

CAD Project Report

On

Cloud Based COVID plasma Bank System

Developed By: -

Anil Prajapati (18162101027)

Yashvi Gandhi (18082271008)

Yash Pendal (18162101026)

Simran Patel (18162101023)

Guided By:-

Prof. Rahul Shrimali

Prof. Bhavesh Jain

**Submitted to
Department of Computer Science
& Engineering Institute of
Computer Technology
Ganpat University**

Year: 2021

CERTIFICATE

This is to certify that the Cloud Application Development (CAD) Project entitled **“Cloud Based COVID plazma Bank system”** by **Yashvi Gandhi(Enrolment No.18082271008), Anil Prajapati(Enrolment No.18162101027), Yash Pental (EnrolmentNo.18162101026) and Simran Petal (Enrolment No.18162101023)** of Ganpat University, towards the fulfillment of requirements of the degree of Bachelor of Technology – Computer Science and Engineering, is record of bonafide 7th semester - Cloud Application Development (CAD) Project work, carried out by them in the CSE (CBA) Department. The results/findings contained in this Project have not been submitted in part or full to any other University / Institute for award of any other Degree/Diploma.

Name & Signature of Guide

Place:

ICT

GUNI

Date:

ACKNOWLEDGEMENT

CAD project has been a golden opportunity for learning and self-development. We consider ourselves incredibly honored to have so many wonderful people lead us through in completion of this project. First and foremost, we would like to thank all faculty members of Computer Science and Engineering, who gave us an opportunity to undertake this project. We extend our gratitude to **Prof. Rahul Shrimali and Prof. Bhavesh Jain** for their guidance in project work **Cloud Based Covid Plasma Bank System** who mainly guided us from the very beginning till successful completion of our project. CSE department monitored our progress and extended their expertise towards enlightening us wherever needed. We choose this moment to acknowledge their contribution gratefully.

Yashvi Gandhi(Enrolment No.18082271008)

Anil Prajapati(Enrolment No.18162101027)

Yash Pendal (EnrolmentNo.18162101026)

Simran Petal (Enrolment No.18162101023)

ABSTRACT

Convalescent plasma from patients who have already recovered from coronavirus disease 2019 (COVID-19) may contain antibodies against COVID-19. Donation of this plasma to hospitalized people currently fighting COVID-19 may help them recover. Convalescent refers to anyone recovering from a disease. Hence plasma holds high value nowadays and so is the system to maintain and monitor its presence. Applications to maintain records, for authentication, and much more. Application being fully automated, all it needs is just a click for uploading and all its services will be created along with its required condition for smooth flow.

INDEX

Title	Page no
Introduction	
Project Scope	
Cloud Services	
Project plan	
Implementation Details	
Execution	
Conclusion and Future Plan	
Reference	

Chapter 1: Introduction

There has been an increased demand in Convalescent plasma for treating COVID patients and hence people are being encouraged for donation. Convalescent plasma contains antibodies against COVID-19 and not only it is used for treatment but also for observation for the development of vaccines. Acknowledging these benefits and accepting this approach, people have been donating plasma. And that campaign needs a system to carry out all the functionality. From registering to sending requests to get plasma if needed, and from donation to getting certified for it along with authentication, smart traffic management, secured data collection and validity check for authorized people. This project brings all the functionality using cloud services which are fully automated.

Chapter 2: project Scope

This module contains donation, receiving, admin panel etc. which broadens its scope so much that other than its definite usage during COVID-19 pandemic phase, it can be used for generic blood donations are well, and there is no substitute for blood, it can't be manufactured and hence there is always need for it, be it for saving lives or for extraction of plasma or academics, etc. This constant demand for these donations makes this module very useful since it provides receiver, donation, admin phases with authentication, validation and smart data management. This application being fully automated gives ease for deployment as well as for hosting, all client needs to is to upload compressed file of the program and all services will be self created along with all the parameters needed for smooth functioning.

Chapter 3: Cloud Services

EC2 (Amazon Elastic Compute Cloud)
SES(Amazon Simple Email Service)
EBS(Amazon Elastic Block Store)
S3 (Amazon Simple Storage Service)
SQS(Amazon Simple Queue Service)
Lambda
Cloud watch
IAM(AWS Identity and Access Management)
RDS(Amazon Relational Database Service)
Cloud Formation

Chapter 4: Implementation Details

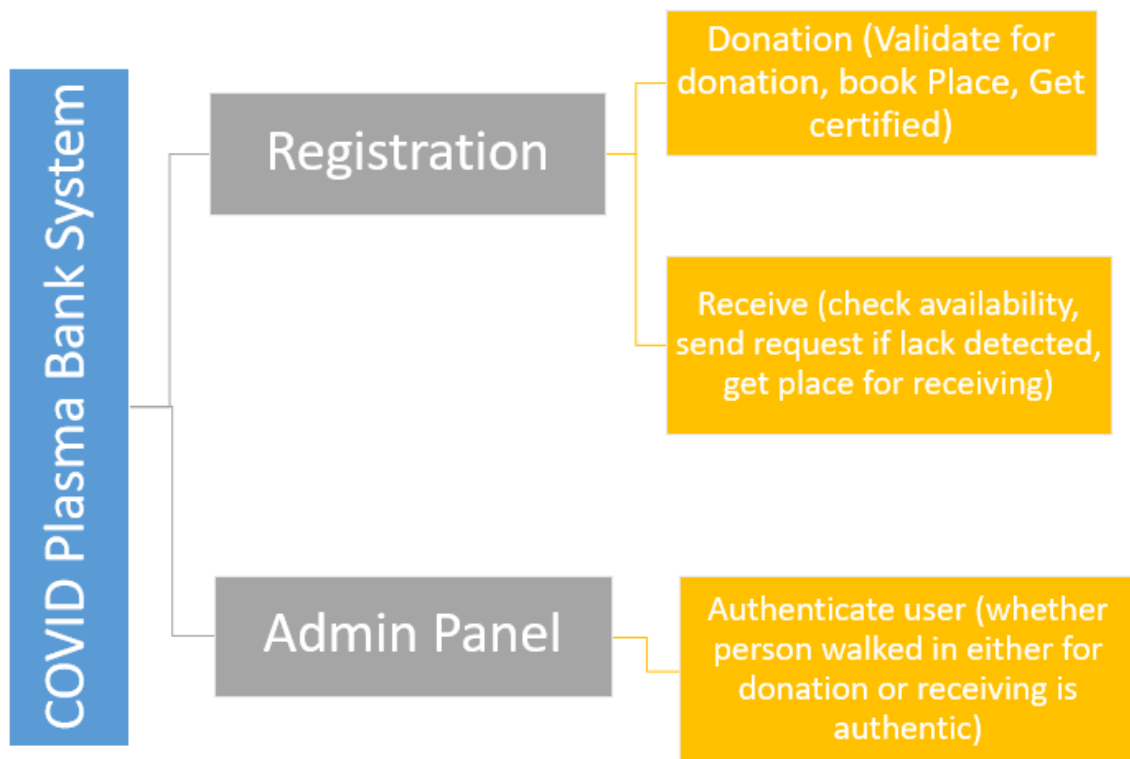


Figure 1: flowchart

Our project has been carried out in the Django framework using python where each class has been attached to HTML for UI and

fields for databases have been directed towards storing the inputs that users put in.

Donation and receiving modules

```
cad-project-submission-2018batch-group_10_covid_plazma-master > BankApp > models.py
1  from django.db import models
2  from django.contrib.auth.models import User
3
4  # Create your models here.
5
6
7  class Banks(models.Model):
8      Bcity = models.CharField(max_length=100)
9      BState = models.CharField(max_length=100)
10     def __str__(self):
11         return self.Bcity
12
13     class DonnerModel(models.Model):
14         fullname = models.CharField(max_length=100)
15         Did = models.ForeignKey(User, on_delete=models.CASCADE)
16         phone_no = models.IntegerField()
17         BloodGrp = models.CharField(max_length=20)
18         Age = models.IntegerField()
19         is_diabitic = models.BooleanField()
20         Bid = models.ForeignKey(Banks, on_delete=models.CASCADE)
21         donnote_date = models.DateField()
22         is_recived_by_bank = models.BooleanField(default=False)
23         Secret = models.CharField(max_length=100,default="SGN1ONS")
24
25     def __str__(self):
26         return self.fullname
27
```

```

class Blood(models.Model):
    Did = models.ForeignKey(User, on_delete=models.CASCADE)
    Bid = models.ForeignKey(Banks, on_delete=models.CASCADE)
    BloodGrp = models.CharField(max_length=20)
    Secret = models.CharField(max_length=100)
    is_available = models.BooleanField(default=True)

class RequesterModel(models.Model):
    fullname = models.CharField(max_length=100)
    Rid = models.ForeignKey(User, on_delete=models.CASCADE)
    phone_no = models.IntegerField()
    BloodGrp = models.CharField(max_length=20)
    Age = models.IntegerField()
    is_diabitic = models.BooleanField()
    def __str__(self):
        return self.fullname

class RequestNoBlood(models.Model):
    RNid = models.ForeignKey(User, on_delete=models.CASCADE)

class RequestedBlood(models.Model):
    Rqid = models.ForeignKey(User, on_delete=models.CASCADE)
    Dnid = models.IntegerField()
    Bid = models.ForeignKey(Banks, on_delete=models.CASCADE)
    Bloodid = models.ForeignKey(Blood, on_delete=models.CASCADE)
    Secret = models.CharField(max_length=100)

