# BLOOD BANK MANAGEMENT SYSTEM USING CLOUD COMPUTING

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF

**BACHELOR OF TECHNOLOGY**
IN
**COMPUTER SCIENCE & ENGINEERING**

SUBMITTED BY


| Name | University Roll No |
|---|---|
| Aditya Jha | 10800118131 |
| Priyank Bhattacharyya | 10800118077 |
| Gourav Thakur | 10800118102 |
| Ranit Pal | 10800118069 |


UNDER THE GUIDANCE OF
**MR. UDDALOK SEN**
Assistant/Associate Professor



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
**ASANSOL ENGINEERING COLLEGE**
AFFILIATED TO
MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY


June, 2022

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**
**ASANSOL ENGINEERING COLLEGE**
Vivekananda Sarani, Kanyapur,Asansol, West Bengal – 713305

## *Certificate of Recommendation*

I hereby recommend that the preliminary thesis report entitled, "**Blood Bank Management System using Cloud Computing**" carried out under my supervision by the group of students listed below may be accept in partial fulfillment of the requirement for the degree of "Bachelor of Technology in **Computer Science & Engineering**" of Asansol Engineering College under MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY.

| Name | University Roll No |
|---|---|
| Aditya Jha | 10800118131 |
| Priyank Bhattacharyya | 10800118077 |
| Gourav Thakur | 10800118102 |
| Ranit Pal | 10800118069 |

……..…….
Mr. Uddalok Sen,
Thesis Supervisor,
Department of Computer
Science & Engineering,
Asansol Engineering College,
Asansol – 713305

Countersigned:

……..…….
Dr.Debasis Chakraborty,
Head of the Department,
Department of Computer
Science & Engineering,
Asansol Engineering College,
Asansol – 713305

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
ASANSOL ENGINEERING COLLEGE
Vivekananda Sarani, Kanyapur,Asansol, West Bengal – 713305

## *Certificate of Approval*

The forgoing thesis is hereby approved as creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it is submitted.

……..…….

Mr. Uddalok Sen,
Thesis Supervisor,
Department of Computer
Science & Engineering,
Asansol Engineering College,
Asansol – 713305

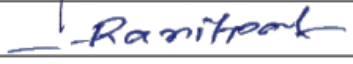
# *Acknowledgment*

It is our great privilege to express our profound and sincere gratitude to our Project Supervisor, **Mr. Uddalok Sen** for providing us with very cooperative and precious guidance at every stage of the present project work being carried out under his/her supervision. His valuable advice and instructions in carrying out the present study has been very rewarding and guidance at every stage of the present project work being carried out under his/her supervision. pleasurable experience that has greatly benefited us throughout the course of work.

We would like to convey our sincere gratitude towards **Dr. Debasis Chakraborty**, Head of the Department of **Computer Science & Engineering** for providing us the requisite support for timely completion of our work. We would also like to pay our heartiest thanks and gratitude to all the teachers of the Department of **Computer Science & Engineering**, Asansol Engineering College for various suggestions being provided in attaining success in our work.

We would like to express our earnest thanks to **Mr. Suman Mallick**, of CSE Project Lab for his technical assistance provided during our project work.

Finally, I would like to express my deep sense of gratitude to my parents for their constant motivation and support throughout my work.

| Student's name | Student's signature |
|---|---|
| Aditya Jha |  |
| Priyank Bhattacharyya |  |
| Gourav Thakur |  |
| Ranit Pal |  |

# *Abstract*

Web-based Blood Bank Management System is a management tool that enables individuals who want to donate blood to help the people in need. It also enables hospitals to record and store the data for people who want to communicate with them, and it also provides a centralized blood bank database. The system is developed using React, Node JS and MongoDB as a database system to manage and store the data.

The system focuses on two types of users: the public who wants to donate blood and the recipients who need the donated blood.

This project is built to serve those in urgent need of blood. Our system is designed for donors to access and register themselves, and for people to search and find their donors. Our platform helps those in need of blood to find and connect with their donors. We store all the donor information in our database systems and deliver to the recipient on their request.

The main purpose of this project revolves around easing the daily and regular processes of a blood donation system into an automated computerized retrieval process.

The main objective of this project is to provide the solution for all the problems that may arise during urgent need of blood. Our system tries to minimize shortage of all groups of blood, and makes sure that they are available in the time of need.

Our systems maintain a centralized blood bank database which helps us to store and record the data of donors and provides them to those who need donation at any given time.

# *Contents*

## *List of figures*



## *List of tables*

# 1. Preface

## 1.1. Introduction

Blood Bank Management System which is built to serve those in need of blood. It manages and maintains the record of all the blood donors across the country and delivers to the recipient in time of need.
Individuals, NGOs, Hospitals can enroll themselves for donation and anyone in need of blood could get in touch easily through our website.
This web application has been made with a user friendly interface, so that anyone can use it and as a security we have provided admin username and password.

## 1.2. Motivation of the Project

In the current scenario recipients sometimes find it hard to get their required blood type of blood in the correct amount in their area and it sometimes becomes too chaotic in the time of need for the people in need. So to find a solution to this problem, we have designed and developed a digital platform where anyone can easily access their profiles and know the best deal of their required blood in their nearby area. With the help of our web application we are trying to reach out and help to the masses and organizations who are trying to save someone who is in urgent need of blood for their survival.

## 1.3. Basic Description of the Project

This project is built to serve those in urgent need of blood. Our system is designed for donors to access and register themselves, and for people to search and find their donors. Our platform helps those in need of blood to find and connect with their donors. We store all the donor information in our database systems and deliver to the recipient on their request.
The main purpose of this project revolves around easing the daily and regular processes of a blood donation system into an automated computerized retrieval process.
The main objective of this project is to provide the solution for all the problems that may arise during urgent need of blood. Our system tries to minimize shortage of a certain group of blood, and makes sure blood of any group is available in time of emergency.
Our systems maintain a centralized blood bank database which helps us to store and record the data of donors and provides them to those who need donation at any given time.

## 2. Literature review

### 2.1. General

The project was built keeping in mind the problems that someone who urgently needs blood for donation, faces. We are trying to minimize all kinds of problems that may arise during urgent need of blood. The main idea of this project came from Drew *et. al., 2017.*

The main objective of this project is to provide the solution for all the problems that may arise during urgent need of blood. Our system tries to minimize shortage of a certain group of blood, and makes sure blood of any group is available in time of emergency.

This website is designed for users to be easily accessible by the users. Our system is designed for donors to access and register themselves, and for people to search and find their donors. Our platform helps those in need of blood to find and connect with their donors. We store all the donor information in our database systems and deliver to the recipient on their request.
The main purpose of this project revolves around easing the daily and regular processes of a blood donation system into an automated computerized retrieval process.
The main objective of this project is to provide the solution for all the problems that may arise during urgent need of blood. Our system tries to minimize shortage of a certain group of blood, and makes sure blood of any group is available in time of emergency.

