Project Document: Blood Management System

Table of Contents

1. Introduction

1.1. Purpose of the Document

The purpose of this document is to provide an overview of the Blood Management System project developed using Core Java, Spring. It outlines the system's requirements, architecture, features, user interface, testing procedures.

1.2. Project Overview

The Blood Management system is a comprehensive application designed to efficiently manage and organize records related to blood donation by various donors. Developed using Java and Spring, this system provides a user-friendly interface that allows users to perform a range of operations related to blood donors and their donations.

1.3. Scope

The scope of the Blood Management Management System project includes the following functionalities:

- Login credential for Admin

- Donor Portal

- Receiver portal

- Report

2. System Requirements

2.1. Functional Requirements

1. Login Credential: The Admin should be able to login with the username and password provided by the management.

2. Donor portal:

    2.1 Add the donor details

    2.2 Delete the donor details

    3.3 Update the donor details

3. Receiver portal:

    3.1 Add the receiver details

    3.2 Delete the receiver details

    3.3 Update the receiver details

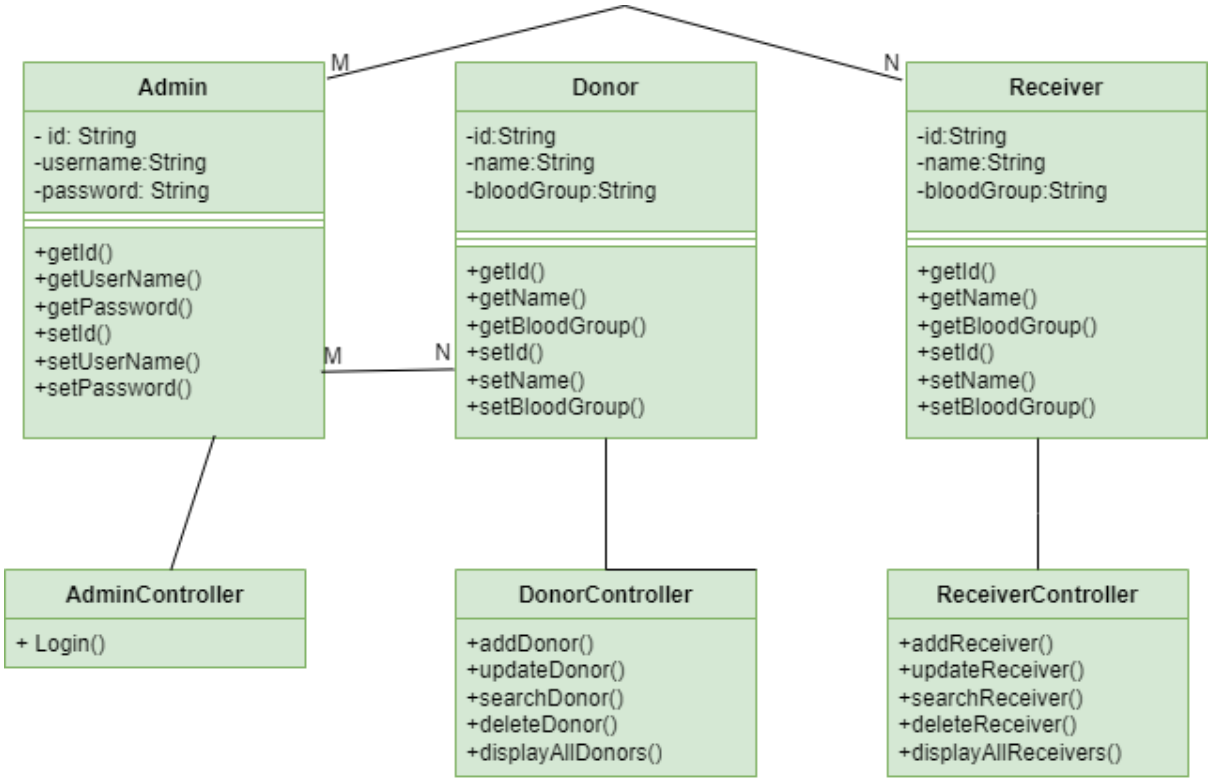
4. Report: Admin should generate the report consisting of total blood available for each blood group.

