Tampere University of Applied Sciences

Digitalized Blood Bank

Designing and implementing a web-based application.

Md Forkan Hossain

BACHELOR'S THESIS
May 2024

Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Software Engineering.

HOSSAIN, MD FORKAN:
Digitalized Blood Bank
Designing and Implementing a Web-based Application

Bachelor's thesis 60 pages
May 2024

Digitalized Blood Bank is a virtual blood donation and receiving center that manages the storage of various blood types, donors, and recipient information, as well as data about blood testing, classification, and transfusion. This system helps to address issues that may arise during the blood donation process and ensures that blood is safely transferred to the blood bank, where it can be used for future transfusions.

The blood bank management system's goals are to keep track of blood donors, recipients, donation programs, and bank stocks up-to-date while streamlining and automating the process of finding blood in an emergency.

The purpose of this thesis was to develop an application that provides profile management for donors, recipients, and admins, an authentication procedure, and a user registration and login system. It was developed using React for the frontend interface, NodeJS for server-side logic, MongoDB for database management, and Express for routing and middleware. The technology, framework, system design, and implementation process of the application are discussed in this thesis.

Keywords: blood bank management system, react, Javascript, nodejs, mongodb

CONTENTS

Abbreviations and Terms

API	Application Programming Interface.
Backend	Data access layer of software.
CSS	Cascading Style Sheets.
Database	An organized collection of structured information.
DBBMS	Digitalized Blood Bank Management System.
DOM	Document Object Model.
Frontend	Graphical user interface of a website.
GUI	Graphical User Interface.
HTTP	Hypertext Transfer Protocol.
JavaScript	A web-based programming language.
JSON	JavaScript object notation.
JWT	JSON Web Token.
MERN	MongoDB, Express, React, Node.
MVC	Model View Controller.
ODM	Object Document Mapping.
TAMK	Tampere University of Applied Science.
URL	Uniform Resource Locator.
UI	User Interface.
VDOM	Virtual DOM
VDOM	Virtual Dom.
WHO	World Health Organization.

1 INTRODUCTION

Blood is very important for every single living thing. In Bangladesh, the safety of blood transfusions is a major public health problem. A sufficient supply of safe blood products for all blood types is vital to maintain public confidence in the nation's healthcare infrastructure. Unfortunately, morbidity and mortality are still rising in Bangladesh because of the distribution of dangerous blood products and the shortage of safe blood products.

The Blood Bank Management System makes it easy for users to book appointments for blood donation by providing essential information such as their blood group, contact details, and availability. This information is then used to coordinate blood donations and match donors with those in need. Moreover, the online platform can be used to efficiently manage the stock of blood and keep a record of blood donation history. By doing so, it can ensure that there is always an adequate supply of blood that is not expired. This can significantly enhance the performance of blood banks and enable them to offer better services to individuals in need.

All things considered, blood banks, donors, and recipients depend heavily on the digitalized Blood Bank Management System, as it helps to simplify blood donation processes and save lives. To request blood for a patient, the receiver or administrator must enter the patient's details, such as their name, blood group, and hospital location. Once entered, the system will then search for available blood in storage. If the blood bag is available in storage, the receiver can collect blood directly from the blood bank. In case a blood bag is not available in storage for a patient, the system will search for eligible donors who can donate blood. The system will consider factors such as the location of the donor, their availability on the requested day, and the compatibility of their blood type with that of the patient. If there are any eligible donors available, the system will send them an SMS or an email, after which the donor can log in to the system and view the full details of the request. If the donor is willing and able to donate blood, they can confirm their availability through the system.

2 BLOOD BANK

A digital blood bank uses technology to enhance blood donation, testing, storage, and distribution processes. Through digital platforms and systems, donors can schedule appointments, complete pre-screening questionnaires, and receive reminders for donation follow-ups. These platforms also facilitate efficient communication between blood banks and hospitals, enabling timely responses to blood requests and optimizing inventory management. Additionally, digitalization enhances the tracking and traceability of donated blood units, ensuring their safe handling and reducing the risk of errors. Overall, digitalized blood banks improve accessibility, efficiency, and transparency in blood transfusion services, ultimately benefiting both donors and recipients in need of life-saving blood products (LifeBank, n.d.).

2.1 Background of Study

In a blood bank known as a blood collection facility, hospitals can keep collected blood bags fresh and in use for blood transfusion services. When blood or blood products are required, a blood transfusion is a medical treatment aimed at preserving the life of the patient (OpenAI, 2024).

2.1.1 Problem Statement

The population growth rate In Bangladesh has increased rapidly. In Bangladesh, the annual demand for blood is estimated at 1,000,000 units, with voluntary donors supplying 32% of this requirement. The remaining 68% is sourced from the friends and relatives of donors. Most blood banks still depend largely on manual for their operations. Managing donor records manually presents challenges, including the risk of losing records due to accidents or natural disasters, as they may not be securely stored. Moreover, errors can arise when staff create multiple records for the same donor (World Health Organization, 2022).

A centralized database of willing donors does not exist. Therefore, looking for blood in an emergency becomes tiresome. The only choice is to manually find

donors, match them, and then contact each one individually. Every bank maintains a unique collection of donor documents. If a donor gives blood to another hospital, they must carry their donation certificate with them in order to view their prior records. Therefore, whenever a donor gives blood in a new place, they are regarded as first-timers.

To address these problems and limits, a web-based system for managing blood banks may be required.

2.1.2 Assumptions and Prerequisites

To use the online blood management system, an internet connection is required. The speed of your internet may affect how quickly you can use the system. If a donor agrees to donate blood and is available, they can confirm their donation through the system. Confirmations can only be made by medical professionals or doctors. The success of the medical operation is crucial to ensuring the overall safety of the process.

Identify some of the hypotheses:

1. There are significant variations between manual and online blood banks in terms of the safety of blood transfusions.
2. When compared to manual-based systems, using online blood bank management systems increases the safety of blood transfusions.

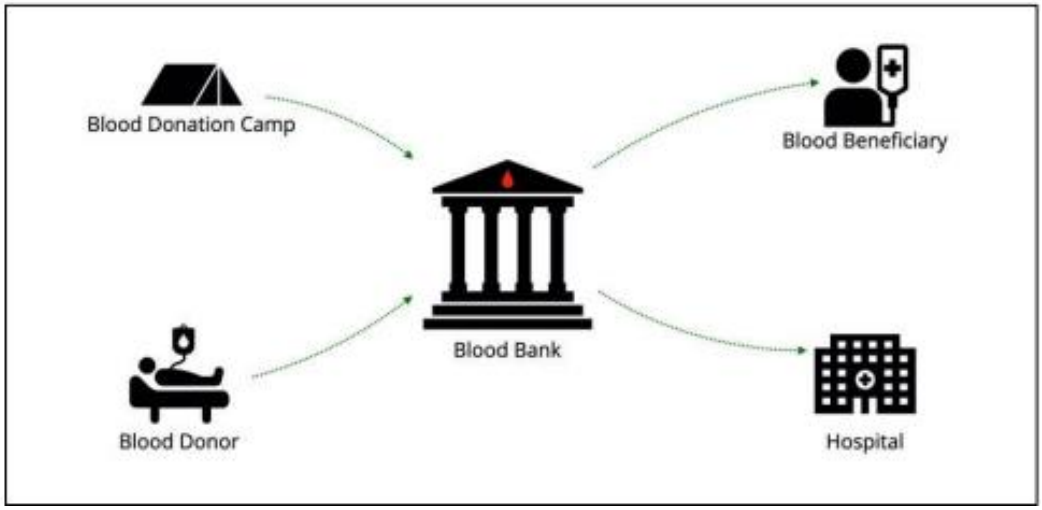

FIGURE 1. Blood Bank Activities.

