PLASMA DONAR APPLICATION

PROJECT REPORT

Team ID: PNT2022TMID31865

Team Members

Mithun Kumar N (Team Leader)

Mohan Prasath M

Mohanprasanth S

Banu Prakash M K

Tarunika Reddy D

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 problem

1. Application Name: Give Blood by NHS Blood & Transplant

NHS Blood & Transplant is a blood donation service in England, who owns the Give Blood Application. This application helps people by providing Plasma transplantation, in which the donor can register in online and they can donate plasma in a Transplant Centre by booking an appointment. The user can even view, change or cancel the appointment. And we could update our personal details. The user could view their recent donation history within the last 05 years.

Disadvantage:

• It requires Internet Connectivity. There are people who don't use smartphones. In case he is a plasma donor, we cannot able to send plasma requirement request to the donor.

• Also, the Give Blood application provides service only in England.

2. Application Name: Delhi Fights Corona

The Delhi Fights Corona is owned by the government of Delhi, India. It's the country's first plasma bank. The person has to register for donating plasma. And have to travel to ILBS Hospital in New Delhi for donating plasma. If the person hasn't been tested covid-negative after being tested positive, they can be tested there.

Disadvantages:

• The plasma donor or the person in need of plasma has to travel a way, to New Delhi for plasma transplantation (By the way the government arranges for travelling to ILBS Hospital or reimburse your travel cost).

2.2 References

1. URL: https://www.blood.co.uk/

2. URL: https://delhifightscorona.in/donateplasma/

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

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

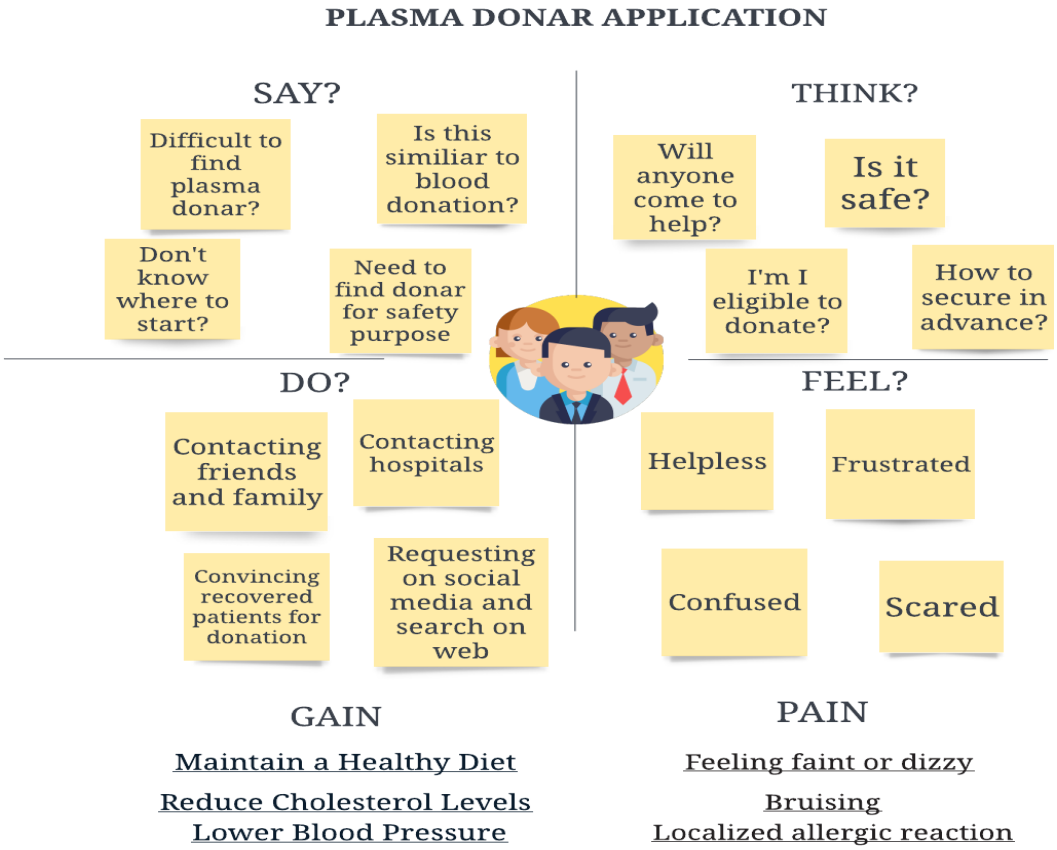
3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

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.

