Vithya M<br>4. Arun S |
| **Sprint-2** | Login | USN-2 | As a user, I can log into the application by entering email & password. | 20 | High | 1.Mathan Raj A R<br>2.Austin S S<br>3.Vithya M<br>4. Arun S |
| **Sprint-3** | Donor Information | USN-3 | Donors can update their personal information e.g.: blood group. | 20 | High | 1.Mathan Raj A R<br>2.Austin S S<br>3.Vithya M<br>4. Arun S |

Velocity:

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

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

## 6.2 Sprint Delivery Schedule:

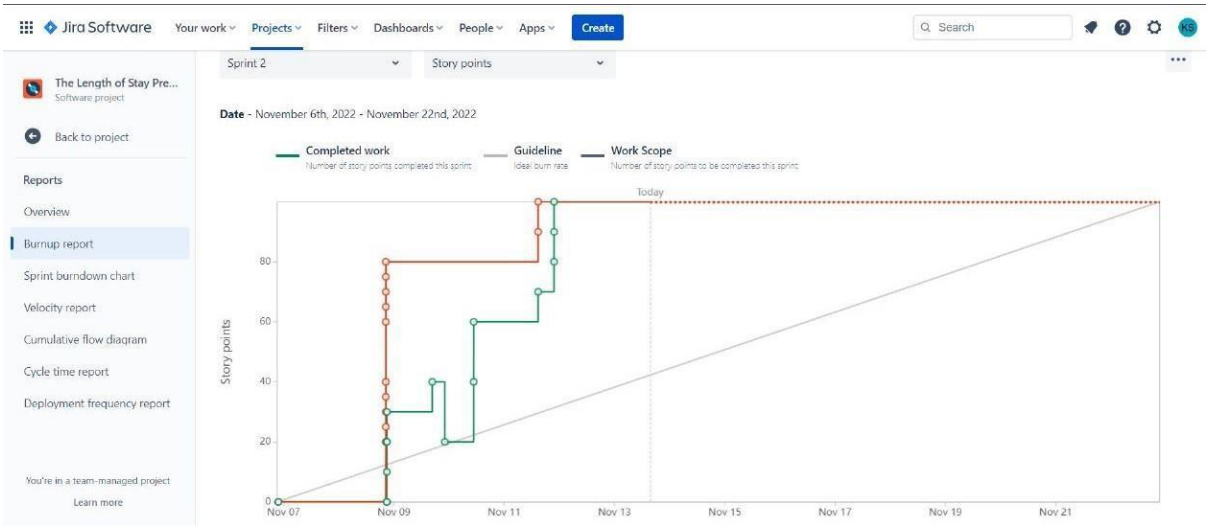
| TITLE | DESCRIPTION | DATE |
|---|---|---|
| Literature Survey & Information Gathering | Literature Survey on the selected project & gathering information by referring the technical papers, research publications, journals etc. | 29 August 2022 |
| Prepare Empathy map | Prepare Empathy map canvas to capture the user Pains & gains, prepare list of problem statements that are to be solved by this project. | 5 September 2022 |
| Ideation | List the ideas by organizing a brainstorm session and prioritize the top 3 ideas based on the feasibility & importance. | 12 September 2022 |
| Proposed Solution | Prepare the proposed solution document, which includes novelty, feasibility of idea, revenue model, social impact, scalability of solution, etc. | 19 September 2022 |
| Problem Solution Fit | Prepare problem – solution fit document. | 26 September 2022 |
| Solution Architecture | Prepare solution architecture document. | 1 October 2022 |
| Customer Journey | Prepare the customer journey maps to understand the user interactions & experiences with the application (entry to exit). | 3 October 2022 |
| Functional Requirement | Prepare the functional requirement document. | 10 October 2022 |
| Data Flow Diagrams | Draw the data flow diagrams and submit for review. | 15 October 2022 |
| Technology Architecture | Prepare the technology architecture diagram. | 15 October 2022 |