With the help of Amazon Web Services, users from anywhere in the world can access our portal. Not only that, our website is capable of handling users who are concurrently trying to access our portal. Our website is running in http://http://54.184.111.82:3000/.

### 2.2. Review of Related works

We have scientifically and systematically reviewed the website designed under the patient name of Bhudev Network Charitable Foundation (2022). They have a user login/sign up system, they have an amazing searching system where users can search for their donors and can apply many search filters. Apart from that, users can post requests asking for donations. They also have chat rooms where users can chat with their donors. They provide a centralized donor database from where users can search for donors all over the country. Also, their website provides useful information about Blood Donation, such as tips, scientific information, facts, etc. Their website is truly serving mankind.

## 3. Related theories and algorithms

### 3.1. Fundamental theories underlying the work

This project is based on MERN (MongoDB, Express, React, Node JS) architecture. This project is built to solve problems of blood donation and minimize the shortage of any group of blood. There are two aspects of blood donation, the donors and the recipients. We are providing a platform to users in the form of a website, where users can log on and can find their donors, as well as for donors to log on and create a request that they want to donate their blood.
We are managing the user data in our centralized database, and performing get or fetch queries upon request from the user.
The project is deployed in an EC2 instance in Amazon Web Services, any user, anywhere in the world can access our project.

Following are a list of packages and libraries used for this project:

| No. | Name | Description |
|---|---|---|
| 1 | CSS | For designing |
| 2 | HTML | For the framework |
| 3 | Javascript | For the functions |
| 4 | React | React components helping us to display the website to the users |
| 5 | React-Router-Dom | For creating routes and navigation |
| 6 | Axios | For sending http request |
| 7 | Express | For connection between front-end and back-end |
| 8 | Mongoose | For sending queries to database |
| 9 | MongoDB | For storing user data |
| 10 | GitHub | Repository for the project |
| 11 | Amazon Web Services | For hosting the project |

Table 3.1.1

### 3.2. Fundamental Algorithms

Our project has two fundamental parts, the web application and the cloud server. There are three tiers to the web application namely, the front end, the back end and the middleware. The front end has been designed using NodeJS, React, HTML and CSS. React components including HTML and CSS with help of NodeJS modules constitute the main framework for the front end. In the middleware part, with

the help of Express server, we are able to connect the front end with the database server. The MongoDB database server running on port 27017, helps us to store the user data.

Our project is deployed and running in Amazon Web Services, anyone, anywhere in the world can access our website. Here are the fundamental algorithms with the help of which, this project was successful.

1. Creating the web-application

We are building a web-application based on MERN(MongoDB, Express, React, Node JS).

2. Push the project code into a github repository using git bash

To enable the EC2 instance to GET our code.
Git is a software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows.
With the help of github, we are able to maintain a centralized repository of the project.

3. Create an EC2 instance in Amazon Web Services

Create an EC2 instance, and configure the instance.

4. Create a login key to safely login to the instance

A safety key is required to safely login into the instance. Retain the key file as everytime we need to remotely connect to the instance to perform maintenance, we will need this key file.
This safety key is required because without it anyone could access our instance, which might lead to security issues.

5. Choose the system image and storage size of the instance

We have used Ubuntu 18 as our system image and 20 gigabyte of storage size for the instance.

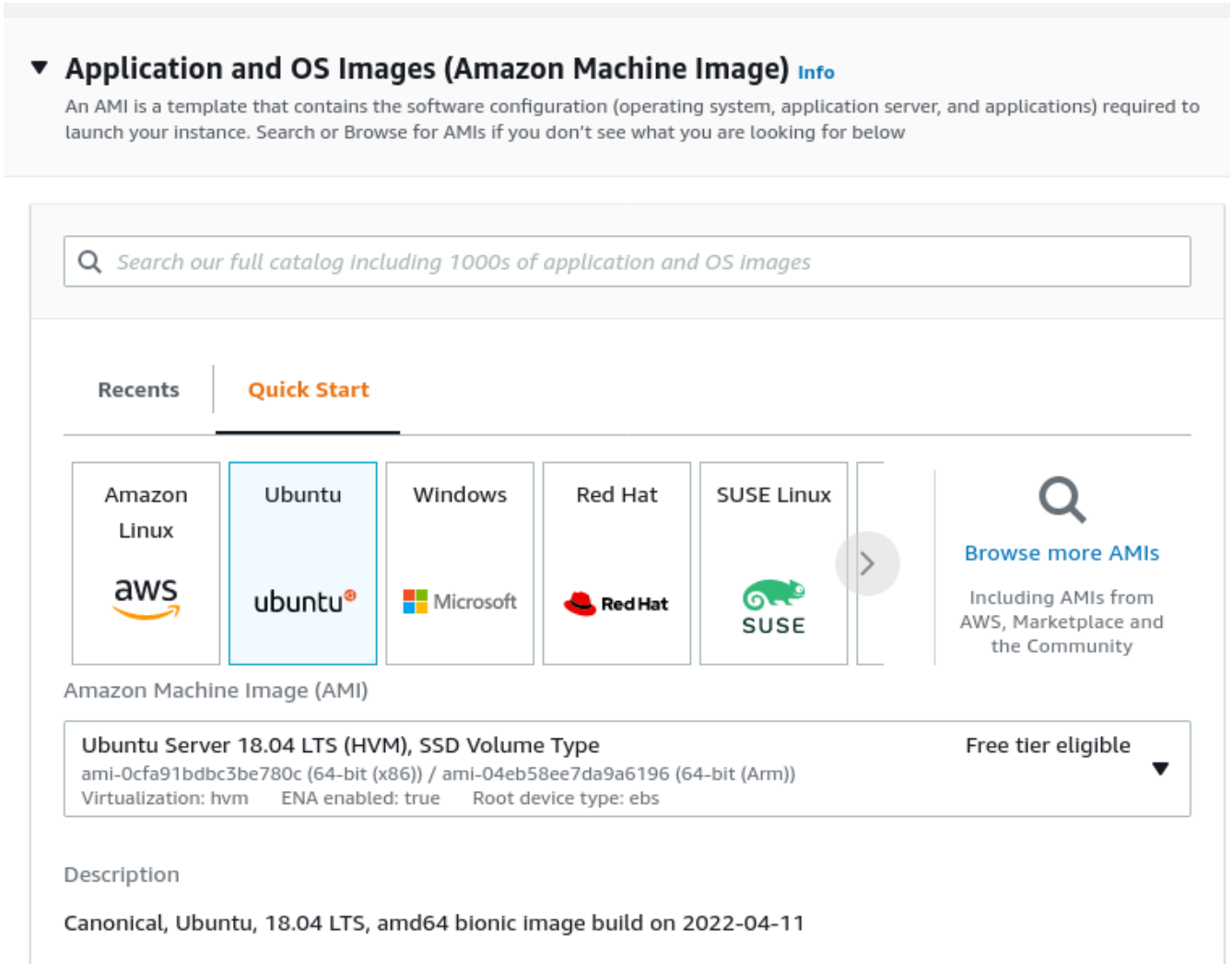



Figure 3.2.1

Ubuntu is the world's most popular cloud operating system across public clouds. With its security, versatility and policy of regular updates, Ubuntu is the leading cloud guest OS and the only free cloud operating system with the option of enterprise-grade commercial support. Ubuntu has all the services we would need and that is the reason why we decided to use Ubuntu.

