

ABSTRACT

This project is aimed at developing an online Blood Donation Management System. Blood Donation Management System is a browser-based system that is designed to store, process, retrieve and manage information on blood donors as well as Blood Banks. It maintains all the information pertaining to blood donors, different blood groups available, and blood banks and helps them manage in a better way. The aim is to provide transparency in this field and make the process of donating blood to a blood bank hassle free which makes the system effective while also increasing the availability of donors for those in need.

ACKNOWLEDGEMENT

The successful completion of any significant task is the outcome of invaluable aggregate combination of different people in radial direction explicitly and implicitly. We would therefore take opportunity to thank and express our gratitude to all those without whom the completion of project would not be possible.

We express our thanks to **Mr. Shivaraj BG, Assistant Professor**, Department of Computer Science and Engineering for having provided all the facilities that helped us in timely completion of this report.

We express our sincere gratitude to **Prof. Ravinarayana B, Associate Professor, Head of the Department, Computer Science and Engineering** for his support and guidance.

We would like to thank **Dr. M S Ganesha Prasad, Principal, Mangalore Institute of Technology and Engineering, Moodabidri** for his support and encouragement.

I express my sincere gratitude to our institution and management for providing us with good infrastructure, laboratory facilities, qualified and inspiring staffs, and whose guidance was of immense help in completion of this seminar successfully.

HEGDE SUMANTH SHYAM 4MT20CS073

PUSHPA MANGAL GOND 4MT20CS120

TABLE OF CONTENTS

CHAPTER	CONTENT	PAGE NO
1	INTRODUCTION	1-2
2	REQUIREMENT ANALYSIS AND SPECIFICATION	2-5
3	SYSTEM DESIGN	6-8
4	IMPLEMENTATION	9-13
5	TESTING	14-16
6	RESULTS AND DISCUSSION	17-22
7	CONCLUSION AND FUTURE ENHANCEMENTS	23
8	REFEERENCES	24

Chapter 1

INTRODUCTION

A blood donation is a process whereby a person voluntarily has blood drawn to be used for future transfusions when in need at hospitals for treatment procedures that require them. Donations may be of whole blood (blood drawn directly from the body) or of specific components of the blood; such as red blood cells, white blood cells, plasma, and platelets. Blood banks often participate in the process of collecting blood and other procedures such as conducting donation drives, approving blood requests, and updating donation information and requirements.

The inspiration of this project is to help blood banks and to develop a blood bank information system that focuses on making an online system that is accessible for both donors and blood banks.

The purpose of the Blood Donation Management System project is to develop and implement a computerized system to manage the information of blood donors and to connect the blood banks to these donors. This system acts as a bridge between Donors and Blood Banks for an easy, fast, and efficient donation of blood. Donors can directly register themselves for blood donations, by providing the necessary information. They can also update their personal information through the system, without having to contact the blood bank registry or an administrator. The administrator is also responsible for maintaining the records of registered blood banks and their blood requests.

The entire project has been developed keeping in view the distributed client-server computing technology, in mind. The Blood Donation Admin is to create an e-Information about the donor and organization or blood banks that are related to donating blood. Through this application, any person who is interested in donating blood can register himself. But if any organization wants to register itself with this site it has to be verified and approved by Admin. Moreover, if any registered blood bank wants to make a request for blood online, it can also take the help of this site. Admin is the main authority who can do addition, deletion, and modification if required.

The project has been planned to be having a 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 MySQL through XAMPP Server and all the user interfaces

have been designed using Web technologies. User The database connectivity is planned using the “Apache Web Server” methodology. The standards of security and data protective mechanisms have been given a big choice for proper usage. The application takes care of different modules and their associated reports, which are produced as per the applicable strategies and standards.

The entire project has been developed keeping in view the distributed client-server computing technology, in mind. The user interfaces are browser independent to provide uniform access to the overall system. The internal database has been selected as MySQL database. The basic constructs of table spaces, clusters, and indexes have been exploited to provide higher consistency and reliability for data storage. MySQL was a choice because it provides the constructs of high-level reliability and security. The total front end was implemented completely using Handlebars, CSS and JavaScript while the backend is managed by NodeJs. We have extensively used ExpressJs as Web Framework for NodeJs. At all proper levels, high care was taken to check that the system managed the data consistency with proper rules or validations. The database connectivity was planned using the latest “Apache Web Server” technology provided by XAMPP. The authentication and authorization were crosschecked at all the relevant stages.

1.1 Objectives

- To have latest updates on blood donation drive and donors available.
- Use of Web Services and Remoting.
- To have a centralized system to manage blood donation.
- To effectively monitor or manage the related data.
- To provide security with different level of authentication.
- To have proper co-ordination between Blood Banks, Donors, Users and Applications.
- To improve donor recruitment and retention by providing a user-friendly online donation process, tracking donor information and allowing them to view donation drives.
- To increase data security and compliance with regulations by providing a secure and auditable system for storing and managing sensitive donor and patient information.
- To spread awareness on Blood donation and educate the people

Chapter 2

REQUIREMENT ANALYSIS AND SPECIFICATION

2.1 Functional Requirements

The functional requirements of a Blood Donation Management System (BDMS) will vary depending on the specific needs and goals of the blood banks that are using it. However, some common functional requirements that might be included are:

Donor registration and management: The BDMS should enable the registration of new donors, login of existing donors, and the management of existing donor information, including personal details, contact information, etc.

Donation drives: The system should allow donors to view and contact blood donation drives by blood banks.

