PLASMA DONOR APPLICATION

TEAM ID: PNT2022TMID21915

ANNAMALAI G	**142219205004**
AKASH S	**142219205002**
ALLAN VIJAY M	**142219205003**
HARIHARAN C	**142219205026**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The main goal of our project is to design a user-friendly web application that is like a scientific vehicle from which we can help reduce mortality or help those affected by COVID19 by donating plasma from patients who have recovered without approved antiretroviral therapy planning for a deadly COVID19 infection, plasma therapy is an experimental approach to treat those COVID-positive patients and help them recover faster. Therapy, which is considered reliable and safe. If a particular person has fully recovered from COVID19, they are eligible to donate their plasma.

As we all know, the traditional methods of finding plasma, one has to find out for oneself by looking at hospital records and contacting donors have been recovered, sometimes may not be available at home and move to other places. In this type of scenario, the health of those who are sick becomes disastrous. Therefore, it is not considered a rapid process to find plasma. The main purpose of the proposed system, the donor who wants to donate plasma can simply upload their covid19 traced 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 plasma for the patient and can take the plasma from the blood bank.

1.2 PURPOSE

- Plasma donor application, which puts the power to save a lives in the palm of your hand.
- The main purpose of Plasma Donor App is to create and manage a platform for all donors of the world and remove the recent crisis
- This app provides quick access to donors for an immediate requirement of blood.

- In case of an emergency/surgery, blood procurement is always a major problem which consumes a lot of time. This helps serve the major time-lapse in which a life can be saved.
- Donor who wants to donate blood can register and login to the site and check for any updates on requirements. If they wish to donate they can get into contact with the recipient and proceed.
- Recipient who has a requirement of blood can register for the first time and then log in from the next for any requirement of blood.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING PROBLEM

The existing system provides a web-based application that is acutely useful for emergency services. It will come very useful in urgent times by providing donors information filtered by area and blood type. It allows the donors to communicate with other donors using our Chat-Bot API to inform them about emergencies. The system consists of a well-maintained database to keep all the registered records. It also provides news and information about the on-going coronavirus pandemic. In the end, it provided us the knowledge regarding the latest technology required to build a web-based application. A database has been set up to store historical data related to donation and reception of blood and also to store data from camps so as to take future decisions based on concrete analytical results. Here the server has the duty to fetch the COVID API through the request library and later it gets converted to JavaScript object notation which decreases the weight and makes it simple to work with the data. Here data gets cleaned till a request status of 200. And now cleansing of the data gets provoked where each column gets stored as an array of lists which are later assigned to a dictionary. Finally, data gets returned and rendered to the page. As the process data of country and state gets fetched, cleaned, and stored to uplift the precise result. Data of the world get updates for every 24 hours which includes columns such as Country, country code, total cases, total deaths, total recovered, date, and states of India updates for every 20 min which includes columns such as Province/State, confirmed total deaths, total recovered, active cases.

2.2 REFERENCES

[1] Dennis O’Neil(1999). “Blood Components”.Palomar College. Archived from the original on June 5,2013.

[2] Tuskegee University(May 29, 2013).”Chapter 9 Blood”.tuskegee.edu. Archived from the original on December 28, 2013.

[3] ”Ways to Keep Your Blood Plasma Healthy”. Archived from the original on November 1, 2013.Retrieved November 10, 2011.

[4] Jump up to Maton, Anthea; Jean Hopkins; Charles Wiliam McLaughlin; Susan Johnson; Maryanna Quon Warner LaHart; David LaHart; Jill D. Wright (1993), Human Biology and Health, Englewood Cliffs, NewJersey,USA.

[5] The Physics Factbook - Density of Blood.[6]Basic Biology(2015).”Blood cells”.

[6] Elkassabany NM, Meny GM, Doria RR, Marcucci C (2008). “Green Plasma Revisited”. Anesthesiology108(4);

[7] “19th WHO Model List of Essential Medicines(April 2015)”(PDF). WHO April 2015. RetrievedMay 10,2015.

[8] Tripathi S, Kumar V,Prabhakar A, Joshi S, Agarwal A(2015).”Passive blood plasma separation at themicroscale; a review of design principles and microdevices”. J.Micromech, Microeng 25(8); 083001.

[9] Guo, Weijin; Hansson, Jonas; van der wijngaart, Wouter(2020).”Synthetic Paper Separates Plasma from Whole Blood with Low Protein Loss”.Analytical Chemistry.92(9): 6194-6199.

[10] Mani A, Poornima AP, Gupta D(2019) “Greenish discoloration of plasma: Is it really a matter of concern?”, Asian Journal of Transfusion Science.

[11] Starr, Douglas P.(2000), Blood: An Epic History of Medicine and Commerce. New York:Quill.

[12] “Implementation of Blood Donation Application using android Smartphone” by Monika Mandole, Pradnya Jagtap, Prachi Mhaske, Sonali Vidhate.

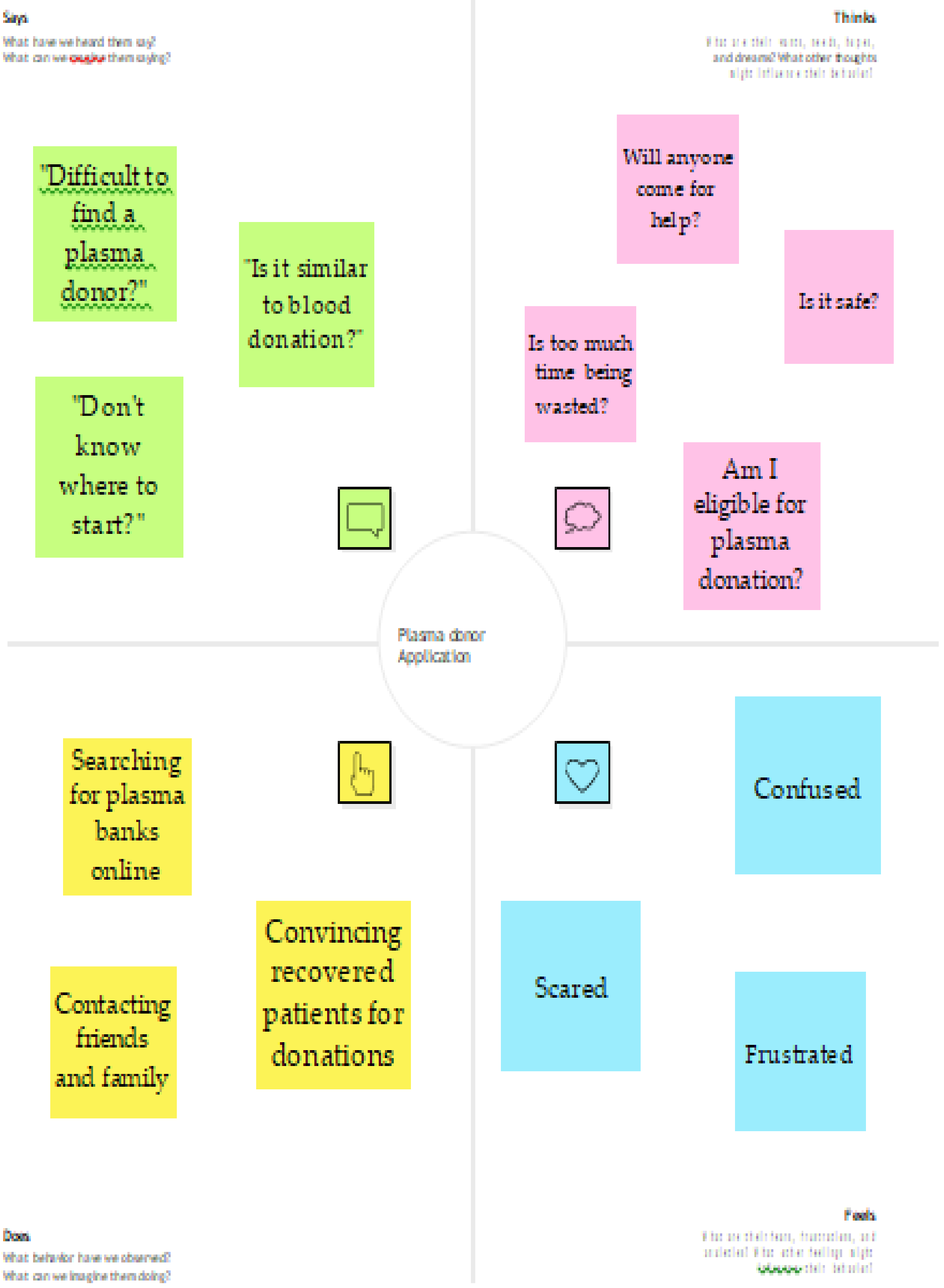
2.3 PROBLEM STATEMENT DEFINITION

People who are affected by COVID are in need of a Plasma Donor. This system aims at connecting the donors and the patients by an online application. By using this application, the users can either raise a request for plasma donation. In case of emergency for a patient, he or she has to contact a compatible donor on their circle, family and friends but it is difficult to find suitable donor. During the COVID-19 crisis, the requirement of plasma became high and the donor count being low. It is very difficult to find the respective blood group donors when someone is in need. We hereby took a step forward to build a system to create a network of people who can help each other in need. We propose an application where the plasma banks can timely update the Plasma Stock Availability (PSA) and register themselves to donor and user can find Plasma Availability nearby him/her. The high – priority time of a plasma requirement, user can quickly check for plasma banks, hospitals or donor as per requirement matching a particular or related and reach out to them through the website. There is also a steady increase in blood donation requests as being noticed in the number of posts on these sites such as Facebook and Twitter seeking blood donors.