2.2 Definition of Terms

- **Donors** are individuals who give a part of their body or blood to help those who are ill.
- **Blood bags**: Whole blood and blood components are to be collected, processed, and stored in blood bags. They function as a closed system to lower the danger of contamination and offer aseptic conditions for the separation of blood components.

- **Blood collection:** Blood donation campaigns can run more smoothly if the blood donation campground has enough staff members with the necessary training and equipment, as well as blood donors who volunteer their time. Blood units have been collected by the blood bank. Blood donation camps are typically located at shopping centers, hospitals, workplaces, or universities. The blood unit stock pile-up is the outcome of the blood donation camp. To encourage more donors to participate in the blood donation campaign, the blood bank can advertise it in newspapers, television, and social media.

- **Blood testing and classifying**: Before storing blood, blood testing is an essential procedure. The blood will be tested for several conditions, including HIV, Hepatitis B, and other infectious diseases, using three distinct bags to isolate platelets, red blood cells, and plasma. Blood kept at this temperature is no longer usable. Each bank has its records about donors. A donor's prior records are not accessible if they donate to another hospital unless they carry their donor certificate with them. Therefore, whenever a donor gives blood in a new place, they are regarded as first-timers. The results of the blood test must be categorized and stored based on the kind of blood class.

- **Blood storage:** Blood units need to be stored carefully to prevent bacterial contamination and damage that could cause the units to lose their effectiveness. Red blood cells, platelets, and plasma all have distinct lifespans. Platelets have the shortest lifespan and plasma the longest. The appropriate equipment must also be used to store blood units while they are being moved from one location to another. Blood at room temperature loses its effectiveness after extended periods of time. After being sterilized, blood

units must be disposed of in a location that poses a serious safety risk when the blood cannot be utilized or has expired. Human tissues and blood must be labeled for proper blood storage. The management of blood storage is an essential procedure for blood banks since errors in this process might be caused by human or technical faults. Blood storage management should schedule any red blood cells that have been stored for more than five days to be used as soon as feasible before the blood unit expires, maintain the blood supply, and constantly monitor the blood transfusion process (World Health Organization, 2022).

- **Blood distribution:** A bank for blood is required to manage the proper distribution of blood. Provide only what is necessary to the hospitals, departments, or patients. Blood banks are responsible for overseeing the allocation of blood supplies. Give priority to any emergency patient before moving on to the next priority queue in the list. The attending physician must request a blood unit for the patient, stating the exact kind and quantity of blood required.

2.3 Importance of This Topic

The goal of this digital blood bank management system (BBMS) is to run a blood bank or blood donation facility as efficiently and effectively as possible. The goal of this project is to provide transparent basic blood bank management, simple and easy blood-obtaining procedures, and an easy-to-use, corruption-free method for transferring and distributing blood and blood products to individuals and other franchises on the same network.

2.4 Safe Blood Transfusion

To establish a safe environment for blood transfusion, this project aims to implement transparent policies.

2.4.1 Increase the Accessibility to Public

After implementing this system, access to safe blood will be possible as this system removes all the hurdles currently created by the manual system.

2.4.2 Enhance Operational Capabilities

This system provides management capabilities for blood banks. Basic data related to blood inventory and sample tests are saved and accessible to authorized panels only.

2.4.3 Saving Resources

This project aims to replace the outdated manual system with a modern and well-managed automatic system. By doing so, it will decrease the demand for human and capital resources, resulting in a reduction of resource consumption compared to the current system.

2.4.4 Paper Less Environment

In this modern age, a paperless environment is becoming a requirement for most organizations. As manual systems are being replaced with advanced management systems, access to information has become faster, more accurate, and easier to manage. The implementation of a Computerized Blood Bank Management System (CBBMS) will automate the Blood 4 Bank Process and reduce paperwork to just 5%. This will result in a more efficient and effective blood bank management system.

2.4.5 Discourage Paid Blood Activities

In the Quran, Allah states that "to save one life is to save all of humanity". Donating blood is an act of kindness towards mankind, whereas selling blood is considered a social curse. Blood donation is crucial in saving lives during emergencies, natural disasters, or any other catastrophic event. To discourage blood selling, there is a need for a transparent system. DBBMS is a solution that can help regulate and control the sale of blood.

The following are the main goals of this project:

- The main objective of the program is to connect all blood donors and recipients on a single online platform.
- To enable the potential beneficiaries to look up, match, and request blood from the voluntary donors.

- To notify the blood bank staff when the bloodstock has expired, or the blood amount is below par to facilitate effective donor and bloodstock management.
- Check to find out if the donor has given blood within the last three months.
- To ensure accurate record-keeping regarding the donor and their blood donation activities.
- To offer a combined, compatible donor and bloodstock database.
- Encourage quick searching to locate match blood bags for the appropriate individual.
- Automate processes by providing efficient and continuous software support.

Features:

- Centralized architecture for databases.
- Allows donors or receivers to register, create profiles, and update their information.
- Enables users to schedule, reschedule, or cancel donation appointments online.
- Apply the search facility to locate patients and blood donors using different search criteria.
- Keeps a record of users' past donations or receiving history, including dates, locations, and types of donations.

3 TECHNOLOGY

3.1 MERN Stack

Developers worldwide use the MERN stack, a collection of technologies that makes app development faster. The MERN stack enables developers to create applications using only JavaScript, offering a convenient and seamless development experience. This is possible because all four technologies that make up the stack are based on JS (Duggal, 2021).

This application uses a JavaScript stack that combines Node.js, Express, React.js, and MongoDB. This approach results in a scalable application that is easy to maintain, test, and deploy new features to our users. We can create single-page applications with ease without the need for a server (Geeksforgeeks, 2019).

a. MongoDB: It is a NoSQL database that stores and manages data in JSON format.
b. ExpressJS: This is the framework used for developing the backend of a web application and its corresponding API.
c. NodeJS: A runtime environment for the backend allows for the execution of JavaScript code outside of a web browser.
d. ReactJS: A JavaScript library is a tool that is utilized to construct the user interface.