Blood Bank Information management: The system should enable the registration and management of blood banks and all the related information including contact information and their requirements and blood drives.

User management: The BDMS should have a user management system that controls access to the system and different functionalities based on the user's role.

Data security: The system should have robust security features to protect donor information and comply with data privacy regulations.

Compliance: The system should comply with all relevant regulations and guidelines for blood donation and management.

Mobile access: The system should have a mobile version to enable access from multiple devices and locations

Integration: The system should be able to integrate with other systems, such as electronic medical records, to improve information sharing and coordination.

Feedback and suggestion management: The system should have a mechanism for receiving and managing feedback, messages, and suggestions from users, to improve the system and user experience.

Dashboard and notifications: The system should have a dashboard to provide real-time data on key performance indicators, such as donor count, donor activity, and drive information. The system should also have tabs to enable users to view drives from various blood banks or any other important information.

Hardware requirements

- Processor : Intel i3/i5, 1.8GHz machine or above.
- Main memory : 4GB RAM or more.
- Hard disk drive : 1GB or more

Software requirements

- Operating System : Windows 7 and higher.
- Front-end Interface : HandleBars, CSS, JavaScript.
- Back-end : NodeJs, ExpressJs, MySQL.
- Tools : Visual Studio code, XAMPP.

2.2 Non-Functional Requirements

Non-functional requirements are requirements that are not directly concerned with the specific functions delivered by the system. They may relate to emergent system properties such as reliability, response time, and store occupancy. Alternatively, they may define constraints on the system such as the capabilities of I/O devices and the data representations used in system interfaces. The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture. Non-functional requirements are often called qualities of a system. Other terms for non-functional requirements are “constraints”, “quality attributes”, “quality goals”, “quality of service requirements” and “non-behavioral requirements”. It describes aspects of the system that are concerned with how the system provides the functional requirements.

Some of non-functional requirements for our Blood Donation Management System (BDMS) include:

Performance: The system is able to handle a large number of concurrent users, process data quickly and efficiently, and generate reports and statistics in a timely manner.

BLOOD DONATION MANAGEMENT

Security: The system is designed to protect donor and patient information from unauthorized access and ensure compliance with data privacy regulations.

Usability: The system is user-friendly, easy to navigate and understand, and accessible for all users, including those with disabilities.

Scalability: The system is able to handle an increase in the number of donors, blood banks, and blood requests over time.

Maintainability: The system is easy to maintain and update, with minimal downtime and disruptions to service.

Accessibility: The system is accessible remotely via web and mobile devices to allow easy access by users and blood banks.

Integration: The system is able to integrate with other relevant systems such as electronic medical records and inventory management systems.

Compliance: The system should be compliant with all relevant regulations, standards, and guidelines for blood donation and management.

Chapter 3

SYSTEM DESIGN

The system design process partitions the system into subsystems based on the requirements. It establishes overall system architecture and is concerned with identifying various components, specifying relationships among components, specifying software structure, maintaining a record of design decisions and providing a blueprint for the implementation phase. Design consists of architecture design and detailed design. Architecture design is concerned with the details of how to package processing modules and how to implement the processing algorithms, data structures and interconnections among modules and data structures.

For the designing of the project, we used Handlebars for the layout of the website, CSS for styling the page, and JavaScript for interaction. Firstly, we have designed a home page where the user is able to sign up and log in, and a similar case with blood banks. Then we have written the logical part on the server side using NodeJs and ExpressJs. Then we connected the MySQL database using Apache Web Server in XAMPP. Now users could view, update or unregister from the website as per his/her wish.