6. Allow HTTP/HTTPs traffic

Allow HTTP/HTTPS traffic for users to access our application.

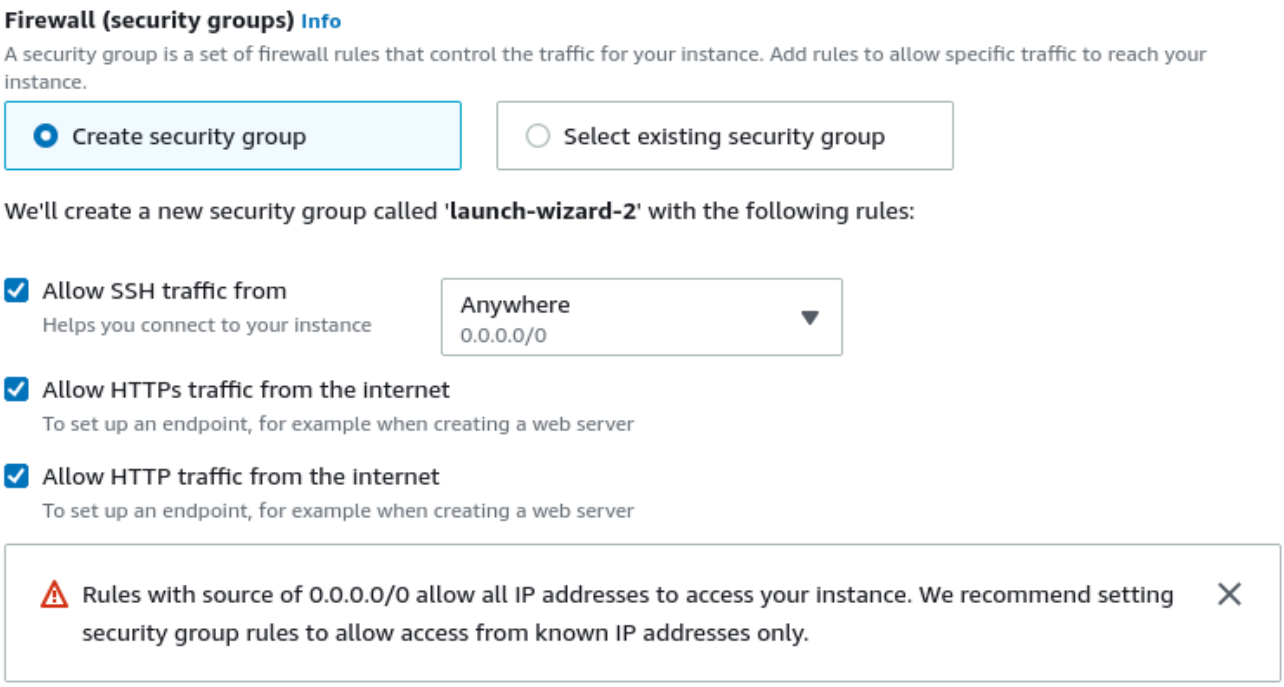



Figure 3.2.2

7. Remotely connect to the instance

After creating the instance, we need to remotely connect to the instance, with the help of the security key and the following line of code, we can remotely connect to the instance.

```
ssh -i "bloodbank.pem" ubuntu@ip_address
```

8. Install required softwares in the instance

Install the necessary software like NodeJs, MongoDB in the instance.

```
sudo apt install mongodb

curl -fsSL https://deb.nodesource.com/setup_current.x | sudo -E bash -
sudo apt-get install -y nodejs
```

9. Clone the project from the github repository

Clone the project from the github repository.

10. Install all dependent node modules

Install all the node modules with the help of the following npm function

npm install

11. Set outbound rules for the instance

Add 3000,4000,27017 to the outbound rules for the instance. These ports are blocked by default in the instance so they are needed to be added so our users can access the application.

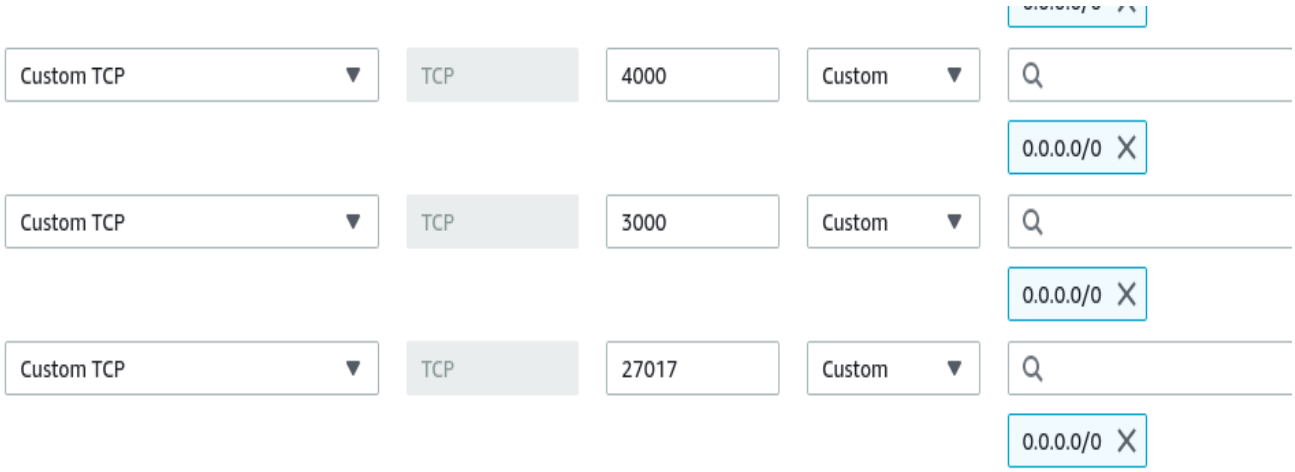



Figure 3.2.3

12. Start the server

Start the front end server using

```
npm start
```

Start the database server

```
npx nodemon index.js
```

## 4. Proposed models/algorithms

### 4.1. Proposed models

For the front-end, we have used Javascript, CSS and HTML inside React components. React-Router-Dom, which is a Node JS package, is helping us to create routes and navigate to the different pages in our website. Express server in the back-end is connecting the front-end to the MongoDB database. Any requests from the front-end are sent to the express server using the Axios package (Axios is a promise based http client for browser Node JS). Express server establishes the communication from the back-end with the database, and upon receiving any requests from the front-end, is sending requests to perform various query operations in the database server with the help of mongoose(MongoDB driver) and returning the data that the user has asked for. The NPM(Node

Package Manager) tool is helping us secure all the libraries and packages we need for this project and is also helping us to host the react server.