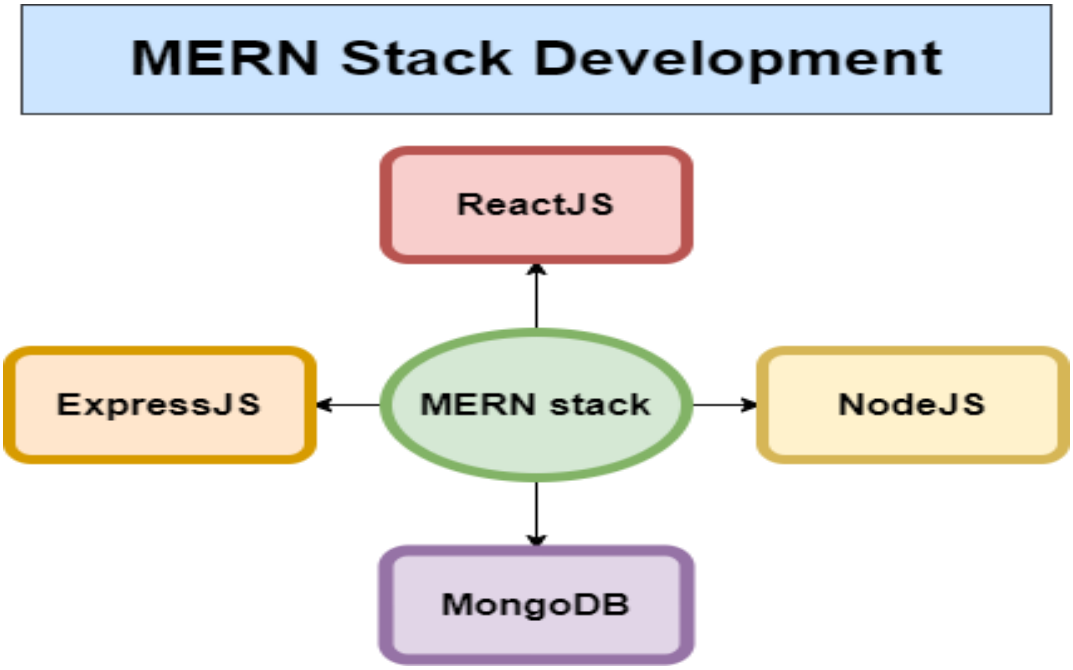

FIGURE 2. MERN stack components.

Here are the advantages of using the MERN stack for an online blood bank management system with details:

- Single Language: Using JavaScript consistently for both frontend and backend is recommended.
- Full Stack JavaScript: Developers can work on the entire technology stack using a single programming language, which can significantly boost efficiency.
- React for Dynamic UI: React enables highly interactive user interfaces.
- Node.js for Scalable Backend: Node.js supports scalable, high-performance server-side applications.
- Express.js for RESTful APIs: Simplifies backend development with robust features for API creation.
- MongoDB for Flexible Data Storage: NoSQL flexibility with dynamic schemas, aligning well with JavaScript.
- Network & The environment: There is a sizable, vibrant group that offers a wealth of resources for growth.
- Cross-Platform Compatibility: Applications can be deployed across various platforms, ensuring wider reach.

3.2 MVC Model

The majority of modern web apps use a variety of designs to produce powerful and efficient online apps. MVC stands for Model, View, and Controller and is one of the most widely used and well-liked designs for architecture. The logical data flow and interconnection of web applications are represented by these three MVC components. This architectural pattern is strictly followed by the MERN stack (Codecademy, n.d.).

Model: The model layer in the MERN stack is controlled by MongoDB. It controls the application's business logic, chooses what data to save, and how to manipulate and store it.

View: ReactJs handles the view layer in the MERN stack, managing the UI of applications and visualizing data.

Controller: Logical data flow in an application is managed by the controller layer. In order to access or modify data, it works with the model layer via orders it receives from the view layer. ExpressJS and NodeJS handle the controller layer in the MERN architecture.

3.3 Frontend

Creating and designing the Graphical User Interface (GUI) that enables users to interact with web applications is referred to as front-end web development, also known as client-side development. Important web tools and technologies including HTML, CSS, and JavaScript are used in this process.

HTML is a markup language that serves as the foundation of any web application by defining its structure. CSS, on the other hand, is a style sheet language that adds visual enhancements and styles to HTML documents. In essence, HTML provides the skeleton of a web page while CSS makes it look visually appealing.

3.3.1 Overview of React

The "R" in the MERN stack is handled by ReactJs, making it an essential part of the stack. User interfaces for single-page applications are frequently created using React, a well-liked open-source JavaScript component package. It was first created by Facebook programmer Jordan Walke, and both Facebook and a global developer community update it on a regular basis. React was first used on Facebook's newsfeed in 2011 (Fullstackopen.com).

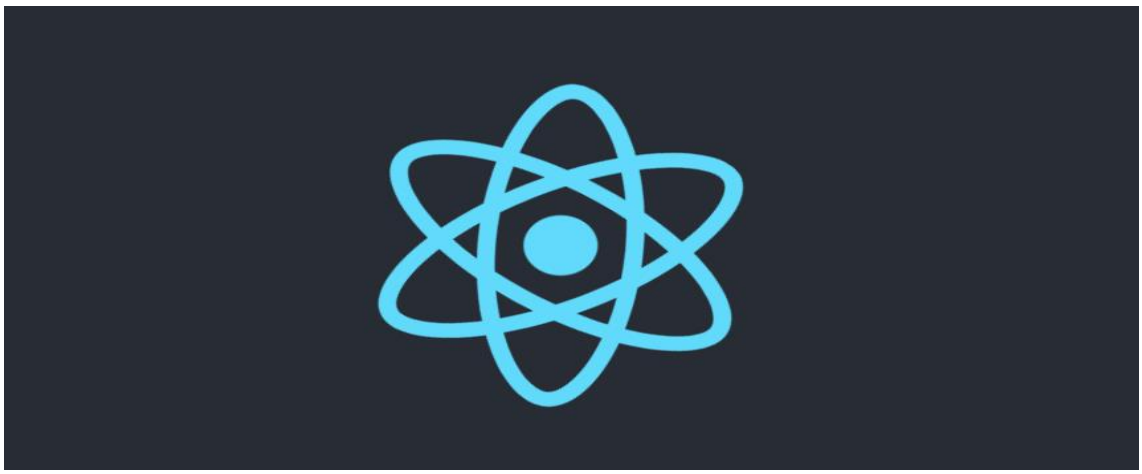

FIGURE 3. React logo.

ReactJS is a useful tool for UI development due to its important features.

1. **Component-Based**

A JavaScript library with a component-based architecture is called ReactJs. This means that the entire application is broken down into individual components that are responsible for specific tasks. Each component has its state and can be re-used as needed ReactJs components have two varieties: parent and child components. While a child component can only send modifications back to the parent component, a parent component can transmit its state to a child component. This helps to ensure that ReactJs just re-renders the precise changes made within a component, rather than the entire page. ReactJs places a lot of emphasis on this, key element of single-page apps (Patel, 2019).