```

Model for admin panel

```

cad-project-submission-2018batch-group_10_covid_plazma-master > BankApp > admin.py
1  from django.contrib import admin
2
3  # Register your models here.
4
5  from .models import Banks, Blood, RequesterModel , DonnerModel ,RequestedBlood
6
7  @admin.register(Banks)
8  class RequestDemoAdmin(admin.ModelAdmin):
9      list_display = [field.name for field in
10 Banks._meta.get_fields()]

```

Now lets individually understand each functionality

```

def index(request):
    return HttpResponse("Hello, world. Site is under Cunstruction")

def home(request):
    return render(request, 'home.html')

def GetSecret():
    return ''.join(random.choices(string.ascii_uppercase +
    string.digits, k = 7))

```

Index :- if some functionality are yet to be made or if modifications are worked upon, index will be called to pass on the message

Home:- gets directed to home page

GetSecret:- generation of random varchar for authentication

```

def getAuthenticateEmail(email):
    sqs = boto3.client('sqs')

    # Send message to SQS queue
    response = sqs.send_message(
        QueueUrl=queue_url,
        DelaySeconds=10,
        MessageAttributes={
            'email': {
                'DataType': 'String',
                'StringValue': email
            },
            'is_secret': {
                'DataType': 'String',
                'StringValue': "no"
            }
        },
        MessageBody=(
            'sgnons'
        )
    )
    print("\n\n\n\n\n")
    print(response['MessageId'])

```

getAuthenticateEmail- Validity of email is checked since that input will be crucial for sending authentication password, email generation

your Blood Donation is Booked on 2021-08-29 at 1

Plase Bring this secret with you while donate blood

VTE1JE0

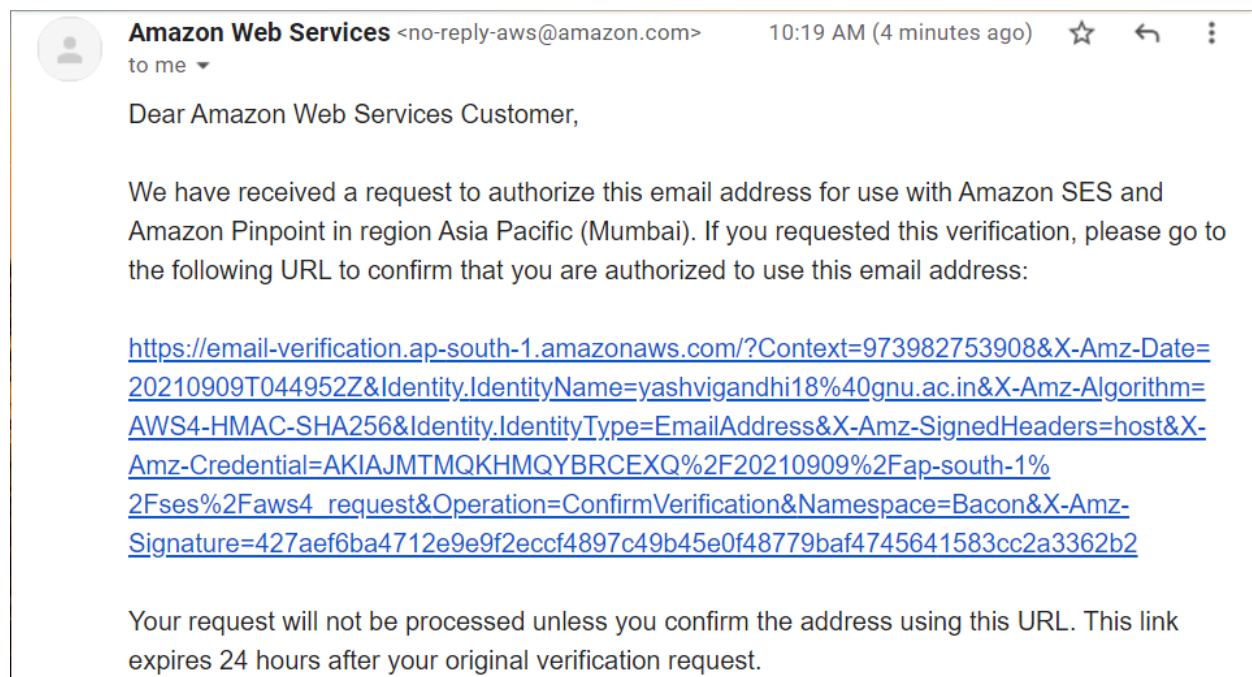
Mail sent for code in order to validate

your Blood Donation Certificate :

https://s3-prac5-bucket.s3.ap-south-1.amazonaws.com/certi/Yashvi_Gandhi.html

Plase Bring this secret with you while donate blood

Mail sent for authentication



Congratulations!

You have successfully verified an email address. You can now start sending email from this address.

For new Amazon SES users—If you have not yet applied for a sending limit increase, then you are still in the [sandbox environment](#), and you can only send email to addresses that have been verified. To verify a new email address or domain, see the **Identity Management** section of the [Amazon SES console](#).

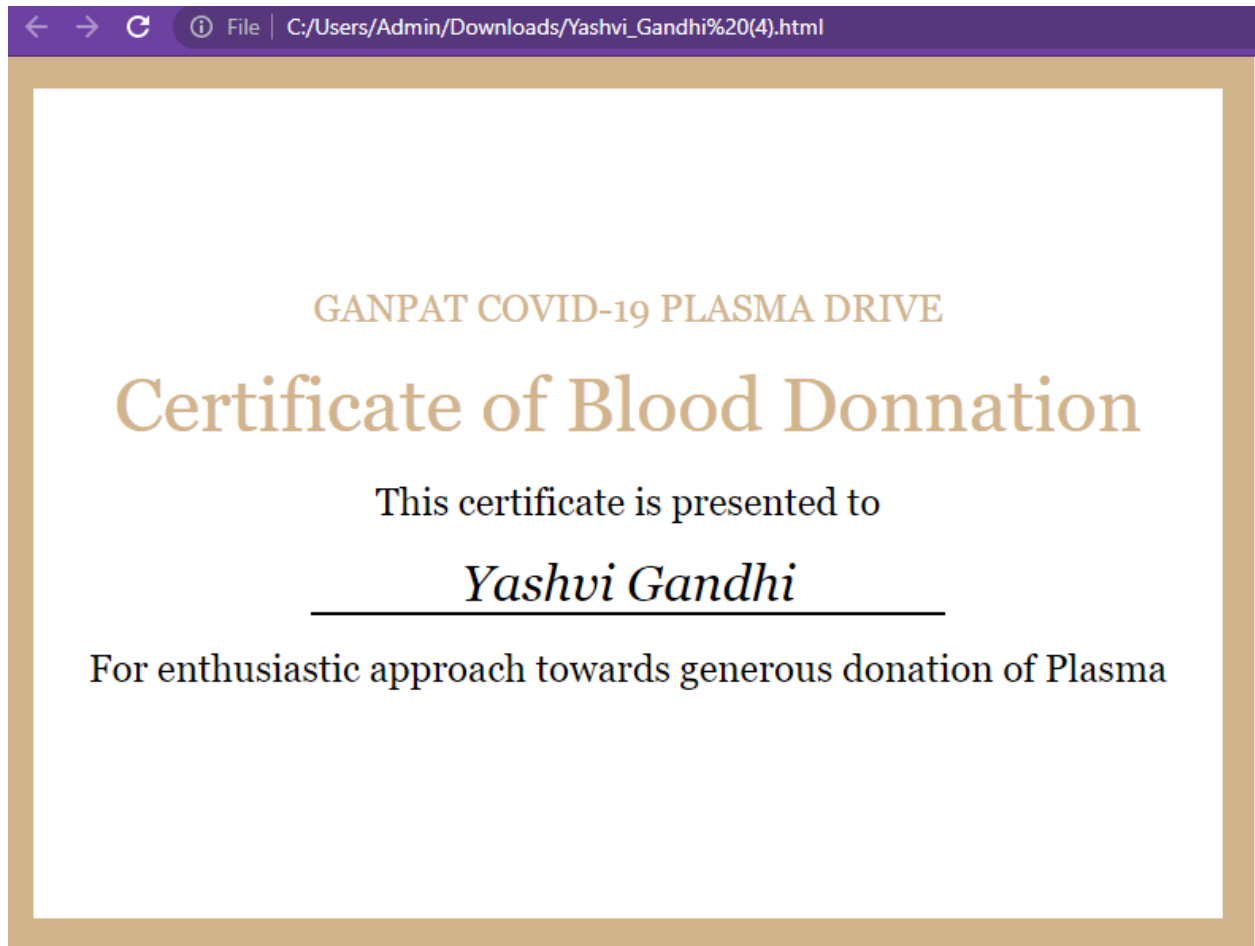
For new Amazon Pinpoint users—If you have not yet applied for a sending limit increase, then you are still in the [sandbox environment](#), and you can only send email to addresses that have been verified. To verify a new email address or domain, see the **Settings > Channels** page on the [Amazon Pinpoint console](#).