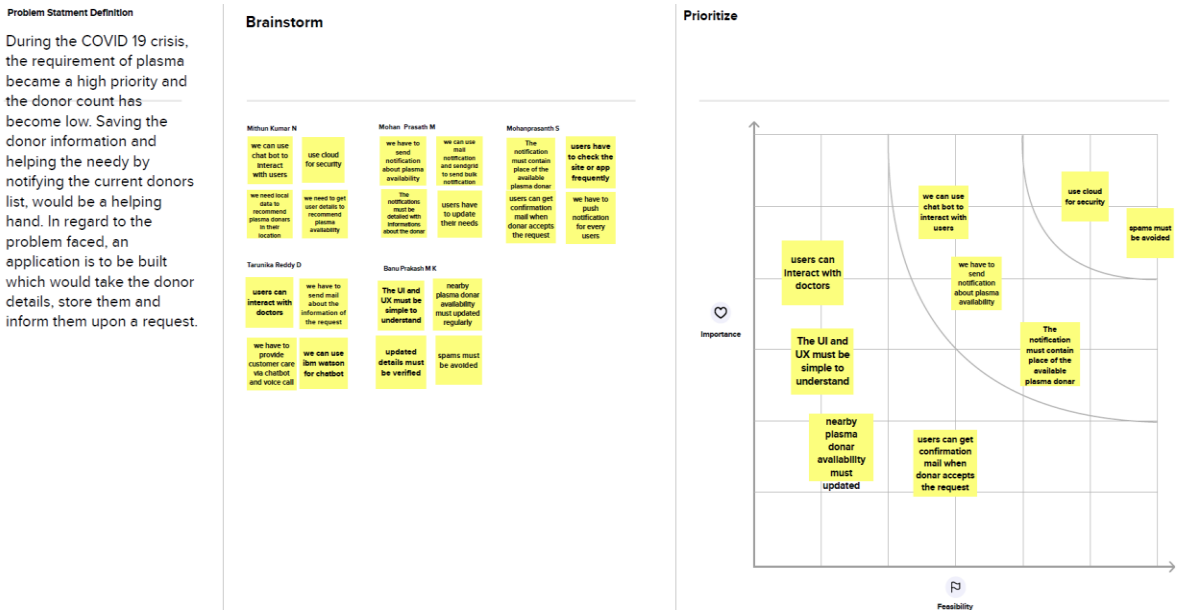
Here we defined those four sectors by including the thoughts, it explained what the user's first thought while hearing about our project. It also includes what are the clarification to be done before initiating the usage of the app. This map also consists the initial measures and actions done by the 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 occurring 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 necessary ideas and concluded with the session.

The main aim of the session is to give a 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.



3.3 Proposed Solution:

S. No.	Parameter	Description
1	Problem Statement (Problem to be solved)	Blood banks are required to maintain account of blood bags in the inventory. This increases with each blood donation recorded in our system and decreases as they are checked out upon hospital requests. Our system will need to keep the information up to date to ensure correctness of the inventory.

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	Donors who wish to donate plasma can donate by uploading their COVID19 recovery certificate on the donor's page. If the donor is new, they must register before log in. If the donor is an existing user they need to login. Username and e-mail provided at the time of registration.
4	Social Impact / Customer Satisfaction	The application is user friendly and anyone with basic knowledge can access it. The application seamlessly connects the donor and the person who need it.
5	Business Model (Revenue Model)	People will get used to this application, by collaborating with government and organizing blood donation camps.
6	Scalability of the Solution	Since the app is going to store its data in cloud, it will continue to be efficient when large number of people uses it. Also, when the number of requests for plasma increases, the ca l notification work fine without any disruption.

3.4 Problem Solution Fit:

Problem-Solution fit canvas 2.0

Purpose / Vision

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Who is your customer?
i.e. working parents of 0-5 y.o. kids

Who is in need plasma donor .He is the customer.
The people suffered from Covid-19 suffer may act as a customer.