2. **Declarative**

ReactJs manages the process of monitoring and modifying modifications, guaranteeing that developers do not need to be concerned with the management of state or data consequences. Components possess both state and view, which are stored and updated with Virtual DOM. ReactJs effectively and rapidly manipulates the Virtual DOM, implementing modifications to the actual DOM in the browser.

3. **Isomorphic**

ReactJs is a flexible framework that can be employed in web browsers and servers with NodeJS to enhance search engine optimization and develop contemporary mobile applications. It complies to the principle of "learn once, write anywhere," enabling developers to apply the same code across multiple platforms.

3.3.2 JavaScript

JavaScript is an object-oriented, lightweight scripting language that works on multiple platforms. It is mainly designed to add interactivity to modern web pages and applications. Along with HTML and CSS, it is one of the three primary components of front-end technologies. Brendan Eich, an American programmer, developed JavaScript at Netscape in 1995. It was formerly known as "LiveScript" and "Mocha." The third beta version was released in December 1995, at which point it was given its present name (Tutorials Point, 2019).

With JavaScript, programmers may design dynamic, interactive client-side interfaces that can recognize and carry out platform-specific actions depending on the browser and operating system of the user. An object-oriented programming language that allows for built-in objects with inheritance is JavaScript. This animations and handle multimedia, add animations, and give static HTML pages dynamic capabilities. Furthermore, JavaScript offers several third-party tools and APIs to facilitate the creation of dynamic webpages.

3.4 Backend

Back-end web development, sometimes called server-side development, is the process of developing databases and computer programs that provide information and functionality to a web application's client side. In the early days of web development, a front-end may not have been necessary for an application to be classified as a web application if its server-side was operational. Back-end development encompasses a range of technologies, such as programming languages, frameworks, and database management systems.

3.4.1 Overview of Node

Node.js is a widely used free to use, multiple platforms JavaScript runtime framework designed for executing JavaScript code outside of web browsers. Ryan Dahl developed it in 2009 and has since then become an essential tool for server-side development because of its asynchronous, event-driven architecture (Node.js, n.d.).

Node.js is a server-side programming language that enables developers to utilize JavaScript for efficient development and code reusability. The system employs an event-driven, single-threaded framework to effectively manage asynchronous I/O activities, enabling the simultaneous management of many connections without impeding code performance.

The default package manager, npm, simplifies dependency management and enables the integration of third-party modules. Node.js is a dynamic runtime environment that provides developers with connectivity and flexibility across many operating systems. It provides web server functionality, including modules and APIs for designing web servers and developing web-based applications.

Node.js applications are easily horizontally scalable, making it suitable for managing high traffic levels and growing demands. A dynamic developer community actively contributes to the advancement of Node.js, fostering innovation and expansion.

3.4.2 Express Overview

Express.js, often called Express, is a well-known and widely used web application framework designed for Node.js. It offers a powerful range of tools for creating web applications and APIs with Node.js, making it a go-to choose for many developers due to its ease of use, adaptability, and efficiencyExpress.js combines middleware functions into a sequence of operations to manage HTTP requests, including the POST, PUT, GET, and DELETE methods (Pabbly, n.d.).

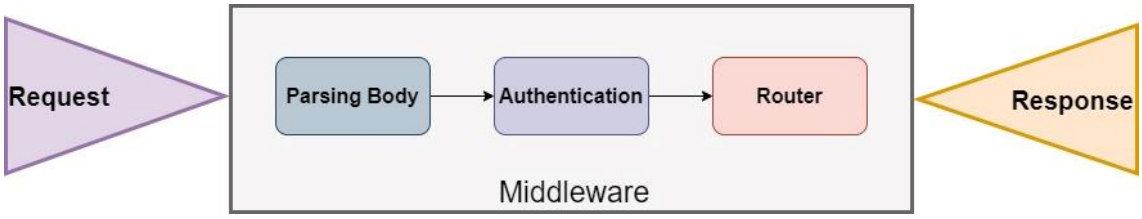

FIGURE 4. The Middleware request-response cycle.

Express is a powerful web application framework that simplifies the management of HTTP requests and responses. It uses middleware functions to access the request object (req), response object (res), and subsequent middleware functions, making it easier to manage incoming requests and answers.
Express also offers a user-friendly route creation technique for various HTTP methods, including POST, GET, PUT, and DELETE. It also provides utility functions for managing HTTP requests and responses, such as res.send(), res.json(), res.render(), and res.redirect().

It supports various templating engines, including EJS, Pug, and Handlebars, and has integrated error handling tools. Express's static file serving feature allows for easy serving of various files. Its modularity allows developers to add more middleware and libraries as needed. Its scalability and performance make it a popular choice for web applications and APIs.

3.5 Overview of MongoDB

MongoDB is a specific kind of database that saves data in a document format. It is often referred to as a non-relational database and can be installed on a local computer or hosted in the cloud. MongoDB was first released on February 11, 2009. It uses BSON as its storage format, which is a binary-encoded serialization of JSON-like documents (Data Management, n.d.).

This method allows for a smooth integration with modern development frameworks and languages. MongoDB stores data into groups, which have similarities to records in relational databases. A group consists of multiple documents. One of MongoDB's key features is its schema-less architecture. This enables developers to store varying data types in the same collection without predefined schemas. This flexibility speeds up development cycles by eliminating the need for extensive schema migrations. Queries in MongoDB are executed using MongoDB Query Language (MQL). It is a powerful and intuitive syntax for interacting with the database (Sazib, 2022).

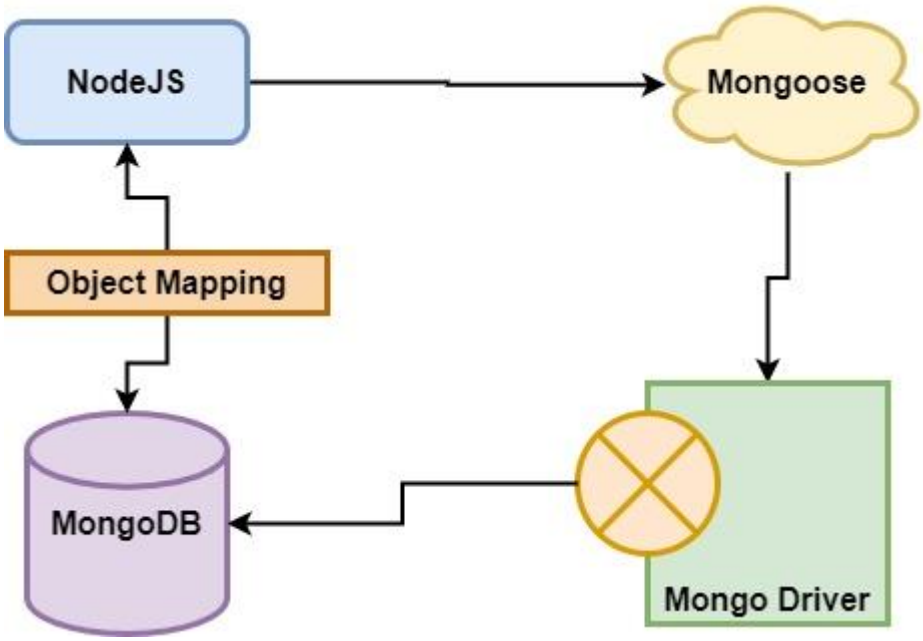

FIGURE 5. Mongoose maps objects between Node and MongoDB.