If you have already been approved for a sending limit increase, then you can start sending email to non-verified addresses.

Thank you for using Amazon Web Services!

[Go to the Amazon SES detail page.](#)

Mail sent for certification



To call SQS service

```

def callSQS(Secret,email,Ddate,Baddress):
    sqs = boto3.client('sqs')

    #queue_url = 'https://sqs.ap-south-1.amazonaws.com/675270067251/SgnCovidPlazmaSQS'

    # Send message to SQS queue
    response = sqs.send_message(
        QueueUrl=queue_url,
        DelaySeconds=10,
        MessageAttributes={
            'Secret': {
                'DataType': 'String',
                'StringValue': Secret
            },
            'email': {
                'DataType': 'String',
                'StringValue': email
            },
            'Ddate': {
                'DataType': 'String',
                'StringValue': str(Ddate)
            },
            'is_secret': {
                'DataType': 'String',
                'StringValue': "yes"
            },
            'Baddress':{
                'DataType': 'String',
                'StringValue': str(Baddress)
            }
        }
    ),

```

This is for registering new user


```

def register(request):
    print(1)
    if request.method == "POST":
        print(2)
        form = UserRegistrationForm(request.POST)
        if form.is_valid():
            print(3)
            getAuthenticateEmail(request.POST['email'])
            user = form.save()
            login(request, user)
            messages.success(request, "Registration successful." )
            return redirect("home")
        messages.error(request, "Unsuccessful registration. Invalid information.")
    form = UserRegistrationForm()
    return render (request=request, template_name="registration/register.html", context={"register_form":form})

def getBankList():
    l=[]
    bobj = Banks.objects.all()
    for i in bobj:
        l.append([i.id,i.Bcity])
    return l

```

Donation module consist of the form for donation and code will be sent to user using SQS via lambda triggeration

```

def Donner(request):
    if request.method == "POST":
        print("\n\n\n sgn 0")
        form = DonnerForm(request.POST)
        if int(request.POST['Age']) < 18:
            return render (request=request, template_name="Donner.html", context={"register_form":form,
                                                                                     "isShow":False,
                                                                                     "Blist":getBankList(),
                                                                                     "SecVal":GetSecret(),
                                                                                     "Emess":"you can't Donnate Blood"})

        donnatedate = request.POST['donnate_date']
        donnatedate = donnatedate.split('-')
        newdate = datetime.date(int(donnatedate[0]),int(donnatedate[1]),int(donnatedate[2]))
        print("\n\n",donnatedate)
        newdate=relativedelta( newdate , datetime.datetime.today())
        print(newdate.years ," ",newdate.days )
        if newdate.years < -1 or newdate.months < -1 or newdate.days < -1:
            return render (request=request, template_name="Donner.html", context={"register_form":form,
                                                                                     "isShow":False,
                                                                                     "Blist":getBankList(),
                                                                                     "SecVal":GetSecret(),
                                                                                     "Emess":"Please choose date correctly"})

        print("for save")
        if form.is_valid():
            print("\n\n\n sgn 1")
            form.save()
            #email secret values to user.
            callsQS(request.POST['Secret'],request.user.email,request.POST['donnate_date'],request.POST['Bid'])
            messages.success(request, "Registration successful." )
            return render(request,'home.html',context={"text":"Thank You for Donation",

```

```

        print("for save")
        if form.is_valid():
            print("\n\n\n sgn 1")
            form.save()
            #email secret values to user.
            callSQS(request.POST['Secret'],request.user.email,request.POST['donnate_date'],request.POST['Bid'])
            messages.success(request, "Registration successful." )
            return render(request,'home.html',context={"text":"Thank You for Donation",
                                                         "SecVal":request.POST['Secret']})
#
#         B=Blood(BloodGrp=request.POST['BloodGrp'],
#                 Bid_id=request.POST['Bid'],
#                 Did=request.user,
#                 Secret=GetSecret(),
#                 is_available=True)
#         print("\n\n\n sgn 2")
#         B.save()
donner1 = DonnerModel.objects.filter(Did_id=request.user.id)
if len(donner1) == 1:
    return redirect(is_donner_already_donated)
print("\n\n\n sgn 5")
form = DonnerForm()
return render (request=request, template_name="Donner.html", context={"register_form":form,
                                                                    "isShow":False,
                                                                    "Blist":getBankList(),
                                                                    "SecVal":GetSecret(),
                                                                    "form":form},
                )

```

```

#if user already created donner id do code here
def is_donner_already_donated(request):
    donner1 = DonnerModel.objects.filter(Did_id=request.user.id)
    dateNow = donner1[0].donnate_date
    a_month = relativedelta(months=6)
    date_plus_month = dateNow + a_month
    if request.method == "GET":
        return render(request=request, template_name="Donner_already.html",context={
                                                                    "Ddate":date_plus_month })

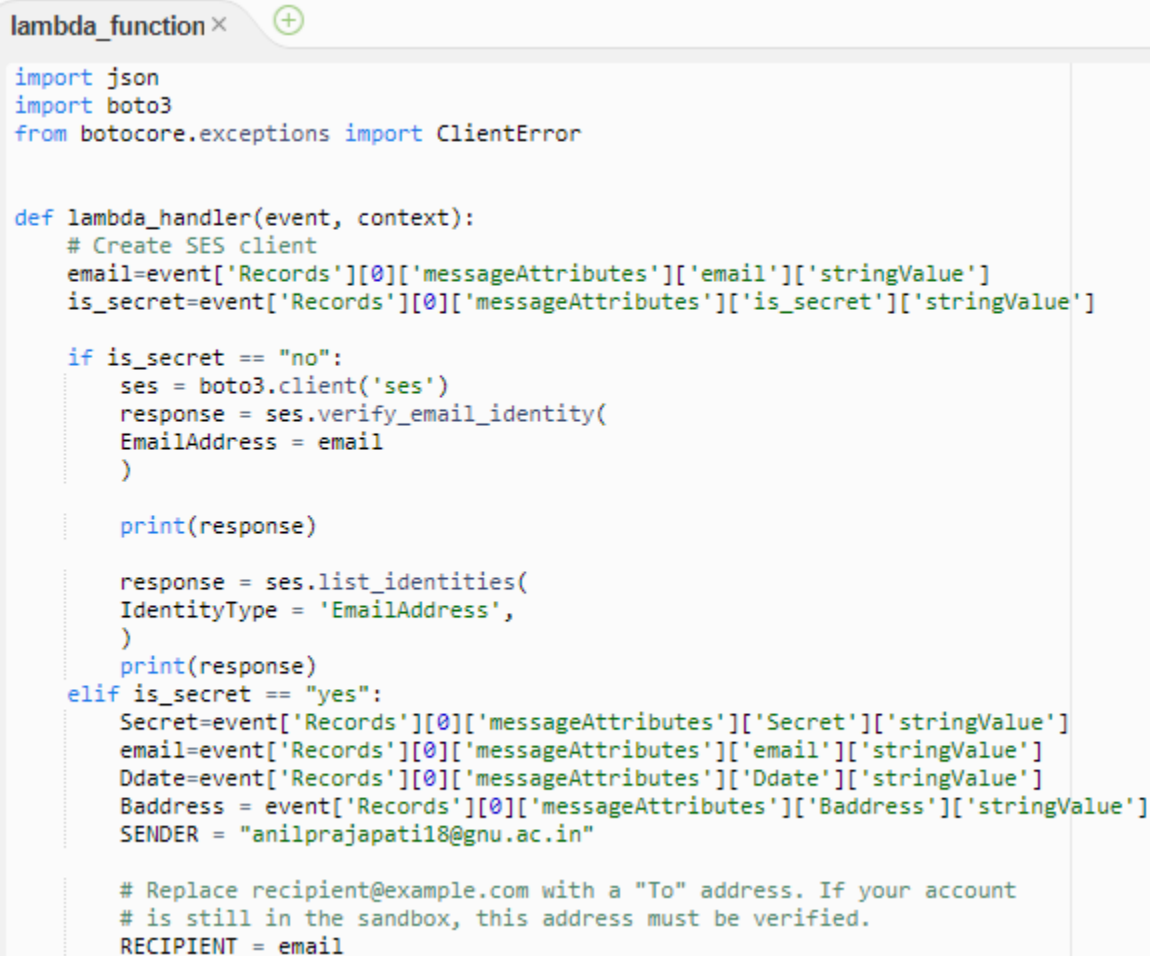
    if request.method == "POST":
        if donner1[0].Did_id==request.user.id:
            print("\n\n",donner1[0].BloodGrp," ",)
            dateNow=donner1[0].donnate_date
            print(request.POST['DonnerData'])
            donnatedate = request.POST['DonnerData']
            donnatedate = donnatedate.split('-')
            newdate = datetime.date(int(donnatedate[0]),int(donnatedate[1]),int(donnatedate[2]))
            print("\n\n",donnatedate)
            newdate=relativedelta( newdate , date_plus_month )
            print(newdate.years , " ",newdate.months , " ",newdate.days)
            if newdate.years > -1 and newdate.months > -1 and newdate.days > -1:
                addBlood = DonnerModel.objects.get(Did=request.POST['id1'])
                addBlood.is_recived_by_bank = False
                addBlood.Secret = GetSecret()
                addBlood.donnate_date = request.POST['DonnerData']
                addBlood.save()
                #call sqs
                callSQS(addBlood.Secret,request.user.email,request.POST['DonnerData'],addBlood.Bid)
                return render(request,'home.html',context={"text":"Thank You For Donation Plase Save Secret" ,
                                                                    "Ddate" :addBlood.Secret})