3. Architecture

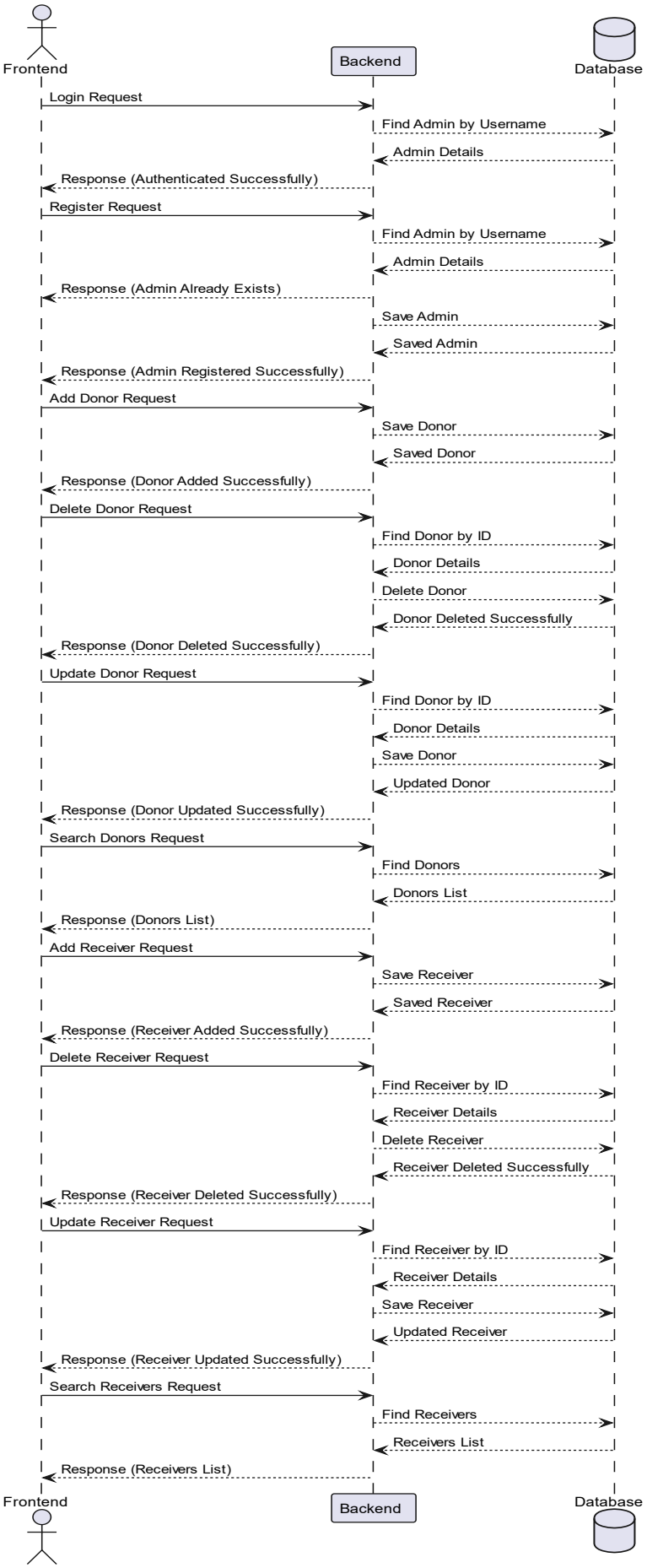
3.1. High-Level Architecture

The Blood Management System follows a client-server architecture where the client is a React.js frontend and the server is built using Java Spring Boot. The client communicates with the server using RESTful APIs. The backend manages donor and receiver records and provides authentication and registration for admins.

