INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD
PRAYAGRAJ, UTTAR PRADESH, INDIA 211015

DEPARTMENT OF INFORMATION TECHNOLOGY

A Mini-Project Report
On

"BLOOD BANK MANAGEMENT SYSTEM"

Submitted by
Group – 9

Members

Dhiraj Damani (IIT2020014)
Nikhil Dubey (IIT2020016)
Rahul Mahto (IIT2020022)
Prahlad Nagula (IIT2020090)
Jhambhule Sahas Devidas(IIT2020105)

Teaching Assist.
Bhura Manohar

Prof.
Dr. Ranjana Vyas
Dr. Anjali Gautam

.

IIIT Allahabad
Indian Institute of Information Technology Allahabad
Devghat, Jhalwa, Prayagraj-211015, U. P. INDIA
Phone: 91-532-2922000, Fax: 91-532-2922125, Email: contact@iiita.ac.in

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO BLOOD BANK MANAGEMENT SYSTEM

The population of the world is multiplying with each coming year and so are the diseases and health issues. With an increase in the population there is an increase in the need of blood. The growing population of the world results in a lot of potential blood donors. But in spite of this not more than 10% of the total world population participates in blood donation. With the growing population and the advancement in medical science the demand for blood has also increased. Due to the lack of communication between the blood donors and the blood recipients, most of the patients in need of blood do not get blood on time and hence lose their lives. There is a dire need of synchronization between the blood donors and the blood bank. This improper management of blood leads to wastage of the available blood inventory. Improper communication and synchronization between the blood bank and needy leads to wastage of the blood available. These problems can be dealt with by automating the existing manual blood bank management system. A high-end, efficient, highly available and scalable system has to be developed to bridge the gap between the donors and the recipients and to reduce the efforts required to search for blood donors.

- The project blood bank management system is known to be a pilot project that is designed for the blood bank to gather blood from various sources and distribute it to the needy people who have high requirements for it.
- The software is designed to handle the daily transactions of the blood bank and search the details when required.
- It also helps to register the details of donors, blood collection details as well as blood issued reports.
- The software application is designed in such a manner that it can suit the needs of all the blood bank requirements in the course of the future.

This project in the Blood Bank Management System will develop an efficient system for blood transactions.

1.2 NEED FOR BLOOD BANK MANAGEMENT SYSTEM

The management is ad-hoc with no semblance of organization or standard operating procedures. Donors cannot access blood from blood banks other than the bank where they have donated blood. In the present system all the blood banks are attached to hospitals and there is no stand-alone blood bank. Some hospital has its own systems and limitations. Because of the low number of donors and more number of blood banks, the efficiency and quality of blood banks are low, resulting in wastage of blood and blood components. There is Scarcity of rare blood group, Unavailability of blood during emergency, Less awareness among people about blood donation and blood transfusion, Deaths due to lack of blood during operations.

1.3 AIM OF THIS PROJECT

The basic building aim is to provide blood donation service to the city recently. Blood Bank Management System (BBMS) is a Web-based application that is designed to store, process, retrieve and analyse information concerned with the administrative and inventory management within a blood bank. This project aims at maintaining all the information pertaining to blood donors, different blood groups available in blood bank and help them manage in a better way.

Also, project aim is to provide transparency in this field, make the process of obtaining blood from a blood bank hassle-free and corruption-free and make the system of blood bank management effective.

CHAPTER 2

LITERATURE REVIEW

2.1 HISTORY OF BLOOD BANK MANAGEMENT SYSTEM

Blood bank, an organization that collects, stores, processes, and transfuses blood. During World War I it was demonstrated that stored blood could safely be used, allowing for the development of the first blood bank in 1932. Before the first blood banks came into operation, a physician determined the blood types of the patient's relatives and friends until the proper type was found, performed the cross match, bled the donor, and gave the transfusion to the patient. In the 1940s the discovery of many blood types and of several cross matching techniques led to the rapid development of blood banking as a specialized field and to a gradual shift of responsibility for the technical aspects of transfusion from practicing physicians to technicians and clinical pathologists. The practicality of storing fresh blood and blood components for future needs made possible such innovations as artificial kidneys, heart-lung pumps for open-heart surgery, and exchange transfusions for infants with erythroblastosis fetalis.

Whole blood is donated and stored in units of about 450 ml (slightly less than one pint). Whole blood can be stored only for a limited time, but various components (e.g., red blood cells and plasma) can be frozen and stored for a year or longer. Therefore, most blood donations are separated and stored as components by the blood bank. These components include platelets to control bleeding; concentrated red blood cells to correct anemia;and plasma fractions, such as fibrinogen to aid clotting, immune globulins to prevent and treat a number of infectious diseases, and serum albumin to augment the blood volume in cases of shock. Thus, it is possible to serve the varying needs of five or more patients with a single blood donation. Despite such replacement programs, many blood banks face continual problems in obtaining sufficient donations. The chronic shortage of donors has been alleviated somewhat by the development of apheresis, a technique by which only a desired blood component is taken from the donor's blood, with the remaining fluid and blood cells immediately transfused back into the donor. This technique allows the collection of large amounts of a particular component, such as plasma or platelets, from a single donor.

2.2 BASE PAPER / EXISTING SYSTEM

2.2.1 Base Paper

India has four types of blood banks/centers from the administrative point of view. They are managed by the public (government) sector, Indian Red Cross Society, Nongovernment Organizations (NGOs, on a not for profit basis), and corporate or commercial sectors. As in 2013, there are a total of 2545 designated blood banks in India.Although there is an increasing trend regarding the number of blood banks in India over the years, the point of concern is the pace of increment.

Another concern is the blood banks in the public sector. Whatever increase in the number of blood storage facilities India has witnessed, it was mostly because of private sectors. Growth in the government sector is too slow paced. As per the National Blood Policy every First Referral Unit like district hospital (DH), Subdivisional Hospital (SDH), Community Health Centres (CHC), etc. should be equipped with round the clock blood transfusion facility. There are more than 6000 facilities such as DH, SDH, and CHC present across India excluding medical colleges and more than 80% of these lack blood storage facility.Such severe lack in a storage facility is also translated into blood collections. The total recorded blood collection in India is four million units, which meet only 40% of need against a minimum requirement of 10 million units (calculated at 1% of 1 billion populations).Moreover, the blood banks in private sectors are not accessible or affordable to the rural population due to cost constraints.

In addition to a lack of facilities, lack of proper management and functioning of existing blood transfusion services make the situation even worse. Blood transfusion is an emergency service and should be provided round the clock but non functioning of blood banks 24×7 is a hindrance to providing emergency care. Linking of district blood banks with peripheral health institutions is also lacking.Moreover, lack of manpower, training, monitoring, and supervision of the existing blood banks are some of the other issues which make the situation even worse.