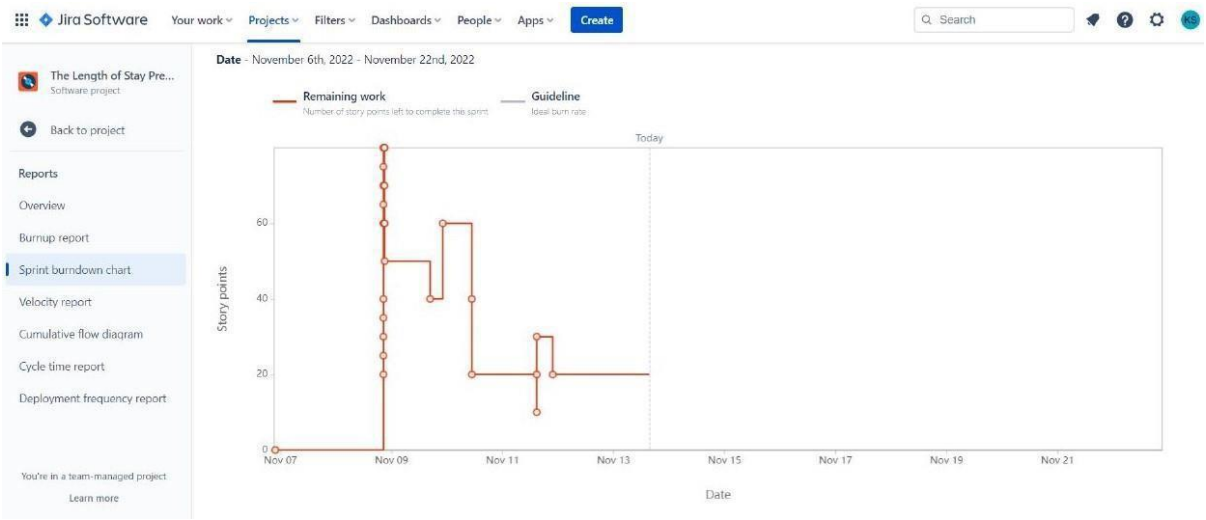
| | | |
|---|---|---|
| Prepare Milestone & Activity List | Preparing the schedule and report for milestones activities. | 16 November 2022 |
| Project Development – Delivery of Sprint – 1,2,3 & 4 | Dividing the project in sectors and submitting it. | 17 November 2022 |

## 6.3 Reports from JIRA

### Burndown Chart:

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



# 7. CODING & SOLUTIONING

**Feature 1:**

**Sprint4/utils.py:**

```
import ibm_db
from main import conn
import datetime


def donor_req_count(donor_email):
    sql = 'select count(DONOR_EMAIL) from DONATE_REQUESTS where
DONOR_EMAIL=?'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, donor_email)
    ibm_db.execute(stmt)
    return ibm_db.fetch_assoc(stmt)


def donate_req(donor_email, org_email, org_name, b_group, donor_name, donor_contact):
    date_format = "%Y-%m-%d %H:%M:%S"

    sql = 'insert into DONATE_REQUESTS values (?, ?, ?, ?, ?, ?, ?, ?)'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, donor_email)
    ibm_db.bind_param(stmt, 2, org_email)
    ibm_db.bind_param(stmt, 3, 'PENDING')
    ibm_db.bind_param(stmt, 4, datetime.datetime.strftime(datetime.datetime.now(),
date_format))
    ibm_db.bind_param(stmt, 5, org_name)
    ibm_db.bind_param(stmt, 6, b_group)
    ibm_db.bind_param(stmt, 7, donor_name)
    ibm_db.bind_param(stmt, 8, donor_contact)
    ibm_db.execute(stmt)


def donors_info(email):
    sql = 'select * from PERSONALDETAILS where EMAIL=?'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, email)
    ibm_db.execute(stmt)
    fetch = ibm_db.fetch_assoc(stmt)
    d = {'B_group': fetch['BLOODGROUP'], 'Name': fetch['FIRSTNAME']+'
'+fetch['LASTNAME'], 'Contact': fetch['PHONENUMBER']}
    return d
```

**Sprint4/routes.py:**

```
from sprint4.utils import donor_req_count, donate_req, donors_info


    @sprint4.route('/donate-plasma', methods=['POST', 'GET'])
def donate():
   if 'user' in session:
      if request.method == 'POST':
         response = dict()
         select = request.json

            ……
            ……

         elif 'email' in select:
            counter = donor_req_count(session['donor-email'])

            date_format = "%Y-%m-%d %H:%M:%S"
            if counter['1'] < 5:
               temp = donors_info(session['donor-email'])
               if counter['1'] == 0:
                  donate_req(session['donor-email'], select['email'], select['name'], temp['B_group'],
temp['Name'], temp['Contact'])
                  response['donate_req_status'] = 'Success'
               else:
                  sql = 'select ORG_EMAIL from DONATE_REQUESTS where
DONOR_EMAIL=? and ORG_EMAIL=?'
                  stmt = ibm_db.prepare(conn, sql)
                  ibm_db.bind_param(stmt, 1, session['donor-email'])
                  ibm_db.bind_param(stmt, 2, select['email'])
                  ibm_db.execute(stmt)
                  fetch = ibm_db.fetch_assoc(stmt)

                  if fetch:
                     response['donate_req_status'] = 'Already'
                  else:
                     donate_req(session['donor-email'], select['email'], select['name'],
temp['B_group'], temp['Name'], temp['Contact'])
                     response['donate_req_status'] = 'Success'
            elif counter['1'] >= 5:
               sql = 'select * from DONATE_REQUESTS where DONOR_EMAIL=?'
               stmt = ibm_db.prepare(conn, sql)
               ibm_db.bind_param(stmt, 1, session['donor-email'])
```