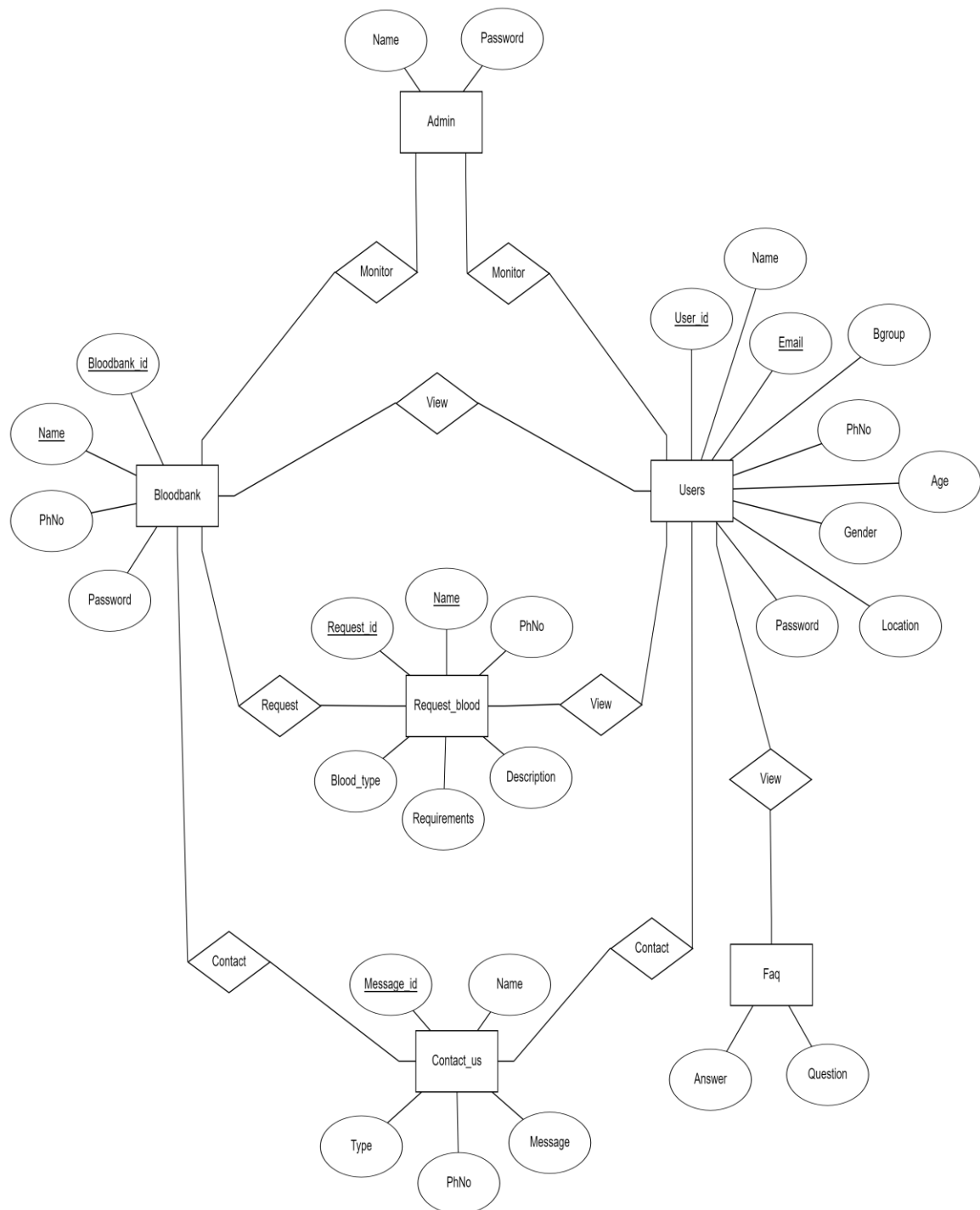
3.1 Entity-Relationship(ER) Diagram

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases.

- Helps you to define terms related to entity relationship modelling
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- Helps to describe entities, attributes, relationships.
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of an ER diagram.

BLOOD DONATION MANAGEMENT

Entity Relationship Diagram is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes, and relationships. ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes, and diamond shapes to represent relationships.



3.2 Schema Diagram

A Schema Diagram is a diagram that contains entities and the attributes that will define that schema. A schema diagram only shows us the database design. It does not show the actual data of the database. A schema can be a single table or it can have more than one table which is related. The schema represents the relationship between these tables.

Admin

Name	Password
------	----------

Users

<u>User_id</u>	Name	<u>Email</u>	Bgroup	PhNo	Age	Gender	Location	Password
----------------	------	--------------	--------	------	-----	--------	----------	----------

Bloodbank

<u>Bloodbank_id</u>	<u>Name</u>	PhNo	Password
---------------------	-------------	------	----------

Request_Blood

<u>Request_id</u>	<u>Name</u>	PhNo	Blood_type	Requirements	Description
-------------------	-------------	------	------------	--------------	-------------

Contact_us

<u>Message_id</u>	Name	Type	PhNo	Message
-------------------	------	------	------	---------

FAQ

Question	Answer
----------	--------

Chapter 4

IMPLEMENTATION

The idea of this whole project is to overcome all the above difficulties while providing a centralized database to manage all the data related to blood donation and also to provide various functionalities to this data at the same time.

This approach includes the usage of Web services to provide a platform to connect both blood banks and blood donors. It also includes a section to educate the people about the importance of blood donation and also to spread awareness in society and answer all frequently asked questions.

The entire project has been developed keeping in view of the distributed client-server computing technology, in mind. The user interfaces are browser independent to provide uniform access to the overall system. The internal database has been selected as the MySQL database. The whole front end was implemented completely using Handlebars, CSS and JavaScript. The backend was supported by NodeJs. We have extensively used ExpressJs as Web Framework for NodeJs. This provides high efficiency and fast response times. The database connectivity was planned using the latest “Apache Web Server” technology provided by XAMPP. The authentication and authorization were crosschecked at all the relevant stages.

Technologies used in this project development are: - Handlebars, CSS, JavaScript, MySQL, Apache Web Server, XAMPP, NodeJs, ExpressJs, Body-Parser, Nodemon, Git, and GitHub.

4.1 Technologies

Handlebars

Handlebars is a popular, open-source JavaScript library that is used for building semantic templates. It provides a simple way to create and manage templates for dynamic web pages. It allows developers to separate the structure of a web page (i.e. the HTML) from the data that is displayed on the page, reducing the amount of code needed to display data on a web page, which makes it easier to maintain and update the code.

Handlebars can be used with a variety of JavaScript frameworks, such as Ember.js and Backbone.js, as well as with server-side frameworks like Express.js. It is compatible with both client-side and server-side rendering.

CSS

CSS, or Cascading Style Sheets, is a style sheet language used for describing the presentation of a document written in a markup language. It is used to control the layout and formatting of web pages, including colors, fonts, spacing, and other visual elements.

With CSS, developers can separate the presentation of a web page from its structure and content, making it easier to maintain and update the design of a website. This is done by linking a CSS file to an HTML document.

