PLASMA DONOR APPLICATION

Submitted by

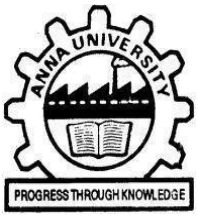
ATHIRA A (961719104004)
JISNA T (961719104006)
LEKSHMI MR (961719104007)
NIMMY BS (961719104009)
PARVATHY KS (961719104010)

in partial fulfilment for the award of the degreeof

BACHELOR OF ENGINEERING
in

COMPUTER SCIENCE AND ENGINEERING

NARAYANAGURU COLLEGE OF ENGINEERING
MANJALUMOODU-629151



ANNA UNIVERSITY: CHENNAI 600025
NOVEMBER 2022

BONAFIDE CERTIFICATE

Certified that this project report **"PLASMA DONOR APPLICATION"** is the bonafide work of **—ATHIRA A** (961719104004),**JISNA T** (961719104006) ,**LEKSHMI MR** (961719104007),**NIMMY BS** (961719104009) and **PARVATHY K S** (961719104010)‖ who carried out the project workunder my supervision.

SIGNATURE

Mr SYAM DEV R S

HEAD OF THE DEPARTMENT

Department of CSE
Narayanaguru College of engineering
Manjalumoodu,
Manjalumoodu,KanyakumarDistrict

SIGNATURE

Dr L NISHA EVANGELIN

ASSISTANT PROFESSOR

Department of CSE
Narayanaguru College of engineering
Kanyakumari District

Submitted for B.E degree viva-voice examination held at

NARAYANAGURU COLLEGE OF ENGINEERING, Manjalumoodu

on……..

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

This project itself is the acknowledgement to the intensity, passion, dedication, and technical brilliance of many individual who have guided us in this project to its completion.First and foremost we express our heartfelt gratitude to the AlmightyGod for giving us the opportunity to excel in our effort to complete the project in time.We are thankful for the blessings and support of chairman of our college Dr.Sidharthan G.We are deeply indebted to Prof. Dr. C.P.Jesuthanam, Principal, Narayanaguru College of Engineering and Technology, Manjalumoodu for the inspiration inculcated in us and for the apt guidance.We would also like to express our sincere thanks to Asst.Prof SyamDev R.S, Head of department, Computer Science and Engineering for the kind support and valuable guidance that proved to be the real support for the completion of project. We also express our sincere and heartfelt thanks to our Project guide Asst. Prof.Nisha Evangelin L, and our lab assistant for their motivation, inspiration and encouragement to undertake this work. We also extend our thanks to all staff members and friends of Narayanaguru College of Engineering for their direct and indirect cooperation to complete our project. Last but not the least we deliver our heartiest thanks to our Parents who had been a moral support and prayed for our success.

TABLE OF CONTENT

INTRODUCTION

1.1.Project Overview

The Blood Donation Agent is to create an e-Information about the donor and organization that are related to donating the blood. Through this application any person who is interested in donating the blood can register himself in the same way if any organization wants to register itself with this site that can also register. Moreover if any general consumer wants to make request blood online he can also take the help of this site. Admin is the main authority who can do addition, deletion, and modification if required. This project is aimed to developing an online Blood Donation Information. The entire project has been developed keeping in view of the distributed client server computing technology, in mind.

Admin is the main authority who can do addition, deletion, and modificationifrequired.The project has been planned to be having the view of distributed architecture, with centralized storage of the database. The application for the storage of the data has been planned. Using the constructs of MS-SQL Server and all the user interfaces have been designed using the ASP.Net technologies. The database connectivity is planned using the ―SQL Connection‖ methodology. The standards of security and data protective mechanism have been given a big choice for proper usage.

1.2 Purpose

The purpose of the online blood donation application is that, any men who is in risk of getting plasma ,he can easily fetch the required one with this application. They don’t have to search here and there for the blood. With in a short period of time he can get the blood.

LITERATURE SURVEY

2.1 Existing System

- Cannot Upload and Download the latest updates.
- No use of Web Services and Remoting.
- Risk of mismanagement and of data when the project is under development.
- Less Security.
- No proper coordination between different Applications and Users.

●Fewer Users – Friendly

- **Disadvantages**
- User friendliness is provided in the application with various controls.
- The system makes the overall project management much easier and flexible.

- Readily upload the latest updates, allows user to download the alerts by clicking the URL.
- There is no risk of data mismanagement at any level while the project development is under process.
- It provides high level of security with different level of authentication.

2.2 References

1.Safe blood and blood products. Module 1: Safe blood donation. Geneva: WorldHealthOrganization;2002.[17August2012].http://www.who.int/bloodsafety/transfusion_services/bts_learningmaterials/en/index.html.

2.Blood donor selection. Guidelines on assessing donor suitability for blood donation. Annex 3. Geneva: World Health Organization; 2012. [17 August 2012].http://www.who.int/bloodsafety/voluntary_donation/blood_donor_selection_counselling/en/[PubMed]

3.Aide-mémoire. Blood safety. Geneva: World Health Organization; 2002. [17 August 2012].http://www.who.int/bloodsafety/publications/who_bct_02_03/en/index.html.

4.WHO/IFRC. Towards 100% voluntary blood donation: A global framework for action. Geneva: World Health Organization; 2010. [17 August 2012].http://www.who.int/bloodsafety/publications/9789241599696/en/[PubMed]

5.The Melbourne Declaration on 100% voluntary non-remunerated donation of blood and blood components. Geneva: World Health Organization; 2009. [17 August 2012].http://www.who.int/worldblooddonorday/Melbourne_Declaration_VNRBD_2009.pdf.

6.WHO/CDC/IFRC. Blood donor counselling: Implementation guidelines. Geneva: World Health Organization; 2012. [17 August 2012].http://www.who.int/bloodsafety/voluntary_donation/blood_donor_selection_counselling/en/[PubMed]

7.Screening donated blood for transfusion-transmissible infections. Geneva: World Health Organization; 2010. [17 August 2012].http://www.who.int/bloodsafety/publications/bts_screendondbloodtransf/en/index.html.[PubMed]

8.Reiss RF. Blood donor well-being: a primary responsibility of blood collection agencies. Annals of Clinical & Laboratory Science. 2011;41(1):3–7. [PubMed]

9.Global Database on Blood Safety. Summary report 2011. Geneva: World Health Organization; 2011. [22 August 2012].http://www.who.int/entity/bloodsafety/global_database/GDBS_Summary_Report_2011.pdf.

10.Boulton F. Evidence-based criteria for the care and selection of blood donors, with some comments on the relationship to blood supply and emphasis on the management of donation-induced iron depletion. Transfusion Medicine. 2008;18:13–27. [PubMed]

11.Eder A, et al. Selection criteria to protect the blood donor in North America and Europe: past (dogma), present (evidence), and future (hemovigilance). Transfusion Medicine Reviews. 2009;23(3):205– 220. [PubMed]

12.Moreno J. ―Creeping precautionism‖ and the blood supply. Transfusion. 2003;43:840–842. [PubMed]

13.Farrugia A. The mantra of blood safety: time for a new tune? VoxSanguinis. 2004;86:1–7. [PubMed]

14.Fifty-Eighth World Health Assembly. Resolution WHA58.13: *Blood safety: proposal to establish World Blood Donor Day*. Geneva: World Health Organization; 2005. [17 August 2012].http://www.who.int/bloodsafety/resolutions/en/index.html&http://www.who.int/entity/bloodsafety/BTS_ResolutionsAdopted.pdf.

15.Sixty-Third World Health Assembly. Resolution WHA63.12: *Availability, safety and quality of blood products*. Geneva: World Health Organization; 2010. [17 August 2012].http://www.who.int/bloodsafety/resolutions/en/index.html&http://www.who.int/entity/bloodsafety/BTS_ResolutionsAdopted.pdf.

16.WHO handbook for guideline development. Geneva: World Health Organization; 2012. [10 August 2012].http://apps.who.int/iris/bitstream/10665/75146/1/9789241548441_eng.pdf.

17.WHO/CDC. Donor selection and counselling. Report of inter-regional workshop on blood donor selection and donor counselling for priority countries in the African and Eastern Mediterranean regions. Geneva: World Health Organization; Jun, 2011. [17 August 2012]. 2011. (http://www.who.int/entity/bloodsafety/WHO-CDC_ReportDonorSelectionCounsellingWorkshopKenya.2011.pdf.

18.Heddle NM. Evidence-based decision making in transfusion medicine. VoxSanguinis. 2006;91(3):214–220. [PubMed]

19.Vamvakas EC. Evidence-based practice of transfusion medicine: is it possible and what do the words mean? Transfusion Medicine Reviews. 2004;18(4):267–278. [PubMed]

20.Franklin IM. Is there a right to donate blood? Patient rights; donor responsibilities. Transfusion Medicine. 2007;17(3):161–168. [PubMed]

2.3 Problem Statement

Despite advancement in technology,today's blood bank system are running in manual system.As much,there is a prevalent problem in the availability of needed blood types. For instance, when a person needs a certain type of blood and this type is not available in the hospital,family members send message through social media to those who can donate to them and this process takes longer than the life of the patient to the most dangerous.In addition,It seems that there is lack of proper documentation about blood donors and its medical history.This may lead to blood bag contamination and may affect the blood transfusion safety.

Generally,this study aims to determine how the use of online bank management system enhance blood transfusion safety.Subsequently,this study seeks to answer the following specific problems:

1)What are the expected system users?

2)What is the system architecture to use?