Because of MongoDB's flexibility and high accessibility features, enterprises may easily grow to meet growing demands while maintaining uninterrupted access to critical data.

The capacity to manage huge amounts of data and enable agile development methodologies has made MongoDB a top choice for many different kinds of applications, including content management systems and real-time analytics platforms.

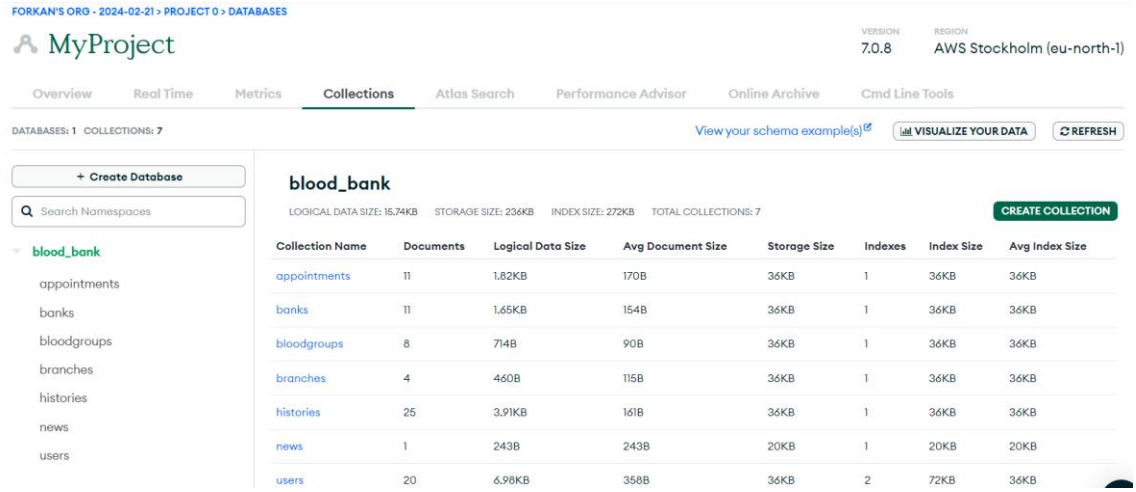

FIGURE 6. MongoDB database for Digitalized Blood Bank.

4 SYSTEM DESIGN

Design patterns are important for properly maintaining and altering web applications across the following stages when changes are required. The Model-View-Controller (MVC) is an operating system design framework used to structure and build user interfaces. The given software application is divided into three interconnected components, which separate the internal representations of information from the methods of presenting or receiving information from the user (Emmanuel, 2023).

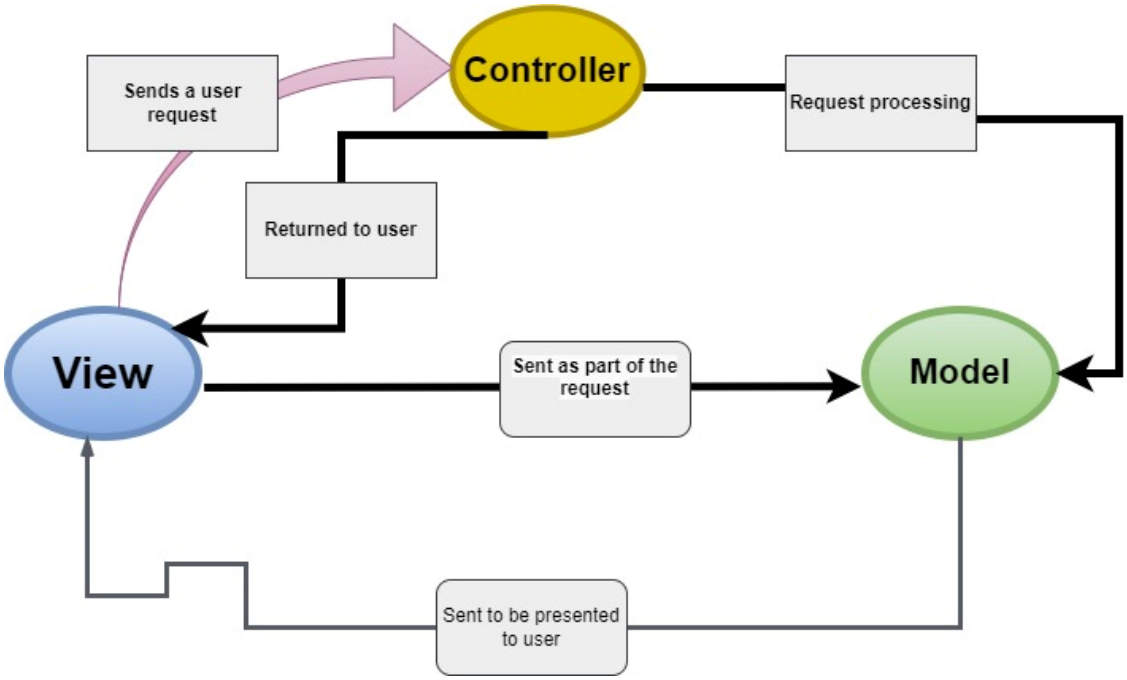

FIGURE 7. System Design Pattern.

The Controller is used as a link between the Model's elements and the view, while the Model represents the entities or classes and the View relates to the web pages or user interface.

4.1 Requirements Analysis

To digitalize a blood bank, requirement analysis plays a crucial role. It involves a systematic process of gathering, documenting, analysing, and prioritizing the needs and expectations of users (OpenAI ChatGPT).

4.1.1 Use Case

The use case defines interactions between users and a system to achieve specific goals or tasks. In a digitalized blood bank application, use cases outline various scenarios including donor registration, blood donation, inventory management, blood request handling, reporting, and more (Scribd, n.d.).

The use cases for a digitalized blood bank application are given below:

1. A new user has registered on the platform and provided their personal information as well as their blood type.
2. The system verifies eligibility criteria such as age and health history.
3. Once verified, the user's information is added to the database.
4. A registered schedules a donation or receives a blood appointment through the app.
5. At the blood bank, staff members record the donation details including blood type, location, and date.
6. The donor's donation and receiving history are updated, and the blood is added to the inventory.
7. Staff members monitor the blood inventory in real time through the application dashboard.
8. Donors receive notifications about urgent needs and are encouraged to donate if eligible.
9. The system maintains accurate records of donations, screenings, and distributions.

4.1.2 Functional Requirements

The functional requirements for a digitalized blood bank application are given below (Scribd, n.d.):

- Users can register and log in securely.
- Different levels of access are provided based on user roles (admin, user).
- Ability to register new donors and receivers with their personal information.
- Track donor eligibility criteria such as age, weight, health history, and donation frequency.
- Maintain a history of donations and receiving for each user.
- Record of incoming blood donations, including the blood type, quantity, and expiration date.
- Notify staff when inventory levels are low or when blood is nearing expiration.
- Enable users to schedule donations and receive appointments online.
- Generate reports on blood inventory levels, donations, and distributions.