JavaScript

JavaScript is a versatile and powerful language that can be used for both front-end and back-end development. On the front end, it can be used to create and handle interactive user interfaces. On the back end, it can be used to create servers, web applications and work with databases. JavaScript is a programming language that is primarily used to create interactive and dynamic user interfaces for websites. It can be used to add a wide range of functionality to web pages, such as form validation, image sliders, etc.

MySQL

MySQL is a free and open-source relational database management system (RDBMS). It is widely used in web applications and data-driven websites to store and manage data. MySQL is known for its reliability, ease of use, and performance.

MySQL stores data in tables, with rows and columns. Each field contains a specific type of data, such as text, numbers, or dates. MySQL uses a variant of the SQL (Structured Query Language) to interact with the database, which allows developers to insert, update, retrieve, and delete data in the database.

XAMPP

XAMPP is an open-source software that provides an easy way to install and run Apache web server, PHP, and MySQL on a local machine. It stands for Cross-Platform (X), Apache (A), Maria DB (M) (a fork of MySQL), PHP (P), and Perl (P). This package allows developers to create and test web applications on their local computers, without the need for a live web server.

Apache Web Server

Apache is an open-source web server software that is widely used on the Internet. It is known for its stability, security, and flexibility, and it is supported by a large community of developers.

Apache is responsible for handling HTTP requests and returning the appropriate response to the client. When a client (such as a web browser) requests a web page, the request is sent to the Apache web server, which then processes the request and returns the requested page to the client.

NodeJs

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a web browser. It allows developers to use JavaScript on the server-side to create server-side applications, networking tools, and other types of back-end tools. It also provides a large library of modules, known as the Node Package Manager (NPM), which can be easily installed and used in Node.js applications.

Node.js uses the JavaScript language, which makes it a popular choice among web developers who are already familiar with JavaScript and want to use it for both the front-end and back-end of their web applications. Node.js can also be integrated with other technologies such as MySQL, MongoDB, ExpressJs, and AngularJS to build full-stack web applications.

ExpressJs

Express.js is a web application framework for Node.js, it is a minimal and flexible framework that provides a robust set of features for web and mobile applications. It is designed to simplify the process of building web applications with Node.js. It provides a simple and easy-to-use API for handling HTTP requests and responses, middleware, etc. It also allows developers to easily connect with databases and other web services.

4.2 Code Snippets

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host : 'localhost',
  user : 'admin',
  password : 'admin',
  database: 'blood_donation',
  port :3306,
})
connection.connect(function(err) {
  if(err) {
    console.log("Not connected")
    throw err
  }
  console.log("Connected")
})
module.exports = {connection, mysql}
```

Figure 4.2.1 Code Snippet of Connection Module

```
exports.userSignUp = async (req, res) => {
  const userName = req.body.name
  const userEmail = req.body.email
  const userPassword = req.body.password
  const userGroup = req.body.group
  const userPhNo = req.body.PhNo
  const userAge = req.body.age
  const userGender = req.body.gender
  const userLocation = req.body.location
  const encryptedPassword = userPassword
  connection.query(`insert into users(Name, Email, Bgroup, PhNo, Age, Gender, Location, Password)
    Values ("${userName}", "${userEmail}", "${userGroup}", "${userPhNo}", "${userAge}",
    "${userGender}", "${userLocation}", "${encryptedPassword}")`, (err, results, field) => {
    if (err) { console.log(err)
      if (err.errno === 1062) {
        return res.status(406).send({ "message": "The entered email is already registered" })
      }
      res.status(500).send({ "message": "Server error" })
    } else {
      const finish = () => { res.redirect('/') }
      setTimeout(finish, 1500)
    }
  })
};
```

Figure 4.2.2 Code Snippet of User SignUp

```

exports.bloodBankLogin = (req, res) => {
  const userName = req.body.name
  const userPassword = req.body.password
  const userReq = req.body.requirements
  const userPhNo = req.body.phno
  const updateFlag = req.body.flag
  const userDesc = req.body.desc
  if(updateFlag == 1) { //Requesting blood
    connection.query(`insert into Request_Blood(Name, PhNo, Requirements, Description) Values
    ("${userName}", "${userPhNo}", "${userReq}", "${userDesc}")`, (err, results, field) => {
      if (err) { console.log(err)
        if (err.errno === 1062) {
          return res.status(406).send({ "message": "The entered email is already registered" })
        }
        res.status(500).send({ "message": "Server error" })
      } })
  } else if(updateFlag == 2) { //updating the amount of blood requested
    connection.query(`update Request_Blood Set Requirements = "${userReq}", Description =
    "${userDesc}" where Name = "${userName}"`, (error, userresults, fields) => {
      if(error) { console.log(error)
        res.status(500).send({ "message": "Server error" })
      }
    })
  } //redirecting back to blood bank dashboard after authenticating
  connection.query(`select * from BloodBank where Name = "${userName}"`, async (err, results, field)
  => {
    if (err) { res.status(500).send({ "message": "Server error" })
      console.log(err)
    } else {
      if (results.length == 0) { return res.status(401).send({ "message": "The entered credentials do
not match" }) }
      match = (userPassword == results[0].Password) &&(userName == results[0].Name)
      if (match) {
        connection.query(`Select * from users`, async (error, userresults, fields) => {
          res.render('bloodbankdash', {
            result: userresults,
            user: userName,
            'phno': results[0].PhNo,
            'requirements': userReq,
            password: userPassword
          }) }) } })
      }
    }
  };

```

Figure 4.2.3 Code Snippet of Bloodbank Login, requesting blood and updating blood requirements