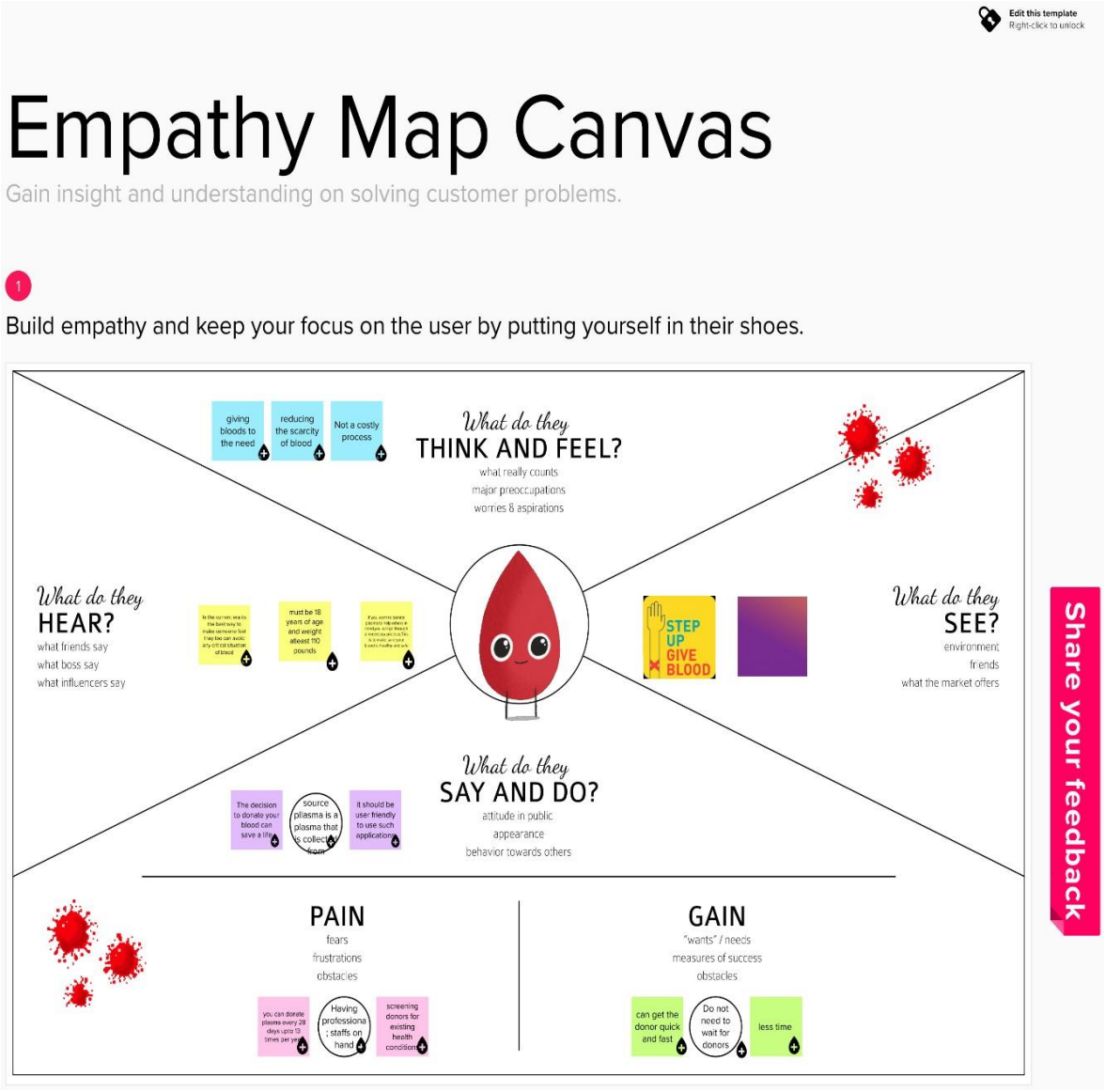
3)What is the level of risk in blood transfusion in Oman?

4)To what extend the online blood donation system can enhance blood transfusion safety?

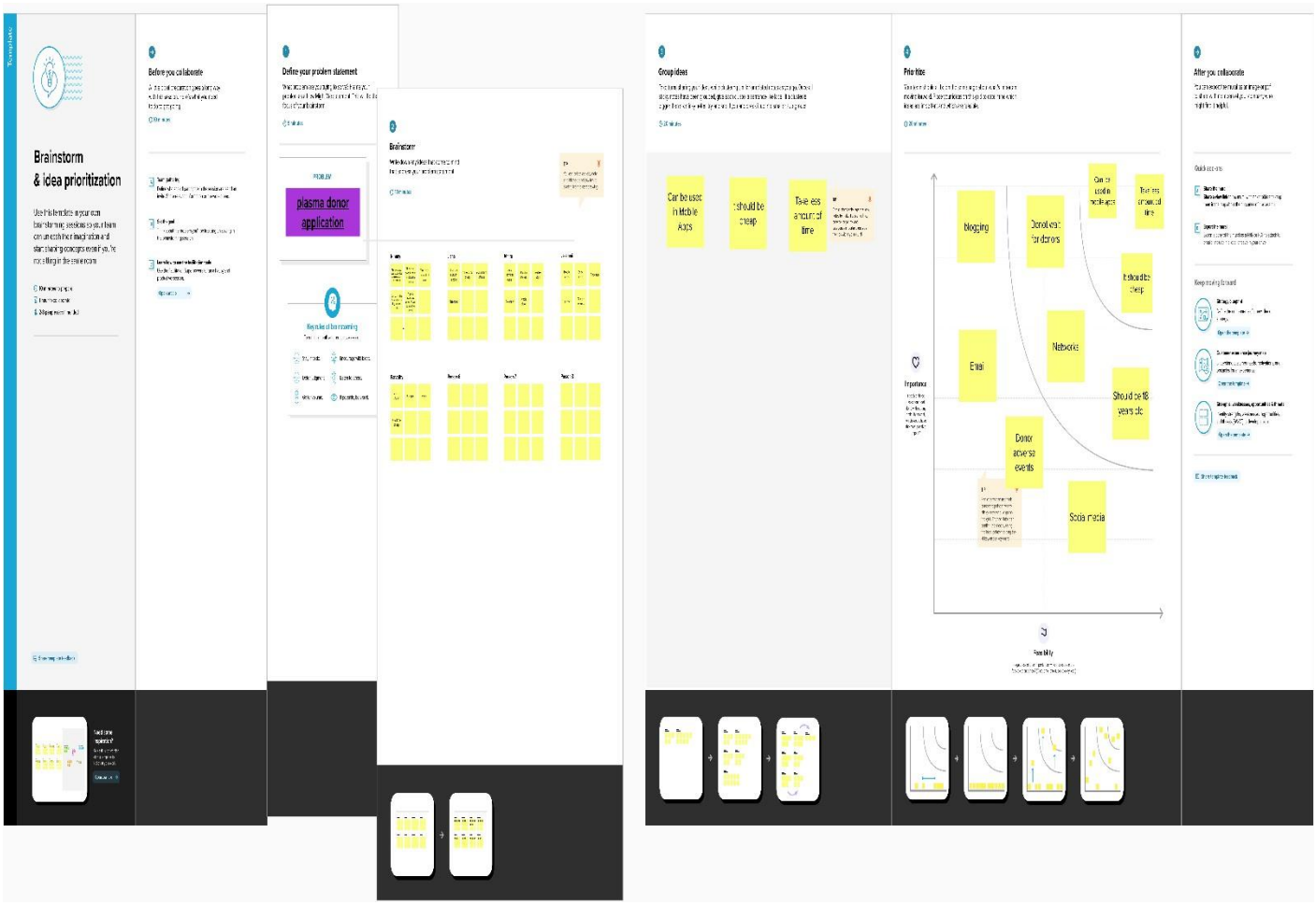
5)Is there a significant difference in the level of risk in blood transfusion between manual blood and online blood donation system?

IDEATION& PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS



3.2 IDEATION & BRAINSTORMING



3.3 PROPOSED SOLUTION

To debug the existing system, remove procedures those cause data redundancy, make navigational sequence proper. To provide information about audits on different level and also to reflect the current work status depending on organization/auditor or date. To build strong password mechanism.

Advantages

- User friendliness I provided in the application with various controls.
- The system makes the overall project management much easier and flexible.
- Readily upload the latest updates ,allows user to download the alerts by clicking the url.
- It provides high level of security with different level of authentication.

