PLASMA DONOR APPLICATION

PROJECT REPORT

Submitted by

DHEIVANI.A	710319104005
PARAMASAKTHI.G	710319104017
SANTHOSHKUMAR.K	710319104024
SNEHA.S	710319104026

TEAM ID : PNT2022TMID42337

ANGELCOLLEGE OF TECHNOLOGY,

TITUPUR

DHARAPPURAM ROAD, PK PALAYAM–641665

TABLE OF CONTENT

1. INTRODUCTION

a. **Project Overview**

1. This project aims at building a web App that automatically estimates food attributes such as ingredients and nutritional value by classifying the input image of food.

2. Due to the ignorance of healthy food habits, obesity rates are increasing at an alarming speed, and this is reflective of the risks to people's health

3. People need to control their daily calorie intake by eating healthier foods, which is the most basic method to avoid obesity

4. However, although food packaging comes with nutrition (and calorie) labels, it's still not very convenient for people to refer to App-based nutrient dashboard systems which can analyze real-time images of a meal and analyze it for nutritional content which can be very handy and improves the dietary habits, and therefore, helps in maintaining a healthy lifestyle.

B.**Purpose**

The Purpose of our Project is

It helps dieticians with providing proper nutrition at healthcare facilities.

They determine patients nutritional needs.

It assess factors and plans, meals and menus.

They also ensure proper sterilization of plates and utensils.

Nutitionists work to help people establish good connections between healthy weights and overall health.

1. LITERATURE SURVEY

a. **Existing Problem :**

1. Patients who have to maintain diet have to give their body health details.

2. They have check their BMI value to predict the food for them.
3. Then the image or url of a food have to upload to know the further details of food.
4. Finally,the patients have to follow the predicted food and maintain diet with respect to the nutrition details of a doof which is obtained.

b. **References :**

☆https://www.ibm.com/blogs/internet-of-things/connected-trains-rail-travel/

☆https://indianrailways.gov.in/railwayboard/uploads/directorate/secretary_branches/IR_Reforms/Innovation%20in%20Indian% 20Railways%20.pdf

☆Indian Railways Vision 2020 (Para 6.1 Reinventing Passenger Services with Change for a better tomorrow as the motto, Page 8-9)

☆Travel, Transportation & Hospitality page on www.tcs.com

☆https://international-railway-saftey-council.com/about-us
☆http://sts.hitachirsil.com/en/products-services/glossary

C. Problem Statement definition :

Due to the ignorance of healthy food habits, obesity rates are increasing at an alarming speed, and this is reflective of the risks to people's health.

People need to control their daily calorie intake by eating healthier foods, which is the most basic method to avoid obesity. However, although food packaging comes with nutrition (and calorie) labels, it's still not very convenient for people to refer to App-based nutrient dashboard systems which can analyze real-time images of a meal and analyze it for nutritional content which can be very handy and improves the dietary habits, and therefore, helps in maintaining a healthy lifes

The main objective of this project is to building a web App that automatically estimates food attributes such as ingredients and nutritional value by classifying the input image of food.

1.Who are all affected by this issue?

➢People from all age group who are all careless about their health due to their busy schedule and high calorie diet.

➢This leads to an unhealthy lifestyle because of their eating habits.

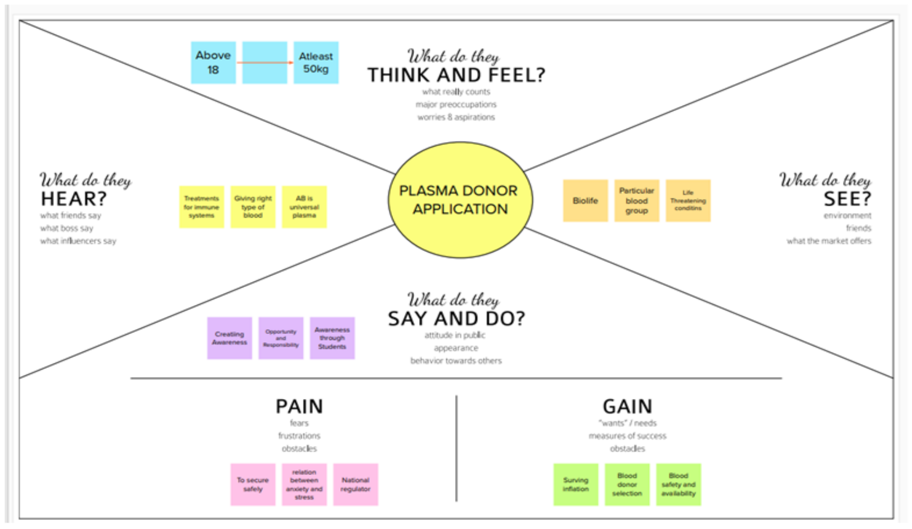
➢Thus leads to many health issues like obesity, heart attack, diabetics and rise in cholesterol level.

2.What are the boundaries of the problem?

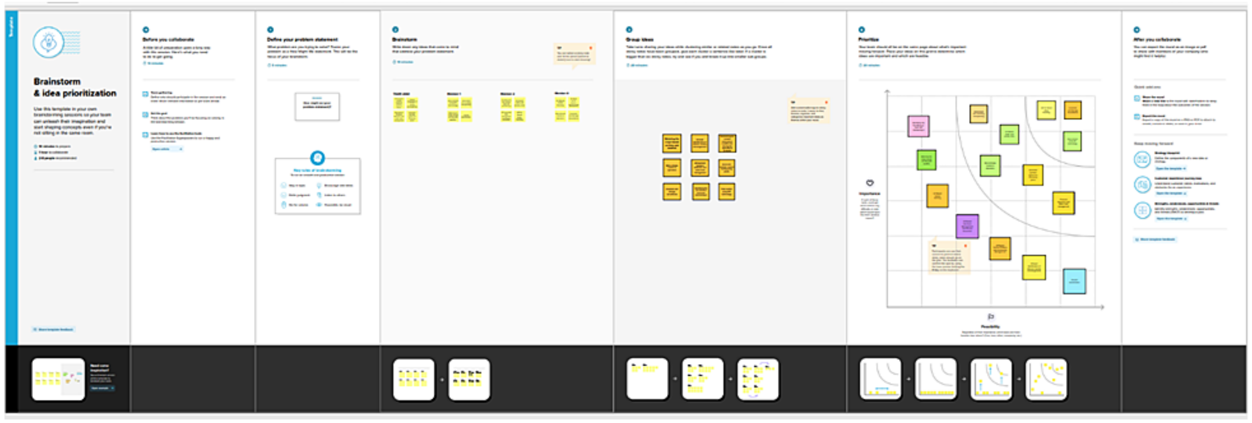
➢Based on the information collected from the user, if the user is diagnosed with diabetes/Heart attack/obesity then the application provides information about diet.
ems with digestion so they will be provided with that information.

IDEATION & PROPOSED SOLUTION

a. EMPATHY MAP CANVAS

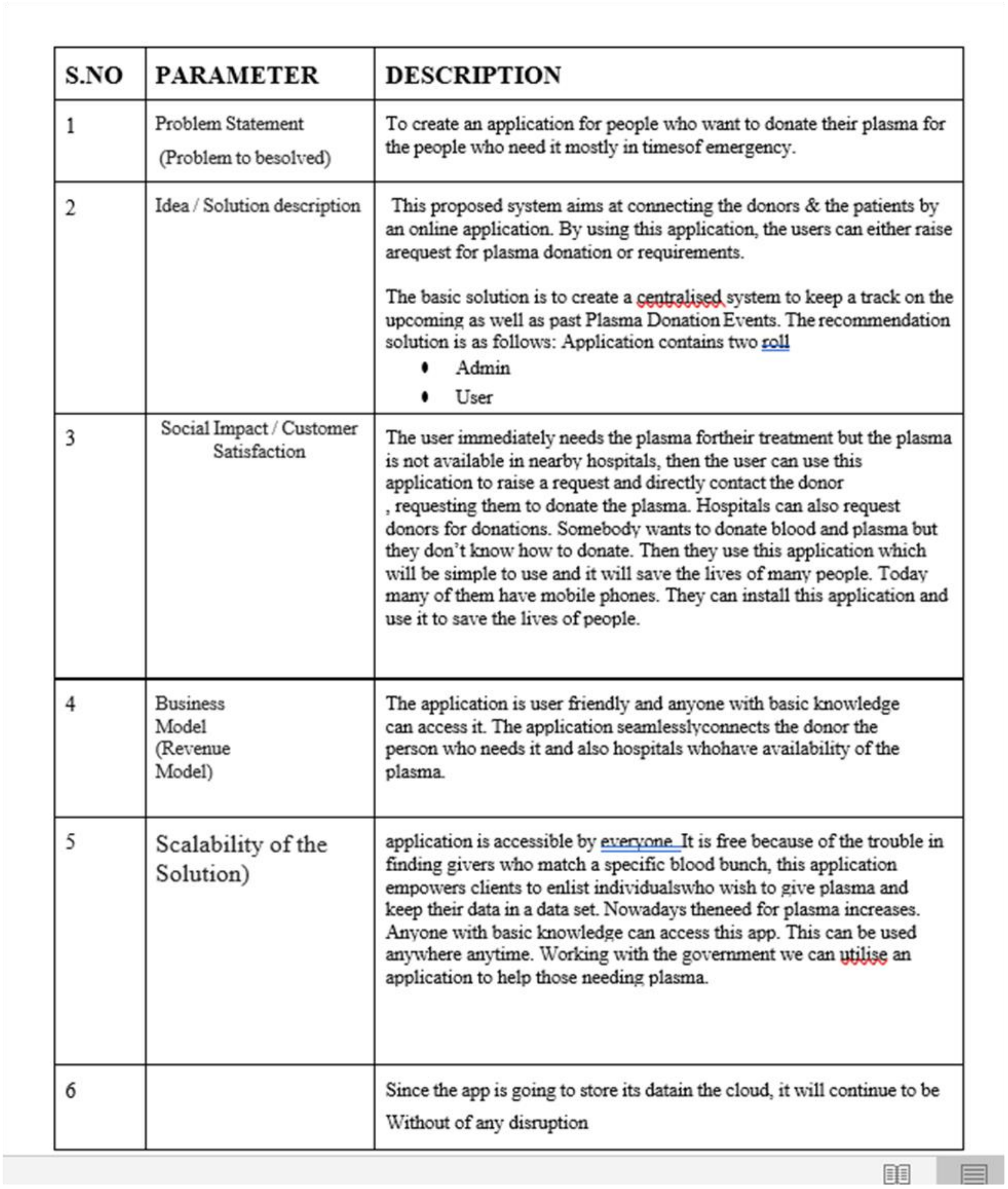


3.2 Ideation & Brainstorming



[BRAINSTORMING • solnForRailways (mural.co)]

