Red Hat

OPENSHIFT

AND

IBM

Smart Internz

IN ASSOCIATION WITH

All India Council for Technical Education AICTE योगः कर्मसु कौशलम्

PRESENTS

PROJECT BUILD-A-THON

PROJECT REPORT

ON

PLASMA DONOR APPLICATION

Submitted By

Sunil K. Joseph

Assistant Professor

Department of Computer Science

Mar Augusthinose College, Ramapuram, Kerala 686576

sunilkjoseph@mac.edu.in

CONTENTS

1.INTRODUCTION

1.1. Overview

During the COVID 19 crisis, the requirement for plasma became high and the donor count being low. Saving the donor information and helping the need by notifying the current donors would be a helping hand.

In regard to the problem faced, an application is to be built which would take the donor details, store it and inform them upon a request. Users need to register an account and login to the application. Once the user logins, he will have a dashboard to view the total number of donors and count of people with specific blood groups.

1.2. Purpose

The main objective of this project is to provide the recipient with a donor who is in good form with no health ailments to donate blood of the corresponding blood group. This project 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. User will have the option to request the blood. Once the user requests, all the people with that blood group will be notified with an SMS.

2.LITERATURE SURVEY

2.1. Existing problem

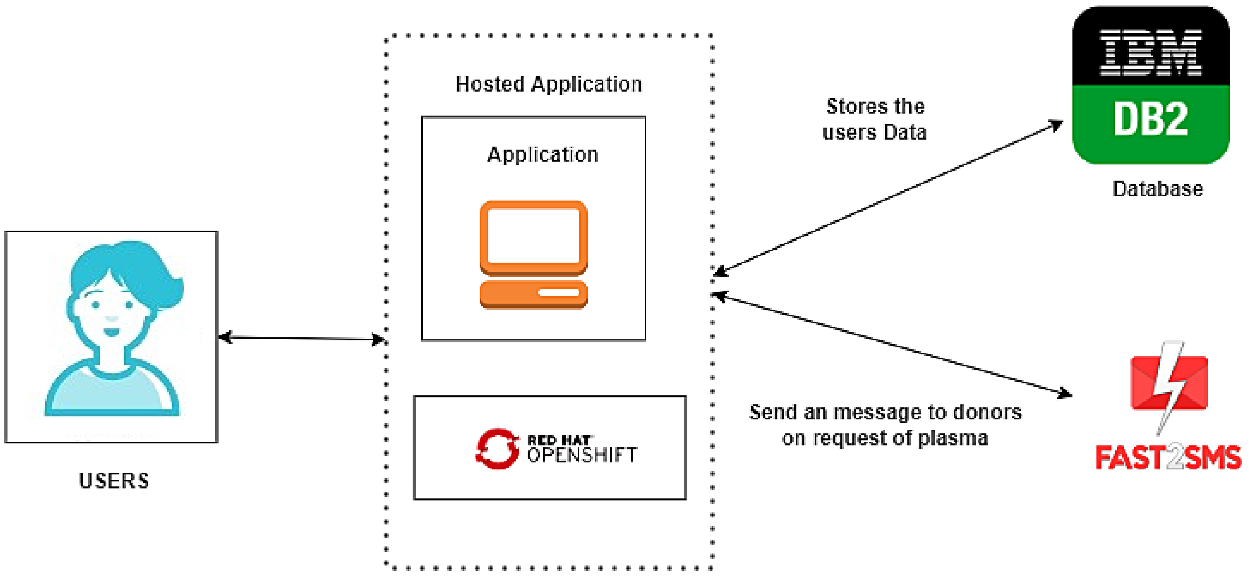
Despite advances in technology, nowadays, most blood bank systems are manual system. As such, there is a prevalent problem in the availability of needed blood types. For instance, when a person needs a certain type of blood and this type is not available in the hospital, family members send messages through social media to those who can donate to them and this process takes longer than the life of the patient to the most dangerous.

2.2. Proposed solution

To build a web application that is capable of acting as a medium for recipients and donors of blood. The application must be deployed on Cloud Foundry. Create an API Endpoint for the model with the help of IBM Cloud Functions. An alert is to be sent using the Fast2Sms Service to all the registered users whenever a request for blood is posted.

3.THEORITICAL ANALYSIS

3.1. Block Diagram



3.2. Hardware / Software Design

We'll be able to work on IBM DB2, creating flask application, making an application into an docker image and deploying app in Redhat Openshift dev space. Build a flask application which will take the user inputs, update the IBM DB2 database and notify the user upon request.

