LIST OF ABBREVIATIONS

EHR (electronic health record): A digital version of a patient's medical history that is stored electronically and can be accessed by authorized healthcare providers.

LIS (laboratory information system): A system that helps manage and track the processing of laboratory tests and results.

SOP (standard operating procedure): A set of instructions that outlines the steps to be followed when performing a specific task or process.

WMS (warehouse management system): A system that helps manage the storage and distribution of inventory, including tracking the location and status of stock.

API (application programming interface): A set of rules and protocols that allows different software systems to communicate with each other.

GUI (graphical user interface): A type of user interface that allows users to interact with a software application using visual elements such as icons and menus, rather than typing commands.

LIST OF FIGURES

Figure No.	**Name of Figure**	**Page No.**
Figure 3.1	Software Development Life Cycle (SDLC)	9
Figure 4.2	Flow Chart	11
Figure 4.4.1	Use case diagram	20
Figure 4.4.2	Activity diagram	22
Figure 4.4.3	Class diagram	26
Figure 4.4.4	Sequence diagram	29
Figure 4.4.5	Collaboration diagram	30
Figure 4.4.6	Machine diagram	33
Figure 4.4.7	Component diagram	35
Figure 4.4.8	Deployment diagram	36
Figure 4.4.9	DFD Level 0	42
Figure 4.4.10	DFD Level 1	43
Figure 4.4.11	DFD Level 2	44

LIST OF TABLES

Table No.	**Name of Table**	**Page No.**
Table 4.4.1	Values	25
Table 4.4.2	Tools	29
Table 4.4.3	Tools	32

ABSTRACT

The Blood Bank Management System (BBMS) is a browser-based system that stores, processes, retrieves, and analyses data related to administrative and inventory management inside a blood bank, such as donor registration, blood collection, storage, and distribution. This initiative aims to keep all information regarding blood donors and the various blood groups available in each blood bank up-to-date, as well as to aid blood banks in more successfully managing their operations.

The technology is intended to improve blood bank operations and ensure the availability of blood products for patient transfusions. The purpose is to provide transparency in this field, to simplify and eliminate corruption in the blood bank collection process, and to strengthen the blood bank administration system.

This project is intended for two groups of users: donors and recipients, as well as hospitals and clinics (doctors). A hospital user can upload blood samples to their blood bank, request blood, track the status of their request, update their information, and so on. The giver or recipient has the same option. Donor registration and management, inventory management, blood request and distribution management, and reporting are all components of the BBMS. The system is intended to be user-friendly, with a simple and intuitive interface that blood bank staff can quickly operate.

Modern software development tools and processes, such as web-based technologies, database management systems, and agile development methodologies, will be used in this project. Front-end and back-end technologies such as HTML, CSS, JavaScript, PHP, and MySQL are used to build the system.

The BBMS has the potential to greatly enhance blood bank management, resulting in faster response times, more accuracy, and lower costs. It is envisaged that the approach will help to improve healthcare services overall, notably in the areas of blood transfusion and emergency treatment.

TABLE OF CONTENTS

Page No.

Certificate i

Declaration ii

Acknowledgements iii

List of Abbreviations iv

List of Figures v

List of Tables vi

Abstract vii

CHAPTER I

INTRODUCTION

- 1.1 Introduction 1
- 1.2 Study of the problem 1
- 1.3 Scope 2
- 1.4 Objectives 2
- 1.5 Infrastructure 3

CHAPTER II

LITERATURE SURVEY 4

CHAPTER III

SYSTEM ANALYSIS

- 3.1 Objectives 5
- 3.2 Problem specification 5
- 3.3 Proposed system 6
- 3.4 Applications 7
- 3.5 Modules and their functionalities 7
- 3.6 Software and Hardware Requirements 9

CHAPTER IV

DESIGN

- 4.1 Architecture 10
- 4.2 Flow chart 11

4.3 Algorithms 11
4.4 UML diagram 12
4.5 ER diagram 36
4.6 DFD 40

CHAPTER V

IMPLEMENTATION

5.1 Partial code 45
5.2 Screen shots 55

CHAPTER VI

CONCLUSION AND FUTURE ENHANCEMENT 59

CHAPTER VII

REFERENCES 60

CHAPTER I

INTRODUCTION

1.1 Introduction

The issue of an integrated blood supply to all tables and stakeholders is the blood bank management system, which is very helpful to us in hospitals. The programmed Visual Studio links certain tables together to organize and make it easier to access the information. This project makes it easier for the hospital to manage everything manually. A summary of the present state of blood banking should be given in the project's introduction.

The importance of having a good system in place to control the donation process. Along with outlining the system's technical features and functions, it should also list the project's aims and objectives.

Smart Data Management: A vital component of any management system for a blood bank is efficient data management. The system must be capable of storing and retrieving information on blood donations, such as donor details, blood type, and test results, precisely and quickly. Each unit of blood should also be able to be tracked by the system from the time it is collected until it is used or runs out of time.

Strong data validation and error-checking methods should be included in the system to guarantee the quality and integrity of the data. Security safeguards must be in place to prevent unauthorized access to the data.

The system should be able to manage blood donation and unit data as well as produce reports and analytics using this information. This can be used to monitor blood supply trends and patterns, spot areas for donation process improvement, and help with blood inventory and distribution decisions.

1.2 Study of the problem

The management of the collection, testing, and distribution of blood and blood products to people in need falls under the purview of the blood bank management system. However, there are a number of difficulties that could occur in this procedure, affecting the system's effectiveness and efficiency. The lack of blood donations is one of the major issues that blood bank management systems must deal with. Many people are unable or reluctant to donate blood, even though it has the potential to save up to three lives. Particularly during times of crisis or excessive demand, this may result in shortages.

The requirement to guarantee the security and quality of the blood supply is a further issue. Strict regulations must be followed when collecting, storing, and transporting blood in order to avoid contamination or degradation. In order to do this, a reliable system must be in place for testing and monitoring the blood as well as for detecting and recalling any potentially dangerous units. The blood supply must also be efficiently tracked and distributed by blood bank management systems in order to guarantee that it is accessible when and where it is needed. Given that the recipient's blood type must be identified and that blood may need to be transported over significant distances, this can be a challenging task. Overall, a system that is effective and efficient for managing this crucial resource is required, as shown by the analysis of the issues blood bank management systems confront.

1.3 Scope

The set of features and functions that a blood bank management system is intended to enable is referred to as its scope. The following components should be present in a thorough management system for blood banks:

Blood donation management: is the process of gathering, testing, and storing donated blood, as well as keeping track of donor data and test outcomes.

Management of blood inventory: The system must be able to track each unit of blood's location and condition, as well as its origin, expiration date, and usage status.

Blood distribution management: The system should be able to coordinate the provision of blood to hospitals and other healthcare facilities by matching blood units to particular blood types.

Reporting and analytics: Based on information on blood donations and the blood supply, the system should be able to produce reports and analytics.

Security and data protection: The system must have safeguards in place to prevent unauthorized access to data and guarantee the accuracy of the data.

A blood bank management system can assist in ensuring that blood is accessible when and where it is required while also maintaining the supply's safety and quality by taking care of these issues.

1.4 Objectives

A blood bank management system's primary goal is to increase the effectiveness and efficiency of the blood donation and distribution processes. This can be done in a number of ways, including:

Increasing the quantity of blood donations: The system can contribute to increasing the overall blood supply by streamlining the donation process and making it simpler for people to donate blood.

Ensuring the blood supply's safety and quality: Strong processes should be in place in the system for testing and monitoring the blood as well as for recognizing and recalling any potentially dangerous units. The system should be able to efficiently match blood units to particular blood types and make it easier for blood to be distributed to hospitals and other medical facilities.

Providing data-driven insights: The system can assist decision-makers in identifying trends and patterns and in making well-informed decisions about blood inventory and distribution by producing reports and analytics based on data about blood donations and the blood supply.

A blood bank management system's overall goal is to help ensure that blood is accessible when and where it is required, all the while ensuring the supply's safety and quality.

1.5 Infrastructure

The hardware, software, and other resources necessary to support a blood bank management system project are referred to as the infrastructure. Computers, servers, storage units, and other system-running hardware may make up the hardware infrastructure. Any specialized tools required for blood collection, testing, and storage may also fall under this category.

The programmes and apps that are utilized to support the system are referred to as software infrastructure. In addition to specialized software for managing the blood donation process, tracking blood units, and producing reports and analytics, this may contain a database management system to store and retrieve data.

A blood bank management system project's infrastructure may also comprise personnel and educational materials in addition to technology and software. Additionally, it could comprise infrastructure and facilities for blood collection and storage, as well as logistical and transportation assets to make it easier to get blood to hospitals and other medical facilities. In general, the infrastructure of a blood bank management system project is crucial to the system's success since it establishes the system's capabilities and capacity to support the process of blood donation and distribution.

CHAPTER II

LITERATURE SURVEY

S. K. Bag and K. S. Choudhury's "Design and Development of a Blood Bank Management System" In order to increase the effectiveness and efficiency of blood bank operations, this research paper describes the design and development of a web-based blood bank management system.

R. Sharma and R. K. Gupta's "Blood bank management: a review" An overview of the current state of blood bank management systems is given in this article, along with information on their advantages, difficulties, and prospects for the future.

M. Praveen Kumar and K. Vijaya's "An Intelligent Blood Bank Management System Using IoT": In order to increase the effectiveness and dependability of blood bank operations, this paper describes an intelligent blood bank management system that makes use of Internet of Things (IoT) technologies.

By P. S. P. Raju and K. Sreenivasulu, "Blood Bank Management System Using Android App": The design and development of an Android-based blood bank management system are presented in this research paper. It enables blood donors to register and donate blood through a mobile app.

N. K. Singh and A. Gupta's "A Comprehensive Review on Blood Bank Management Systems": This article offers a thorough analysis of blood bank management systems, covering all of their features, benefits, and drawbacks.

S. Y. Ong, L. T. Teng, and C. K. Lim's "Development of a Web-based Blood Bank Management System": In order to increase the effectiveness and accuracy of blood bank operations, a web-based management system was developed, as described in this research paper.

N N Singh and R K Srivastava's "A Comparative Study of Blood Bank Management Systems": This essay compares and contrasts various blood bank management systems, highlighting their advantages and disadvantages. By P. Patil and M. A. Ansari, "Blood Bank Information Management System: A Review": The main characteristics and advantages of blood bank information management systems are reviewed in this article.

CHAPTER III

SYSTEM ANALYSIS

3.1 Objectives

Finding the requirements: The next goal is to find the blood bank management system's requirements. Understanding the demands and goals of the various parties, including donors, recipients, the medical community, and the support staff, is necessary.

Specifying the scope: After the project's requirements have been determined, the project's scope must be specified. This entails determining the functions, features, and procedures that the management system for blood banks will support.

Creating use cases: Use cases are hypothetical situations that illustrate how system users will interact with it. Creating use cases will aid in determining the features and functionalities that the system must support.

Making a data model: A data model is a graphic representation of the data and the connections between it and other elements of the system. The data that the system needs to collect and store can be identified with the aid of a data model.

Finding restrictions: The following goal would be to find any restrictions that might affect how the system is developed. These might be imposed by budgetary, legislative, or technical restrictions. The system architecture is designed, and it outlines the elements, modules, and interfaces that go into creating the system. Making sure the system is scalable, adaptable, and maintainable will be made easier by designing the system architecture. The testing strategy specifies how the system will be tested to see if it complies with the requirements. This entails deciding on the testing methodologies, tools, and techniques that will be applied.

Making a project plan: The final goal is to make a project plan that details the tasks, deadlines, and materials required to develop and implement the blood bank management system. To identify and reduce any potential risks, the project plan should also include a risk management strategy.

3.2 Problem specifications

A blood bank is a facility where donated blood is processed and stored in preparation for transfusion to patients in need. For an adequate blood supply to be available for both emergency and routine transfusions, blood bank management is essential. However, managing blood banks manually can be time-consuming, prone to mistakes, and ineffective. Designing a blood bank management system that automates and streamlines the management of blood banks is the goal of this mini-project. The system's design will take into account the following needs:

Donor Management: The system should allow donors to register themselves and provide information about themselves, such as their blood type, age, gender, and contact details.

Blood Donations: The system should make it possible to record blood donations, including the donor's information, the date of the donation, and the type of blood collected. The system should also permit the storage of blood and the monitoring of blood unit expiration dates.

Blood Request and Transfusion: The system should make it possible to record blood requests from hospitals, clinics, or other healthcare facilities. These requests should include patient information, the blood group that the patient needs, and the amount of blood that is required. Additionally, the system should permit the creation of reports on blood usage and the tracking of blood transfusions for patients.

Inventory Management: The system should make it possible to keep track of the inventory of blood units, including the quantity of each blood type that is currently in stock, the number of units that have expired, and the current stock levels.

User Management: The system ought to permit the creation and administration of user accounts with various levels of access, such as those for administrators, personnel at blood banks, and medical professionals.

Security and Privacy: The system must protect sensitive donor and patient information and adhere to all applicable data protection laws. A web-based application called the Blood Bank Management System will be created using HTML, CSS, JavaScript, and a server-side programming language like PHP or Python. For storing and retrieving data, the system will use a database management system, such as MySQL.

3.3 Proposed problem

To debug the current system, remove processes that duplicate data, and correct the navigational order. To build strong password mechanisms, information about audits at various levels is provided, along with a reflection of the current work status based on the organization, auditor, or date.

a) Goals:

- To streamline the blood donation and reception process.
- To enhance the current system.
- To create a system that is scalable.

- To be very accessible.

b) Scope:

Make sure the programme is straightforward and simple to use and that it covers all the functions of a manual blood bank. It should also include all blood banks, at the very least, within a city.

3.4 Applications

An effective tool for managing the donation, collection, storage, and distribution of blood is a blood bank management system. Here are some potential uses for a mini-project on a blood bank management system:

Donor registration: The system can offer a platform for donors to register themselves, providing their contact information and basic information like name, age, and gender. The system can also gather information about past donations and blood types.

Management of blood inventory: The system can keep track of the blood units that are on hand at the blood bank, as well as their blood type and expiration dates. Additionally, it can monitor the demand for various blood types and order the distribution of blood units accordingly.

Scheduling blood donation events: Depending on the donors' availability and the needs of the blood bank, the system can help schedule blood donation events and appointments with donors.

Monitoring the status of blood donations: The system can monitor the progress of a blood unit from the time it is donated until it is used. This can make sure that the blood is transported and stored in the right way to preserve its quality.

Rewards and recognition for donors: The system can encourage blood donation by rewarding and recognizing loyal and frequent donors. This can include diplomas, badges, and other kinds of acknowledgments.

Analytics and reporting: The system is capable of producing reports on a variety of blood bank operations, including inventory levels, donor statistics, and distribution patterns. This can help pinpoint problem areas and streamline the blood bank's operations.

Integration with mobile apps: The system can work with a mobile app to let donors register, make appointments, and get alerts and reminders on their phones. As a result, donation rates may rise, and donors may find it more convenient and accessible.

3.5 Modules and their functionalities

3.5.1 Project Phases and Modules

3.5.1.1 Software Development Life Cycle (SDLC)

The term "software development process models" refers to a variety of defined and designed software development approaches that are used or employed during the development process of software (e.g., waterfall model, incremental model, V-model, iterative model, etc.). Each process model adheres to a specific life cycle to guarantee the success of the software development process. The phases of the software cycle and the order in which they are carried out are described by software life cycle models. Each stage of the life cycle results in deliverables needed for the following stage. Design is translated from requirements. The development phase refers to the process of producing code in accordance with the design. Following coding and development, testing confirms that the deliverable of the implementation phase complies with the specifications.

3.5.1.2 Life Cycle Phases

Every software development life cycle model has the following six phases:

➢ Requirement gathering and analysis

➢ Design

➢ Implementation or coding

➢ Testing

➢ Deployment

➢ Maintenance

Requirement gathering and analysis: In this stage, business requirements are gathered. The project managers and stakeholders are primarily focused on this phase. To determine the needs, such as who will use the system, meetings are held with managers, stakeholders, and users. How will they employ the apparatus? What information needs to be added to the system? What information should the system output? These are general inquiries that are addressed in a After gathering requirements, these requirements are examined for validity, and the likelihood of incorporating them into the system under development is also investigated. Finally, a requirement specification document is produced as a guide for the model's subsequent phase.

Design: Based on the requirement specifications that were examined in the first phase, the system and software design are created in this phase. System design aids in defining the overall system architecture as well as the hardware and system requirements. The system design requirements are used as input for the model's subsequent stages.

Implementation or coding: After receiving the system design documents, the work is divided into modules and units, and actual coding is then begun. The developer's primary focus during this phase is on producing code. This is the stage of the software development life cycle that lasts the longest.