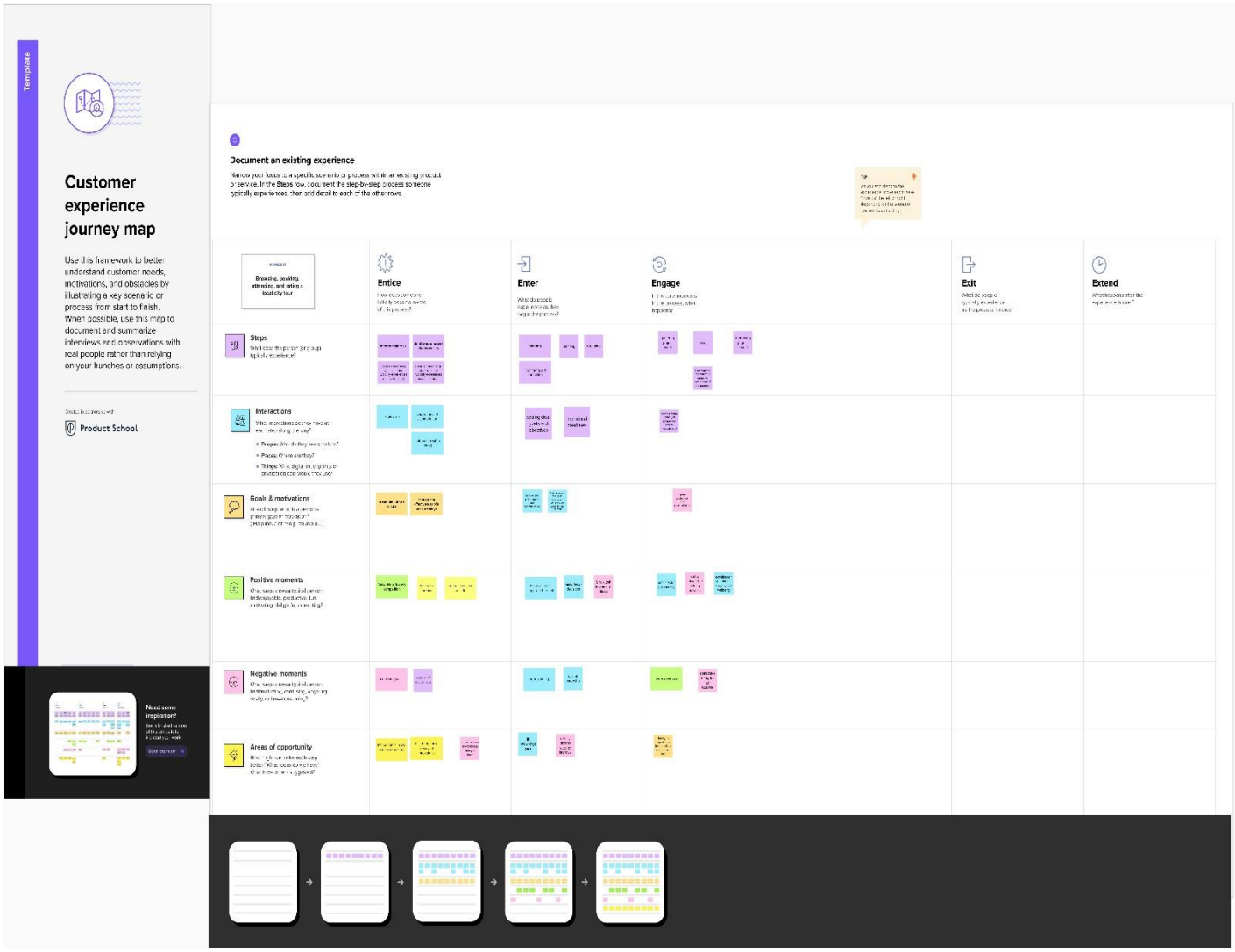
3.4 PROBLEM SOLUTION FIT

- Blood donation App notifies the latest news or information about blood donation criteria and method details.
- A better connection via the mobile application at places where is slow hospitality services.
- The system provides authenticated and authorized features to the current system where private and confidential data can only be viewed by authorized users.
- The system provides the recording function for every process of the blood in order to keep track of the blood stock accurately.

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

CUSTOMER JOURNRY MAP



PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS

A data flow diagram is graphical tool used to describe and analyze movement of data through a system. These are the central tool and the basis from which the other components are developed. The transformation of data from input to output, through processed, may be described logically and independently of physical components associated with the system. These are known as the logical data flow diagrams.

The physical data flow diagrams show the actual implements and movement of data between people, departments and workstations. A full description of a system actually consists of a set of data flow diagrams. Using two familiar notations Yourdon, Gane and Sarson notation develops the data flow diagrams. Each component in a DFD is labeled with a descriptive name. Process is further identified with a number that will be used for identification purpose.

The development of DFD'S is done in several levels. Each process in lower level diagrams can be broken down into a more detailed DFD in the next level. The lop-level diagram is often called context diagram. It consists a single process bit, which plays vital role in studying the current system.

The process in the context level diagram is exploded into other process at the first level DFD.

The idea behind the explosion of a process into more process is that understanding at one level of detail is exploded into greater detail at the next level. This is done until further explosion is necessary and an adequate amount of detail is described for analyst to understand the process.

Larry Constantine first developed the DFD as a way of expressing system requirements in a graphical from, this lead to the modular design.

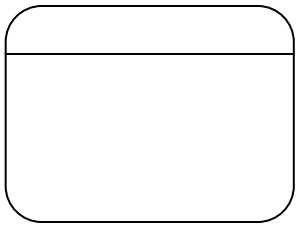
A DFD is also known as a ―bubble Chart‖ has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design. So it is the starting point of the design to the lowest level of detail. A DFD consists of a series of bubbles joined by data flows in the system.

DFD SYMBOLS:

In the DFD, there are four symbols

- A square defines a source(originator) or destination of system data
- An arrow identifies data flow. It is the pipeline through which the information flows
- A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.

An open rectangle is a data store, data at rest or a temporary repository of data



Process that transforms data flow.

Source or Destination of data

Data flow

Data Store

CONSTRUCTING A DFD:

Several rules of thumb are used in drawing DFD'S:

1. Process should be named and numbered for an easy reference. Each name should be representative of the process.
2. The direction of flow is from top to bottom and from left to right. Data traditionally flow from source to the destination although they may flow back to the source. One way to indicate this is to draw long flow line back

to a source. An alternative way is to repeat the source symbol as a destination. Since it is used more than once in the DFD it is marked with a short diagonal.

3.When a process is exploded into lower level details, they are numbered.

4.The names of data stores and destinations are written in capital letters. Process and dataflow names have the first letter of each work capitalized

A DFD typically shows the minimum contents of data store. Each data store should contain all the data elements that flow in and out.

Questionnaires should contain all the data elements that flow in and out. Missing interfaces redundancies and like is then accounted for often through interviews.

SAILENT FEATURES OF DFD'S

1. The DFD shows flow of data, not of control loops and decision are controlled considerations do not appear on a DFD.
2. The DFD does not indicate the time factor involved in any process whether the dataflow take place daily, weekly, monthly or yearly.
3. The sequence of events is not brought out on the DFD.

TYPES OF DATA FLOW DIAGRAMS

1. Current Physical
2. Current Logical

3. New Logical
4. New Physical

CURRENT PHYSICAL:

In Current Physical DFD proecess label include the name of people or their positions or the names of computer systems that might provide some of the overall system-processing label includes an identification of the technology used to process the data. Similarly data flows and data stores are often labels with the names of the actual physical media on which data are stored such as file folders, computer files, business forms or computer tapes.

CURRENT LOGICAL:

The physical aspects at the system are removed as mush as possible so that the current system is reduced to its essence to the data and the processors that transform them regardless of actual physical form.

NEW LOGICAL:

This is exactly like a current logical model if the user were completely happy with he user were completely happy with the functionality of the current system but had problems with how it was implemented typically through the new logical model will differ from current logical model while having additional functions, absolute function removal and inefficient flows recognized.

NEW PHYSICAL:

The new physical represents only the physical implementation of the new system.

RULES GOVERNING THE DFD'S

PROCESS

1) No process can have only outputs.
2) No process can have only inputs. If an object has only inputs than it must be a sink.
3) A process has a verb phrase label.

DATA STORE

1) Data cannot move directly from one data store to another data store, a process must move data.
2) Data cannot move directly from an outside source to a data store, a process, which receives, must move data from the source and place the data into data store
3) A data store has a noun phrase label.

SOURCE OR SINK

The origin and /or destination of data.

1) Data cannot move direly from a source to sink it must be moved by a process
2) A source and /or sink has a noun phrase land

DATA FLOW

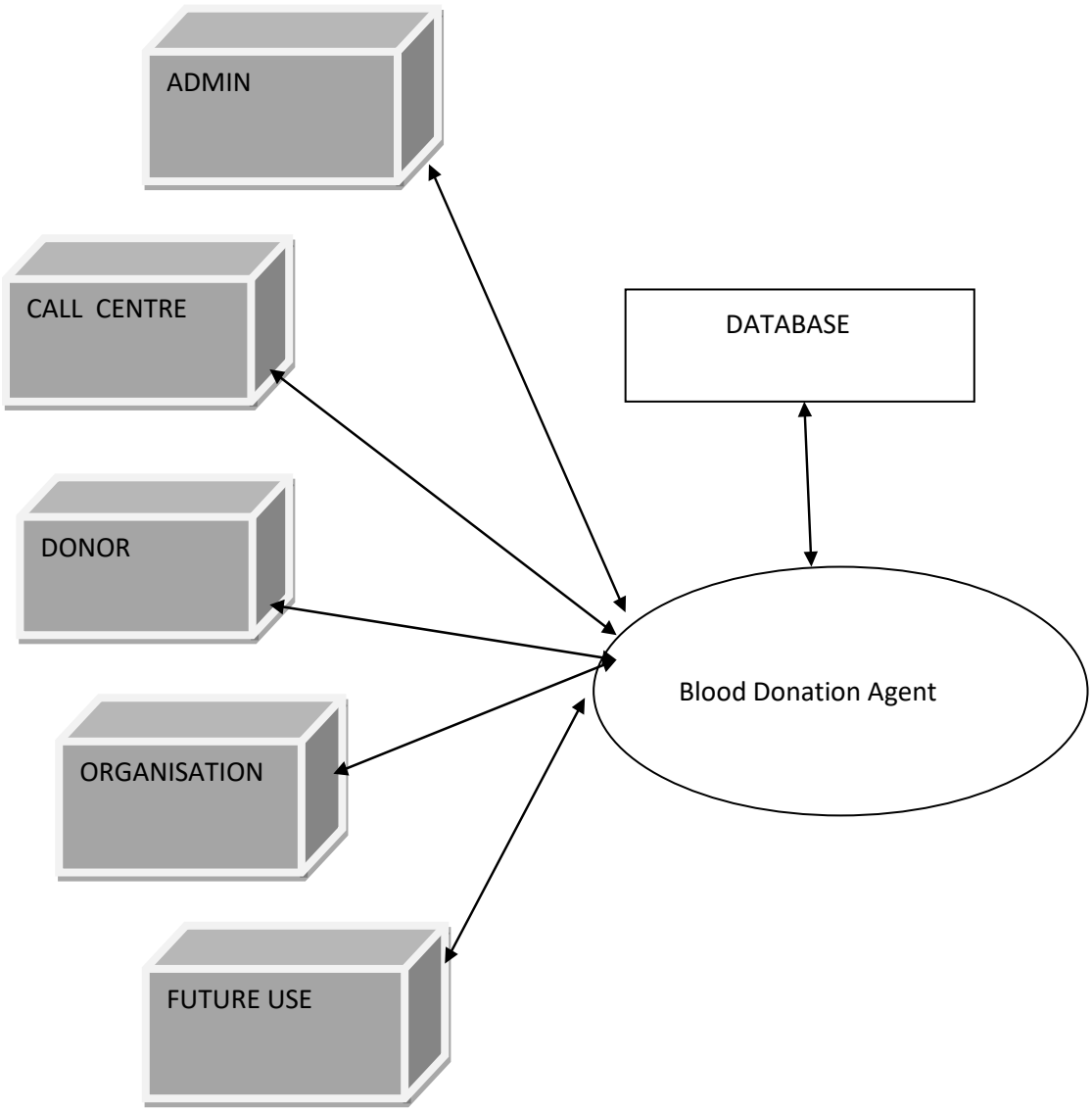
1) A Data Flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The later is usually indicated however by two separate arrows since these happen at different type.