```

exports.unregister = async (req, res) => {
  connection.query(`delete from users where Email="${req.body.email}"`, (err, results, field) => {
    if (err) {
      console.log(err)
      res.status(500).send({
        "message": "Server error"
      })
    } else {
      // res.status(200).send()
      res.redirect("/")
    }
  })
};

```

Figure 4.2.4 Code Snippet of user unregistering from the website and database

Chapter 5

TESTING

Software testing is the process used to identify the correctness, security, completeness, and quality of developed computer software. This includes the process of executing the program or applications with the intent of finding errors. An individual unit, functions, or procedures of the developed project is verified and validated and these units are fit for use.

5.1 Testing Process

Best testing process is to test each subsystem separately, as we have done in project. Best done during implementation. Best done after small sub-steps of the implementation rather than large chunks. Once each lowest level unit has been tested, units are combined with related units and retested in combination. This proceeds hierarchically bottom-up until the entire system is tested as a whole. Typical levels of testing:

- 1 **Module-** package, abstract data type, class
- 2 **Sub-system-** collection of related modules, cluster of classes, method-message paths
- 3 **Acceptance testing-** whole system with real data (involve customer, user, etc).

Alpha testing is acceptance testing with a single client (common for bespoke systems). Beta testing involves distributing system to potential customers to use and provide feedback. In this project, beta testing has been followed. This exposes system to situations and errors that might not be anticipated by us.

5.1.1 Unit Testing

Unit testing is the process of testing individual software components units or modules. Since it needs detailed knowledge of the internal program design and code this task is done by the programmer and not by testers.

5.1.2 Integration Testing

Integration testing is another aspect of testing that is generally done in order to uncover errors associated with the flow of data across interfaces. The unit-tested modules are grouped together and tested in a small segment, which makes it easier to isolate and correct errors. This approach is continued until we have integrated all modules to form the system.

As completion of each module, it has been combined with the remaining module to ensure that the project is working properly as expected.

5.1.3 System Testing

