IBM NALAIYATHIRAN PROJECT REPORT

**PLASMA DONOR APPLICATION**

|  |  |
| --- | --- |
| **Team Id** | **PNT2022TMID02342** |
| **Project Name** | **Plasma Donor Application** |
| **Team Members** | **- Mithul Sudharsan R (2116190701273)**  **- Vishva Prasad P (2116190701254)**  **- Vignesh P (2116190701241)**  **- Vishal V (2116190701251)** |

# Table Of Contents

|  |  |  |
| --- | --- | --- |
| **SI**  **No** | **Title** | **Page No** |
| 1 | **INTRODUCTION**   * 1. Project Overview   2. Purpose | 4  4 |
| 2 | **LITERATURE SURVEY**   * 1. Existing problem   2. References   3. Problem Statement Definition | 5  5  5 |
| 3 | **IDEATION & PROPOSED SOLUTION**   * 1. Empathy Map Canvas   2. Ideation & Brainstorming   3. Proposed Solution   4. Problem Solution fit | 6  7  8  10 |
| 4 | **REQUIREMENT ANALYSIS**   * 1. Functional requirement   2. Non-Functional requirements | 11  11 |
| 5 | **PROJECT DESIGN**   * 1. Data Flow Diagrams   2. Solution & Technical Architecture   3. User Stories | 12  13  14 |

|  |  |  |
| --- | --- | --- |
| 6 | **PROJECT PLANNING & SCHEDULING**   * 1. Sprint Planning & Estimation   2. Sprint Delivery Schedule   3. Reports from JIRA | 15  16  16 |
| 7 | **CODING & SOLUTIONING**   * 1. Feature 1   2. Feature 2   3. Database Schema (if Applicable) | 17  18  18 |
| 8 | **TESTING**   * 1. Test Cases   2. User Acceptance Testing | 20  21 |
| 9 | **RESULTS**  9.1 Performance Metrics | 22 |
| 10 | **ADVANTAGES & DISADVANTAGES** | 28 |
| 11 | **CONCLUSION** | 29 |
| 12 | **FUTURE SCOPE** | 29 |
| 13 | **APPENDIX**   * 1. Source Code   2. Github & Demo link | 30  42 |

* 1. **Project Overview**

# INTRODUCTION

A vital component of the treatment for many significant health issues is plasma. As a result, people are asked to donate blood plasma at blood drives. Our project's major objective is to make it simpler for COVID-19 patients to find a plasma donor and donate plasma once they have recovered. The system is designed for two different user types: those looking to donate plasma and those in need of plasma. Additionally, the user can check the number of active cases, neighbouring immunisation facilities, and hospitals' addresses.

The primary goal of creating the website is to make it simpler for COVID-19 patients to find a plasma donor quickly and conveniently. However, the demand for plasma-derived products has been rising steadily for a while, and blood collection organisations must change to fulfil this need. This essay seeks to cover the primary incentives and disincentives for donating whole blood and to contrast them with those that we currently know about donating plasma. Current study reveals parallels between these behaviours, as well as distinctions that point to the necessity for more plasma donation research.

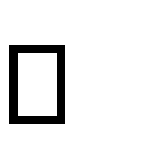
## Purpose

The need for plasma became urgent during the COVID 19 crisis, and the number of donors has declined.

It would be helpful to save the donor information and assist the less fortunate by informing the list of current donors. In order to solve the issue, a programme will be created that will collect donor information, store it, and provide information upon request.

# LITERATURE SURVEY

## EXIXTING PROBLEM

* + - Only mobile based system is available web-based system is available
    - Less Security
    - No proper coordination between different applications and users
    - Cannot upload and download the latest updates at right time Fewer users-friendly

## REFERENCE

Several experiments have been carried out over the years by different groups of researchers. Here are some of the following groups:

1. Denuis O'Neil (1999). "Blood component" Archived from the original on June 5, 2013.
2. ways to keep your plasma healthy, Original Archived November 1, 2013, Accessed November 11, 2011.
3. Ripathis S, Kumar V, Prabhakar A, Joshi S, Agarwal A (2015). "Microscale Passive Plasma Separation: A Review of Design Principles and Microdevices," J. Micromech Micro 25 (8): 083001;
4. P. C. P. C. a. V. I. M. Yan, “Building a chatbot with server less computing,” IBM watson research center, 2016.
5. S. E. a. B. J. J. Short, ““Cloud Event Programming Paradigms: Applications and Analysis,”,” 9th IEEE International Conference on Cloud Computing (CLOUD), pp. pp. 4 00-406, 2017.

## Problem Statement Definition

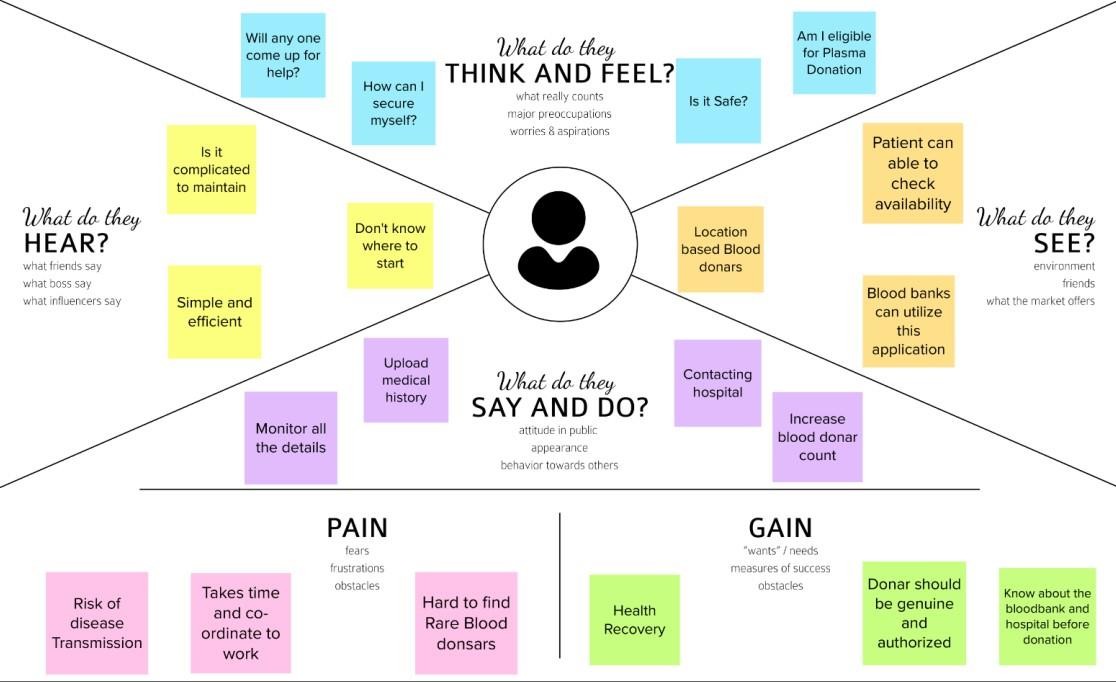
The need for plasma surged significantly during the COVID 19 crisis since there were no vaccines available to treat the infected patients.

Finding a plasma donor in such a situation was extremely difficult, and determining which donors are eligible to donate plasma as well as whether they have previously been infected and have recovered was a difficult task.

As the plasma therapy was one of the ways to treat the infected patients getting the donor details played a major role.