2) A join in DFD means that exactly the same data comes from any of two or more different processes data store or sink to a common location.

3) A data flow cannot go directly back to the same process it leads. There must be atleast one other process that handles the data flow produce some other data flow returns the original data into the beginning process.

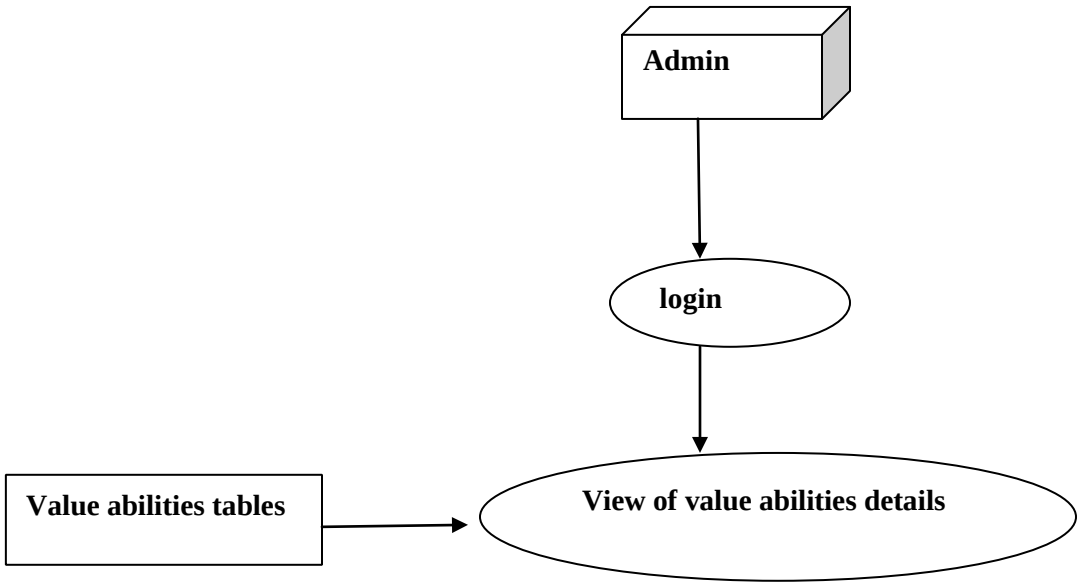
4) A Data flow to a data store means update (delete or change). A data Flow from a data store means retrieve or use