System testing tests a completely integrated system to verify that it meets its requirements. After the completion of the entire module, they are combined together to test whether the entire project is working properly.

5.2 Test Cases

A Test Case is a software testing document, which consists of events, actions, input, output, expected result, and actual result. Technically a test case includes a test description, procedure, expected result, and remarks. Test cases should be based primarily on the software requirements and developed to verify correct functionality and establish conditions that reveal potential errors.

Test cases no	Test Case	Expected results	Status
1	Logging	Username, email and password correct	Successful
2	Logging	Username incorrect	Unsuccessful
3	Logging	Email Incorrect	Unsuccessful
4	Logging	Password Incorrect	Unsuccessful
5	Logging	Any field left empty	Unsuccessful

Table 5.2.1 Test Case for user login

Test cases no	Test Case	Expected results	Status
1	Registration for new user	All details provided correctly	Successful
2	Registration for new user	Any one field is incorrect	Unsuccessful
3	Registration for new user	Any field left empty	Unsuccessful

Table 5.2.2 Test Case for user signup

Test cases no	Test Case	Expected results	Status
1	Editing user information	All details provided correctly	Successful
2	Editing user information	Current password incorrect	Unsuccessful
3	Editing user information	Any field left empty	Successful (No change)

Table 5.2.3 Test Case for editing user information

Test cases no	Test Case	Expected results	Status
1	Logging	Name and password correct	Successful
2	Logging	Name incorrect	Unsuccessful
3	Logging	Password Incorrect	Unsuccessful
4	Logging	Any field left empty	Unsuccessful

Table 5.2.4 Test Case for blood bank login

Test cases no	Test Case	Expected results	Status
1	Requesting blood	All details correct	Successful
2	Updating requested blood	All details correct	Unsuccessful

Table 5.2.5 Test Case for blood bank requesting/updating blood requirements

Chapter 6

RESULT AND DISCUSSION

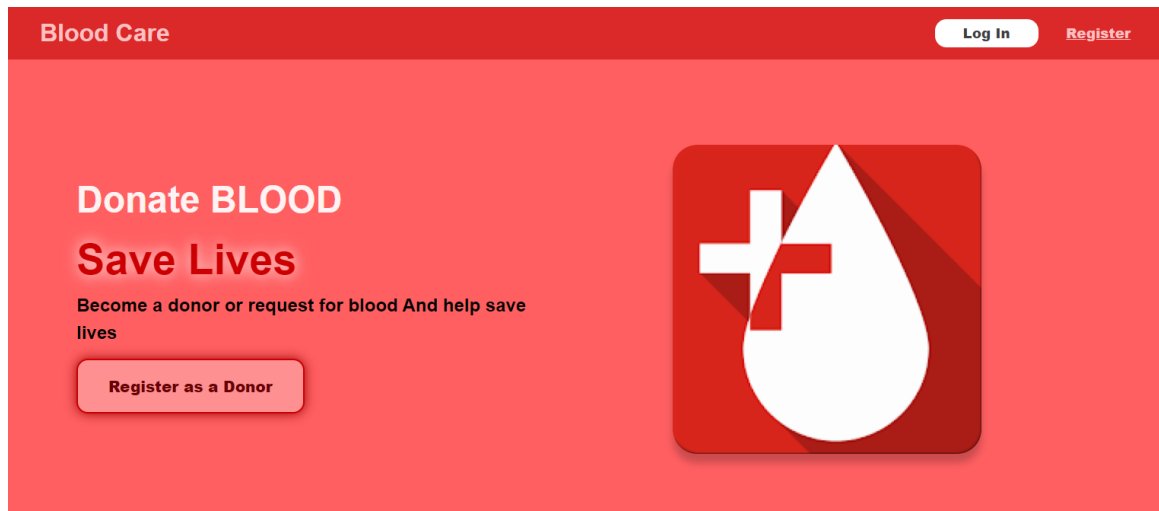


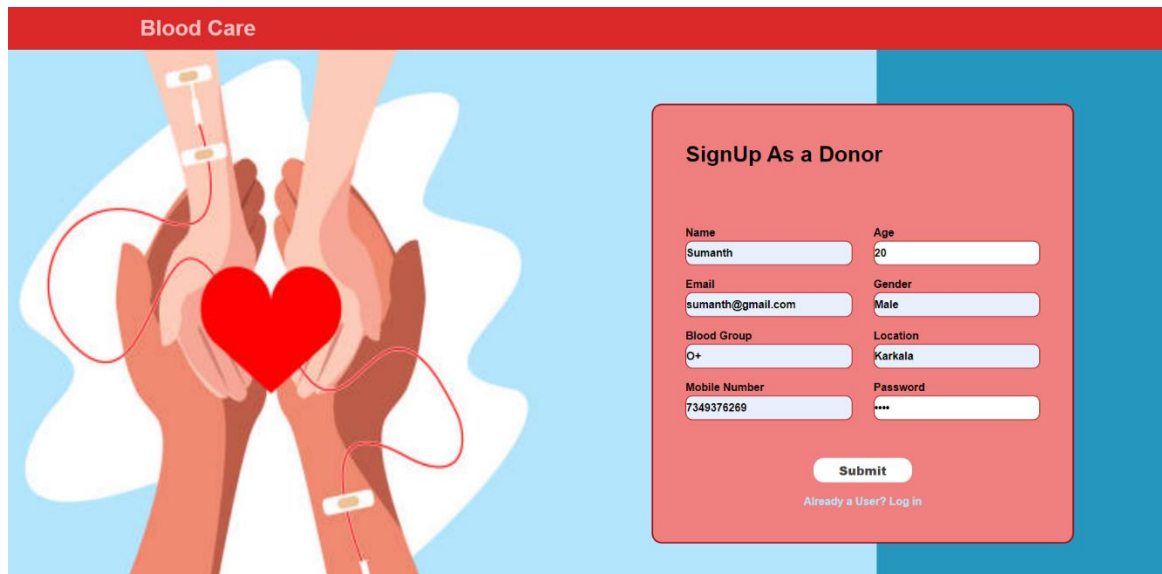
Figure 6.1.1 Snapshot of Home Page

Our Super Heroes

We depend on volunteers! Volunteers make up 96% of our total workforce and carry on our humanitarian work. Blood donation is healthy, our volunteers are available 24/7 to help and donate blood. Blood banks store blood bags but our volunteers are there with you in an emergency for a blood transfusion real time.

1	hema	b
2	Hegde Sumanth Shyam	O+
3	Pushpa	O+
4	ruchitha	a+
5	Sanketh	O+

Figure 6.1.2 Snapshot of Donors registered recently