Testing: The code is tested against the requirements after it has been developed to ensure that it is actually addressing the needs gathered during the requirements phase. Unit testing, integration testing, system testing, and acceptance testing are carried out during this phase.

Deployment: The product is given to the customer for use after successful testing.

Maintenance: When the developed system is used by the customers, actual issues arise and require periodic resolution. Maintenance is the process of taking good care of a developed product.

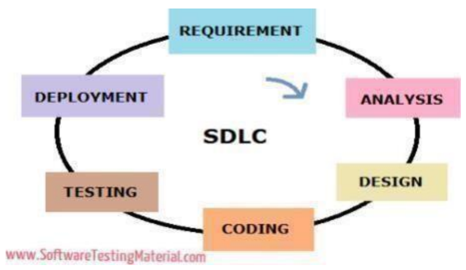


Fig No. 3.1 Software Development Life Cycle (SDLC)

3.6 Software and Hardware Requirements

3.6.1 Hardware Requirements

- RAM: 4 GB
- Hard Disk: 10 GB
- Processor: Intel core i3

3.6.2 Software Requirements

- Operating System: Windows 10
- Front Design: Visual Studio 2008
- Front-End Language: HTML, CSS
- Back-End Language: PHP, MySQL

CHAPTER IV

DESIGN

4.1 Architecture

A blood bank management system's architecture typically consists of a number of interconnected parts that cooperate to accomplish the system's goals. For your small project, you can use the following basic architecture:

User interface: This part offers a graphical user interface (GUI) so that users can communicate with the system. Users should be able to quickly access the information they require by using a GUI that is simple to use and intuitive.

The database is where all the data on blood donors, recipients, and inventory is kept. The database should be created in a way that allows for efficient data retrieval and storage while handling large volumes of data.

The management of the blood inventory in the blood bank is handled by this module. It ought to be able to keep track of the various blood types that are available and alert the staff when the supply is getting low.

Module for managing donors: This module controls data pertaining to blood donors. Information about donors, including their identifying characteristics, medical background, and blood type, should be able to be stored and retrieved.

Module for managing recipients: This module handles data pertaining to recipients of blood. Receivers' personal information, medical history, and blood type should be able to be stored and retrieved by it.

Module for scheduling blood donations: This module enables donors to book appointments for blood donations. It should be able to coordinate the appointment schedule for blood donations and notify donors when their appointments are approaching.

Module for blood transfusion: This module oversees the transfer of blood from the blood bank to the recipient. It ought to be able to keep track of the blood units given to the recipient and modify the inventory as necessary.

Reporting module: This module creates reports that give important information about how the blood bank functions. Reports on blood inventory levels, donor and recipient information, blood donation schedules, and blood transfusion records ought to be produced by it.

Some of the essential elements of a management system for blood banks include those mentioned above. The architecture can be modified to meet your unique needs, and, if necessary, extra features or modules can be added.

4.2 Flow chart

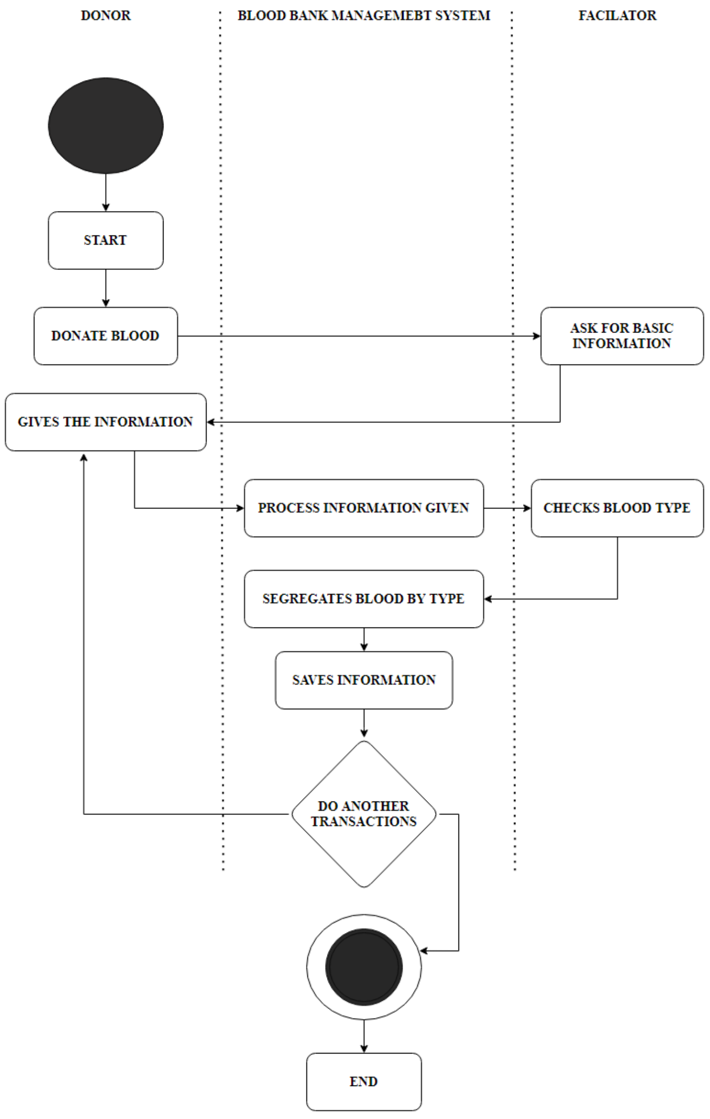


Fig No. 4.2 Flow Chart

4.3 Algorithms

A software programme called a blood bank management system aids in the management of the various procedures necessary to operate a blood bank. The management of the blood supply, donor data, and blood donation procedures can all be efficiently and accurately managed thanks to algorithms, which can play a crucial role in the design of such a system. The following algorithms can be used in the development of a mini-project for a blood bank management system:

1. The blood inventory management algorithm can be used to keep track of the various blood types that are on hand at the blood bank. Additionally, it can assist in preserving the minimum stock level for each blood type and in sending out alerts when the stock drops below the

required level. The algorithm can also be used to distribute blood to various clinics and hospitals in accordance with their needs.

2. Donor Management Algorithm: This algorithm is capable of managing donor data, such as their name, blood type, and history of donations. Additionally, it can be used to keep track of donors' eligibility, make appointment times for donations, and remind donors to make them.

3. Blood Donation Process Algorithm: Using this algorithm, you can control every step of the blood donation procedure, from signing up to collecting and testing the blood. It can be used to monitor each blood donation's progress, make sure that all required tests are carried out, and update the donor's information once the donation is finished.

4. Blood Transfusion Algorithm: This algorithm is useful for controlling the blood transfusion procedure, including the choice of compatible blood types, tracking of transfused blood units, and keeping track of patient outcomes.

5. Data Analysis Algorithm: Reports on blood usage, donor demographics, and other important performance indicators can be produced using this algorithm to analyze blood bank data. Additionally, it can be used to spot trends and patterns in the data, which can be useful for determining the best ways to manage the blood supply and find donors.

These are merely a few illustrations of algorithms that might be applied in the creation of a miniature blood bank management system. Additional algorithms might be required to ensure the effective and efficient management of blood bank processes, depending on the specific project requirements.

4.4 UML diagram

4.4.1 Unified Modeling Language (UML):

1. Model

- Real life is simplified into a model.
- The schematics for a system are provided by a model.
- A model could be behavioral, emphasizing the dynamics of the system, or structural, emphasizing the way the system is organized.
- We create models in order to better comprehend the system we are creating.
- We create models of complex systems because we are unable to fully understand them.

Four objectives are attained through modelling.

- With the aid of models, we can see a system as it is or as we want it to be.
- We can specify a system's structure or behaviour using models.
- Models provide us with a framework that directs our construction of a system.
- Models serve as a record of our choices.

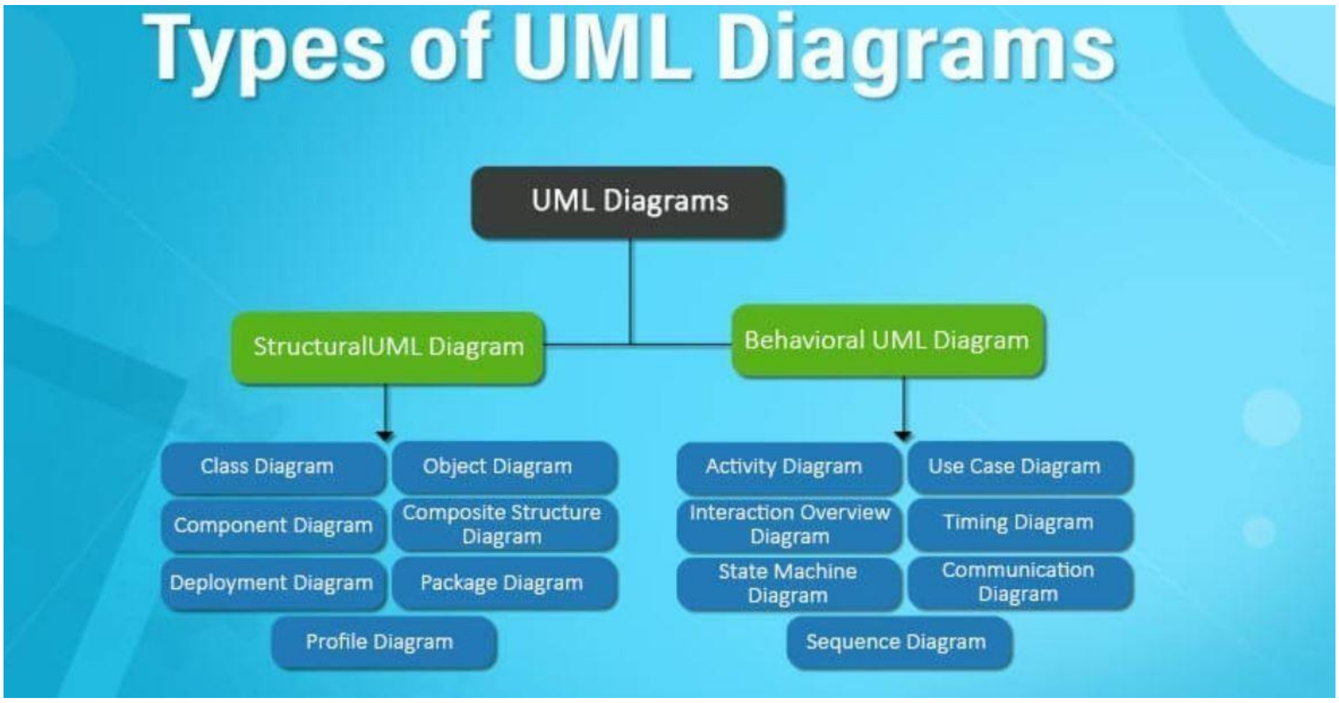
2. Principles of Modeling

 The models that are created have a significant impact on how a problem is approached and how a solution is developed.

 Different precision levels can be expressed for every model.

 The most accurate models are grounded in reality.

 No one model works well enough. The best way to tackle any nontrivial system is with a condensed set of essentially independent models.



UML is a graphical notation that is used to visualize, define, build, and document software-intensive artefacts. Enterprise information systems, distributed web-based applications, and even hard, real-time embedded systems can all be modelled using UML. In order for UML to work effectively, the language must first be conceptually modelled.

Diagrams come in two varieties: Those are

1. Static Diagrams

   a) Use case diagrams

   b) Class diagrams

   c) Object diagrams

   d) Component diagrams

e) Deployment diagrams

2. Dynamic diagrams

a) Interaction diagrams

i) Sequence diagrams

ii) Collaboration diagrams

b) State machine diagrams

c) Activity diagrams

Applications of UML:

Software-intensive systems are the primary target audience for UML. It has been utilised successfully in areas like

1. Enterprise Information Systems
2. Banking and Financial Services
3. Telecommunications
4. Transportation
5. Defense and Aerospace
6. Retail
7. Medical Electronics
8. Scientific
9. Distributed Web-based Services

Basic building blocks of UML:

The components of UML can be divided into two categories:

1. Things
2. Relationships and
3. Diagrams

Things: - The most crucial UML building block is a thing. Things could

a) Structural

b) Behavioral

c) Grouping

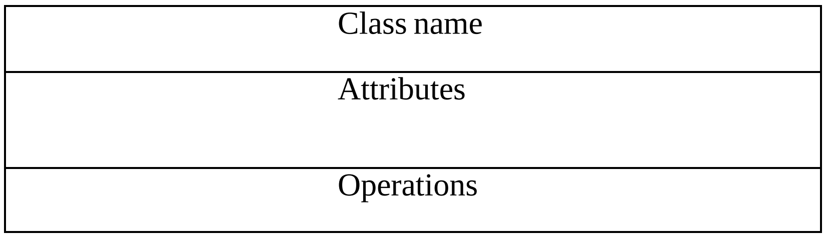
d) Annotational

a) <u>Structural Things</u>: These things

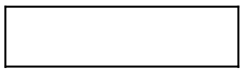
define the model's static portion. They stand for both concrete and abstract elements.

Here are the structural things:

1. <u>Class</u>: - It describes a group of objects that are similar in terms of their semantics, relationships, operations, and attributes.

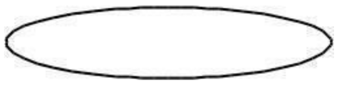


2. <u>Object</u>: An object is a group of operations that define a class's or component's service.

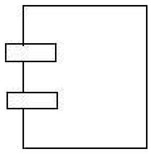


3. <u>Collaboration</u>: - It describes how different elements interact.

4. <u>Use case</u>: - They are used to distinguish various software project use-case components. It is applied to the operation's modelling.

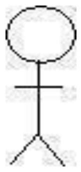


5. <u>Component</u>: - It is a tangible, replaceable component that attests to and offers the realization of a number of interfaces.



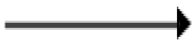
6. Node: - A physical resource that serves as a computational resource and is present at runtime.

7. Actor: - The external party that interacts with a system. Typically, a person playing a role on an external device.

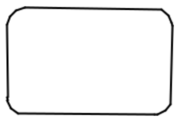


b) Behavioral Things: These are made up of the UML model's dynamic components. The following are examples of behavior:

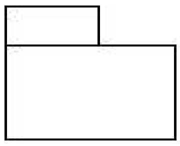
1. Interaction: - It is described as a behavior made up of a series of messages exchanged between components in order to complete a particular task message.



2. State machine: - When an object's life cycle states are known, it is helpful. It outlines the progression of states that an object experiences in response to events.

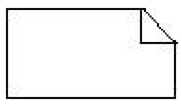


c) Grouping Things: They can be explained as a way to organize the components of a UML model. Only one grouping option, known as a package, is available. For assembling structural and behavioral items, use a package.



d) Annotational Things: - They can be described as a tool for recording comments, descriptions, and observations about UML model elements. Note: This is the sole notational item that is available.

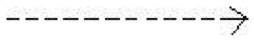
A UML element's comments, constraints, and other information are rendered using Note.



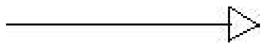
Relationships: -

The relationship is yet another crucial UML building block. They demonstrate the relationships between elements and how those relationships describe an application's functionality. Five different kinds of relationships exist.

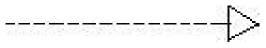
1. Dependency: This is a relationship between two things where a change in one also affects the other.



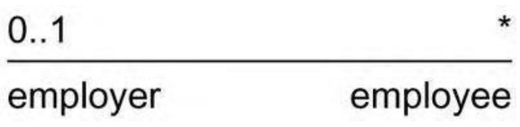
2. Generalization: It can be described as a connection between specialized and generalized elements. In essence, it describes how an object's inheritance relationship works. There is a hierarchy in place.



3. Realization: It can be characterized as a connection between two elements. One component outlines a duty that is not carried out, while the second one does so. This connection is present when there are interfaces.



4. Association: It is a system of connections between the various parts of a UML model.



UML Diagrams: -

1. Use case diagram

2. Activity diagram

3. Class diagram
4. Interaction diagrams
    i) Sequence diagram
    ii) Collaboration diagram
5. State machine diagram
6. Component diagram
7. Deployment diagram

The Unified Modelling Language (UML) is a modelling and development environment that is used by Rational Software Architect (RSA) to design the architecture of web services and C++ and Java EE (JEE) applications. The Eclipse open-source software framework serves as the foundation for Rational Software Architect, which has features geared towards architectural code analysis, C++, and model-driven development (MDD) with the UML for building applications and web services.

1. USE CASE DIAGRAM

Problem Definition To create a blood bank management system, use a case diagram.

Problem Description

A use case diagram describes a collection of sequences, where each sequence denotes a connection to external entities. The interaction between an actor and a system is a use case. Three different kinds of relationships exist:

1. Association
2. Dependency
3. Generalization

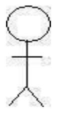
- Actors – “things” outside the system.
- Use cases – system parameters specifying what the system is supposed to do.