2.2.2 Existing system:

The operation of the blood bank still now is maintained in the manual system.

• The operation is tedious, time consuming and space consuming.

• It creates room for errors as the data is entered manually by the persons.

• It includes the risk of the documents being lost over the years and maintenance of the records is difficult.

• The data recorded during testing or while acquiring the details of different aspects of blood bank management system is not so accurate and precise.

• Maintaining the stock of blood and the daily transactions without computerisation also poses a challenge.

2.3 PROPOSED SYSTEM:

The proposed system (Blood Bank Management System) is designed to help the Blood Bank administrator to meet the demand of Blood by sending and/or serving the request for Blood as and when required.The proposed system gives the procedural approach of how to bridge the gap between Recipient, Donor, and Blood Bank. This Application will provide a common ground for all the three parties (i.e. Recipient, Donor, and Blood Bank) and will ensure the fulfillment of demand for Blood requested by Recipient and/or Blood Bank.The proposed system consists of the following goals and has the scope as follows:

a) Goals:

- To ease the process of blood donation and reception.
- To improve the existing system.
- To develop a scalable system.
- To be highly available

b) Scope:

- Ensure that all the functionalities of a manual blood bank are covered.
- Make sure the program is simple and easy to use

CHAPTER 3

SPECIFICATIONS

3.1 HARDWARE REQUIREMENTS

- PROCESSOR : Intel Pentium or Higher Version
- RAM : Minimum 1GB
- HARD DISK : 60GB and above

3.2 SOFTWARE REQUIREMENTS

- SOFTWARE : Python 3.3 or greater
- DATABASE : Flask-MySQLdb-0.2.0
- SUPPORTED BROWSERS : Google Chrome / Mozilla Firefox / Internet Explorer
- EDITOR : Atom / Visual Studio Code
- FRAMEWORK : Flask 1.1.1
- OPERATING SYSTEM : Windows , or MACos, or Linux (32/64 bit)

3.3 FUNCTIONAL REQUIREMENTS

The Functional Requirements Specification documents the Operations and activities that a system must be able to perform. Functional Requirements include:

- Manage the information of donors.
- Manage the information of employees.
- Descriptions of operations performed by each system.
- Descriptions of work-flows performed by the system.
- To maintain storage area and data storage.
- Who can enter the data into the system.

The Functional Requirements Specifications is designed to be read by a general audience. Readers should understand the system, but no particular technical knowledge should be required to understand the document.

These are the functional requirements specification documents for the project analysis. A software requirement specification helps to attenuate the time and energy needed by the developers to attain their desired goals and additionally minimizes the value of development.

Following Factors are used to measure software development quality:

Each attribute may be accustomed measure of the product performance. These attributes may be used for Quality assurance similarly as quality control. Quality assurance activities are directed towards prevention of introduction of defects and internal control activities are aimed toward detecting defects in products and services.

1. Reliability

Measure if product is reliable enough to sustain in any condition. Give systematically correct results. Product dependability is measured in terms of operation of project underneath different operating atmosphere and different conditions.

2. Maintainability

Different versions of the product ought to be easy to maintain. For development it ought to be easy to feature code to existing system, ought to be easy to upgrade for brand new options and new technologies from time to time. Maintenance ought to be value effective and simple. System be easy to take care of and correcting defects or making a change within the software system.

3. Usability

This can be measured in terms of ease of use. Application should be user friendly. Easy to use for input preparation, operation and also for interpreting of output.

4. Portability

This can be measured in terms of Costing issues related to porting, Technical issues related to porting, Behavioural issues related to porting.

3.4 NON-FUNCTIONAL REQUIREMENTS

Satisfactory will probably not be assessed on the system where the program is developed, tested or first installed.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

System design is the first design stage for devising the basic approach to solving the problem. During system design, developers decide the overall structures and styles. The system architecture determines the organization of the system into subsystems. In addition, the architecture provides the context for the detailed decisions that are made in later stages .during design, developers make decisions about how the problem will be solved, first at the high level and then with more detail.

4.1.1 ARCHITECTURAL DIAGRAM

Any software should have a design structure of its functionality i.e. the architecture which defines about it's inside view, likewise there is a database architecture in DBMS. The interaction of the database in DBMS with the system and the languages used in the database architecture is as shown in the below diagram and At the end of this article, you will be given a free pdf copy of Database Architecture in DBMS.The database architecture has three levels and they are as follows:

1. External level

2. Conceptual level

3. Internal level

4.1.2 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system. Data Flow modules are used to show how data flows through a sequence of processing steps. The data is transformed at each step before moving on to the next stage. These processing steps or transformations are program functions when Data Flow Diagrams are used to document a software design.

Data flow modules are an intuitive way of showing how data is processed by a system. At the analysis level, they should be used to module the way in which data is processed in the existing system. The notation used in these modules represents functional processing, data stores and data movements between functions.

With a data flow diagram, users are able to visualize how the system will operate, what the system will accomplish and how the system will be implemented. Old system data flow diagrams can be drawn up and compared with the new system data flow.

These are several common modelling rules to be followed while creating DFD's are as follows:

-All processes must have at least one data flow in and one data flow out.

-All processes should modify the incoming data, producing a new form of outgoing data.

-Each data store must be involved with at least one data flow.
-Each external entity must be involved with at least one data flow.
-A data flow must be attached to at least one process.

4.1.3 USE CASE DIAGRAMS

The use case module is a catalogue of system functionality described using UML use cases. Each use case represents a single, repeatable interactions that a user or actor experiences when using the system. A use case typically includes one or more "scenarios" which describes the interactions that go on between the actor and the system, and documents the results and exceptions that occur from the user's perspective. Use case may include other use cases as part of a larger pattern of interaction and may also extended by other use cases to handle exceptional conditions.

A use case is a coherent piece of functionality that a system can provide by interacting with actors. For example, a customer actor can buy a beverage from a vending machine. The customer inserts money into the machine, makes a selection, and ultimately receives a beverage. Similarly, a repair technician can perform scheduled maintenance on a vending machine.