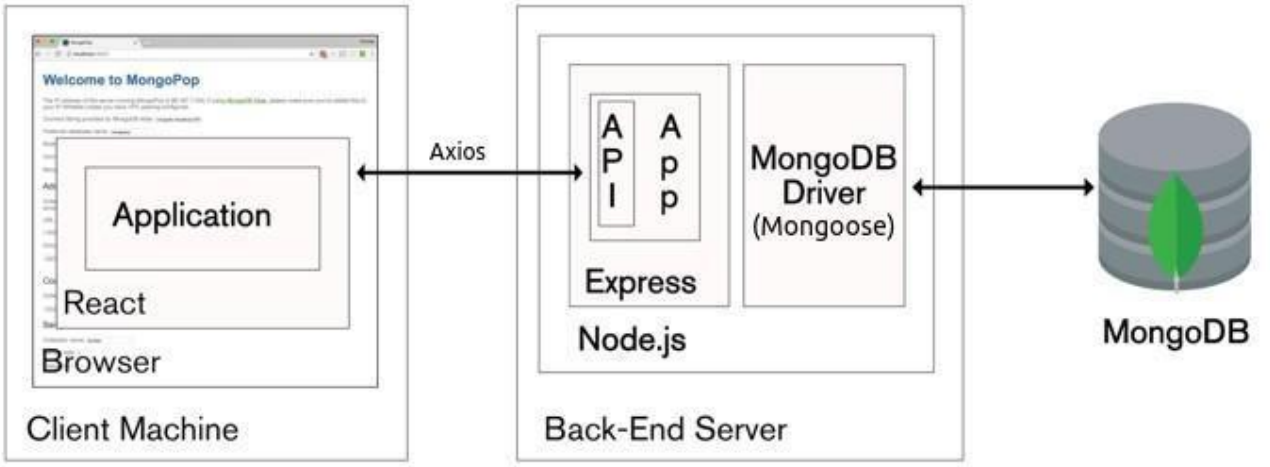



Figure 4.1.1

We have deployed the project in Amazon Web Services. Our cloud architecture is given below in the following diagram

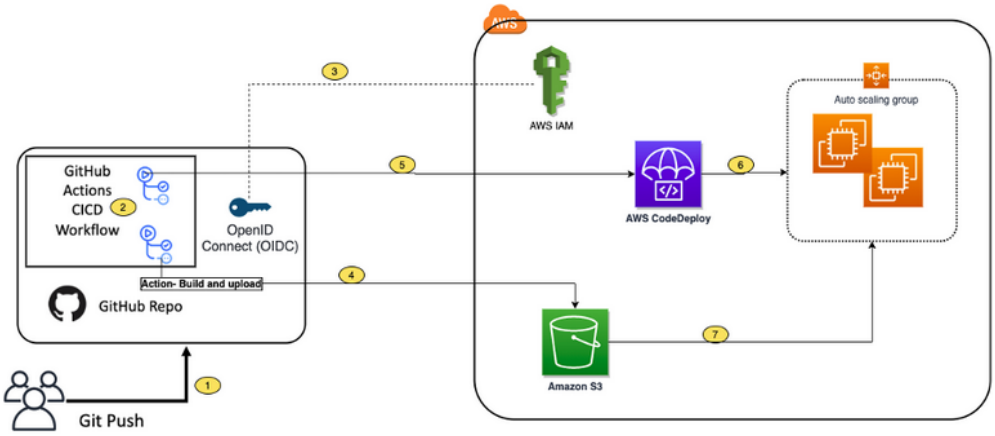



Figure 4.1.2

AWS (Amazon Web Services) is a comprehensive, evolving cloud computing platform provided by Amazon that includes a mixture of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings. AWS services can offer organization tools such as compute power, database storage and content delivery services.

AWS launched in 2006 from the internal infrastructure that Amazon.com built to handle its online retail operations. AWS was one of the first companies to introduce a pay-as-you-go cloud computing model that scales to provide users with compute, storage or throughput as needed.

AWS offers many different tools and solutions for enterprises and software developers that can be used in data centers in up to 190 countries. Groups such as government agencies, education institutions, nonprofits and private organizations can use AWS services.
An Amazon EC2 instance is a virtual computing environment that you create and configure using Amazon Elastic Compute Cloud. Amazon EC2 provides scalable computing capacity in the AWS Cloud.

Amazon Web Services allow our users from anywhere in the world to access our portal. Not only that, our website is capable of handling users who are concurrently trying to access our portal. Our website is running in http://http://54.184.111.82:3000/.

### 4.2. Proposed algorithms

We have successfully implemented and deployed the project in Amazon Web Services. Our features include the user authentication systems, user credential database system, donor registration systems, donor database management systems, user searching and requests systems. All these features are currently working perfectly. For the user side we have also successfully implemented and designed a web-application where users can interact with all these systems. All the operable functions are mentioned once again below for clear understanding.

Current functionality of our project:-

1. Users can create their own accounts in our portal.
2. Users can login to our website using their credentials.
3. For security purposes, we have limited the use of a single email for every account that the users create. So that means that every user can have only one account and two accounts with the same email id cannot be created. In that case, our system can detect and show the user an error reminding him that an account with the same email id already exists.
4. Users can register donors in our portal by filling up a form.
5. Users can search for donors in our database according to their preferences.
6. Users can make requests to donors for blood donation.
7. Visitor counter for the website.
8. Users can now change their passwords.