Use case diagrams can be used during analysis to describe the needs of the system and understand its intended functionality. Use-case diagrams can be used to specify how the system will behave when it is put into use during the design phase.

Actor: - A system user is represented by an actor. They provide a clearer picture of what the system should do and aid in defining the system requirements. An actor is anyone or anything who

- Uses the system or interacts with it.
- Informs the system with input and receives data from it.

- Has no influence over the use cases and is not a part of the system.



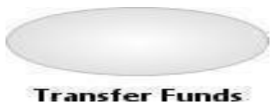
Customer (actor)

Actors are found by looking at:

- Who directly uses the system?
- Who is in charge of system maintenance?
- The system's external hardware components
- Additional systems that must communicate with the system

Use case: - From the perspective of users (actors), a use case is a particular way to use a system. A use case is defined as:

- A pattern of behavior the system demonstrates.
- A series of connected transactions carried out by a system and an actor
- Giving the actor something of value



Note: use cases frequently begin with a "verb".

Use cases offer a way to, among other things, capture system requirements.

- Interact with domain specialists and end users.
- Run a system test

A textual description is provided for each graphical representation. A use case specification contains a description of each use case. The use case specification includes:

The use case's starting method and time are specified in the precondition.

The use case's main flow lists the series of operations it performs.

The use-case execution exceptions that could occur are listed in Alternate Flow. The post-condition displays the outcome following the successful completion of the use case.

Blood Bank Management System

UML Diagrams

Use Case Diagram

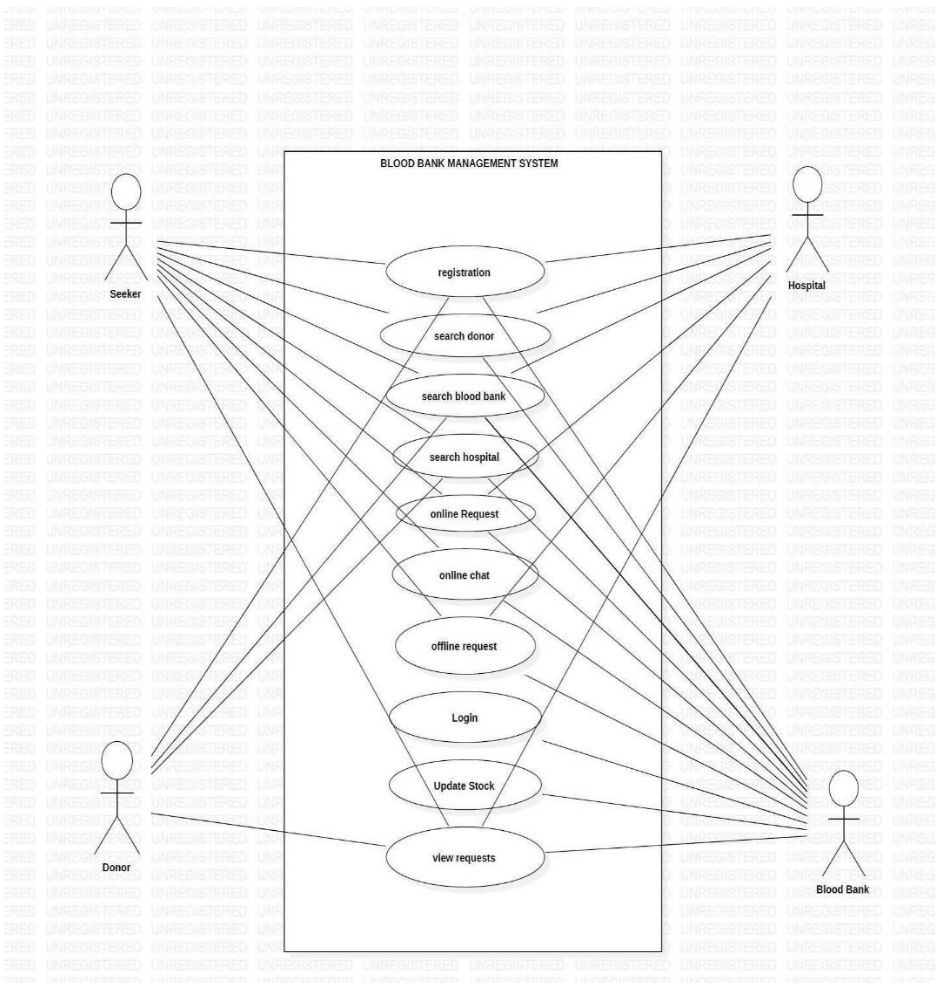


Fig No. 4.4.1: Use case diagram for Blood Bank Management System

2. USE CASE DIAGRAM

Problem Definition

To create a blood bank management system activity diagram.

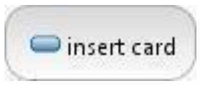
Problem Description

A specific type of state diagram is an activity diagram. Similar to a flow chart, an activity diagram depicts the movement of a control from one activity to another. The system's dynamic elements are modelled using an activity diagram.

1. Activity states and action states are included in the activity diagram.

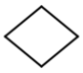
2.Transition object

Action state: - This type of computation represents the execution of an action and is atomic and executable.

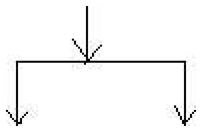


Activity state: - They can break down. In other words, other activities serve to represent their own diagrams.

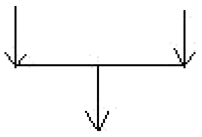
Branching: - One incoming transition and two or more outgoing transitions are present in branching.



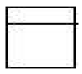
Forking: - A single flow of control is divided into several flows of control during this process. A fork typically has multiple outgoing flows of control but only one incoming flow of control.

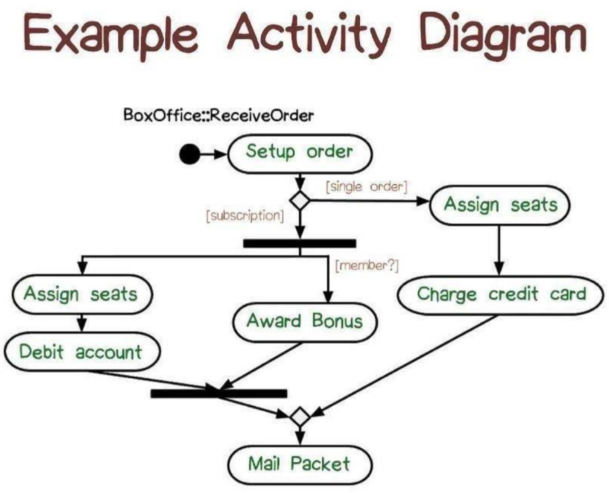


Joining: - The exact opposite of forking is this. It typically has one outgoing flow of control but numerous incoming flows.



Swim-lanes: - In order to group related activities, they stand in for the columns in the activity diagram. These take the shape of a partitioned region. Swim-lanes can represent organizational units or roles within a business model, which makes them useful when modelling a business work flow. Because they make it possible to identify who is playing a particular role, swim lanes are very similar to objects in this regard.





Pseudo code

1. Right-click the model once.
2. Pick Activity Diagram under Add Diagram.

Another crucial UML diagram for describing the system's dynamic elements is the activity diagram.

An activity diagram is essentially a flowchart that shows how one activity leads to another. The action can be referred to as a system operation.

One operation leads to the next in the control flow. This flow may be parallel, concurrent, or branched. Activity diagrams use various elements, such as fork, join, etc., to deal with all types of flow control.

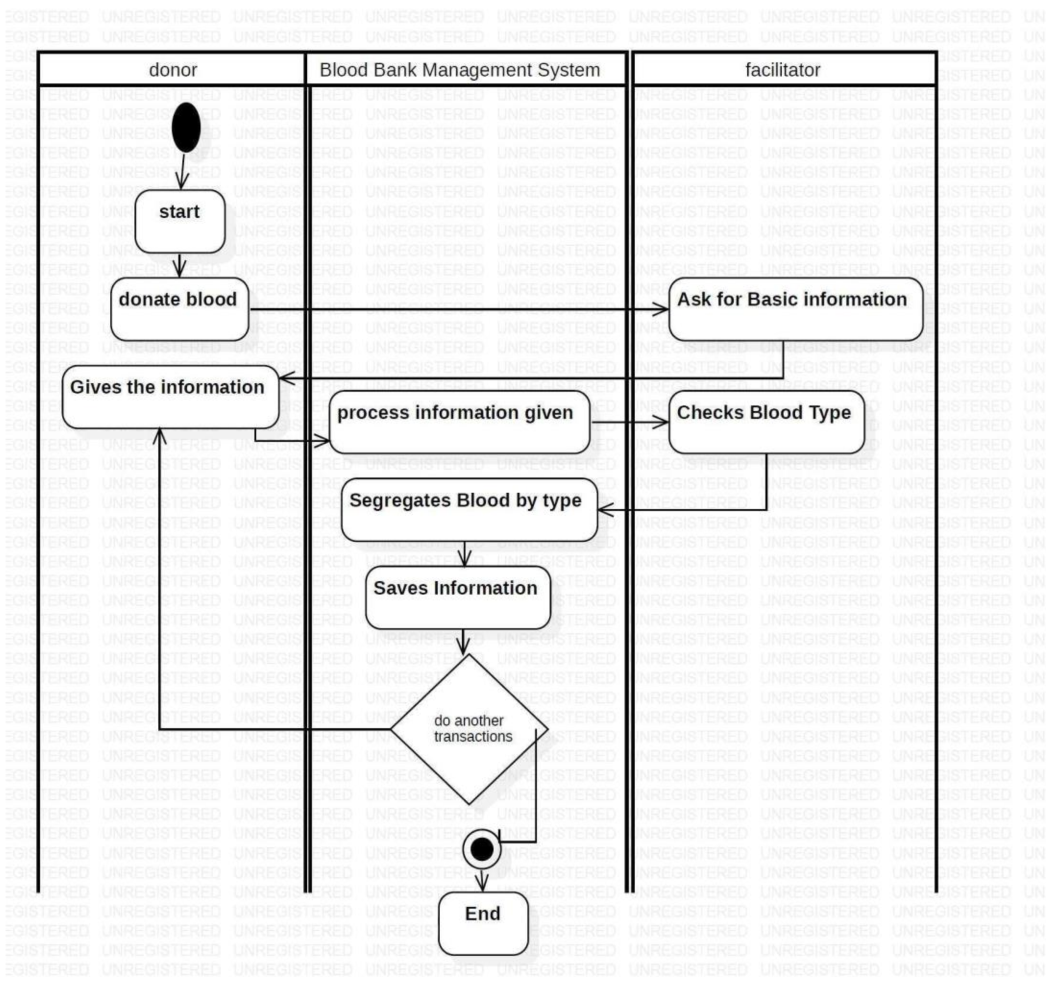


Fig No. 4.4.2: Activity diagram

3. CLASS DIAGRAM

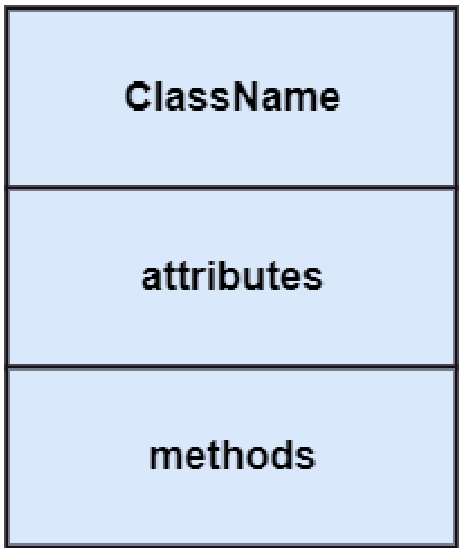
Problem Definition

The Blood Bank Management System class diagram.

Problem Description

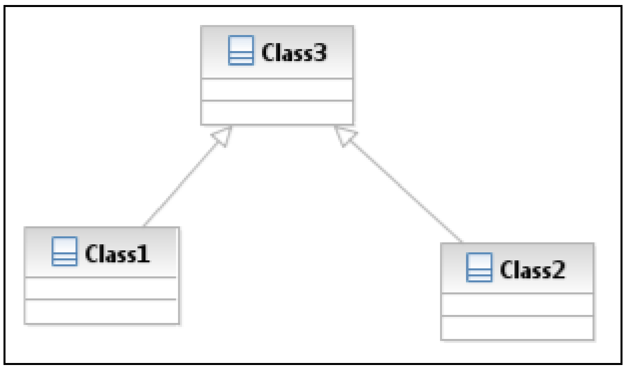
Icons in a class diagram show how classes, interfaces, and their connections are related.

Class: - A class is a group of objects that have the same attributes, operations, and semantics, as well as a common structure and behavior. A class is a representation of things in the real world. These things are instances of the class and are referred to as objects when they are real-world objects. A 3-part box serves as a class icon.



Generalization relationship for classes: It demonstrates how certain super classes' defined structures or behaviors are shared by their subclasses. To demonstrate that there "is a" relationship, use a general relationship.

Super class



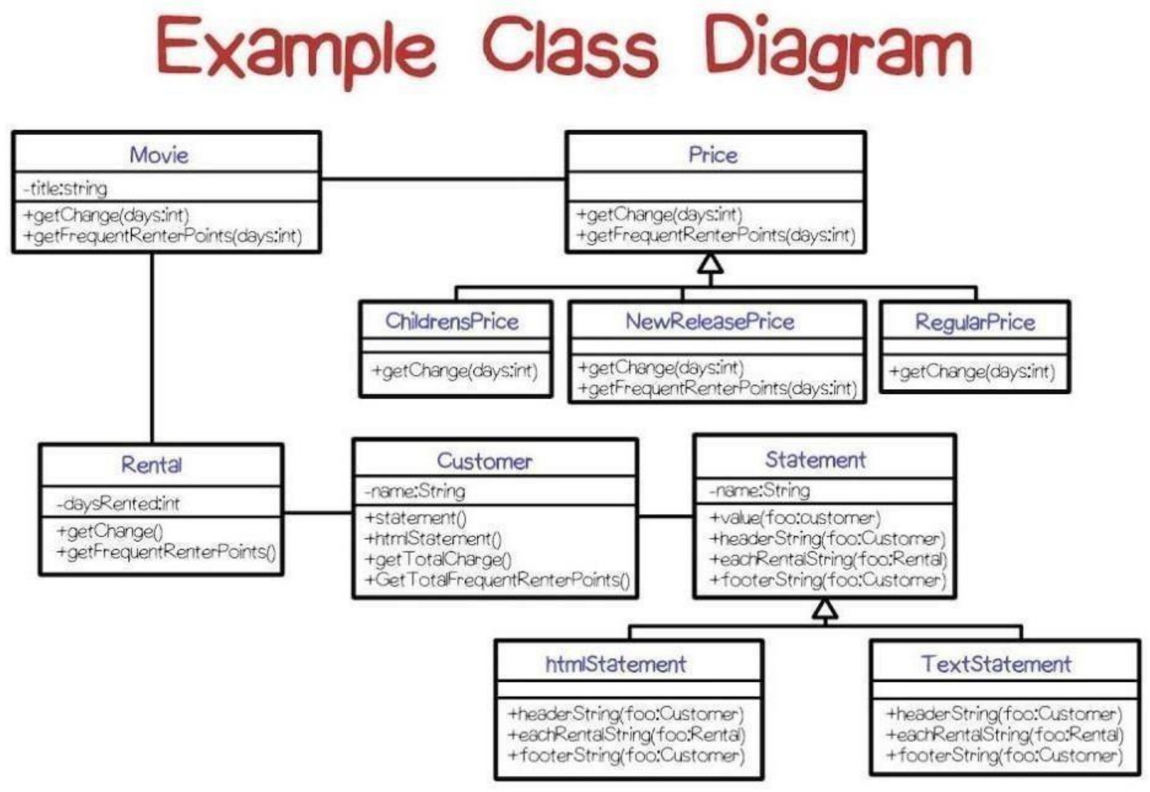
Sub class

Dependency Relationship: A relationship known as a dependency exists between two model elements in which a change in one of the elements will have an impact on the other. Dependency relationships in class diagrams typically show that client operations call supplier operations.

Cardinality Adornment: - The term "cardinality" refers to the number of instances of one class that can be joined to a single instance of another class. You can specify the maximum number of instances for a class by applying a cardinality adornment to it. When you mention a relationship, you're talking about the number of connections that are permitted between instances of one class and those of another.

Interface: - A class or component's externally visible operations are specified by an interface, which lacks an implementation of its own. Only a small portion of a class's or component's behavior is specified by an interface.