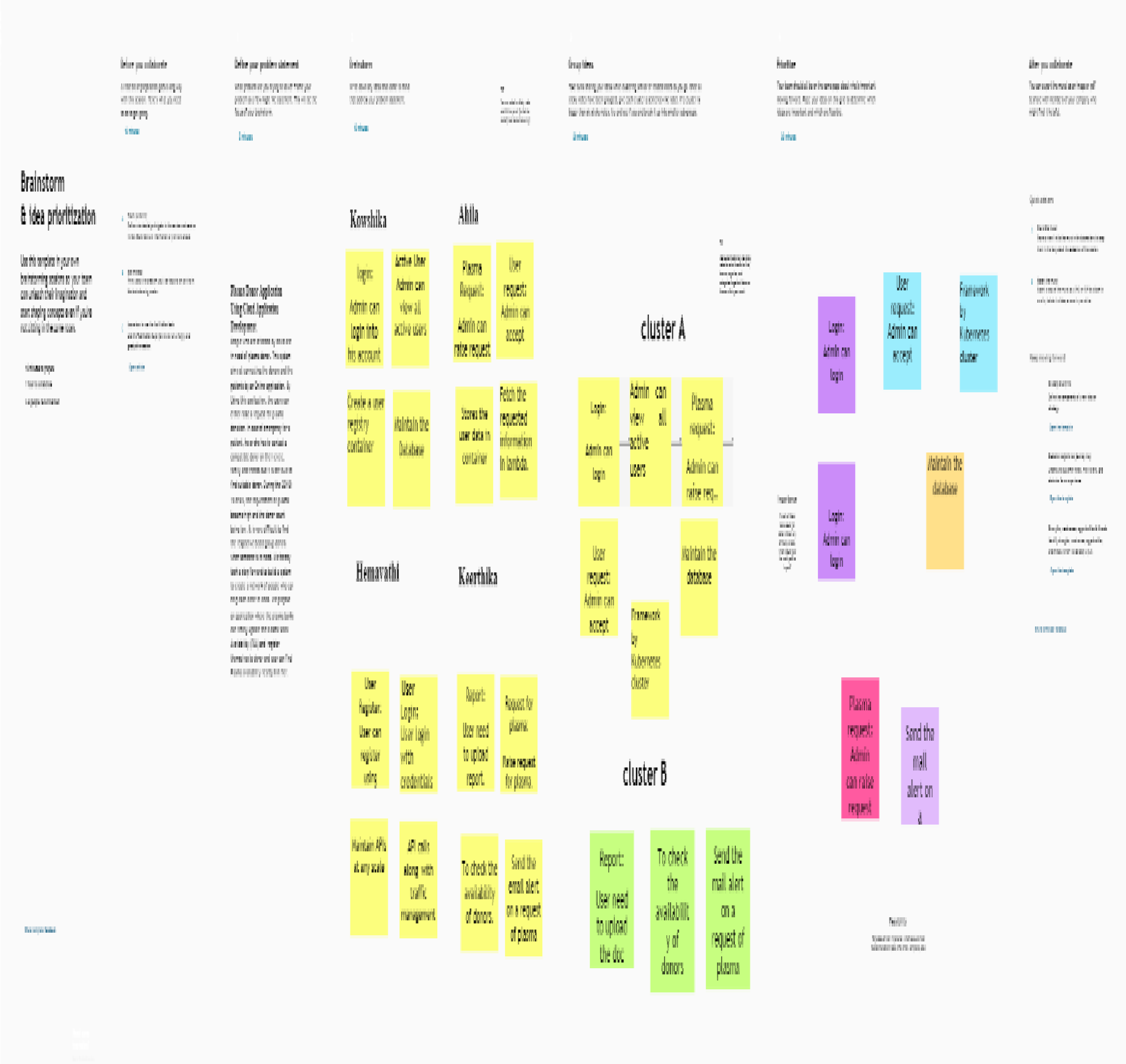
CHAPTER 3

IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS


Says
What have we heard them say?
What can we imagine them saying?
"Difficult to find a plasma donor?"
"Is it similar to blood donation?"
"Don't know where to start?"
Thinks
What are their wants, needs, hopes, and dreams? What other thoughts might influence their behavior?
Will anyone come for help?
Is it safe?
Is too much time being wasted?
Am I eligible for plasma donation?
Plasma donor Application
Searching for plasma banks online
Contacting friends and family
Convincing recovered patients for donations
Confused
Scared
Frustrated
Does
What behavior have we observed?
What can we imagine them doing?
Feels
What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?

3.2 IDEATION & BRAINSTORMING


Before you collaborate
Define your problem statement
Brainstorm
Group ideas
Prioritize
After you collaborate
Brainstorm
& Idea prioritization
Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.
Plasma Donor Application Using Cloud Application Development
Kowshika
Login: Admin can login into his account
Active User Admin can view all active users
Create a user registry container
Maintain the Database
Akila
Plasma Request: Admin can raise request
User request: Admin can accept
Stores the user data in container
Fetch the requested information in lambda.
Hemavathi
User Register: User can register using
User Login: User login with credentials
Maintain APIs at any scale
API calls along with traffic management
Keerthika
Report: User need to upload report.
Request for plasma: Raise request for plasma.
To check the availability of donors.
Send the email alert on a request of plasma
cluster A
Login: Admin can login
Admin can view all active users
Plasma request: Admin can raise req..
User request: Admin can accept
Framework by Kubernetes cluster
Maintain the database
cluster B
Report: User need to upload the doc
To check the availability of donors
Send the mail alert on a request of plasma
Login: Admin can login
User request: Admin can accept
Framework by Kubernetes cluster
Login: Admin can login
Maintain the database
Plasma request: Admin can raise request
Send the mail alert on a

3.3 PROPOSED SOLUTION

S.NO	PARAMETER	DESCRIPTION
1.	Problem Statement (Problem to be solved)	Plasma is used for the treatment of many serious health problems. This is why there are blood drives asking people to donate blood, plasma . Plasma is utilized to treat different irresistible sicknesses and it is one of the most established strategies known as plasma treatment. Coronavirus emergency the necessity for plasma expanded radically as there wereno immunization found to treat the contaminated patients, with plasma therapy the recovery rates where high but the donor count was very low and in such situations it was very important to get the information about the plasma donors. Saving the contributor data and telling about the ongoing givers would be some assistance as it can save time and assist the clients with finding the vital data about the contributors.
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 a request for plasma donation or requirements. The basic solution is to create a centralized system to keep a track on the upcoming as well as past Plasma Donation or therapy Events. The recommendation solution is as follows: Application contains two roles:

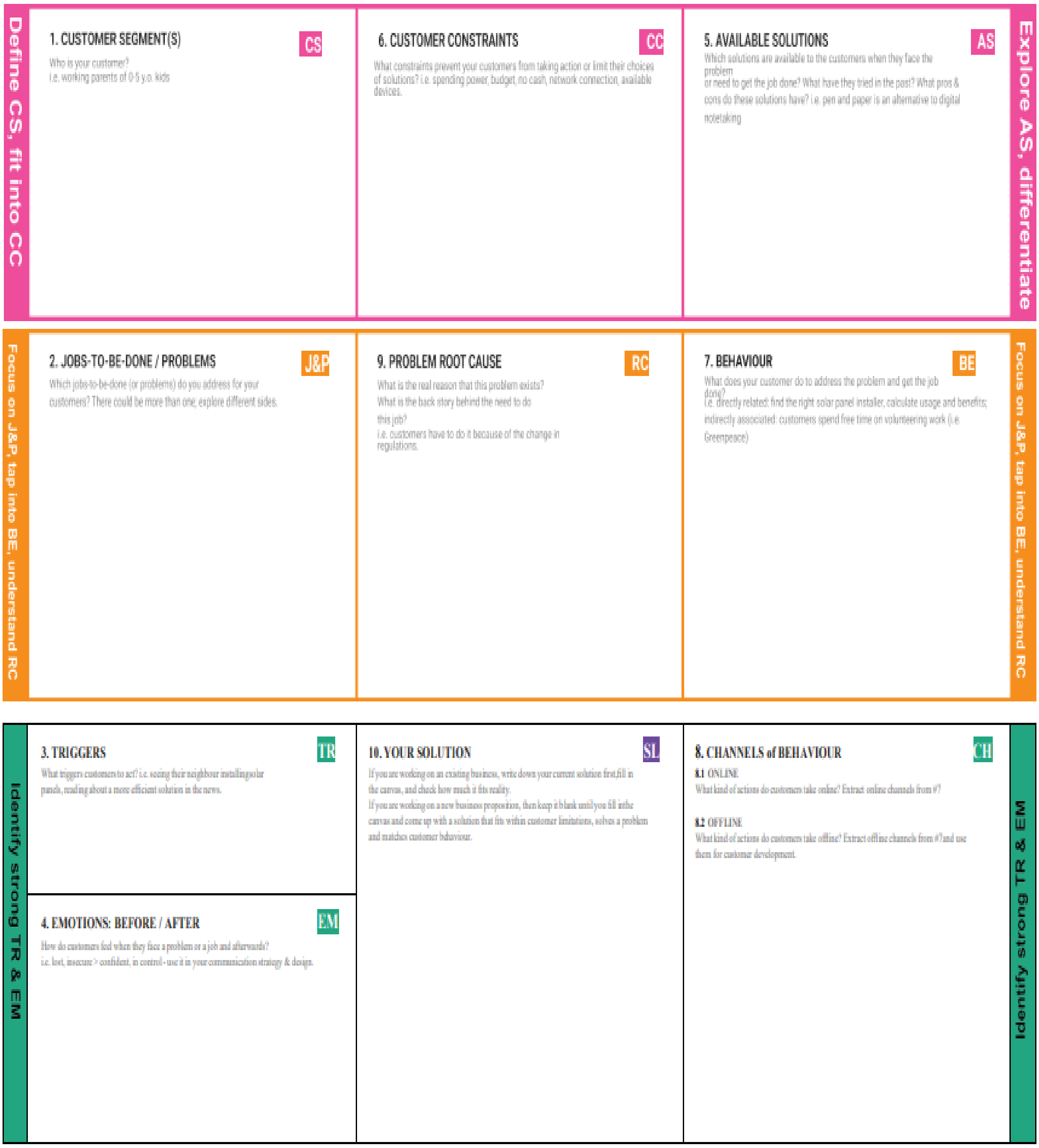
		• Admin • User **Admin**: ➢ Admin can login using their credential. ➢ Admin can edit the request. ➢ Admin can delete the request. Admin can add volunteers. **User:** ❖ If the user wants to donate or receive they have to register with their personal details. After successful registration of user. ❖ A successful registration email is send to the user. ❖ After successful registration

		user will be directed to home page. ❖ They will be asked to press whether they will be donor or receiver. ❖ If the user is donor then he/she will fill the donation interest form which includes their Name, blood group details, location, last time donated date , phone number, email id. ❖ After filling the donation form he/she will be redirected to page in which

		he/she can download the e-certificate.
3.	Novelty / Uniqueness	A User Interface is simple for users to understand. We can use the application anywhere anytime. The user immediately need the plasma for their treatment but the plasma is not available in nearby hospitals, then user can use this application to raise request and directly contact the donor , request them to donate the plasma. Hospitals can also raise request donors for donation. Somebody wants to donate blood and plasma but they don't know the way to donate then they use this application which will simple to use and it will save 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.	Social Impact / Customer Satisfaction	We are living in a modern world and everything can be accessed online. Even though there are many application there is no proper application for plasma donation . Many of them wish to donate blood and plasma but they are unaware about donation and how they can donate. This application provides opportunity to those who want to donate plasma. Donation of plasma are happening in many places many of them come forward to donate but it is not available at right time for use. Sometimes there is a shortage of plasma of particular type. Additional facilities that we need is to access the patients

		information quickly before plasma transfusion. To solve this issue software applications are employedwith Cloud computing and Internet of Things tool which enable features such as information retrieval and continuous data tracking with analytics. This application avoids circulating of wrong information. A single platform for maintaining genuineinformation and increase the trust of participants involved int his activity.
5.	Business Model (Revenue Model)	This application is accessible by everyone. Because of the trouble in finding givers who match a specific blood bunch, this application empowers clients to enlist individuals who 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.
6.	Scalability of the Solution	This application helps users to find plasma donors by sitting in home itself instead of searching donors everywhere. When there is a emergency then plasma request to send to everyone. Once the donor is ready to donate receiver is notified about donation. Receiver can contact the donor. With this app donor can know the eligibility to donate and making it easier to locate suitable donor at right time.