4.1.3 Non-Functional Requirements

Non-functional requirements deal with topics such as performance, reliability, security, usability, and scalability (OpenAI, 2024).

- The system is fast, responding quickly to user actions.
- It handles many users at once without slowing down.
- User data is safe and authorized people are able to access sensitive information.
- Usability:
- easy to use, even for people who aren't tech-savvy.
- If more people start using the system, it will still work well.
- The code is well organized and easy to understand, update, and fix problems.

4.2 E-R diagram

Creating an Entity-Relationship (E-R) diagram for a digitalized blood bank involves identifying the main entities involved, their attributes, and the relationships between them (GeeksforGeeks, 2020).

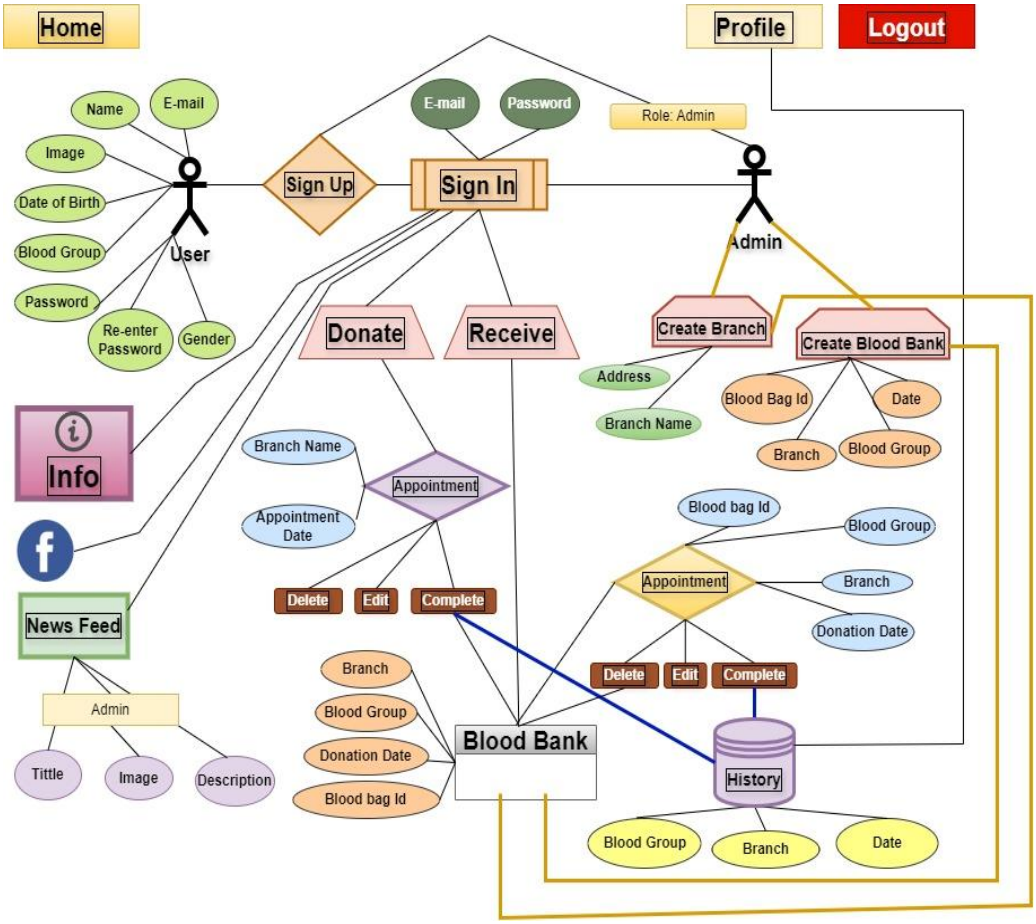

FIGURE 8. E-R diagram design.

4.3 Architecture Design

The architecture for a digitalized blood bank website entails a comprehensive approach spanning frontend, backend, data management, integration, security, scalability, monitoring, and disaster recovery. On the frontend, a user-friendly interface powered by frameworks like React.js facilitates seamless interaction with features such as donor search, appointment scheduling, and inventory management. The backend, comprising a server and database layer, handles client requests and stores crucial data including user profiles, donor information, blood inventory, and transaction records. Integration with external APIs enables additional functionalities like location-based donor search and notification services.
This project's application has a three-tier design with presentation, logic, and data levels. The presentation layer manages the user interface, whereas the logic layer enables communication between the database and presentation levels. The database layer is where the data is stored (Matina, 2023).