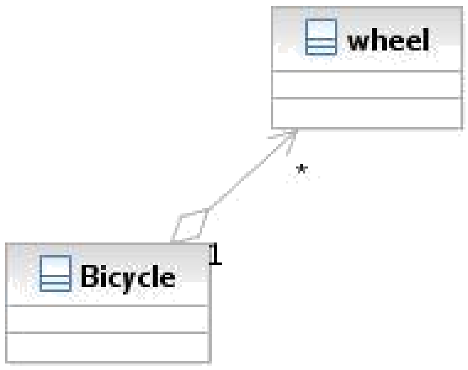
Association relationship: Relationship between associations: An association offers a communication channel. Use cases, actors, classes, and interfaces are all possible channels of communication. An association exists when two classes are typically viewed independently of one another.



Example Class Diagram
Movie
-title:string
+getChange(days:int)
+getFrequentRenterPoints(days:int)
Price
+getChange(days:int)
+getFrequentRenterPoints(days:int)
ChildrensPrice
+getChange(days:int)
NewReleasePrice
+getChange(days:int)
+getFrequentRenterPoints(days:int)
RegularPrice
+getChange(days:int)
Rental
-daysRented:int
+getChange()
+getFrequentRenterPoints()
Customer
-name:String
+statement()
+htmlStatement()
+getTotalCharge()
+GetTotalFrequentRenterPoints()
Statement
-name:String
+value(foo:customer)
+headerString(foo:Customer)
+eachRentalString(foo:Rental)
+footerString(foo:Customer)
htmlStatement
+headerString(foo:Customer)
+eachRentalString(foo:Rental)
+footerString(foo:Customer)
TextStatement
+headerString(foo:Customer)
+eachRentalString(foo:Rental)
+footerString(foo:Customer)

Valid Values:	**Value Description**
0..0	Zero
0..1	zero or one
0.. n	zero or more
1..1	One
1.. n	one or more
N	unlimited number

Table No. 4.4.1 Values



The diamond end represents the class which is whole.

Pseudo code

1. Right click on the model
2. Select Add Diagram – Class diagram

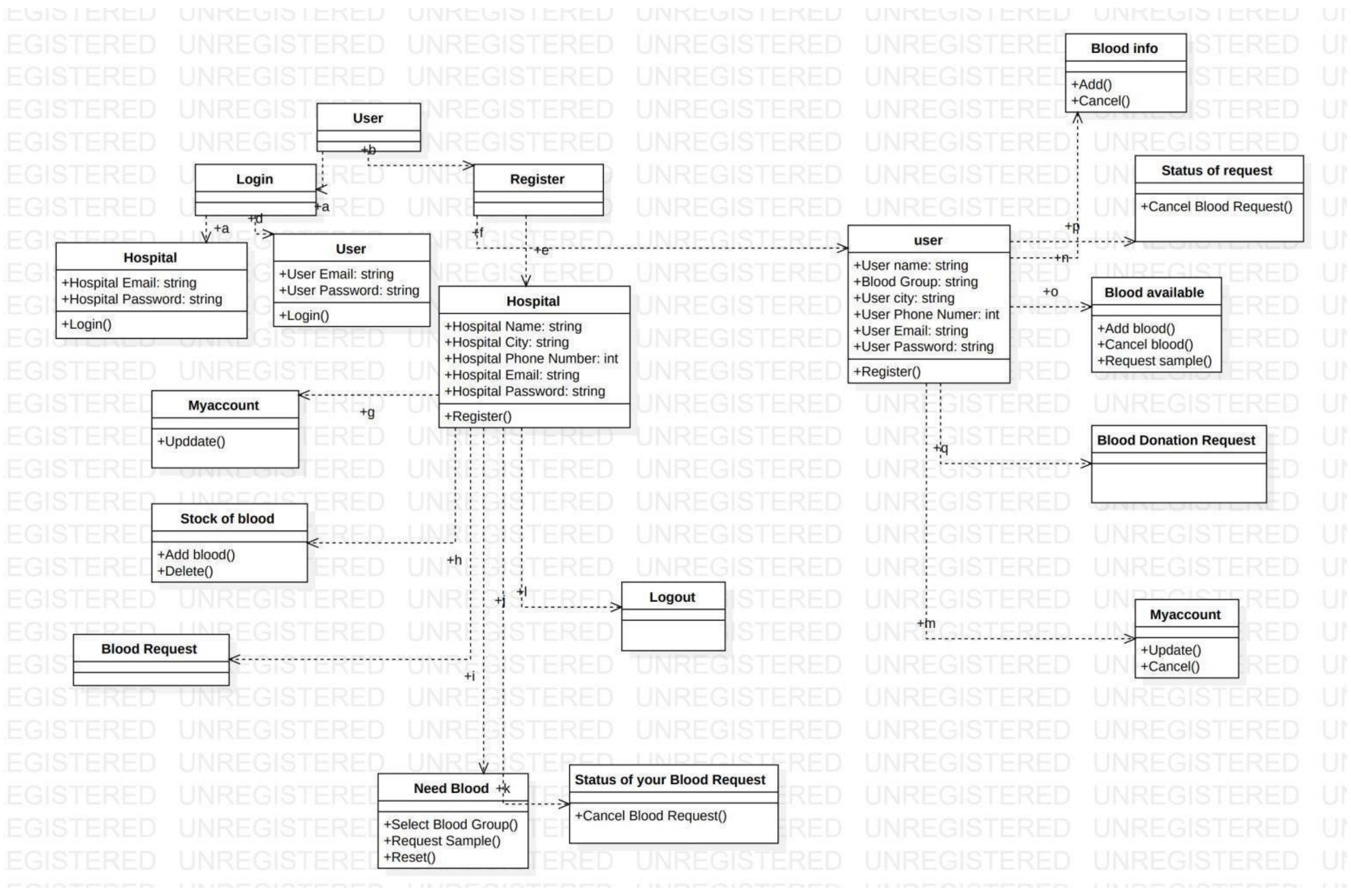


Fig No. 4.4.3 Class diagram for Blood Bank Management System

4. INTERACTION DIAGRAM

Problem Definition: Drawing interaction diagrams for the blood bank management system is the problem at hand.

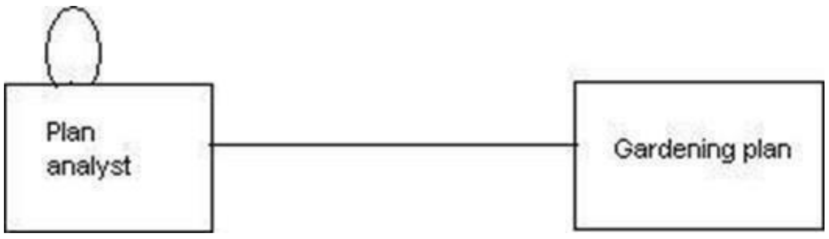
Problem description: An interaction is a significant series of interactions between objects, which is the problem.

Two categories of interaction diagrams exist:

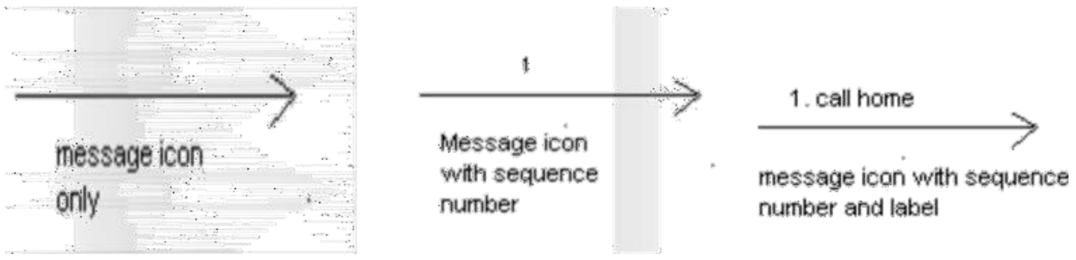
i) Sequence Diagrams: A sequence diagram is a graphical representation of a scenario that depicts object interaction in a time-based sequence, including what occurs first and what follows.

Sequence diagrams define object roles and help determine class responsibilities and interfaces by supplying crucial information. A sequence diagram has two dimensions: time is represented vertically, and different objects are represented horizontally.

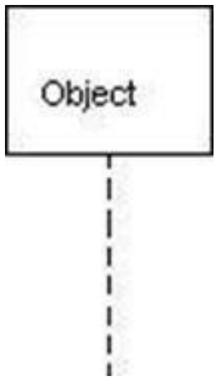
Link: Objects communicate with one another through links to other objects. Similar to how an object is an instance of a class, a link is an instance of an association. Only when there is a connection between the corresponding classes of two objects, including class utilities, should a link between them exist.



Message icons: A message icon shows how objects communicate and anticipate a subsequent action. The message icon is a solid, horizontal arrow joining two lifelines. In a sequence diagram, a message icon denotes just one message.


message icon only
1
Message icon with sequence number
1. call home
message icon with sequence number and label

Lifeline: Each object on the sequence diagram has a dashed vertical line called a "lifeline" that indicates its location at a specific moment in time. The lifeline also serves as a starting and ending point for messages and a location for the center of control.



Message or event: A message is communication between two objects that causes an event to occur. In collaboration and sequence diagrams, a message is represented by a message icon, which typically denotes its synchronization. Supported types of synchronization.

➔ Synchronous Call

➔ Asynchronous Call

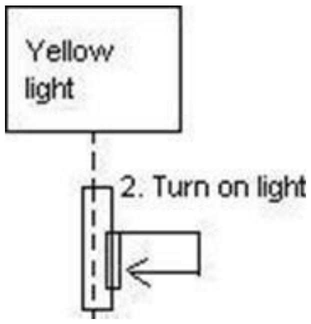
➔ Asynchronous Signal

➔ Create

➔ Delete

➔ Reply

Message to self: It is a device that relays messages from one object to another. The message's sender and recipient are one and the same.



Tools:

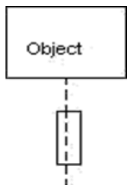
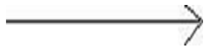
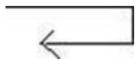
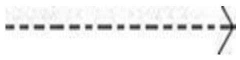
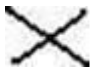
1.	Object	
2.	Object message	
3.	Message to self	
4.	Return message	
5.	Destruction marker	

Table No. 4.4.2 Tools

Pseudo Code:

➔ Right click on the model

➔ Select add diagram from the dropdown menu, click on add Diagram

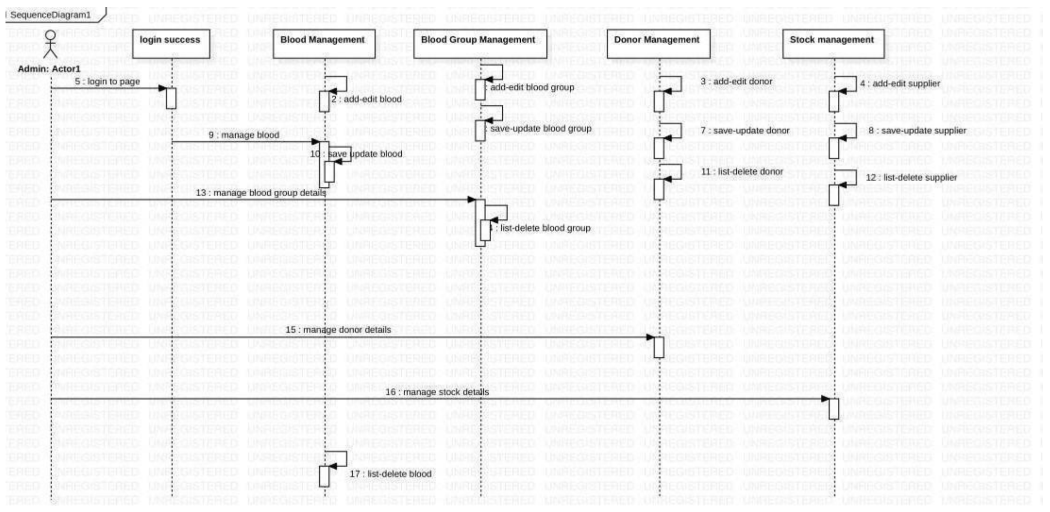


Fig No. 4.4.4 Sequence diagram for Blood Bank Management System

ii) Collaboration Diagrams: A collaboration diagram is an interaction diagram that displays the sequence in which messages are sent in order to carry out a transaction or operation. Collaboration diagrams display the objects, links, and messages associated with them. They may also include utility class instances and simple class instances. The interactions or structural relationships that exist between objects and object-like entities in the current model are shown in each collaboration diagram. Icons for objects are present in collaboration diagrams. Since they both depict how various objects interact with one another, sequence and collaboration diagrams are semantically equivalent. We are able to create additional diagrams from one diagram. Right-click the sequence diagram and choose Add Diagram—Communication

Diagram to create a collaboration diagram from it. A collaboration diagram can also be used to create a sequence diagram.

Pseudo Code:

➔ Right-click on the model. From the drop-down menu

➔ Choose "Add Diagram, and then click "Add Diagram."

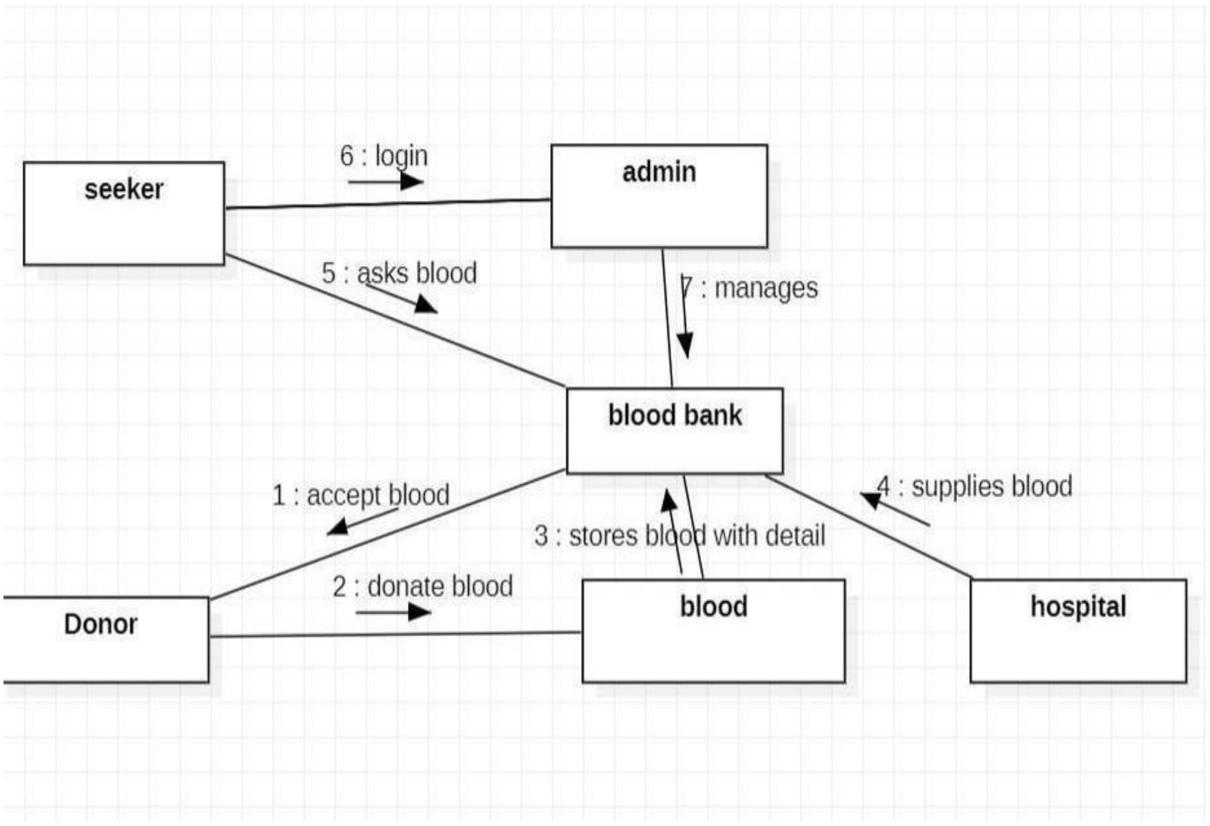


Fig No. 4.4.5 Collaboration Diagram for Blood Bank Management System

4. STATE MACHINE DIAGRAM

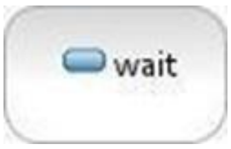
Problem Definition: to create a state machine diagram for the blood bank management system, which defines the problem.

Problem Description: state Machine diagrams can represent the dynamic behaviour of any type of object, including discrete classes. They display the progression of states through which an object experiences event.

Both the events that bring about a change in state and the actions that follow it A state Machine diagrams are frequently used to represent the distinct phases of an object's life. One start state and several end states are typical components of a state machine diagram.