CONTEXT DIAGRAM

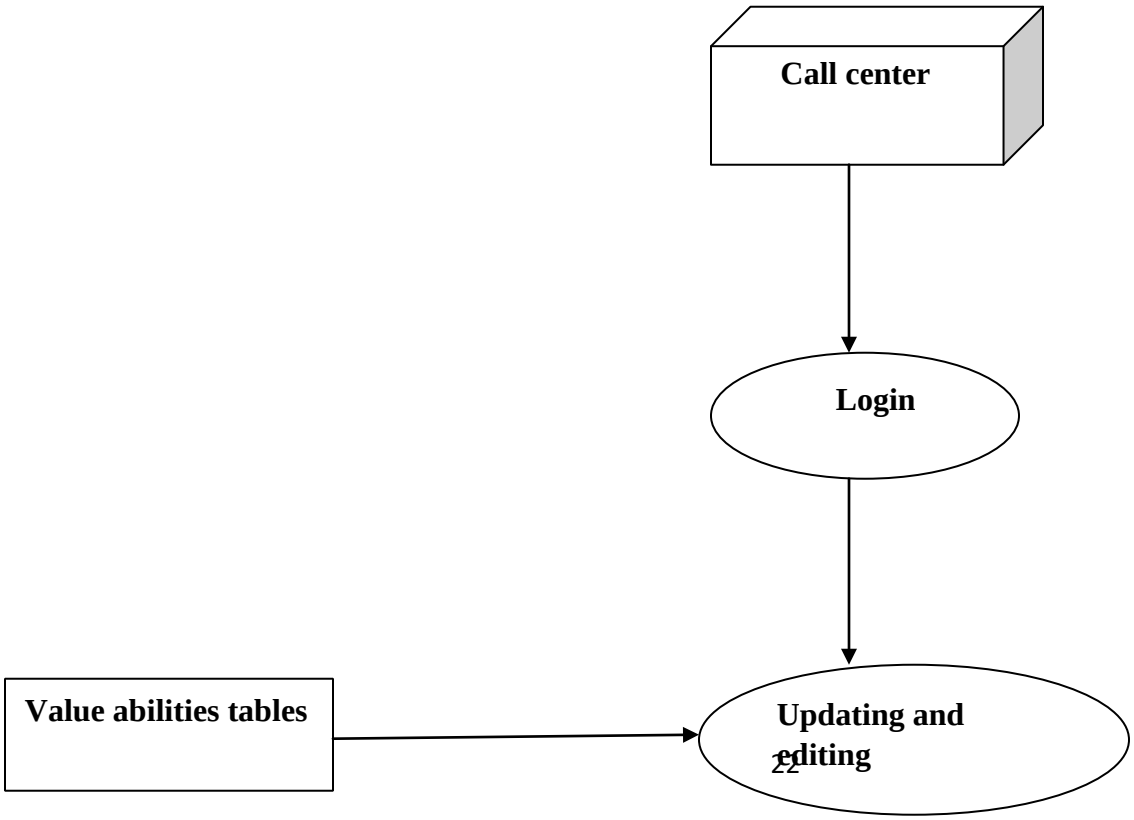


1st Level DFD's

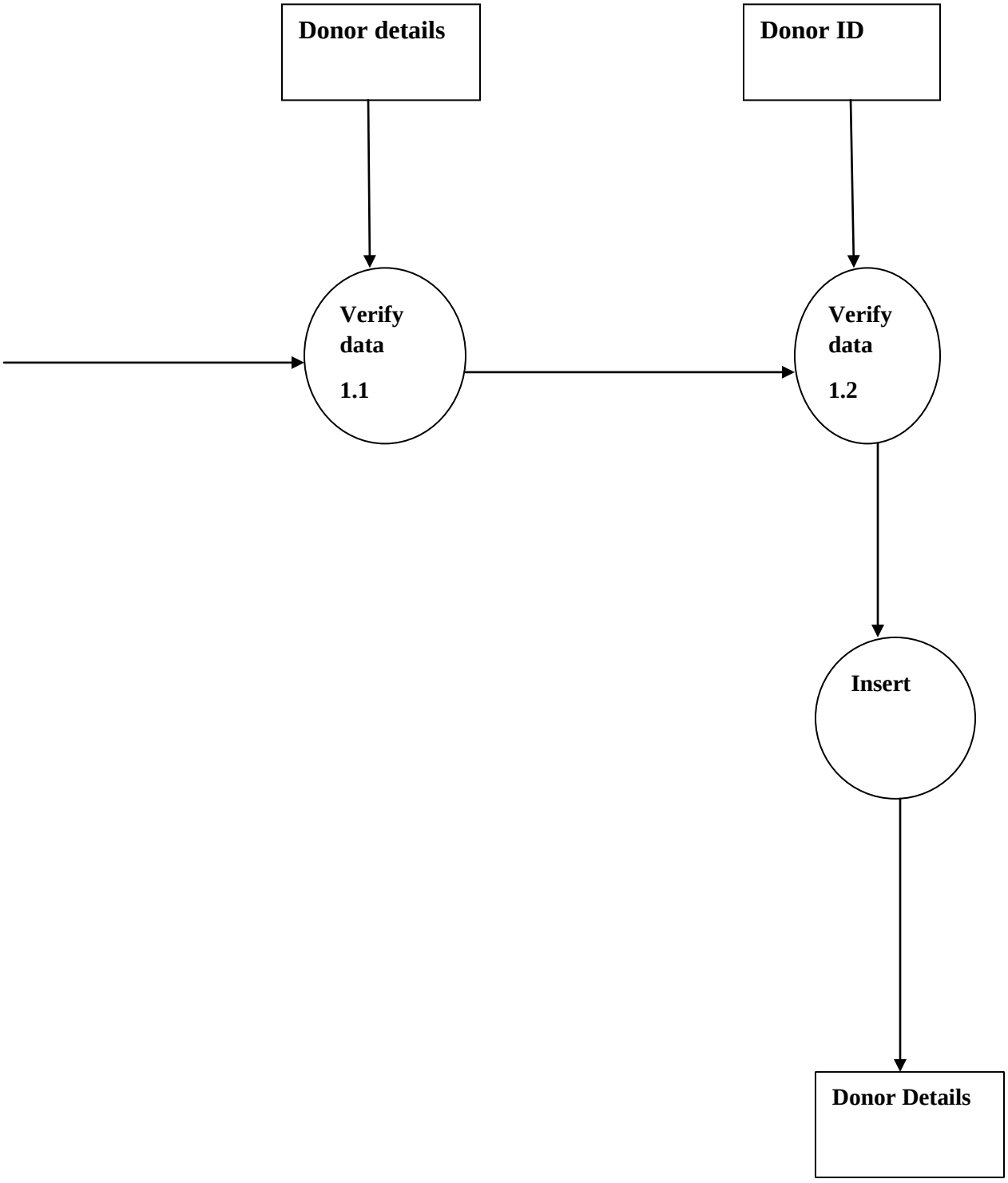
Level 1 DFD: For Admin Module



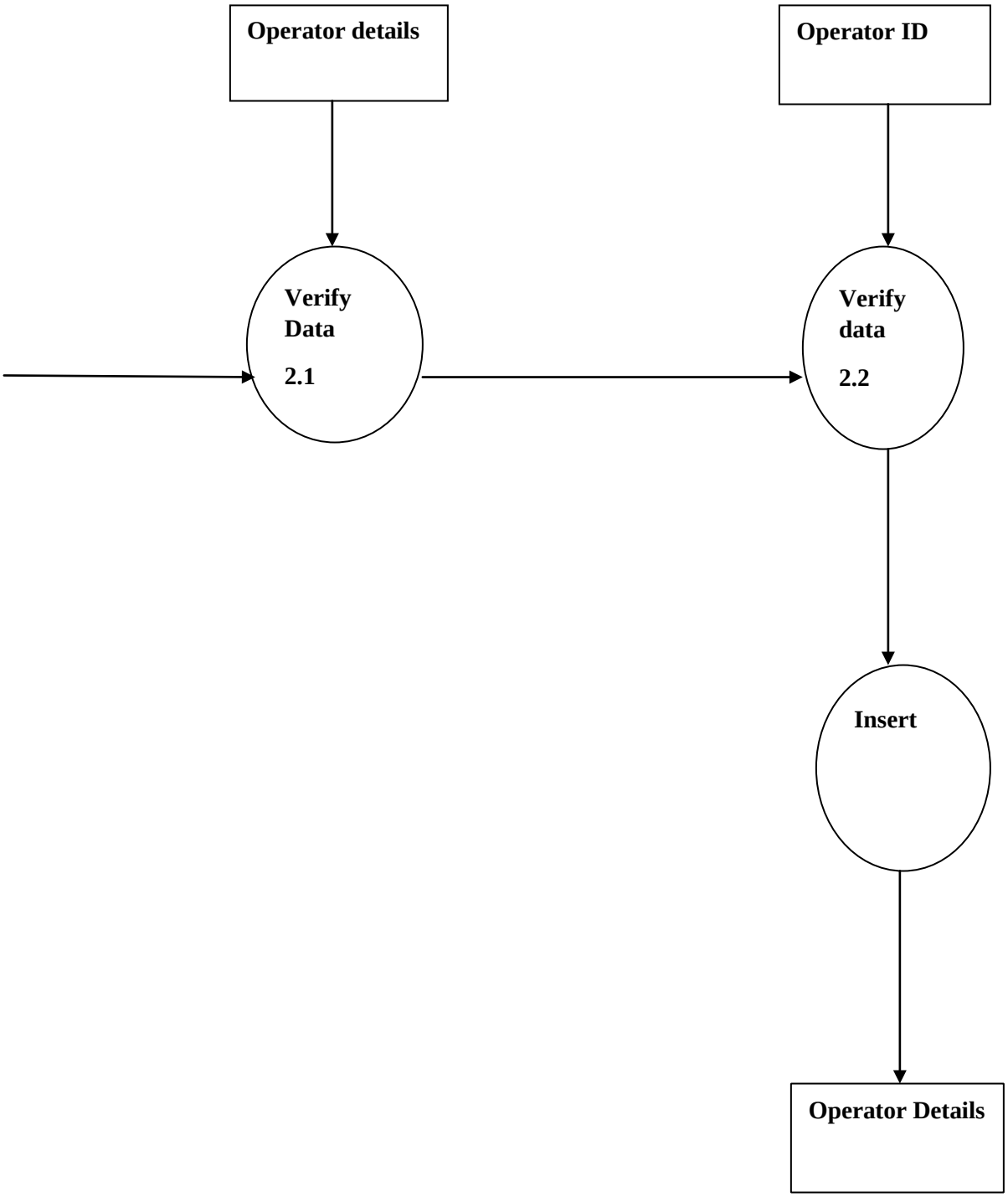
Level 1 DFD: For Users Module



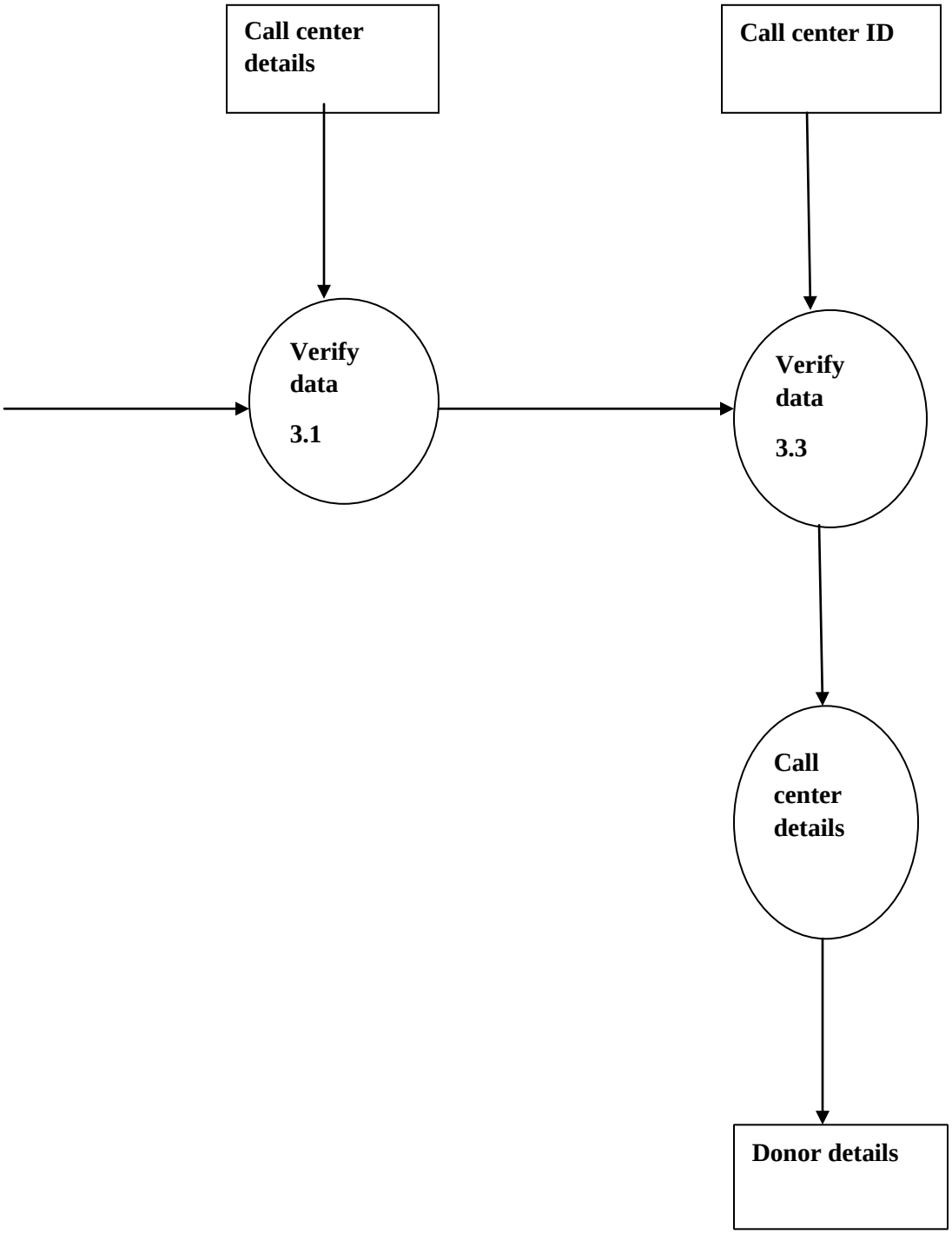
2nd DFD For Donor Creation



DFD for Operator Creation

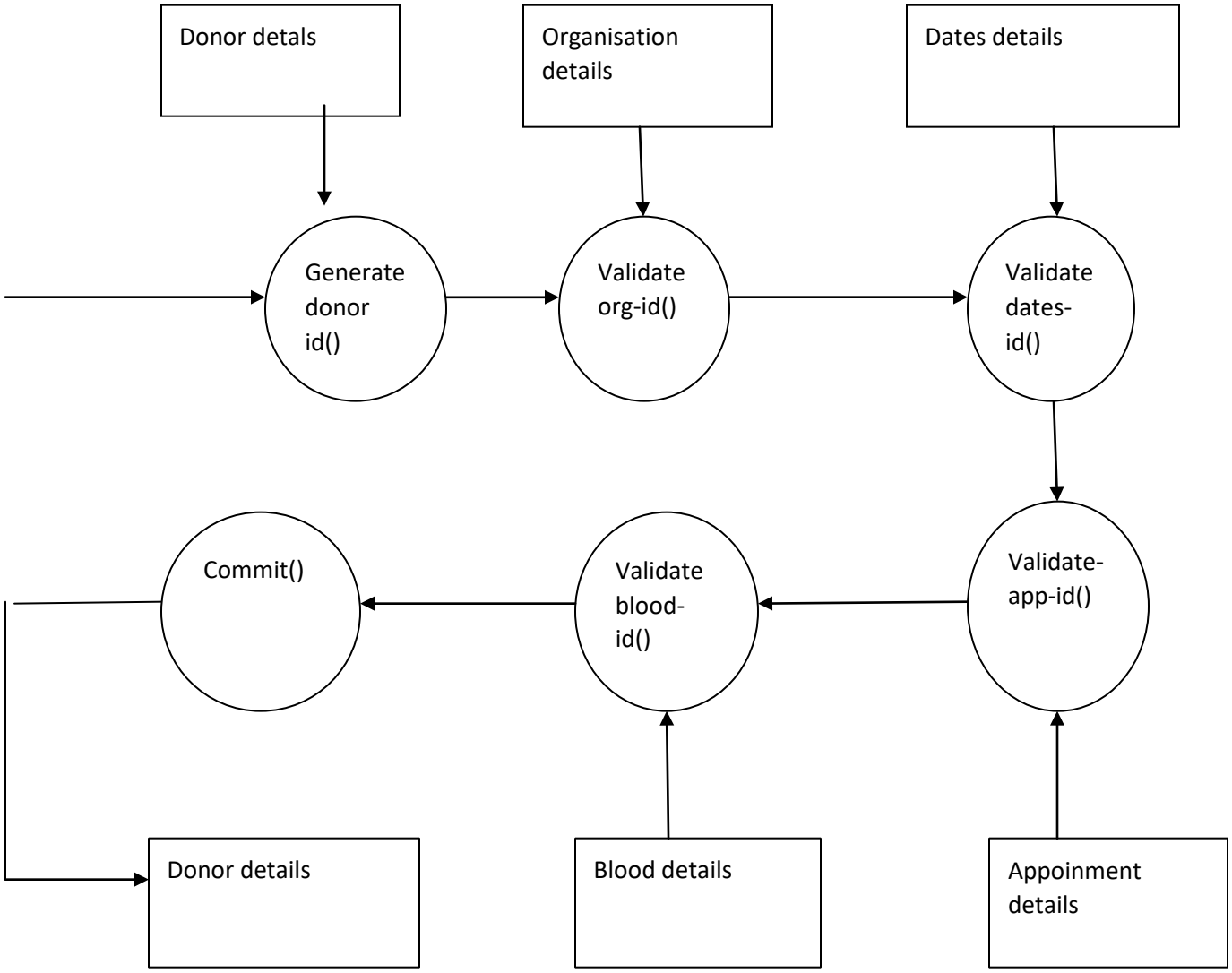


DFD For Call Center Creation

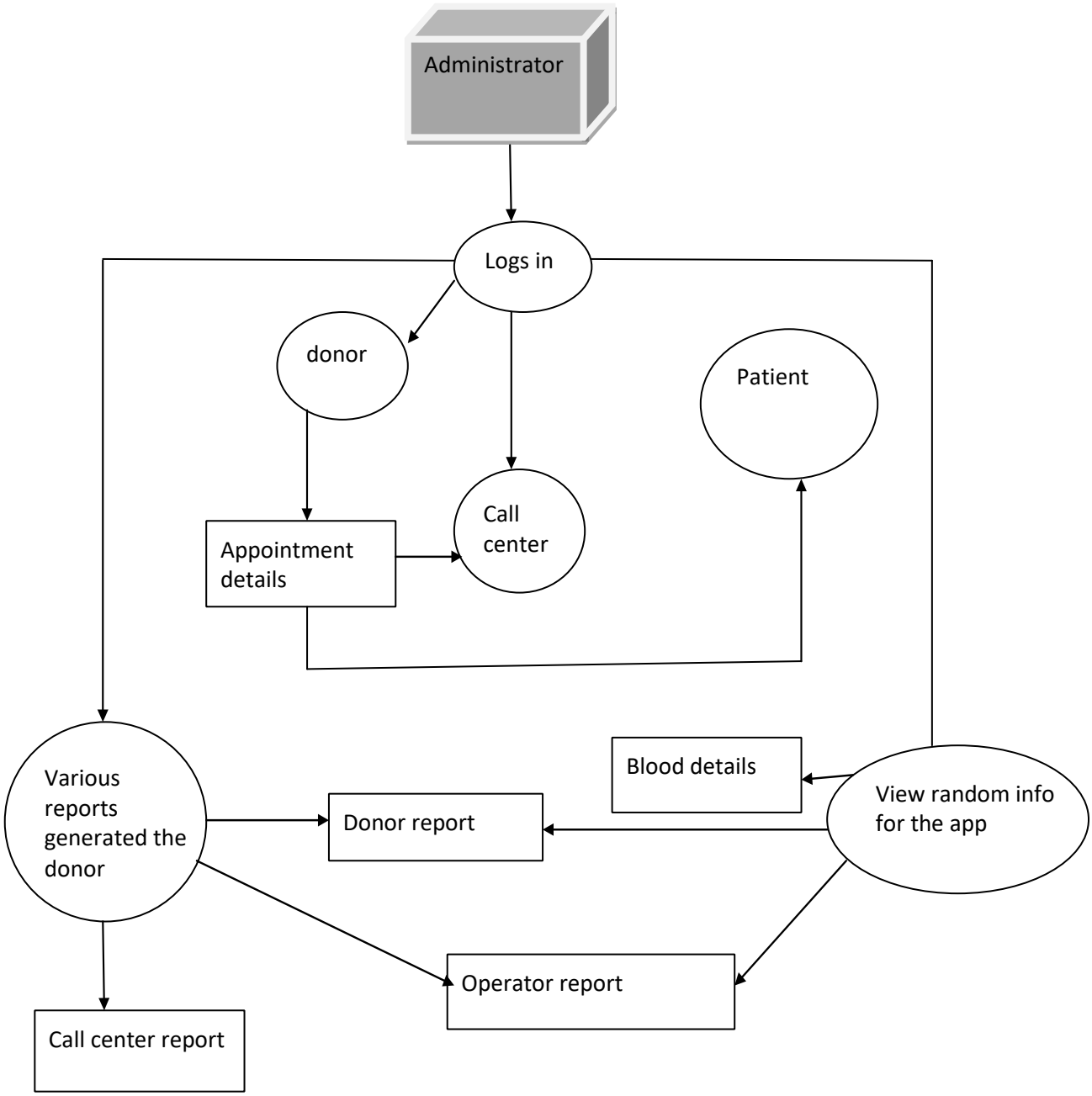


3rd Level DFD'S

DFD For Donor Creation



Work Flow of All Modules



Database Design

Entities with Attributes:

1.Administration:

User Account:

- AccountID
- Username
- Password
- HintQuestion
- Answer
- RoleID
- Active

UserRole:

- RoleID
- RoleName
- Active

BDAState:

- StateID
- StateName
- StateCode
- StateDesc

- CountryID
- Active

Country:

- CountryID
- CountryName
- CountryCode

BDACity

- CityID
- CityName
- CityDesc
- CityCode
- StateID
- Active

:

BDALocation:

- LocationID
- LocationName
- LocationDesc
- LocationCode
- CityID ●Pin code
- Active

BloodGroup:

- BloodGroupID
- BloodGroup
- Description
- Active

BloodType:

- BloodTypeID
- TypeName
- Active **PersonalDetails:**
- UserAccountID
- FirstName
- MiddleName
- LastName
- Email
- DOB
- Weight
- Gender
- ImageURL
- BloodGroupID
- BloodType
- BloodType
- AddressID
- ContactNo_Office

- ContactNo_Residence
- MobileNo
- Active

EmployeeDetail:

- EmpId
- Name
- Address
- Phone
- Email
- Active**DonationFrequencies:**
- FrequencyID
- Frequency
- Description
- Active

Donor Preferred Organization:

- UserAcountID
- OrganisationID
- Active

Organisation:

- OrgID

- OrgName
- OrgType
- Email
- OrgAddrID
- OrgImageURL
- OrgDescription
- ContactNo
- MobileNo
- Active
- Comment **OrganisationType:**
- TypeID
- TypeName
- TypeDescription
- OrgImage
- Active

2.Data Dictionary