```

This is to authenticate if the person has already donated within 6 months, if so then donation will be declined and user will be suggested with valid date after which their donation will be accepted

```
else:
    return render(request, 'Donner_already.html', context={"text": "Sorry You have to wait till" , "Ddate" : date_p
return render(request, 'home.html', context={"text": ""})
```

Triggering of lambda if donation is validated for sending them authentication password



```
lambda_function × (+)

import json
import boto3
from botocore.exceptions import ClientError

def lambda_handler(event, context):
    # Create SES client
    email=event['Records'][0]['messageAttributes']['email']['stringValue']
    is_secret=event['Records'][0]['messageAttributes']['is_secret']['stringValue']

    if is_secret == "no":
        ses = boto3.client('ses')
        response = ses.verify_email_identity(
            EmailAddress = email
        )

        print(response)

        response = ses.list_identities(
            IdentityType = 'EmailAddress',
        )
        print(response)
    elif is_secret == "yes":
        Secret=event['Records'][0]['messageAttributes']['Secret']['stringValue']
        email=event['Records'][0]['messageAttributes']['email']['stringValue']
        Ddate=event['Records'][0]['messageAttributes']['Ddate']['stringValue']
        Baddress = event['Records'][0]['messageAttributes']['Baddress']['stringValue']
        SENDER = "anilprajapati18@gnu.ac.in"

        # Replace recipient@example.com with a "To" address. If your account
        # is still in the sandbox, this address must be verified.
        RECIPIENT = email
```

During donation, consumer would be asked with the password and authorised person has to fill in the password for authenticating the donor

Activities Firefox Web Browser Sep 9 10:52 AM

Bot Read x New Tab x +

127.0.0.1:8000/BankAdminPage

Home	Hiii admin!	Admin	Log Out
------	-------------	-------	---------

Name	Phone No	Blood Grp	Age	Is_diabetic	Secret	Receieved	
yashvi gandhi	1991919191	AB+	21	No	<input type="text" value="Secret"/>	No	<input type="button" value="Receieved"/>

© 2021 Copyright: Covid Plasma Grp10

```
def BankAdminPage(request):
    if request.method == "POST":
        print(request.POST["Secret"],"\n\n\n")
        print(request.POST['id1'])
        addBlood = DonnerModel.objects.get(id=request.POST['id1'])
        if addBlood.Secret == request.POST['Secret']:
            addBlood.is_recived_by_bank = True
            addBlood.save()
            SendBankCertificate(request.POST['fullname'],request.POST['Did'])
            #send email to user with certificate.
            DonnerData = DonnerModel.objects.filter(is_recived_by_bank=0)
            return render (request=request, template_name="BankAdmin.html", context={"DonnerData":DonnerData})
        else:
            DonnerData = DonnerModel.objects.filter(is_recived_by_bank=0)
            return render (request=request, template_name="BankAdmin.html", context={"DonnerData":DonnerData,
                                                                                      "Perror":"Plase Check Secret and enter again" })

    DonnerData = DonnerModel.objects.filter(is_recived_by_bank=0)
    return render (request=request, template_name="BankAdmin.html", context={"DonnerData":DonnerData})
```

After successful donation, certificate will be generated, saved in AWS S3 bucket and sent by SQS via Lambda trigger

```

def SendBankCertificate(fullname,Did):
    u1 = User.objects.get(id=Did)
    email = u1.email
    sqs = boto3.client('sqs')

    #queue_url = 'https://sqs.ap-south-1.amazonaws.com/675270067251/SgnCovidPlazmaSQS'

    # Send message to SQS queue
    s3_client = boto3.client('s3')
    html = open("certi/sgncerti.html")

    # Parse HTML file in BeautifulSoup
    soup = bs(html, 'html.parser')

    # Give location where text is
    # stored which you wish to alter
    old_text = soup.find("div", {"id": "fullname"})

    # Replace the already stored text with
    # the new text which you wish to assign
    new_text = old_text.find(text=re.compile(
        'sgnons')).replace_with(fullname)
    fullname = fullname.replace(" ", "_")
    new_file_name="certi/"+str(fullname)+".html"
    # Alter HTML file to see the changes done
    with open(new_file_name, "wb") as f_output:
        f_output.write(soup.prettify("utf-8"))

# hti = Html2Image()
# new_png=fullname+".png"
# hti.screenshot(
#     html_file=new_file_name,

```

Ln 267, Col 39 Ta

```

response = s3_client.upload_file(new_file_name, "s3-prac5-bucket", new_file_name)
print("\n\nfile uploaded")
response = sqs.send_message(
    QueueUrl=queue_url,
    DelaySeconds=10,
    MessageAttributes={
        'filename': {
            'DataType': 'String',
            'StringValue': str(new_file_name)
        },
        'email': {
            'DataType': 'String',
            'StringValue': email
        },
        'is_secret': {
            'DataType': 'String',
            'StringValue': "certi"
        }
    },
    MessageBody=(
        'sgnons'
    )
)

print(response['MessageId'])

```

Lambda trigger where s3 bucket is mounted

```

elif is_secret == 'certi':
    print(event)
    filename=event['Records'][0]['messageAttributes']['filename']['stringValue']
    email=event['Records'][0]['messageAttributes']['email']['stringValue']
    SENDER = "anilprajapati18@gnu.ac.in"
    # Replace recipient@example.com with a "To" address. If your account
    # is still in the sandbox, this address must be verified.
    RECIPIENT = email

    # Specify a configuration set. If you do not want to use a configuration
    # set, comment the following variable, and the
    # ConfigurationSetName=CONFIGURATION_SET argument below.
    CONFIGURATION_SET = "ConfigSet"

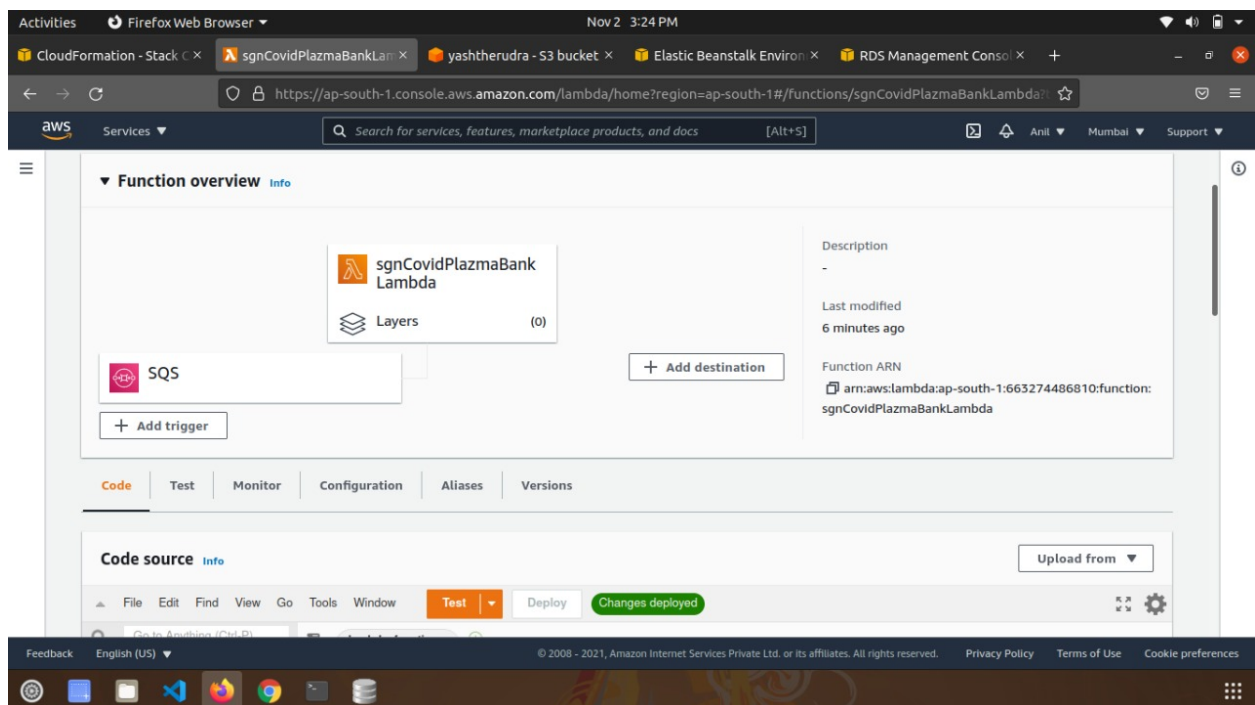
    # If necessary, replace us-west-2 with the AWS Region you're using for Amazon SES.
    AWS_REGION = "ap-south-1"

    # The subject line for the email.
    SUBJECT = "Amazon SES Test (SDK for Python)"

    # The email body for recipients with non-HTML email clients.
    BODY_TEXT = ("Amazon SES Test (Python)\r\n"
                 "This email was sent with Amazon SES using the "
                 "AWS SDK for Python (Boto).")

    filename = "https://s3-prac5-bucket.s3.ap-south-1.amazonaws.com/"+filename
    # The HTML body of the email.
    BODY_HTML = """<html>
<head></head>
<body>
  <h1>Covid Plazma Bank </h1>
  <p> your Blood Donnation Certificate : </p> <p>"""+ filename+""""
  <p>Please bring this password during donation</p>