6. CUSTOMER CONSTRAINTS CC

What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.

Network connection may not.
Difficult to charge a device everywhere.
Device may not available everywhere.

5. AVAILABLE SOLUTIONS AS

Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking

Bringing recovered patient back to hospital.
And further next donation is impossible.
Plasma demand and supply gap has grown even bigger.

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

Customer couldn't know how to find the nearest donor.
Sometimes donor couldn't know how to donate plasma in plasma center.

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists?
What is the back story behind the need to do this job?
i.e. customers have to do it because of the change in regulations.

Seek help from more experienced plasma specialist.
Plasma is available.
More number of information are available.

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done?
i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

Donor identification.
Swab test(RT PCR) positive report.
14-28 days after discharge.
Haemoglobin%>12.5 gm.

Focus on J&P, tap into BE, understand RC

Identify strong TR & EM

3. TRIGGERS TR

What triggers customers to act? i.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news.

Make advertisment on social media about application.

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards?
i.e. lost, insecure > confident, in control - use it in your communication strategy & design.

Before:Feel very depressed to find the donor in critical situation.
After:Feel happy no need to suffer to find the donor in critical situation.

10. YOUR SOLUTION SL

If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality.
If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.

Online web application can be created for identifying plasma donors
All information should be available on application.

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

All feature are accessible during online.

8.2 OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.

All feature cannot be accessible in offline.

Extract online & offline CH of BE

AMALTAMA

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 Website
FR-2	User Confirmation	Confirmation via Email
FR-3	User Login	Login using Registered email Id
FR-4	Sent Request	If plasma is required, the receiver will contact the donor
FR-5	Contact Donor	Contact the donor directly if a phone number is given
FR-6	View donation camps	View the list of donation camps happening nearby

4.2 Non-Functional requirement

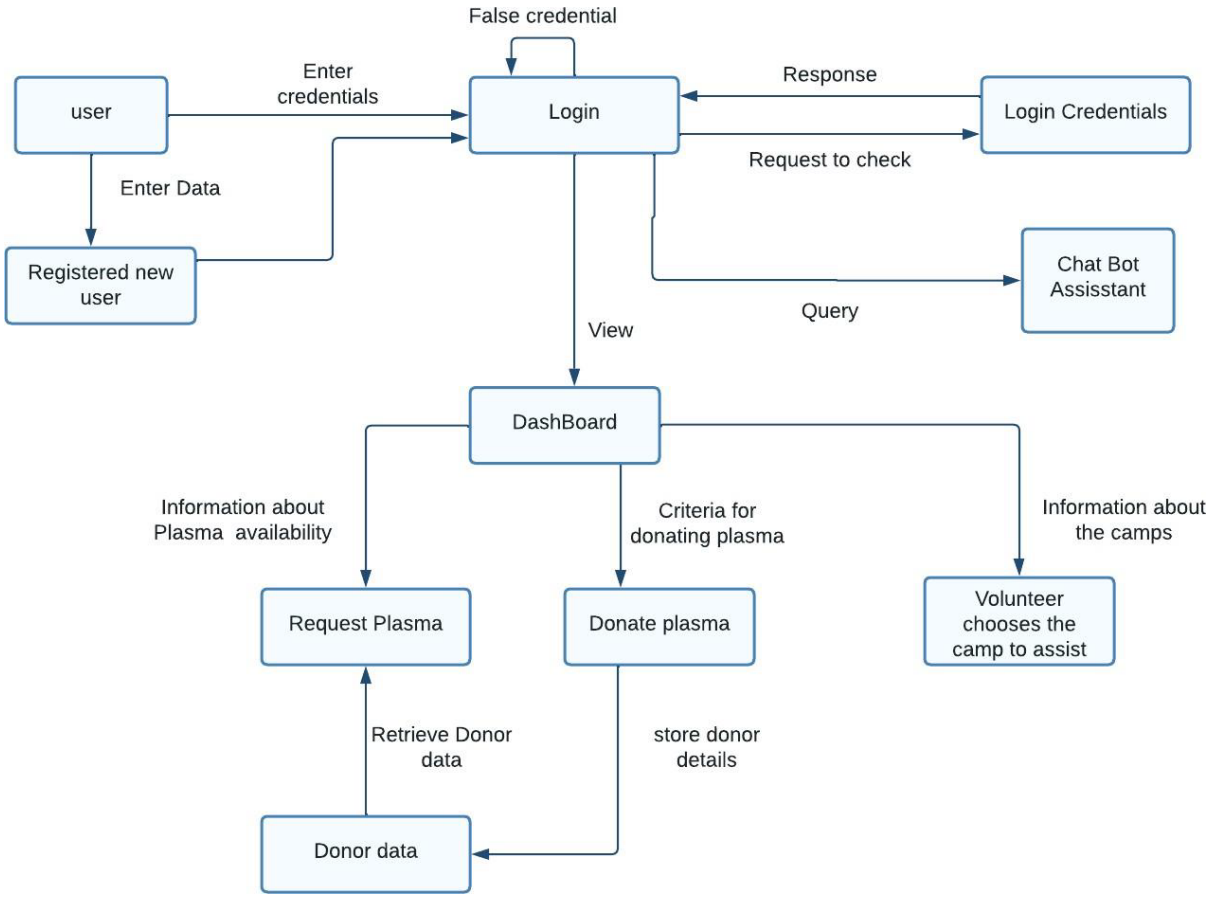
FR No.	Non-Functional Requirement	Description
NFR-1	Usability	The user interface of the plasma donor system must be well-designed and welcoming.
NFR-2	Security	Data storage is required by security systems, just like it is by many other applications. Databases are able to keep all the donor information that is viewed by applications. It must be secured with email Id and password.
NFR-3	Reliability	The system has the ability to work all the times without failures apart from network failure. A donor can have the faith on the system. The authorities will keep the privacy of all donors in a proper manner
NFR-4	Performance	The Plasma donor System must perform well in different scenarios. The system is interactive and delays involved are less.
NFR-5	Availability	The system, including the online components, should be available 24/7.
NFR-6	Scalability	The system offers the proper resources for issue solutions and is designed to protect sensitive information during all phases of operation