3.2. Class Diagram



3.3. Sequence Diagrams


Frontend
Backend
Database
Login Request
Find Admin by Username
Admin Details
Response (Authenticated Successfully)
Register Request
Find Admin by Username
Admin Details
Response (Admin Already Exists)
Save Admin
Saved Admin
Response (Admin Registered Successfully)
Add Donor Request
Save Donor
Saved Donor
Response (Donor Added Successfully)
Delete Donor Request
Find Donor by ID
Donor Details
Delete Donor
Donor Deleted Successfully
Response (Donor Deleted Successfully)
Update Donor Request
Find Donor by ID
Donor Details
Save Donor
Updated Donor
Response (Donor Updated Successfully)
Search Donors Request
Find Donors
Donors List
Response (Donors List)
Add Receiver Request
Save Receiver
Saved Receiver
Response (Receiver Added Successfully)
Delete Receiver Request
Find Receiver by ID
Receiver Details
Delete Receiver
Receiver Deleted Successfully
Response (Receiver Deleted Successfully)
Update Receiver Request
Find Receiver by ID
Receiver Details
Save Receiver
Updated Receiver
Response (Receiver Updated Successfully)
Search Receivers Request
Find Receivers
Receivers List
Response (Receivers List)
Frontend
Backend
Database

4. User Interface

The user interface (UI) of the Blood Management System is designed to provide a user-friendly and intuitive experience for managing blood donor records. The UI is built using React.js, ensuring responsiveness and modern design principles. The layout is designed to be clean, organized, and easy to navigate, facilitating efficient interaction with the application's features.

Design Elements:

Color Palette: The UI employs a balanced color palette that complements the nature of the application. A mix of soothing colors helps convey a sense of professionalism and empathy.

Typography: Clear and readable fonts are chosen for headings, labels, and content to ensure optimal readability.

Navigation: The navigation menu is prominently displayed, allowing users to easily switch between different sections of the application, such as the dashboard, donor management, receiver management, and search features.

Dashboard: The dashboard provides an overview of key metrics and important information, such as the total number of donors, receivers

Donor and Receiver Management: The donor and receiver management sections are designed with organized tables or lists, presenting essential information like name, blood group Action buttons for editing, deleting, and adding new entries are clearly visible.

Search Functionality: The search feature allows users to search for donors based on blood group. The search results are displayed in a well-structured format.

Responsive Design: The UI is responsive and adapts to various screen sizes and devices, ensuring a seamless experience on desktops, tablets, and mobile devices.

5. Technologies Used

The Blood Management System is developed using a combination of programming languages, libraries, frameworks, and tools to create a robust and efficient application. Here is a list of the technologies used:

Frontend:

Programming Language: JavaScript (ES6+)

User Interface Library: React.js

State Management: React Hooks (useState, useEffect, etc.)

Styling: CSS, CSS Modules, or CSS-in-JS libraries (e.g., styled-components),material UI Icon

Routing: react-router-dom

HTTP Requests: axios or fetch API

Package Manager: npm

Code Editor: Visual Studio Code

Backend:

Programming Language: Java

Framework: Spring Boot

RESTful API: Spring Web MVC

Database Interaction: Spring Data MongoDB

Dependency Management: Maven

Integrated Development Environment (IDE): E IntelliJ IDEA

Database:

Database System: MongoDB

Database Management: MongoDB Server (for local development)

API Testing and Debugging:

Postman: Used for testing and validating API endpoints during development and debugging.

Unit Testing:

JUnit, Mockito

Other Tools:

Node.js: Required for building and running React applications.

MongoDB Compass: GUI tool for managing MongoDB databases.

This technology stack ensures efficient development, scalability, and maintainability of the Blood Management System. It combines modern frontend and backend technologies, along with robust database management, to create a comprehensive and functional application.

6. Testing-

Backend achieved a test coverage of approximately 80%. This indicates that a significant portion of the codebase has been tested, contributing to the overall quality of the system.