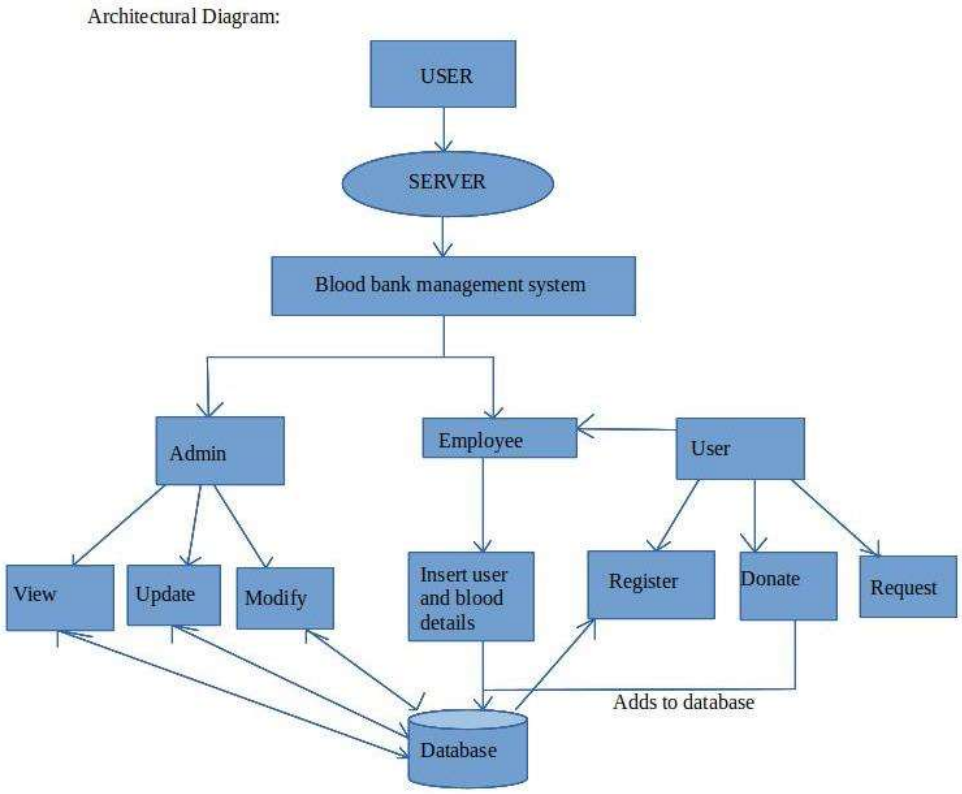
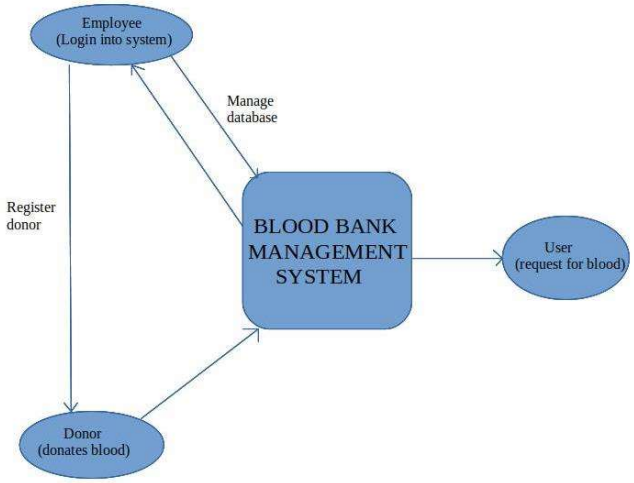
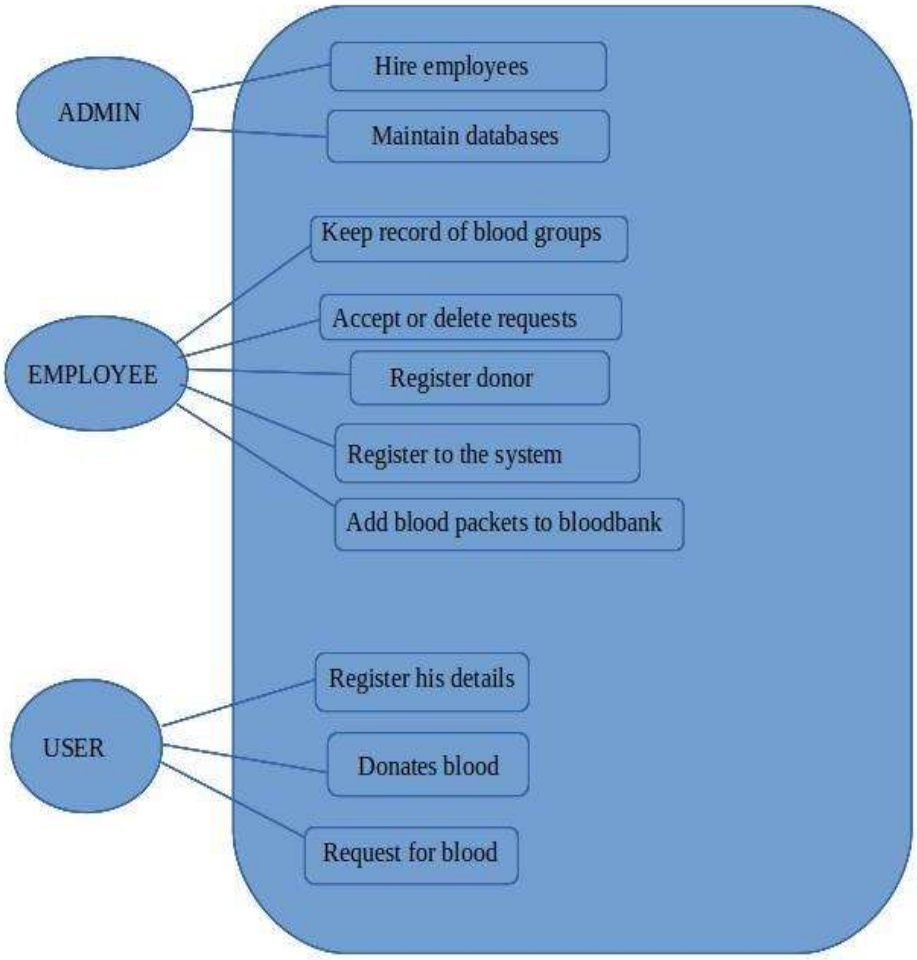


Fig 4.1



Dataflow Diagram: Fig 4.2



Case Diagram Fig 4.3

4.2 TABLES AND DATA TYPES

1.CREATE TABLE RECEPTION(

E_ID VARCHAR(54) PRIMARY KEY,

NAME VARCHAR(100),

EMAIL VARCHAR(100),

PASSWORD VARCHAR(100),

REGISTER_DATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP);