The Use-case diagram of this project is as follows:

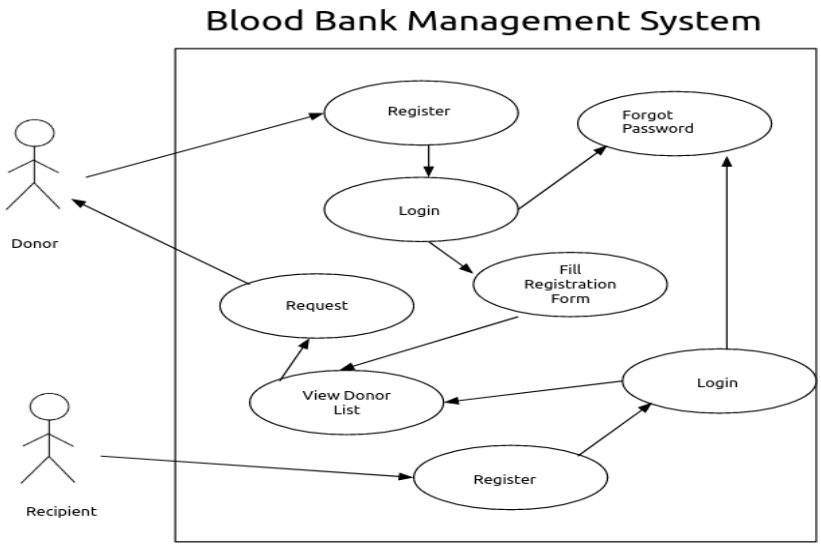



Figure 4.2.1

These are the list of user data that are being collected.

For login credentials

| No. | Data | Reason |
|---|---|---|
| 1 | Name | For identification purposes |
| 2 | Email | For login, and for website security |
| 3 | Password | For users to securely login |

Table 4.2.1

For donor

| No. | Data | Reason |
|---|---|---|
| 1 | Name | For identification purposes |
| 2 | Email | For contacting purposes |
| 3 | Phone | For contacting purposes |
| 4 | Location | For optimizing user search |
| 5 | Prior Diseases | For recipient's knowledge |

| | | |
|---|---|---|
| 6 | Donor is diabetic or not | For recipient's knowledge |
| 7 | Pricing | Donor charges to donate (if any) |

Table 4.2.2

For recipient

| No | Data | Reason |
|---|---|---|
| 1 | Search query | For understanding user’s need |
| 2 | Request information | For record management |

Table 4.2.3

## 5. Simulation result and analysis

### 5.1. Experimental setup

Our website is hosted in an EC2 instance in Amazon Web Services, the EC2 instance has Ubuntu as its system image. The instance contains the project and the necessary dependencies of the project.
The project files have been cloned from the github repository, and all the necessary dependencies are installed in the instance.
Our web application can be started from the AWS Management Console. If the instance is already not running, start the instance.

Remotely connect to the instance using terminal or command prompt using the following line of code:

```
ssh -i “bloodbank.pem” ubuntu@ip_address
```

In order to start react server the following command is being used:

```
npm start
```

For the express server the following command is being used:

```
npx nodemon index.js
```

### 5.2. Experimental results

Our react server terminal

```
Compiled successfully!
```

```
You can now view blood-bank in the browser
Note that the development build is not optimized
To create a production build, use npm run build.

assets by status 12 MiB [cached] 7 assets
assets by path . 2.04 MiB
  assets by info 10.2 KiB [immutable]
       asset main.32a92b6ad3fcf9ee1b14.hot-update.js 10.2 KiB [emitted] [immutable]
[hmr] (name: main) 1 related asset
      asset main.32a92b6ad3fcf9ee1b14.hot-update.json 28 bytes [emitted] [immutable]
[hmr]
  asset static/js/bundle.js 2.02 MiB [emitted] (name: main) 1 related asset
  asset index.html 1.85 KiB [emitted]
  asset asset-manifest.json 1.05 KiB [emitted]
Entrypoint main 2.03 MiB (13.9 MiB) = static/js/bundle.js 2.02 MiB
main.32a92b6ad3fcf9ee1b14.hot-update.js 10.2 KiB 8 auxiliary assets
cached modules 1.75 MiB (javascript) 12 MiB (asset) [cached] 206 modules
runtime modules 31.5 KiB 16 modules
./src/components/SignUp.js 6.9 KiB [built] [code generated]
webpack 5.65.0 compiled successfully in 13245 ms
```

Our express server terminal

```
[nodemon] 2.0.15
[nodemon] to restart at any time, enter ‘rs’
[nodemon] watching paths(s): *.*’
[nodemon] watching extension: js,mjs,json
[nodemon] starting ‘node index.js
Server started at port 4000
MongoDB connection successful
```

Our website is running in http://http://54.184.111.82:3000/.