## IDEATION AND PROPOSED SYSTEM

* 1. **Empathy Map Canvas**



## Ideation and Brainstorming

## 

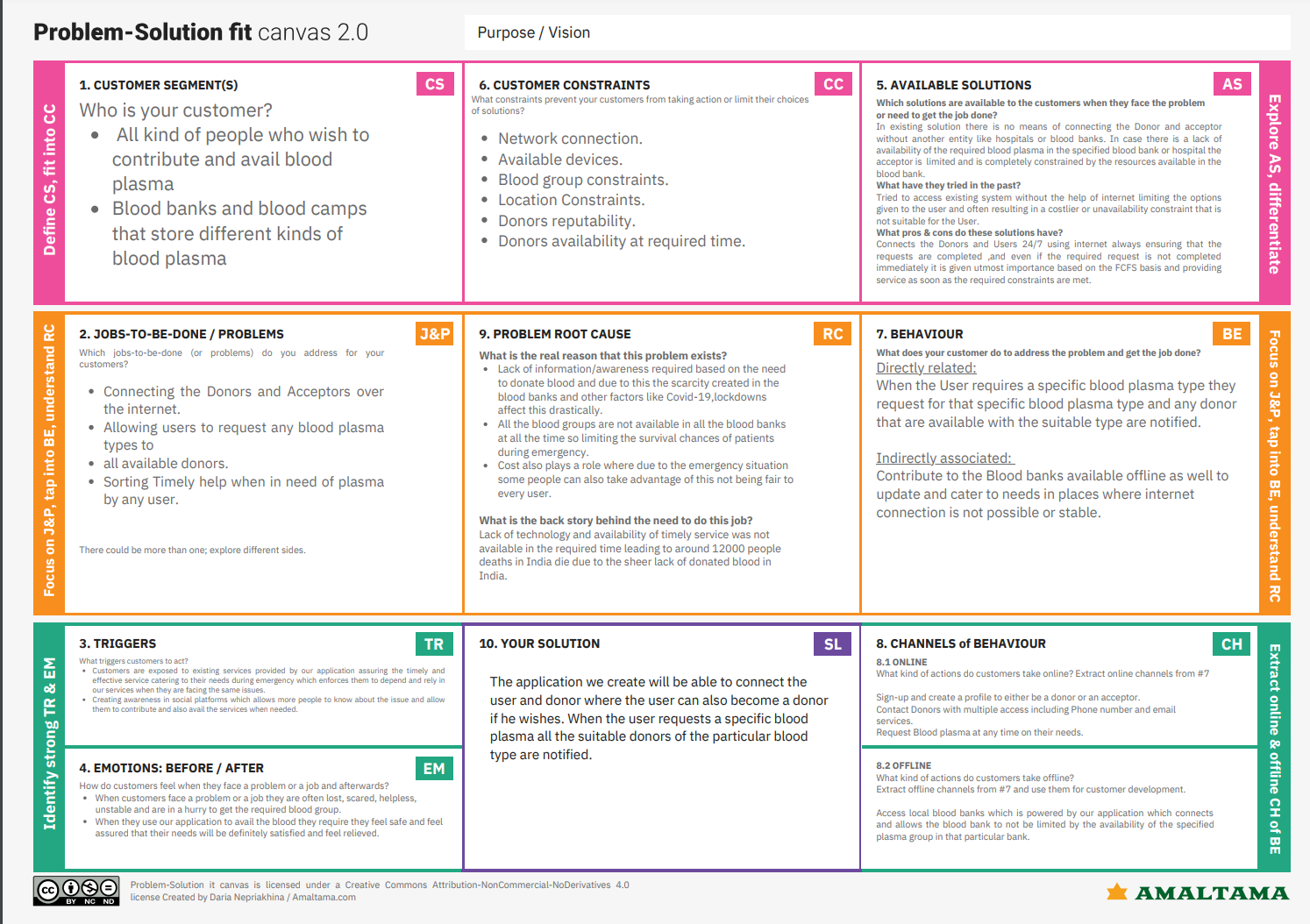
* 1. **Proposed Solution**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Parameter** | **Description** |
| 1. | Problem Statement (Problem to be  solved) | The need for plasma surged significantly during the COVID 19 crisis since there were no vaccines available to treat the infected patients.  Finding a plasma donor in such a case was extremely difficult, and determining which donors are eligible to donate plasma as well as whether they had previously been infected and have recovered was a difficult effort. |

|  |  |  |
| --- | --- | --- |
| 2. | Idea / Solution description | The application we develop will be able to link users and donors, with the user having the option to opt-in as a donor as well. 2. All eligible donors of the requested blood type are alerted when a user asks a particular blood plasma. |
| 3. | Novelty / Uniqueness | 1. A user-friendly interface that connects the donor and acceptor in a quick, seamless, and efficient manner. 2. Establishes a Plasma donation community where contributors and recipients benefit equally, fostering a sense of security and assurance when discussing their needs for urgent blood plasma requirements. |
| 4. | Social Impact / Customer Satisfaction | You can gain from using the software properly in a variety of ways, and it also makes management very simple and error-free. Donor tracking, prompt and accurate reports when needed, and centralised data storage with security are all made possible with the aid of the software. Additionally, the software will contribute to customer satisfaction. |
| 5. | Business Model (Revenue Model) | 1. The creation of a global community through global connectedness ensures that all emergency requirements are recognised and met at the appropriate moment. 2. Create a reliability factor for each user to guarantee service delivery based on user ratings. |
|  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6. | Scalability Solution | of | the | This programme allows users to find plasma donors while lounging at home rather than searching the entire world. When an emergency occurs, make a plasma request to everyone. The contributor is ready to The donation is disclosed to the donor receiver. The donor may be contacted by the recipient. As a result, donors can use an app to check their eligibility, which also makes it easier to find a compatible donor. |

* 1. **Problem Statement Fit**

****

# REQUIREMENT ANALYSIS

## 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 developers should be able to access web applications on a computer using a browser or something similar. |
| FR-2 | **Software operator Registration** | The online application should allow the software operator to register. User name, gender, blood/plasma group, location, and contact information are required from the donor software operator. |
| FR-3 | **Login/logout/update details** | The login information will be stored on the database for future use. |
| FR-4 | **Search for donor** | A list of search results can be seen. Each item on the list corresponds to a particular donor, complete with donor information. |
| FR-5 | **User plasma request** | By completing the request form on the page, users can submit a request to donate plasma. They will receive an email once the request has been submitted. |
| FR-6 | **View distribution**  **details** | The plasma bank should have access to the distribution |

## Non-functional Requirements:

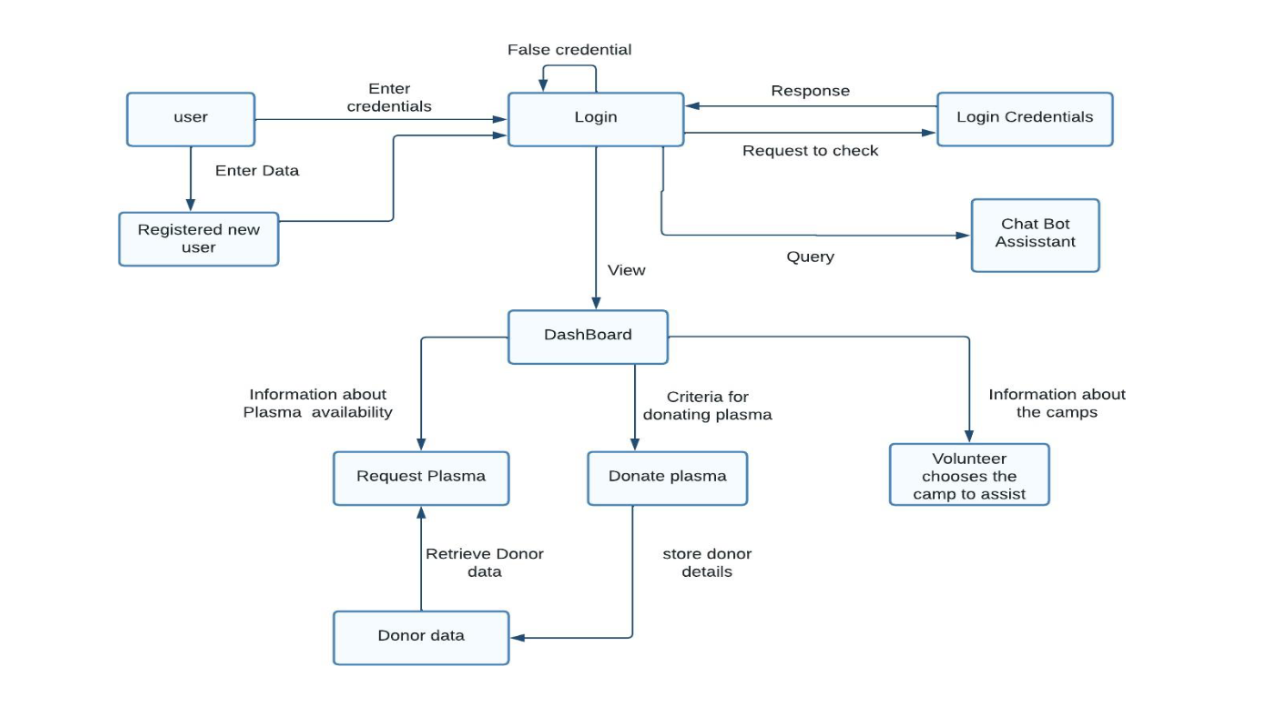
Following are the non-functional requirements of the proposed solution.

|  |  |  |
| --- | --- | --- |
| **FR**  **No.** | **Non-Functional Requirement** | **Description** |
| NFR-1 | **Usability** | The user interface of the plasma donor application needs to be attractive and functional. |
| NFR-2 | **Security** | Proper user names and passwords must be used to secure the plasma donor application. |
| 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 performance and cost of the plasma donor application should be able to change in response to modifications in the requirements for application and system processing. |