```



Requester

```
def Requester(request):
    request.session['Bloodgrp'] = ""
    if request.method == "POST":
        form = RequesterForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, "Registration successful." )
            request.session['Bloodgrp'] = request.POST['BloodGrp']
            print("dgn done1")
            return redirect("RequesterBloodIsAvailable")
        messages.error(request, "Unsuccessful registration. Invalid information.")
    req1=RequesterModel.objects.filter(Rid=request.user)
    if len(req1) == 1:
        return redirect("RequesterBloodIsAvailable")
    form = RequesterForm()
    return render (request=request, template_name="Requester.html", context={"register_form":form,"isShow":False})
```

Whenever there is need for blood, availability can be checked and asked for. If there is lack of resources to fulfil the need of blood, then email will be sent asking for that particular type of blood

```
def Requester(request):
    request.session['Bloodgrp'] = ""
    if request.method == "POST":
        form = RequesterForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, "Registration successful." )
            request.session['Bloodgrp'] = request.POST['BloodGrp']
            print("dgn done1")
            return redirect("RequesterBloodIsAvailable")
        messages.error(request, "Unsuccessful registration. Invalid information.")
    req1=RequesterModel.objects.filter(Rid=request.user)
    if len(req1) == 1:
        return redirect("RequesterBloodIsAvailable")
    form = RequesterForm()
    return render (request=request, template_name="Requester.html", context={"register_form":form,"isShow":False})
```

This is the condition if blood will not be in stock

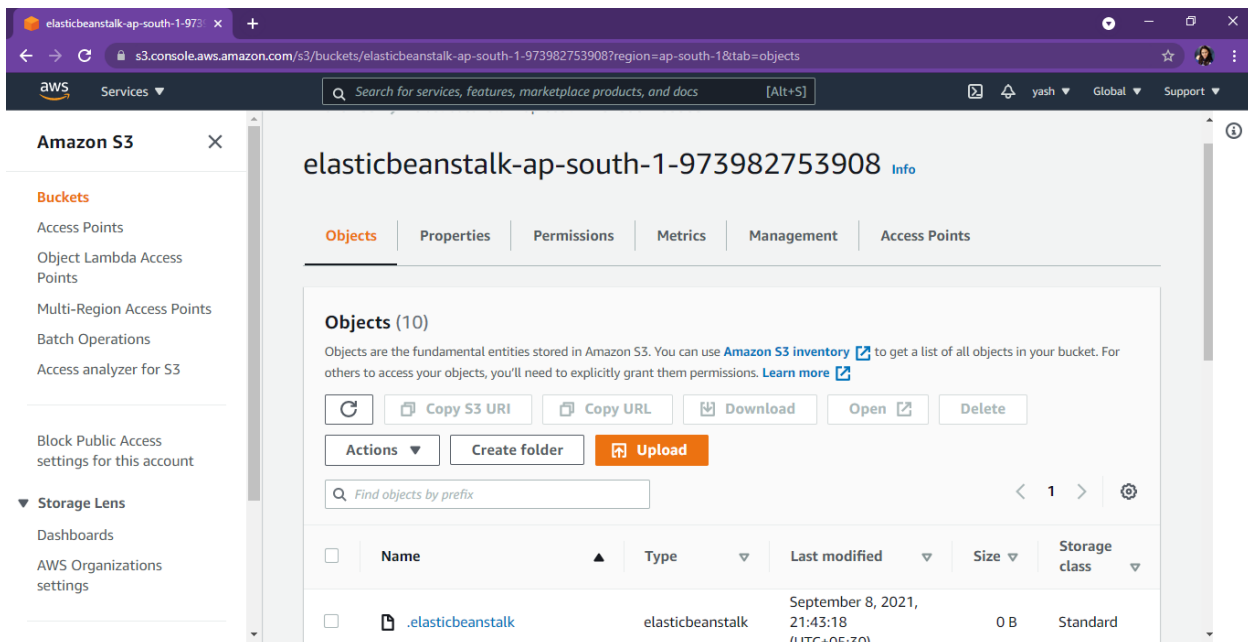

```
# make condition if blood is not present

def RequesterBloodIsAvailable(request):
    print("dgn done2")
    if request.method == "POST":
        print("dgn done3")
        DonnerList = Blood.objects.filter(BloodGrp=request.POST['BloodGrp'],is_available=1)
        if len(DonnerList)>0:
            print("dgn done4")
            #now just give the first blood but after add date and give oldest blood
            print(DonnerList[0].Bid.id," ",type(DonnerList[0].Bid)," ",str(DonnerList[0].Bid))
            BankDetails = Banks.objects.get(id=DonnerList[0].Bid.id)

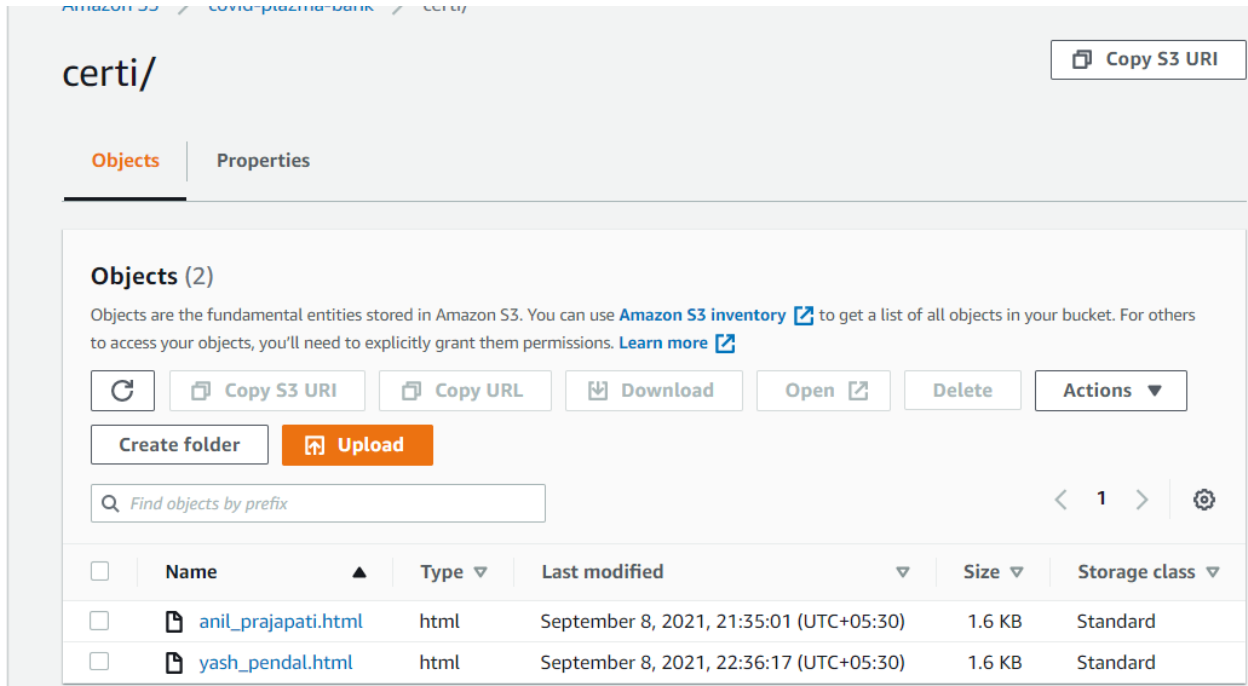
            DonnerData = [DonnerList[0].BloodGrp,BankDetails.Bcity,BankDetails.BState]
            #add here a data on which date blood will given.
            ReqObj = RequestedBlood(Dnid=DonnerList[0].Did.id,
                                   Bid=BankDetails,
                                   Rqid=request.user,
                                   Secret=GetSecret(),
                                   Bloodid=DonnerList[0])

            ReqObj.save()
            print("\n\n",DonnerList[0].BloodGrp," ",BankDetails.Bcity," ",BankDetails.BState,"\n\n")
            BloodD = Blood.objects.get(id=DonnerList[0].id)
            BloodD.is_available = False
            BloodD.save()
            return render (request=request, template_name="RequesterForBlood.html", context={"DonnerData":DonnerData,"is_da
return render (request=request, template_name="RequesterForBlood.html", context={"BloodGrp":request.session['Bloodgrp']
```