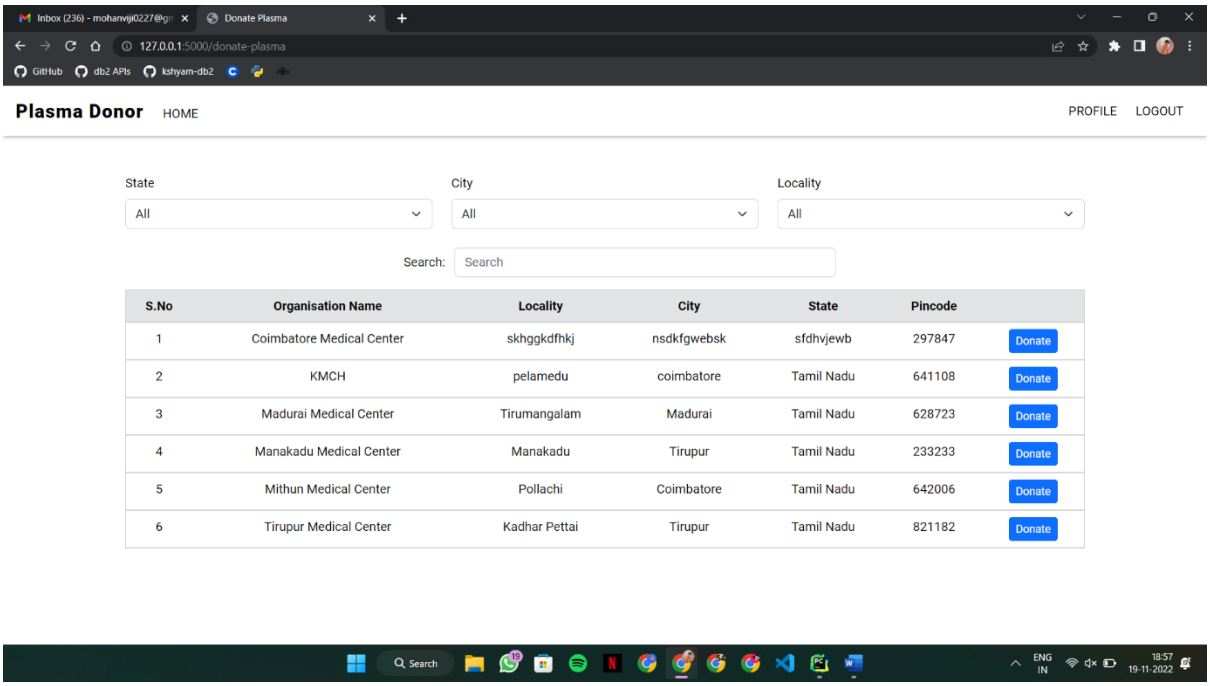
```
                ibm_db.execute(stmt)
                fetch = ibm_db.fetch_assoc(stmt)

                while fetch:
                    no_of_days = (datetime.datetime.now() -
datetime.datetime.strptime(fetch['REQUEST_MADE_TIME'], date_format)).days
                    if fetch['ORG_EMAIL'] == 'email' and no_of_days >= 2:
                        sql2 = 'delete from DONATE_REQUESTS where DONOR_EMAIL=? and
ORG_EMAIL=?'
                        stmt2 = ibm_db.prepare(conn, sql2)
                        ibm_db.bind_param(stmt2, 1, session['donor-email'])
                        ibm_db.bind_param(stmt2, 2, fetch['ORG_EMAIL'])
                        ibm_db.execcute(stmt2)
                    fetch = ibm_db.fetch_assoc(stmt)

                if donor_req_count(session['donor-email'])['1'] < 5:
                    temp = donors_info(session['donor-email'])
                    donate_req(session['donor-email'], select['email'], select['name'], temp['B_group'],
temp['Name'], temp['Contact'])
                    response['donate_req_status'] = 'Success'
                else:
                    response['donate_req_status'] = 'After'
            return response
        else:
            sql = 'select * from ORGANISATION_DETAILS where APPROVED=\'YES\' order by
NAME'
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.execute(stmt)
            fetch = ibm_db.fetch_assoc(stmt)

            res, cities, states, locality = [], [], [], []
            while fetch:
                if fetch['CITY'] not in cities:
                    cities.append(fetch['CITY'])
                if fetch['STATE'] not in states:
                    states.append(fetch['STATE'])
                if fetch['LOCALITY'] not in locality:
                    locality.append(fetch['LOCALITY'])
                res.append(fetch)
                fetch = ibm_db.fetch_assoc(stmt)

            states.sort()
            cities.sort()
            locality.sort()
            return render_template('donate.html', res=res, cities=cities, states=states,
locality=locality)
```

```
    else:
        return redirect(url_for('sprint3.login_as', redirect_to='donor'))
```