BDA_ Address

Sno	Column name	Data type	Constraint	reference
1	AddressID	Bigint	Primary Key	
2	AddressLine1	varchar(512)	Not null	
3	LocationID	Bigint	Foreign key	BDA_Location
4	CityID	Bigint	Foreign key	BDA_City
5	StateID	Bigint	Foreign key	BDA_state
6	CountryID	Bigint	Allow Null	BDA_Country
7	Zipcode	varchar(10)	Allow Null	
8	Active	Bit	Not Null	

BDA_BloodDonationDetails

Sno	Column name	Datatype	Constraint	reference
1	ID	Bigint	Primary Key	
2	DonorID	Bigint	Foreign key	BDA_BloodRequest
3	DonationDate	Datetime	Notnull	

4	TakerID	Bigint	Foreign key	BDA_UserAccount
5	OrgID	Bigint	Foreign key	Bda_Organisation
6	Quantity	varchar(10)	Allow Null	
7	Comment	varchar(1024)	Allow Null	
8	Active	Bit	Notnull	

BDA_BloodDonationPreference

Sno	Columnname	Datatype	Constraint	Reference
1	PreferenceID	bigint	Primary key	
2	UserAccountID	bigint	Foreign key	BDA_UserAccount
3	DonationFrequencyID	tinyint	Foreign key	BDA_Donation Frequencies
4	WantToDonateWhitecells	bit	AllowNull	
5	Active	bit	AllowNull	

BDA_BloodGroup

Sno	Columnname	Datatype	Constraint	reference
1	BloodGroupID	tinyint	Primary Key	
2	BloodGroup	varchar(10)	Not Null	
3	Description	varchar(1024)	Allow Null	

4	Active	bit	Not Null	

BDA_BloodRequest

Sno	Columnname	Datatype	Constraint	Reference
1	RequestID	int	Primary Key	
2	DonorId	bigint	Foreign key	BDA_UserAccount
3	OrgId	bigint	Foreign key	BDA_Organisation
4	Name	varchar(50)	Allow Null	
5	Email	varchar(50)	Allow Null	
6	Phone	varchar(20)	Allow Null	
7	Country	varchar(50)	Allow Null	
8	State	varchar(50)	Allow Null	
9	City	varchar(50)	Allow Null	
10	Location	varchar(50)	Allow Null	
11	BloodRequireAddress	varchar(100)	Allow Null	
12	BloodType	varchar(50)	Allow Null	
13	BloodGroup	varchar(50)	Allow Null	
14	AppDate	datetime	Allow Null	
15	ReqDate	datetime	Allow Null	