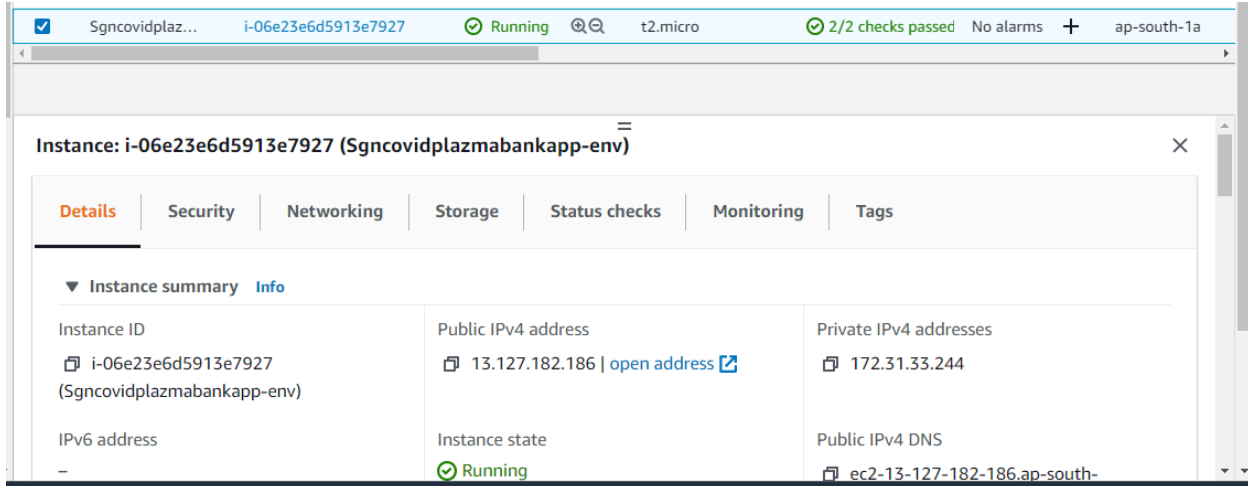
This application will be fully on web via EBS

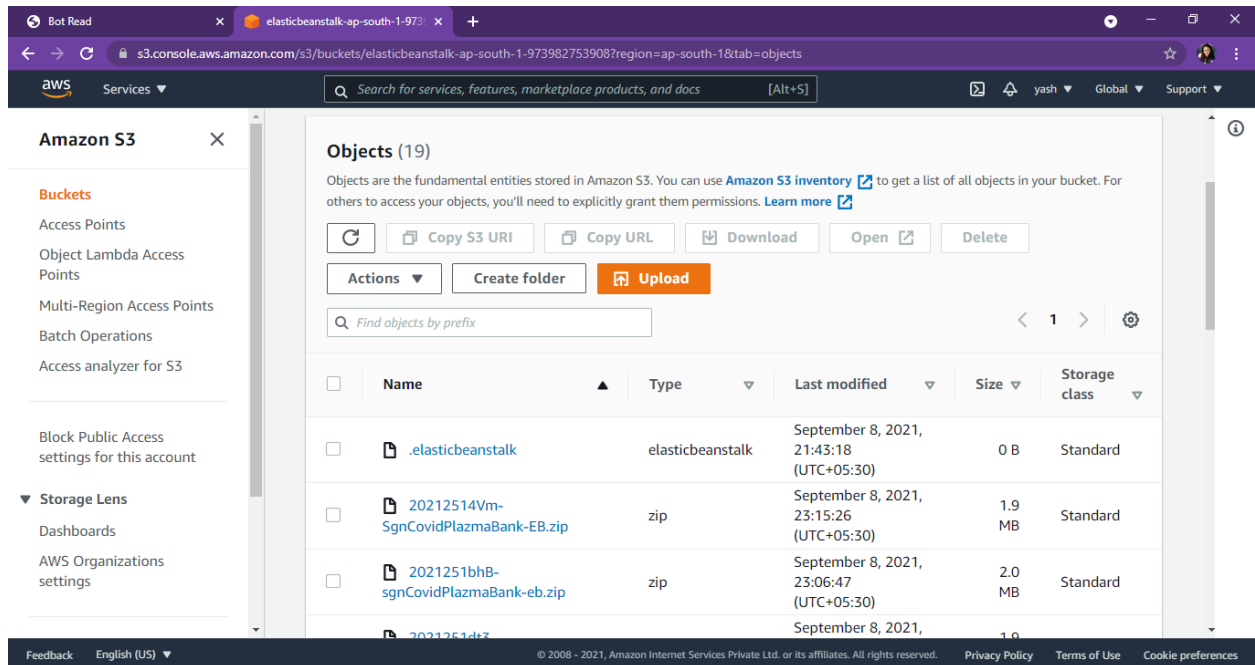


After generation of certificate, it will be first stored in s3 bucket and from there it will be given access to

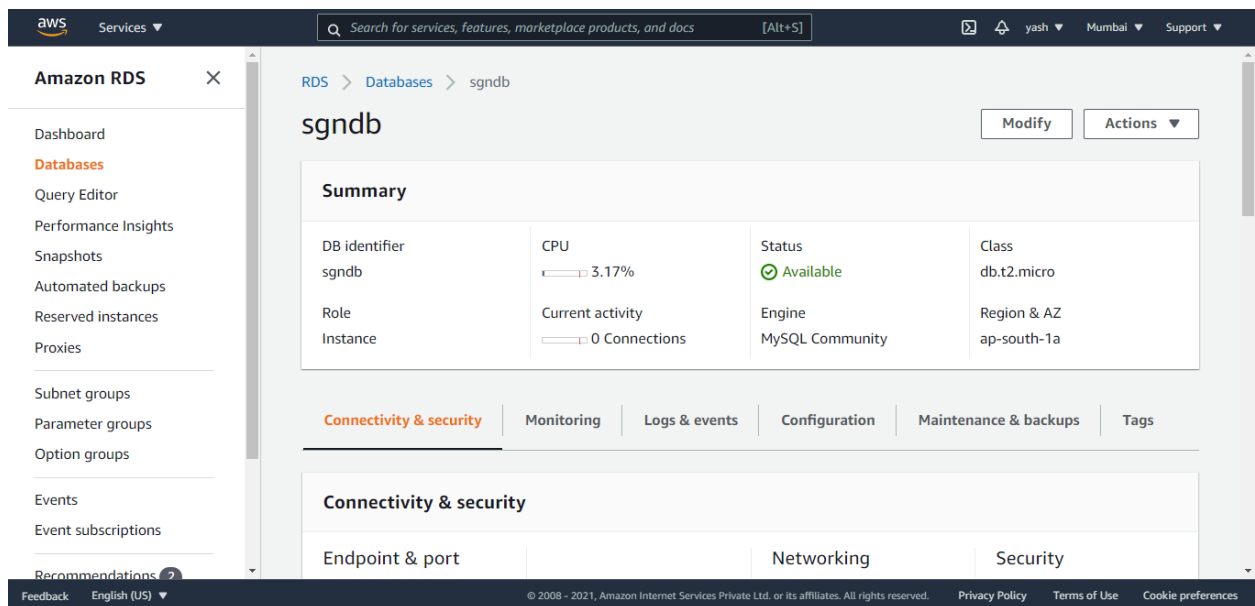


EC2 instance has been created for deployment with total avoidance of downtime and project has been fully deployed on EBS