5. PROJECT DESIGN

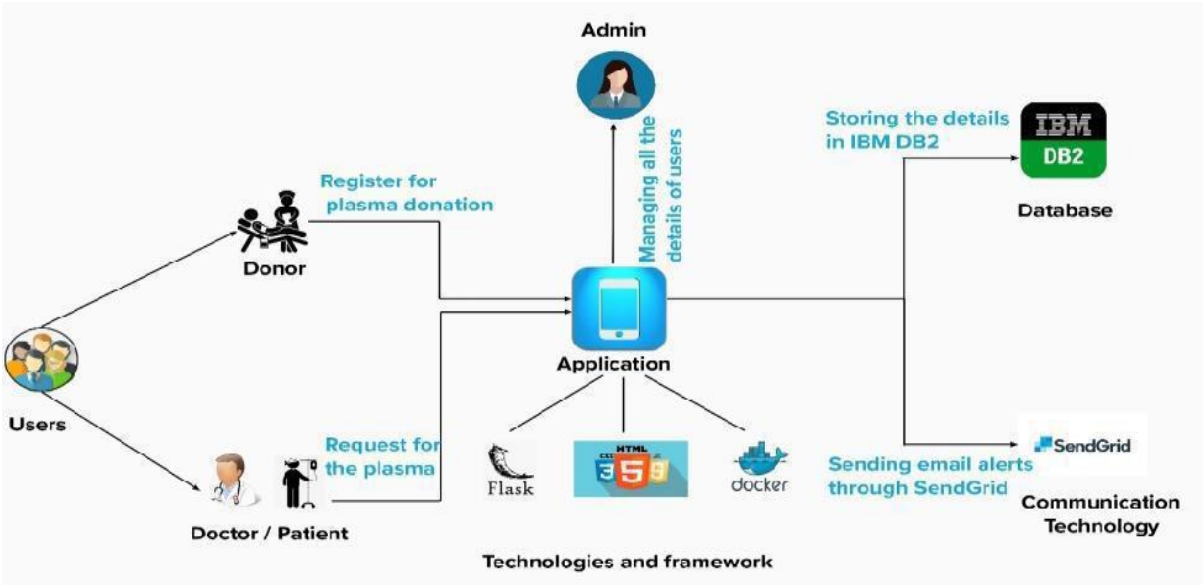
5.1 Data Flow Diagrams

A data flow diagram 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

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 my password.	20	High	1.Mithun Kumar N 2.Mohan Prasath M 3.Mohanprasanth S 4.Banu Prakash M K 5.Tarunika Reddy D
Sprint-2	Login	USN-2	As a user, I can log into the application by entering email & password.	20	High	1.Mithun Kumar N 2.Mohan Prasath M 3.Mohanprasanth S 4.Banu Prakash M K 5.Tarunika Reddy D
Sprint-3	Donor Information	USN-3	Donors can update their personal information e.g.: blood group.	20	High	1.Mithun Kumar N 2.Mohan Prasath M 3.Mohanprasanth S 4.Banu Prakash M K 5.Tarunika Reddy D

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-4	Finding the donor	USN-4	The patient can find the donor with their similar blood group	20	High	1.Mithun Kumar N 2.Mohan Prasath M 3.Mohanprasanth S 4.Banu Prakash M K 5.Tarunika Reddy D

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

6.2 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)
Sprint-1	20	3 Days	24 Oct 2022	26 Oct 2022	20
Sprint-2	20	3 Days	27 Oct 2022	30 Oct 2022	20
Sprint-3	20	6 Days	31 Oct 2022	5 Nov 2022	20
Sprint-4	20	6 Days	06 Nov 2022	12 Nov 2022	20

Velocity:

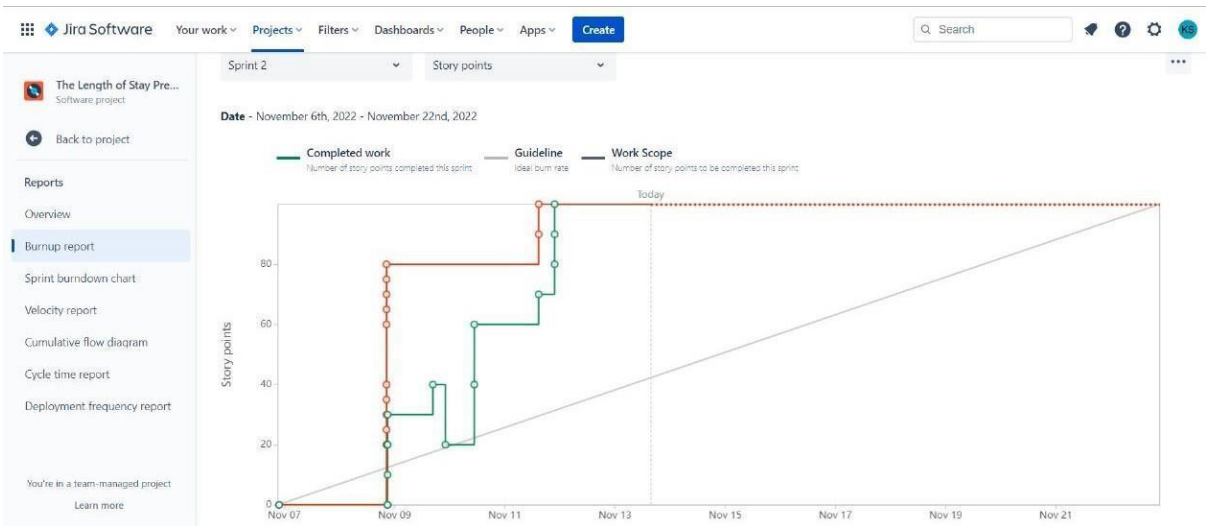
Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