3.4 PROBLEM SOLUTION FIT



Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

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

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.

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

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.

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.

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)

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 installingsolar panels, reading about a more efficient solution in the news.

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.

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 inthe canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.

8. CHANNELS of BEHAVIOUR CH

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

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

Identify strong TR & EM

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

Following are the functional requirements of the proposed solution.

FR NO.	**FUNCTIONAL REQUIREMENT**	**SUB REQUIREMENT (STORY/SUB-TASK)**
FR-1	User Authentication	Receiving the plasma request from Clinic, the blood or plasma stock in the plasma Bank Inventory will be searched to match the requested plasma request. Thus matched plasma units will be sent to the Clinic.
FR-2	Web Service Management Process	Given that software operator has accessed webapplication, then the software operator should be able to register through the web application. The donor software operator must provide first-name, gender, plasma group, location, contact, software operator name and password.
FR-3	Data Management	Automatic generation of components form donor form based on the date of collection, the system automatically derives the date of expiry and disallows

		issue of component if unit has expired
FR-4	Testing	Applying the algorithms on the test data
FR-5	Confirmation	Display whether Plasma is available or not

4.2 NON-FUNCTIONAL REQUIREMENTS

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

FR NO.	NON-FUNCTIONAL REQUIREMENTS	DESCRIPTION
NFR-1	Usability	Usability defines how well the application meets the requirements of the user and consumer by being intuitive, easy to localize and globalize, providing good access for disabled users, and resulting in good overall user experience.
NFR-2	Security	Defines how the system should

		confront the malwares. How well are the system and its data protected against attacks? The plasma donor application Management must be secured with proper user name and passwords.
NFR-3	Reliability	Specifies the probability of the software performing without failure for a specific number of amount of time.
NFR-4	Performance	The Plasma donor application system must perform well in different scenarios. Deals with the measure of the system's response time under different load conditions. Example: The landing page supporting 5,000 users per hour must provide 6 seconds or less response time in a Chrome desktop browser, including the rendering of text and images. over an LTE connection
NFR-5	Availability	The Plasma donor System must be available 24 hours a day with no bandwidth issues. Stands for the system's reliability and accessibility to the user.
NFR-6	Scalability	Assesses the highest workloads under which the system will still meet the

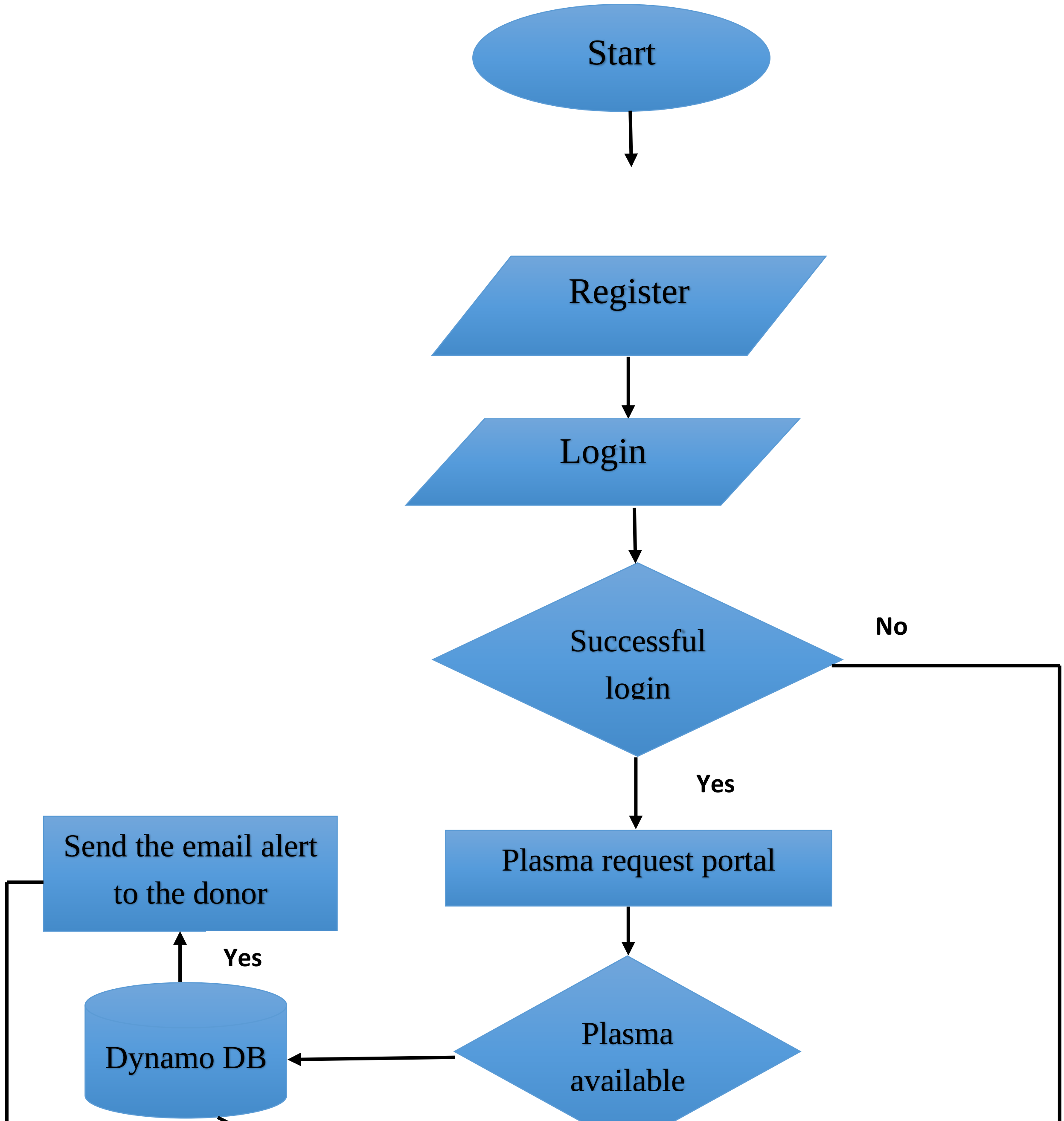
		performance requirements.

CHAPTER-5

PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS

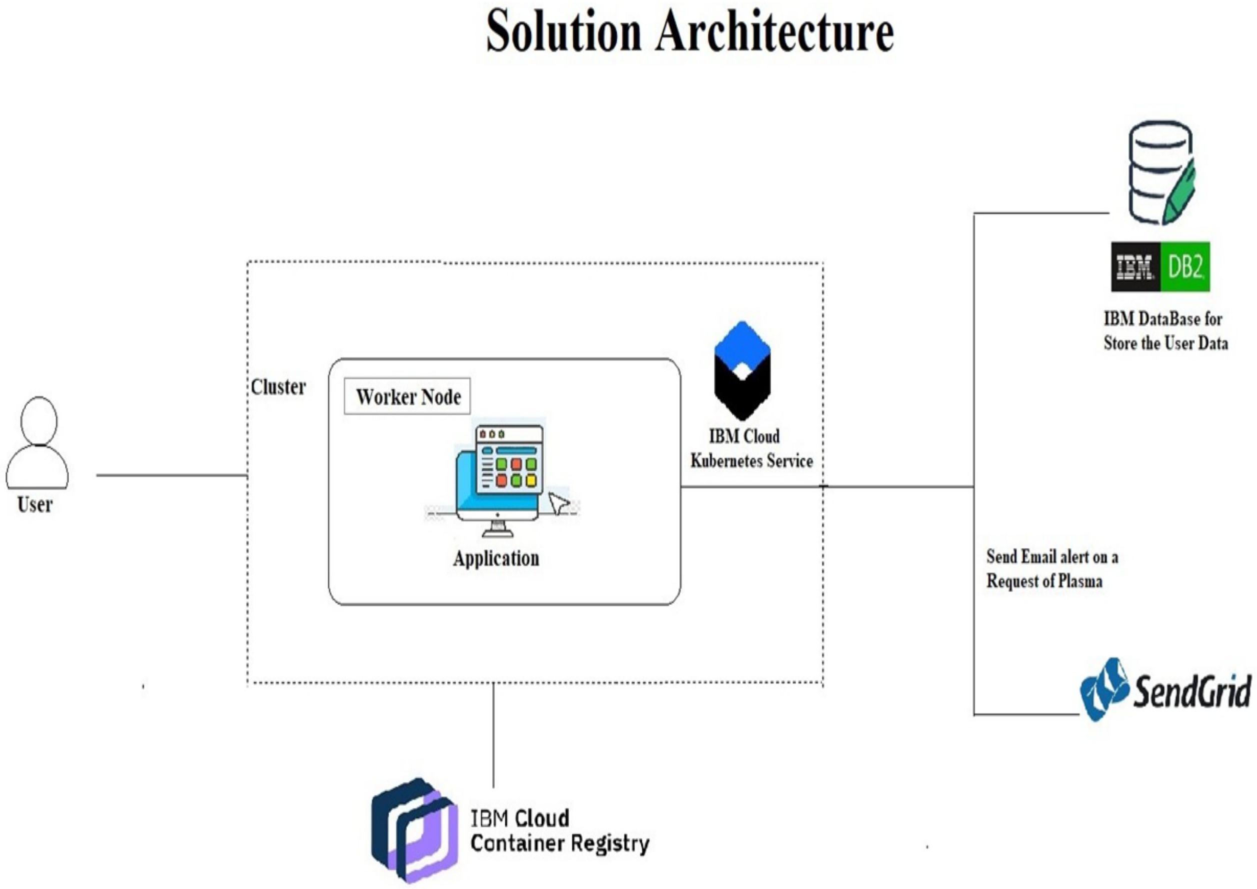
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