The donor can send a request to an organisation or a verified plasma transplant centre by clicking the 'DONATE' button. The donor can send maximum of five requests to different plasma transplant centres, if he/she has to make a new request after the maximum number of requests have reached, they can make a request after 48hours or can request after cancelling the previous request. The transplant centre can either accept or decline a request sent by plasma donor and the response will be sent to the user through email.

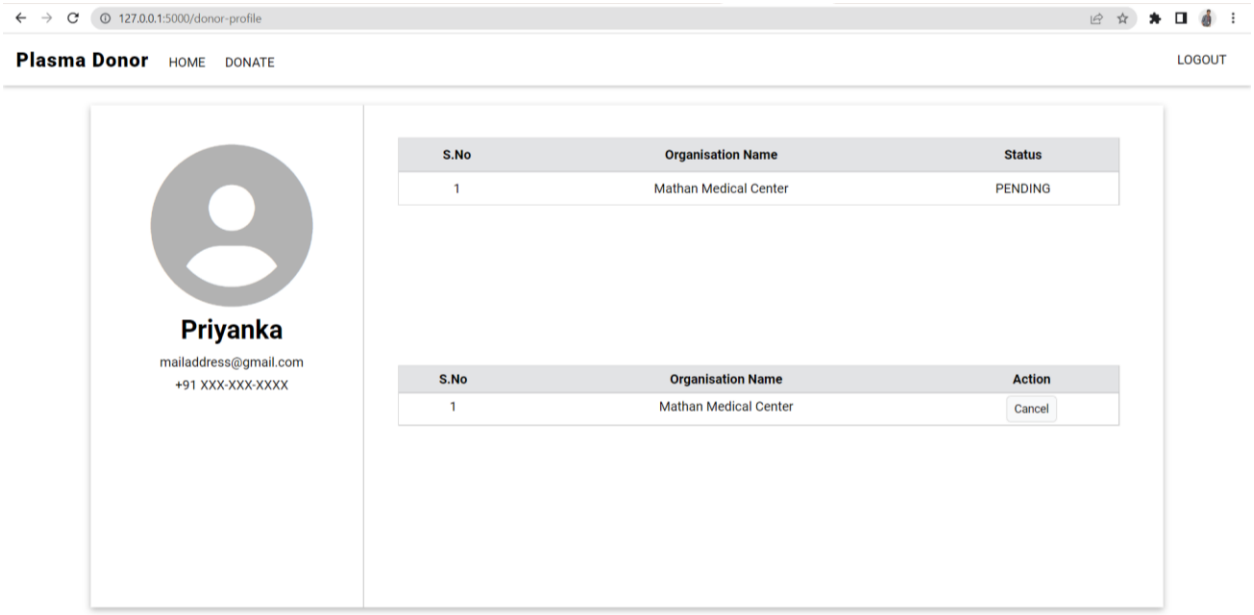
```
@sprint4.route('/donor-profile', methods=['GET', 'POST'])
def donor_profile():
    if 'user' in session:
        if request.method == 'POST':
            response = dict()
            org_email = request.json
            sql = 'delete from DONATE_REQUESTS where DONOR_EMAIL=? and
ORG_EMAIL=?'
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, session['donor-email'])
            ibm_db.bind_param(stmt, 2, org_email['org-email'])
            ibm_db.execute(stmt)

            response['CancelStatus'] = 'True'
            return response
        else:
            sql = 'select * from DONATE_REQUESTS where DONOR_EMAIL=?'
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, session['donor-email'])
            ibm_db.execute(stmt)
            fetch = ibm_db.fetch_assoc(stmt)

            res = []
            while fetch:
                res.append(fetch)
                fetch = ibm_db.fetch_assoc(stmt)
            return render_template('donor_profile.html', res=res)
    else:
        return redirect(url_for('sprint3.login_as', redirect_to='donor'))
```

The above code is the profile-page of the donor, here the donor can see the status of the request they have sent to the plasma transplant centre. The donor can also cancel their request here.