2.CREATE TABLE DONOR(

D_ID INT(3) NOT NULL AUTO_INCREMENT,

```
DNAME VARCHAR(50),
SEX VARCHAR(10),
AGE INT(3),
WEIGHT INT(3),
ADDRESS VARCHAR(150),
DISEASE VARCHAR(50),
DEMAIL VARCHAR(100),
E_ID VARCHAR(54),
DONOR_DATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT PK_2 PRIMARY KEY(D_ID)
CONSTRAINT FK_1 FOREIGN KEY(E_ID) REFERENCES RECEPTION(E_ID)
ON DELETE CASCADE ON UPDATE CASCADE);
```

3.CREATE TABLE BLOODBANK(

```
B_GROUP VARCHAR(4),
TOTAL_PACKETS INT(4),
CONSTRAINT PK_3 PRIMARY KEY(B_GROUP));
```

4.CREATE TABLE BLOOD(

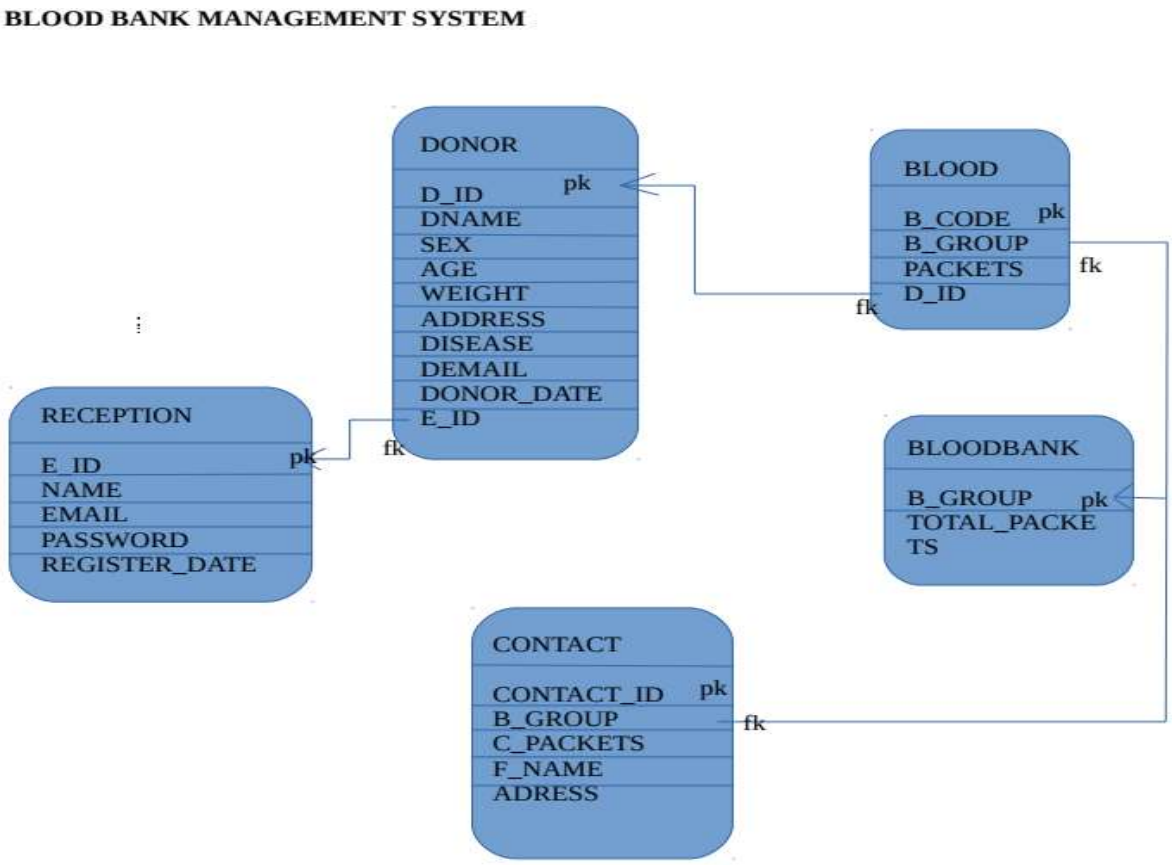
```
B_CODE INT(4) NOT NULL AUTO_INCREMENT,
D_ID INT(3),
B_GROUP VARCHAR(4),
PACKETS INT(2),
CONSTRAINT PK_4 PRIMARY KEY(B_CODE),
CONSTRAINT FK_1 FOREIGN KEY(D_ID) REFERENCES DONOR(D_ID) ON
DELETE CASCADE ON UPDATE CASCADE,
```

```
CONSTRAINT FK_2 FOREIGN KEY(B_GROUP) REFERENCES
BLOODBANK(B_GROUP) ON DELETE CASCADE ON UPDATE CASCADE);
```

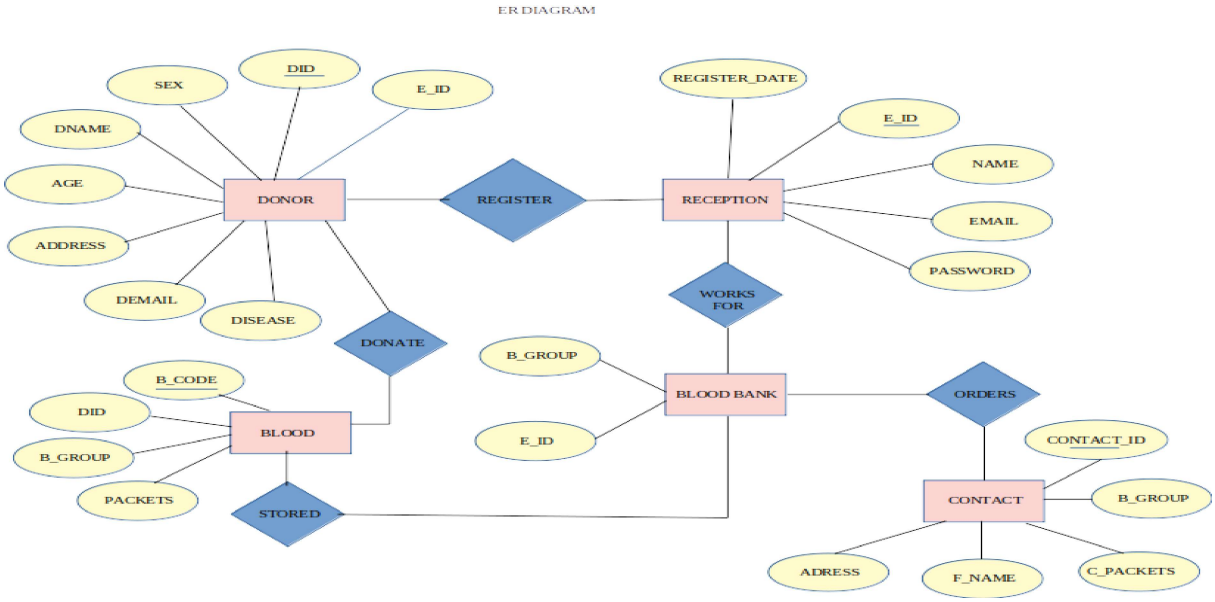
5.CREATE TABLE CONTACT(

```
CONTACT_ID INT(3) NOT NULL AUTO_INCREMENT,

B_GROUP VARCHAR(4),

C_PACKETS INT(2),

F_NAME VARCHAR(50),

ADRESS VARCHAR(250),

CONSTRAINT PK_5 PRIMARY KEY(CONTACT_ID),

CONSTRAINT FK_3 FOREIGN KEY(B_GROUP) REFERENCES
BLOODBANK(B_GROUP) ON DELETE CASCADE ON UPDATE CASCADE

)ENGINE=InnoDB AUTO_INCREMENT=100 DEFAULT CHARSET=latin1;
```