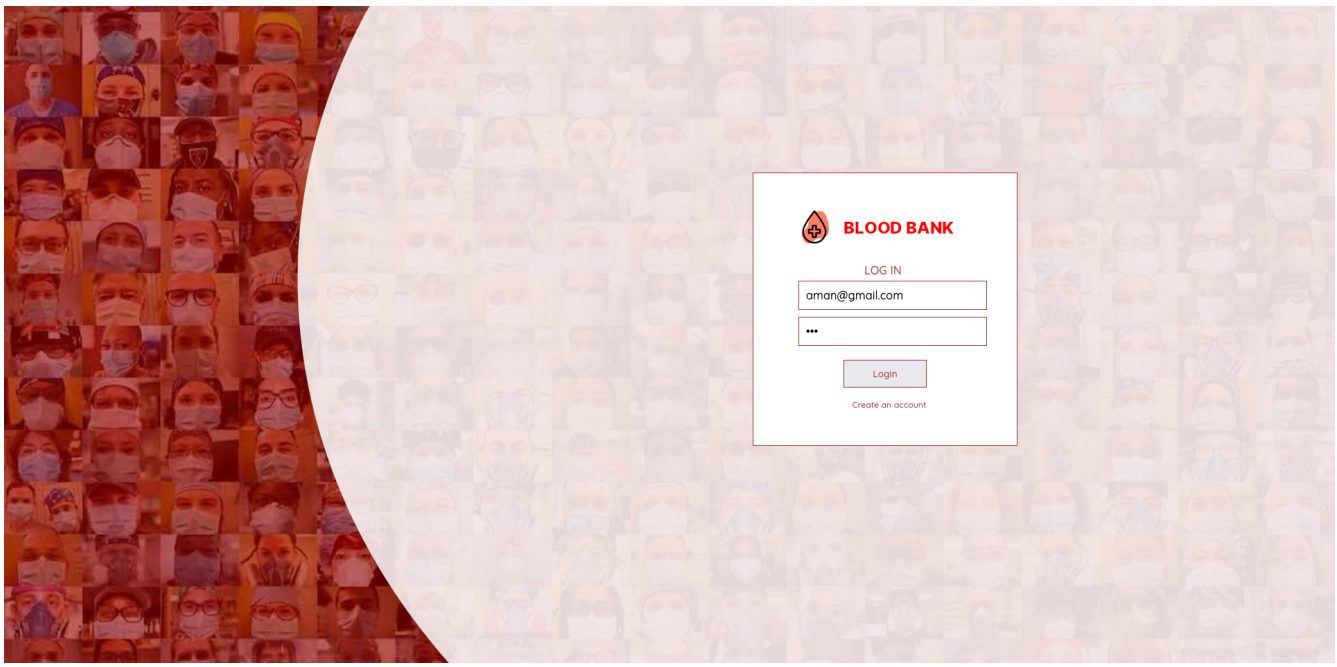



Figure 5.2.1

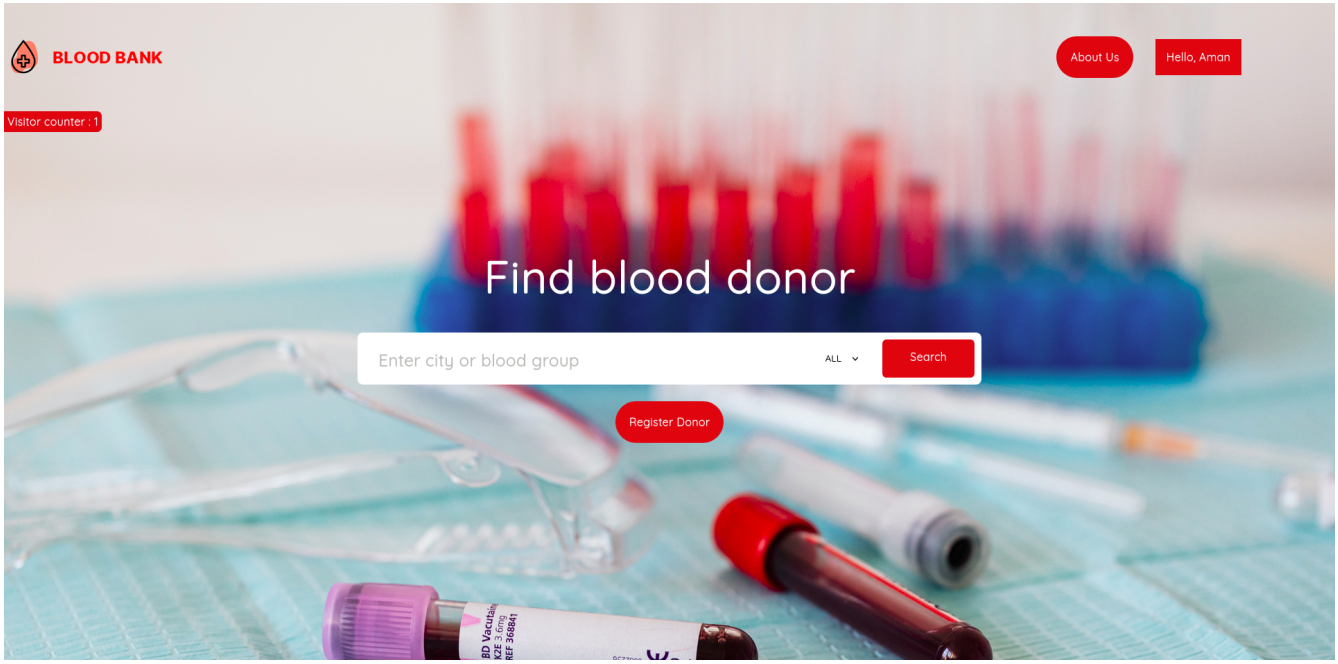



Figure 5.2.2

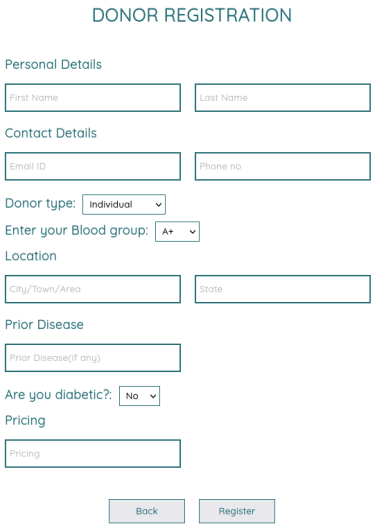



Figure 5.2.3

# 6. Discussion and Conclusion

## 6.1. Discussion

The development of this project commenced last September, we began researching about the topic and discovered that there is this problem that exists in the world today regarding blood donation, patients who are in need of blood are struggling to get the blood that they need to save their life. So we decided to work on this problem, and we came up with a conclusion to build a project that will connect donors with their recipients. The idea was to create a digital portal where donors can enroll themselves, and recipients could easily find their donors. So, we decided to build a traditional react web application and then we hosted it with the help of Amazon Web Services for users to access it.

As of now, our project supports all the features of a Blood Bank Management System, but we are looking to make improvements to it by adding more features in the future.

We are at the moment looking for scopes to improve, add more features to this project, and trying to optimize the user interface. We are also working on improving the designs of the project. So our users can expect more features in the coming future.

Our team is keeping an eye for production bugs, and continuously working to fix them, so that our users can access our website without any issue.

## 6.2. Conclusion

After extreme hard work and effort over these last few months, our team has reached a point in this project where all the features like donor registration system, donor database management system, user authentication and login system, user searching and request systems, etc. are fully functional and operable, but we have identified features and upgrades that can make this project even more useful. We are currently trying to implement those upgrades in our systems and are also working to improve the existing features. The project has been deployed in Amazon Web Services, and anyone from anywhere in the world can access our website. But, we are constantly looking for scopes to improve the project and constantly rectifying any errors or bugs that exist in the project.

## 7.0. References

Drew, V.J., Barro, L., Seghatchian, J., & Burnouf, T.(2017). Towards pathogen inactivation of red blood cells and whole blood targeting viral DNA/RNA: design, technologies, and future prospects for developing countries. Blood Transfusion, 15(6), 512.