Hardware Requirements

A PC with internet connectivity

Software Requirements

Operating System	- Windows 11
Front End Tool	- Python 3.10.5
IDE	- Visual Studio Code
Code Repository	- GitHub
Back End Tool	- IBM DB2 service on IBM Cloud
App Deployment Environment	- RedHat OpenShift Dedicated

Red Hat OpenShift Container :

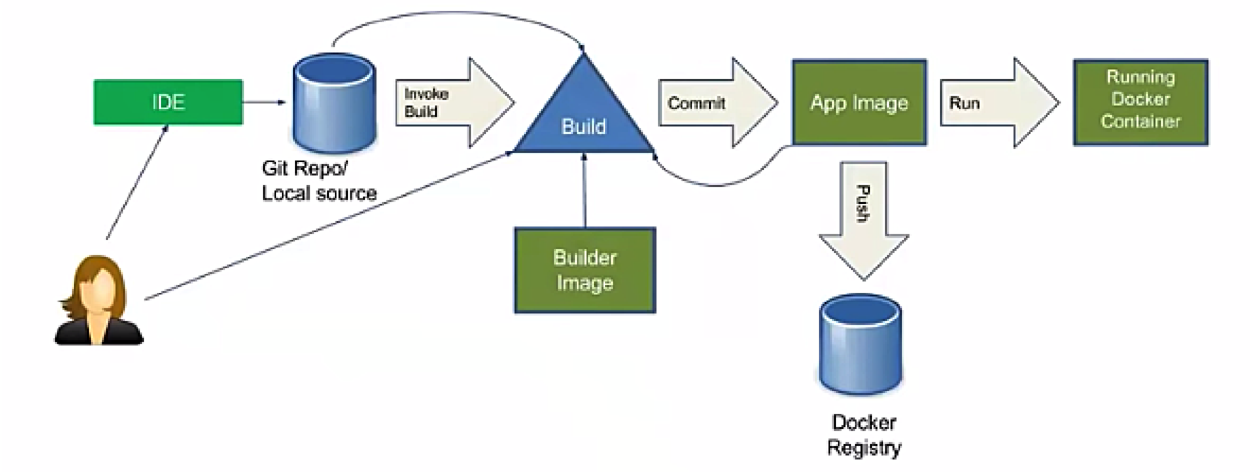
Red Hat OpenShift is a cloud-based Kubernetes platform that helps developers build applications. It delivers a single, consistent Kubernetes platform anywhere that Red Hat Enterprise Linux runs. The platform ships with a user-friendly console to view and manage all your clusters OpenShift gives organizations the ability to build, deploy, and scale applications faster both on-premises and in the cloud. It also protects your development infrastructure at scale with enterprise-grade security. Being a Paas , it could be used with any other web services too.

OpenShift is a great platform to use for building and shipping cloud-native applications. It also supports the developers by making the development and testing
workflow of the applications much easier by ensuring that the developers do not have to worry about switching between the physical and the virtual servers whenever required. Thus, it helps to increase the productivity and efficiency of the existing application workflow with reduced maintenance costs.

Docker

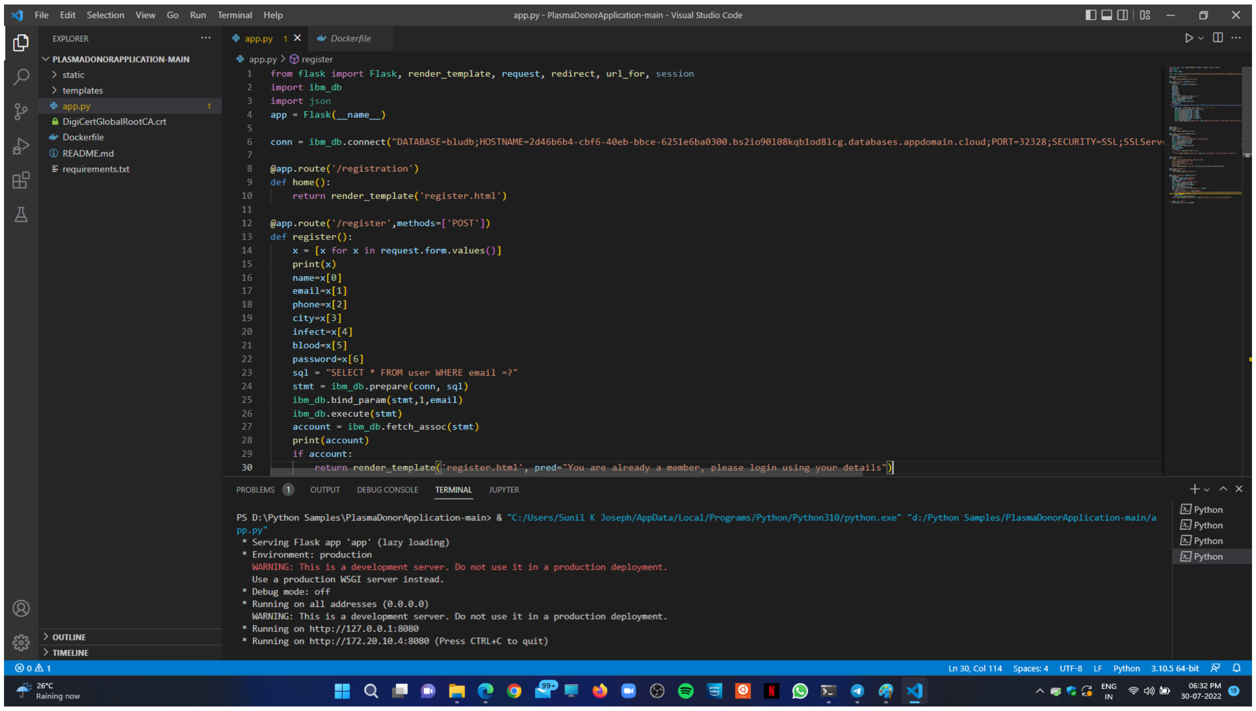
Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. It is used for building and seamlessly integrating legacy projects enabling organizations to achieve high-velocity innovations. It encourages the concept of DevOps methodology through CI/CD (Continuous Integration/Continuous Development). Thus,
the developers can integrate their code into a shared repository as early as possible and deploy it quickly and efficiently. Thus, the local development setup behaves like a live server. It comes up with integrated developer tools. Also, the virtual machine image is openly accessible and shareable. The applications developed on it can be reused and are shareable. It is open-source and available on Github. It reduces the setup cost on the part of the customers and increases the efficiency and the productivity of the existing application workflow as both are open source technologies. This also ensures the scalability of the existing application workflow. Also, as both of the technologies form an integral part of the cloud platform, they can be used independently.