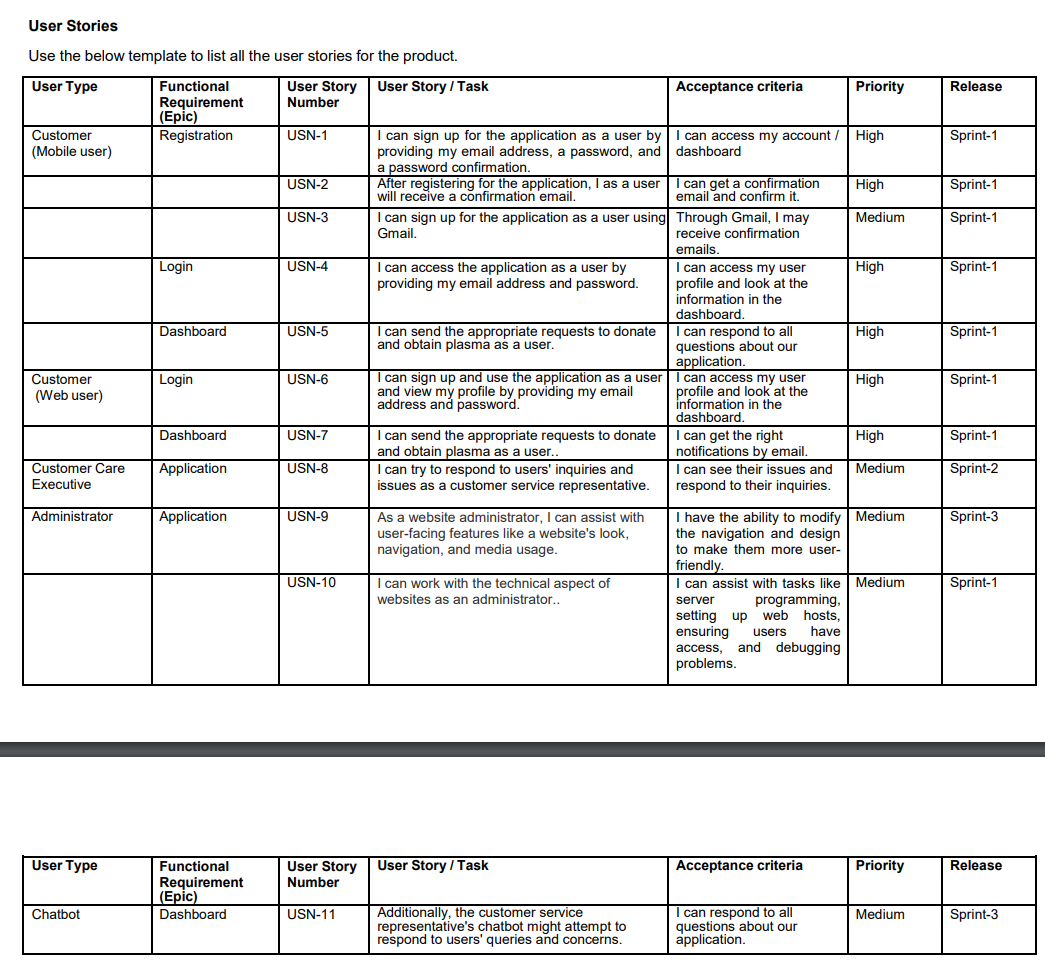
* 1. **Data Flow Diagram:**

# PROJECT DESIGN

****

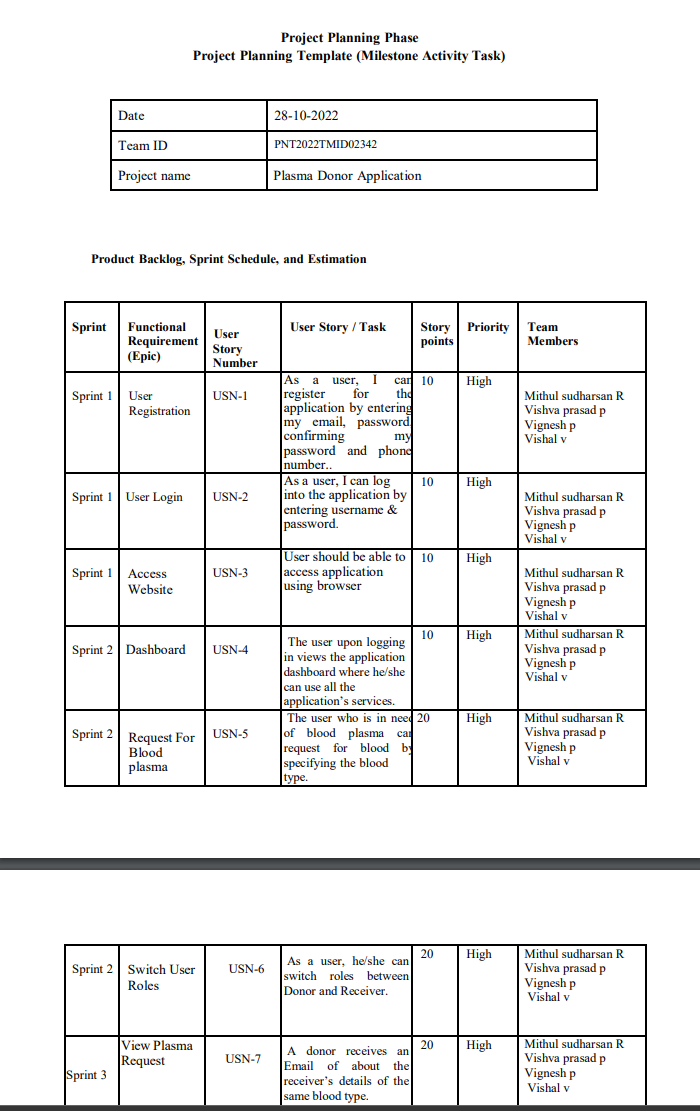
## Solution and Technical Architecture :

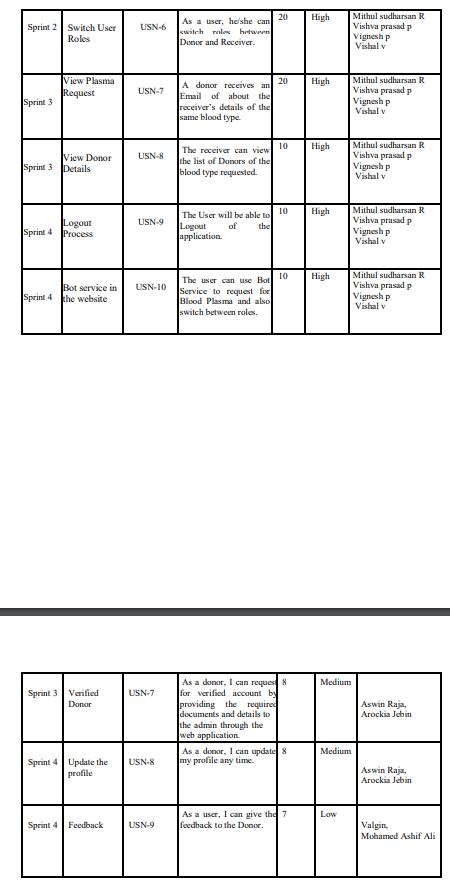
* 1. **User Stories:**

****

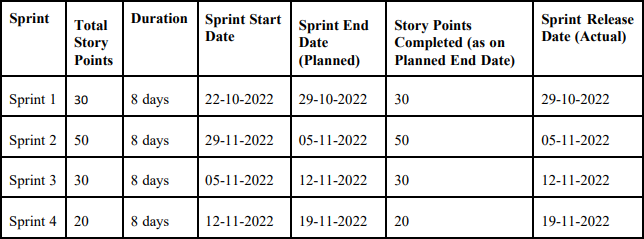
## PROJECT PLANNING AND SCHEDULING

* 1. **Sprint Planning & Estimation**

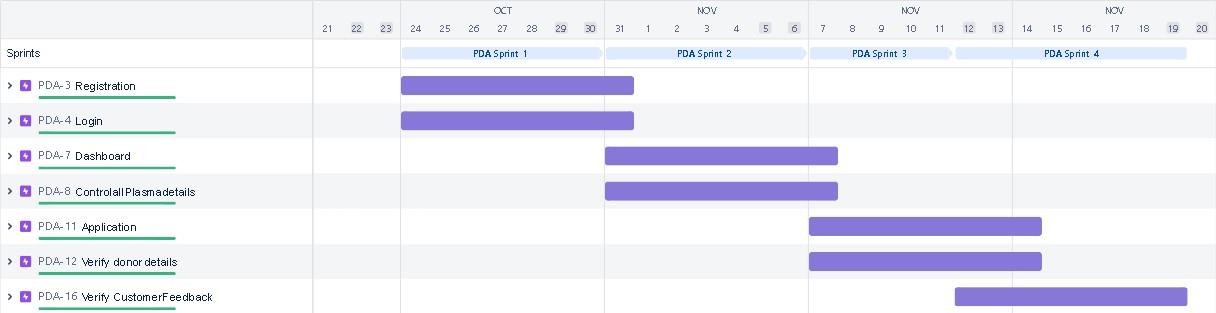
****



## Sprint delivery schedule



* 1. **Reports from JIRA**



## Feature 1:

1. **CODING & SOLUTIONING**

## Python

* Python is a popular, interpreted, object-oriented, high-level, dynamically semantic, and general-purpose programming language. Whether they are aware of it or not, consumers use many Python-powered devices on a regular basis.
* On February 20, 1991, Guido van Rossum released the first version of Python.
* Several languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, the Unix shell, and other scripting languages are the ancestors of Python.
* Python possesses a copyright. Python source code is now accessible under the GNU General Public License, just like Perl (GPL)
* It is simple to learn - learning Python takes less time than learning many other languages, making it possible to get started on programming right away.
* It is simple to use for creating new software, and Python frequently makes it possible to write code more quickly.
* Python is free, open, and multiplatform; not all languages can make that claim. It is simple to get, install, and deploy.
* Programming abilities are necessary if you wish to rise to increasingly complex and lucrative software development and engineering roles and equip you for careers in practically any industry.
* Currently, a core development team at the institute is responsible for maintaining Python, though Guido van Rossum continues to play a key role in guiding its development.