3.3 Proposed Solution



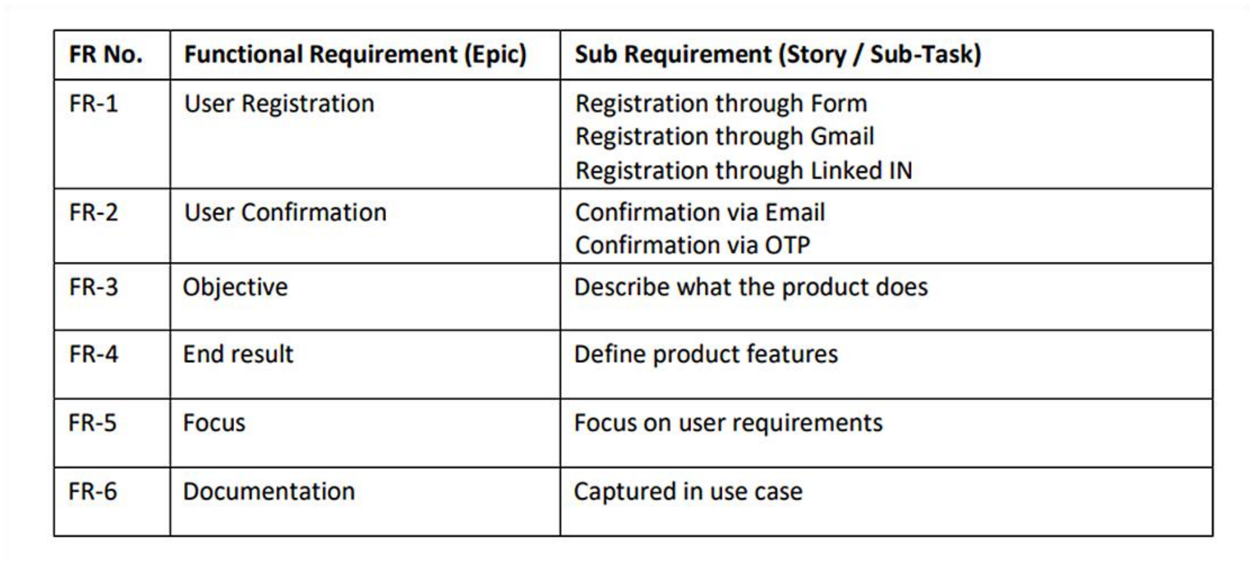
S.NO	PARAMETER	DESCRIPTION
1	Problem Statement (Problem to besolved)	To create an application for people who want to donate their plasma for the people who need it mostly in timesof emergency.
2	Idea / Solution description	This proposed system aims at connecting the donors & the patients by an online application. By using this application, the users can either raise arequest for plasma donation or requirements. The basic solution is to create a centralised system to keep a track on the upcoming as well as past Plasma Donation Events. The recommendation solution is as follows: Application contains two roll • Admin • User
3	Social Impact / Customer Satisfaction	The user immediately needs the plasma fortheir treatment but the plasma is not available in nearby hospitals, then the user can use this application to raise a request and directly contact the donor , requesting them to donate the plasma. Hospitals can also request donors for donations. Somebody wants to donate blood and plasma but they don't know how to donate. Then they use this application which will be simple to use and it will save the lives of many people. Today many of them have mobile phones. They can install this application and use it to save the lives of people.
4	Business Model (Revenue Model)	The application is user friendly and anyone with basic knowledge can access it. The application seamlesslyconnects the donor the person who needs it and also hospitals whohave availability of the plasma.
5	Scalability of the Solution)	application is accessible by everyone. It is free because of the trouble in finding givers who match a specific blood bunch, this application empowers clients to enlist individualswho wish to give plasma and keep their data in a data set. Nowadays theneed for plasma increases. Anyone with basic knowledge can access this app. This can be used anywhere anytime. Working with the government we can utilise an application to help those needing plasma.
6		Since the app is going to store its datain the cloud, it will continue to be Without of any disruption

3.4Problem Solution fit

REQUIREMENTANALYSIS

Functional requirement :

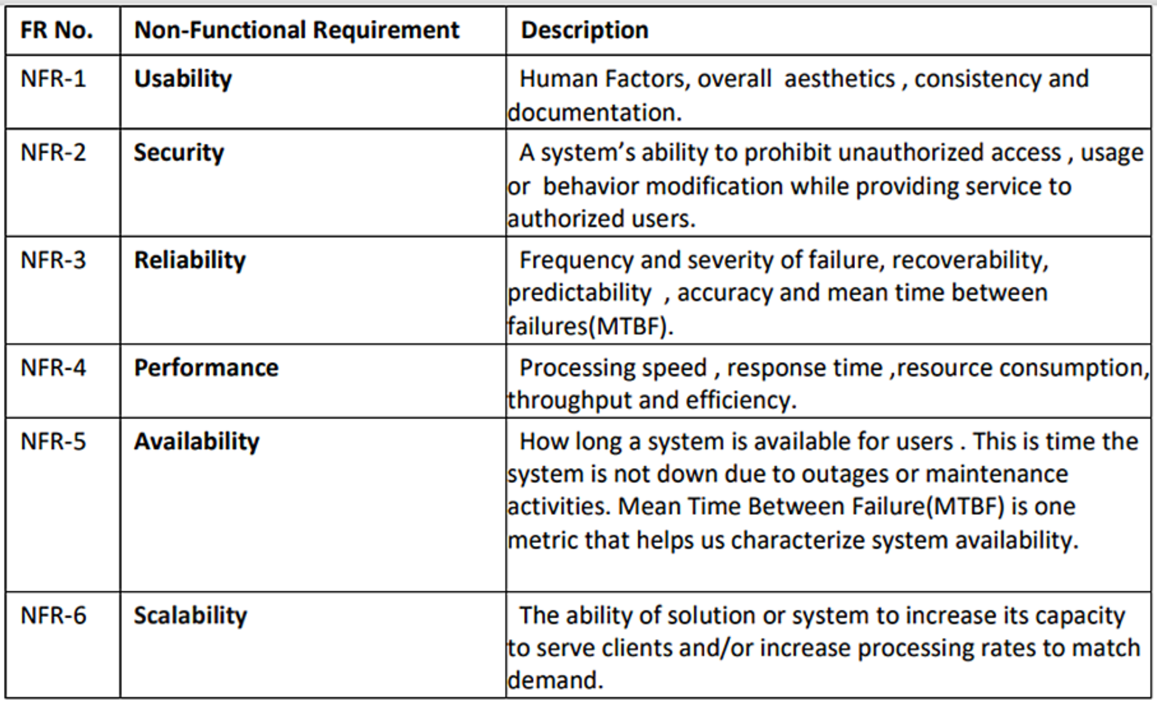
Following are the functional requirements of the proposed solution.



FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through Linked IN
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Objective	Describe what the product does
FR-4	End result	Define product features
FR-5	Focus	Focus on user requirements
FR-6	Documentation	Captured in use case

Non-Functional requirement

non-functional requirements of the proposed solution.

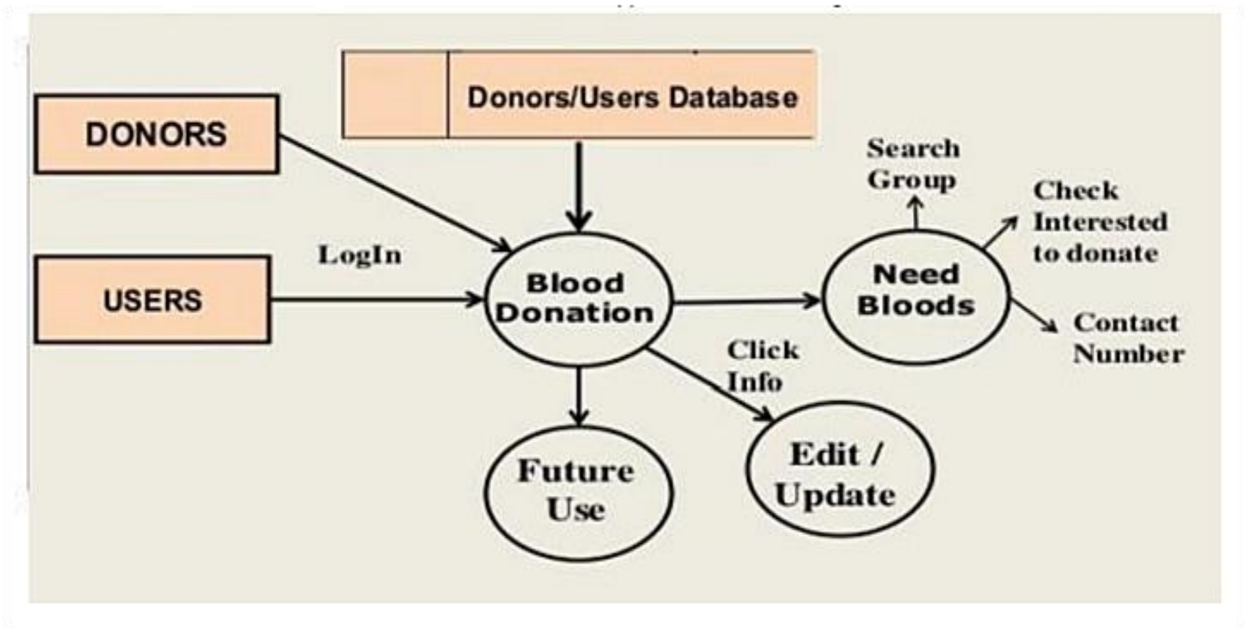


FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Human Factors, overall aesthetics , consistency and documentation.
NFR-2	**Security**	A system's ability to prohibit unauthorized access , usage or behavior modification while providing service to authorized users.
NFR-3	**Reliability**	Frequency and severity of failure, recoverability, predictability , accuracy and mean time between failures(MTBF).
NFR-4	**Performance**	Processing speed , response time ,resource consumption, throughput and efficiency.
NFR-5	**Availability**	How long a system is available for users . This is time the system is not down due to outages or maintenance activities. Mean Time Between Failure(MTBF) is one metric that helps us characterize system availability.
NFR-6	**Scalability**	The ability of solution or system to increase its capacity to serve clients and/or increase processing rates to match demand.