4.FLOWCHART



The application is built by importing Dockerfile from a Git repository along with other source files. It allows one to manage everything easily by keeping all the build scripts in the same repository as your code. Using Git repository gives the advantage of version control of the docker files too.

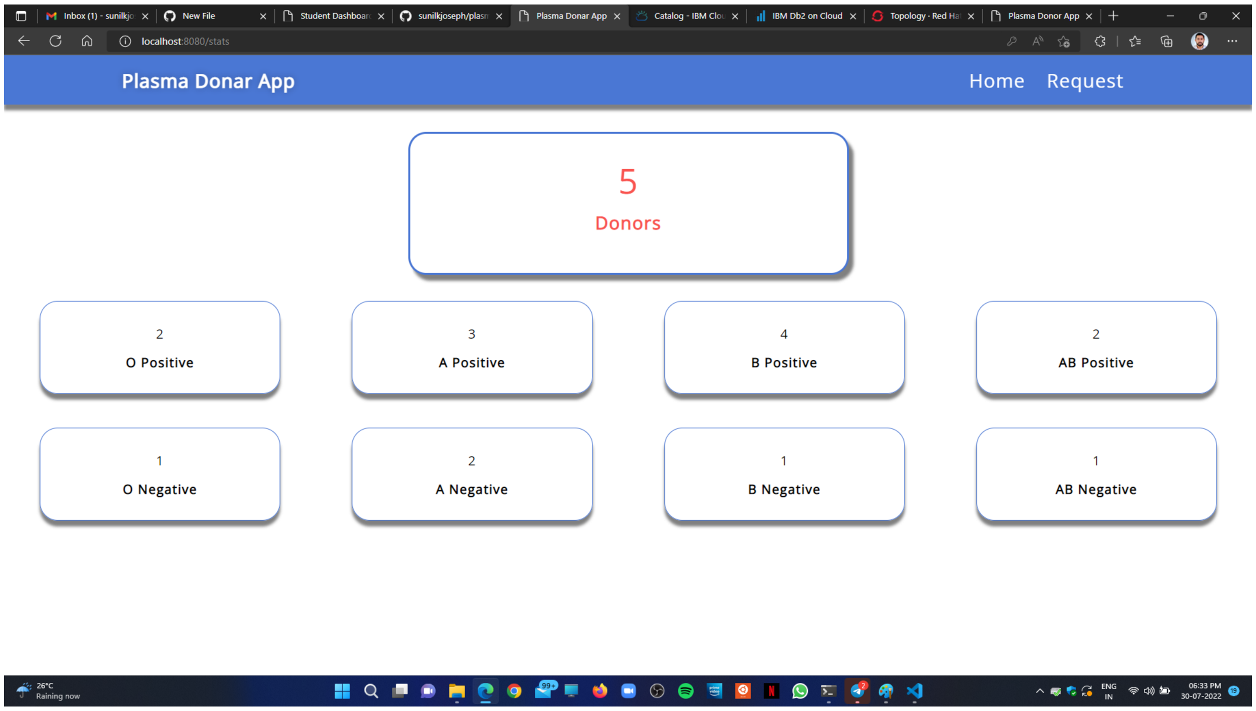
5.RESULT



```python
from flask import Flask, render_template, request, redirect, url_for, session
import ibm_db
import json
app = Flask(__name__)

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=2d46b6b4-cbf6-40eb-bbce-6251e6ba0300.bs2io90108kqb1od8lcg.databases.appdomain.cloud;PORT=32328;SECURITY=SSL;SSLServ

@app.route('/registration')
def home():
    return render_template('register.html')

@app.route('/register',methods=['POST'])
def register():
    x = [x for x in request.form.values()]
    print(x)
    name=x[0]
    email=x[1]
    phone=x[2]
    city=x[3]
    infect=x[4]
    blood=x[5]
    password=x[6]
    sql = "SELECT * FROM user WHERE email =?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,email)
    ibm_db.execute(stmt)
    account = ibm_db.fetch_assoc(stmt)
    print(account)
    if account:
        return render_template('register.html', pred="You are already a member, please login using your details")
```

```
PS D:\Python Samples\PlasmaDonorApplication-main> & "C:/Users/Sunil K Joseph/AppData/Local/Programs/Python/Python310/python.exe" "d:/Python Samples/PlasmaDonorApplication-main/a
pp.py"
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:8080
 * Running on http://172.20.10.4:8080 (Press CTRL+C to quit)
```