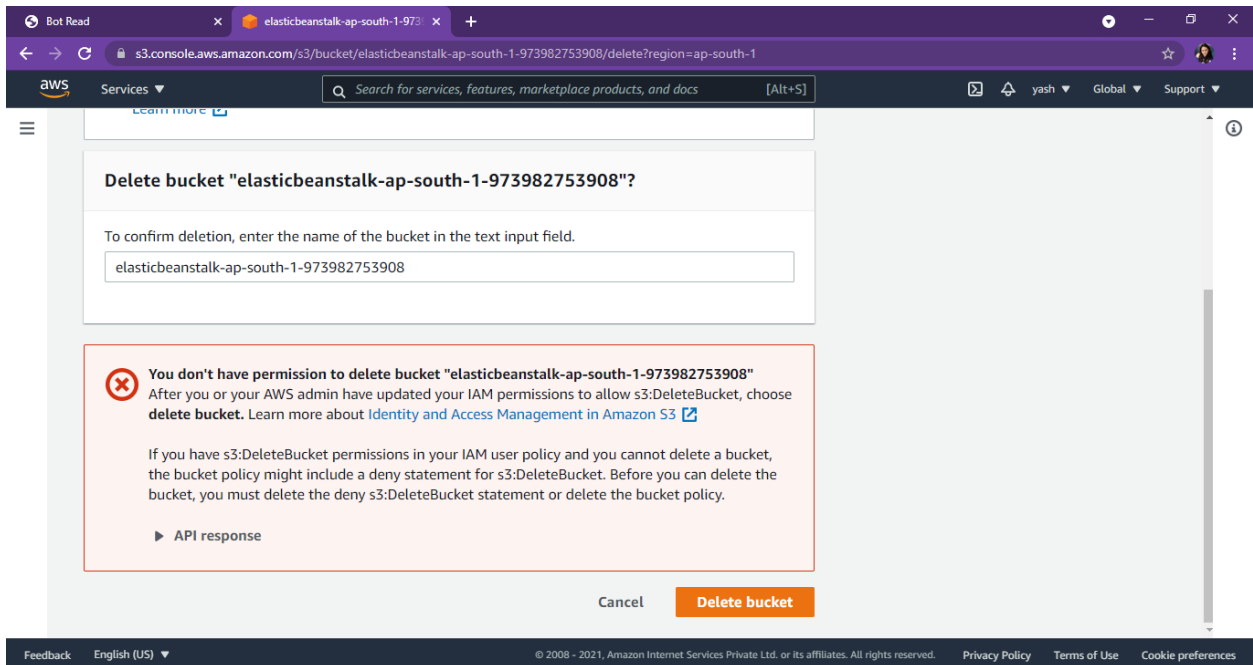




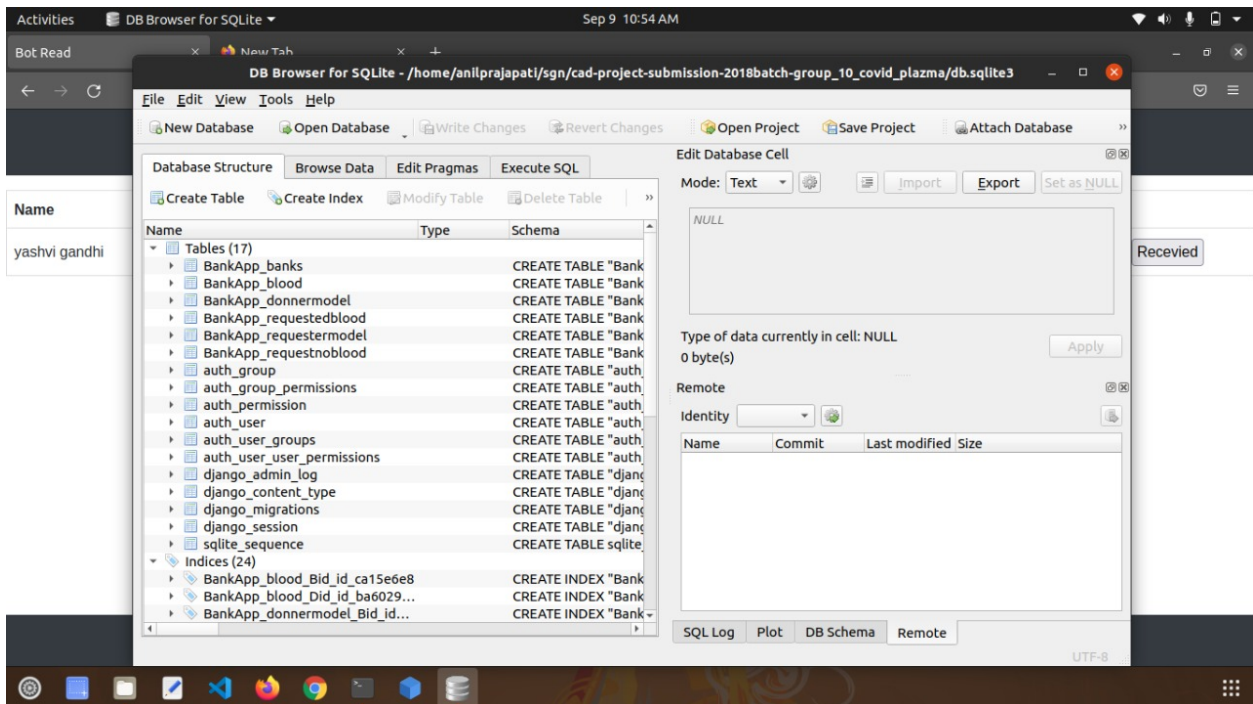
All the modules has been attached to RDS for database which in indeed worked upon on SQL workbench locally



Also there has been usage of IAM for accessing the project in accordance with the amount of permission allotted to them



And hence this not only authenticates but also avoids unnecessary modifications

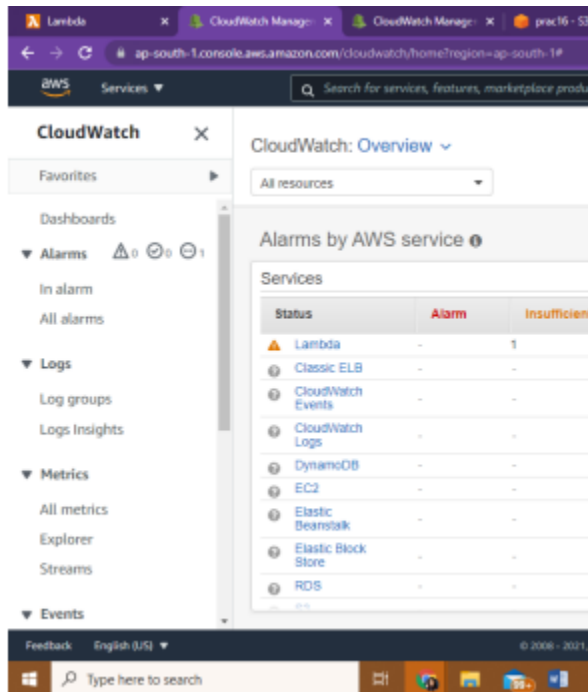


Cloud watch (alarm triggering)

Whenever stress is increased on the server depending upon the traffic, on the basis of CPU utilization, parameter of the

threshold, if satisfied, will lead to an alarm triggered and our project makes sure that the alert is sent on email as well.

Similarly it will also trigger itself when lambda function is called (can be during certificate generation or sending notification to users for the need of particular blood)



All these services should be automatically created through cloud formation. From EB, hosting of site to all S3 buckets, connection of emails through SNS(along with email verification), to all EC2 instances along with autoscaling facility and alarm through cloud watch to send alert of any type of glitch.

```

"Description": "Creates ASG with Specified Min, Max, and desired capacity in each specified subnet",
"Parameters": {
  "MaxSizeASG": {
    "Description": "Enter the Max Size for the ASG",
    "Type": "String"
  },
  "MinSizeASG": {
    "Description": "Enter the Min Size for the ASG",
    "Type": "String"
  },
  "DesiredCapacityASG": {
    "Description": "Enter the desired capacity for the ASG",
    "Type": "String"
  },
  "VPCZoneIdentifier": {
    "Description": "List the Subnet Ids of the instances",
    "Type": "List<AWS::EC2::Subnet::Id>"
  },
  "KeyName": {
    "Description": "EC2 instance key name",
    "Type": "String"
  },
  "InstanceType": {
    "Description": "EC2 instance type",
    "Type": "String"
  },
  "InstanceSecurityGroup": {
    "Description": "List of Security Group IDs",
    "Type": "List<AWS::EC2::SecurityGroup::Id>"
  }
},
"Mappings": {
  "RegionMap": {
    "us-east-1": {
      "HVM64": "ami-0ff8a91507f77f867"
    },
    "us-west-1": {
      "HVM64": "ami-0bdb828fd58c52235"
    }
  }
}