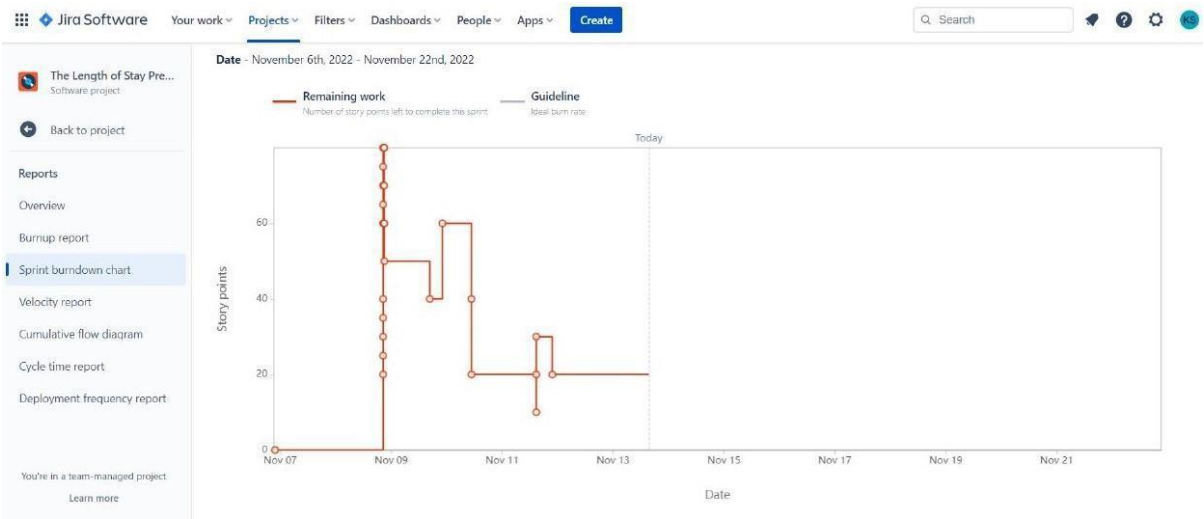
$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

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.




Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
The Length of Stay Pre...
Software project
Back to project
Reports
Overview
Burnup report
Sprint burndown chart
Velocity report
Cumulative flow diagram
Cycle time report
Deployment frequency report
You're in a team-managed project
Learn more
Date - November 6th, 2022 - November 22nd, 2022
Remaining work
Number of story points left to complete this sprint
Guideline
Ideal burn rate
Today
Story points
0
20
40
60
Nov 07
Nov 09
Nov 11
Nov 13
Nov 15
Nov 17
Nov 19
Nov 21
Date