1. PROJECT DESIGN

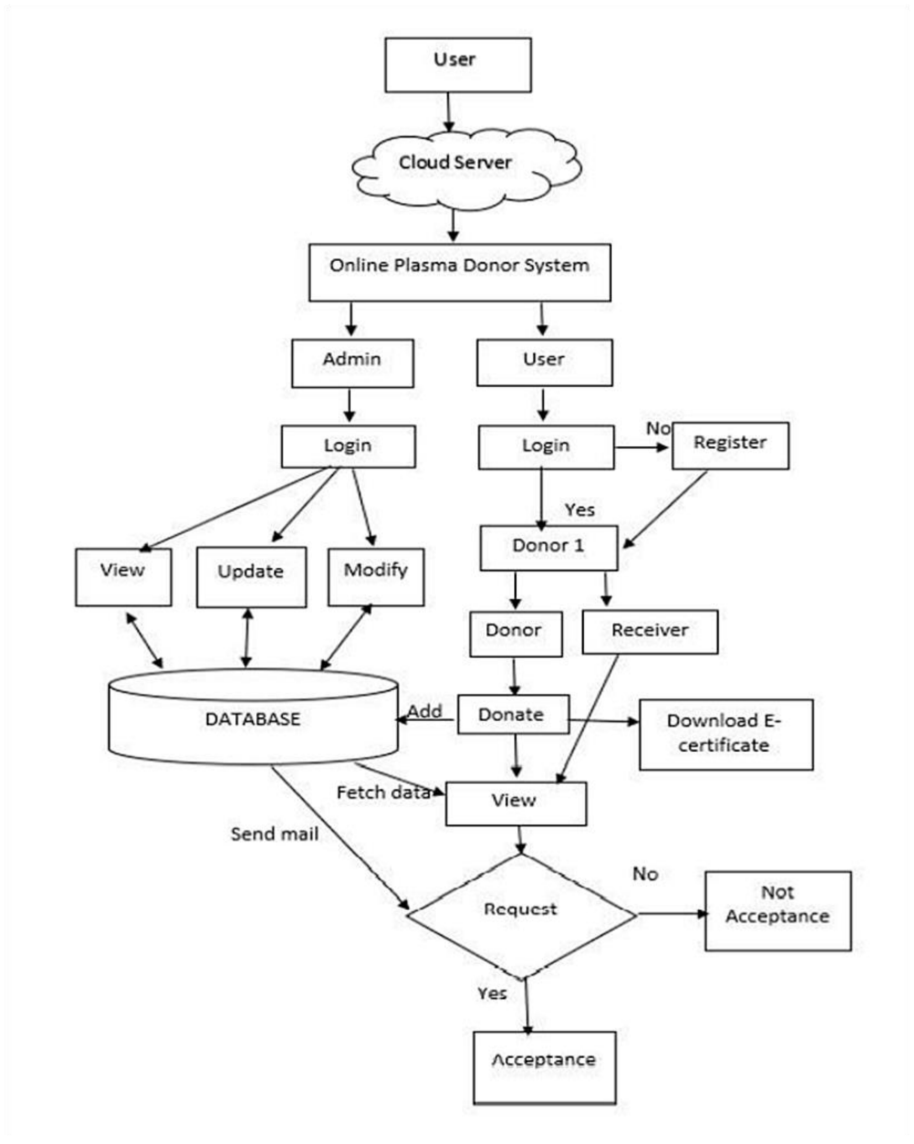
Data Flow Diagram

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

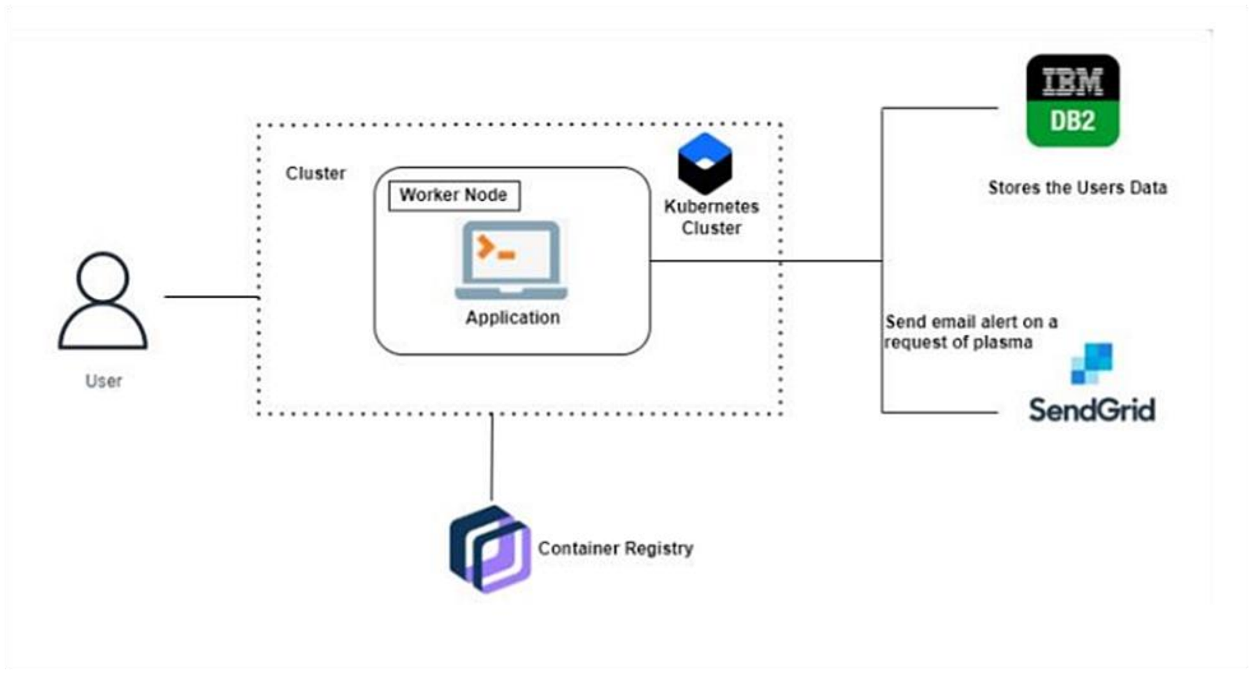


<u>Solution and Technical Architect</u>

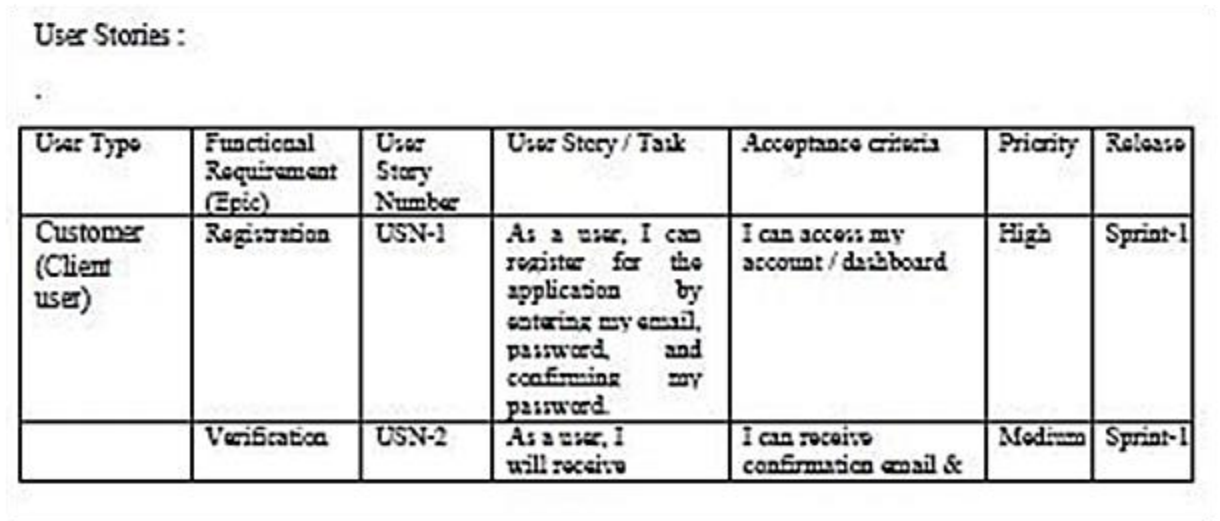
Solution Architecture



Technical Architecture:



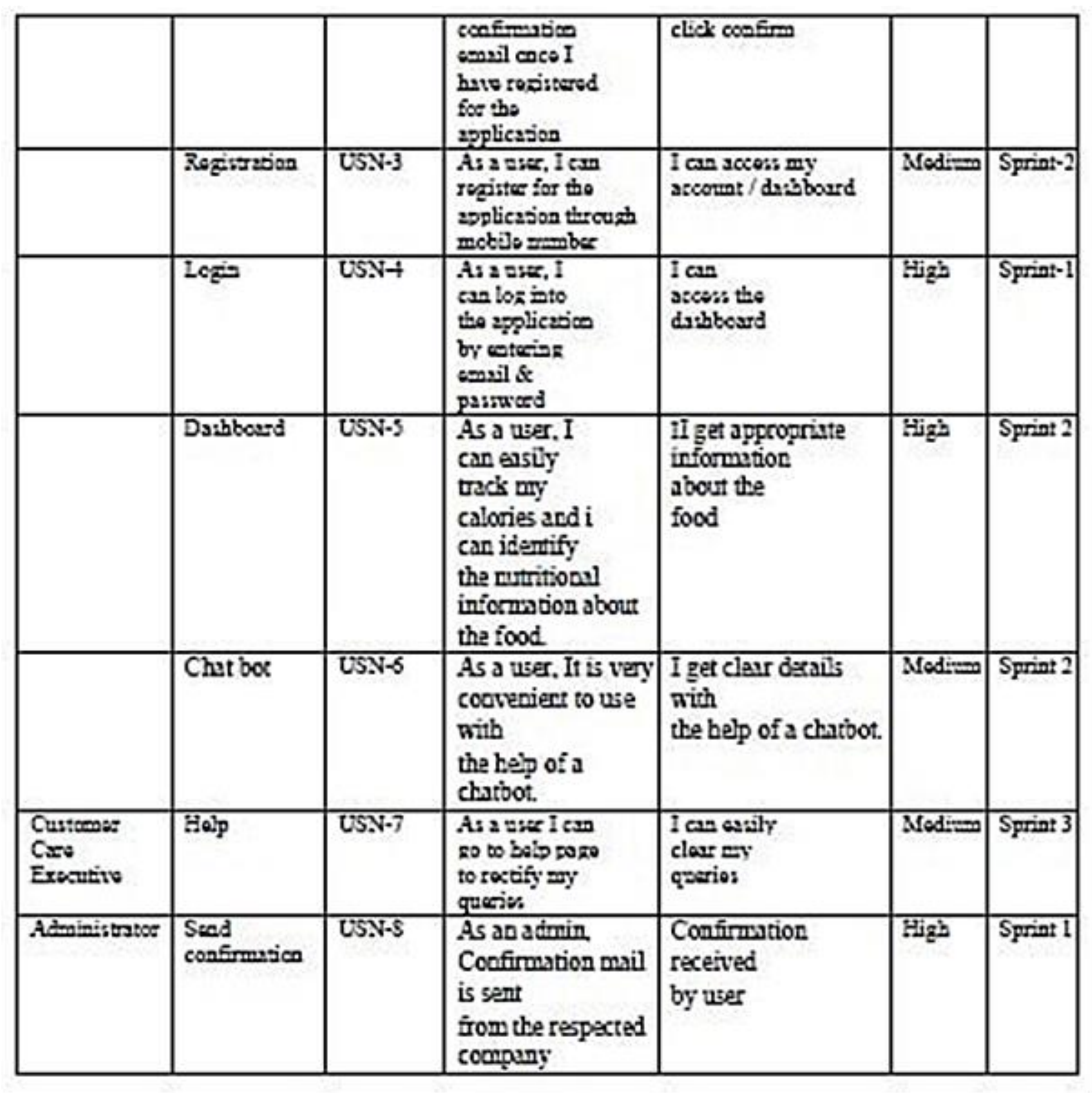
3 User Stories



User Stories :

.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Client 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
	Verification	USN-2	As a user, I will receive	I can receive confirmation email &	Medium	Sprint-1



			confirmation email once I have registered for the application	click confirm		
	Registration	USN-3	As a user, I can register for the application through mobile number	I can access my account / dashboard	Medium	Sprint-2
	Login	USN-4	As a user, I can log into the application by entering email & password	I can access the dashboard	High	Sprint-1
	Dashboard	USN-5	As a user, I can easily track my calories and i can identify the nutritional information about the food.	II get appropriate information about the food	High	Sprint 2
	Chat bot	USN-6	As a user, It is very convenient to use with the help of a chatbot.	I get clear details with the help of a chatbot.	Medium	Sprint 2
Customer Care Executive	Help	USN-7	As a user I can go to help page to rectify my queries	I can easily clear my queries	Medium	Sprint 3
Administrator	Send confirmation	USN-8	As an admin, Confirmation mail is sent from the respected company	Confirmation received by user	High	Sprint 1

PROJECT PLANNING & SCHEDULING
Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Donor Registration	USN-1	As a user, I can register in the donor application by entering my name, phone_no, Email id, blood group ,aadhar no	9	High	Team Lead (A.dheivani)
Sprint-1	Login	USN-2	As a admin, I can log into the application by entering email & password	9	High	Team Lead
Sprint-1	Chatbot	USN-3	As a user I can ask query in chatbot.	2	Medium	Team Lead
Sprint -2	Confimation	USN-4	As a user, I can receive confirmation mail.	4	Medium	Team Lead
Sprint - 2	Dashboard	USN-5	As a user, I can view dashboard and select	5	Medium	Team Member 1
Sprint-2	View Donor List	USN-6	As a user, I can view all the donor list and contact them directly	9	High	Team Lead
Sprint-2	Search Donor	USN-7	As a user, I can search for the donor	9	Medium	Team Lead
Sprint-3	About us	USN-8	As a User, I can view the about us page which contains all contact information	5	Medium	Team Member 2

Sprint Delivery Schedule:

Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	5 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

CODING &SOLUTION

Feature 1

Nutrition Assistant Application:

Description:

In feature 1 we have designed a webpage using node red to book the train ticket.The user can login in into the webpage using username and password . After successful login, the user wil be redirected to the Ticket booking form. In this form ,users are asked to fill the personal details and the jouney details . After entering the appropriate details the confirmation message is shown and QR code is generated.

HOME PAGE :

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Home</title>
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.1/css/all.css">
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="{{ url_for('static',filename='css/style.css') }}">
  </head>
  <body>
    {% extends "template.html" %}
    {% block content %}
    <form action="{{ url_for('requested')}}" method="post">
    <input type="text" name="name" placeholder="Enter Name" required="required"
style="color:black" />
        <input type="email" name="email" placeholder="Enter Email"
required="required" style="color:black"/>
            <input type="text" name="phone" placeholder="Enter 10-digit mobile number"
required="required" style="color:black"/>
            <select name="bloodgrp">
                                <option value="select" selected>Choose your blood
group</option>
                                <option value="O+">O Positive</option>
                                <option value="A+">A Positive</option>
                                <option value="B+">B Positive</option>
                                <option value="AB+">AB Positive</option>
                                <option value="O-">O Negative</option>
                                <option value="A-">A Negative</option>
                                <option value="B-">B Negative</option>
                                <option value="AB-">AB Negative</option>
            </select>
            <textarea rows="4" placeholder="Enter the address" required="required"
style="color:black" name="address"></textarea>
        <button type="submit" class="btn btn-primary btn-block btn-large">Submit the
request</button>

    </form>
    <div>
 {{ pred }}</div>
    {% endblock %}
  </body>
</html>
```

Login Page :

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
```

```
    <meta charset="utf-8" name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Login</title>
    <link rel="stylesheet" href="{{ url_for('static',    filename='css/login.css')
}}">

</head>
  <body>
    <div class="main">
      <p class="sign" align="center">Log in</p>
      {% with messages = get_flashed_messages() %}
            {% if messages %}
                  {% for message in messages %}
                  <div class="msg"> <p>{{ message }}</p>  </div>
                  {% endfor %}
            {% endif %}
          {% endwith %}

      <form class="form1" action="/loginmethod" method="get">
        <input class="un " name="uname" type="text" align="center"
placeholder="Username">
        <input class="pass" name="psw" type="password" align="center"
placeholder="Password">
        <input type="submit" class="submit" align="center"/>
        <p class="forgot" align="center"><a href="#">Forgot Password?</p>
        <a href="/signup"><p class="forgot" align="center">Don't have an
account? Signup</p></a>
      </form>
    </div>

{% block content %}
{% endblock %}

 </body>
</html>
```

Sign up page :

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
```

```
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Signup</title>
    <link rel="stylesheet" href="{{ url_for('static',    filename='css/login.css')
}}">

</head>
  <body>
    <div class="main">
        <p class="sign" align="center">Sign in</p>
          {% with messages = get_flashed_messages() %}
            {% if messages %}
                  {% for message in messages %}
                  <div class="msg"> <p>{{ message }}</p>  </div>
                  {% endfor %}
            {% endif %}
          {% endwith %}

        <form class="form1" action="/signupmethod" method="post">
          <input class="un" name="name" type="text" align="center"
placeholder="Name">
          <input class="un" name="email" type="email" align="center"
placeholder="Email ID">
          <input class="un" name="dob" type="text"
onblur="(this.type='text')" onfocus="(this.type = 'date')"  id="date"
align="center" placeholder="Date of Birth">
          <input class="un" name="uname" type="text" align="center"
placeholder="Username">
          <input class="pass" name="psw" type="password" align="center"
placeholder="Password">
          <input class="pass" name="con_psw" type="password"
align="center" placeholder="Confirm Password">
          <input type="submit" class="submit" align="center"/>
          <p class="forgot" align="center"><a href="#">Forgot
Password?</a></p>
            <a href="/"><p class="forgot" align="center">Already have an
account? Login</p></a>
        </form>
    </div>