The image shows a web application interface for blood donation management. The header is a red bar with the text "Blood Care" in white. The background features an illustration of four hands of different skin tones holding a red heart, with medical tubes and bandages on the wrists. On the right side, there is a red rectangular box titled "SignUp As a Donor". Inside this box, there are two columns of input fields: "Name" (Sumanth), "Age" (20), "Email" (sumanth@gmail.com), "Gender" (Male), "Blood Group" (O+), "Location" (Karkala), "Mobile Number" (7349376269), and "Password" (masked with four asterisks). Below these fields is a "Submit" button and a link that says "Already a User? Log in".

Blood Care

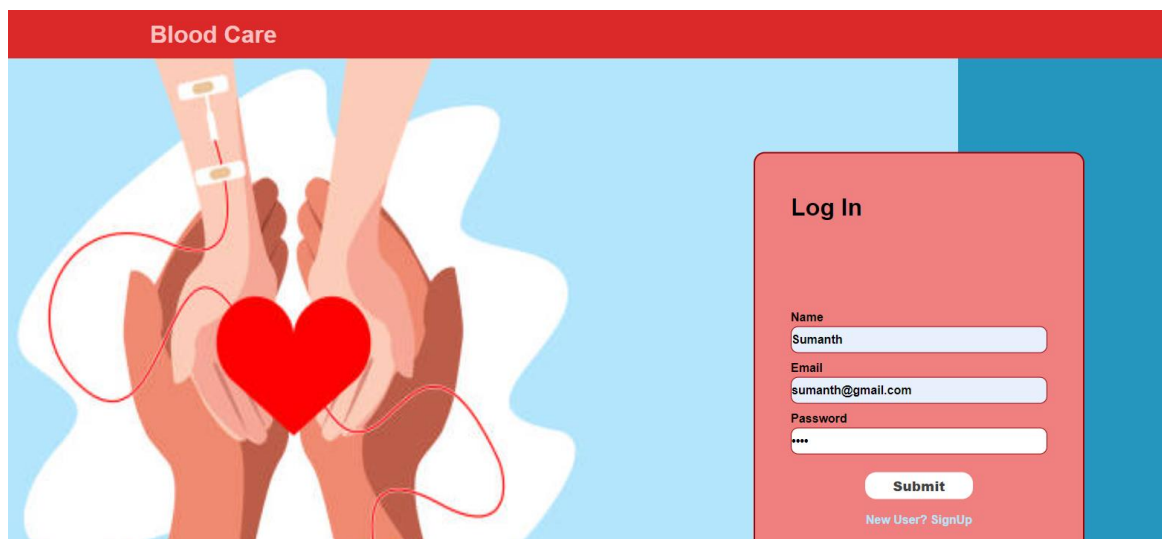
SignUp As a Donor

Name: Sumanth, Age: 20, Email: sumanth@gmail.com, Gender: Male, Blood Group: O+, Location: Karkala, Mobile Number: 7349376269, Password: ****

Submit

Already a User? Log in

Figure 6.1.3 Snapshot of user/donor signup page



The image shows the same web application interface as Figure 6.1.3, but with the "Log In" form displayed on the right. The header and background illustration remain the same. The red box on the right is titled "Log In" and contains three input fields: "Name" (Sumanth), "Email" (sumanth@gmail.com), and "Password" (masked with four asterisks). Below these fields is a "Submit" button and a link that says "New User? SignUp".

Blood Care

Log In

Name: Sumanth, Email: sumanth@gmail.com, Password: ****

Submit

New User? SignUp

Figure 6.1.4 Snapshot of user/donor login page

Blood Care[Requests](#)[Logout](#)


Welcome Sumanth

Your Data:-[Edit Information ?](#)

Name :	Sumanth
Email :	sumanth@gmail.com
Blood Group :	O+
Mobile Number :	7349376269
Age :	20
Gender :	Male
Location :	Karkala

Change Password

Unregister




donate
BLOOD
save life

Figure 6.1.5 Snapshot of user/donor dashboard

Blood Care

[DashBoard](#)/[Edit Password](#)

**Share life
Give blood**



Email*	sumanth@gmail.com
Age	23
Gender	Male
Location	Karkala

Change

Figure 6.1.6 Snapshot of user/donor edit info page

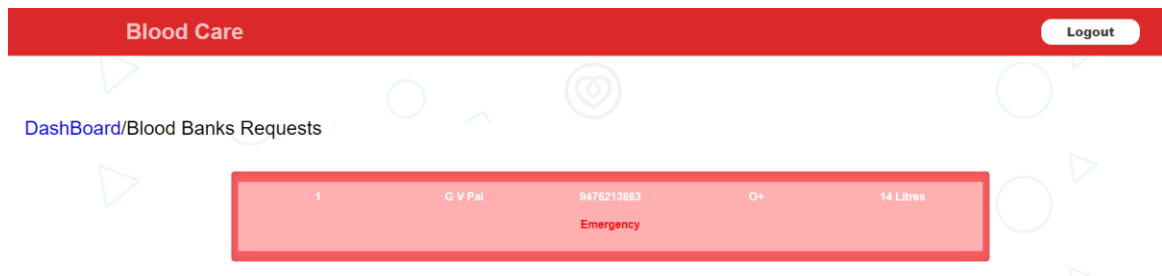


Figure 6.1.7 Snapshot of user/donor request notification page

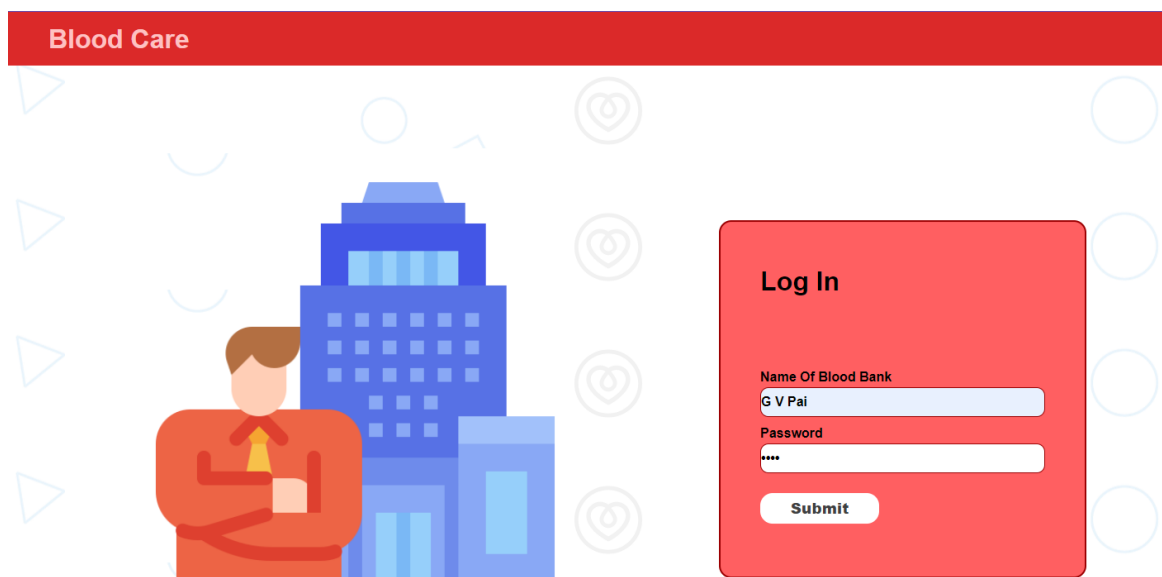


Figure 6.1.8 Snapshot of blood bank login page

Blood Care			Request Blood	Logout
Welcome G V Pai				
hema	b	kaiga	View More	
Hegde Sumanth Shyam	O+		View More	
Pushpa	O+		View More	
ruchitha	a+	kundapura	View More	
Sanketh	O+	Karkala	View More	
Sumanth	O+	Karkala	View More	
Sandhya1	O+		View More	
Sandhya	O+		View More	

Figure 6.1.9 Snapshot of blood bank dashboard

Blood Care		Logout