16	Status	varchar(10)	Allow Null	
17	RequestType	varchar(50)	Allow Null	

BDA_City

Sno	Columnname	Datatype	Constraint	Reference
1	CityID	bigint	Primary key	
2	CityName	varchar(100)	Not null	
3	CityDesc	varchar(1024)	allownull	
4	CityCode	varchar(5)	Allownull	
5	StateID	bigint	Foreign key	BDA_State
6	Active	bit	notnull	

BDA_BloodType

Sno	Columnnames	Datatypes	Constraint	reference
1	BloodTypeID	tinyint	Primary Key	
2	TypeName	varchar(50)	Not null	

3	TypeDesc	varchar(1024)	AllowNull	
4	Active	bit	Notnull	

BDA_Country

Sno	Columnname	Datatype	Constraint	Reference
1	CountryID	bigint	Primary key	
2	CountryName	varchar(100)	Not null	
3	CountryDesc	varchar(1024)	Allow Null	
4	CountryCode	varchar(5)	Allow Null	
5	Active	bit	Not null	

BDA_DonationFrequencies

Sno	Columnname	Datatype	Constraint	Reference
1	FrequencyID	tinyint	Primary key	
2	Frequency	varchar(50)	Not null	

3	Description	varchar(255)	Allow null	
4	Active	bit	Allow null	

BDA_DonorPreferredOrganisation

Sno	Columnname	Datatype	Constraint	Reference
1	UserAcountID	bigint	Primary key	
2	OrganisationID	bigint	Not null	
3	Active	bit	Allow null	

BDA_EmployeeDetail

Sno	Columnname	Datatype	Constraint	reference
1	EmpId	bigint	Primary Key	
2	Name	varchar(60)	Allow Null	
3	Address	varchar(150)	Allow Null	
4	Phone	varchar(20)	Allow Null	
5	Email	varchar(25)	Allow Null	
6	Active	tinyint	Allow Null	

BDA_FAQ

Sno	**Columnname**	**Datatype**	**Constraint**	**reference**
1	FaqID	bigint	Primary key	
2	Question	varchar(300)	Allow null	
3	Answer	varchar(1000)	Allow null	
4	Active	bit	Allow null	

BDA_Location

Sno	Columnname	Datatype	Constraints	reference
1	LocationID	bigint	Primary Key	
2	LocationName	varchar(100)	Not Null	
3	LocationDesc	varchar(1024)	Allow null	
4	LocationCode	varchar(5)	Allow null	
5	CityID	bigint	Allow null	
6	Pincode	varchar(10)	Allow null	
7	Active	bit	Allow null	

BDA_Organisation

Sno	Columnname	Datatype	Constraint	reference
1	OrgID	bigint	Primary key	
2	OrgName	varchar(100)	Not null	
3	OrgType	tinyint	Foreign key	BDA_Organisation Type
4	Email	varchar(50)	Allownull	
5	OrgAddrID	bigint	Foreign key	BDA_Address
6	OrgImageURL	varchar(155)	Allownull	
7	OrgDescription	varchar(1024)	Allownull	
8	ContactNo	varchar(20)	Allownull	
9	MobileNo	varchar(20)	Allownull	
10	Active	bit	Allownull	
11	Comment	varchar(512)	Allownull	

BDA_OrganisationType

Sno	**Columnname**	**Datatype**	**Constraint**	**Reference**
1	TypeID	tinyint	Primary key	
2	TypeName	varchar(50)	Not null	
3	TypeDescription	varchar(50)	Allow null	

4	OrgImage	varchar(300)	Allow null	
5	Active	bit	Allow null	

BDA_PreferredDonationDayTime

Sno	Columnnames	Datattype	Constarint	Reference
1	DonorPreferenceID	bigint	Primary Key	
2	WeekDay	varchar(10)	Not null	
3	TimeFrom	varchar(10)	Allow null	
4	TimeUpto	varchar(10)	Allow null	
5	UserAccountID	bigint	Allow null	
6	Comment	varchar(1024)	Allow null	
7	Active	bit	Not null	
8				

BDA_PersonalDetails

Sno	Columnname	Datatype	Constraint	Reference
1	UserAccountID	bigint	Primary Key	
2	FirstName	varchar(50)	Not Null	
3	MiddleName	varchar(50)	Allow Null	
4	LastName	varchar(50)	Allow Null	

5	Email	varchar(100)	Allow Null	
6	DOB	datetime	Notnull	
7	Weight	float	Allow Null	
8	Gender	varchar(6)	Not Null	
9	ImageURL	varchar(155)	Allow Null	
10	BloodGroupID	tinyint	Foreign key	BDA_Bloodgroup
11	BloodType	tinyint	Foreign key	BDA_BloodType
12	AddressID	bigint	Foreign key	BDA_Address
13	ContactNo_Office	varchar(20)	Allow Null	
14	ContactNo_Residence	varchar(20)	Allow Null	
15	MobileNo	varchar(20)	Not Null	
16	Active	bit	Allow Null	

BDA_UserRole

Sno	**Columnname**	**DAtatype**	**Constraint**	**Reference**
1	RoleID	tinyint	Primary key	
2	RoleName	varchar(50)	Not null	
3	RoleDesc	varchar(1024)	Allow null	

4	Active	bit	Not null	

BDA_State

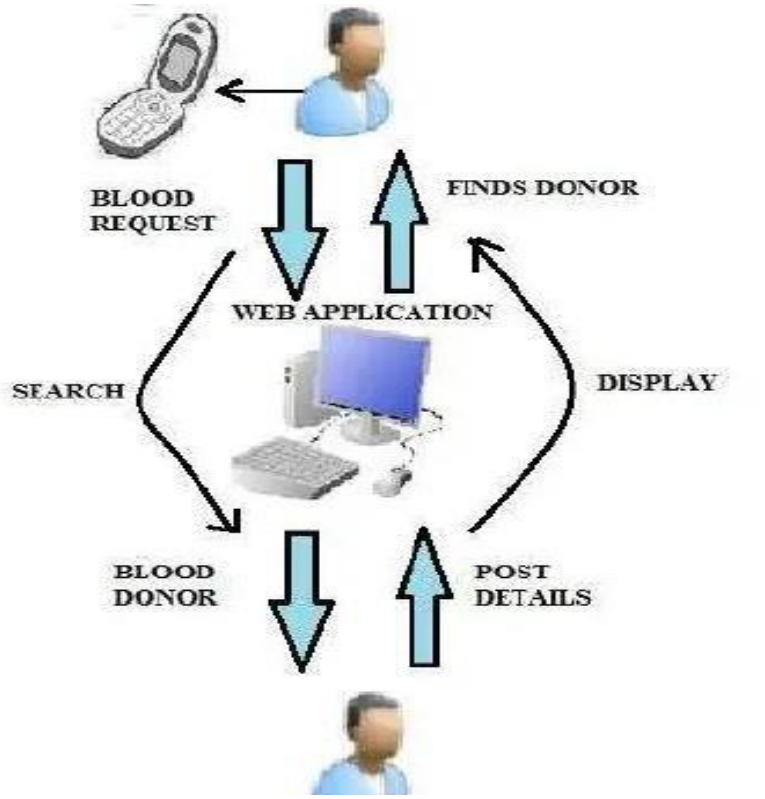
Sno	Columnname	Datatype	Constraint	Reference
1	StateID	bigint	Primary key	
2	StateName	varchar(100)	Not null	
3	StateCode	varchar(5)	Allow null	
4	StateDesc	varchar(1024)	Allow null	
5	CountryID	bigint	Foreign Key	BDA_Country
6	Active	bit	Not null	

BDA_UserAccount

Sno	Columnname	Datatype	Constraint	Reference
1	AccountID	bigint	Primary key	
2	Username	varchar(100)	Not null	
3	Password	varchar(100)	Not null	

4	UserDesc	varchar(1024)	Allow null	
5	HintQuestion	varchar(155)	Allow null	
6	Answer	varchar(155)	Allow null	
7	RoleID	tinyint	Foreign key	BDA_UserRole
8	Active	bit	Not null	

5.2 TECHNICAL ARCHITECTURE



PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

DATE

20/10/2022	**Html file**
22/10/2022	**Server file**
30/10/2022	**Front end**
7/11/2022	**Back end**
	Database connectivity
15/11/2022	

6.2 SPRINT DELIVERY SCHEDULE

DATE	
21/10/2022	Uploading html
25/10/2022	Uploading server file
1/11/2022	Uploading f ront end
10/11/2022	Uploading back end

6.3 REPORT FROM JIRA

CODING AND SOLUTIONING

7.1 FEATURE 1

The overall work is implemented in PYTHON.

Python is ahigh-level,general-purpose programming language. Its design philosophy emphasizescode readabilitywith the use ofsignificant indentation.

Python isdynamically-typedandgarbage-collected. It supports multipleprogramming paradigms, includingstructured(particularlyprocedural),object-orientedandfunctionalprogramming. It is often described as a "batteries included" language due to its comprehensivestandard library

Python is amulti-paradigm programming language.Object-orientedprogrammingandstructured programmingare fully supported, and many of their features support functional programmingandaspectorientedprogramming(includingmetaprogrammingand metaobjects).Many other paradigms are supported via extensions, including design by contract andlogic programming.

Python usesdynamic typingand a combination ofreference countingand a cycle-detecting garbage collector formemory management.It uses dynamicnameresolution(late binding), which binds method and variable names during program execution.

- Beautiful is better than ugly.
- Explicit is better than implicit. •Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

7.2 FEATURE 2

The work is done in Cloud Computing Platform.

Simply put, cloud computing is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the Internet (―the cloud‖) to offer faster innovation, flexible resources, and economies of scale.

There are four main types of cloud computing: private clouds, public clouds, hybrid clouds, and multi clouds. There are also three main types of cloud computing services: Infrastructure-as-a-Service (IaaS), Platformsas-a-Service (PaaS) , and Software-as-a-Service (SaaS)

Benefits of Cloud Computing

- Faster time to market. You can spin up new instances or retire them in seconds, allowing developers to accelerate development with quick deployments. ...
- Scalability and flexibility. ...
- Cost savings. ...
- Better collaboration. ... •Advanced security. ...
- Data loss prevention

TESTING

8.1 TEST CASES

Unit Testing

Unit testing focuses verification effort on the smallest unit of software design, the module. The unit testing we have is white box oriented and some modules the steps are conducted in parallel.