## Feature 2:

## Sprint2/utils.py:



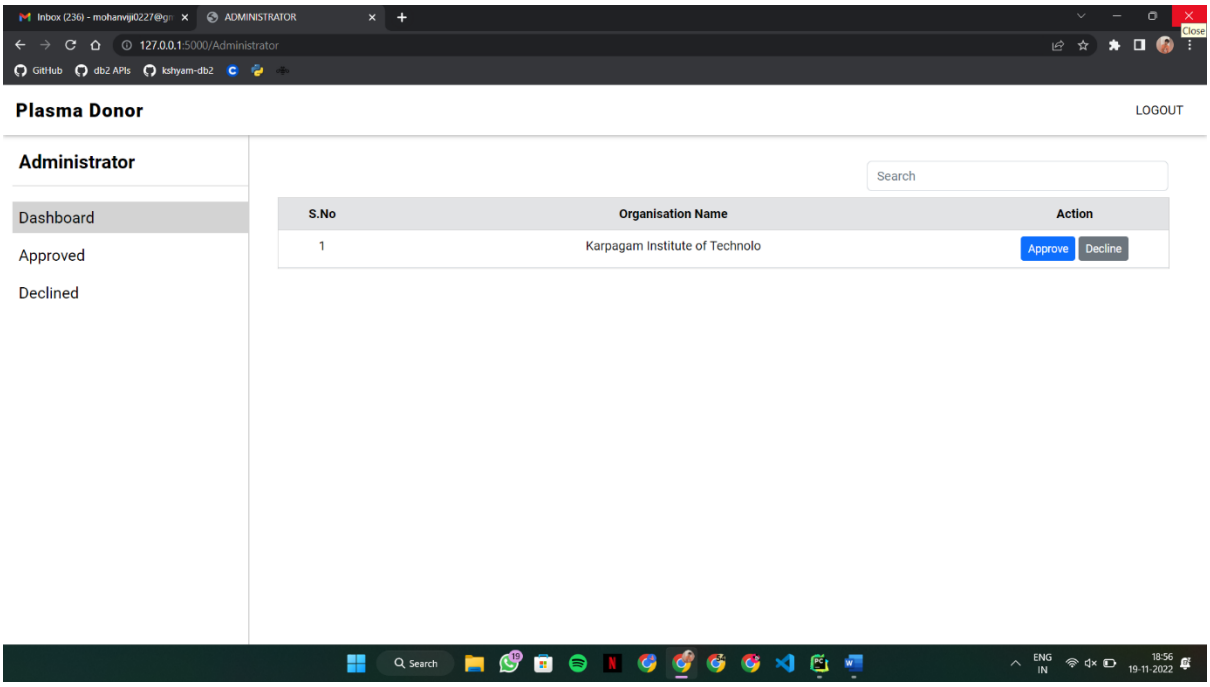
```
def generate_random_password():
    max_len = 8
    digits = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
    lower_case = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
'z']
    upper_case = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
'W', 'X', 'Y', 'Z']
    symbols = ['@', '#', '$', '%', '?', '.', '/', '*']
    combined_list = digits + upper_case + lower_case + symbols

    rand_digit = random.choice(digits)
    rand_upper = random.choice(upper_case)
    rand_lower = random.choice(lower_case)
    rand_symbol = random.choice(symbols)

    temp_pwd = rand_digit + rand_upper + rand_lower + rand_symbol
    temp_pass_list = []
    for x in range(max_len - 4):
        temp_pwd = temp_pwd + random.choice(combined_list)
        temp_pass_list = array.array('u', temp_pwd)
        random.shuffle(temp_pass_list)

    password = ""
    for x in temp_pass_list:
        password = password + x

    return password
```

**Sprint2/routes.py:**

```
@sprint2.route('/Administrator', methods=['GET', 'POST'])
def administrator():
    if 'ADMINISTRATOR' in session:
        if request.method == 'POST':
            data = request.json
            if 'email' in data and 'action' in data:
                response = dict()
                if data['action'] == 'approve':
                    pwd = generate_random_password()

                    sql = 'insert into USER_TABLE values (?, ?, NULL, ?)'
                    stmt = ibm_db.prepare(conn, sql)
                    ibm_db.bind_param(stmt, 1, data['email'])
                    ibm_db.bind_param(stmt, 2, (flask_bcrypt.generate_password_hash(pwd,
rounds=12)))
```

```
                ibm_db.bind_param(stmt, 3, 'Organisation')
                ibm_db.execute(stmt)

                sql2 = 'update ORGANISATION_DETAILS set APPROVED=? where EMAIL=?'
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2, 1, 'YES')
                ibm_db.bind_param(stmt2, 2, data['email'])
                ibm_db.execute(stmt2)

                msg = Message('Verification Status', sender='noreplyplasmadonor@gmail.com',
recipients=[data['email']])
                msg.body = f"'We have approved your organisation. Find your login credentials
below,
Username: { data['email'] }
Password: { pwd }'"
                mail.send(msg)
                response['action'] = 'Approved'
            else:
                sql2 = 'update ORGANISATION_DETAILS set APPROVED=? where EMAIL=?'
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2, 1, 'NO')
                ibm_db.bind_param(stmt2, 2, data['email'])
                ibm_db.execute(stmt2)
                response['action'] = 'Declined'
            return response
        else:
            sql = 'select * from ORGANISATION_DETAILS'
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.execute(stmt)
            fetch = ibm_db.fetch_assoc(stmt)

            res = []
            res1 = []
            res2 = []
            while fetch:
                if fetch['APPROVED'] is None:
                    res.append(fetch)
                elif fetch['APPROVED'] == 'YES':
                    res1.append(fetch)
                else:
                    res2.append(fetch)
                fetch = ibm_db.fetch_assoc(stmt)

            return render_template('administrator.html', res=res, approved=res1, declined=res2)
    else:
        return redirect(url_for('sprint2.administrator_login'))
```