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
```

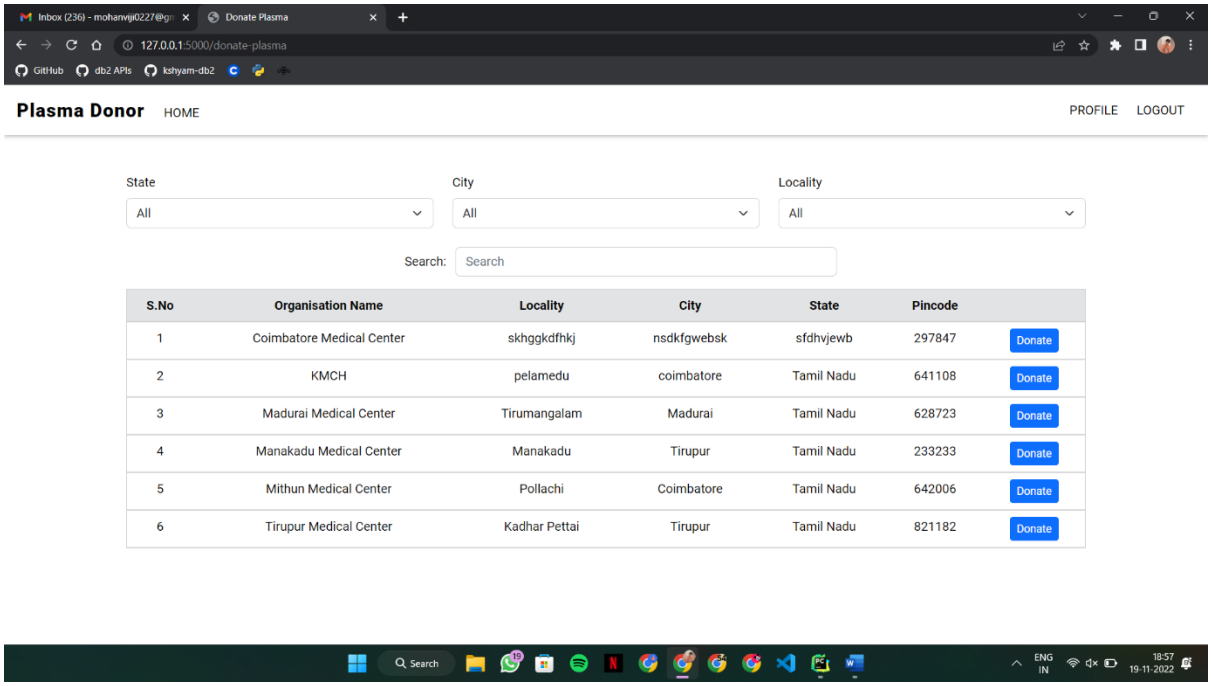
```
                if fetch['ORG_EMAIL'] == 'email' and no_of_days >= 2:
                    sql2 = 'delete from DONATE_REQUESTS where DONOR_EMAIL=? and
ORG_EMAIL=?'
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2, 1, session['donor-email'])
                    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:
```

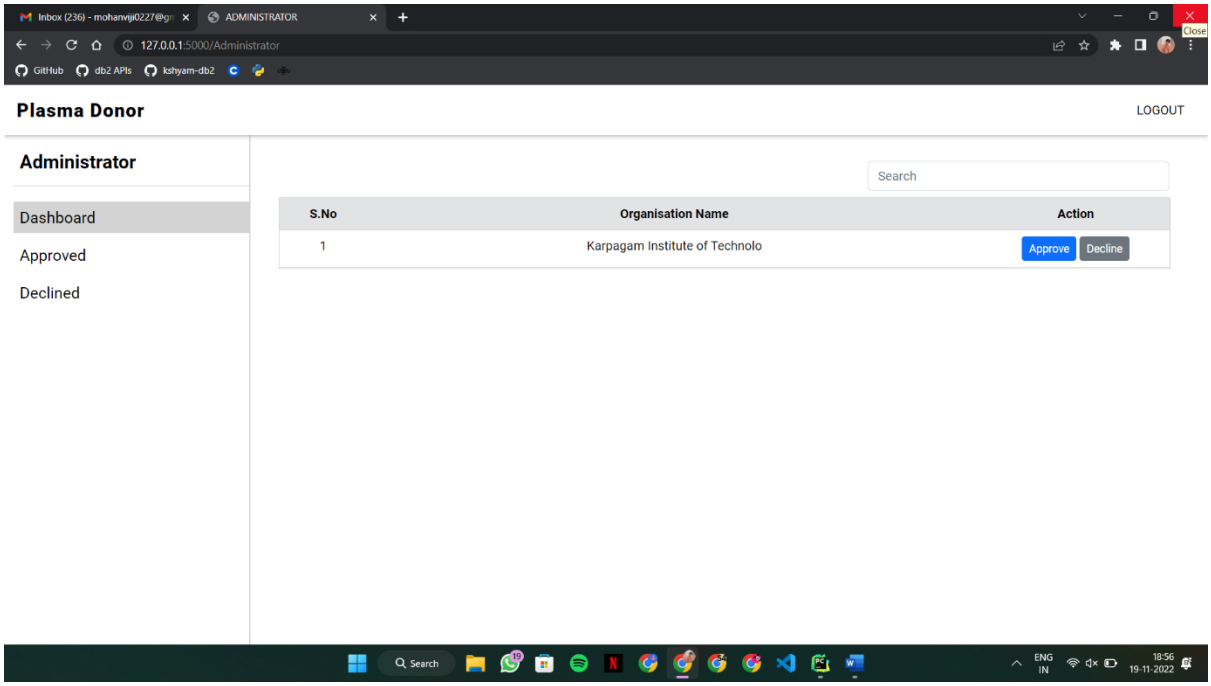
```
        password = password + x

    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.


Inbox (236) - mohanvijji0227@gm
ADMINISTRATOR
127.0.0.1:5000/Administrator
GitHub
db2 APIs
kshyam-db2
Plasma Donor
LOGOUT
Administrator
Dashboard
Approved
Declined
Search
S.No
Organisation Name
Action
1
Karpagam Institute of Technolo
Approve
Decline
Search
ENG
IN
18:56
19-11-2022

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', ss

@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/IBM-EPBL/IBM-Project-39980-1660574337

Demo Video Link:

https://drive.google.com/file/d/1CC6UjY20Bm9vbampjOOBK8Ei_CFOXE9H/view?usp=share_link