Fig. 1 App source code in Visual Studio Code



localhost:8080/stats

Plasma Donar App

Home Request

5
Donors

2
O Positive

3
A Positive

4
B Positive

2
AB Positive

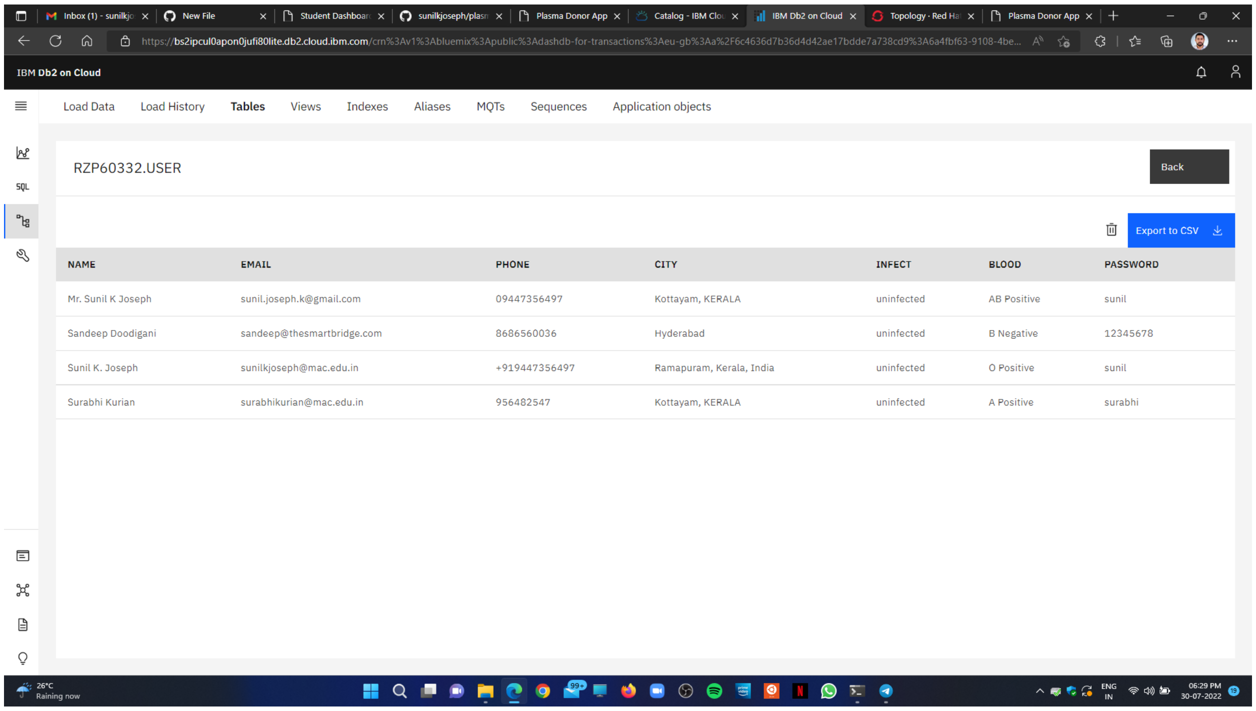
1
O Negative

2
A Negative

1
B Negative

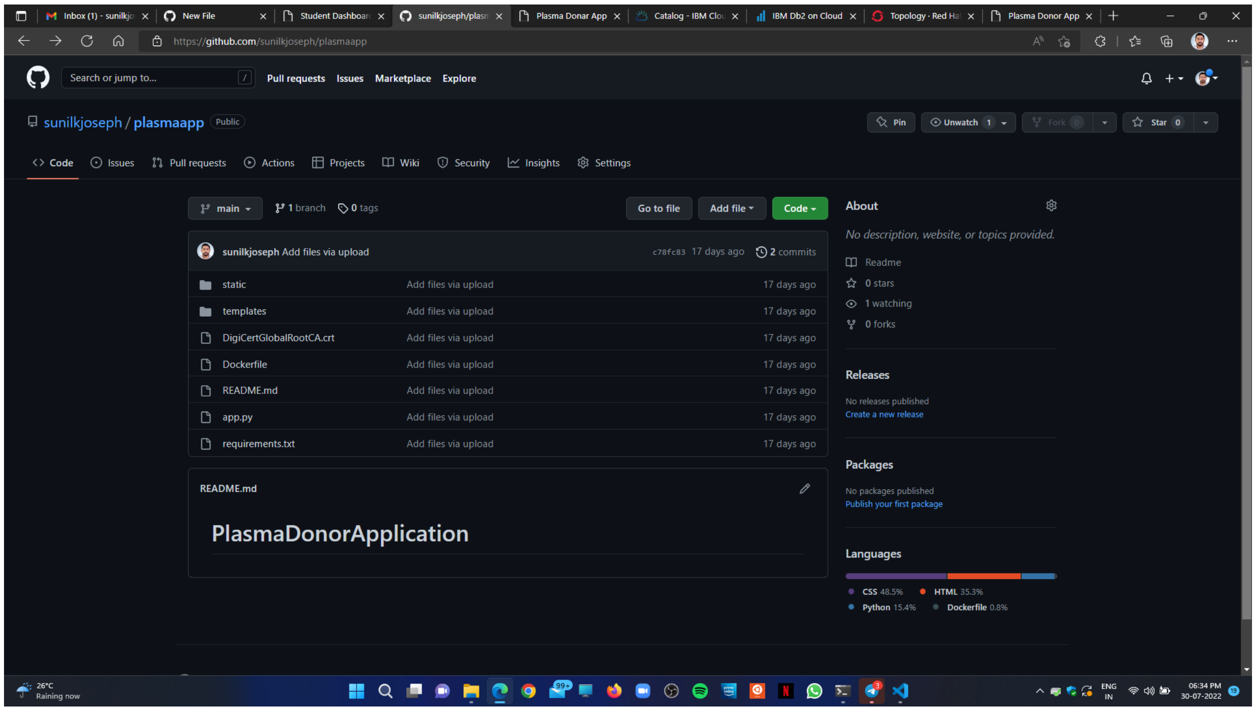
1
AB Negative

Fig. 2 App Successfully deployed in the local machine


RZP60332.USER

NAME	EMAIL	PHONE	CITY	INFECT	BLOOD	PASSWORD
Mr. Sunil K Joseph	sunil.joseph.k@gmail.com	09447356497	Kottayam, KERALA	uninfected	AB Positive	sunil
Sandeep Doodigani	sandeep@thesmartbridge.com	8686560036	Hyderabad	uninfected	B Negative	12345678