The Administrator page is where the table of plasma transplant centre's register requests are viewed. To request a plasma from a donor a plasma transplant centre has to be registered. So, when a plasma transplant centre wants to sign up the provided information are displayed to the administrator, so that the administrator can verify the information and he/she can approve the centre, so that they have the authorization for requesting plasma. The username & password will be provided to the plasma transplant centre through the registered email after approval of the administrator. A strong password is generated through a random password generator function. The plasma transplant centre can change their password with forgot password.



# 8.TESTING

## 8.1 Test Cases

Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do. The benefits of testing include preventing bugs, reducing development costs and improving performance.

STEPS IN TESTING:

- Requirement analysis.
- Test planning.
- Test case design and development.
- Test environment setup.
- Test execution.
- Test cycle closure.

## 8.2 User Acceptance Testing:

**User Acceptance Testing (UAT)** is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. UAT is done in the final phase of testing after functional, integration and system testing is done.
The main **Purpose of UAT** is to validate end to end business flow. It does not focus on cosmetic errors, spelling mistakes or system testing. User Acceptance Testing is carried out in a separate testing environment with production-like data setup. It is kind of black box testing where two or more end-users will be involved.

UAT is performed by –

- Client
- End users

### Need of User Acceptance Testing:

**Need of User Acceptance Testing** arises once software has undergone Unit, Integration and System testing because developers might have built software based on requirements document by their own understanding and further required changes during development may not be effectively communicated to them, so for testing whether the final product is accepted by client/end-user, user acceptance testing is needed.

# 9.RESULTS

## 9.1 Performance Testing:

### Front-end Performance Metrics:

When looked at an app's performance from the end user's perspective, the factors that come to mind are front-end metrics. These elements have to be worked upon on a priority basis as they are directly related to the end-user.

It includes

- Protection against app crashes
- Fit into screen
- Resource consumption
- Response time

Back-end Performance Metrics

Front-end and back-end performance metrics go hand in hand. This means some of the back-end performance elements directly influence the user's experience and account for its UI/UX.

It includes

- Time to first byte
- HTTP request
- Connection and DNS lookups

## 10.Advantages:

- Our web app shows complete details of the donor/requestor.
- The details provided by the users will be verified.
- We provide separate unique password for each and every user.

# 11.CONCLUSION

In the world of information technology where whole world is becomes global village, where end user can get the information just sitting at home on one click. In fact government has taken a step in order to transform the system. Management information system helps to make the system paper less. Now the end user has to enrol himself and his job is done. All the money transaction is made possible because of the management information system. Researcher believes that by developing the management information system for the blood bank make the revolutionary changes in the system. It is small contribution of the researcher in order to serve the mankind.

# 12. FUTURE SCOPE

This paper proposes a Blood Bank Management System which we believe will bring remarkable change. Support of various regional languages for better reach. The size of the database may increase exponentially, so our BBMS is made such that it is scalable and can be deployed on cloud storage systems like Amazon Elastic Compute Cloud (EC2) or Google's Kubernetes Engine (GKE) after containerizing the application.

We have already entered the age of Information Technology, where all the paper work / manually managed files are about to finish, now with the help of this user-friendly software all the files stored in the computer can be very well formatted. With little more modifications it will become the good software for Blood Bank. The present [3]Blood Bank´ project may be further developed for more complex transactions and to meet the requirements of modern-day dynamic System Operation New options and their respective implementation may be done for this purpose.