4.3 SCHEMA DIAGRAM



4.4 ER-DIAGRAM



CHAPTER 5

IMPLEMENTATION

5.1 STAGE 1: INSTALLATION

Flask is a Python framework for creating web applications. From the official site,

> *Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions.*

When we think about Python, the de facto framework that comes to our mind is the Django framework. But from a Python beginner's perspective, Flask is easier to get started with, when compared to Django.

Setting Up Flask

Setting up Flask is pretty simple and quick. With `pip` package manager, all we need to do is:

```
1. pip install flask
```

Once you're done with installing Flask, create a folder called `FlaskApp`. Navigate to the `FlaskApp` folder and create a file called `app.py`. Import the `flask` module and create an app using Flask as shown:

```
2. from flask import Flask

3. app = Flask(__name__)
```

Now define the basic route `/` and its corresponding request handler:

```
1. @app.route("/")

2. def main():

3. return "Welcome!"
```

Next, check if the executed file is the main program and run the app:

```
1. if __name__ == "__main__":

2. app.run()
```

Save the changes and execute app.py

Start the server:

```
$ export FLASK_APP='app.py'
```

Run the app:

```
$ flask run
```

Point your browser to http://localhost:5000/ and you should have the welcome message.

Install MySQL Server

Install the MySQL server by using the Ubuntu package manager.

```
$ sudo apt-get install mysql-server
```

Setting up the Database

We'll be using MySQL as the back end. So log into MySQL from the command line, or if you prefer a GUI like MySQL workbench, you can use that too. First, create a database called BucketList. From the command line:

```
$ mysql -u <username> -p
$ sudo mysql -u root -p

$ ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY
'new-password';

$  sudo service mysql stop
$  sudo service mysql start
```

Enter the required password and when logged in, execute the following command to create the database:

```
mysql> CREATE DATABASE bloodbank;

mysql> SHOW DATABASES;

mysql> USE bloodbank;
```

5.2 STAGE 2: INSERT DATA

1.RECEPTION

EID	VARCHAR
NAME	VARCHAR
EMAIL	VARCHAR
PASSWORD	VARCHAR
REGISTER_DATE	TIMESTAMP

CODE:

```
cur.execute("INSERT INTO RECEPTION(E_ID,NAME,EMAIL,PASSWORD)
VALUES(%s, %s, %s, %s)",(e_id, name, email, password))
```

2.DONOR

D_ID	INT NOT NULL AUTO_INCREMENT
DNAME	VARCHAR
SEX	VARCHAR
AGE	INT
WEIGHT	INT
ADDRESS	VARCHAR
DISEASE	VARCHAR

DEMAIL	VARCHAR
DONOR_DATE	TIMESTAMP

CODE:

```
cur.execute("INSERT INTO
DONOR(DNAME,SEX,AGE,WEIGHT,ADDRESS,DISEASE,DEMAIL) VALUES(%s, %s,
%s, %s, %s, %s, %s)",(dname , sex, age, weight, address, disease, demail))
```

3.BLOOD

B_CODE	INT NOT NULL AUTO_INCREMENT
D_ID	INT
B_GROUP	VARCHAR
PACKETS	INT

CODE:

```
cur.execute("INSERT INTO BLOOD(D_ID,B_GROUP,PACKETS) VALUES(%s, %s,
%s)",(d_id , blood_group, packets))
```

4.BLOODBANK

B_GROUP	VARCHCAR
TOTAL_PACKETS	INT

CODE:

```
cur.execute("SELECT * FROM BLOODBANK")

records = cur.fetchall()

cur.execute("UPDATE BLOODBANK SET TOTAL_PACKETS = TOTAL_PACKETS+%s
WHERE B_GROUP = %s",(packets,blood_group))
```

5.CONTACT

CONTACT_ID	INT
B_GROUP	VARCHAR
C_PACKETS	INT
F_NAME	VARCHAR
ADRESS	VARCHAR

CODE:

```
cur.execute("INSERT INTO CONTACT(B_GROUP,C_PACKETS,F_NAME,ADRESS)
VALUES(%s, %s, %s, %s)",(bgroup, bpackets, fname, adress))
```

5.3 STAGE 3: AUTHORISATION

1.REGISTER

```
class RegisterForm(Form):
    name =
StringField('Name',[validators.DataRequired(),validators.Length(min=1,max=25)])
    email
StringField('Email',[validators.DataRequired(),validators.Length(min=10,max=50)])
 password = PasswordField('Password', [ validators.DataRequired(),
        validators.EqualTo('confirm',message='Password do not match')])
 confirm = PasswordField('Confirm Password')
```

Then fill the register form .i.e.,insert into RECEPTION table.

2.LOGIN

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        e_id = request.form["e_id"]
        password_candidate = request.form["password"]
        cur = mysql.connection.cursor()

        result = cur.execute("SELECT * FROM RECEPTION WHERE E_ID = %s", [e_id])
```

```
        if result > 0:
            data = cur.fetchone()
            password = data['PASSWORD']
            if sha256_crypt.verify(password_candidate, password):
                session['logged_in'] = True
                session['e_id'] = e_id
                flash('You are now logged in', 'success')
                return redirect(url_for('dashboard'))
            else:
                error = 'Invalid login'
                return render_template('login.html', error=error)
            cur.close()
        else:
            error = 'Employee ID not found'
            return render_template('login.html', error=error)

    return render_template('login.html')
```

3.LOGOUT

```
# Check if user logged in
def is_logged_in(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if 'logged_in' in session:
            return f(*args, **kwargs)
        else:
            flash('Unauthorized, Please login!', 'danger')
            return redirect(url_for('login'))
    return wrap