## Feature 2:

**Flask**

* Python-based Flask is a microweb framework. Due to the fact that it doesn't require any specific tools or libraries, it is categorised as a microframework.
* It lacks any components where pre-existing third-party libraries already provide common functions, such as a database abstraction layer, form validation, or other components. However, Flask allows for extensions that may be used to add application functionalities just like they were built into the core of Flask.
* There are extensions for object-relational mappers, form validation, upload handling, different open authentication technologies, and a number of tools associated with common frameworks.
* Pinterest and LinkedIn are two programmes that utilise the Flask framework.

## Database Scheme

**IBM Db2**

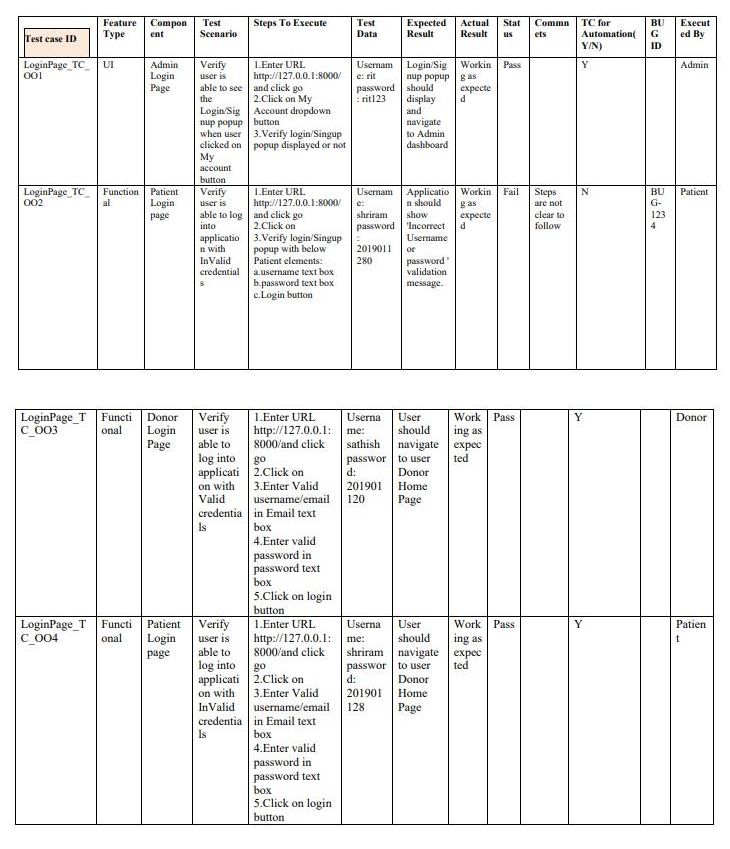
* A database product from IBM is called DB2.
* A relational database management system is what it is. DB2 is made to efficiently store, analyze, and retrieve the data.
* XML-based non-relational structures and support for Object-Oriented features are added to the DB2 product.
* Offer a massively parallel processing (MPP) architecture that simultaneously uses Apache Spark, HBase, and Hive for the best possible analytic performance.
* provides federation capabilities, high performance, and low latency support for ad-hoc and complex queries. Advanced row and column security is possible thanks to the understanding of dialects from other vendors and a variety of Oracle, IBM Db2, and IBM Netezza products.

# Kubernates

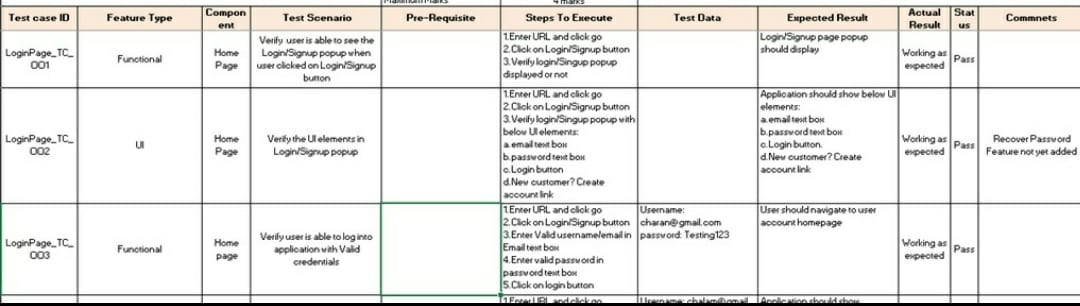
* Kubernetes is also referred to as "k8s."
* A flexible, portable, and open-source platform called Kubernetes was created by Google in 2014.
* Its primary usage is to automatically deploy, scale, and run container-based applications across a cluster of nodes.
* In a variety of physical, virtual, and cloud contexts, Kubernetes aids in managing containerized applications.
* To reliably deliver complex applications running on clusters of hundreds to thousands of individual servers, Google Kubernetes is a highly flexible container tool.
* The Linux kernel used for distributed systems is known as Kubernetes.
  1. **Test case**

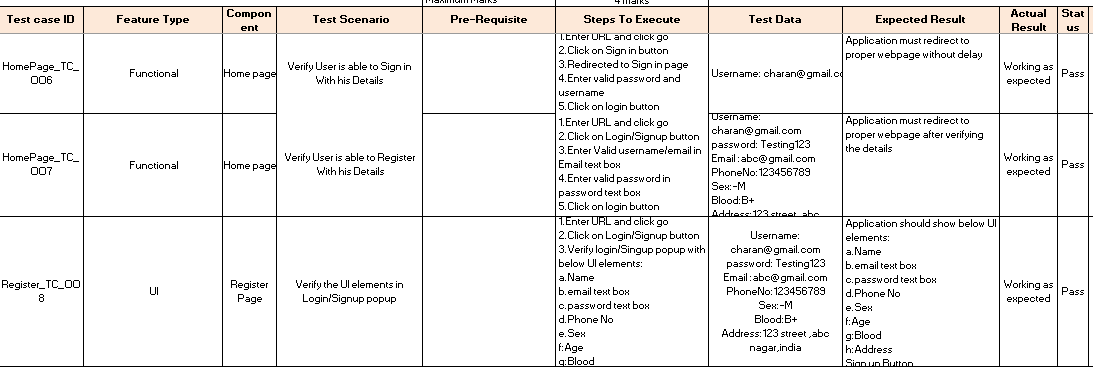
# TESTING

* It is the process of testing software with the goal of ensuring that it satisfies user expectations and meets requirements without failing in an unacceptable way.
* Different test types exist. Every test type responds to a distinct testing requirement

.

## User Acceptance Testing





1. **Purpose of Document**

The purpose of this document is to briefly explain the test coverage and open issues of the Plasma Donation Application project at the time of the release to User Acceptance Testing (UAT).

## Defect Analysis

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Resolution** | **Severity 1** | **Severity 2** | **Severity 3** | **Severity 4** | **Sub total** |
| **By Design** | 8 | 4 | 2 | 3 | 17 |
| **Duplicate** | 1 | 0 | 2 | 1 | 4 |
| **External** | 2 | 3 | 0 | 1 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Fixed** | 10 | 2 | 5 | 18 | 35 |
| **Not Reproduced** | 0 | 0 | 1 | 0 | 1 |
| **Skipped** | 0 | 0 | 1 | 1 | 2 |
| **Won't Fix** | 0 | 3 | 2 | 1 | 6 |
| **Totals** | 21 | 12 | 13 | 25 | 7  1 |