{% block content %}
{% endblock %}

 </body>
</html>
```

About :

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>About</title>

    <link href='https://fonts.googleapis.com/css?family=Nunito:400,300'
rel='stylesheet' type='text/css'>

    <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.8.1/css/all.css">
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="{{ url_for('static',filename='css/style.css') }}">
    <script type="text/javascript" src="{{ url_for('static',
filename='js/logreg.js') }}"></script>
  </head>
  <body>
    {% extends "template.html" %}
    {% block content %}
    <div class="center">
      <div>
        <button id = "Update" class = "update">Update</button>
      </div>
      <div>
        <button id = "Cancel" class = "update"
onClick="window.location.reload();">Cancel</button>
      </div>
      </div>
    <form action="/details" method="post">

      <h1>Personal Details</h1>

      <fieldset>
        <legend><span class="number">1</span>Enter Your Details</legend>
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" readonly="readonly"
placeholder="Enter Name" value="{{ account['NAME'] }}">

        <label for="uname">Username:</label>
        <input type="text" id="uname" name="uname" readonly="readonly"
value="{{ account['USERNAME'] }}">

  <label for="email">Email ID:</label>
        <input type="email" id="email" name="email" readonly="readonly"
placeholder="Enter Email" value="{{ account['EMAIL'] }}">

  <label for="dob">Date of Birth:</label>
        <input id = "dob" name="dob" type="text" readonly="readonly"
onblur="(this.type='text')" onfocus="(this.type = 'date')" placeholder="Date
of Birth" value="{{ account['DOB'] }}"">

        <label for="age">Age:</label>
        <input type="number" id="age" name="age" readonly="readonly"
placeholder="Enter Age" value="{{ account['AGE'] }}">

        <label for="avail">Availability:</label>
        <select id="avail" name="avail" value="{{ account['AVAILABILITY']
}}" disabled>
            <option value=None style="color: black; background-color:
#6eff2b">{{ account['AVAILABILITY'] }}</option>
            <option value="Available">Available</option>
            <option value="Not Available">Not Available</option>
        </select>

        <label for="phone">Contact No:</label>
        <input type="text" id="phone" name="phone" readonly="readonly"
value="{{ account['PHONE'] }}" placeholder="Enter Phone Number">
```

```
        <label for="city">City:</label>
        <input type="text" id="city" name="city" readonly="readonly"
value="{{ account['CITY'] }}" placeholder="Enter City">

        <label for="state">State:</label>
        <input type="text" id="state" name="state" readonly="readonly"
value="{{ account['STATE'] }}" placeholder="Enter State">

        <label for="country">Country:</label>
        <input type="text" id="country" name="country"
readonly="readonly" value="{{ account['COUNTRY'] }}"
placeholder="Enter Country">

       </fieldset>

       <fieldset>
        <legend><span class="number">2</span>Health Information</legend>
  <br></br>

        <label for="bloodtype">Select Blood Type:</label>
          <select id="bloodtype" name="bloodtype" disabled>
             <option value=None style="color: black; background-color:
#6eff2b">{{ account['BLOODTYPE'] }}</option>
             <option value="A+">A+</option>
             <option value="A-">A-</option>
             <option value="B+">B+</option>
             <option value="B-">B-</option>
             <option value="O+">O+</option>
             <option value="O-">O-</option>
             <option value="AB+">AB+</option>
             <option value="AB-">AB-</option>
          </select>

        <label for="description">Description:</label>
        <textarea id="description" name="description" readonly="readonly"
placeholder="Enter Description">{{ account['DESCRIPTION'] }}</textarea>
       </fieldset>
       <fieldset>
       <label for="photo">Upload Profile Picture:</label>
       <input type="file" name="photo" id="photo"/>
```

```
      <label for="certificate">Upload Covid Certificate:</label>
      <input type="file" name="certificate" id="certificate"/>

      </fieldset>
      <button type="submit">Submit</button>
    </form>
  {% endblock %}
 </body>
</html>
```

7.2 Feature 2

TICKET VERIFICATION AND TRAIN TRACKING

DESCRIPTION:

In this we have included a feature to verify the ticket by using the QR code scanner at TTE side and retrieve the passenger details from the Cloudant database to check the details of the passenger . If it is a valid QR code then the ticket is verified. For an invalid QR code the pop up message is displayed " INVALID TICKET" .
For Train tracking the current location status of the train is displayed in the webpage . The users can easily know about the arrival and departure of the train .

PYTHON CODE :

```
from flask import *
import ibm_db
from sendgridmail import sendmail
import os
from dotenv import load_dotenv

load_dotenv()

app = Flask(__name__)

try:
  conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=54a2f15b-5c0f-46df-8954-
7e38e612c2bd.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32733;SECURITY=SSL;S
SLServerCertificate=DigiCertGlobalRootCA.crt;PROTOCOL=TCPIP;UID=ssz03200;PWD=ROE
cnQjP3TOa1Yxw;", "", "")
  print("success")
except:
  print(ibm_db.conn_errormsg())

# conn = ibm_db2.connect(os.getenv('DB_KEY'),'','')


app.app_context().push()
app.config["TEMPLATES_AUTO_RELOAD"] = True
app.config['SECRET_KEY'] = 'AJDJRJS24$($(#$$33--'

@app.route("/signup")
def signup():
    return render_template("signup.html")

@app.route("/")
def login():
    return render_template("login.html")

# Login
@app.route("/loginmethod", methods = ['GET'])
def loginmethod():
    global userid
    msg = ''

    if request.method == 'GET':
        uname = request.args.get("uname")
        psw = request.args.get("psw")

        sql = "SELECT * FROM accounts WHERE uname =? AND psw=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, uname)
        ibm_db.bind_param(stmt, 2, psw)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)

        if account:
            session['loggedin'] = True
            session['id'] = account['UNAME']
            userid = account['UNAME']
            session['username'] = account['UNAME']
            return redirect(url_for("about"))
        else:
            msg = 'Incorrect Username and Password'
            flash(msg)
            return redirect(url_for("login"))

# Signup
@app.route("/signupmethod", methods = ['POST'])
def signupmethod():
    msg = ''
    if request.method == 'POST':
        uname = request.form['uname']
        email = request.form['email']
        name = request.form['name']
        dob = request.form['dob']
        psw = request.form['psw']
        con_psw = request.form['con_psw']
        sql = "SELECT * FROM accounts WHERE uname =?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, uname)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)

        if account:
            msg = 'Account already exists !'
            flash(msg)
            return redirect(url_for("signup"))
        elif psw != con_psw:
            msg = "Password and Confirm Password do not match."
            flash(msg)
            return redirect(url_for("signup"))
        else:
            insert_sql = "INSERT INTO accounts VALUES (?, ?, ?, ?, ?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, name)
            ibm_db.bind_param(prep_stmt, 2, email)
            ibm_db.bind_param(prep_stmt, 3, dob)
            ibm_db.bind_param(prep_stmt, 4, uname)
            ibm_db.bind_param(prep_stmt, 5, psw)
            ibm_db.execute(prep_stmt)

            insert_donor = "INSERT INTO deonor(Name,Username,Email,DOB,Availability) VALUES
(?, ?, ?, ?, ?)"
            prep_stmt = ibm_db.prepare(conn, insert_donor)
            ibm_db.bind_param(prep_stmt, 1, name)
            ibm_db.bind_param(prep_stmt, 2, uname)
            ibm_db.bind_param(prep_stmt, 3, email)
            ibm_db.bind_param(prep_stmt, 4, dob)
            ibm_db.bind_param(prep_stmt, 5, "Not Available")
            ibm_db.execute(prep_stmt)

            sendmail(email, 'Plasma deonor App login',name, 'You are successfully Registered!')

            return redirect(url_for("login"))

    elif request.method == 'POST':
        msg = 'Please fill out the form !'
        flash(msg)
        return redirect(url_for("signup"))

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

@app.route('/requester')
def requester():
    if session['loggedin'] == True:
        return render_template('home.html')
    else:
        msg = 'Please login!'
        return render_template('login.html', msg = msg)

@app.route('/dash')
def dash():
    if session['loggedin'] == True:
        sql = "SELECT COUNT(*), (SELECT COUNT(*) FROM DEONOR WHERE BloodType= 'O
Positive'), (SELECT COUNT(*) FROM DEONOR WHERE BloodType='A Positive'), (SELECT
COUNT(*) FROM DEONOR WHERE BloodType='B Positive'), (SELECT COUNT(*) FROM
DEONOR WHERE BloodType='AB Positive'), (SELECT COUNT(*) FROM DEONOR WHERE
BloodType='O Negative'), (SELECT COUNT(*) FROM DEONOR WHERE BloodType='A
Negative'), (SELECT COUNT(*) FROM DEONOR WHERE BloodType='B Negative'), (SELECT
COUNT(*) FROM DEONOR WHERE BloodType='AB Negative') FROM dEonor"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        return render_template('dashboard.html',b=account)
    else:
        msg = 'Please login!'
        return render_template('login.html', msg = msg)

@app.route('/requested',methods=['POST'])
def requested():
    bloodgrp = request.form['bloodgrp']
    address = request.form['address']
    name = request.form['name']
    email = request.form['email']
    phone = request.form['phone']
    insert_sql = "INSERT INTO requested VALUES (?, ?, ?, ?, ?, ?)"
    prep_stmt = ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prep_stmt, 1, session["username"])
    ibm_db.bind_param(prep_stmt, 2, bloodgrp)
    ibm_db.bind_param(prep_stmt, 3, address)
    ibm_db.bind_param(prep_stmt, 4, name)
    ibm_db.bind_param(prep_stmt, 5, email)
    ibm_db.bind_param(prep_stmt, 6, phone)
    ibm_db.execute(prep_stmt)
    sendmail(email,'Plasma donor App plasma request', name ,'Your request for plasma is
received.')

    # send_sql = "SELECT EMAIL FROM donor where BloodTypeTYPE = ?"
    # prep_stmt = ibm_db.prepare(conn, send_sql)
    # ibm_db.bind_param(prep_stmt, 1, bloodgrp)
    # ibm_db.execute(prep_stmt)
    # send = ibm_db.fetch_assoc(prep_stmt)
    # print(send)
    # for i in send:
```

```
    #     sendmail(i.strip(), 'Plasma donor App plasma request', name, 'A Donee has requested for
Blood.')

    return render_template('home.html', pred="Your request is sent to the concerned people.")

@app.route('/about')
def about():
    print(session["username"], session['id'])

    display_sql = "SELECT * FROM deonor WHERE username = ?"
    prep_stmt = ibm_db.prepare(conn, display_sql)
    ibm_db.bind_param(prep_stmt, 1, session['id'])
    ibm_db.execute(prep_stmt)
    account = ibm_db.fetch_assoc(prep_stmt)
    print(account)
    deonors = {}
    for values in account:
        if type(account[values]) == str:
            deonors[values] = account[values].strip()
        else:
            deonors[values] = account[values]

    print(deonors)
    return render_template("about.html", account = deonors)

@app.route('/details', methods = ['POST'])
def details():
    if request.method == 'POST':
        uname = request.form['uname']
        email = request.form['email']
        name = request.form['name']
        dob = request.form['dob']
        age = request.form['age']
        phone = request.form['phone']
        city = request.form['city']
        state = request.form['state']
        country = request.form['country']
        bloodtype = request.form['bloodtype']
        description = request.form['description']
        avail = request.form['avail']

        sql = "SELECT * FROM deonor WHERE Username =?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, uname)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)

        if account:
            update_sql = "UPDATE deonor set Name=?, Username=?, Email=?, DOB=?, Age=?,
Phone=?, City=?, State=?, Country=?, BloodType=?,Description=?,Availability=? where Username
= ?"
```

```
            prep_stmt = ibm_db.prepare(conn, update_sql)
            ibm_db.bind_param(prep_stmt, 1, name)
            ibm_db.bind_param(prep_stmt, 2, uname)
            ibm_db.bind_param(prep_stmt, 3, email)
            ibm_db.bind_param(prep_stmt, 4, dob)
            ibm_db.bind_param(prep_stmt, 5, age)
            ibm_db.bind_param(prep_stmt, 6, phone)
            ibm_db.bind_param(prep_stmt, 7, city)
            ibm_db.bind_param(prep_stmt, 8, state)
            ibm_db.bind_param(prep_stmt, 9, country)
            ibm_db.bind_param(prep_stmt, 10, bloodtype)
            ibm_db.bind_param(prep_stmt, 11, description)
            ibm_db.bind_param(prep_stmt, 12, avail)
            ibm_db.bind_param(prep_stmt, 13, uname)
            ibm_db.execute(prep_stmt)
            print("Update Success")
            return redirect(url_for("about"))
            # return render_template('about.html', pred="Your details updated")

        else:
            insert_sql = "INSERT INTO deonor VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, name)
            ibm_db.bind_param(prep_stmt, 2, uname)
            ibm_db.bind_param(prep_stmt, 3, email)
            ibm_db.bind_param(prep_stmt, 4, dob)
            ibm_db.bind_param(prep_stmt, 5, age)
            ibm_db.bind_param(prep_stmt, 6, phone)
            ibm_db.bind_param(prep_stmt, 7, city)
            ibm_db.bind_param(prep_stmt, 8, state)
            ibm_db.bind_param(prep_stmt, 9, country)
            ibm_db.bind_param(prep_stmt, 10, bloodtype)
            ibm_db.bind_param(prep_stmt, 11, description)
            ibm_db.bind_param(prep_stmt, 12, avail)
            ibm_db.bind_param(prep_stmt, 13, (str(False)))

            ibm_db.execute(prep_stmt)
            print("Sucess")
            return redirect(url_for("about"))

@app.route('/logout')

def logout():
    session.pop('loggedin', None)
    session.pop('id', None)
    session.pop('username', None)
    return render_template('login.html')

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

SENDGRID MESSAGE:

```
import os
from dotenv import load_dotenv

load_dotenv()
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail

def sendmail(usermail,subject,name,content):
    message =
Mail(from_email='litchypeddie@gmail.com',to_emails=usermail,subject=subject,html_content='<h
6>Hello {}, </h6><br/><strong> {} </strong><br/><p>Best Wishes,</p><p>Team
Plasma</p>'.format(name,content))
    try:
        sg = SendGridAPIClient(os.getenv('API_KEY'))
        response = sg.send(message)
        print(response.status_code)
        print(response.body)
        print(response.headers)
    except Exception as e:
        print(e.message)
```

SENDGRID INTEGARATION WITH PYTHON:

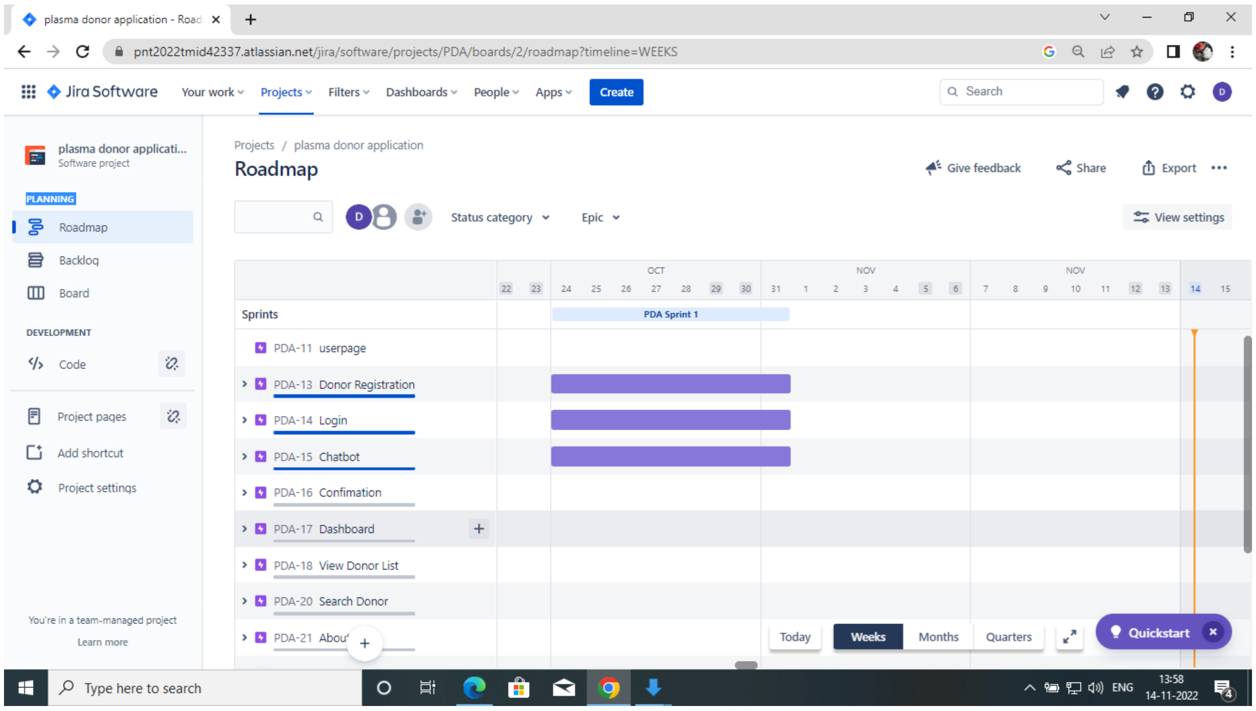
```
import os
import json

from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import *


# NOTE: you will need move this file to the root
# directory of this project to execute properly.


def build_hello_email():
    ## Send a Single Email to a Single Recipient

    message = Mail(from_email=From('from@example.com', 'Example From Name'),
            to_emails=To('to@example.com', 'Example To Name'),
            subject=Subject('Sending with SendGrid is Fun'),
            plain_text_content=PlainTextContent('and easy to do anywhere, even with Python'),
            html_content=HtmlContent('<strong>and easy to do anywhere, even with
Python</strong>'))

    try:
        print(json.dumps(message.get(), sort_keys=True, indent=4))
        return message.get()

    except SendGridException as e:
        print(e.message)

    mock_personalization = Personalization()
    personalization_dict = get_mock_personalization_dict()

    for cc_addr in personalization_dict['cc_list']:
        mock_personalization.add_to(cc_addr)

    for bcc_addr in personalization_dict['bcc_list']:
        mock_personalization.add_bcc(bcc_addr)

    for header in personalization_dict['headers']:
        mock_personalization.add_header(header)

    for substitution in personalization_dict['substitutions']:
        mock_personalization.add_substitution(substitution)

    for arg in personalization_dict['custom_args']:
        mock_personalization.add_custom_arg(arg)

    mock_personalization.subject = personalization_dict['subject']
    mock_personalization.send_at = personalization_dict['send_at']

    message.add_personalization(mock_personalization)

    return message

def get_mock_personalization_dict():
    """Get a dict of personalization mock."""
    mock_pers = dict()

    mock_pers['to_list'] = [To("test1@example.com",
                              "Example User"),
                           To("test2@example.com",
                              "Example User")]

    mock_pers['cc_list'] = [To("test3@example.com",
                              "Example User"),
                           To("test4@example.com",
                              "Example User")]

    mock_pers['bcc_list'] = [To("test5@example.com"),
                            To("test6@example.com")]

    mock_pers['subject'] = ("Hello World from the Personalized "
                           "SendGrid Python Library")

    mock_pers['headers'] = [Header("X-Test", "test"),
                           Header("X-Mock", "true")]

    mock_pers['substitutions'] = [Substitution("%name%", "Example User"),
                        Substitution("%city%", "Denver")]

    mock_pers['custom_args'] = [CustomArg("user_id", "343"),
                        CustomArg("type", "marketing")]

    mock_pers['send_at'] = 1443636843
    return mock_pers

def build_multiple_emails_personalized():
    # Note that the domain for all From email addresses must match