State: A state is a condition or circumstance that an object experiences during its lifetime while it fulfils a requirement or waits for an event. Each state is an accumulation of all of its previous behaviors.

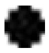
State machines are capable of sharing states. Transitions are private.



Naming: Within the state, each enclosing class's name must be distinct.

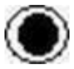
Actions: State actions can happen in one of four ways: upon entry, upon exit, or upon an event.

Start state: On a state machine diagram, a start state (also known as an "initial state") explicitly depicts the start of the state machine's execution, while an activity diagram's start state depicts the start of the workflow. Normal practise is to place one outgoing transition from the start state. However, if at least one of the transitions is marked with a condition, then multiple transitions may be placed in the start state. There can be no incoming transitions. A small, filled circle with the word "begin process" may be the start state icon.



Begin process

End state: On an activity or state machine diagram, an end state denotes the ultimate or terminal state. Only in an end state can a transition take place. A filled circle inside a slightly larger, empty circle that might also contain the name "end process" is the symbol for the end state.



End process

State transition: A state transition denotes that when a specific event occurs or when specific conditions are met, an action in the source state will carry out certain specified actions and enter the destination state. A relationship between two states, two activities, or between an activity and a state is referred to as a state transition. A line with an arrowhead pointing in the direction of the target state or activity serves as the state transition icon. As long as each transition is distinct, we can demonstrate one or more state transitions. There must be conditions on the event if transitions from the same state are to have different outcomes.

Naming: Each state transition should be given the name of at least one event that brought it about. Since the same event can result in a transition to numerous different states or activities, we do not need to use specific labels for the state transitions.

Tools:

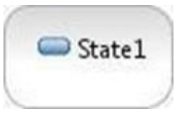
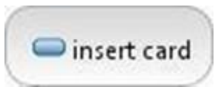
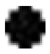
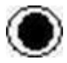
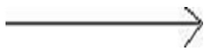
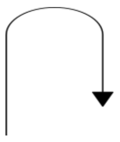
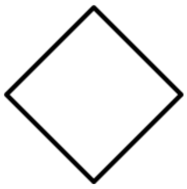
1.	State	
2.	Activity	
3.	Start state	
4.	End state	
5.	State transition	
6.	Transition to self	
7.	Horizontal synchronization	
8.	Vertical synchronization	
9.	Decision	

Table No. 4.4.3 Tools

Pseudo code:

➔ Right click on the model

➔ Select add diagram from the dropdown menu, click on add State Chart Diagram

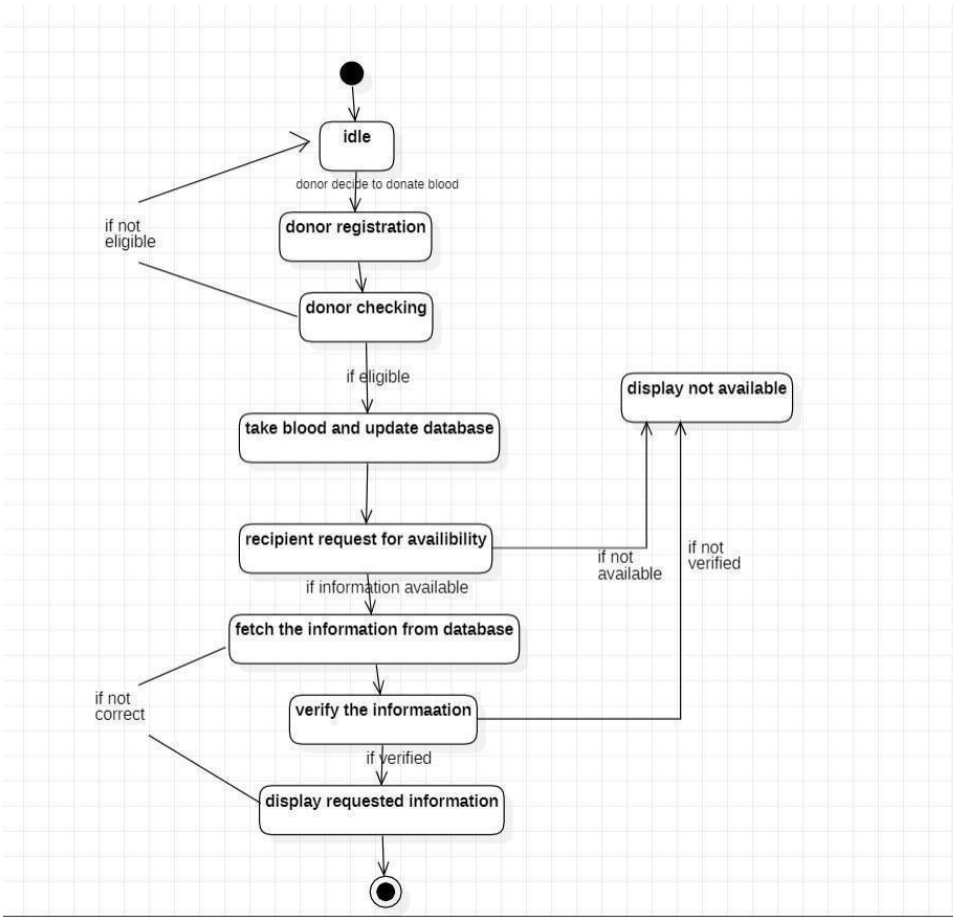


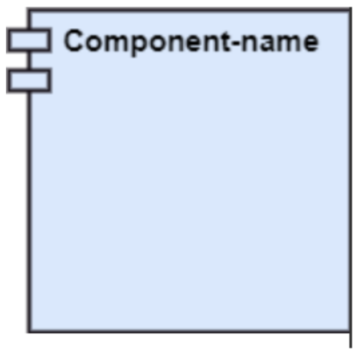
Fig No. 4.4.6 Machine diagram for Blood Bank Management System

6. COMPONENT DIAGRAM

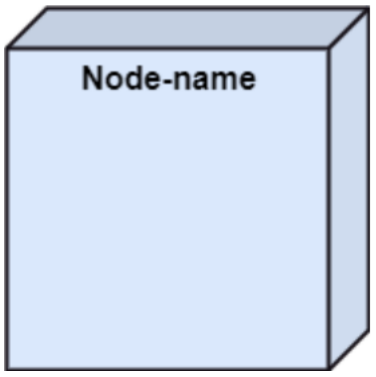
To make a large object-oriented system more manageable, the smaller components are separated out using a component diagram. It simulates the physical view of a system, including its internal nodes' executables, files, libraries, etc. It depicts the connections and hierarchies that exist between the system's components. It aids in the creation of an executable system. An individual, replaceable, and executable system unit is referred to as a component. A component's implementation details are concealed, so an interface is required to carry out a function. It functions like a "black box," with the provided and necessary interfaces explaining its behavior.

Notation of a Component Diagram

a) A component



b) A node



Purpose of a Component Diagram

Given that it is a unique variety of UML diagram, it serves particular functions. It doesn't describe the system's functionalities; it only describes all the individual parts that go into making each one. The system's physical parts are represented visually. The components could be files, libraries, packages, etc. The static view of a system, which includes the arrangement of components at a specific time, is also described by the component diagram. The assortment of component diagrams depicts the entire system. The following list summarizes the component diagram's primary goals:

1. It imagines every part of a system.
2. It incorporates forward and reverse engineering to create the executable.
3. It shows how the various parts are arranged and related to one another.

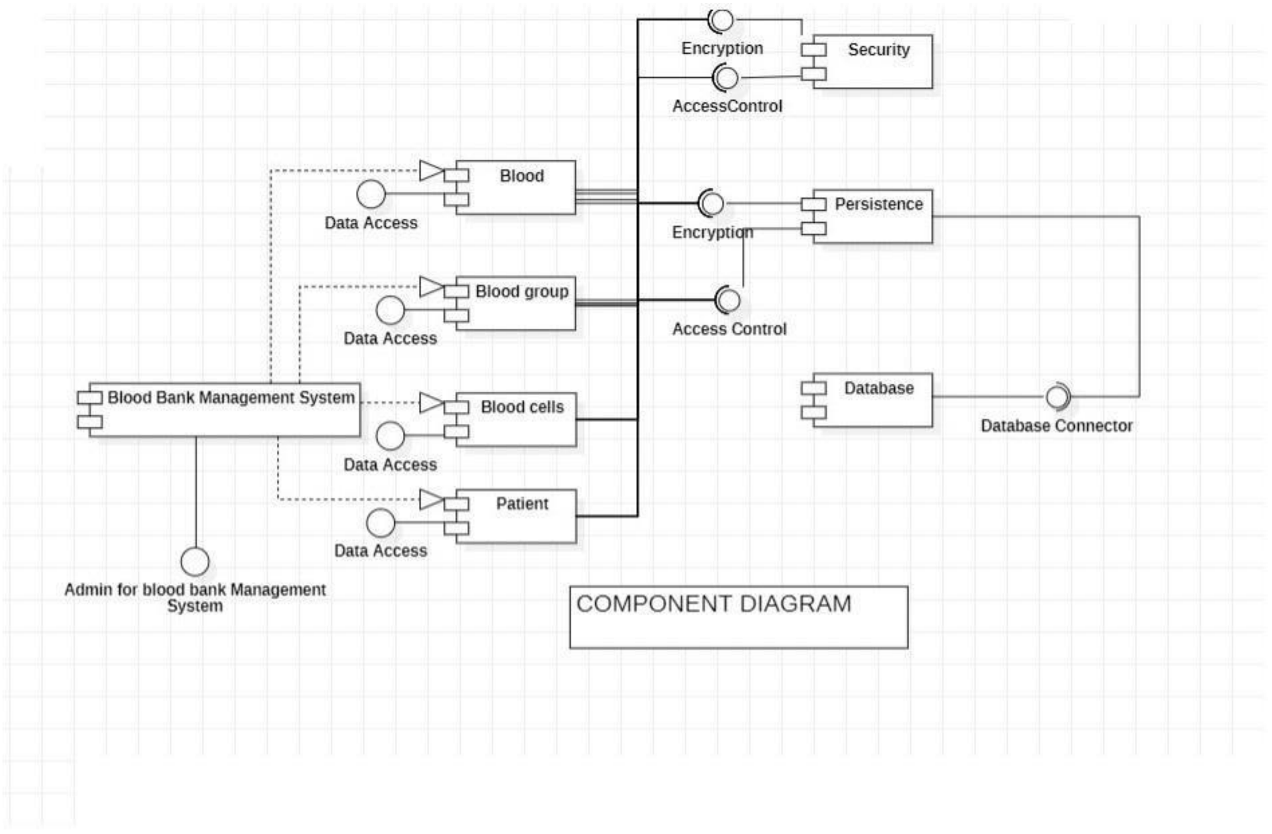


Fig No. 4.4.7 Component diagram of blood bank management system

7. DEPLOYMENT DIAGRAM

Problem Definition: To create a deployment diagram for the Clinic Management System, which is the problem.

Problem Description: A deployment diagram of the issue displays processors, devices, and connections. A single deployment diagram that depicts the connections between nodes is included in each model.

Node: A node is a piece of hardware that can run programmes. Since nodes represent hardware rather than software entities, there are no restrictions on the names that nodes can take.

Pseudo Code:

➔ Click Add Diagram from the dropdown menu,

➔ And then click Add Deployment Diagram after performing a right click on the model.

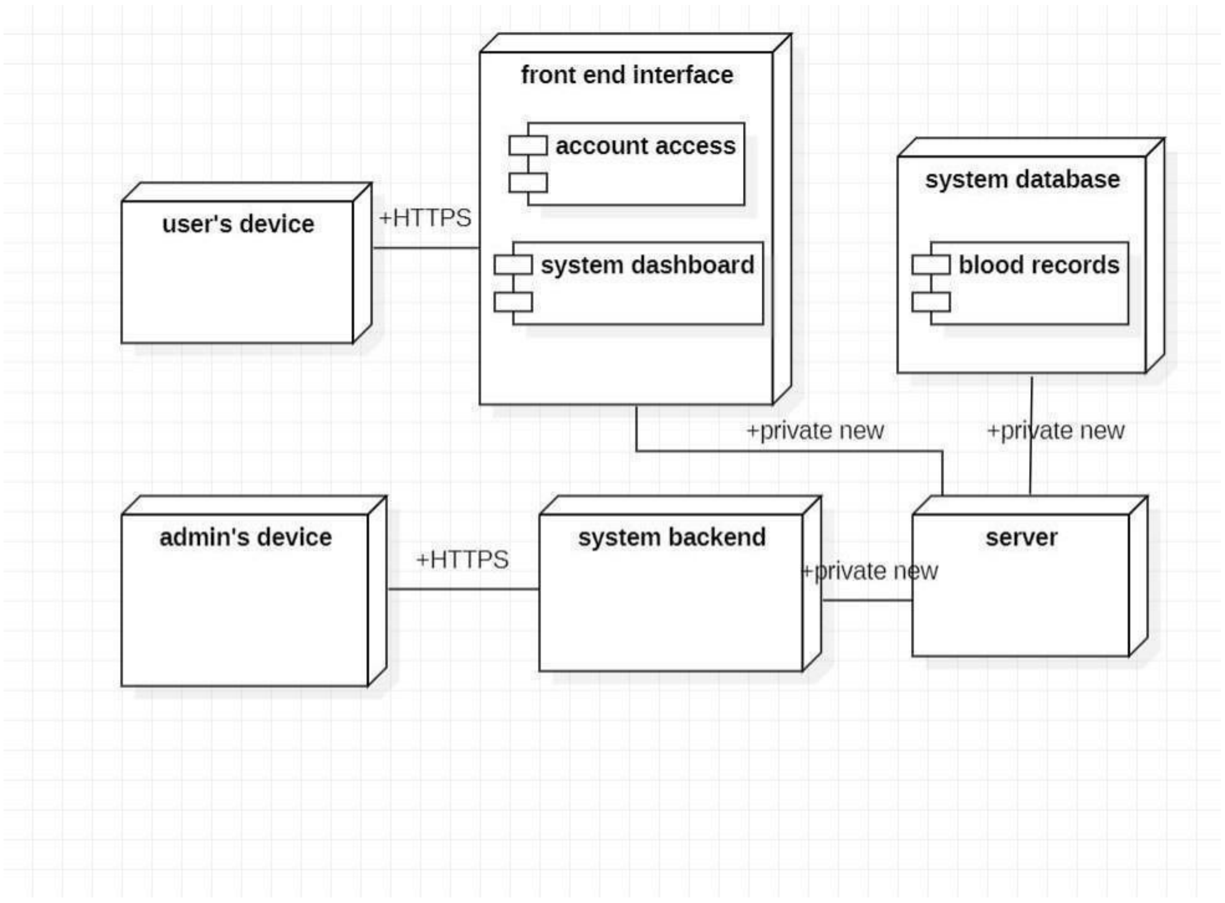


Fig No. 4.4.8 Deployment diagram for Blood Bank Management System

4.5 ER diagram

An ER diagram for a Blood Bank Management System.

Entities:

1. Donor
2. Blood Bank
3. Blood Donation
4. Blood Test
5. Blood Request
6. Blood Inventory
7. Staff

Attributes:

1. Donor

- Donor ID (primary key)
- Name
- Gender
- Age
- Blood group
- Phone number
- Address

2. Blood Bank

- Blood Bank ID (primary key)
- Name
- Address
- Phone number
- Email

3. Blood Donation

- Donation ID (primary key)
- Donor ID (foreign key)
- Donation date
- Blood group
- Quantity

4. Blood Test

- Test ID (primary key)
- Donation ID (foreign key)
- Test date
- HIV status
- Hepatitis B status
- Hepatitis C status

- Syphilis status
- Malaria status

5. Blood Request

- Request ID (primary key)
- Blood group
- Quantity
- Date requested
- Date fulfilled
- Hospital name
- Doctor's name

6. Blood Inventory

- Blood group
- Quantity

7. Staff

- Staff ID (primary key)
- Name
- Gender
- Age
- Phone number
- Address
- Position
- Salary

Relationships:

1. 1. A donor may give blood more than once, and each donation is linked to a single donor.

2. While each blood test can be linked to multiple blood donations, each blood donation is only linked to one blood test.

3. There may be multiple blood inventories at a blood bank, but each blood inventory is only linked to one blood bank.

4. A hospital or doctor may submit more than one request for blood, but each request is

linked to a single hospital and one doctor.