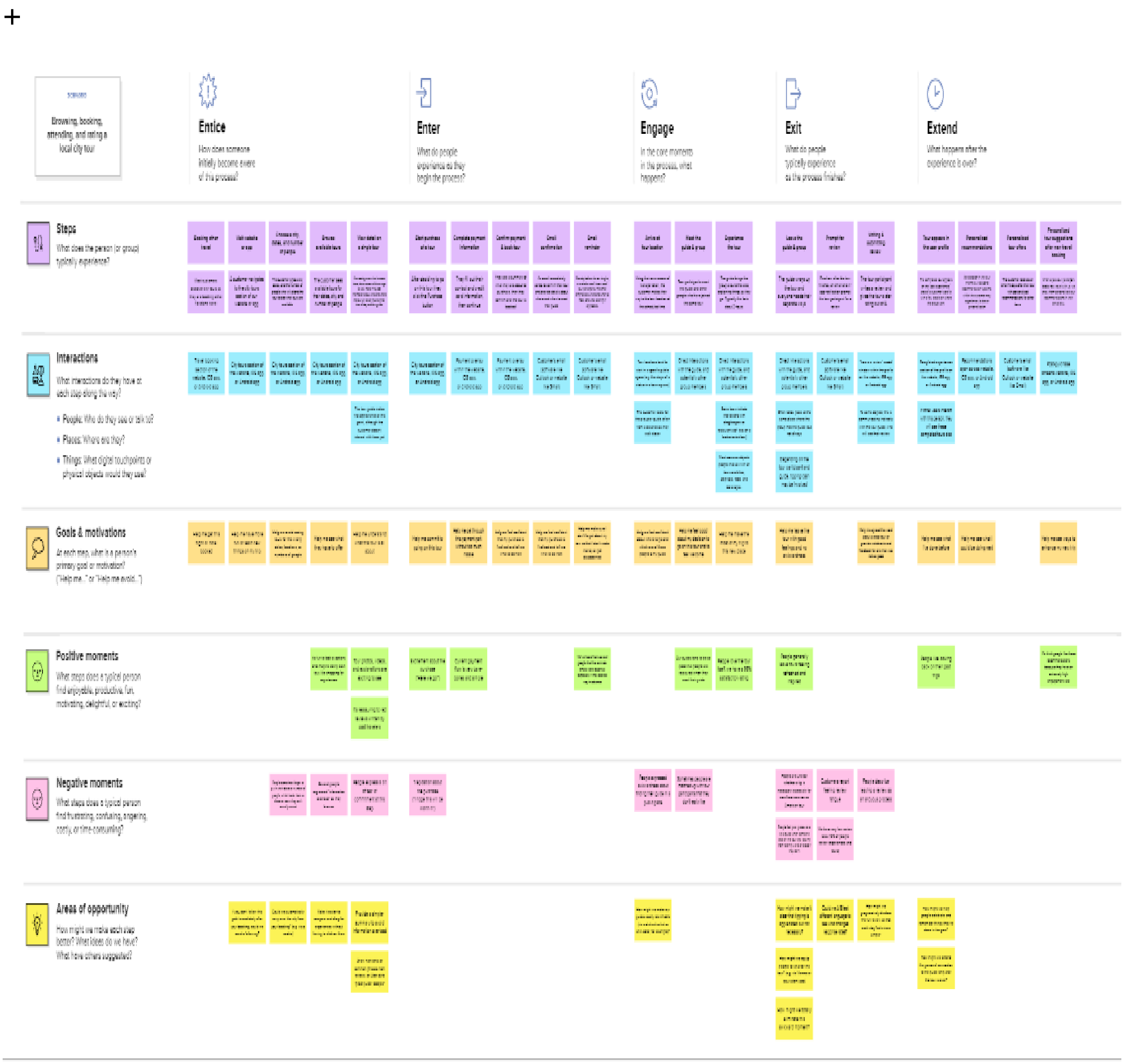
5.2 SOLUTION & TECHNICAL ARCHITECTURE

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behaviour, and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed and delivered.


Solution Architecture
User
Cluster
Worker Node
Application
IBM Cloud Kubernetes Service
IBM DB2
IBM DataBase for Store the User Data
Send Email alert on a Request of Plasma
SendGrid
IBM Cloud Container Registry

5.3 USER STORIES


Browsing, booking, attending, and rating a local city tour
Entice
How does someone initially become aware of this process?
Enter
What do people experience as they begin the process?
Engage
In the core moments in the process, what happens?
Exit
What do people typically experience as the process finishes?
Extend
What happens after the experience is over?
Steps
What does the person (or group) typically experience?
Visit website or app
Choose a city, dates, and number of people
Browse available tours
View details on a single tour
Start purchase of a tour
Complete payment information
Confirm payment & book tour
Email confirmation
Email reminder
Arrive at tour location
Meet the guide & group
Experience the tour
Leave the guide & group
Prompt for review
Writing & submitting review
Tour appears in the user profile
Personalized recommendations
Personalized tour offers
Personalized tour suggestions after new travel booking
Interactions
What interactions do they have at each step along the way?
People: Who do they see or talk to?
Places: Where are they?
Things: What digital touchpoints or physical objects would they use?
Goals & motivations
At each step, what is a person's primary goal or motivation? ("Help me..." or "Help me avoid...")
Positive moments
What steps does a typical person find enjoyable, productive, fun, motivating, delightful, or exciting?
Negative moments
What steps does a typical person find frustrating, confusing, angering, costly, or time-consuming?
Areas of opportunity
How might we make each step better? What ideas do we have? What have others suggested?

CHAPTER 6

PROJECT PLANNING AND SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

Sprint	Functional Requirements	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	REGISTRATION	USN-1	As a user, I can register for the application by entering the email, password and confirming in the retype password.	3	High	4
Sprint-2	LOGIN	USN-2	As a user, I can login to the application by entering the email and password.	3	High	4
Sprint-3	REGISTER TO DONATE	USN-3	As a user, I can login to the application and find the current bank to donate plasma and confirm my booking.	2	Medium	4

Sprint-1,2	FIND THE BANK	USN-4	As a user, I can directly access the application and find the plasma available bank.	3	High	4
Sprint-3	REQUEST FOR PLASMA	USN-5	As a user, I can enter into the application and find the current bank and request for plasma and state the emergency.	2	Medium	4
Sprint-3	MAINTAIN THE APPLICATION	USN-6	As Administrator, I can login to the application by entering the email, password and maintaining the details.	5	High	4
Sprint-4	CONNECT BANK WITH USERS	USN-7	As Administrator, I can hold the good communication between bank and user.	1	Low	4
Sprint-4	MAINTAIN DATABASE	USN-8	As Administrator, I can hold the exact details of donor and patient and also bank for requesting and available of	3	High	4

			plasma.			
Sprint-4	HELP THE USER WITH BOT MESSAGE	USN-9	As AI BOT, I can hold the good communication between bank and user and also help the user whenever it requires.	3	Medium	4

6.2 SPRINT DELIVERY SCHEDULE

SPRINT	TOTAL STORY POINTS	DURATION	SPRINT START DATE	SPRINT END DATE	STORY POINTS COMPLETED (AS PLANNED END DATE)	SPRINT RELEASE DATE (ACTUAL)
Sprint-1	20	6 Days	22 Oct 2022	27 Oct 2022	20	28 Oct 2022
Sprint-2	20	6 Days	28 Oct 2022	3 Nov 2022	20	4 Nov 2022
Sprint-3	20	6 Days	4 Nov 2022	10 Nov 2022	20	11 Nov 2022
Sprint-4	20	6 Days	11 Nov 2022	17 Nov 2022	20	18 Nov 2022

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)

Sprint duration = 6 days

Velocity of the team = 20 points

Average velocity (AV) = Velocity/ Sprint duration

AV = 20/6 = 3.34

Average Velocity = 3.34

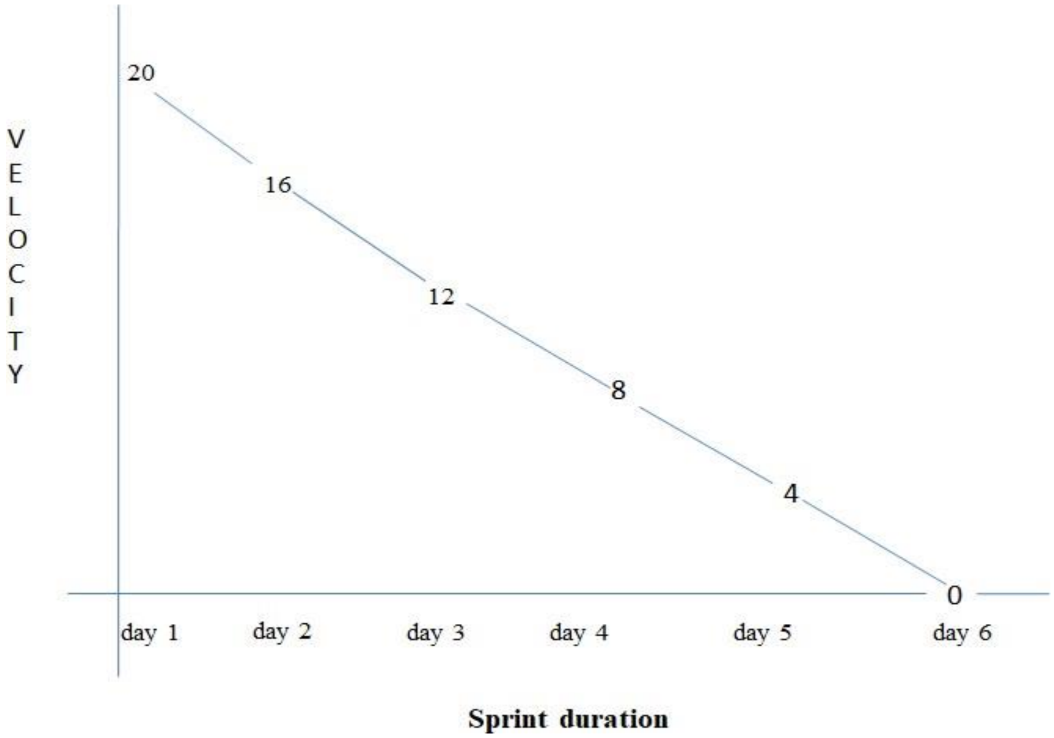
6.3 REPORTS FROM JIRA

BURN-DOWN 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.

This chart tracks the total work remaining, also whether the sprint is achieving the project goal or not. It helps the team to manage the progress and respond accordingly.

A burndown chart is a project management chart that shows how quickly a team is working to a customer’s user stories. This agile tool captures the description of a feature from an end-user perspective and shows the total effort against the amount of work for each iteration or agile sprint.