    message = Mail(from_email=From('from@example.com', 'Example From Name'),
            subject=Subject('Sending with SendGrid is Fun'),
            plain_text_content=PlainTextContent('and easy to do anywhere, even with Python'),
            html_content=HtmlContent('<strong>and easy to do anywhere, even with
Python</strong>'))

    mock_personalization = Personalization()
    mock_personalization.add_to(To('test@example.com', 'Example User 1'))
    mock_personalization.add_cc(Cc('test1@example.com', 'Example User 2'))
    message.add_personalization(mock_personalization)

    mock_personalization_2 = Personalization()
    mock_personalization_2.add_to(To('test2@example.com', 'Example User 3'))
    mock_personalization_2.set_from(From('from@example.com', 'Example From Name 2'))
    mock_personalization_2.add_bcc(Bcc('test3@example.com', 'Example User 4'))
    message.add_personalization(mock_personalization_2)

    try:
        print(json.dumps(message.get(), sort_keys=True, indent=4))
        return message.get()

    except SendGridException as e:
        print(e.message)

    return message

def build_attachment1():
    """Build attachment mock. Make sure your content is base64 encoded before passing into
attachment.content.
    Another example: https://github.com/sendgrid/sendgrid-
python/blob/HEAD/use_cases/attachment.md"""

    attachment = Attachment()
    attachment.file_content = ("TG9yZW0gaXBzdW0gZG9sb3Igc2l0IGFtZXQsIGNvbnNl"
                    "Y3RldHVyIGFkaXBpc2NpbmcgZWxpdC4gQ3JhcyBwdW12")
    attachment.file_type = "application/pdf"
    attachment.file_name = "balance_001.pdf"
    attachment.disposition = "attachment"
    attachment.content_id = "Balance Sheet"
    return attachment
```

```
def build_attachment2():
    """Build attachment mock."""
    attachment = Attachment()
    attachment.file_content = "BwdW"
    attachment.file_type = "image/png"
    attachment.file_name = "banner.png"
    attachment.disposition = "inline"
    attachment.content_id = "Banner"
    return attachment

def build_kitchen_sink():
    """All settings set"""
    from sendgrid.helpers.mail import (
        Mail, From, To, Cc, Bcc, Subject, PlainTextContent,
        HtmlContent, SendGridException, Substitution,
        Header, CustomArg, SendAt, Content, MimeType, Attachment,
        FileName, FileContent, FileType, Disposition, ContentId,
        TemplateId, Section, ReplyTo, Category, BatchId, Asm,
        GroupId, GroupsToDisplay, IpPoolName, MailSettings,
        BccSettings, BccSettingsEmail, BypassListManagement,
        FooterSettings, FooterText, FooterHtml, SandBoxMode,
        SpamCheck, SpamThreshold, SpamUrl, TrackingSettings,
        ClickTracking, SubscriptionTracking, SubscriptionText,
        SubscriptionHtml, SubscriptionSubstitutionTag,
        OpenTracking, OpenTrackingSubstitutionTag, Ganalytics,
        UtmSource, UtmMedium, UtmTerm, UtmContent, UtmCampaign)
    import time
    import datetime

    message = Mail()

    # Define Personalizations

    message.to = To('test1@sendgrid.com', 'Example User1', p=0)
    message.to = [
        To('test2@sendgrid.com', 'Example User2', p=0),
        To('test3@sendgrid.com', 'Example User3', p=0)
    ]

    message.cc = Cc('test4@example.com', 'Example User4', p=0)
    message.cc = [
        Cc('test5@example.com', 'Example User5', p=0),
        Cc('test6@example.com', 'Example User6', p=0)
    ]

    message.bcc = Bcc('test7@example.com', 'Example User7', p=0)
    message.bcc = [
        Bcc('test8@example.com', 'Example User8', p=0),
        Bcc('test9@example.com', 'Example User9', p=0)
    ]

    message.subject = Subject('Sending with SendGrid is Fun 0', p=0)

    message.header = Header('X-Test1', 'Test1', p=0)
    message.header = Header('X-Test2', 'Test2', p=0)
    message.header = [
        Header('X-Test3', 'Test3', p=0),
        Header('X-Test4', 'Test4', p=0)
    ]

    message.substitution = Substitution('%name1%', 'Example Name 1', p=0)
    message.substitution = Substitution('%city1%', 'Example City 1', p=0)
    message.substitution = [
        Substitution('%name2%', 'Example Name 2', p=0),
        Substitution('%city2%', 'Example City 2', p=0)
    ]

    message.custom_arg = CustomArg('marketing1', 'true', p=0)
    message.custom_arg = CustomArg('transactional1', 'false', p=0)
    message.custom_arg = [
        CustomArg('marketing2', 'false', p=0),
        CustomArg('transactional2', 'true', p=0)
    ]

    message.send_at = SendAt(1461775051, p=0)

    message.to = To('test10@example.com', 'Example User10', p=1)
    message.to = [
        To('test11@example.com', 'Example User11', p=1),
        To('test12@example.com', 'Example User12', p=1)
    ]

    message.cc = Cc('test13@example.com', 'Example User13', p=1)
    message.cc = [
        Cc('test14@example.com', 'Example User14', p=1),
        Cc('test15@example.com', 'Example User15', p=1)
    ]

    message.bcc = Bcc('test16@example.com', 'Example User16', p=1)
    message.bcc = [
        Bcc('test17@example.com', 'Example User17', p=1),
        Bcc('test18@example.com', 'Example User18', p=1)
    ]

    message.header = Header('X-Test5', 'Test5', p=1)
    message.header = Header('X-Test6', 'Test6', p=1)
    message.header = [
        Header('X-Test7', 'Test7', p=1),
        Header('X-Test8', 'Test8', p=1)
    ]
```

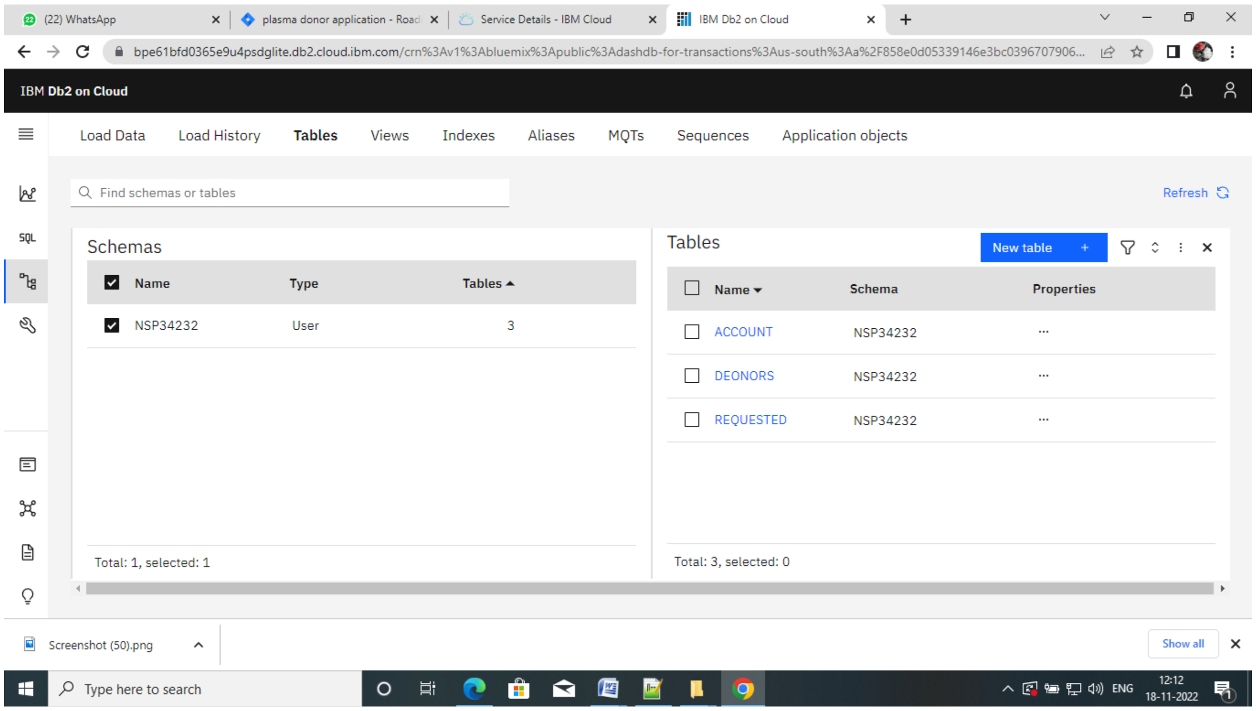
```
    message.substitution = Substitution('%name3%', 'Example Name 3', p=1)
    message.substitution = Substitution('%city3%', 'Example City 3', p=1)
    message.substitution = [
        Substitution('%name4%', 'Example Name 4', p=1),
        Substitution('%city4%', 'Example City 4', p=1)
    ]

    message.custom_arg = CustomArg('marketing3', 'true', p=1)
    message.custom_arg = CustomArg('transactional3', 'false', p=1)
    message.custom_arg = [
        CustomArg('marketing4', 'false', p=1),
        CustomArg('transactional4', 'true', p=1)
    ]

    message.send_at = SendAt(1461775052, p=1)

    message.subject = Subject('Sending with SendGrid is Fun 1', p=1)