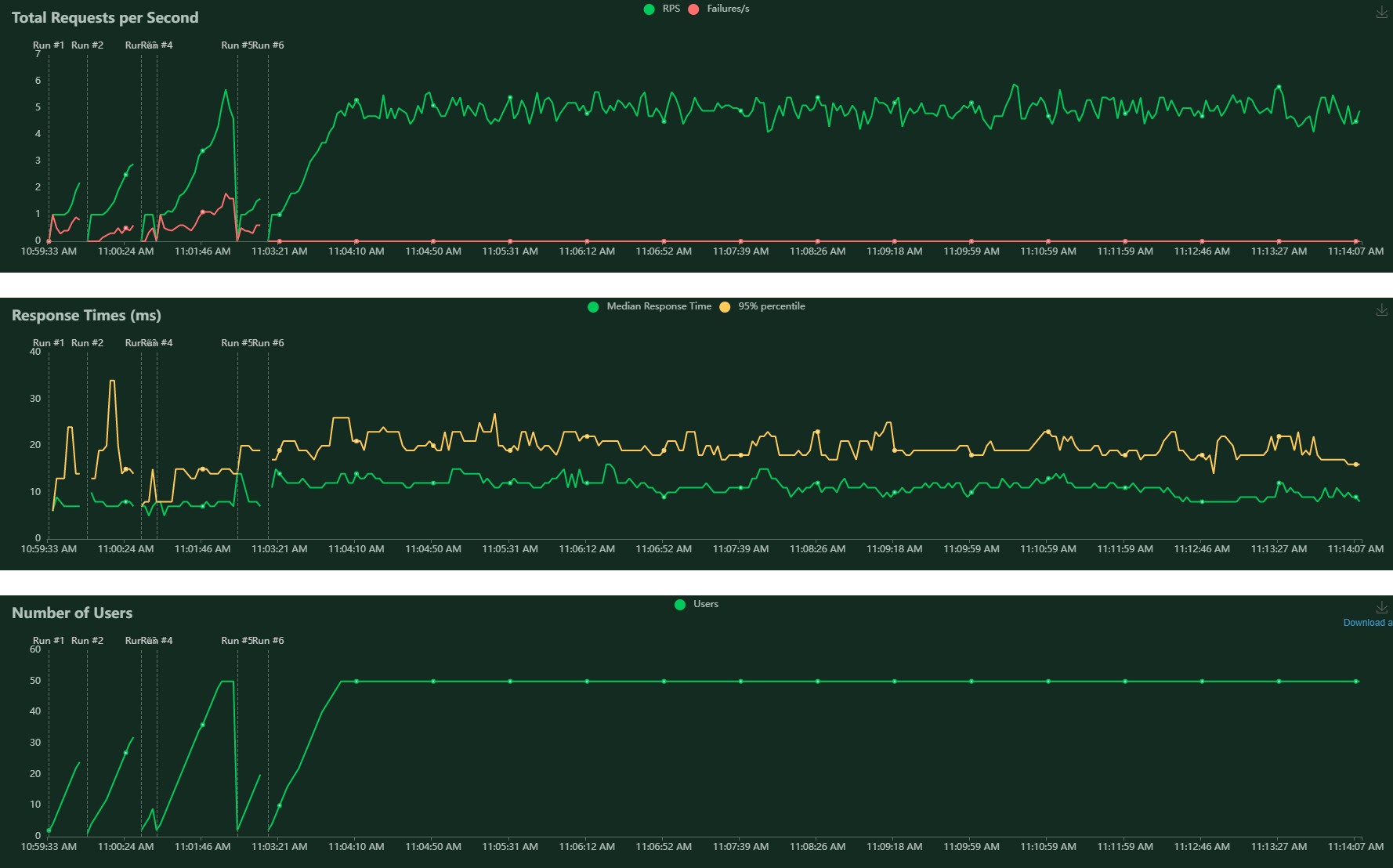
1. Test Case Analysis

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Section** | **Total Cases** | **Not Tested** | **Fail** | **Pass** |
| **Print Engine** | 8 | 0 | 0 | 8 |
| **Client Application** | 50 | 0 | 0 | 50 |
| **Security** | 2 | 0 | 0 | 2 |
| **Outsource Shipping** | 3 | 0 | 0 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exception Reporting** | 10 | 0 | 0 | 10 |
| **Final Report Output** | 6 | 0 | 0 | 6 |
| **Version Control** | 3 | 0 | 0 | 3 |