Sunil K. Joseph	suniljoseph@mac.edu.in	+919447356497	Ramapuram, Kerala, India	uninfected	O Positive	sunil
Surabhi Kurian	surabhikurian@mac.edu.in	956482547	Kottayam, KERALA	uninfected	A Positive	surabhi

Fig. 3 IBM DB2 Service running on IBM Cloud


sunilkjoseph / plasmaapp

sunilkjoseph Add files via upload c78fc83 17 days ago 2 commits

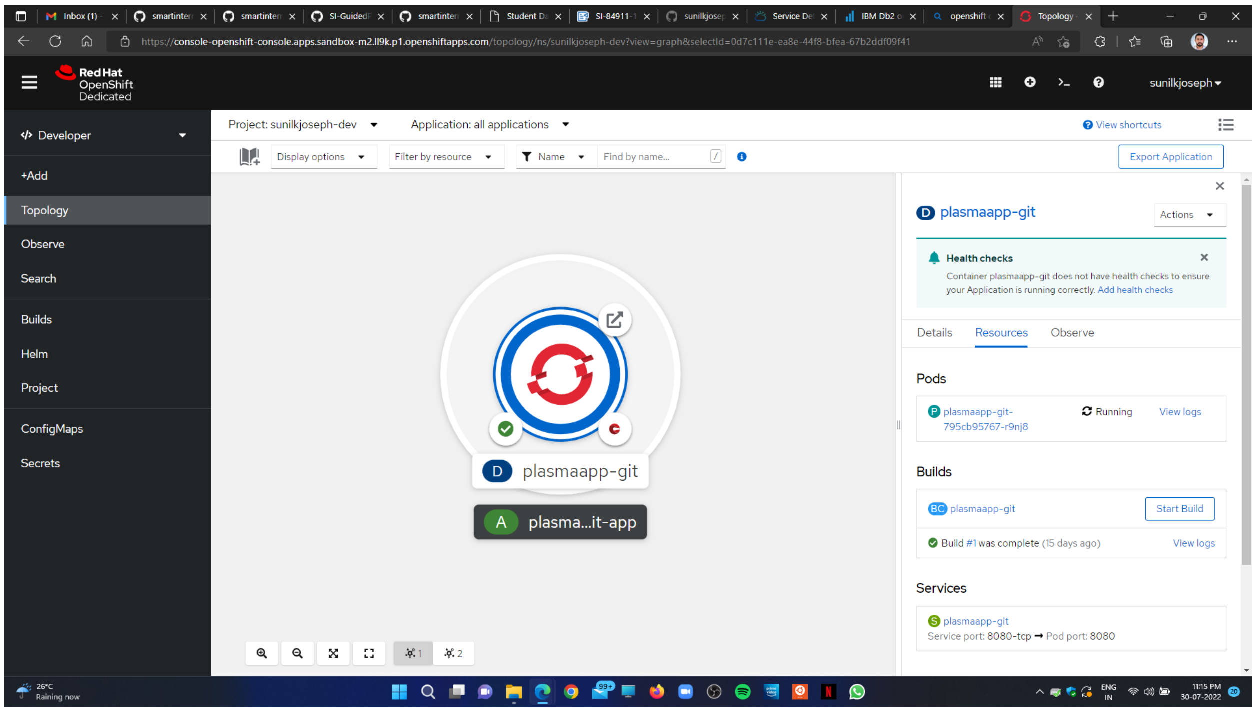
static	Add files via upload	17 days ago
templates	Add files via upload	17 days ago
DigiCertGlobalRootCA.crt	Add files via upload	17 days ago
Dockerfile	Add files via upload	17 days ago
README.md	Add files via upload	17 days ago
app.py	Add files via upload	17 days ago
requirements.txt	Add files via upload	17 days ago