WHITE BOX TESTING

This type of testing ensures that,

- All independent paths have been exercised at least once
- All logical decisions have been exercised on their true and false sides
- All loops are executed at their boundaries and within their operational bounds
- All internal data structures have been exercised to assure their validity. To follow the concept of white box testing we have tested each form .we have created independently to verify that Data flow is correct, All conditions are exercised to check their validity, All loops are executed on their boundaries.

BASIC PATH TESTING

Established technique of flow graph with Cyclomatic complexity was used to derive test cases for all the functions. The main steps in deriving test cases were:

Use the design of the code and draw correspondent flow graph.

Determine the Cyclomatic complexity of resultant flow graph, using formula:

V(G)=E-N+2 or

V(G)=P+1 or

V(G)=Number Of Regions

Where V(G) is Cyclomatic complexity,

E is the number of edges,

N is the number of flow graph nodes,

P is the number of predicate nodes.

Determine the basis of set of linearly independent paths.

3. CONDITIONAL TESTING

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested. So that each path that may be generate on particular condition is traced to uncover any possible errors.

4. DATA FLOW TESTING

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used

only when some local variable were declared. The definition-use chain method was used in this type of testing. These were particularly useful in nested statements.

5. LOOP TESTING

In this type of testing all the loops are tested to all the limits possible. The following exercise was adopted for all loops:

- All the loops were tested at their limits, just above them and just below them.
- All the loops were skipped at least once.
- For nested loops test the inner most loop first and then work outwards.
- For concatenated loops the values of dependent loops were set with the help of connected loop.
- Unstructured loops were resolved into nested loops or concatenated loops and tested as above.

Each unit has been separately tested by the development team itself and all the input have been validated.

8.2 USER ACCEPTANCE TESTING

User acceptance testing (UAT), also called *application testing* or *end-user testing*, is a phase of software development in which the software is tested in the real world by its intended audience. UAT is often the last phase of thesoftware testingprocess and is performed before the tested software is released to its intended market. The goal of UAT is to ensure software can handle real-world tasks and perform up to development specifications.

In UAT, users are given the opportunity to interact with the software before its official release to see if any features have been overlooked or if it contains anybugs. UAT can be done in-house with volunteers, by paid test subjects using the software or by making the test version available for download as a free trial. The results from the early testers are forwarded to the developers, who make final changes before releasing the software commercially.

UAT is effective for ensuring quality in terms of time and software cost, while also increasing transparency with users. UAT also enables developers to work with real cases and data, and if successful, the process can validate business requirements.

User acceptance testing validates the testing done at the end of the development cycle. It is typically completed afterunit testing, quality assurance,system testingandintegration testing. The software may undergo other testing phases and be completely functional but might still not meet its requirements if it is not well received by its intended users. This can happen if software requirements were not clearly defined to the developers, if certain modifications made during development changed the scope of the project or if the software just was not ready to be tested in a dynamic, real-world environment. Overall, UAT safeguards against faulty, ineffective or unfinished software products being released.To be effective, UAT should be thorough and reflect user requirements, while also identifying potential problems not yet detected in previous tests. Without UAT, tested software may be released with bugs or a lack of a clearly defined goal for end users. These issues can be costly and potentially damaging to the software vendor's reputation.

ADVANTAGES & DISADVANTAGES

Advantages.

- Helps Blood Banks to automate blood donor and depository online.
- Encourages blood donors to donate.

- Helps people find blood donors in times of need.
- User can send blood request and they can check donor's list.
- Management can organize Blood Donation Camps.
- Blood donors can register through online.
- Management can manage donor's database by recording their physical and medical statistics.
- Medically qualified management in blood bank for storage and issuance of blood.

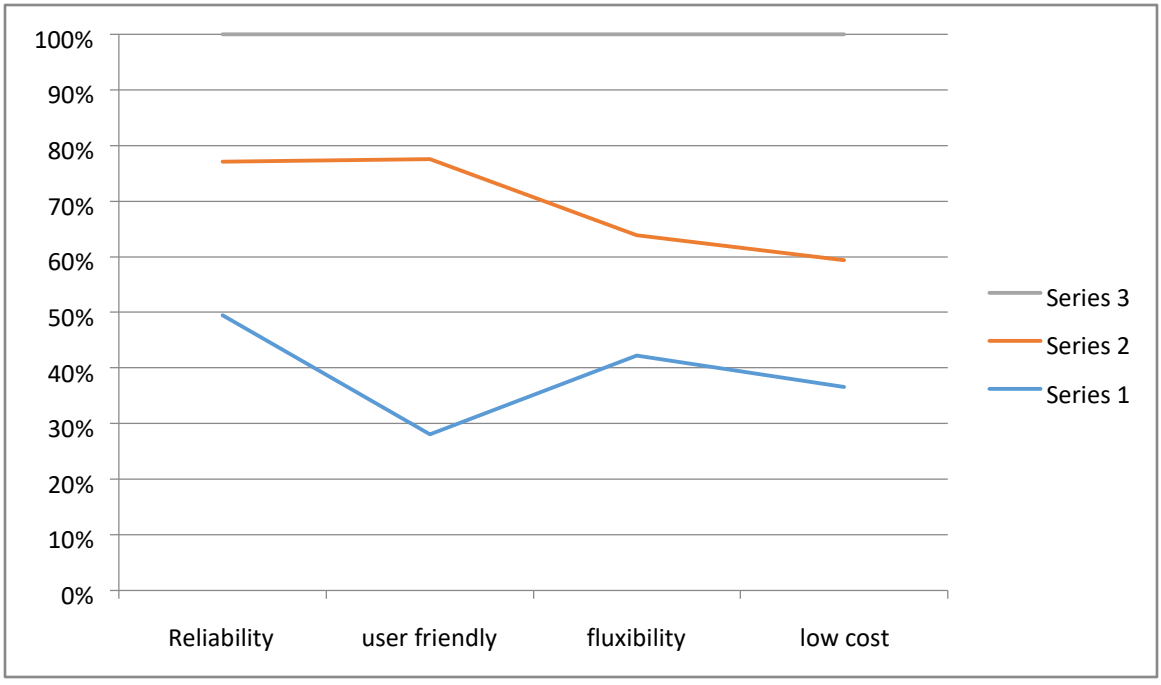
Disadvantages

- Manual document and data entry.
- Only web based system is available no mobile based system available.
- Less Security.
- No proper coordination between different Applications and

  Users.Cannot Upload and Download the latest updates at right time.

RESULTS

9.1 PERFOMANCE MATRICES



ADVANTAGES & DISADVANTAGES

ADVANTAGES

- User friendly
- Location free
- Easy access
- Low cost
- More Data

DISADVANTAGES

- Because we are in a era of internet security data breaches and loss of data may occur.
- The user who aren’t aware of the technologies, may be a victim of hackers**.**

CONCLUSION

This proposed Blood Bank system gives a reliable platform for both donors and acceptors.It is a web based system that can assists the information of blood bag during its handling in the blood bank. With this system, the user of this system can key in the result of blood test that has been conducted to each of the blood bag received by the blood bankEnsures hospitals have good supply or inventories of blood bags.List the availability of blood bags at any given time.Ability to manage the information of its blood donor.Alerts for blood requirement from registered donors.Auto-check if the person donated blood in the last 3 months. The process that takes place in the lab to make sure that donated blood, or blood products, are safe before they are used in blood transfusions and other medical procedures. Blood banking includes typing the blood for transfusion and testing for infectious diseases

FUTURE SCOPE

Its a software application to build such a way that it should suits for all type of blood banks in future. One important future scope is availability of location-based blood bank details and extraction of location-based donor's detail, which is very helpful to the acceptant people. Upgrading the UI that is more user-friendly which will help many users to access the website and also ensures that many plasma donors can be added into the community.

Using elastic load balancer, it helps to handle multiple requests at the same time which will maintain the uptime of the website with negligible downtime

APPENDIX

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

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

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


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


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

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



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

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




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



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

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

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

```
                    rows = cur.fetchall();


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

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

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

                    rows = cur.fetchall();


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



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

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

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

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

                 #messages = request.args['user']




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

                cur = con.cursor()
```

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

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

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

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

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

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

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



                @app.route('/logout')
                def logout():
                  # remove the username from the session if it is there
                session.pop('username', None)
  session.pop('logged_in',None)        try:
```

```
      session.pop('admin',None)          except
KeyError as e:
                        print("I got a KeyError - reason " +str(e))


                    return redirect(url_for('login'))


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

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

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

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

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

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

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



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

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

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

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


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



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


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

  bloodtypestotal =          {'apos':
            Apositive,'aneg':Anegative,'opos':Opositive,'oneg':Onegative,'bpos':Bpositive,'bneg':Bnegative,'abpo
            s':ABpositive,'abneg':ABnegative}




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


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

```
                return render_template('/adddonor.html')


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



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

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

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

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

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

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



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

                      finally:
                         flash('saved successfully')
                         return redirect(url_for('dashboard'))
                         con.close()

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



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

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

                          print(name,addr)



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

```
            con.commit()                    msg = "Record
successfully updated"                 except:
                            con.rollback()
                            msg = "error in insert operation"

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



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

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

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



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

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

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

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


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

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

```
        cur.execute('delete from users Where email=?',(useremail,))
flash('deleted user:'+useremail)                conn.commit()                   conn.close()
        session.pop('username', None)                   session.pop('logged_in',None)
                        return redirect(url_for('index'))

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



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

GitHub & Project Demo Link

https://youtu.be/bHvWPNtHY9Q (Demo link)

https://youtu.be/tzLUFZ3eUxA (presentation link)