CHAPTER 7

CODING & SOLUTIONING

7.1 FEATURE 1

```
from flask import render_template
import sqlite3
# import requests
from flask import Flask
from flask import request,redirect,url_for,session,flash
from flask_wtf import Form
from wtforms import TextField
app = Flask(__name__)
app.secret_key = "super secret key"

@app.route('/')
def hel():
    conn = sqlite3.connect('database.db')
    print("Opened database successfully")
    conn.execute('CREATE TABLE IF NOT EXISTS users (name TEXT, addr TEXT, city TEXT, pin
TEXT, bg TEXT,email TEXT UNIQUE, pass TEXT)')
    print( "Table created successfully")
    conn.close()
    if session.get('username')==True:
        messages = session['username']

    else:
        messages = ""
    user = {'username': messages}
    return redirect(url_for('index',user=user))


@app.route('/reg')
def add():
    return render_template('register.html')


@app.route('/addrec',methods = ['POST', 'GET'])
def addrec():
   msg = ""
   #con = None
   if request.method == 'POST':
      try:
         nm = request.form['nm']
         addr = request.form['add']
         city = request.form['city']
         pin = request.form['pin']
         bg = request.form['bg']
         email = request.form['email']
         passs = request.form['pass']

         with sqlite3.connect("database.db") as con:
             cur = con.cursor()
             cur.execute("INSERT INTO users (name,addr,city,pin,bg,email,pass) VALUES
(?,?,?,?,?,?,?)",(nm,addr,city,pin,bg,email,passs) )
             con.commit()
             msg = "Record successfully added"



      except:
             con.rollback()
             msg = "error in insert operation"

      finally:
             flash('done')
```

```
            return redirect(url_for('index'))
            con.close()





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



    if request.method == 'POST':
        if session.get('username') is not None:
            messages = session['username']

        else:
            messages = ""
        user = {'username': messages}
        print(messages)
        val = request.form['search']
        print(val)
        type = request.form['type']
        print(type)
        if type=='blood':
            con = sqlite3.connect('database.db')
            con.row_factory = sqlite3.Row

            cur = con.cursor()
            cur.execute("select * from users where bg=?",(val,))
            search = cur.fetchall();
            cur.execute("select * from users ")

            rows = cur.fetchall();


            return render_template('index.html', title='Home', user=user,rows=rows,search=search)

        if type=='donorname':
            con = sqlite3.connect('database.db')
            con.row_factory = sqlite3.Row

            cur = con.cursor()
            cur.execute("select * from users where name=?",(val,))
            search = cur.fetchall();
            cur.execute("select * from users ")

            rows = cur.fetchall();


            return render_template('index.html', title='Home', user=user,rows=rows,search=search)



    if session.get('username') is not None:
        messages = session['username']

    else:
        messages = ""
    user = {'username': messages}
    print(messages)
    if request.method=='GET':
        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row

        cur = con.cursor()
        cur.execute("select * from users ")

        rows = cur.fetchall();
        return render_template('index.html', title='Home', user=user, rows=rows)

    #messages = request.args['user']




@app.route('/list')
def list():
   con = sqlite3.connect('database.db')
   con.row_factory = sqlite3.Row

   cur = con.cursor()
   cur.execute("select * from users")

   rows = cur.fetchall();
   print(rows)
   return render_template("list.html",rows = rows)

@app.route('/drop')
def dr():
        con = sqlite3.connect('database.db')
        con.execute("DROP TABLE request")
        return "dropped successfully"
```

7.2 FEATURE 2

```
from flask import render_template
import sqlite3
# import requests
from flask import Flask
from flask import request,redirect,url_for,session,flash
from flask_wtf import Form
from wtforms import TextField
app = Flask(__name__)
app.secret_key = "super secret key"

@app.route('/')
def hel():
    conn = sqlite3.connect('database.db')
    print("Opened database successfully")
    conn.execute('CREATE TABLE IF NOT EXISTS users (name TEXT, addr TEXT, city TEXT, pin
TEXT, bg TEXT,email TEXT UNIQUE, pass TEXT)')
    print( "Table created successfully")
    conn.close()
    if session.get('username')==True:
        messages = session['username']

    else:
        messages = ""
    user = {'username': messages}
    return redirect(url_for('index',user=user))


@app.route('/reg')
def add():
    return render_template('register.html')


@app.route('/addrec',methods = ['POST', 'GET'])
def addrec():
   msg = ""
   #con = None
   if request.method == 'POST':
      try:
         nm = request.form['nm']
         addr = request.form['add']
         city = request.form['city']
         pin = request.form['pin']
         bg = request.form['bg']
         email = request.form['email']
         passs = request.form['pass']

         with sqlite3.connect("database.db") as con:
            cur = con.cursor()
            cur.execute("INSERT INTO users (name,addr,city,pin,bg,email,pass) VALUES
(?,?,?,?,?,?,?)",(nm,addr,city,pin,bg,email,passs) )
            con.commit()
            msg = "Record successfully added"



      except:
            con.rollback()
            msg = "error in insert operation"
      finally:
            flash('done')
            return redirect(url_for('index'))
            con.close()




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



    if request.method == 'POST':
        if session.get('username') is not None:
            messages = session['username']

        else:
            messages = ""
        user = {'username': messages}
        print(messages)
        val = request.form['search']
        print(val)
        type = request.form['type']
        print(type)
        if type=='blood':
            con = sqlite3.connect('database.db')
            con.row_factory = sqlite3.Row

            cur = con.cursor()
            cur.execute("select * from users where bg=?",(val,))
            search = cur.fetchall();
            cur.execute("select * from users ")

            rows = cur.fetchall();
            return render_template('index.html', title='Home', user=user,rows=rows,search=search)

        if type=='donorname':
            con = sqlite3.connect('database.db')
            con.row_factory = sqlite3.Row

            cur = con.cursor()
            cur.execute("select * from users where name=?",(val,))
            search = cur.fetchall();
            cur.execute("select * from users ")

            rows = cur.fetchall();


            return render_template('index.html', title='Home', user=user,rows=rows,search=search)



    if session.get('username') is not None:
        messages = session['username']

    else:
        messages = ""
    user = {'username': messages}
    print(messages)
    if request.method=='GET':
        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row

        cur = con.cursor()
        cur.execute("select * from users ")

        rows = cur.fetchall();
        return render_template('index.html', title='Home', user=user, rows=rows)

    #messages = request.args['user']
```

```
@app.route('/list')
def list():
   con = sqlite3.connect('database.db')
   con.row_factory = sqlite3.Row

   cur = con.cursor()
   cur.execute("select * from users")

   rows = cur.fetchall();
   print(rows)
   return render_template("list.html",rows = rows)

@app.route('/drop')
def dr():
        con = sqlite3.connect('database.db')
        con.execute("DROP TABLE request")
        return "dropped successfully"

@app.route('/login',methods = ['POST', 'GET'])
def login():
    if request.method == 'GET':
        return render_template('/login.html')
    if request.method == 'POST':

        email = request.form['email']
        password = request.form['pass']
        if email == 'admin@bloodbank.com' and password == 'admin':
            a = 'yes'
            session['username'] = email
            #session['logged_in'] = True
            session['admin'] = True
            return redirect(url_for('index'))
        #print((password,email))
        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row

        cur = con.cursor()
        cur.execute("select email,pass from users where email=?",(email,))
        rows = cur.fetchall();
        for row in rows:
            print(row['email'],row['pass'])
            a = row['email']
            session['username'] = a
            session['logged_in'] = True
            print(a)
            u = {'username': a}
            p = row['pass']
            print(p)

            if email == a and password == p:
                return redirect(url_for('index'))
            else:
                return render_template('/login.html')
        return render_template('/login.html')
    else:
        return render_template('/')


@app.route('/logout')
def logout():
    # remove the username from the session if it is there
    session.pop('username', None)
    session.pop('logged_in',None)
    try:
        session.pop('admin',None)
    except KeyError as e:
        print("I got a KeyError - reason " +str(e))


    return redirect(url_for('login'))


@app.route('/dashboard')
def dashboard():
    totalblood=0
    con = sqlite3.connect('database.db')
    con.row_factory = sqlite3.Row
```

```
cur = con.cursor()
cur.execute("select * from blood")

rows = cur.fetchall();
for row in rows:
    totalblood  += int(row['qty'])

cur.execute("select * from users")
users = cur.fetchall();

Apositive=0
Opositive=0
Bpositive=0
Anegative=0
Onegative=0
Bnegative=0
ABpositive=0
ABnegative = 0

print(rows)
cur.execute("select * from blood where type=?",('A+',))
type = cur.fetchall();
for a in type:
    Apositive += int(a['qty'])

cur.execute("select * from blood where type=?",('A-',))
type = cur.fetchall();
for a in type:
    Anegative += int(a['qty'])


cur.execute("select * from blood where type=?",('O+',))
type = cur.fetchall();
for a in type:
    Opositive += int(a['qty'])

cur.execute("select * from blood where type=?",('O-',))
type = cur.fetchall();
```

```
    for a in type:
        Onegative += int(a['qty'])

    cur.execute("select * from blood where type=?",('B+',))
    type = cur.fetchall();
    for a in type:
        Bpositive += int(a['qty'])


    cur.execute("select * from blood where type=?",('B-',))
    type = cur.fetchall();
    for a in type:
        Bnegative += int(a['qty'])



    cur.execute("select * from blood where type=?",('AB+',))
    type = cur.fetchall();
    for a in type:
        ABpositive += int(a['qty'])


    cur.execute("select * from blood where type=?",('AB-',))
    type = cur.fetchall();
    for a in type:
        ABnegative += int(a['qty'])


    bloodtypestotal = {'apos':
Apositive,'aneg':Anegative,'opos':Opositive,'oneg':Onegative,'bpos':Bpositive,'bneg':Bnegative,'abpos':AB
positive,'abneg':ABnegative}




    return render_template("requestdonors.html",rows = rows,totalblood =
totalblood,users=users,bloodtypestotal=bloodtypestotal)
```

```
@app.route('/bloodbank')
def bl():
    conn = sqlite3.connect('database.db')
    print("Opened database successfully")
    conn.execute('CREATE TABLE IF NOT EXISTS blood (id INTEGER PRIMARY KEY
AUTOINCREMENT, type TEXT, donorname TEXT, donorsex TEXT, qty TEXT, dweight TEXT,
donoremail TEXT, phone TEXT)')
    print( "Table created successfully")
    conn.close()
    return render_template('/adddonor.html')


@app.route('/addb',methods =['POST','GET'])
def addb():
    msg = ""
    if request.method == 'POST':
        try:
            type = request.form['blood_group']
            donorname = request.form['donorname']
            donorsex = request.form['gender']
            qty = request.form['qty']
            dweight = request.form['dweight']
            email = request.form['email']
            phone = request.form['phone']



            with sqlite3.connect("database.db") as con:
                cur = con.cursor()
                cur.execute("INSERT INTO blood (type,donorname,donorsex,qty,dweight,donoremail,phone)
VALUES (?,?,?,?,?,?,?)",(type,donorname,donorsex,qty,dweight,email,phone) )
                con.commit()
                msg = "Record successfully added"
        except:
            con.rollback()
            msg = "error in insert operation"

        finally:
             flash("added new entry!")
            return redirect(url_for('dashboard'))
            con.close()

    else:
        return render_template("rest.html",msg=msg)

@app.route("/editdonor/<id>", methods=('GET', 'POST'))
def editdonor(id):
    msg =""
    if request.method == 'GET':
        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row

        cur = con.cursor()
        cur.execute("select * from blood where id=?",(id,))
        rows = cur.fetchall();
        return render_template("editdonor.html",rows = rows)
    if request.method == 'POST':
        try:
           type = request.form['blood_group']
           donorname = request.form['donorname']
           donorsex = request.form['gender']
           qty = request.form['qty']
           dweight = request.form['dweight']
           email = request.form['email']
           phone = request.form['phone']



           with sqlite3.connect("database.db") as con:
              cur = con.cursor()
              cur.execute("UPDATE blood SET type = ?, donorname = ?, donorsex = ?, qty = ?,dweight = ?,
donoremail = ?,phone = ? WHERE id = ?",(type,donorname,donorsex,qty,dweight,email,phone,id) )
              con.commit()
              msg = "Record successfully updated"
        except:
           con.rollback()
           msg = "error in insert operation"
        finally:
            flash('saved successfully')
            return redirect(url_for('dashboard'))
            con.close()

@app.route("/myprofile/<email>", methods=('GET', 'POST'))
def myprofile(email):
    msg =""
    if request.method == 'GET':


        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row

        cur = con.cursor()
        cur.execute("select * from users where email=?",(email,))
        rows = cur.fetchall();
        return render_template("myprofile.html",rows = rows)
    if request.method == 'POST':
        try:
           name = request.form['name']
           addr = request.form['addr']
           city = request.form['city']
           pin = request.form['pin']
           bg = request.form['bg']
           emailid = request.form['email']

           print(name,addr)



           with sqlite3.connect("database.db") as con:
              cur = con.cursor()
              cur.execute("UPDATE users SET name = ?, addr = ?, city = ?, pin = ?,bg = ?, email = ? WHERE
email = ?",(name,addr,city,pin,bg,emailid,email) )
              con.commit()
              msg = "Record successfully updated"
        except:
           con.rollback()
```

```
            msg = "error in insert operation"

         finally:
            flash('profile saved')
            return redirect(url_for('index'))
            con.close()



@app.route('/contactforblood/<emailid>', methods=('GET', 'POST'))
def contactforblood(emailid):
    if request.method == 'GET':
        conn = sqlite3.connect('database.db')
        print("Opened database successfully")
        conn.execute('CREATE TABLE IF NOT EXISTS request (id INTEGER PRIMARY KEY
AUTOINCREMENT, toemail TEXT, formemail TEXT, toname TEXT, toaddr TEXT)')
        print( "Table created successfully")
        fromemail = session['username']
        name = request.form['nm']
        addr = request.form['add']

        print(fromemail,emailid)
        conn.execute("INSERT INTO request (toemail,formemail,toname,toaddr) VALUES
(?,?,?,?)",(emailid,fromemail,name,addr) )
        conn.commit()
        conn.close()
        flash('request sent')
        return redirect(url_for('index'))
    if request.method == 'POST':
        conn = sqlite3.connect('database.db')
        print("Opened database successfully")
        conn.execute('CREATE TABLE IF NOT EXISTS request (id INTEGER PRIMARY KEY
AUTOINCREMENT, toemail TEXT, formemail TEXT, toname TEXT, toaddr TEXT)')
        print( "Table created successfully")
        fromemail = session['username']
        name = request.form['nm']
        addr = request.form['add']

        print(fromemail,emailid)
        conn.execute("INSERT INTO request (toemail,formemail,toname,toaddr) VALUES
(?,?,?,?)",(emailid,fromemail,name,addr) )
        conn.commit()
        conn.close()
        flash('request sent')
        return redirect(url_for('index'))



@app.route('/notifications',methods=('GET','POST'))
def notifications():
    if request.method == 'GET':


            conn = sqlite3.connect('database.db')
            print("Opened database successfully")
            conn.row_factory = sqlite3.Row

            cur = conn.cursor()
            cor = conn.cursor()
            cur.execute('select * from request where toemail=?',(session['username'],))
            cor.execute('select * from request where toemail=?',(session['username'],))
            row = cor.fetchone();
            rows = cur.fetchall();
            if row==None:
                return render_template('notifications.html')
            else:
                return render_template('notifications.html',rows=rows)
```

```
@app.route('/deleteuser/<useremail>',methods=('GET', 'POST'))
def deleteuser(useremail):
    if request.method == 'GET':
        conn = sqlite3.connect('database.db')
        cur = conn.cursor()
        cur.execute('delete from users Where email=?',(useremail,))
        flash('deleted user:'+useremail)
        conn.commit()
        conn.close()
        return redirect(url_for('dashboard'))


@app.route('/deletebloodentry/<id>',methods=('GET', 'POST'))
def deletebloodentry(id):
    if request.method == 'GET':
        conn = sqlite3.connect('database.db')
        cur = conn.cursor()
        cur.execute('delete from blood Where id=?',(id,))
        flash('deleted entry:'+id)
        conn.commit()
        conn.close()
        return redirect(url_for('dashboard'))

@app.route('/deleteme/<useremail>',methods=('GET', 'POST'))
def deleteme(useremail):
    if request.method == 'GET':
        conn = sqlite3.connect('database.db')
        cur = conn.cursor()
        cur.execute('delete from users Where email=?',(useremail,))
        flash('deleted user:'+useremail)
        conn.commit()
        conn.close()
        session.pop('username', None)
        session.pop('logged_in',None)
        return redirect(url_for('index'))
```

```
@app.route('/deletenoti/<id>',methods=('GET', 'POST'))
def deletenoti(id):
    if request.method == 'GET':
        conn = sqlite3.connect('database.db')
        cur = conn.cursor()
        cur.execute('delete from request Where id=?',(id,))
        flash('deleted notification:'+id)
        conn.commit()
        conn.close()
        return redirect(url_for('notifications'))



if __name__ == '__main__':
    app.run(debug=True)
```