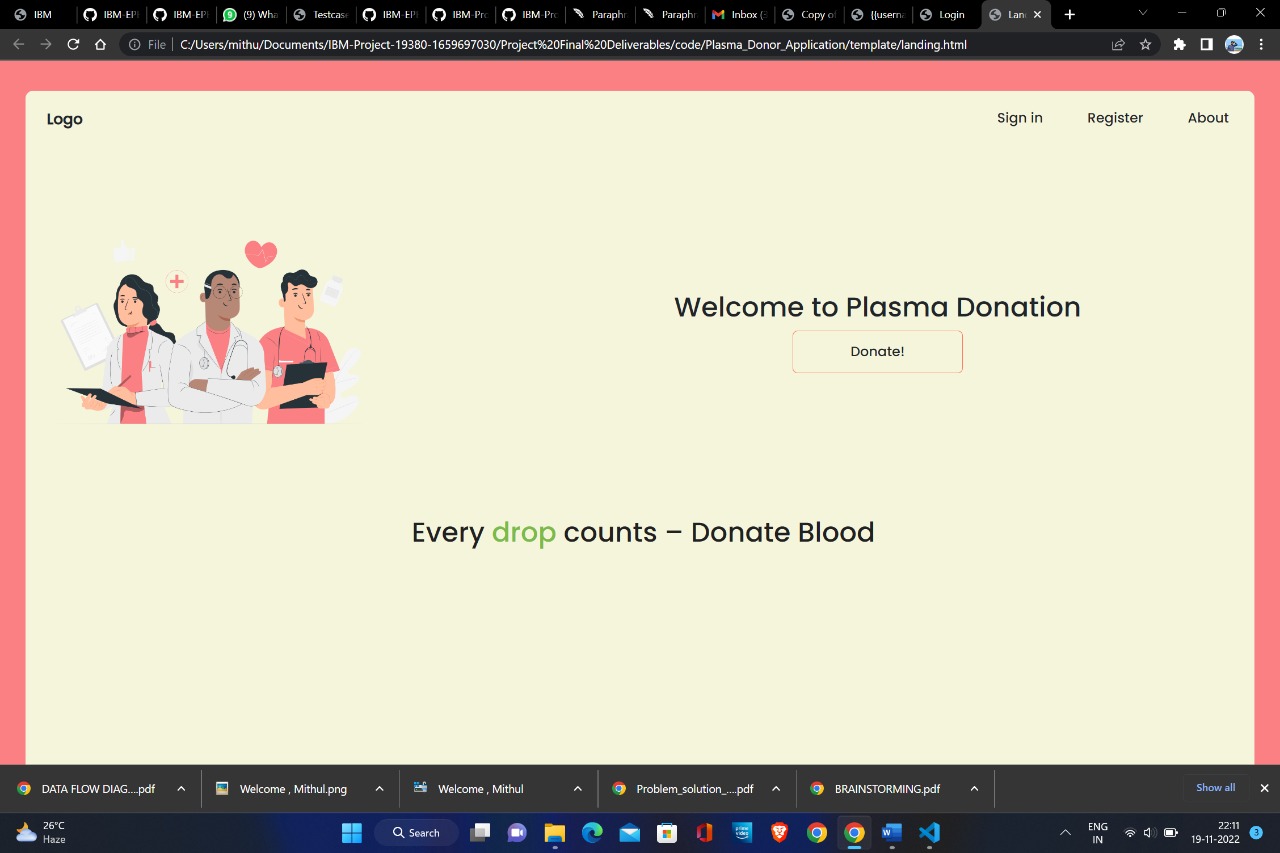
**9.1 Performance Metrics**

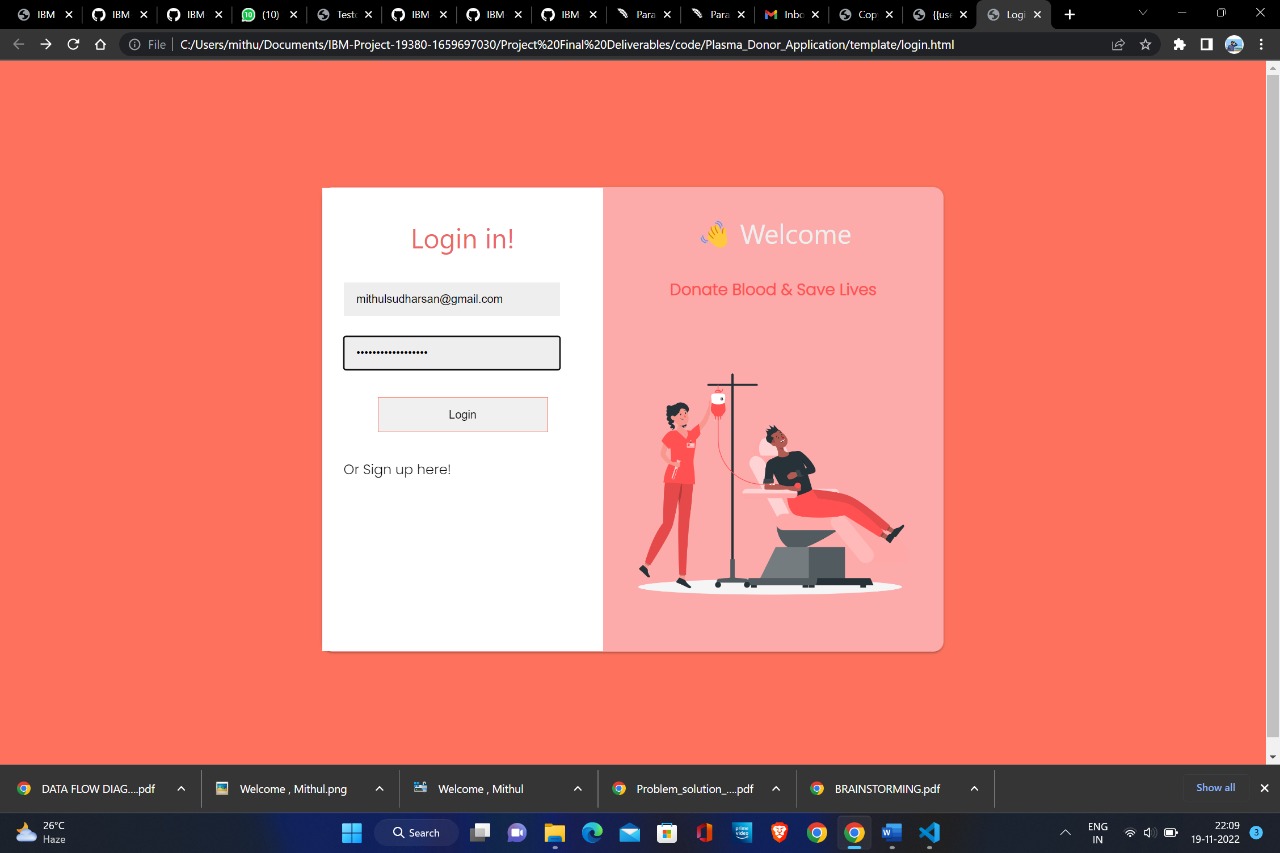
# RESULTS

* Project metrics are used to track the progress and performance of a project.
* Monitoring parts of a project like productivity, scheduling, and scope make it easier for team leaders to see what's on track.
* As a project evolves, managers need access to changing
* deadlines or budgets to meet their client's expectations