Bhudev Network Charitable Foundation, Vadodara,2022; https://bharatbloodbank.in (accessed on 20th January 2022)

## 8.0. Sample codes

**src/app.js**

```
import Home from './components/Home'
import RegisterDonor from './components/RegisterDonor'
import ViewDonor from './components/ViewDonor'
import RegistrationDone from './components/RegistrationDone'
import Purchase from './components/Purchase'
import Login from './components/Login'
import SignUp from './components/SignUp'
import AboutUs from './components/AboutUs'
import ResetPw from './components/ResetPw'
import Intro from './components/Intro'
import {BrowserRouter as Router,Switch,Route} from 'react-router-dom'

const App = () =>{
  return(
    <Router>
      <Switch>
        <Route exact path='/login' component={Login}/>
        <Route exact path='/signup' component={SignUp}/>
        <Route exact path='/home' component={Home}/>
        <Route exact path='/donor' component={RegisterDonor}/>
        <Route exact path='/viewdonor' component={ViewDonor}/>
        <Route exact path='/registrationdone' component={RegistrationDone}/>
        <Route exact path='/purchaseconfirmation' component={Purchase}/>
        <Route exact path='/login' component={Login}/>
        <Route exact path='/aboutus' component={AboutUs}/>
        <Route exact path='/resetpw' component={ResetPw}/>
        <Route exact path ='/' component={Intro}/>
      </Switch>
    </Router>
  )
}

export default App;
```

```
import React,{useEffect, useState} from 'react'
import logo from './css/images/logo.png'
import './css/home.css'
```
**src/components/Home.js**

```
import {Link} from 'react-router-dom'
import axios from 'axios'
import Cookies from 'universal-cookie'

function Home()
{
    const [vis_count,setVis] = useState('0')
    const [search,setSearch] = useState('')
    const [blood_type,setBlood] = useState('ALL')

    const [name,setName] = useState('')

    const[em,setEm]=useState('')

    useEffect(()=>{
        document.title='Blood Bank'

        const cookies = new Cookies()
        setEm(cookies.get('email'))

        axios.get(global.config.ip.ip_db+'/dataHandle/getLogin', {
            params: {
                email: em
            }
        })
        .then((res)=>{
            setName(res.data[0].name)
        })

        axios.get(global.config.ip.ip_db+'/dataHandle/getpwd')
        .then((res)=>{
            setVis(res.data.length)
        })
```

```
    })

    function logout()
    {
        window.location.replace(global.config.ip.ip_react+'/login')
    }

    function change_pw()
    {
        window.location.replace(global.config.ip.ip_react+'/resetpw')
    }


    return(
        <div className='main'>
            <nav>
            <img src={logo} alt="Logo" className='logo'/>
            <ul className='nav-links'>
                <li><a className='register-donor' href='/aboutus'>About
Us</a></li>
                <li><div className='dropdown'>
                    <button className = "dropbtn">Hello, {name}</button>
                    <div className='dropdown-content'>
                        <a onClick={change_pw}>Change password</a>
                        <a onClick={logout}>Log out</a>
                    </div>
                    </div></li>
            </ul>
            </nav>



        <div className='content'>
            <h1>Find blood donor</h1>
            <form>
```

```
                <input autoComplete='off' type='text' id='search-text'
placeholder='Enter city or blood group'
onChange={e=>{setSearch(e.target.value)}}/>
                <select id='search-blood-grp'
onChange={e=>{setBlood(e.target.value)}}>
                    <option value='A+'>A+</option>
                    <option value='A-'>A-</option>
                    <option value='AB+'>AB+</option>
                    <option value='AB-'>AB-</option>
                    <option value='B+'>B+</option>
                    <option value='B-'>B-</option>
                    <option value='O+'>O+</option>
                    <option value='O-'>O-</option>
                    <option value='ALL' selected>ALL</option>
                </select>
               <Link className='home-btn'to
={{pathname:'/viewdonor',state:search,blood_g:blood_type}}>Search</Link>
            </form>
            <div className='register-content'>
            <Link className = 'register-donor'to ='/donor'>Register
Donor</Link>
            </div>
        </div>
        <div className='footer-home'>
                <h4 className='footer-text'>  Visitor counter :
{vis_count}</h4>
            </div>

        </div>

    )
}

export default Home
```

**src/components/Login.js**

```
import React, { useState} from 'react'
```

```
import { Link } from 'react-router-dom'
import axios from 'axios'
import logo from './css/images/logo.png'
import './css/login.css'
import Cookies from 'universal-cookie'


function Login() {
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')



    function tryLogin() {
        axios.get(global.config.ip.ip_db+'/dataHandle/getLogin', {
            params: {
                email: email
            }
        })
            .then(res => {
                if (res.data.length === 0)
                    show_error()

                else {
                    if (password === res.data['0']['password'])
                    {
                        createnewpwd()

window.location.replace(global.config.ip.ip_react+'/home')
                    }

                    else
                        show_error()
                }
            })
    }

    function createnewpwd()
```

```
    {
        const cookies = new Cookies()
        cookies.set('email',email,{path:'/'})
        cookies.set('password',password,{path:'/'})

        axios.post(global.config.ip.ip_db+'/dataHandle/addpwd',{"email":email})
    }

    function hide_error() {
        document.getElementById('err-msg').style.display = 'none'
    }

    function show_error() {
        document.getElementById('err-msg').style.display = 'inline'
    }

    return (
        <div className='login-main'>
            <div className='login-container2'>
                <img className='login-img' src={logo} alt='' /><br />
                <lable className='login-label'>LOG IN</lable><br />
                <input type='text' autoComplete='off' className='login-inputs'
onChange={e => { setEmail(e.target.value) }} placeholder='Email ID' /><br />
                <input type='password' autoComplete='off'
className='login-inputs' onChange={e => { setPassword(e.target.value) }}
placeholder='Password' /><br />
                <button className='login-btn'
onClick={tryLogin}>Login</button><br />

                <Link to='/signup' className='login-btn2'>Create an
account</Link><br/>

            </div>
            <div className='alert-container' >
                <div className="alert" id='err-msg'>
                    <span className="closebtn"
onClick={hide_error}>&times;</span>
                    Email Id or Password is incorrect
```

```
                </div>
            </div>
        </div>
    )
}

export default Login
```

**src/Index.js**

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './config'
import { BrowserRouter } from 'react-router-dom';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
    <BrowserRouter>
    <App />
    </BrowserRouter>,
    document.getElementById('root')
);
reportWebVitals();
```

**server/index.js**

```
require('./db')
const express = require('express')
const cors = require('cors')
const bodyParser = require('body-parser')
var handleDataRoute = require('./controllers/postDataController')
var app = express()
app.use(bodyParser.json())
app.use(cors())
app.listen(4000,()=>console.log('Server started at port 4000'))
app.use('/dataHandle',handleDataRoute)
```

**server/db.js**

```
const mongoose = require('mongoose')
mongoose.connect('mongodb://localhost:27017/DonorData',
{useNewUrlParser:true} ,err =>{
    if(!err)
    console.log('MongoDB connection successful')
    else
    console.log('Error! MongoDB connection failed')
})
```

**server/modules/dataschema.js**

```
const mongoose = require('mongoose')

var data =mongoose.model('data',{
    first_name : {type:String},
    last_name :{type:String},
    type:{type:String},
    email :{type:String},
    phone:{type:String},
    blood_grp:{type:String},
    city:{type:String},
    state:{type:String},
    prior_disease:{type:String},
    diabetic:{type:String},
    price:{type:String}
})

module.exports = {data}
```

**server/controller/postDataController.js**

```
const express = require('express')
var router = express.Router()
var ObjectId = require('mongoose').Types.ObjectId
```