CHAPTER 8

TESTING

8.1 TEST CASES

Test cases for Registration

Test Engineer:	XYZ – Your Name Here
Test Case ID:	TC1
Related UC	UC1
Date:	27-07-2019
Purpose:	In order to verify that the user is registering according to required information and validation.
Pre-Req:	Web Server must be up. And available for the users. And Enter valid information
Test Data:	Name (String) Phone # (String) City (String) Blood group (String) Location (String)
Steps:	Steps to carry out the test. See step formatting rules below. Choose your type from navigation drawer Choose option menu from the action bar Enter Valid Information Press Submit etc.
Status:	Pass

Test cases for Search Blood Donors

Test Engineer:	XYZ – Your Name Here
Test Case ID:	TC2
Related UC	UC2
Date:	27-07-2019
Purpose:	The user can search and filter donors as per the required blood group
Pre-Req:	Web Server must be up or data is already synchronized in case of error in connection
Test Data:	City Blood group
Steps:	Steps to carry out the test. See step formatting rules below. Choose Volunteer Donors from Navigation Drawer Search required blood donor from list Filter the result by city or blood etc.
Status:	Pass

Test cases for Make Online Request

Test Engineer:	XYZ – Your Name Here
Test Case ID:	TC5
Related UC	UC5
Date:	27-07-2019
Purpose:	The user can make online blood request
Pre-Req:	Web Server must be up
Test Data:	Name (String) Phone # (String)

	City (String) Blood group (String) Location (String)
Steps:	Steps to carry out the test. See step formatting rules below. Choose User Type from Navigation Drawer Choose request button from the action bar Enter Valid information Submit etc.
Status:	Pass

Test cases for Login

Test Engineer:	XYZ – Your Name Here
Test Case ID:	TC8
Related UC	UC8
Date:	27-07-2019
Purpose:	In order to get login as a blood bank for stock modification.
Pre-Req:	Web Server must be up
Test Data:	Email (String) Password (String)
Steps:	Steps to carry out the test. See step formatting rules below. Choose Bloodbank from Navigation Drawer Choose Sign in Enter username and password Click login see the terms of use page

	etc.
Status:	Pass

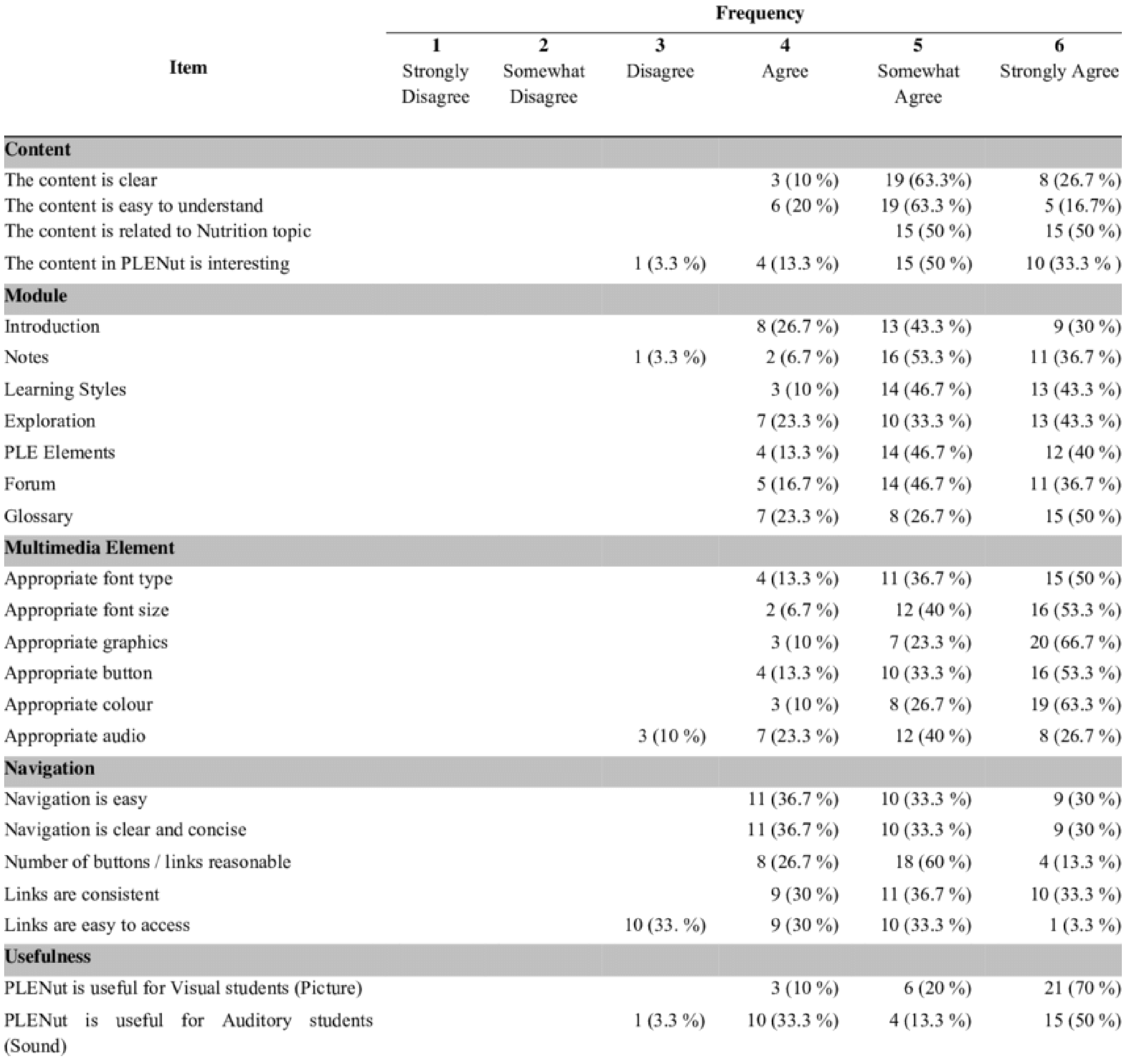
Test cases for Update Stock

Test Engineer:	XYZ – Your Name Here
Test Case ID:	TC9
Related UC	UC9
Date:	27-07-2019
Purpose:	In order to update blood bank stock
Pre-Req:	Web Server must be up
Test Data:	Blood bank Name (String) Blood Type (String) Bag # (String) Hemolysis (String) Quantity (String)
Steps:	Steps to carry out the test. See step formatting rules below. After login as Blood bank Choose Blood Stock Register Enter valid information Submit etc.
Status:	Pass