#Logout
@app.route('/logout')
@is_logged_in
def logout():
    session.clear()
    flash('You are now logged out', 'success')
    return redirect(url_for('index'))
```

5.4 STAGE 4: VIEW DATA

1.Donorlogs

```
cur = mysql.connection.cursor()

result = cur.execute("SELECT * FROM DONOR")

logs = cur.fetchall()
```

```
if result>0:
        return render_template('donorlogs.html',logs=logs)
else:
        msg = ' No requests found '
        return render_template('donorlogs.html',msg=msg)
cur.close()
```

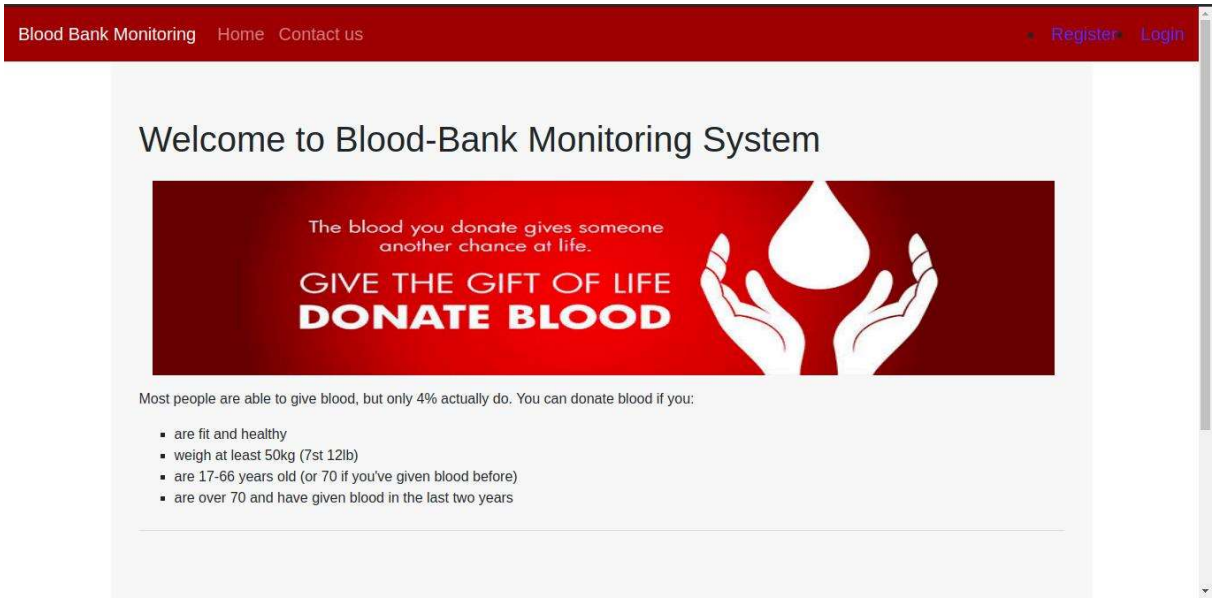
2.Notifications

```
cur = mysql.connection.cursor()
result = cur.execute("SELECT * FROM CONTACT")
requests = cur.fetchall()
if result>0:
        return render_template('notification.html',requests=requests)
else:
        msg = ' No logs found '
        return render_template('notification.html',msg=msg)
cur.close()
```

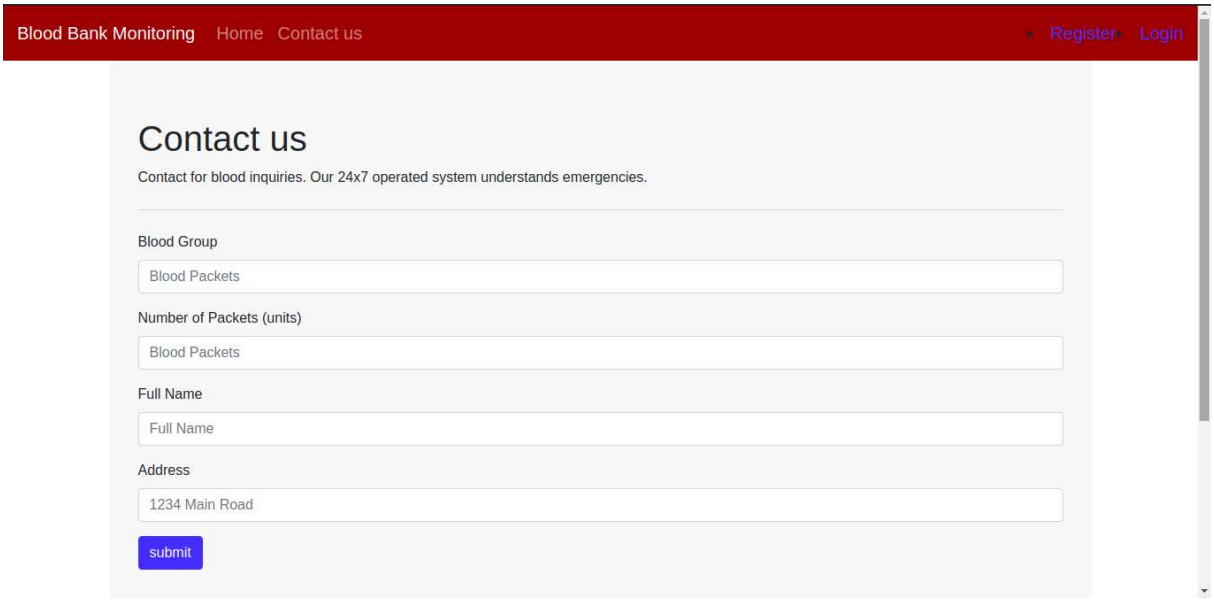
3.Dashboard