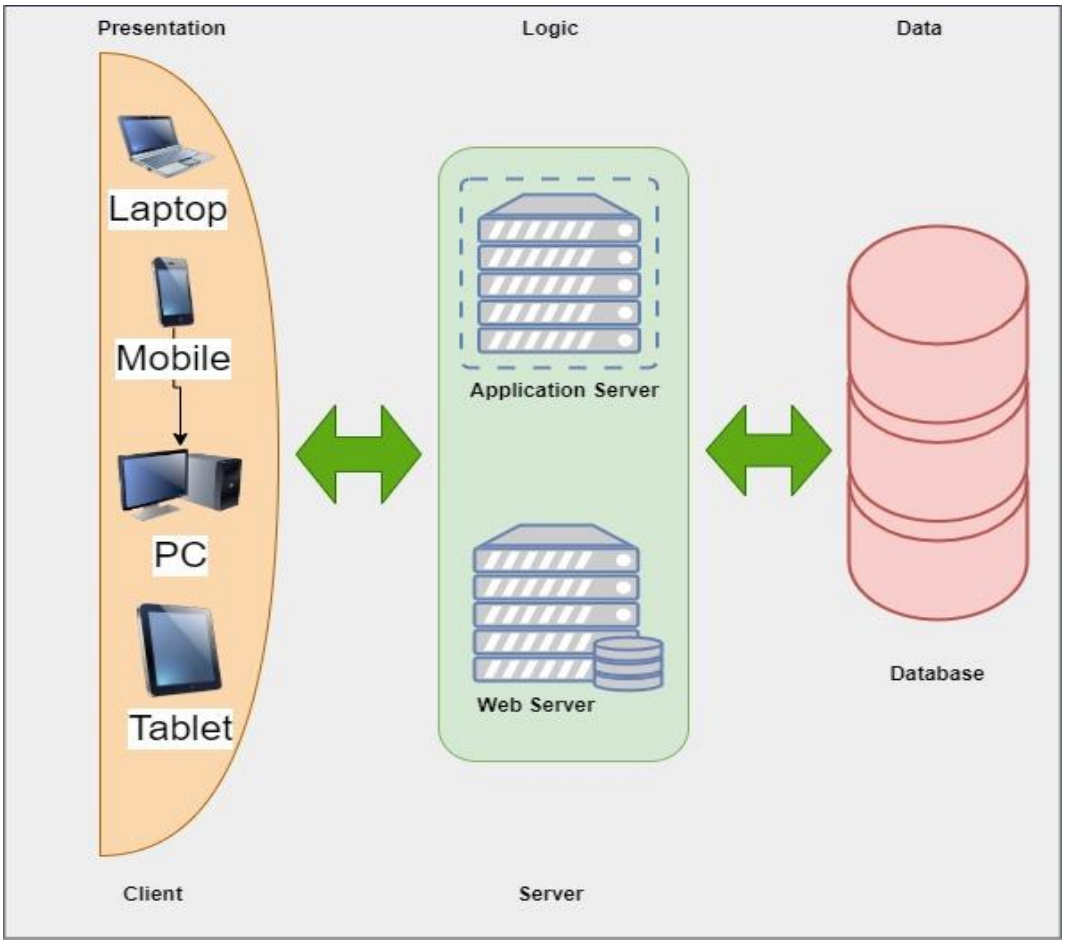

FIGURE 9. Three-tire System Architecture.

4.4 Database Design

The database plays a crucial role in data storage. It provides the structure and organization for the digitalized blood bank platform, allowing for the management of donors, blood samples, recipients, and transactions. Each of these components possesses specific attributes that capture important data points such as donor details, blood types, sample volumes, and transaction records. Maintaining an effective, precise, and secure database design is essential for the day-to-day operations of an online blood bank platform. As such, the database design must be carefully constructed to ensure proper functionality (Microsoft, 2022).

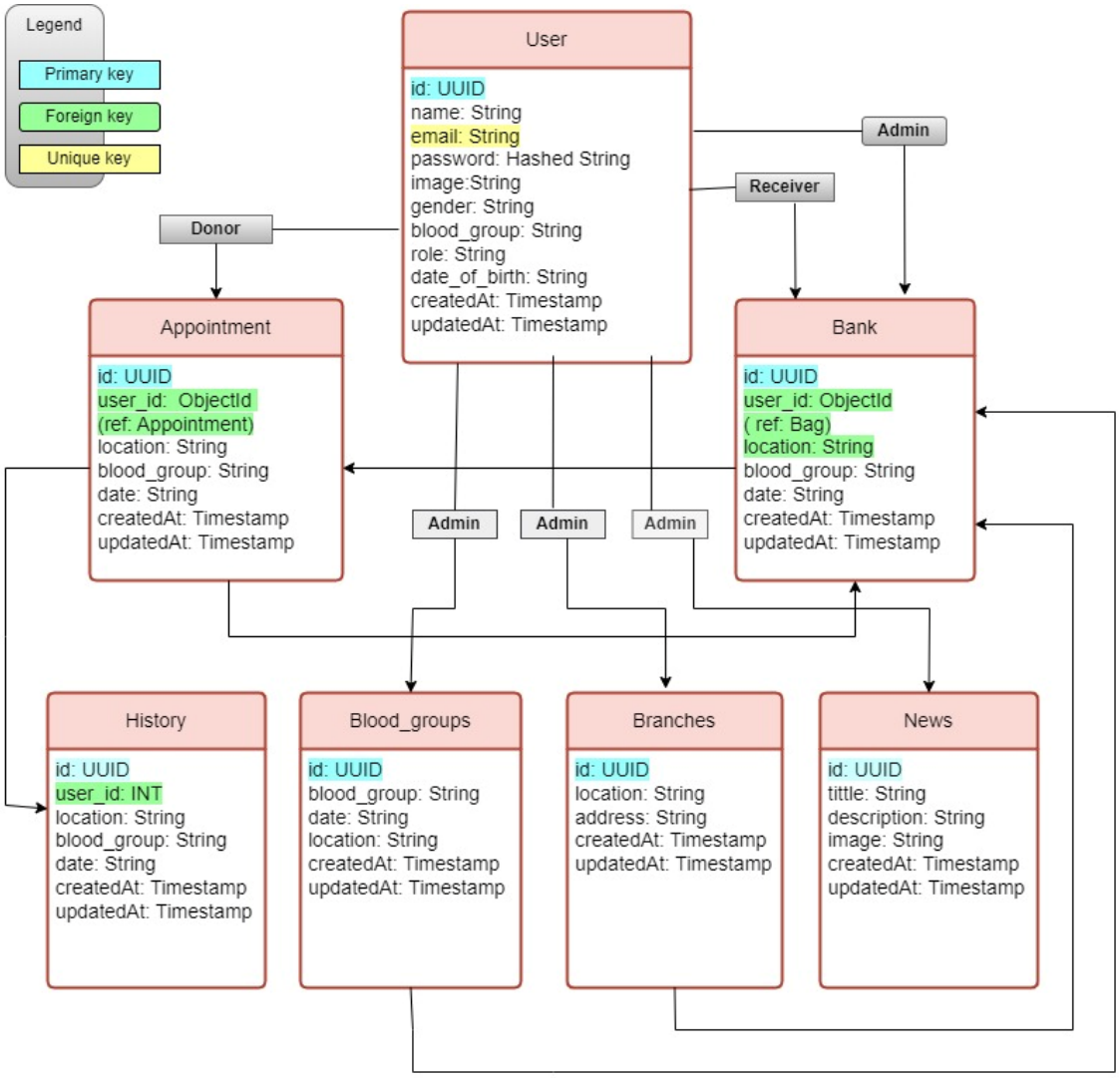

FIGURE 10. Database Design.

5 PROJECT IMPLEMENTATION

The three-step process for implementing the digitalized blood bank application was completed. Work started from the back end. The development of the front end came next, and the application was then deployed. In this part, each of the processes has been explained in detail.

5.1 Environment Setup

Before building the application with the MERN stack, it has been important to configure the programming environment to ensure the smooth integration of technologies and tools while improving the development workflow.

5.1.1 Version Control

An essential component of software development is version control. Developers can examine commit histories, access, and roll back earlier iterations of the code, and determine which particular files and lines of code were changed and by whom. When working in a team, using a version control system is very important for managing multiple development branches and ensuring that each member's project is in sync. The most widely used version control system (VCS) worldwide is called Git. As such, it was used for version control when this program was being developed.

5.1.2 Visual Studio Code

Windows, Linux, and macOS can all run Visual Studio Code (VS Code), a free and open-source text editor. Its numerous features have helped it become more well-liked in the development community since Microsoft published it in 2015. Integrated support for Node.js and JavaScript programming, multilingual support, IntelliSense, syntax highlighting, and an integrated terminal for simpler development are a few of its primary features. Thus, the text editor that was used for developing this application was VS Code.

5.1.3 Browser

It's essential to test web applications in web browsers when developing them. Although there are other browsers available, Google Chrome and Firefox Devel-

oper Edition are frequently used for development. Strong developer tools including network data, performance analysis, and a debugger are included in both Google Chrome and Firefox Developer Edition. React Developer Tools is an extension that helps debug React apps. React Developer Tools are very helpful in debugging React application slowdowns because they show component rendering timings and the reasons for component re-rendering.