```

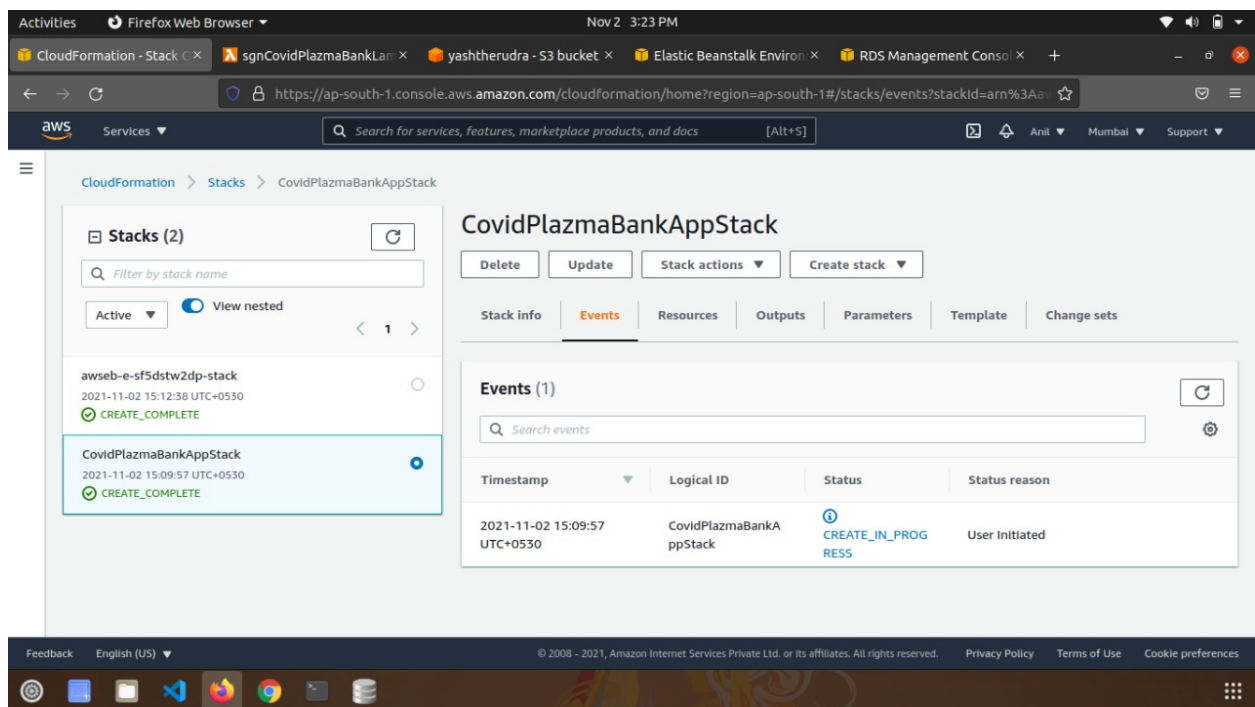
```

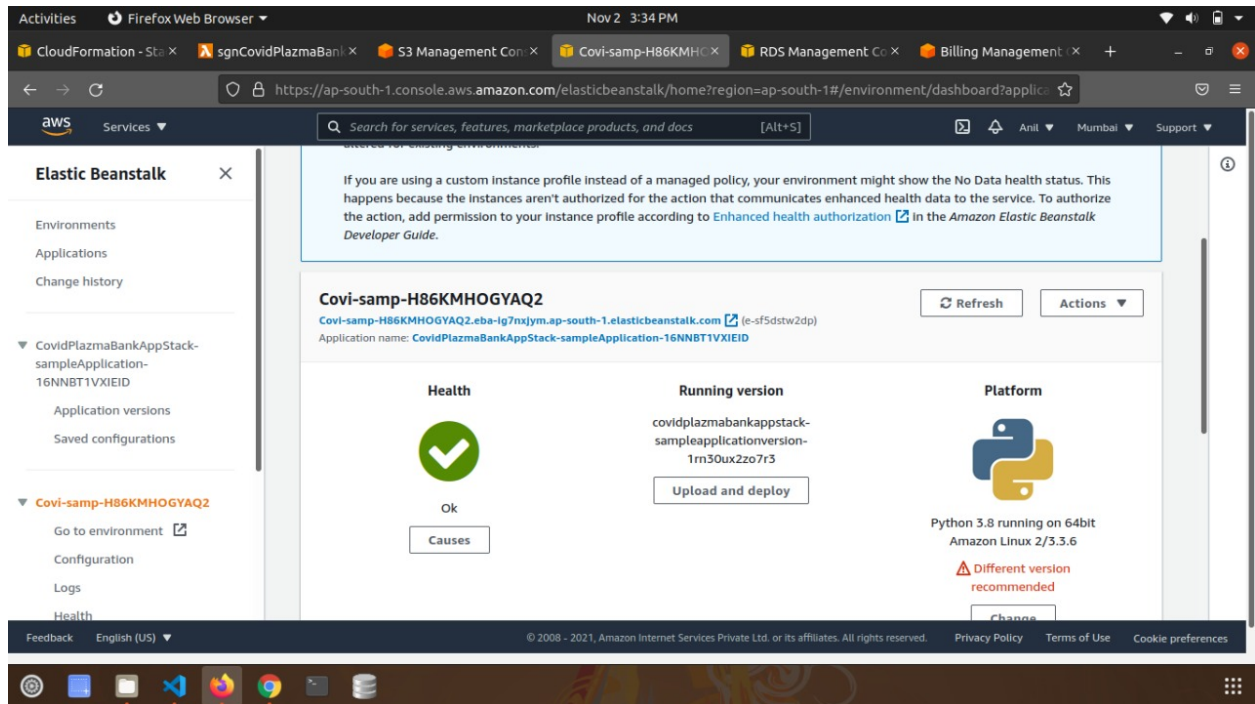
    "LaunchConfig": {
      "Type": "AWS::AutoScaling::LaunchConfiguration",
      "Properties": {
        "KeyName": {
          "Ref": "KeyName"
        },
        "ImageId": {
          "Fn::FindInMap": [
            "RegionMap",
            {
              "Ref": "AWS::Region"
            },
            "HVM64"
          ]
        },
        "SecurityGroups": {
          "Ref": "InstanceSecurityGroup"
        },
        "InstanceType": {
          "Ref": "InstanceType"
        },
        "LaunchConfigurationName": "GarrettCfnLaunchConfiguration",
        "UserData": {
          "Fn::Base64": {
            "Fn::Sub": "=/bin/bash\\nsudo su\\nsudo apt update\\nsudo"
          }
        }
      }
    }
  },
  "Outputs": {
    "AutoscalingGroup": {
      "Description": "The newly created asg",
      "Value": {
        "Ref": "AutoScalingGroup"
      }
    },
    "LaunchConfig": {
      "Description": "the newly created launch config",
      "Value": {
</pre

```

Chapter 5: Execution

Just compress the file into zip and upload it to cloud formation in AWS cloud, from EB hosted link to all EC2 instance with autoscaling group to roles and its policy, such that servers will be allotted in accordance to the traffic and formation of server would be increased and decreased with respect to audience live on the application





Libraries needed for execution of this projects are

asgiref==3.4.1

beautifulsoup4==4.9.3

boto3==1.18.25

botocore==1.21.25

bs4==0.0.1

Django==3.2.5

html2image==2.0.1

jmespath==0.10.0

numpy==1.21.2

pandas==1.3.2

Pillow==8.3.1

python-dateutil==2.8.2

pytz==2021.1

s3transfer==0.5.0

six==1.16.0

soupsieve==2.2.1

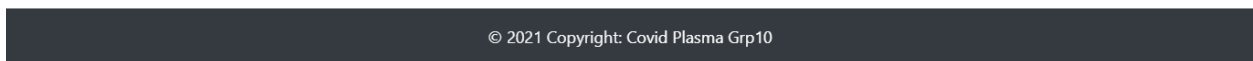
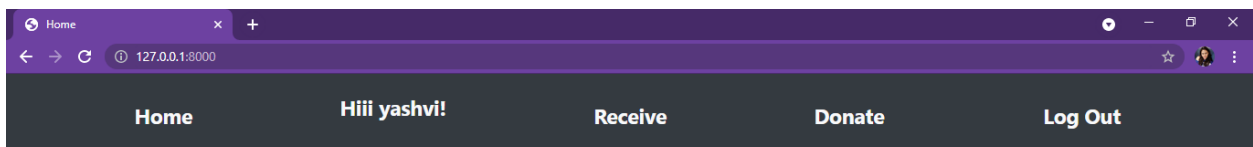
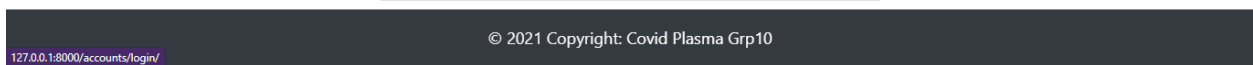
sqlparse==0.4.1

urllib3==1.26.6



Login

New User ?



Bot Read x +

127.0.0.1:8000/Requester

Donate Blood

Fullname

Contact Number

Is diabetic: ☐

Blood Group

Enter your Age

City

State

© 2021 Copyright: Covid Plasma Grp10

Chapter 6: conclusion and future plan

All in all we conclude that this an all-inclusive application for any donation drive and considering it has been developed in python via django, it is platform independent, hence can be worked on via any platform. Furthermore, in future if any modifications are required and any additional feature added in then it can be very easily adjusted into this project.

Chapter 7: Reference

<https://stackoverflow.com>

<https://docs.aws.amazon.com>

<https://aws.amazon.com/cli/>

<https://docs.djangoproject.com/en/3.2/>