5. Each blood group can be linked to multiple blood requests, but each blood group is only linked to one blood request.

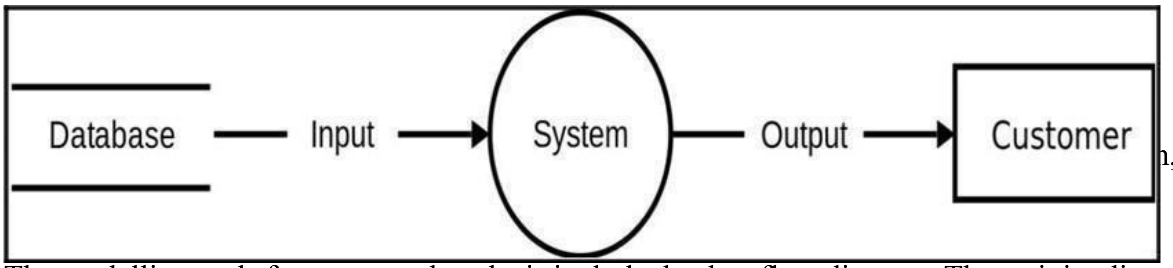
6. A staff member may be connected to more than one blood bank, but only one staff member is connected to each blood bank.

The ER diagram for the Blood Bank Management System:

```
+------------------+         +-----------------+
|     Donor        |         |    Blood Bank   |
+------------------+          +-----------------+
| Donor ID         |      | Blood Bank ID    |
| Name             |      | Name             |
| Gender           |      | Address          |
| Age              |      | Phone number     |
| Blood group      |      | Email            |
| Phone number |
|      Address      |
  +------------------+


+------------------+               +-----------------+
| Blood Donation   |          | Blood Test      |
+------------------+                +-----------------+
| Donation ID       |          | Test ID          |
| Donor ID          |----------| Donation ID  |
| Donation date    |           | Test date       |
| Blood group    |          | HIV status       |
| Quantity        |          | Hepatitis B       |
+------------------+           | Hepatitis C       |
                               | Syphilis status |
                               | Malaria status |
                          +-----------------+


+------------------+             +-----------------+
| Blood Request    |        | Blood Inventory |
+------------------+              +-----------------+
| Request ID      |            | Blood group     |
| Blood group     |----------| Quantity       |
| Quantity            |          +-----------------+
| Date requested   |
| Date fulfilled      |
| Hospital name    |
| Doctor's name    |
  +------------------+


+------------------+
| Staff          |
+------------------+
| Staff ID            |
| Name                |
| Gender            |
| Age                  |
| Phone number |
| Address        | | Position
```

4.5 DFD

A data flow diagram (DFD) is a visual representation of how data flows through a process or system, typically an information system. The DFD also shows the inputs and outputs of each entity as well as the process itself. A data-flow diagram lacks loops, decision rules, and control flows. Using a flowchart, specific operations based on the data can be depicted.



n,

The modelling tools for structured analysis include the data flow diagram. The activity diagram typically replaces the data flow diagram when using UML. A site-oriented data flow plan is a unique type of data flow plan. Processes, flows, warehouses, and terminators are the components of DFD. These DFD components can be viewed in a variety of ways.

Process: A system's process (function, transformation) converts inputs into outputs. Depending on the type of notation, a process is represented by a circle, an oval, a rectangle, or a rectangle with rounded corners. The process is named using a single word, a few words, or a phrase that concisely captures the essence of the operation.

Data Flow: Data flow (flow, dataflow) depicts the movement of information (and occasionally physical objects) from one system component to another. The arrow represents the flow. The name of the flow should specify what data or materials are being moved. Flows that are linked to entities where it is obvious what information is transferred through those entities are exceptions. Systems that are more than just informative models of material shifts Flow ought to only transmit information of one kind (material). If the information going to or coming from the entity is logically dependent, the flow direction can also be indicated by an arrow that is bidirectional. (such as a question-and-answer session). Flows connect terminators, warehouses, and processes.

Warehouse: Data are kept in a storage facility (datastore, data store, file, database) for later use. Two horizontal lines serve as the store's logo; the DFD notation illustrates the other perspective. Orders, for instance, are examples of plural nouns derived from the warehouse's input and output streams. The warehouse need not be limited to data files; it could also include physical items like optical discs, filing cabinets, and folders with documents. So, regardless of implementation, viewing the warehouse in DFD is possible. The reading of data from the warehouse is typically represented by the flow from the warehouse, and the entry or updating of data (and occasionally the deletion of data) is typically represented by the flow to the warehouse. The memory name for the warehouse is located between two parallel lines, and it can be modelled as a UML buffer node.

Terminator: The Terminator is an outsider who interacts with the system while remaining outside of it. It could be, for instance, different organizations (like a bank), groups of people (like customers), authorities (like a tax office), or a division of the same organization (like the human resources department) that does not follow the model system. The terminator could be another system that the modelled system interacts with.

4.5.1 DFD level 0

Another name for it is a context diagram. It is intended to be an abstraction view that presents the system as a lone process with connections to outside entities. It depicts the entire system as a single bubble with incoming and outgoing arrows designating input and output data.

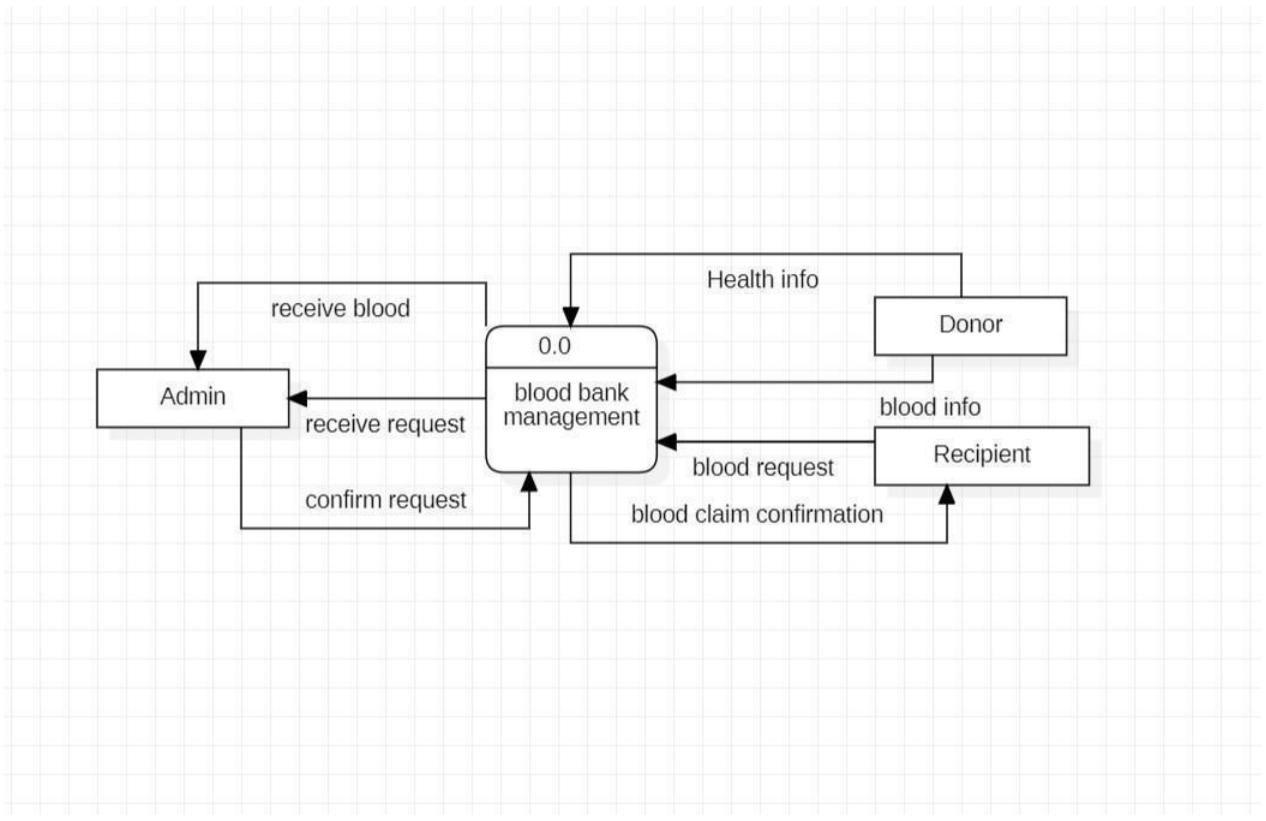


Fig No. 4.4.9 DFD level 0 for Blood Bank Management

4.5.2 DFD level 1

The context diagram is divided into various bubbles and processes in a 1-level DFD. In this level, we highlight the system's primary functions and decompose the high-level DFD process into smaller processes.

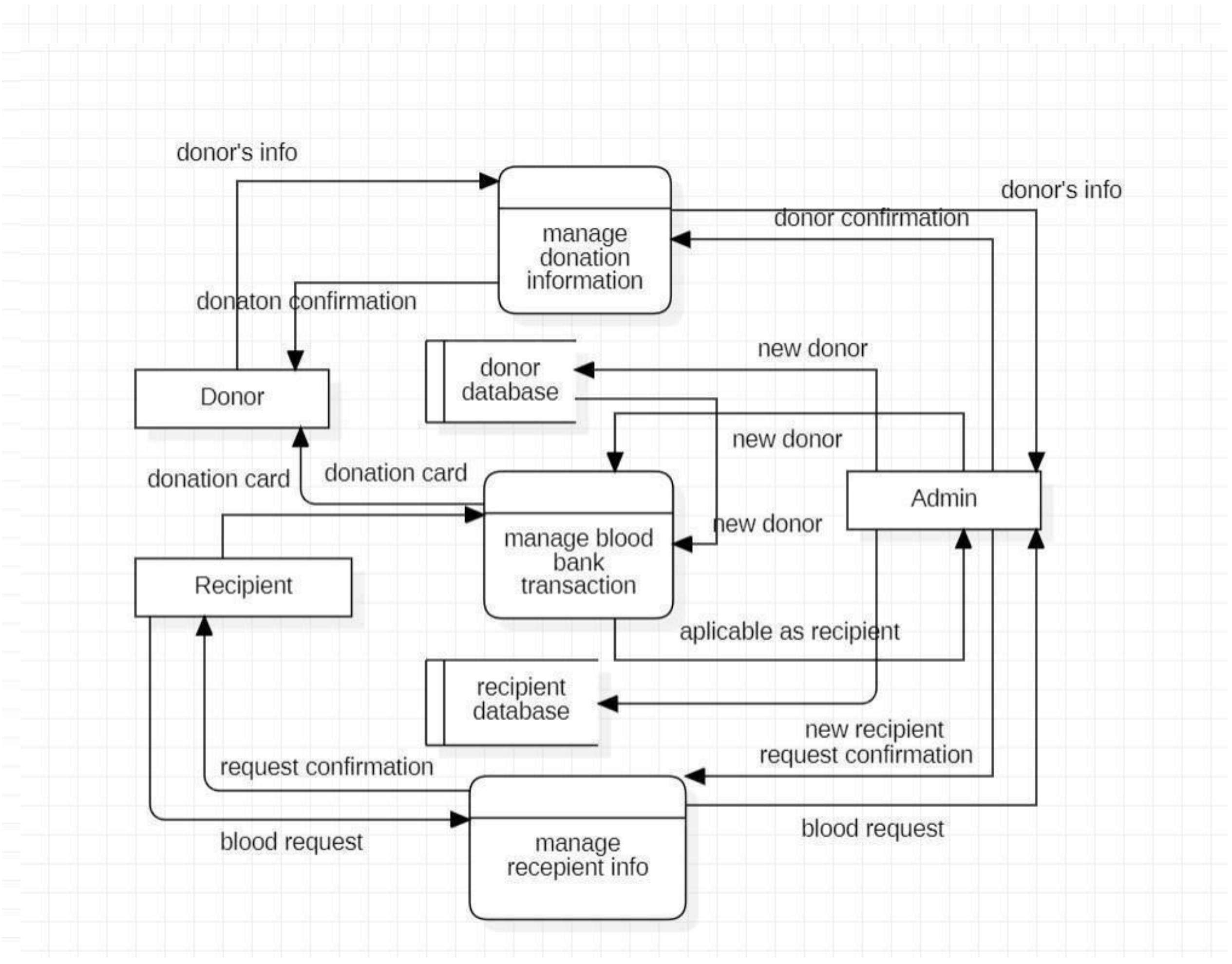


Fig No. 4.4.9 DFD level 1 for Blood Bank Management

3.6.3 DFD level 2

Parts of 1-level DFD are further explored in 2-level DFD. It can be used to plan or keep track of precise or important information about how the system works.