```
var {login} = require('../models/loginschema')
var {data} = require('../models/dataschema')
var {pwd} = require('../models/pwdschema')
const { query } = require('express')


router.post('/addpwd',(req,res)=>{
    var new_pwd = new pwd({
        name:req.body.name,
        email:req.body.email,
        password:req.body.password
    })
    new_pwd.save((err,docs)=>{
        if(!err)
        res.send(docs)
    })
})

router.post('/addlogin',(req,res)=>{
    var new_login = new login({
        name:req.body.name,
        email:req.body.email,
        password:req.body.password
    })

    new_login.save((err,docs)=>{
        if(!err){
            res.send(docs)
            console.log('Login Data inserted successfully')
        }
        else {
            console.log('Error creating new
record'+JSON.stringify(err,undefined,2))
        }
    })
})

router.get('/getLogin',(req,res)=>{
```

```
    var email = req.query.email
    var query = {email:email}
    login.find(query,(err,docs)=>{
        if(!err)
        res.send(docs)
        else
        console.log('Error while retrieving all
records'+JSON.stringify(err,undefined,2))
    })
})

router.get('/getPwd',(req,res)=>{
    pwd.find((err,docs)=>{
        if(!err)
        res.send(docs)
    })
})

router.get('/fetchall',(req,res) =>{
    var search = req.query.search
    var blood = req.query.blood_grp
    var regex = new RegExp(search,'i')
    var query = {}
    if(search.length !=0 && blood == 'ALL')
        query = {$or:[{city:{$regex:regex}},{state:{$regex:regex}}]}
    if(search.length !=0 && blood != 'ALL')
        query =
{$or:[{city:{$regex:regex}},{state:{$regex:regex}}],$and:[{blood_grp:blood}]}
    if(search.length ==0 && blood == 'ALL')
        query = {}
    if(search.length==0 && blood !='ALL')
        query={blood_grp:blood}

    data.find(query,(err,docs)=>{
        if(!err)
        res.send(docs)
        else
```

```
        console.log('Error while retrieving all
records'+JSON.stringify(err,undefined,2))

    })
})


router.post('/add',(req,res)=>{
    var new_data = new data ({
        first_name :req.body.first_name,
        last_name :req.body.last_name,
        type:req.body.type,
        email :req.body.email,
        phone:req.body.phone,
        blood_grp:req.body.blood_grp,
        city:req.body.city,
        state:req.body.state,
        prior_disease:req.body.prior_disease,
        diabetic:req.body.diabetic,
        price:req.body.price
        })
    new_data.save((err,docs) =>{
        if(!err)
        {
            res.send(docs)
            console.log('Data inserted successfully')
        }
        else
        console.log('Error creating new
record'+JSON.stringify(err,undefined,2))
    })
})


router.delete('/deletedata/:id',(req,res)=>{
    if(!ObjectId.isValid(req.params.id))
        return res.status(400).send("No records with given id")
```

```
        data.findByIdAndRemove(req.params.id,(err,docs)=>{
            if(!err) res.send(docs)
            else console.log('Error while deleting given id')
        })
})

router.delete('/deletelogin',(req,res)=>{
    var emailq = req.query.email
    login.findOneAndRemove({email:emailq},function(err){
        login.find({},function(err){
        })
    })
})


router.delete('/deletepwd',(req,res)=>{
    var emailq =req.query.email
    pwd.findOneAndRemove({email:emailq},function(err){
    })
})
module.exports = router
```

**src/components/viewDonor.js**

```
import React,{useState,useEffect} from 'react'
import './css/viewdonor.css'
import axios from 'axios'
import {Link} from 'react-router-dom'

const ViewDonor=(props)=>{

    var search_query = props.location.state
    var search_query_b = props.location.blood_g
    const [data,setData] = useState([{
        _id:'',
        first_name:"",
        last_name:"",
        type:'',
```

```
        email:"",
        phone:"",
        blood_grp:"",
        city:"",
        state:"",
        prior_disease:"",
        diabetic:"",
        price:""
    }])

    const [search,setSearch] = useState(search_query)
    const [blood,setBlood] = useState(search_query_b)

    useEffect(()=>{
        axios.get(global.config.ip.ip_db+'/dataHandle/fetchall',{
            params:{
                search:search,
                blood_grp:blood
            }
        })
        .then(res=>setData(res.data))
})

    return(
        <div>
        <div className='donor-view-header'>
            <input className='search-input' autoComplete='off'
placeholder='Search query here' type='text'
onChange={e=>setSearch(e.target.value)} defaultValue={search_query}/>
            <select className='search-select' id='search-blood-grp'
defaultValue={search_query_b} onChange={e=>{setBlood(e.target.value)}}>
                    <option value='A+'>A+</option>
                    <option value='A-'>A-</option>
                    <option value='AB+'>AB+</option>
                    <option value='AB-'>AB-</option>
                    <option value='B+'>B+</option>
                    <option value='B-'>B-</option>
                    <option value='O+'>O+</option>
```

```
                    <option value='O-'>O-</option>
                    <option value='ALL'>ALL</option>
                </select>
                <Link to='/home' className='home-btn-view'>Go back to
home</Link>
        </div>
        <div className='donor-table'>
            <table>
                <tr>

                    <th className='heading'>Name</th>
                    <th className='heading'>Type</th>
                    <th className='heading'>Email</th>
                    <th className='heading'>Phone</th>
                    <th className='heading'>Blood Group</th>
                    <th className='heading'>City</th>
                    <th className='heading'>State</th>
                    <th className='heading'>Prior Disease</th>
                    <th className='heading'>Diabetic</th>
                    <th className='heading'>Price</th>
                </tr>
                {data.map((val,key)=>{
                    return(
                        <tr key = {key}>
                            <td className='entries'>{val.first_name}{'
'}{val.last_name}</td>
                            <td className='entries'>{val.type}</td>
                            <td className='entries'>{val.email}</td>
                            <td className='entries'>{val.phone}</td>
                            <td className='entries'>{val.blood_grp}</td>
                            <td className='entries'>{val.city}</td>
                            <td className='entries'>{val.state}</td>
                            <td className='entries'>{val.prior_disease}</td>
                            <td className='entries'>{val.diabetic}</td>
                            <td className='entries'>{'₹'}{val.price}</td>
                        </tr>
                    )
                })
```

```
                }
            </table>
        </div>
        {data.length===0 && <h1 className='no-data'>No donors found</h1>}
        </div>
    )
}


export default ViewDonor
```