```
cur = mysql.connection.cursor()
result = cur.callproc('BLOOD_DATA')
details = cur.fetchall()
if result>0:
        return render_template('dashboard.html',details=details)
else:
        msg = ' Blood Bank is empty '
        return render_template('dashboard.html',msg=msg)
cur.close()
```

Some Pictures

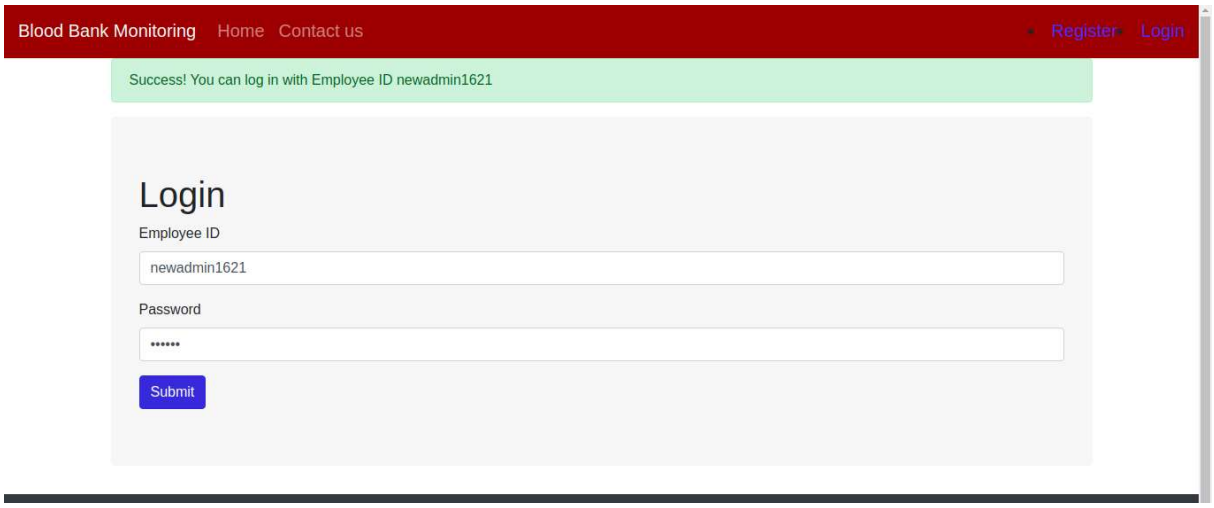


1. Contact Page

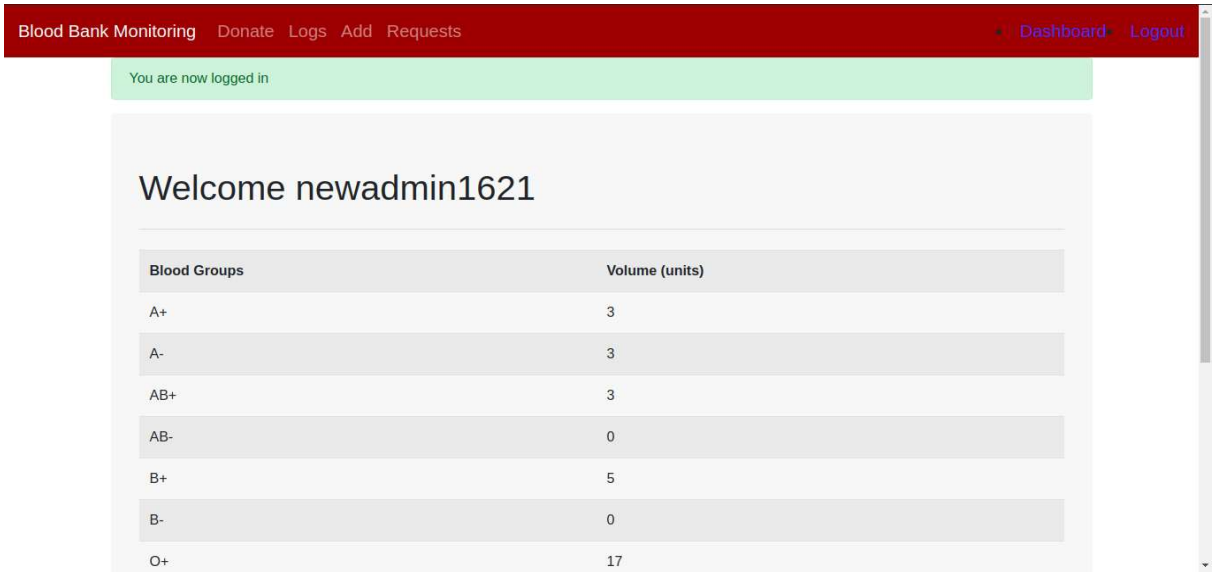


2. Employee Registration

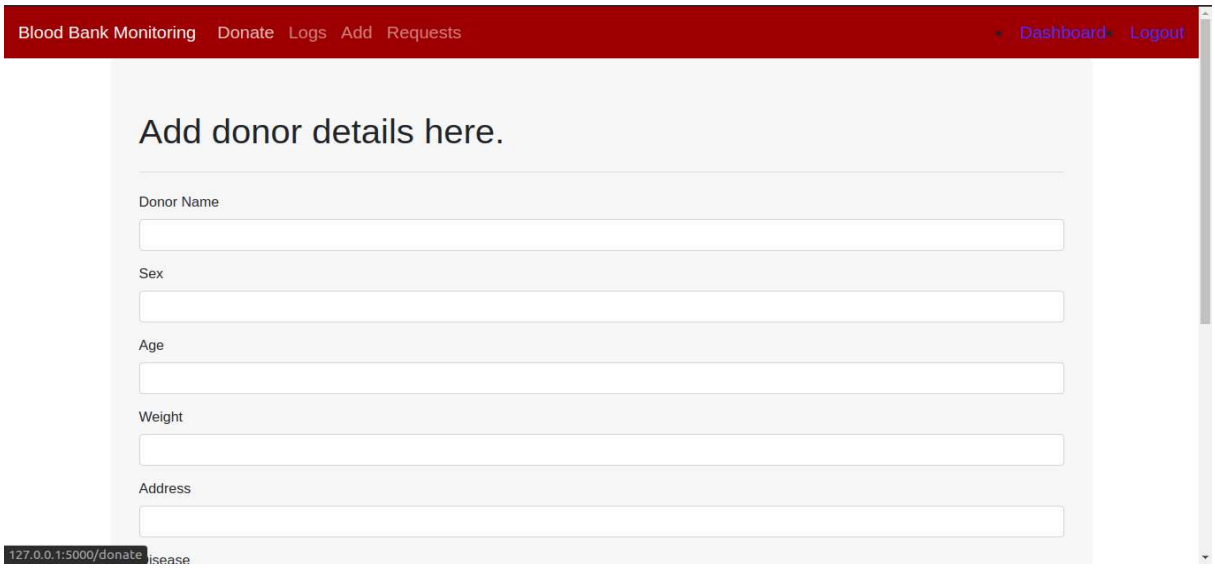
3. Employee Login



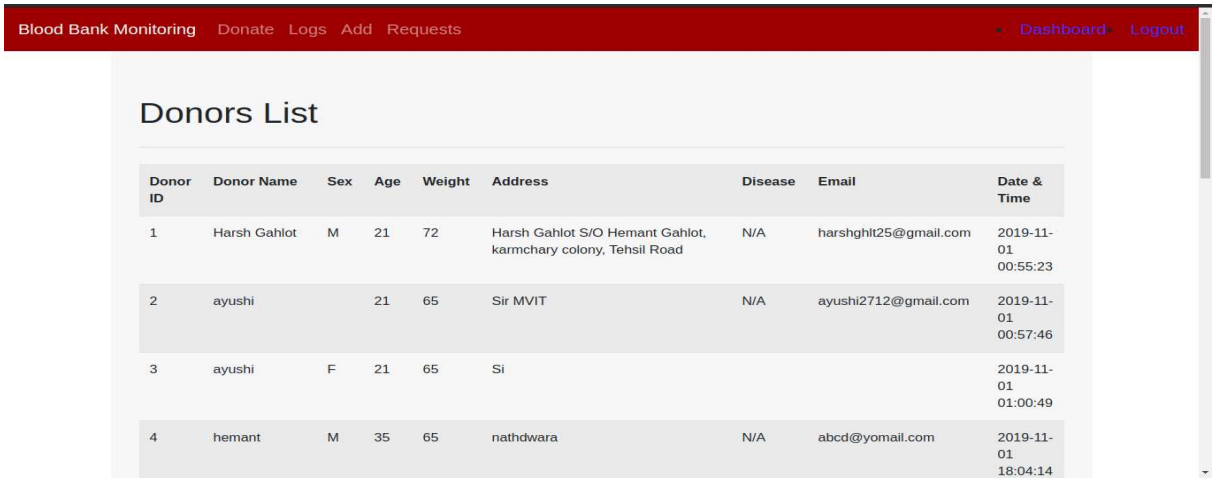
4. Dashboard



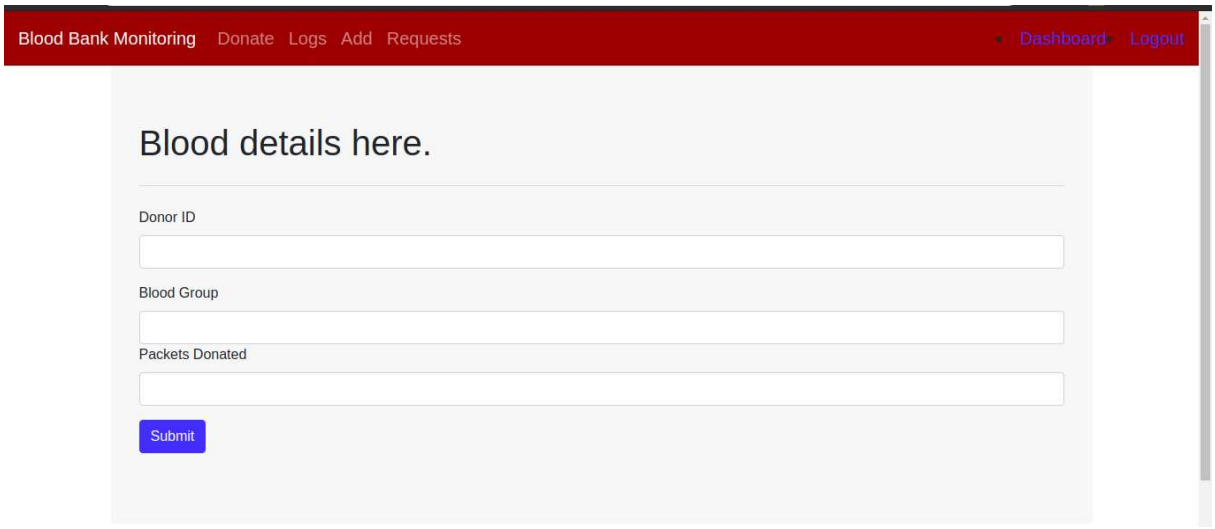
5. Donate Page



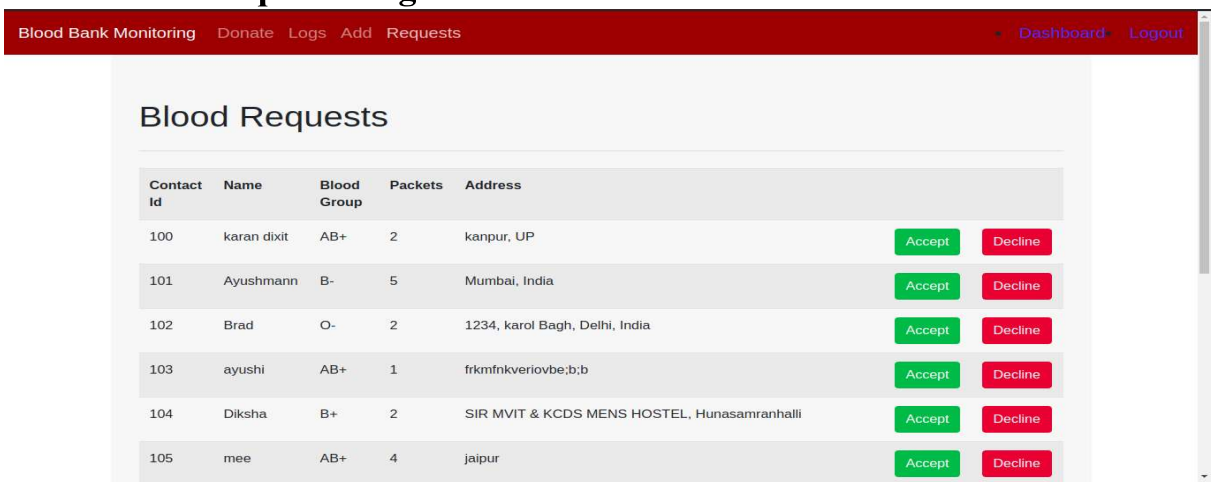
6. Donor Logs



7. Add Donor Blood Details



8. Blood Requests Page



CONCLUSION

Technology is introducing new innovations day by day, thus reducing the time required to do things. The proposed system can be used to reduce the time required to deliver required blood to the needy in cases of emergency. The web application provides a way of communication and synchronization between the Donor and the blood bank. It also provides them with the facility of communicating with the donors in emergency. The database is a vital aspect of the system. The database of the needy and the blood bank must be checked for consistency on regular basis for smooth working of the system

BIBLIOGRAPHY

BOOKS

[1] Python: For Beginners A Crash Course Guide To Learn Python in 1 Week by Timothy C. Needham

[2] MySQL: The Complete Reference by Vikram Vaswani

LINK

[1] http://www.mysqltutorial.org/

[2] https://www.javatpoint.com/mysql-tutorial

[3] https://www.coursera.org/learn/python-databases

[4] https://code.tutsplus.com/tutorials/creating-a-web-app-from-scratch-using-python-flask-and-mysql--cms-22972

[5] https://www.google.com/