DashBoard/Request Blood		
Name G V Pai	Contact 9476213863	
Blood Type O+	Requirements(litres) 14	
Description Emergency		
Update		

Figure 6.1.10 Snapshot of blood bank request blood page

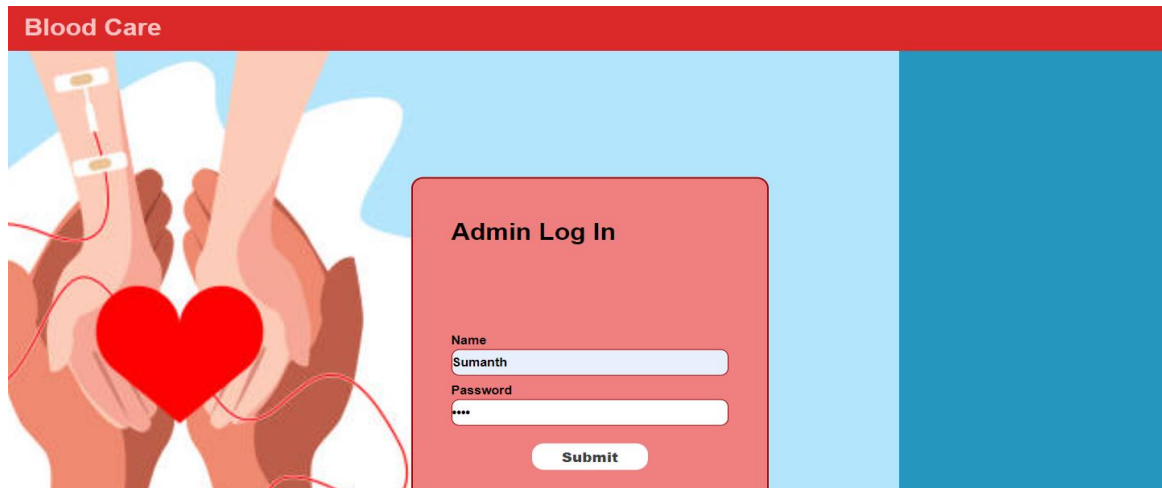


Figure 6.1.11 Snapshot of admin login page



Figure 6.1.12 Snapshot of admin view of user

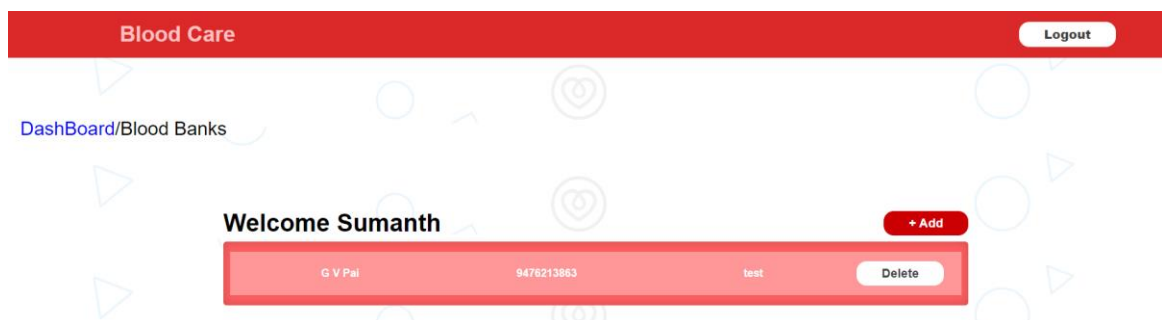


Figure 6.1.13 Snapshot of admin view of blood banks

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

Our proposed project Blood Donation Management helps donors to join the community in the simplest manner which saves time and hassle. Blood Donation Management System is an online registration and connecting technique. There is a database that maintains all the names of donors and blood banks with complete information. Further, users can view various blood requests from blood banks and also contact them if interested. This is not only limited to donors but blood banks also find this system very useful in order to contact donors and conduct blood donation drives. Finally, the admin is the ultimate authority in this system and control, manage all the related data in the system. From this, he can further analyze the data and derive some of the reports and statistics for future enhancements. Through this project, we had a lot to learn and practice. We have gained knowledge of how the client-server architecture actually works and how such systems are developed.

This project provided practical knowledge of not only programming and developing interfaces for users through web applications but also handling data, users, and procedures related to Blood Donation and managing databases. It also provides knowledge about the latest technology used in developing web-enabled applications and database technology that will be in great demand in the future. This will provide better opportunities and guidance in the future for developing projects independently.

7.2 Future Enhancements

In the future enhancement of this project, we would like to include some additional requirements that can be implemented and integrated into the application code. It includes having portals for screening and monitoring donors as well as integrating with other sub-systems in the hospital, making it much more reliable and flexible, and also making much more user interactive.

Chapter 8

REFERENCES

1. Documentation - Node.js at <https://nodejs.org/en/docs/>
2. Node.js Essential Training at LinkedIn Learning
3. Node.js: Web Servers, Tests, Deployment at LinkedIn Learning
4. Nodejs API Complete Guide at Udemy
5. MySQL Documentation at <https://dev.mysql.com/doc/>