    # The values below this comment are global to entire message

    message.from_email = From('help@twilio.com', 'Twilio SendGrid')

    message.reply_to = ReplyTo('help_reply@twilio.com', 'Twilio SendGrid Reply')

    message.subject = Subject('Sending with SendGrid is Fun 2')

    message.content = Content(MimeType.text, 'and easy to do anywhere, even with Python')
    message.content = Content(MimeType.html, '<strong>and easy to do anywhere, even with
Python</strong>')
    message.content = [
        Content('text/calendar', 'Party Time!!'),
        Content('text/custom', 'Party Time 2!!')
    ]

    message.attachment = Attachment(FileContent('base64 encoded content 1'),
                                    FileName('balance_001.pdf'),
                                    FileType('application/pdf'),
                                    Disposition('attachment'),
                                    ContentId('Content ID 1'))
    message.attachment = [
        Attachment(FileContent('base64 encoded content 2'),
                FileName('banner.png'),
                FileType('image/png'),
                Disposition('inline'),
                ContentId('Content ID 2')),
        Attachment(FileContent('base64 encoded content 3'),
                FileName('banner2.png'),
                FileType('image/png'),
                Disposition('inline'),
                ContentId('Content ID 3'))
    ]

    message.template_id = TemplateId('13b8f94f-bcae-4ec6-b752-70d6cb59f932')

    message.section = Section('%section1%', 'Substitution for Section 1 Tag')
    message.section = [
        Section('%section2%', 'Substitution for Section 2 Tag'),
        Section('%section3%', 'Substitution for Section 3 Tag')
    ]

    message.header = Header('X-Test9', 'Test9')
    message.header = Header('X-Test10', 'Test10')
    message.header = [
        Header('X-Test11', 'Test11'),
        Header('X-Test12', 'Test12')
    ]

    message.category = Category('Category 1')
    message.category = Category('Category 2')
    message.category = [
        Category('Category 1'),
        Category('Category 2')
    ]

    message.custom_arg = CustomArg('marketing5', 'false')
    message.custom_arg = CustomArg('transactional5', 'true')
    message.custom_arg = [
        CustomArg('marketing6', 'true'),
        CustomArg('transactional6', 'false')
    ]

    message.send_at = SendAt(1461775053)

    message.batch_id = BatchId("HkJ5yLYULb7Rj8GKSx7u025ouWVlMgAi")

    message.asm = Asm(GroupId(1), GroupsToDisplay([1,2,3,4]))

    message.ip_pool_name = IpPoolName("IP Pool Name")

    mail_settings = MailSettings()
    mail_settings.bcc_settings = BccSettings(False, BccSettingsTo("bcc@twilio.com"))
    mail_settings.bypass_list_management = BypassListManagement(False)
    mail_settings.footer_settings = FooterSettings(True, FooterText("w00t"),
FooterHtml("<string>w00t!<strong>"))
    mail_settings.sandbox_mode = SandBoxMode(True)
    mail_settings.spam_check = SpamCheck(True, SpamThreshold(5),
SpamUrl("https://example.com"))
    message.mail_settings = mail_settings

    tracking_settings = TrackingSettings()
    tracking_settings.click_tracking = ClickTracking(True, False)
    tracking_settings.open_tracking = OpenTracking(True,
OpenTrackingSubstitutionTag("open_tracking"))
    tracking_settings.subscription_tracking = SubscriptionTracking(
        True,
        SubscriptionText("Goodbye"),
        SubscriptionHtml("<strong>Goodbye!</strong>"),
        SubscriptionSubstitutionTag("unsubscribe"))
    tracking_settings.ganalytics = Ganalytics(
        True,
        UtmSource("utm_source"),
        UtmMedium("utm_medium"),
        UtmTerm("utm_term"),
        UtmContent("utm_content"),
        UtmCampaign("utm_campaign"))
    message.tracking_settings = tracking_settings

    return message

def send_multiple_emails_personalized():
    # Assumes you set your environment variable:
    # https://github.com/sendgrid/sendgrid-
python/blob/HEAD/TROUBLESHOOTING.md#environment-variables-and-your-sendgrid-api-key
    message = build_multiple_emails_personalized()
    sendgrid_client = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
    response = sendgrid_client.send(message=message)
    print(response.status_code)
    print(response.body)
    print(response.headers)

def send_hello_email():
    # Assumes you set your environment variable:
    # https://github.com/sendgrid/sendgrid-
python/blob/HEAD/TROUBLESHOOTING.md#environment-variables-and-your-sendgrid-api-key
    message = build_hello_email()
    sendgrid_client = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
    response = sendgrid_client.send(message=message)
    print(response.status_code)
    print(response.body)
    print(response.headers)


def send_kitchen_sink():
    # Assumes you set your environment variable:
    # https://github.com/sendgrid/sendgrid-
python/blob/HEAD/TROUBLESHOOTING.md#environment-variables-and-your-sendgrid-api-key
    message = build_kitchen_sink()
    sendgrid_client = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
    response = sendgrid_client.send(message=message)
    print(response.status_code)
    print(response.body)
    print(response.headers)
```

```
## this will actually send an email
# send_hello_email()

## this will send multiple emails
# send_multiple_emails_personalized()

## this will only send an email if you set SandBox Mode to False
# send_kitchen_sink()
```

REPORT FROM JIRA :



Database Schema:


IBM Db2 on Cloud

Load Data | Load History | Tables | Views | Indexes | Aliases | MQTs | Sequences | Application objects

Schemas

Name	Type	Tables
NSP34232	User	3

Total: 1, selected: 1

Tables

Name	Schema	Properties
ACCOUNT	NSP34232	···
DEONORS	NSP34232	···
REQUESTED	NSP34232	···

Total: 3, selected: 0

TESTING:

TEST CASE

Authentication Module

• Sign Up New user or donor can create an account to use in the blood/plasma donor application and create a password for account verification and create an identity.

• Sign In Donor Sign In to the account for viewing or editing location details and any other personal information.

• Account Verification If donor changes their password or if they forget the password then we have to verify their account using mail verification.

Service Provider Module

• Add New Donor User can be able to register to add donor details.

• List All Donor User can be able to view all Donor who all use our Plasma Donor Application

. • Edit Customer Plan Details User can be able to edit the existing Donor details as the Donor wish.

RESULTS

PerformanceMetrices

1. **Formal code metrics -** Such as Lines of Code (LOC), code complexity, Instruction Path Length, etc. In modern development environments, these are considered less useful.

2. **Developer productivity metrics**—Such as active days, assignment scope, efficiency andcode churn. These metrics can help you understand how much time and work developers are investing in a software project.

1. **Agile process metrics**—Such as lead time, cycle time and velocity. They measure the progress of a dev team in producing working, shipping-quality software features.

2. **Operational metrics**—Such as Mean Time Between Failures (MTBF) and Mean Time to Recover (MTTR). This checks how software is running in production and how effective operations staff are atMainta ining it.

   a. **Test metrics**—Such as code coverage,percent of automatedtests, and defectsin production. This measures how comprehensively a system is tested, which should be correlated with software quality.

   b. **Customer satisfaction**—Such as Net Promoter Score (NPS), Customer Effort Score (CES) and Customer Satisfaction Score (CSAT). The ultimate measurement of how customers experience the software and their interaction with the software vendor.

ADVANTAGES&DISADVANTA

GESAdvantages:

1. It is a user-friendly web application to help people who are affected by COVID19by donating plasmafrom patients who have recovered and help them recover faster.

1. The traditional methods of finding plasma,sometimes may not be available in this case the donor can use this website to donate plasma can simply upload their covid19traced certificate and can donate the plasma to the blood bank, the blood bank can apply for the donor and once the donor has accepted the request, the blood bank can add the units they

   need and the hospital can also send the request to the blood bank that urgently needs the plasmafor the patient and can take the plasma from the blood bank

2. It is a useful website to find compatible plasma donors who can receive plasma request posts in their local area. Clinics can use this web application to maintain theplasma donation activity.

   a. It is reliable and safe application and keep trackof total plasma donations.

Disadvantages:

   b. Absence and lack of integration between plasma centers

   c. The app user will not be able to insert or view detailsif the server goes down.

CONCLUSION:

Today the world has become a global platform where every thing is online.There are so many web based solutions provided in the market for the comfort of the people.But without blood human being is non living, by providing the web solution of plasma management information system is just one more step in order to serve the mankind. Plasma donor application provides a reliable platform to connect local plasma donors with patients. This app creates a communication channel through authenticated clinics whenever a patient needs plasma donation. It is a useful tool to find compatible plasma donors who can receive plasma request posts in their local area. Clinics can use this web application to maintain the plasma donation activity.Plasma donation is the one way to lead a person a healthy life.This is because during the plasma donation our body will be replaced with new blood cell which have a better protein, from this website the plasmadonors can get an awareness of importance of plasma donation and the plasma donation will increase.It presents a high-end system to bridge the gap between the plasma donors and the people in need for plasma.

Plasma donor application aims to act as an important role in saving life of human beings and reduce the panic created in emergency situations. It proposes a plasma donationapplication which can be used by laboratories, clinics, hospitals, or anyone who is in need of. It is developed such that users can view the information about registered plasma donors and plasma banks such as name, address, and phone number along with their details of blood group and other medical information. Not only does it connect users to different donors but also to plasma banks.This system is developedin order to enhance the management, performance and the quality of services for the management of plasma banks.

FUTURE SCOPE:

Plasma donation application is a software application to build in such a way that it should suits for all type of plasma banks in future. One important future scope is availability of location-based plasma bank details and extraction of location-based donor&#39;s detail, which is very helpful to the acceptant people.The scope of the plasma donationapplication are the management of the availability of donors, hospitals,blood banks to the user or member at any time.This application can be installed in ambulances in order to save time. This work proposes a plasma donationapplication which we believe will bring remarkable change.Plasma donation application is a softwareapplication to buildin such a way that it should suits forall type of plasma banks in future.

One important future scope is availability of location-based plasma bank details and extraction of location-based donor&#39;s detail, which is very helpful to the acceptant people.The scope of the plasma donation application are the management of the availability of donors, hospitals, blood banks to the user or member at any time.This application can be installed in ambulances in order to save time

GITHUB LINK:

IBM-EPBL/IBM-Project-44452-1660724726

VIDEO LINK:

http://youtu.be/Zk9VTMG2YIM