README.md

PlasmaDonorApplication

Languages: CSS 48.5%, HTML 35.3%, Python 15.4%, Dockerfile 0.8%

Fig. 4 Source code pushed into Git Repo



Health checks

Container plasmaapp-git does not have health checks to ensure your Application is running correctly. Add health checks

Pods

plasmaapp-git-795cb95767-r9nj8 Running View logs

Builds

plasmaapp-git Start Build

Build #1 was complete (15 days ago) View logs

Services

plasmaapp-git

Service port: 8080-tcp ➞ Pod port: 8080

Fig. 5 Plasma Donor App deployed in RedHat OpenShift Dedicated

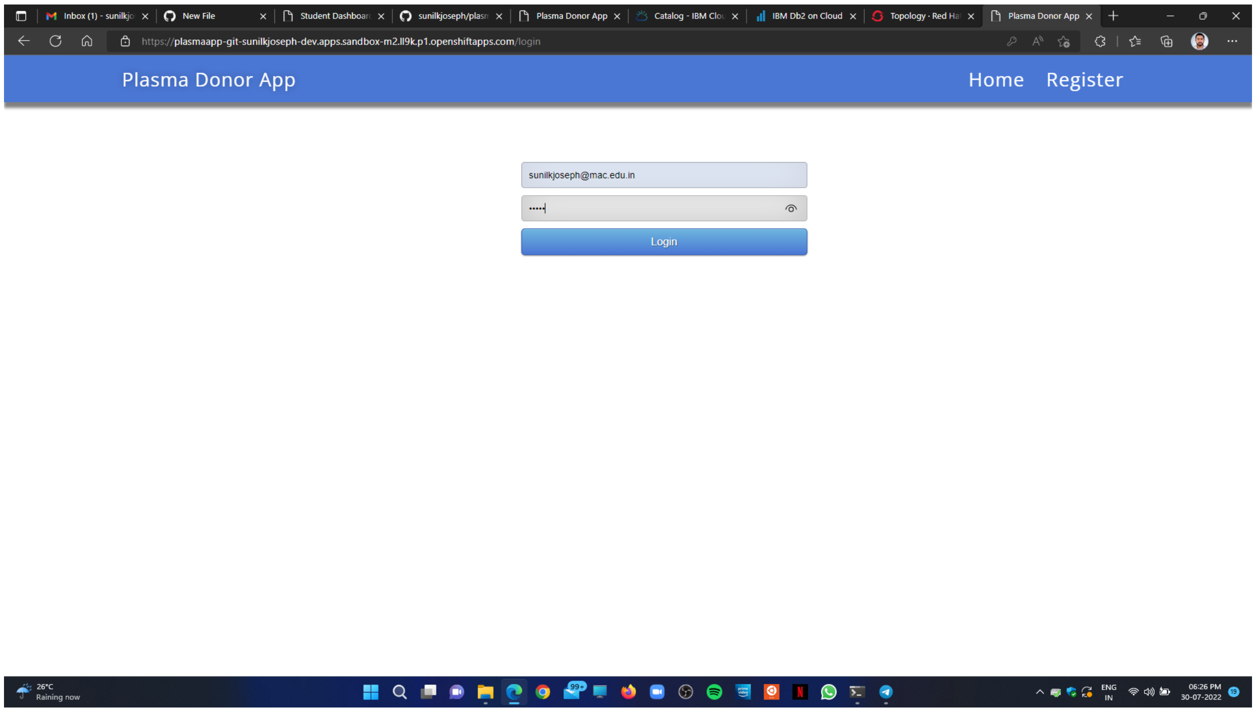
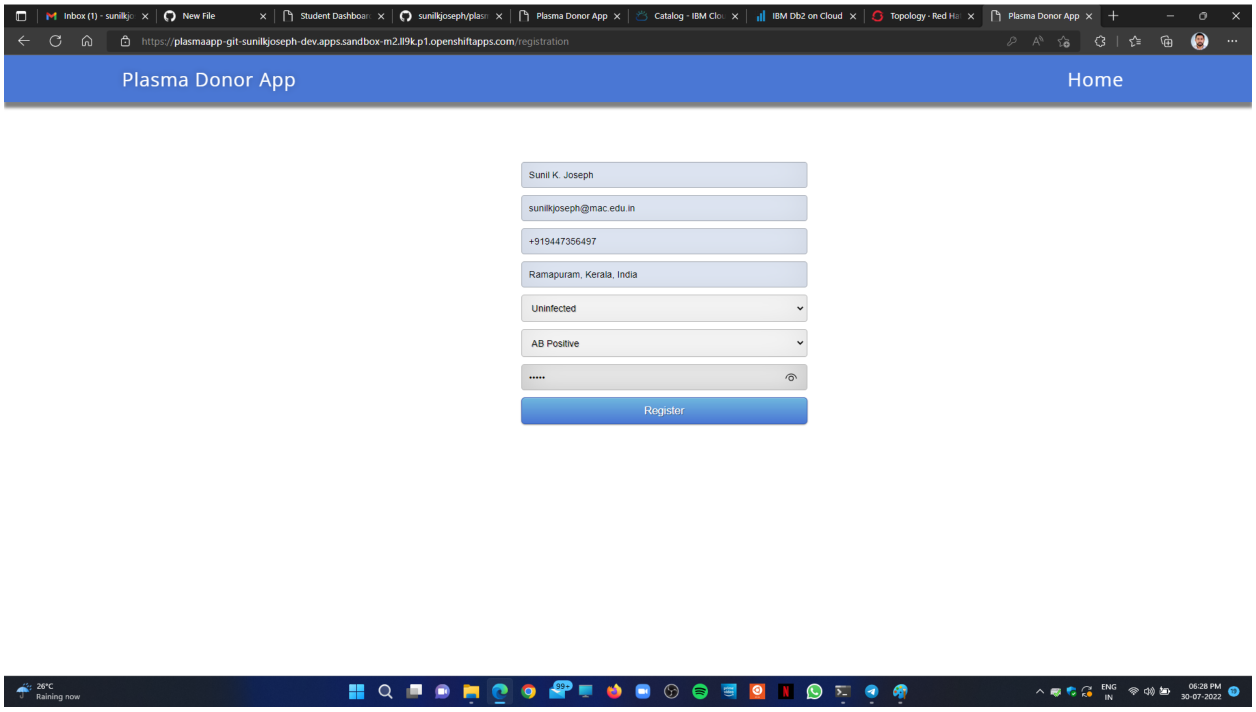