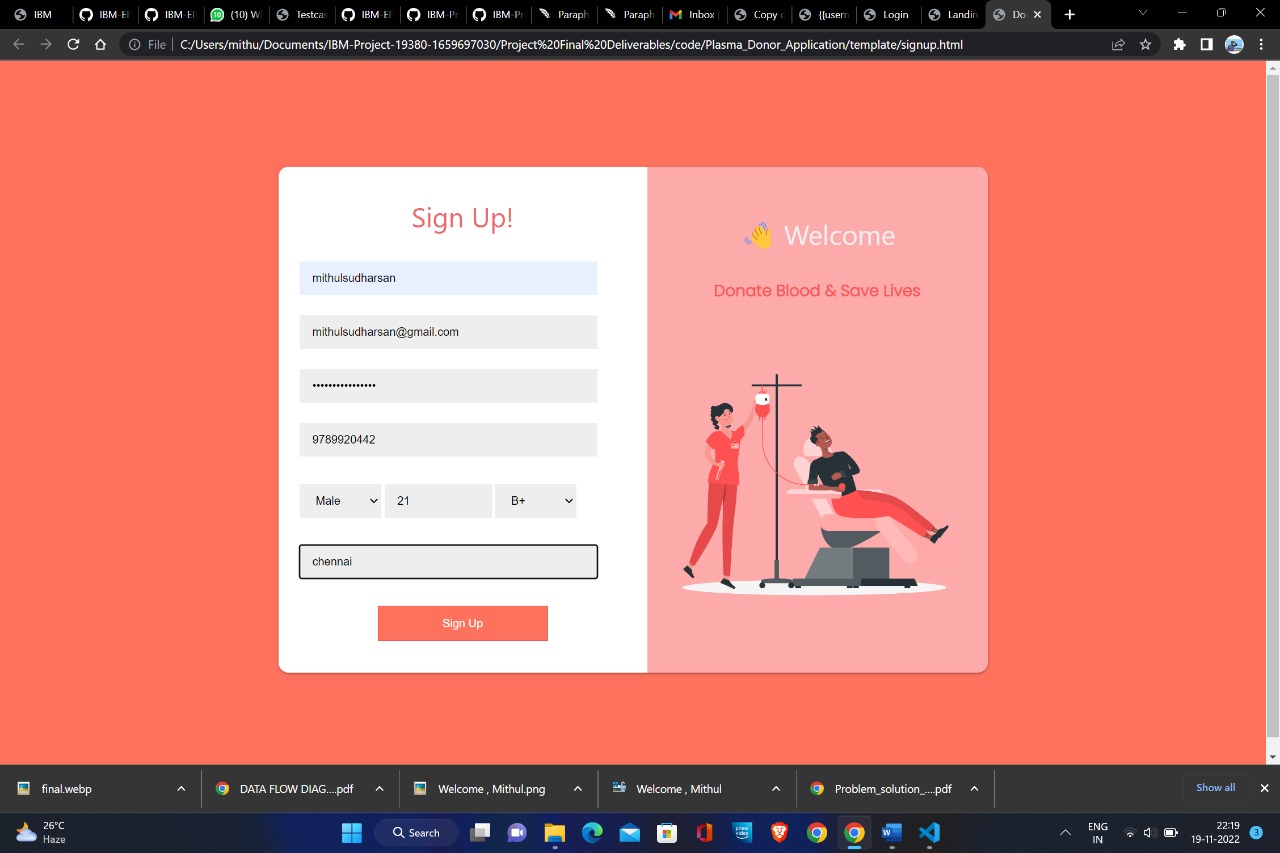
# OUTPUT SCREENS

Landing page

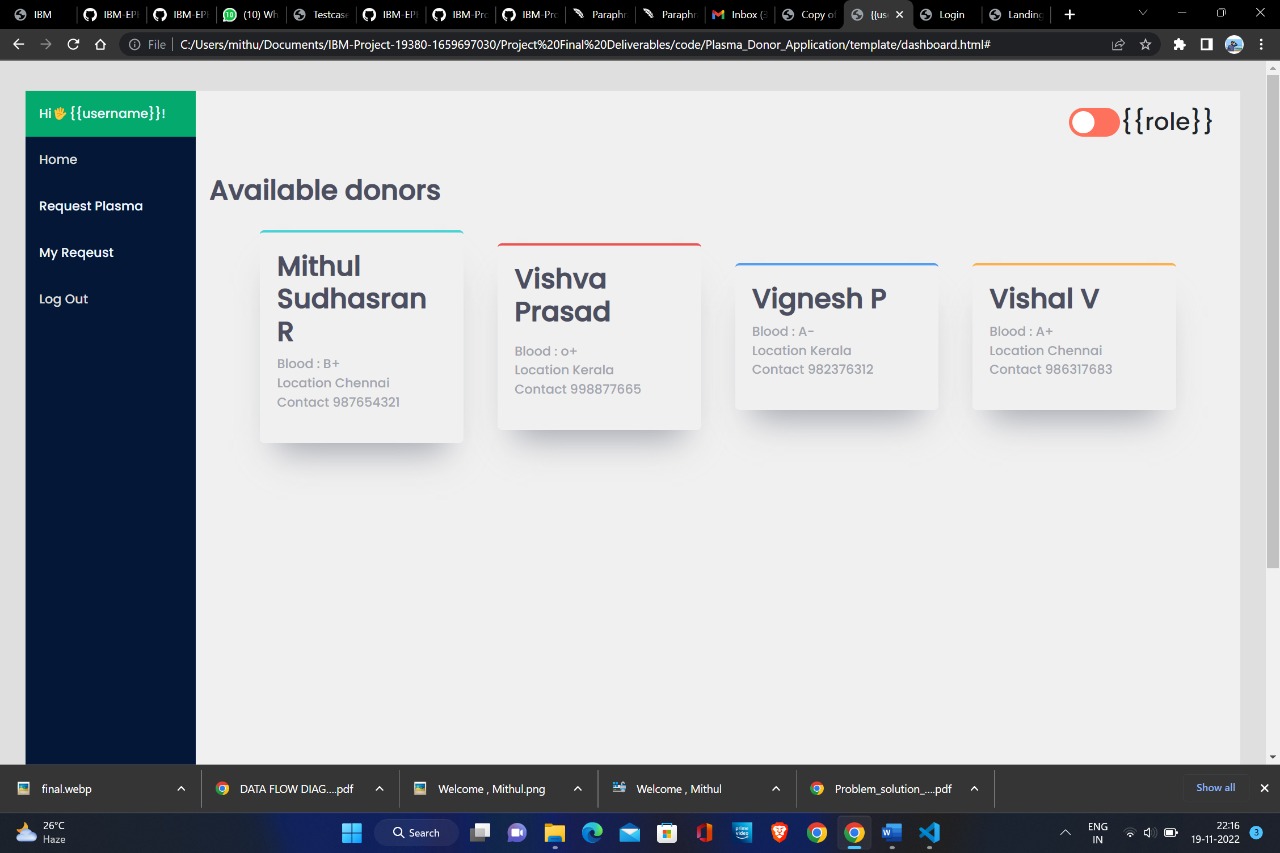




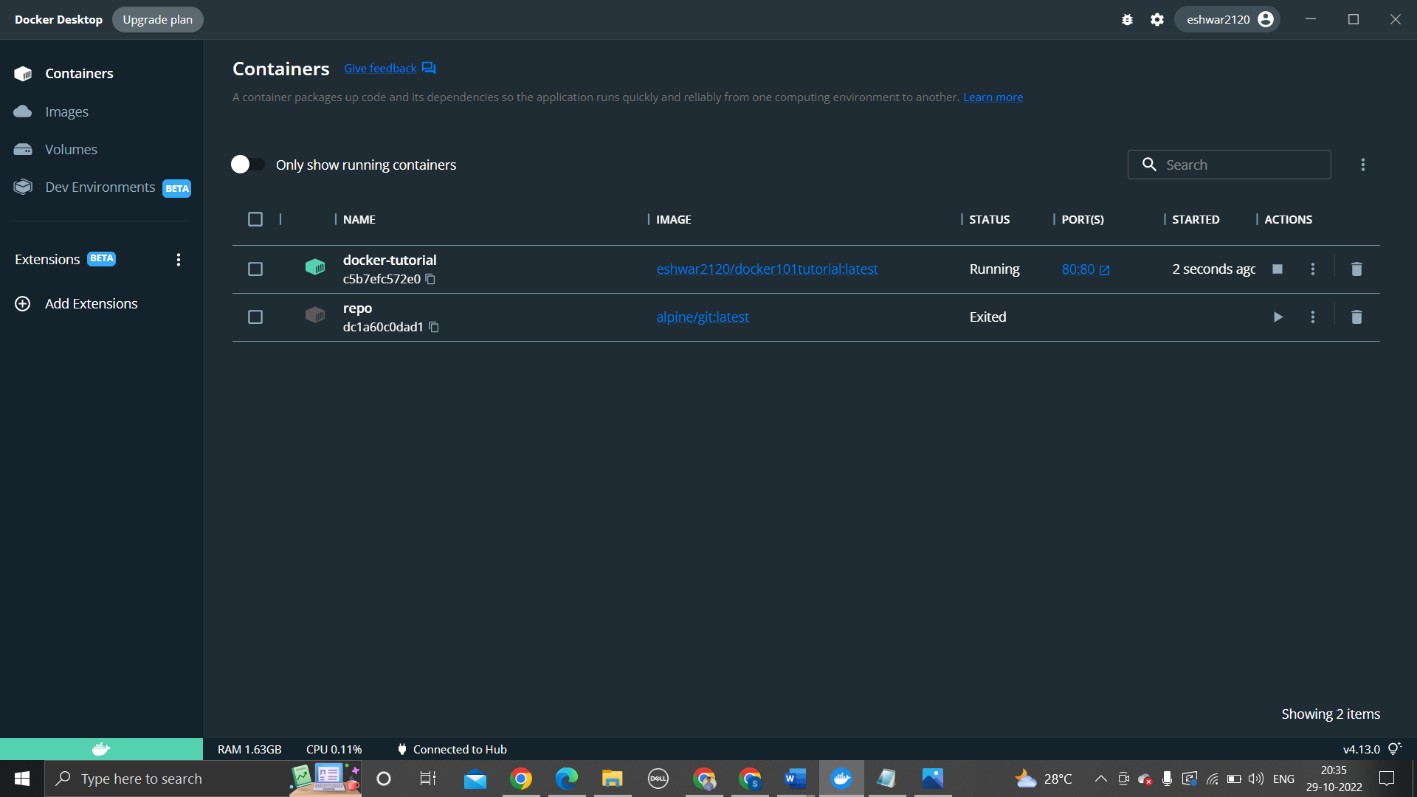
## Register Page



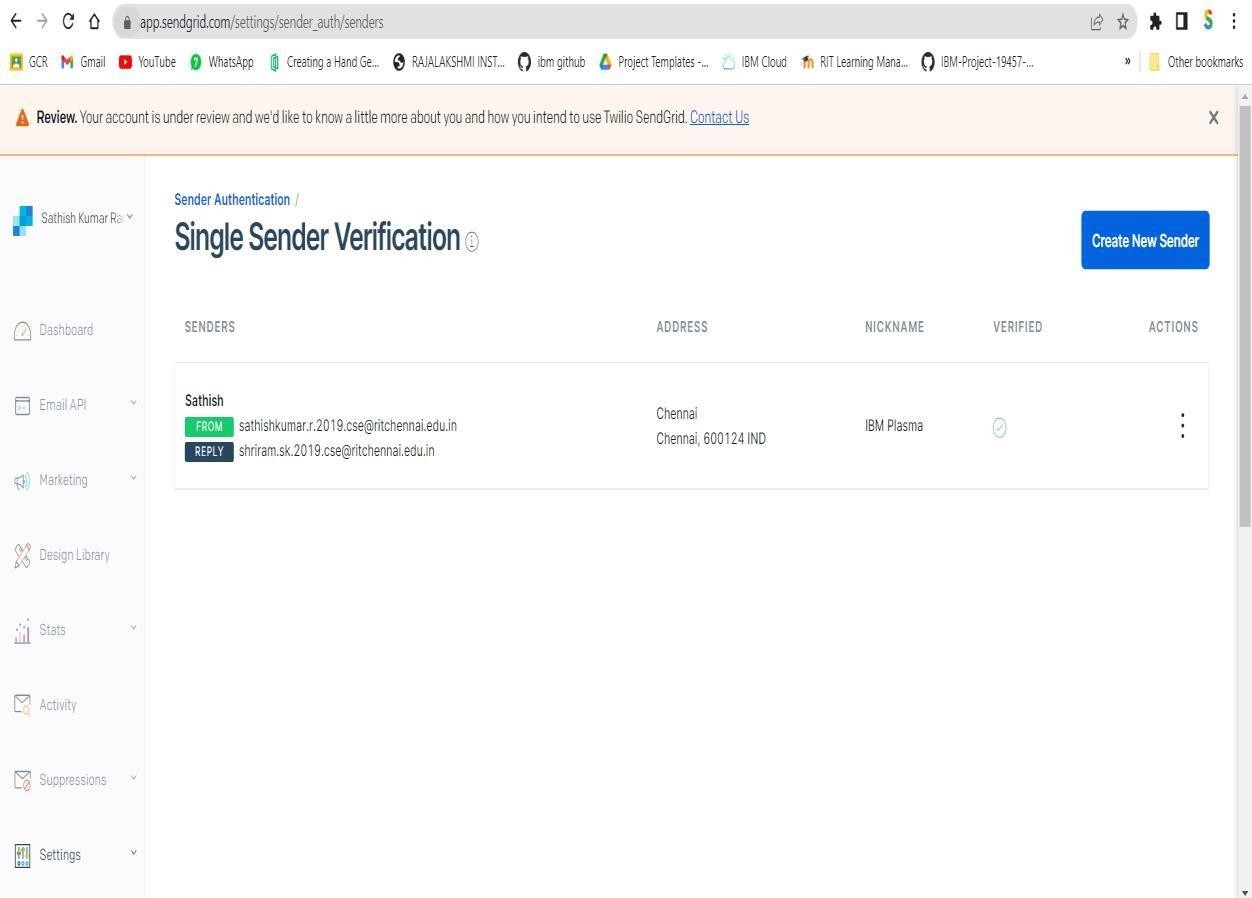
**Dashboard**



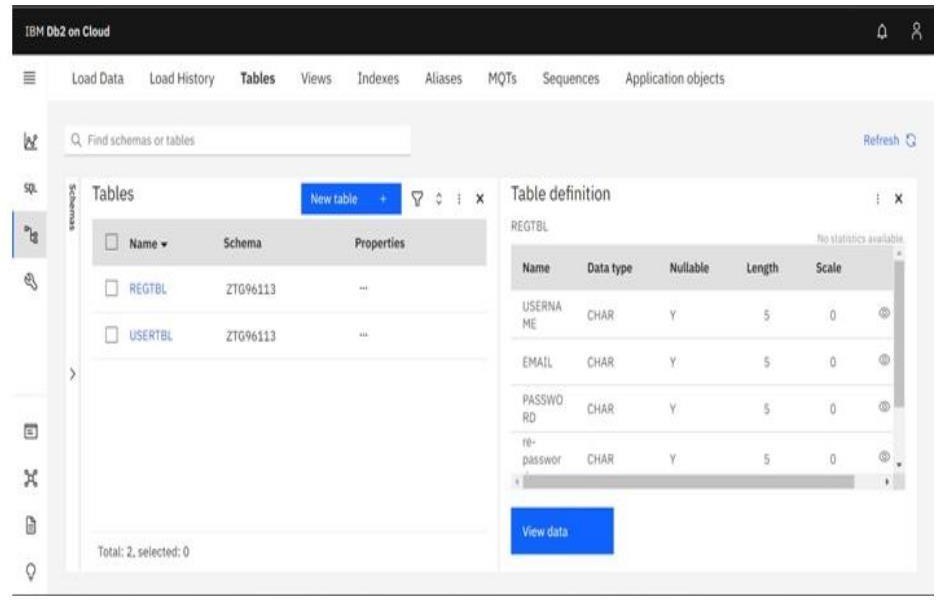
Dockerize the app



## SEND GRID



**IBM DB2**



# ADVANTAGES & DISADVANTAGES

## ADVANTAGES:

* Speed: When compared to manual register keeping, this website is quick and delivers excellent accuracy.
* Less maintenance is needed, and it is incredibly user-friendly and simple to grasp. It is simple to use and open to everyone.
* Fast Results: Depending on its availability, it would assist you in finding plasma donors quickly.