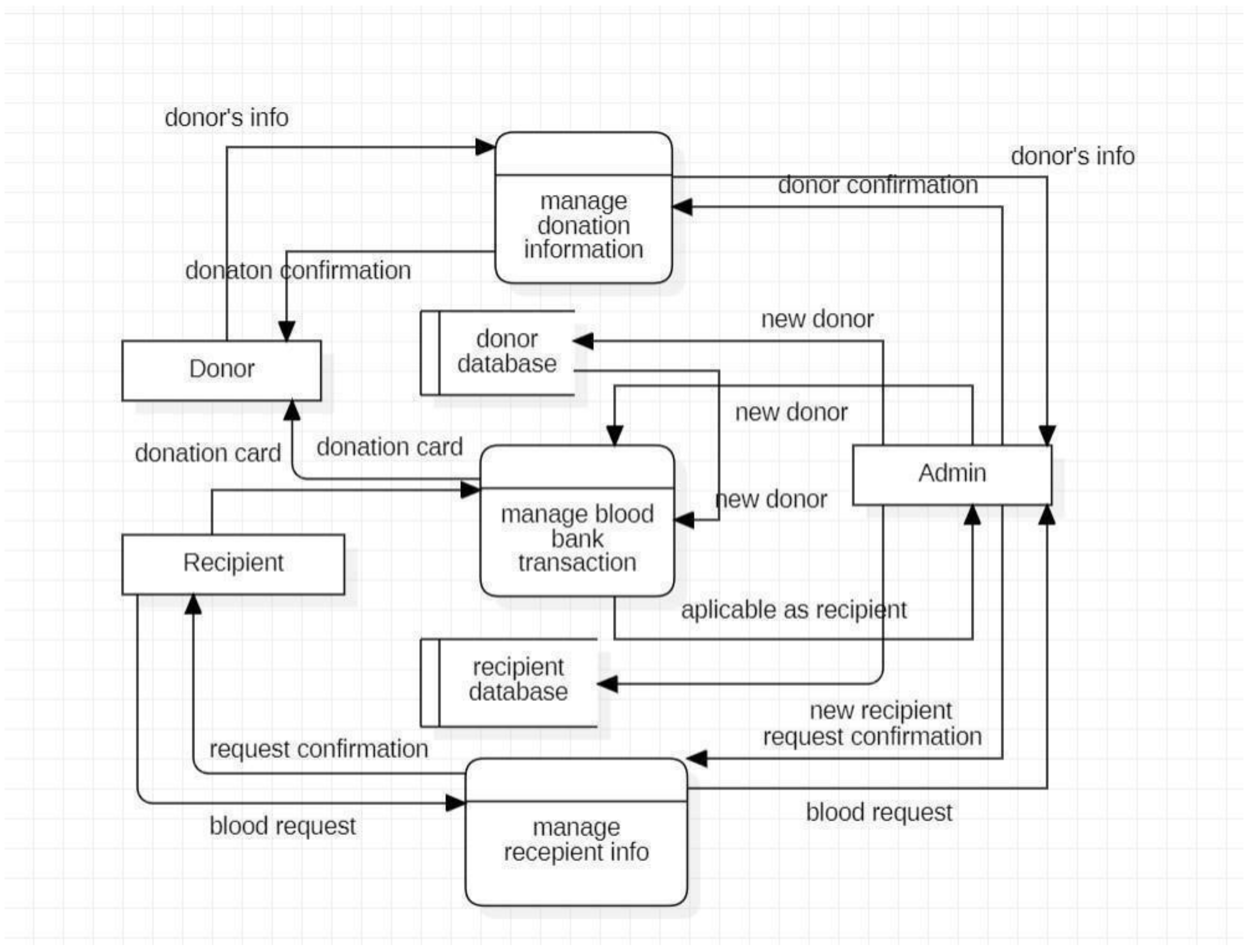


Fig No. 4.4.11 DFD level 2 for Blood Bank Management System

CHAPTER V

IMPLEMENTATION

5.1 Partial code

```
<?php   session_start();

        ?>
<!DOCTYPE html>

<html>
<?php $title="Blood Bank Management System | home page"; ?>
<?php require 'head.php'; ?>
<head>
    <title></title>
      <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <link rel="stylesheet" type="text/css" href="css/styles.css">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <a class="navbar-brand" href="#">Blood Bank Management System</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"data-
target="#navbarSupportedContent"
aria-controls="navbarSupportedContent" aria-expanded="false"aria-
label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav ml-auto">
            <li class="nav-item active">
                <a class="nav-link" href="main.php">Home <span
class="sr-only">(current)</span></a>
            </li>

            <li class="nav-item">
                <a class="nav-link" href="about.php">About</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="contact.php">Contact</a>
            </li>
              <li class="nav-item active">
                <a class="nav-link" href="index.php">Login/Register<span
class="sr-only">(current)</span></a>
            </li>
        </ul>

    </div>
</nav>
<div id="demo" class="carousel slide" data-ride="carousel">

    <!-- Indicators -->
    <ul class="carousel-indicators">
        <li data-target="#demo" data-slide-to="0" class="active"></li>
        <li data-target="#demo" data-slide-to="1"></li>
        <li data-target="#demo" data-slide-to="2"></li>
    </ul>

    <!-- The slideshow -->
    <div class="carousel-inner">
      <div class="carousel-item active">
        <img src="image/slide-1.jpg" alt="Los Angeles" width="1100" height="500">
      </div>
      <div class="carousel-item">
        <img src="image/cc1.jpg" alt="Chicago" width="1100"
  height="500">
      </div>
      <div class="carousel-item">
        <img src="image/blog-4.png" alt="New York" width="1100" height="500">
      </div>
    </div>

    <!-- Left and right controls -->
    <a class="carousel-control-prev" href="#demo" data-slide="prev">
      <span class="carousel-control-prev-icon"></span>
    </a>
    <a class="carousel-control-next" href="#demo" data-slide="next">
      <span class="carousel-control-next-icon"></span>
     </a>
</div>
 <section class="my-5">
    <div class="py-5">
          <h2 class="text-center">About Us</h2>
</div>


    <div class="container-fluid">
      <div class="row">
        <div class="col-lg-6 col-md-6 col-12">
          <img src="image/about1.png" class="img-fluid aboutimg">
```

```
            </div>
            <div class="col-lg-6 col-md-6 col-12">
            <h2>BLOOD - "I'm here to save you!"</h2>
            <p class="py-3">We believe Every life counts!, Every lifematters. Time
is the thing we have and don't. Our goal is to makeblood available in less time and
save your precious life!</p>
            <a href="about.php"></a>
        </div>
      </div>
    </div>
  <section>
  <section class="my-5">
    <div class="py-5">
            <h2 class="text-center">Contact Us</h2>
     </div>
     <div>

  </div>
  </secti
    on>
     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js
"></script>
     <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popp
er.min.js"></script>
     <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min
.js"></script>
     <?php require 'footer.php'; ?>
</body>
</html>
<?php
session_start();
```

```
if (isset($_SESSION['hid'])) {
    header("location:bloodrequest.php");

}elseif (isset($_SESSION['rid'])) {header("location:sentrequest.php");

}else{

?>

<!DOCTYPE html>

<html>

<head>

  <style>

     body{

         background: url(image/RBC11.jpg) no-repeat center;
         background-size: cover;

         min-height: 0;


    }

.login-form{

         width: calc(100% - 20px);
         max-height: 650px;

         max-width: 450px;
         background-color: white;

}

</style>

</head>

<?php $title="Bloodbank | Login"; ?>

<?php require 'head.php'; ?>

<body>

    <?php require 'header.php'; ?>


         <div class="container cont">


             <?php require 'message.php'; ?>
```

```
            <div class="row justify-content-center">
                         <div class="col-lg-4 col-md-5 col-sm-6 col-xs-7 mb-5">

                    <div class="card rounded">
                        <ul class="nav nav-tabs justify-content-center bg-light" style="padding: 20px;">
            <li class="nav-item">
                <a class="nav-link active" data-toggle="tab" href="#hospitals">Hospitals</a>
            </li>
          <li class="nav-item">
                <a class="nav-link" data-toggle="tab" href="#receivers">User</a>
          </li>
        </ul>
        <div class="tab-content">
              <div class="tab-pane container active" id="hospitals">
                <form action="file/hospitalLogin.php" class="login-form" method="post">
                    <label class="text-muted font-weight-bold" class="text-muted font-weight-bold">Hospital Email</label>
                    <input type="email" name="hemail" placeholder="HospitalEmail" class="form-control mb-4">
                    <label class="text-muted font-weight-bold" class="text-muted font-weight-bold">Hospital Password</label>
                    <input type="password" name="hpassword" placeholder="Hospital Password" class="form-control mb-4">
                    <input type="submit" name="hlogin" value="Login" class="btnbtn-primary btn-block mb-4">
                </form>
              </div>
            <divclass="tab-pane container fade" id="receivers">
                <form action="file/receiverLogin.php" class="login-form"
method="post">
                    <label class="text-muted font-weight-bold"
class="text-muted font-weight-bold">User Email</label>
                    <inputtype="email" name="remail" placeholder="User Email"
class="form-control mb-4">
                    <label class="text-muted font-weight-bold"
class="text-muted font-weight-bold">User Password</label>
                    <inputtype="password" name="rpassword" placeholder="User
Password" class="form-control mb-4">
                    <inputtype="submit" name="rlogin" value="Login" class="btnbtn-
primary btn-block mb-4">
                </form>
            </div>

        </div>
        <a href="register.php" class="text-center mb-4" title="Clickhere">Don't have
account?</a>
</div>
</div>
</div>
</div>
<?php require 'footer.php' ?>
</body>
</html>
<?php } ?>
body{

}
.cont{
        padding-top: 40px;
        text-align:justify;
        font-size: 1em;

}

.fas{

        color:#5851DB;

}

.card ,nav, .footer,tr,video{ border-
        radius:    3px; transition:all 200ms
        ease-out;

        box-shadow:0px 0px 7px rgba(0, 0, 0, 0.2);

}

tr,th,td{

        border:0.1px solid silver;
        text-align: center;

}

.nav-item{

margin-left: 10px;

}

.navbar-brand{

        text-shadow: 1px 1px 2px red, 0 0 50px black, 0 0 5px white;font-weight:
        bold;

        font-size: 1.5em;

}

.title{

        font-weight: bold;

        font-size: 1.7em;
        text-transform: capitalize;
        text-align: center;

}

.brand{

        text-shadow: 1px 1px 2px red, 0 0 50px black, 0 0 5px white;

}

.rounded-circle{
        margin-top: -
        2.5px;

}

.card-header{

        font-weight: bolder;
        font-size: 1.5em;

}

.dark-mode {

    background-color: black;
    color: black;

}

form{

        padding-top: 20px;

}

a:link{

        text-decoration: none;

}

.footer{

        padding: 14px;
```

```
}

.alert{
        border:1px solid #5cb85c;
        border-left:5px solid #5cb85c;
```

Application Running

The success of the development of the email client system application from the Android console, which is being used in the emulator, is shown in the screen below. You have access to a number of ways in Android Studio to test web pages as you develop them. Following are your options:

Requirements:

- Xampp Software
- Sublime text/Visual studio code software or any software supporting php, html, css
- Java Jdk

How to start?

- Place this entire folder in htdocs, in xampp(xampp path, installed as per your installation).
- Open Xampp server, start Apache, MySQL.
- In the MySQL row, select Admin.
- Create a database with name "bloodbank" in phpmyadmin. Import the sql file from sql

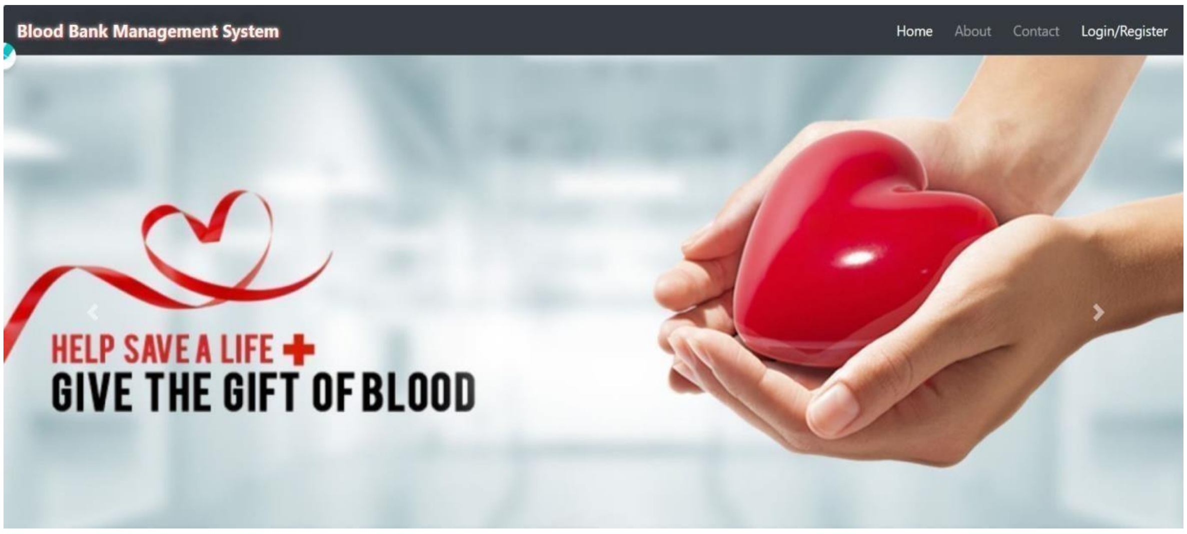
folder.

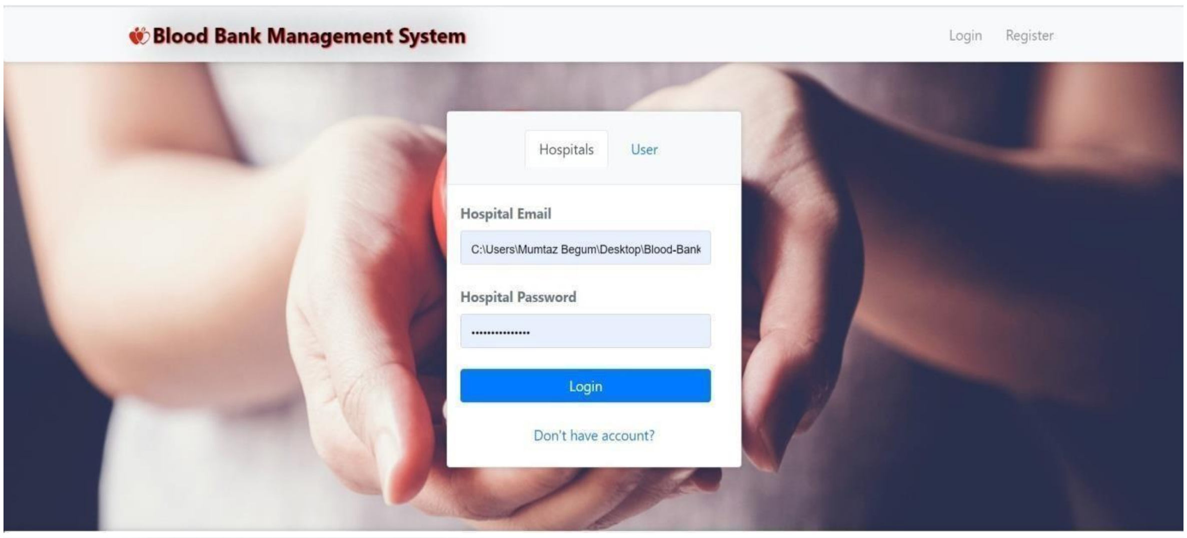
- Open main.php in Sublime text/Visual studio code, on right click copy file path.
- Paste it on any web browser and clear everything before folder name, type

"localhost".

- Ready to go!

5.2 Screenshots



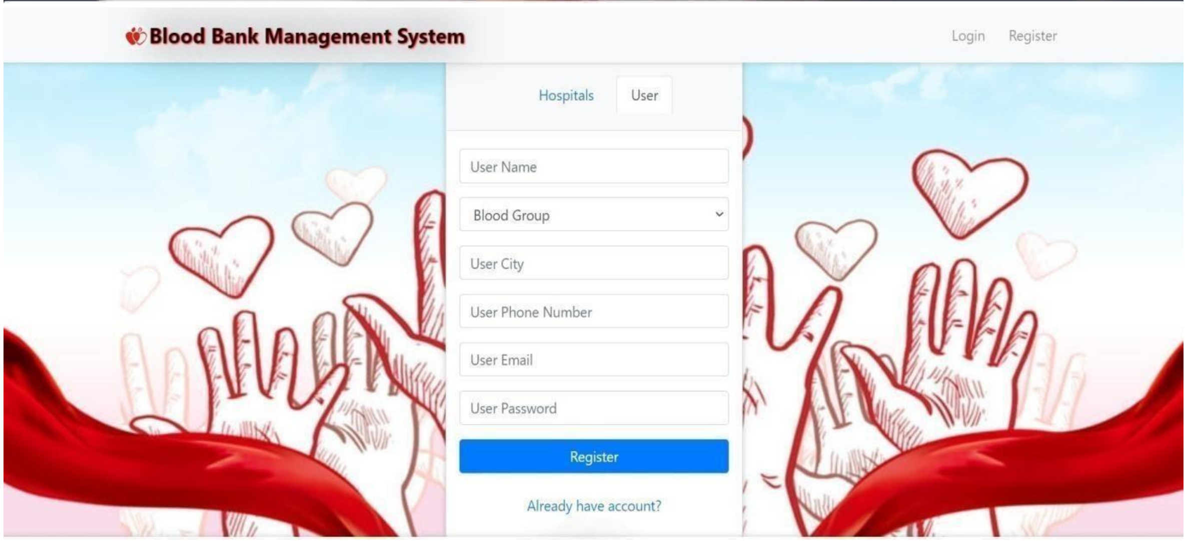


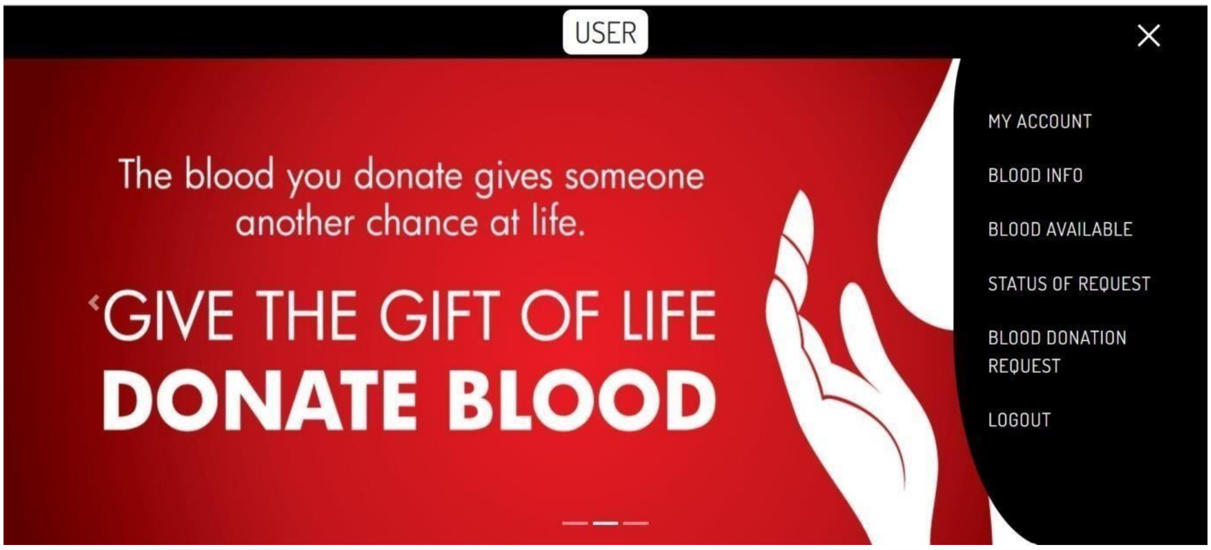
Hospital Email

C:\Users\Mumtaz Begum\Desktop\Blood-Bank

Hospital Password

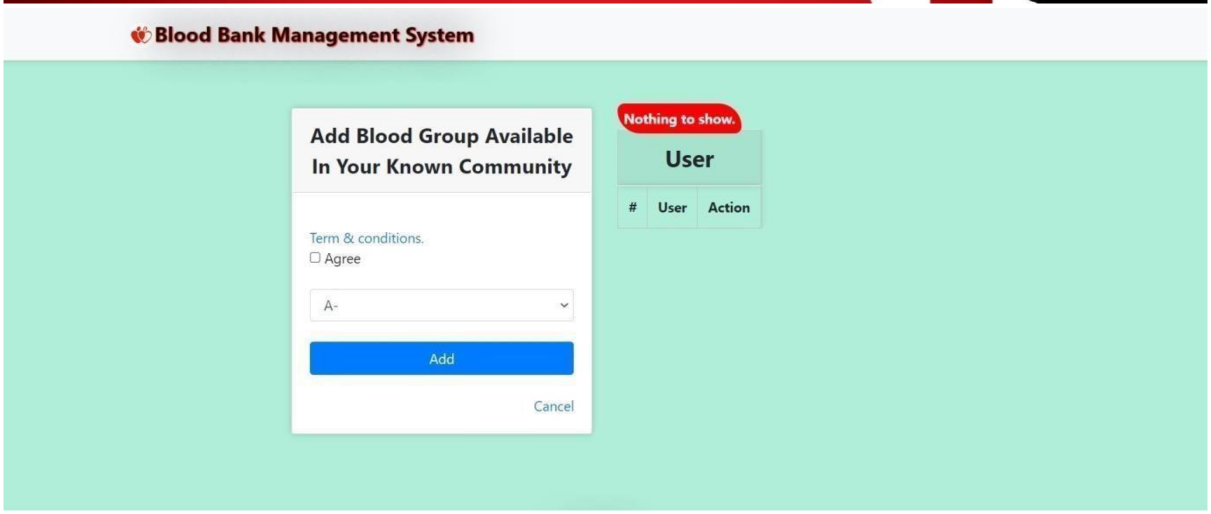
....