Fig. 6 App login page


https://plasmaapp-git-sunilkjoseph-dev.apps.sandbox-m2.ll9k.p1.openshiftapps.com/registration
Plasma Donor App
Home
Register

Sunil K. Joseph

sunilkjoseph@mac.edu.in

+919447356497

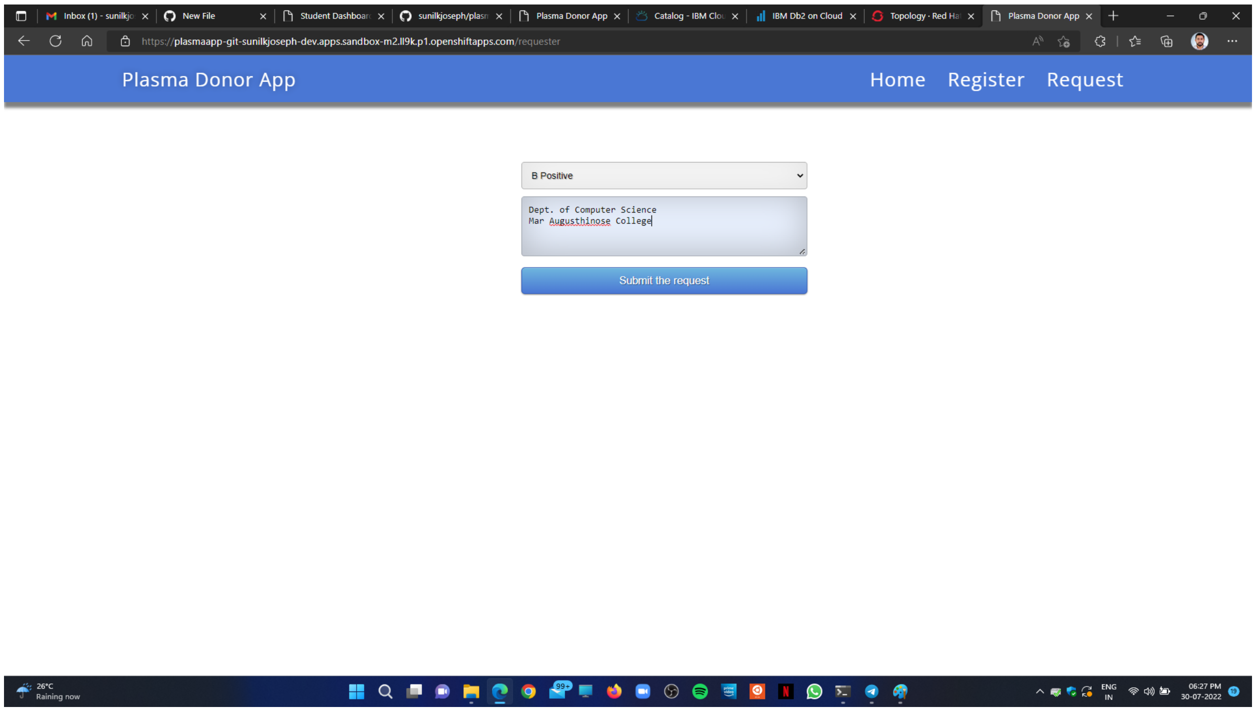
Ramapuram, Kerala, India

Uninfected

AB Positive

•••••

Fig. 7 App registration page


https://plasmaapp-git-sunilkjoseph-dev.apps.sandbox-m2.ll9k.p1.openshiftapps.com/requester
Plasma Donor App
Home
Register
Request
Submit the request