# 13. APPENDIX

## SOURCE CODE:

```
from flask import Blueprint, render_template, request, jsonify, session,
redirect, url_for
from sprint3.utils import *
from sprint4.utils import donor_req_count, donate_req, donors_info
import datetime

sprint4 = Blueprint('sprint4', __name__, template_folder='templates',
static_folder='static', static_url_path='/sprint4/static')


@sprint4.route('/donate-plasma', methods=['POST', 'GET'])
def donate():
    if 'user' in session:
        if request.method == 'POST':
            response = dict()
            select = request.json
            if 'state' in select and 'city' in select and 'locality' in
select:
                if select['state'] != 'all' and select['city'] == 'all' and
select['locality'] == 'all':
                    sql = 'select * from ORGANISATION_DETAILS where
APPROVED=\'YES\' and STATE=? order by NAME'
                    response['filter'] = filter_by_one_param(sql,
select['state'])
                    temp = filter_by_one(select['state'], None, None)
                    response['filterCity'] = temp['res1']
                    response['filterLocality'] = temp['res2']
                    response['filter_city_select'] = 'YES'
                    response['filter_locality_select'] = 'YES'
                    return jsonify(response)
```

```
                elif select['state'] != 'all' and select['city'] != 'all' and
select['locality'] == 'all':
                    sql = 'select * from ORGANISATION_DETAILS where
APPROVED=\'YES\' and STATE=? and CITY=? order by NAME'
                    response['filter'] = filter_by_two_params(sql,
select['state'], select['city'])
                    response['filter_locality_select'] = 'YES'
                    response['filterLocality'] =
filter_by_two(select['state'], select['city'], None)
                    return jsonify(response)

                elif select['state'] != 'all' and select['city'] == 'all' and
select['locality'] != 'all':
                    sql = 'select * from ORGANISATION_DETAILS where
APPROVED=\'YES\' and STATE=? and LOCALITY=? order by NAME'
                    response['filter'] = filter_by_two_params(sql,
select['state'], select['locality'])
                    response['filter_city_select'] = 'YES'
                    response['filterLocality'] =
filter_by_two(select['state'], None, select['locality'])
                    return jsonify(response)

                elif select['state'] == 'all' and select['city'] != 'all' and
select['locality'] == 'all':
                    sql = 'select * from ORGANISATION_DETAILS where
APPROVED=\'YES\' and CITY=? order by NAME'
                    response['filter'] = filter_by_one_param(sql,
select['city'])
                    temp = filter_by_one(None, select['city'], None)
                    response['filter_locality_select'] = 'YES'
                    response['filter_state_select'] = 'YES'
                    response['filterState'] = temp['res1']
                    response['filterLocality'] = temp['res2']
                    return jsonify(response)

                elif select['state'] == 'all' and select['city'] != 'all' and
select['locality'] != 'all':
                    sql = 'select * from ORGANISATION_DETAILS where
APPROVED=\'YES\' and CITY=? and LOCALITY=? order by NAME'
                    response['filter'] = filter_by_two_params(sql,
select['city'], select['locality'])
                    response['filter_state_select'] = 'YES'
                    response['filterState'] = filter_by_two(None,
select['city'], select['locality'])
                    return jsonify(response)

                elif select['state'] == 'all' and select['city'] == 'all' and
select['locality'] != 'all':
                    sql = 'select * from ORGANISATION_DETAILS where
APPROVED=\'YES\' and LOCALITY=? order by NAME'
                    response['filter'] = filter_by_one_param(sql,
select['locality'])
                    temp = filter_by_one(None, None, select['locality'])
                    response['filter_state_select'] = 'YES'
                    response['filter_city_select'] = 'YES'
                    response['filterState'] = temp['res1']
```

```
                    response['filterCity'] = temp['res2']
                    return jsonify(response)

                elif select['state'] == 'all' and select['city'] == 'all' and
select['locality'] == 'all':
                    sql = 'select * from ORGANISATION_DETAILS where
APPROVED=\'YES\' order by NAME'
                    response['filter'] = display_donors(sql)
                    response['filter_state_select'] = 'YES'
                    response['filter_city_select'] = 'YES'
                    response['filter_locality_select'] = 'YES'
                    temp = display_all_option(sql)
                    response['filterState'] = temp['res1']
                    response['filterCity'] = temp['res2']
                    response['filterCity'] = temp['res3']
                    return jsonify(response)

                sql = 'select * from ORGANISATION_DETAILS where
APPROVED=\'YES\' and STATE=? and CITY=? and LOCALITY=? order by NAME'
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(stmt, 1, select['state'])
                ibm_db.bind_param(stmt, 2, select['city'])
                ibm_db.bind_param(stmt, 3, select['locality'])
                ibm_db.execute(stmt)
                fetch = ibm_db.fetch_assoc(stmt)

                res = []
                while fetch:
                    res.append(fetch)
                    fetch = ibm_db.fetch_assoc(stmt)
                response['filter'] = res
                return jsonify(response)
            elif 'email' in select:
                counter = donor_req_count(session['donor-email'])

                date_format = "%Y-%m-%d %H:%M:%S"
                if counter['1'] < 5:
                    temp = donors_info(session['donor-email'])
                    if counter['1'] == 0:
                        donate_req(session['donor-email'], select['email'],
select['name'], temp['B_group'], temp['Name'], temp['Contact'])
                        response['donate_req_status'] = 'Success'
                    else:
                        sql = 'select ORG_EMAIL from DONATE_REQUESTS where
DONOR_EMAIL=? and ORG_EMAIL=?'
                        stmt = ibm_db.prepare(conn, sql)
                        ibm_db.bind_param(stmt, 1, session['donor-email'])
                        ibm_db.bind_param(stmt, 2, select['email'])
                        ibm_db.execute(stmt)
                        fetch = ibm_db.fetch_assoc(stmt)

                        if fetch:
                            response['donate_req_status'] = 'Already'
                        else:
                            donate_req(session['donor-email'],
select['email'], select['name'], temp['B_group'], temp['Name'],
temp['Contact'])
```

```
                            response['donate_req_status'] = 'Success'
                    elif counter['1'] >= 5:
                        sql = 'select * from DONATE_REQUESTS where DONOR_EMAIL=?'
                        stmt = ibm_db.prepare(conn, sql)
                        ibm_db.bind_param(stmt, 1, session['donor-email'])
                        ibm_db.execute(stmt)
                        fetch = ibm_db.fetch_assoc(stmt)

                        while fetch:
                            no_of_days = (datetime.datetime.now() -
datetime.datetime.strptime(fetch['REQUEST_MADE_TIME'], date_format)).days
                            if fetch['ORG_EMAIL'] == 'email' and no_of_days >= 2:
                                sql2 = 'delete from DONATE_REQUESTS where
DONOR_EMAIL=? and ORG_EMAIL=?'
                                stmt2 = ibm_db.prepare(conn, sql2)
                                ibm_db.bind_param(stmt2, 1, session['donor-
email'])
                                ibm_db.bind_param(stmt2, 2, fetch['ORG_EMAIL'])
                                ibm_db.execcute(stmt2)
                            fetch = ibm_db.fetch_assoc(stmt)

                        if donor_req_count(session['donor-email'])['1'] < 5:
                            temp = donors_info(session['donor-email'])
                            donate_req(session['donor-email'], select['email'],
select['name'], temp['B_group'], temp['Name'], temp['Contact'])
                            response['donate_req_status'] = 'Success'
                        else:
                            response['donate_req_status'] = 'After'
                    return response
            else:
                sql = 'select * from ORGANISATION_DETAILS where APPROVED=\'YES\'
order by NAME'
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.execute(stmt)
                fetch = ibm_db.fetch_assoc(stmt)

                res, cities, states, locality = [], [], [], []
                while fetch:
                    if fetch['CITY'] not in cities:
                        cities.append(fetch['CITY'])
                    if fetch['STATE'] not in states:
                        states.append(fetch['STATE'])
                    if fetch['LOCALITY'] not in locality:
                        locality.append(fetch['LOCALITY'])
                    res.append(fetch)
                    fetch = ibm_db.fetch_assoc(stmt)

                states.sort()
                cities.sort()
                locality.sort()
                return render_template('donate.html', res=res, cities=cities,
states=states, locality=locality)
        else:
            return redirect(url_for('sprint3.login_as', redirect_to='donor'))


@sprint4.route('/donor-profile', methods=['GET', 'POST'])
```

```
def donor_profile():
    if 'user' in session:
        if request.method == 'POST':
            response = dict()
            org_email = request.json
            sql = 'delete from DONATE_REQUESTS where DONOR_EMAIL=? and
ORG_EMAIL=?'
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, session['donor-email'])
            ibm_db.bind_param(stmt, 2, org_email['org-email'])
            ibm_db.execute(stmt)

            response['CancelStatus'] = 'True'
            return response
        else:
            sql = 'select * from DONATE_REQUESTS where DONOR_EMAIL=?'
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, session['donor-email'])
            ibm_db.execute(stmt)
            fetch = ibm_db.fetch_assoc(stmt)

            res = []
            while fetch:
                res.append(fetch)
                fetch = ibm_db.fetch_assoc(stmt)
            return render_template('donor_profile.html', res=res)
    else:
        return redirect(url_for('sprint3.login_as', redirect_to='donor'))
```

**MAIN FUNCTION:**

```
from main import app

if __name__ == '__main__':

    app.run(debug=True, host='0.0.0.0', port=5000)
```

**GitHub link:**

https://github.com/mathanrajkumar12/IBM-Project-39988-1660574714

**Demo Video Link:**

https://drive.google.com/file/d/11AXqAUz8f2bO2ETZTAGr2jhp60BMbQg0/view?usp=share_link