The blood you donate gives someone another chance at life.

GIVE THE GIFT OF LIFE
DONATE BLOOD

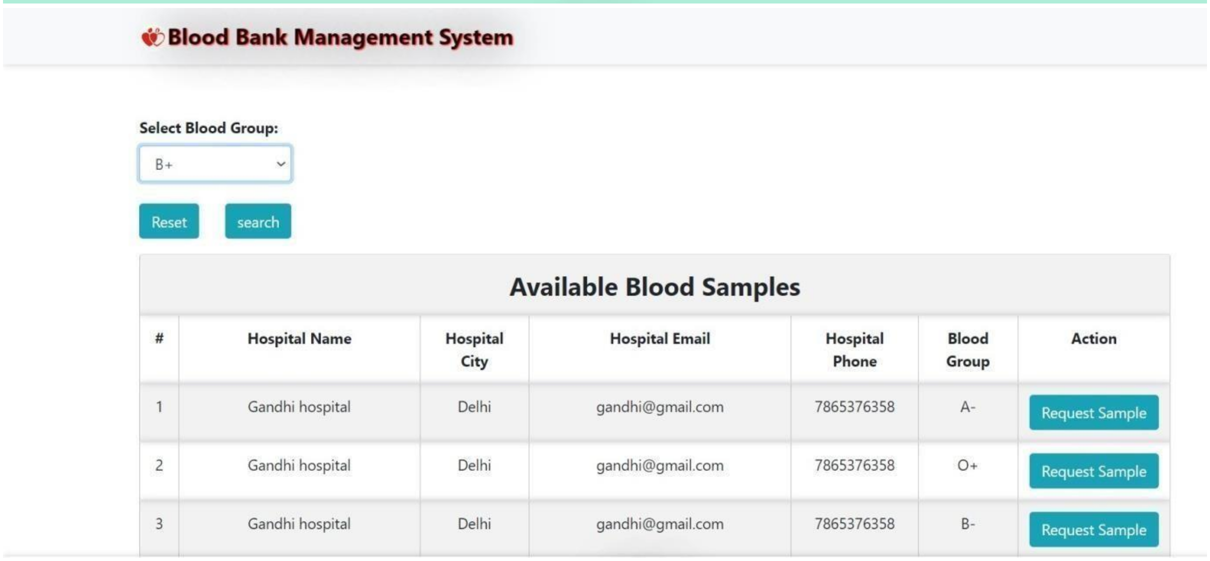


Add Blood Group Available In Your Known Community

Nothing to show.

User

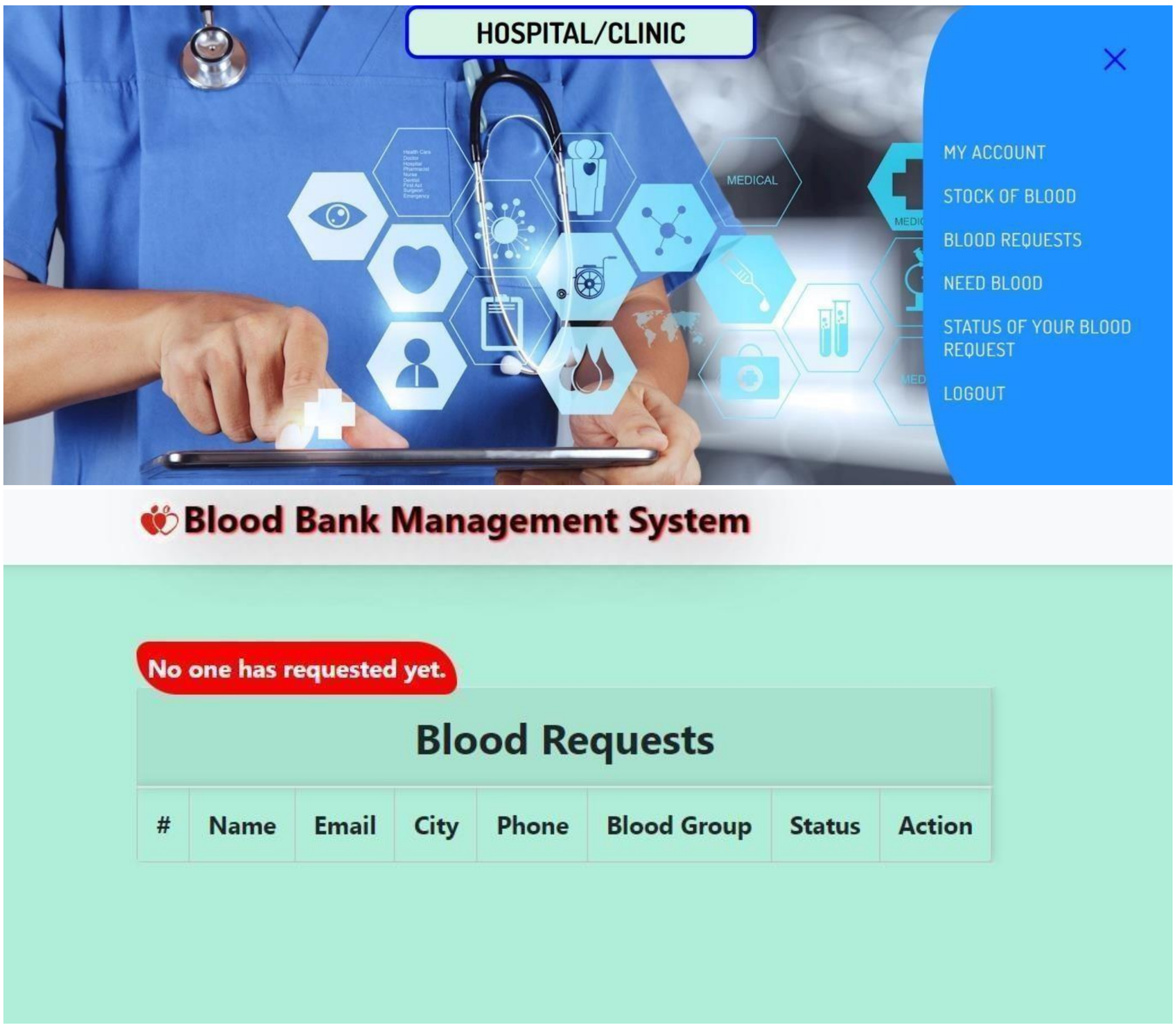
#	User	Action



Select Blood Group: B+

Available Blood Samples

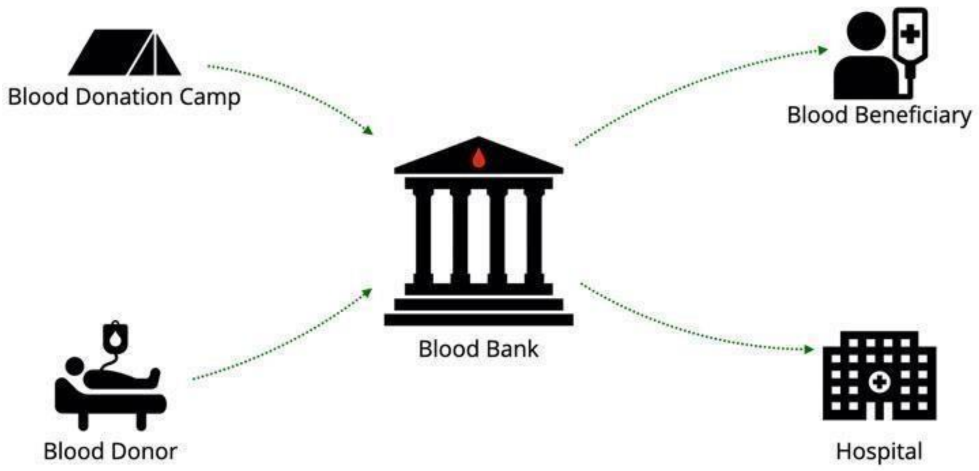
#	Hospital Name	Hospital City	Hospital Email	Hospital Phone	Blood Group	Action
1	Gandhi hospital	Delhi	gandhi@gmail.com	7865376358	A-	Request Sample
2	Gandhi hospital	Delhi	gandhi@gmail.com	7865376358	O+	Request Sample
3	Gandhi hospital	Delhi	gandhi@gmail.com	7865376358	B-	Request Sample

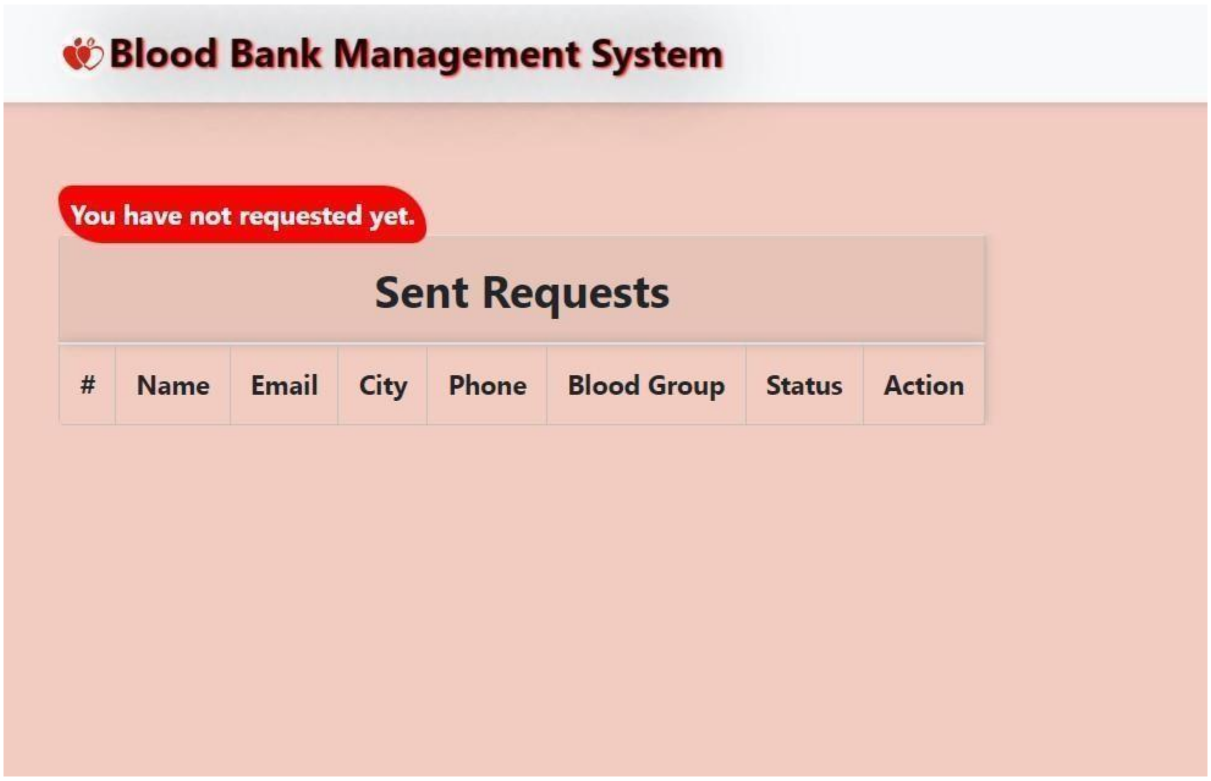


No one has requested yet.

Blood Requests

#	Name	Email	City	Phone	Blood Group	Status	Action

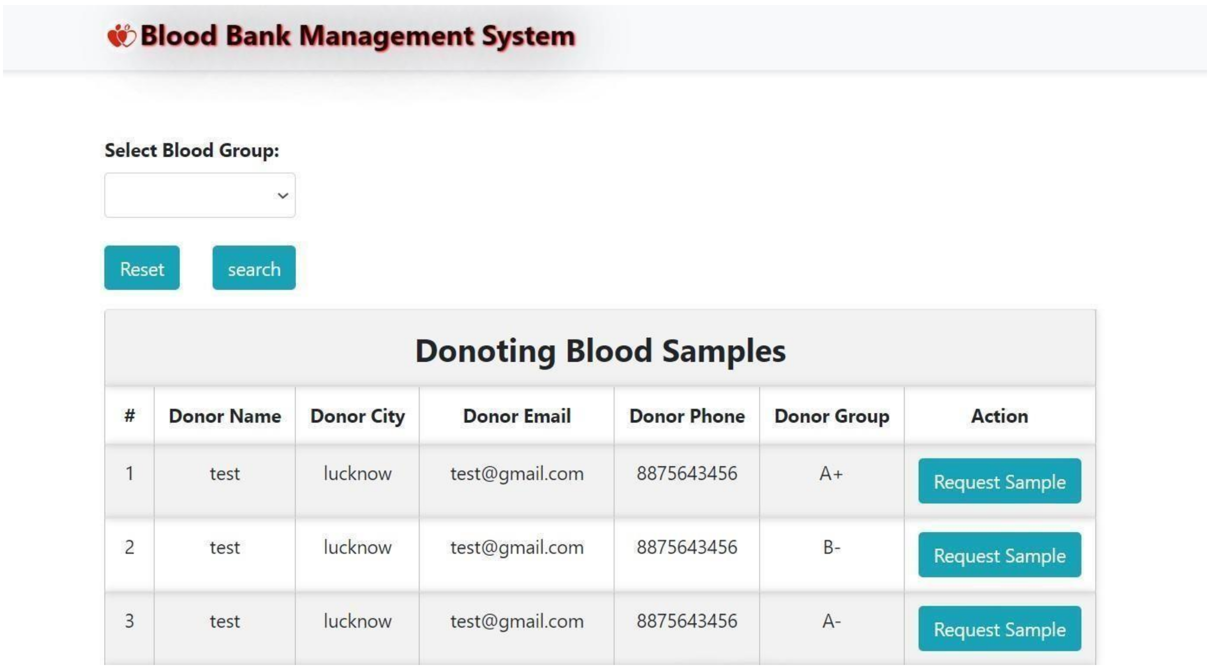



Blood Bank Management System

You have not requested yet.

Sent Requests

#	Name	Email	City	Phone	Blood Group	Status	Action


Blood Bank Management System

Select Blood Group:

Donoting Blood Samples

#	Donor Name	Donor City	Donor Email	Donor Phone	Donor Group	Action
1	test	lucknow	test@gmail.com	8875643456	A+	Request Sample
2	test	lucknow	test@gmail.com	8875643456	B-	Request Sample
3	test	lucknow	test@gmail.com	8875643456	A-	Request Sample

CHAPTER VI

CONCLUSION AND FUTURE ENHANCEMENT

The Blood Bank Management System was created using php.NET and fully achieves the goals for which it was created. All bugs have been eliminated, and the system has reached equilibrium. The system is run with high efficiency, and all the hospitals and users involved in it are aware of its benefits. The issue is resolved by the system. The problem was meant to be solved as a requirement specification.

The project's future potential is enormous. Future internet implementation of the project is possible. The project is very expandable, so it can be updated in the near future as and when the need arises. The client is now able to manage and, as a result, run the entire project in a much better, more accurate, and error-free manner thanks to the proposed software, Database Space Manager, being ready and fully functional.

CHAPTER VII

REFERENCES

Vikas Kulshreshtha, Sharad Maheshwari. (2011).”Blood Bank Management Information System in India”, International Journal of Engineering, 1,2, 260-263. Rational Unified Process, Best Practices for Software Development Teams. (2012). Core Workflows Retrieved from www.ibm.com/developerworks/rational/.../1251_bestpractices Noushin Ashrafi, & Hessam Ashrafi. (2008). Object Oriented Systems Analysis and Design, Pearson Higher Ed USA. Lions Blood Bank & Research Foundation. (2012). Retrieved from http://www.lionsbloodbank.net/ Blood Bank India. (2012). Retrieved from http://www.bloodbankindia.net