B Positive

```
Dept. of Computer Science
Mar Augusthinose College
```

Fig. 8 Donor request page

6.APPLICATIONS

Plasma Donor application can be used for connecting patients promptly with a large pool of donors in the same area. When a patient needs blood they can use the application to contact the blood donors in the vicinity or nearby city. The registered donors will get notification about the requirement when a request is submitted.

7.CONCLUSION

During the days of COVID-19, it is noticed the increase in blood request posts on social media such as Facebook, Twitter, and Instagram. Interestingly there are many people across the world interested in donating blood when there is a need, but those donors don’t have an access to know about the blood donation requests. This is because that there is no platform to connect blood donors with the person in need. Plasma Donor App solves the problem and creates a communication channel whenever a patient needs blood. It is a useful tool to find compatible blood donors who can receive blood request from different area. Clinics can use this web application to maintain the blood donation activity.

Plasma Donor Application can further improve user accessibility via integrating this application with various social networks application program interfaces (APIs). Consequently, users can login and sign up using various social networks. This would increase number of donors and enhances the process of blood donation.

User interface (UI) can be improved in future to accommodate more audience by supporting different languages across the country.

APPENDIX

Application Source Code

```
from flask import Flask, render_template, request, redirect, url_for, session
import ibm_db
import json
app = Flask(__name__)

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=2d46b6b4-cbf6-40eb-bbce-
6251e6ba0300.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=32328;SECURI
TY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=rzp60332;PWD=pQKhhRt
tcrGEyBKA",'','')

@app.route('/registration')
def home():
    return render_template('register.html')

@app.route('/register',methods=['POST'])
def register():
    x = [x for x in request.form.values()]
    print(x)
    name=x[0]
    email=x[1]
    phone=x[2]
    city=x[3]
    infect=x[4]
    blood=x[5]
    password=x[6]
    sql = "SELECT * FROM user WHERE email =?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,email)
    ibm_db.execute(stmt)
    account = ibm_db.fetch_assoc(stmt)
    print(account)
    if account:
        return render_template('register.html', pred="You are already a
member, please login using your details")
    else:
        insert_sql = "INSERT INTO  user 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, phone)
        ibm_db.bind_param(prep_stmt, 4, city)
        ibm_db.bind_param(prep_stmt, 5, infect)
        ibm_db.bind_param(prep_stmt, 6, blood)
        ibm_db.bind_param(prep_stmt, 7, password)
        ibm_db.execute(prep_stmt)
        return render_template('register.html', pred="Registration
Successful, please login using your details")

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

@app.route('/loginpage',methods=['POST'])
def loginpage():
    user = request.form['user']
    passw = request.form['passw']
    sql = "SELECT * FROM user WHERE email =? AND password=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,user)
    ibm_db.bind_param(stmt,2,passw)
    ibm_db.execute(stmt)
    account = ibm_db.fetch_assoc(stmt)
    print (account)
    print(user,passw)
    if account:
            return redirect(url_for('stats'))
    else:
        return render_template('login.html', pred="Login unsuccessful.
Incorrect username / password !")


@app.route('/stats')
def stats():
    '''sql = "SELECT blood FROM user group by blood"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    count = ibm_db.fetch_assoc(stmt)
    print(count)'''
    return
render_template('stats.html',b=5,b1=2,b2=3,b3=4,b4=2,b5=1,b6=2,b7=1,b8=1)

@app.route('/requester')
def requester():
    return render_template('request.html')


@app.route('/requested',methods=['POST'])
def requested():
    bloodgrp = request.form['bloodgrp']
    address = request.form['address']
    print(address)
    sql = "SELECT * FROM user WHERE blood=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,bloodgrp)
    ibm_db.execute(stmt)
    data = ibm_db.fetch_assoc(stmt)
    msg = "Need Plasma of your blood group for: "+address
    while data != False:
        print ("The Phone is : ", data["PHONE"])

url="https://www.fast2sms.com/dev/bulk?authorization=xCXuwWTzyjOD2ARd1EngbH3a
7tKIq5PklJ8YSf0Lh4FQZecs9iNI1dSvuqprxFwCKYJXA5amQkBE36Rl&sender_id=FSTSMS&mes
sage="+msg+"&language=english&route=p&numbers="+str(data["PHONE"])
        result=requests.request("GET",url)
        print(result)
        data = ibm_db.fetch_assoc(stmt)
    return render_template('request.html', pred="Your request is sent to the
concerned people.")


if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8080)
```