6.1. Test Cases

During the testing phase, a comprehensive set of test cases were developed to ensure the reliability and correctness of the Blood Management System. The test cases include both positive and negative scenarios to cover a wide range of potential situations.

Authentication and Authorization

Test Case: Verify successful admin login

Input: Valid admin credentials.

Expected Outcome: Admin is authenticated and granted access.

Test Case: Verify unsuccessful admin login.

Input: Invalid admin credentials.

Expected Outcome: Admin authentication fails, access denied.

Donor Management

Test Case: Verify successful addition of a new donor.

Input: Donor data with valid details.

Expected Outcome: Donor is successfully added to the database.

Test Case: Verify deletion of an existing donor.

Input: Valid donor ID.

Expected Outcome: Donor is deleted from the database.

Test Case: Verify deletion of a non-existing donor.

Input: Invalid donor ID.

Expected Outcome: Deletion fails and an appropriate error response is received.

Test Case: Verify successful update of an existing donor's details.

Input: Updated donor details.

Expected Outcome: Donor details are updated in the database.

Test Case: Verify update of details for a non-existing donor.

Input: Updated donor details with invalid donor ID.

Expected Outcome: Update fails and an appropriate error response is received.

Receiver Management

Test Case: Verify successful addition of a new receiver.

Input: Receiver data with valid details.

Expected Outcome: Receiver is successfully added to the database.

Test Case: Verify deletion of an existing receiver.

Input: Valid receiver ID.

Expected Outcome: Receiver is deleted from the database.

Test Case: Verify deletion of a non-existing receiver.

Input: Invalid receiver ID.

Expected Outcome: Deletion fails and an appropriate error response is received.

Test Case: Verify successful update of an existing receiver's details.

Input: Updated receiver details.

Expected Outcome: Receiver details are updated in the database.

Test Case: Verify update of details for a non-existing receiver.

Input: Updated receiver details with invalid receiver ID.

Expected Outcome: Update fails and an appropriate error response is received.

Search Functionality

Test Case: Verify search functionality with valid donor name and blood group.

Input: Donor name and blood group.

Expected Outcome: List of donors matching the criteria is retrieved.

Test Case: Verify search functionality with valid donor name only.

Input: Donor name.

Expected Outcome: List of donors with matching name is retrieved.

Test Case: Verify search functionality with valid blood group only.

Input: Blood group.

Expected Outcome: List of donors with matching blood group is retrieved.

Test Case: Verify search functionality with no search criteria.

Input: No criteria.

Expected Outcome: List of all donors is retrieved.

6.2. Unit Testing

For unit testing, the JUnit framework was used along with mock objects and mockito for creating test cases. The approach followed for unit testing included testing individual methods, components, and classes in isolation. Mock objects were used to simulate interactions with database repositories and to test different scenarios, such as successful and unsuccessful operations.

The sample unit tests provided above demonstrate how different methods are tested using mock objects to simulate database interactions and expected outcomes.

7. Conclusion

the Blood Management System is a successful integration of React.js and Java Spring, providing an efficient and user-friendly platform for blood donation management. With a well-designed interface, seamless frontend-backend communication, and thorough testing, the system ensures reliability and usability. The project demonstrates effective technology utilization and best practices for building a practical and valuable solution.

8. References

1. React.js Documentation: https://reactjs.org/docs/getting-started.html
2. Spring Framework Documentation: https://spring.io/guides
3. MongoDB Documentation: https://docs.mongodb.com/
4. Axios Documentation: https://axios-http.com/docs/intro
5. Postman Documentation: https://learning.postman.com/docs/getting-started/introduction/
6. Java Documentation: https://docs.oracle.com/en/java/
7. JUnit Documentation: https://junit.org/junit5/docs/current/user-guide/
8. Mockito Documentation: https://site.mockito.org/
9. BCrypt Documentation: https://www.mindrot.org/projects/jBCrypt/
10. Material-UI Documentation: https://mui.com/getting-started/installation/
11. Spring Boot Documentation: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

These resources provided valuable guidance and information throughout the development of the Blood Management System.