## DISADVANTAGES:

* The website would need to be connected to the internet in order to function.
* Auto-verification is not possible because only legitimate users can be confirmed.

# CONCLUSION

* Using the plasma donor website, which is housed on the IBM Cloud platform, afflicted individuals can discover plasma doors in an effective manner.
* To guarantee the efficient running of the website. To ensure that the activities are going smoothly, I have hosted the website in an IBM Db2 and Kubernates Cluster. In order to deploy the application, IBM Db2 service is employed together with cloud lambda function.

# FUTURE SCOPE

* Making the user interface (UI) more user-friendly, which will make it easier for many users to access the website and ensure that there are plenty of new plasma donors who can join the community.
* Using an elastic load balancer makes it possible to manage multiple requests at once, maintaining website uptime with minimal downtime.

# APPENDIXES

## SAMPLE SOURCE CODE: DONOR MAIN.py

from flask import Flask, redirect, url\_for, render\_template, request, make\_response,

jsonify, request import ibm\_db

from flask import request import json

conn = ibm\_db.connect( "DATABASE=bludb;HOSTNAME=764264db-9824-4b7c-82df-

40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=32536;SEC URITY=SSL;SSLServerCertificate=abc.crt;UID=gnq12618;PWD=0glS4tFaR2ciK8fB ",

'', '')

print(conn)

print("connection successful...") app = Flask( name )

import os

from sendgrid import SendGridAPIClient from sendgrid.helpers.mail import Mail

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

return render\_template("landing.html")

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

return render\_template("dashboard.html")

@app.route('/login', methods=['POST', 'GET']) def login():

if request.method == 'POST':

username = request.form['username'] password = request.form['password']

sql = "select \* from user where username=? and password=?" stmt = ibm\_db.prepare(conn, sql)

ibm\_db.bind\_param(stmt, 1, username)

ibm\_db.bind\_param(stmt, 2, password) ibm\_db.execute(stmt)

dic = ibm\_db.fetch\_assoc(stmt) print(dic)

role = str() requests = [] if dic:

role = dic['ROLE']

# sql = "select \* from user where blood\_group=?" # stmt = ibm\_db.prepare(conn, sql)

# ibm\_db.bind\_param(stmt, 1, username) # ibm\_db.execute(stmt)

# dic = ibm\_db.fetch\_assoc(stmt)

# while dic != False:

# single\_request = {

# 'name': dic['NAME'],

# 'age': dic['AGE'],

# 'sex': dic['SEX'],

# 'blood\_type': dic['BLOOD\_TYPE'] # }

# print(single\_request)

# requests.append(single\_request) # dic = ibm\_db.fetch\_assoc(stmt)

return render\_template('dashboard.html', username=username, role=role)

else:

return redirect(url\_for('login')) return redirect(url\_for('home'))

elif request.method == 'GET':

return render\_template('login.html')

@app.route('/signup', methods=['POST', 'GET']) def signup():

if request.method == 'POST':

username = request.form['username'] email = request.form['email'] password = request.form['password'] roll\_no = request.form['roll\_no']

sex = request.form['sex'] age = request.form['age']

address = request.form['address'] blood\_group = request.form['blood\_group'] sql = "insert into user values(?,?,?,?,?,?,?,?,?)" prep\_stmt = ibm\_db.prepare(conn, sql) ibm\_db.bind\_param(prep\_stmt, 1, username)

ibm\_db.bind\_param(prep\_stmt, 2, email)

ibm\_db.bind\_param(prep\_stmt, 3, password)

ibm\_db.bind\_param(prep\_stmt, 4, roll\_no)

ibm\_db.bind\_param(prep\_stmt, 5, sex)

ibm\_db.bind\_param(prep\_stmt, 6, age)

ibm\_db.bind\_param(prep\_stmt, 7, "USER")

ibm\_db.bind\_param(prep\_stmt, 8, address)

ibm\_db.bind\_param(prep\_stmt, 9, blood\_group) ibm\_db.execute(prep\_stmt)

# db post operation

return redirect(url\_for('login')) elif request.method == 'GET':

return render\_template('signup.html')

@app.route('/toggle', methods=['POST']) def toggle\_user():

data = request.get\_json(force=True)

username = data['username']

role = data['role'] print(username) print(role)

sql = "update user set role=? where username=?" prep\_stmt = ibm\_db.prepare(conn, sql) ibm\_db.bind\_param(prep\_stmt, 1, role)

ibm\_db.bind\_param(prep\_stmt, 2, username) ibm\_db.execute(prep\_stmt)

return jsonify( status="success", role=role

)

@app.route('/requestPalsma', methods=['POST']) def requestBloodPlasma():

# fetch mail address of the donors data = request.get\_json(force=True) username = data['username']

name = data['name'] age = data['age'] sex = data['sex']

blood\_type = data['bloodtype'] phone\_number = data['phone\_num']

sql = "select email from user where blood\_group=?" stmt = ibm\_db.prepare(conn, sql) ibm\_db.bind\_param(stmt, 1, blood\_type)

ibm\_db.execute(stmt)

dic = ibm\_db.fetch\_assoc(stmt) email\_list = []

while dic != False: email\_list.append(dic['EMAIL']) print(dic['EMAIL'])

dic = ibm\_db.fetch\_assoc(stmt) # send mail

message = Mail( from\_email='eshwaran.s.2019.cse@rajalakshmi.edu.in', to\_emails=email\_list,

subject='Sending with Twilio SendGrid is Fun',

html\_content='<h1>Need Of Blood</h1><table><tr><th>Name</th><th>' + name + '</th></tr><tr><th>Age</th><th>' + age + '</th></tr><tr><th>Sex</th><th>'

+ sex + '</th></tr><tr><th>Blood Group</th><th>' + blood\_type + '</th></tr><tr><th>Phone Number</th><th>' + phone\_number + '</th></tr></table>'

)

try:

sg = SendGridAPIClient("SG.3iBLSgAYTEuVbfSHu9dCPA.- nrnikWJvaRlNLMONA04\_CuKAyPeV69c46vPAh3vUX0")

response = sg.send(message) print(response.status\_code) print(response.body) print(response.headers)

except Exception as e: print(e.message)

# insert data into requests table

sql = "insert into bloodrequests(username,name,age,sex,blood\_type) values (?,?,?,?,?)"

prep\_stmt = ibm\_db.prepare(conn, sql) ibm\_db.bind\_param(prep\_stmt, 1, username)

ibm\_db.bind\_param(prep\_stmt, 2, name)

ibm\_db.bind\_param(prep\_stmt, 3, age)

ibm\_db.bind\_param(prep\_stmt, 4, sex)

ibm\_db.bind\_param(prep\_stmt, 5, blood\_type) ibm\_db.execute(prep\_stmt)

return jsonify( name=name, age=age, sex=sex,

bloodtype=blood\_type, status="yes"

)

@app.route('/getrequests', methods=['POST']) def getBloodRequests():

data = request.get\_json(force=True) username = data['username']

sql = "select \* from bloodrequests where username=?" stmt = ibm\_db.prepare(conn, sql) ibm\_db.bind\_param(stmt, 1, username) ibm\_db.execute(stmt)

dic = ibm\_db.fetch\_assoc(stmt)

requests = [] print(type(dic)) while dic != False:

single\_request = { 'name': dic['NAME'],

'age': dic['AGE'],

'sex': dic['SEX'],

'blood\_type': dic['BLOOD\_TYPE']

}

print(single\_request) requests.append(single\_request) dic = ibm\_db.fetch\_assoc(stmt)

return jsonify( username=username, requests=requests

)

if name == ' main ': app.run(host="0.0.0.0", debug=True)

## GITHUB

https://github.com/IBM-EPBL/IBM-Project-19380-1659697030

## VIDEO LINK

https://drive.google.com/file/d/1\_vlc3GoltUzG8I9DyYKp-oHgXAIesKS0/view?usp=share\_link