Test cases for View Requests

Test Engineer:	XYZ – Your Name Here
Test Case ID:	TC10
Related UC	UC10
Date:	27-07-2019
Purpose:	In order to view blood requests
Pre-Req:	Web Server must be up
Test Data:	Just Required Internet Connection
Steps:	Steps to carry out the test. See step formatting rules below. Choose User Type from Navigation Drawer Choose request button from the action bar View Requests Filter result by blood or city etc.
Status:	Pass

8.2 USER ACCEPTANCE TESTING



Item	Frequency					
	1 Strongly Disagree	2 Somewhat Disagree	3 Disagree	4 Agree	5 Somewhat Agree	6 Strongly Agree
Content						
The content is clear				3 (10 %)	19 (63.3%)	8 (26.7 %)
The content is easy to understand				6 (20 %)	19 (63.3 %)	5 (16.7%)
The content is related to Nutrition topic					15 (50 %)	15 (50 %)
The content in PLENut is interesting			1 (3.3 %)	4 (13.3 %)	15 (50 %)	10 (33.3 %)
Module						
Introduction				8 (26.7 %)	13 (43.3 %)	9 (30 %)
Notes			1 (3.3 %)	2 (6.7 %)	16 (53.3 %)	11 (36.7 %)
Learning Styles				3 (10 %)	14 (46.7 %)	13 (43.3 %)
Exploration				7 (23.3 %)	10 (33.3 %)	13 (43.3 %)
PLE Elements				4 (13.3 %)	14 (46.7 %)	12 (40 %)
Forum				5 (16.7 %)	14 (46.7 %)	11 (36.7 %)
Glossary				7 (23.3 %)	8 (26.7 %)	15 (50 %)
Multimedia Element						
Appropriate font type				4 (13.3 %)	11 (36.7 %)	15 (50 %)
Appropriate font size				2 (6.7 %)	12 (40 %)	16 (53.3 %)
Appropriate graphics				3 (10 %)	7 (23.3 %)	20 (66.7 %)
Appropriate button				4 (13.3 %)	10 (33.3 %)	16 (53.3 %)
Appropriate colour				3 (10 %)	8 (26.7 %)	19 (63.3 %)
Appropriate audio			3 (10 %)	7 (23.3 %)	12 (40 %)	8 (26.7 %)
Navigation						
Navigation is easy				11 (36.7 %)	10 (33.3 %)	9 (30 %)
Navigation is clear and concise				11 (36.7 %)	10 (33.3 %)	9 (30 %)
Number of buttons / links reasonable				8 (26.7 %)	18 (60 %)	4 (13.3 %)
Links are consistent				9 (30 %)	11 (36.7 %)	10 (33.3 %)
Links are easy to access			10 (33. %)	9 (30 %)	10 (33.3 %)	1 (3.3 %)
Usefulness						
PLENut is useful for Visual students (Picture)				3 (10 %)	6 (20 %)	21 (70 %)
PLENut is useful for Auditory students (Sound)			1 (3.3 %)	10 (33.3 %)	4 (13.3 %)	15 (50 %)

CHAPTER 9

RESULTS

9.1 PERFORMANCE METRICS

Background

Blood transfusions are essential for many medical procedures, but current supplies of blood are insufficient to meet the needs of all patients. Apps offer a means to improve donor recruitment and enhance blood-donation systems, if they are usable and useful. The plasma donation application was developed. However, there is currently no evidence of the app's usability and usefulness among users.

Methods

A mixed-method study was conducted comprising a quantitative questionnaire with donors, and qualitative semi-structured interviews with healthcare professionals. Descriptive analysis was used for the quantitative data and a thematic approach for the qualitative data.

Results

A total of 401 donors completed the questionnaire, 53.7% of whom were males. Most participants were highly satisfied with the Wateen app and found this app easy to use and navigate. Older people found the app less easy to use compared with younger respondents. Key benefits identified by questionnaire respondents included the potential to encourage donation and improve communication. A total of 12 healthcare professionals were interviewed. Most healthcare professionals expressed that the app was generally acceptable and easy to use. They felt that the app has the potential to be effective in enhancing donor awareness, facilitating communication between donors and healthcare professionals, and improving the efficiency of the donation process. Some accessibility issues were identified that need to be considered. It was also suggested that more be done to expand the app functionality and increase awareness of the app.

Donors Demographic Characteristics

Gender

Male	(53.7%)
Female	(46.3%)

Age

18–24	(28.1%)
25–34	(32.6%)
35–44	(17.5%)
45–54	(10.4%)
55–64	(7.6%)
>65	(3.8%)

Nationality

Saudi	(67.2%)

Gender	
Non-Saudi	(32.8%)
Education Level	
Secondary education degree	19.7%
Diploma degree	20%
Bachelor’s degree	35.4%
Master’s degree	21.7%
PhD degree	3.2%
Employment	
Yes	81%
No	19%

Conclusion

This blood-donation app is highly usable and acceptable among donors and healthcare professionals in World, offering several benefits. Some accessibility issues were identified, along with possibilities for improving accessibility and expanding the app’s functionality.

CHAPTER 10

ADVANTAGES AND DISADVANTAGES

ADVANTAGES

- Plasma donor application helps people with medical needs. Blood plasma is often administered to patients with leukemia or burns, and those who have undergone surgery and other injuries.
- Regular donation improves health. By donating plasma or even whole blood, one’s circulatory system is renewed, enabling the body to produce new supply of blood.
- It allows people to help others. People often feel good and happy about helping others.
- It is a relatively safe process. In a typical plasma centre, donors go through strict screening to make sure they are in top shape to donate.
- Regular plasma donations can guide you toward healthier eating habits. At a donor centre, donors are always encouraged to eat nutritious foods.

DISADVANTAGES

- The process can be very uncomfortable. A number of people are outright afraid of a laboratory or hospital setting, more so when they encounter needles.
- Unregulated donation is risky to both donors and recipients. As mentioned, some centres may deliberately bypass the standard screening process and give a go signal to donors who are not supposed to donate because of a host of health issues.
- It depletes the calcium levels in the body. Plasma centres use sodium citrate and other citric-acid derivatives, blood anticoagulant to make extraction faster and easier.
- It is like body prostitution. A number of people criticize plasma, blood and organ donation in exchange for money as a form of prostitution.

CHAPTER 11

CONCLUSION

CONCLUSION

Although the government is carrying out COVID vaccination campaigns on a large scale, the number of vaccines produced is not enough for all the population to get vaccinated at present. As per the data provided by World Health Organization (WHO), more than 2 million people have died due to the corona virus. However, apart from vaccination, there is another method by which a covid infected person can be treated and the death risk can be reduced. This plasma therapy is considered to be safe and promising. Plasma therapy is an experimental approach to treat corona-positive patients and help them recover.

This system proposed here aims at connecting the donors and the patients through online. Firstly, user can register for this application and they can select whether they are donor or donee. If they are donor they have to provide their blood group, address, contact details. Donee can view all the donors and they also have to provide name, address, contact details. Atlast, donor can get a notification and they can contact each other. Admin can see all the user login details and they are able to edit and delete the user details.

CHAPTER 12

FUTURE SCOPE

FUTURE SCOPE

Plasma application can be developed to further improvement like user accessibility via integrating the application with various social networks application program interfaces. It will block the chain of business through plasma and help the poor to find donor at free of cost.

Appointments can be synchronized with google and outlook calendars for the ease of user. User interface (UI) can be improved in future to accommodate global audience by supporting with their own languages across the countries.

Donors will be able to view and share their own personal experience about their donation. This would help the people who are all gets confused about plasma donor therapy. In further improvised manner, this application can work in either online or offline mode. When the user is offline, the data is being saved locally and when he gets online the modified data will be saved automatically in the database.

CHAPTER 13

APPENDIX

Source code:

```
from flask import render_template

import sqlite3

# import requests

from flask import Flask

from flask import request,redirect,url_for,session,flash

from flask_wtf import Form

from wtforms import TextField

app = Flask(__name__)

app.secret_key = "super secret key"


@app.route('/')

def hel():

    conn = sqlite3.connect('database.db')

    print("Opened database successfully")

    conn.execute('CREATE TABLE IF NOT EXISTS users (name TEXT, addr
TEXT, city TEXT, pin TEXT, bg TEXT,email TEXT UNIQUE, pass TEXT)')

    print( "Table created successfully")

    conn.close()

    if session.get('username')==True:

        messages = session['username']
    else:
        messages = ""
    user = {'username': messages}
    return redirect(url_for('index',user=user))



@app.route('/reg')
def add():
    return render_template('register.html')



@app.route('/addrec',methods = ['POST', 'GET'])
def addrec():
   msg = ""
   #con = None
   if request.method == 'POST':
      try:
         nm = request.form['nm']
         addr = request.form['add']
         city = request.form['city']
         pin = request.form['pin']

         bg = request.form['bg']

         email = request.form['email']

         passs = request.form['pass']


         with sqlite3.connect("database.db") as con:

           cur = con.cursor()

           cur.execute("INSERT INTO users (name,addr,city,pin,bg,email,pass)
VALUES (?,?,?,?,?,?,?)",(nm,addr,city,pin,bg,email,passs) )

           con.commit()

           msg = "Record successfully added"




      except:

           con.rollback()

           msg = "error in insert operation"


      finally:

           flash('done')

           return redirect(url_for('index'))
```

```
            con.close()




@app.route('/index',methods = ['POST','GET'])

def index():



    if request.method == 'POST':

        if session.get('username') is not None:

            messages = session['username']


        else:

            messages = ""

        user = {'username': messages}

        print(messages)

        val = request.form['search']
        print(val)

        type = request.form['type']

        print(type)

        if type=='blood':

            con = sqlite3.connect('database.db')

            con.row_factory = sqlite3.Row


            cur = con.cursor()

            cur.execute("select * from users where bg=?",(val,))

            search = cur.fetchall();

            cur.execute("select * from users ")


            rows = cur.fetchall();




            return render_template('index.html', title='Home', 
user=user,rows=rows,search=search)


        if type=='donorname':

            con = sqlite3.connect('database.db')

            con.row_factory = sqlite3.Row
            cur = con.cursor()

            cur.execute("select * from users where name=?",(val,))

            search = cur.fetchall();

            cur.execute("select * from users ")


            rows = cur.fetchall();




            return render_template('index.html', title='Home',
user=user,rows=rows,search=search)






    if session.get('username') is not None:

        messages = session['username']


    else:

        messages = ""

    user = {'username': messages}

    print(messages)
    if request.method=='GET':
        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row

        cur = con.cursor()
        cur.execute("select * from users ")

        rows = cur.fetchall();
        return render_template('index.html', title='Home', user=user, rows=rows)

    #messages = request.args['user']




@app.route('/list')
def list():
   con = sqlite3.connect('database.db')
   con.row_factory = sqlite3.Row
```

```
  cur = con.cursor()

  cur.execute("select * from users")



  rows = cur.fetchall();

  print(rows)

  return render_template("list.html",rows = rows)



@app.route('/drop')

def dr():

      con = sqlite3.connect('database.db')

      con.execute("DROP TABLE request")

      return "dropped successfully"



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

def login():

   if request.method == 'GET':

      return render_template('/login.html')

   if request.method == 'POST':



      email = request.form['email']
```

```
password = request.form['pass']

if email == 'admin@bloodbank.com' and password == 'admin':

    a = 'yes'

    session['username'] = email

    #session['logged_in'] = True

    session['admin'] = True

    return redirect(url_for('index'))

#print((password,email))

con = sqlite3.connect('database.db')

con.row_factory = sqlite3.Row


cur = con.cursor()

cur.execute("select email,pass from users where email=?",(email,))

rows = cur.fetchall();

for row in rows:

    print(row['email'],row['pass'])

    a = row['email']

    session['username'] = a

    session['logged_in'] = True

    print(a)
```

```
            u = {'username': a}

            p = row['pass']

            print(p)


            if email == a and password == p:

                return redirect(url_for('index'))

            else:

                return render_template('/login.html')

        return render_template('/login.html')

    else:

        return render_template('/')




@app.route('/logout')

def logout():

    # remove the username from the session if it is there

    session.pop('username', None)

    session.pop('logged_in',None)

    try:

        session.pop('admin',None)
```

```
  except KeyError as e:

      print("I got a KeyError - reason " +str(e))



  return redirect(url_for('login'))



@app.route('/dashboard')

def dashboard():

  totalblood=0

  con = sqlite3.connect('database.db')

  con.row_factory = sqlite3.Row


  cur = con.cursor()

  cur.execute("select * from blood")


  rows = cur.fetchall();

  for row in rows:

      totalblood  += int(row['qty'])
```

```
cur.execute("select * from users")

users = cur.fetchall();


Apositive=0

Opositive=0

Bpositive=0

Anegative=0

Onegative=0

Bnegative=0

ABpositive=0

ABnegative = 0


print(rows)

cur.execute("select * from blood where type=?",('A+',))

type = cur.fetchall();

for a in type:

    Apositive += int(a['qty'])


cur.execute("select * from blood where type=?",('A-',))

type = cur.fetchall();
```

```
for a in type:
    Anegative += int(a['qty'])



cur.execute("select * from blood where type=?",('O+',))
type = cur.fetchall();
for a in type:
    Opositive += int(a['qty'])

cur.execute("select * from blood where type=?",('O-',))
type = cur.fetchall();
for a in type:
    Onegative += int(a['qty'])

cur.execute("select * from blood where type=?",('B+',))
type = cur.fetchall();
for a in type:
    Bpositive += int(a['qty'])
```

```
cur.execute("select * from blood where type=?",('B-',))
type = cur.fetchall();
for a in type:
    Bnegative += int(a['qty'])



cur.execute("select * from blood where type=?",('AB+',))
type = cur.fetchall();
for a in type:
    ABpositive += int(a['qty'])



cur.execute("select * from blood where type=?",('AB-',))
type = cur.fetchall();
for a in type:
    ABnegative += int(a['qty'])
```

```
   bloodtypestotal = {'apos':
Apositive,'aneg':Anegative,'opos':Opositive,'oneg':Onegative,'bpos':Bpositive,'b
neg':Bnegative,'abpos':ABpositive,'abneg':ABnegative}







   return render_template("requestdonors.html",rows = rows,totalblood =
totalblood,users=users,bloodtypestotal=bloodtypestotal)



@app.route('/bloodbank')

def bl():

    conn = sqlite3.connect('database.db')

    print("Opened database successfully")

    conn.execute('CREATE TABLE IF NOT EXISTS blood (id INTEGER
PRIMARY KEY AUTOINCREMENT, type TEXT, donorname TEXT,
donorsex TEXT, qty TEXT, dweight TEXT, donoremail TEXT, phone TEXT)')

    print( "Table created successfully")

    conn.close()
    return render_template('/adddonor.html')



@app.route('/addb',methods =['POST','GET'])
def addb():
    msg = ""
    if request.method == 'POST':
        try:
           type = request.form['blood_group']
           donorname = request.form['donorname']
           donorsex = request.form['gender']
           qty = request.form['qty']
           dweight = request.form['dweight']
           email = request.form['email']
           phone = request.form['phone']




           with sqlite3.connect("database.db") as con:
              cur = con.cursor()
              cur.execute("INSERT INTO blood
(type,donorname,donorsex,qty,dweight,donoremail,phone) VALUES
(?,?,?,?,?,?,?)",(type,donorname,donorsex,qty,dweight,email,phone) )

              con.commit()

              msg = "Record successfully added"

        except:

           con.rollback()

           msg = "error in insert operation"



        finally:

            flash("added new entry!")

            return redirect(url_for('dashboard'))

            con.close()



    else:

        return render_template("rest.html",msg=msg)



@app.route("/editdonor/<id>", methods=('GET', 'POST'))

def editdonor(id):

    msg =""

    if request.method == 'GET':
```

```
   con = sqlite3.connect('database.db')
   con.row_factory = sqlite3.Row


   cur = con.cursor()
   cur.execute("select * from blood where id=?",(id,))
   rows = cur.fetchall();
   return render_template("editdonor.html",rows = rows)
if request.method == 'POST':
   try:
     type = request.form['blood_group']
     donorname = request.form['donorname']
     donorsex = request.form['gender']
     qty = request.form['qty']
     dweight = request.form['dweight']
     email = request.form['email']
     phone = request.form['phone']




     with sqlite3.connect("database.db") as con:
```

```
              cur = con.cursor()

              cur.execute("UPDATE blood SET type = ?, donorname = ?, donorsex
= ?, qty = ?,dweight = ?, donoremail = ?,phone = ? WHERE id =
?",(type,donorname,donorsex,qty,dweight,email,phone,id) )

              con.commit()

              msg = "Record successfully updated"

         except:

            con.rollback()

            msg = "error in insert operation"


         finally:

             flash('saved successfully')

             return redirect(url_for('dashboard'))

             con.close()


@app.route("/myprofile/<email>", methods=('GET', 'POST'))

def myprofile(email):

    msg =""

    if request.method == 'GET':
```

```
    con = sqlite3.connect('database.db')
    con.row_factory = sqlite3.Row


    cur = con.cursor()
    cur.execute("select * from users where email=?",(email,))
    rows = cur.fetchall();
    return render_template("myprofile.html",rows = rows)
if request.method == 'POST':
    try:
       name = request.form['name']
       addr = request.form['addr']
       city = request.form['city']
       pin = request.form['pin']
       bg = request.form['bg']
       emailid = request.form['email']


       print(name,addr)
```

```
        with sqlite3.connect("database.db") as con:
          cur = con.cursor()
          cur.execute("UPDATE users SET name = ?, addr = ?, city = ?, pin = ?,bg = ?, email = ? WHERE email = ?",(name,addr,city,pin,bg,emailid,email) )
          con.commit()
          msg = "Record successfully updated"
      except:
        con.rollback()
        msg = "error in insert operation"

      finally:
        flash('profile saved')
        return redirect(url_for('index'))
        con.close()




@app.route('/contactforblood/<emailid>', methods=('GET', 'POST'))
def contactforblood(emailid):
   if request.method == 'GET':
      conn = sqlite3.connect('database.db')
```

```
        print("Opened database successfully")
        conn.execute('CREATE TABLE IF NOT EXISTS request (id INTEGER PRIMARY KEY AUTOINCREMENT, toemail TEXT, formemail TEXT, toname TEXT, toaddr TEXT)')
        print( "Table created successfully")
        fromemail = session['username']
        name = request.form['nm']
        addr = request.form['add']


        print(fromemail,emailid)
        conn.execute("INSERT INTO request (toemail,formemail,toname,toaddr) VALUES (?,?,?,?)",(emailid,fromemail,name,addr) )
        conn.commit()
        conn.close()
        flash('request sent')
        return redirect(url_for('index'))
    if request.method == 'POST':
        conn = sqlite3.connect('database.db')
        print("Opened database successfully")
        conn.execute('CREATE TABLE IF NOT EXISTS request (id INTEGER PRIMARY KEY AUTOINCREMENT, toemail TEXT, formemail TEXT, toname TEXT, toaddr TEXT)')
        print( "Table created successfully")

        fromemail = session['username']

        name = request.form['nm']

        addr = request.form['add']


        print(fromemail,emailid)

        conn.execute("INSERT INTO request (toemail,formemail,toname,toaddr)
VALUES (?,?,?,?)",(emailid,fromemail,name,addr) )

        conn.commit()

        conn.close()

        flash('request sent')

        return redirect(url_for('index'))




@app.route('/notifications',methods=('GET','POST'))

def notifications():

    if request.method == 'GET':



            conn = sqlite3.connect('database.db')
            print("Opened database successfully")
            conn.row_factory = sqlite3.Row


            cur = conn.cursor()
            cor = conn.cursor()
            cur.execute('select * from request where
toemail=?',(session['username'],))
            cor.execute('select * from request where
toemail=?',(session['username'],))
            row = cor.fetchone();
            rows = cur.fetchall();
            if row==None:
               return render_template('notifications.html')
            else:
               return render_template('notifications.html',rows=rows)
```

```
@app.route('/deleteuser/<useremail>',methods=('GET', 'POST'))

def deleteuser(useremail):

    if request.method == 'GET':

        conn = sqlite3.connect('database.db')

        cur = conn.cursor()

        cur.execute('delete from users Where email=?',(useremail,))

        flash('deleted user:'+useremail)

        conn.commit()

        conn.close()

        return redirect(url_for('dashboard'))



@app.route('/deletebloodentry/<id>',methods=('GET', 'POST'))
```

```
def deletebloodentry(id):
    if request.method == 'GET':
        conn = sqlite3.connect('database.db')
        cur = conn.cursor()
        cur.execute('delete from blood Where id=?',(id,))
        flash('deleted entry:'+id)
        conn.commit()
        conn.close()
        return redirect(url_for('dashboard'))


@app.route('/deleteme/<useremail>',methods=('GET', 'POST'))
def deleteme(useremail):
    if request.method == 'GET':
        conn = sqlite3.connect('database.db')
        cur = conn.cursor()
        cur.execute('delete from users Where email=?',(useremail,))
        flash('deleted user:'+useremail)
        conn.commit()
        conn.close()
        session.pop('username', None)
        session.pop('logged_in',None)

        return redirect(url_for('index'))


@app.route('/deletenoti/<id>',methods=('GET', 'POST'))

def deletenoti(id):

    if request.method == 'GET':

        conn = sqlite3.connect('database.db')

        cur = conn.cursor()

        cur.execute('delete from request Where id=?',(id,))

        flash('deleted notification:'+id)

        conn.commit()

        conn.close()

        return redirect(url_for('notifications'))




if __name__ == '__main__':

    app.run(debug=True)
```

GitHub link:

https://github.com/IBM-EPBL/IBM-Project-11855-1659348680