5.2 Application Setup

I started by establishing the application's fundamental configuration after configuring the development environment. I configured the application's frontend after establishing the backend. Next, I created the link between the application's frontend and backend.

5.2.1 Frontend Implementation

The user interface that ends users interact with is referred to as the frontend. React.js was used for developing the application's client side and React Router Dom was used to design it. Only the frontend, which performs all the operations specified based on our view pages, allows the client to interact with the user interface. From a design view, there are eight separate pages in total (Signup, Signin, Home, Profile, Info, News, Receive, Donate).

5.2.2 Installing React Application

For the initial setup of the React project command "**npx create-react-app frontend**" can be used. In the "**frontend**" folder, this code makes a React app. So, then used the command "cd frontend" to go to the program directory. Then start the application with the command "**npm start**", which shows the React app's UI with its name.

When "**http://localhost:3000**" is typed into a web browser, the app's layout and functionality are visible right away. Packages for React.js, such as react-router-dom and react-icons, allowed for the integration of extra functionality into the project. After React is installed and configured, can focus on the app's looks and functionality.

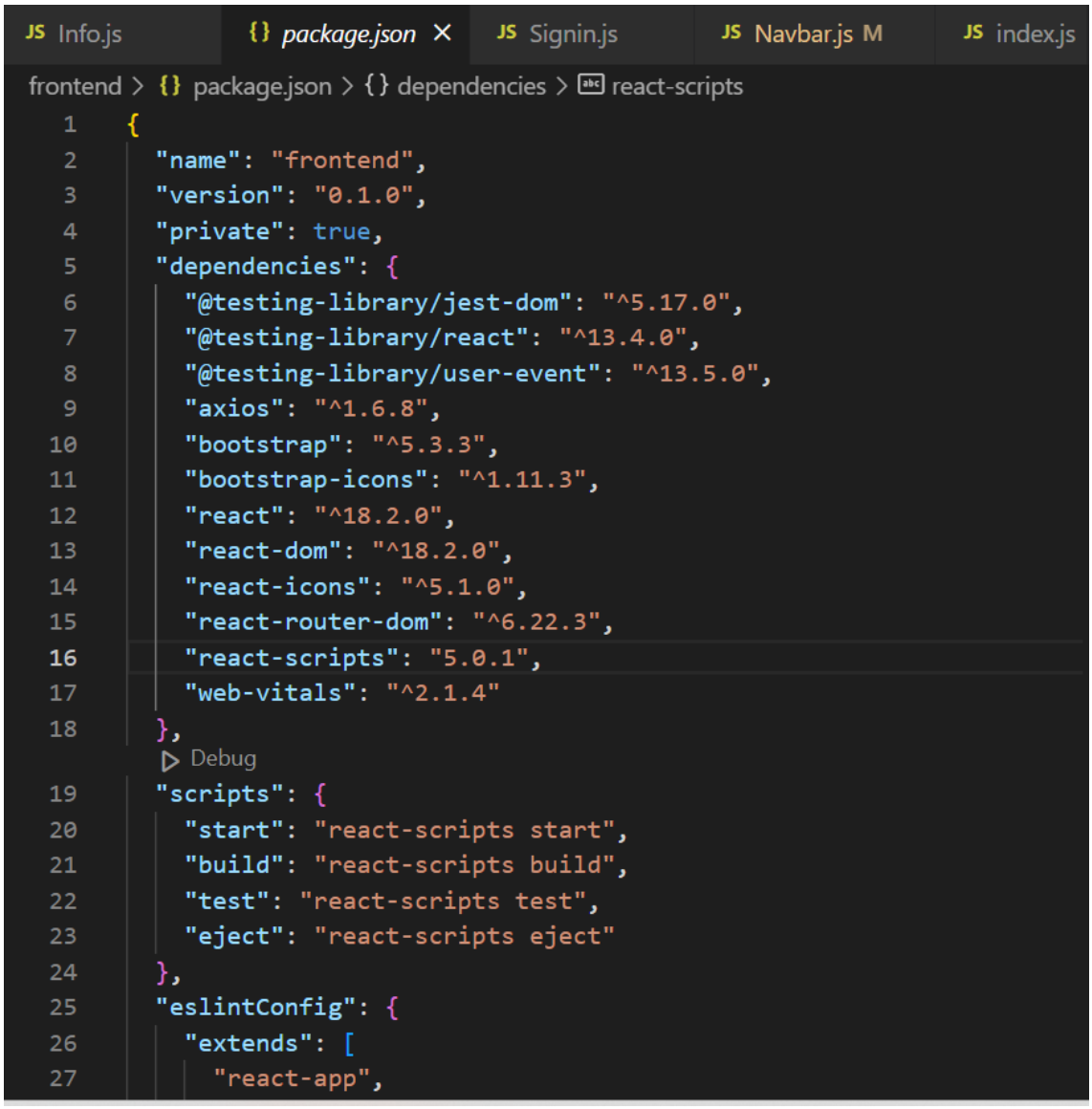

FIGURE 11. Front end's package.json

5.2.3 Folder Structure

Making a clear and well-organized project structure is essential for developers to effectively manage and implement logic across different files and folders.

FIGURE 12. The frontend react folder's structure.

Node_modules: contains the necessary dependencies for creating React applications, along with the "public" folder holding all of the static files. The "src" (source) directory is an essential component of the project structure in a React

application. The source code is kept in the directory together with the CSS and JavaScript files. For instance, the main entry file, often known as "**index.js**" or "App.js," which acts as the beginning point for a React application, can be found in the "**src**" directory. Additionally, "**src**" folders are frequently used to organize additional parts, utility functions, and stylesheets. In a React project, the "**src**" directory is crucial for keeping the codebase organized and tidy. In addition, the package.json file is created. It contains a list of dependencies and their corresponding versions, frontend metadata, and configured scripts.

Two files are in the **components** folder. First, as our application uses React navigation, we have the "**ContextAPI.js**" file, which manages to deliver data from the child component to the parent and vice versa. This eliminates the need to provide a prop through every application tree level, which will be time-consuming. The second file is called **Navbar.js**, and it contains the upper bar navigation component that allows the user to access all the view pages by navigating across our application hierarchy. The client can access a total of 8 public pages in the "Screens" directory. beginning with the user's initial visit to the website, which is the "Home" page. Before being able to see the main view pages, the purpose of "Signup" and "Signin" is to authenticate the user.

Profile screen, where the recipient or donor can view all of their data as well as their history of appointments, donations, and receiving.

Info page, where users can read all necessary information about blood banks and blood donations.

Receive and Donation pages, where users can find all the blood bank branches and make an appointment to donate blood or receive blood according to their wish using a search filter.

5.2.4 Data Fetching and API Integration

In MERN-based full-stack application development, API integration and data fetching refer to the procedure of establishing a connection between the frontend and backend to access and modify data from the database or other sources (FreeCodeCamp.org, 2023).

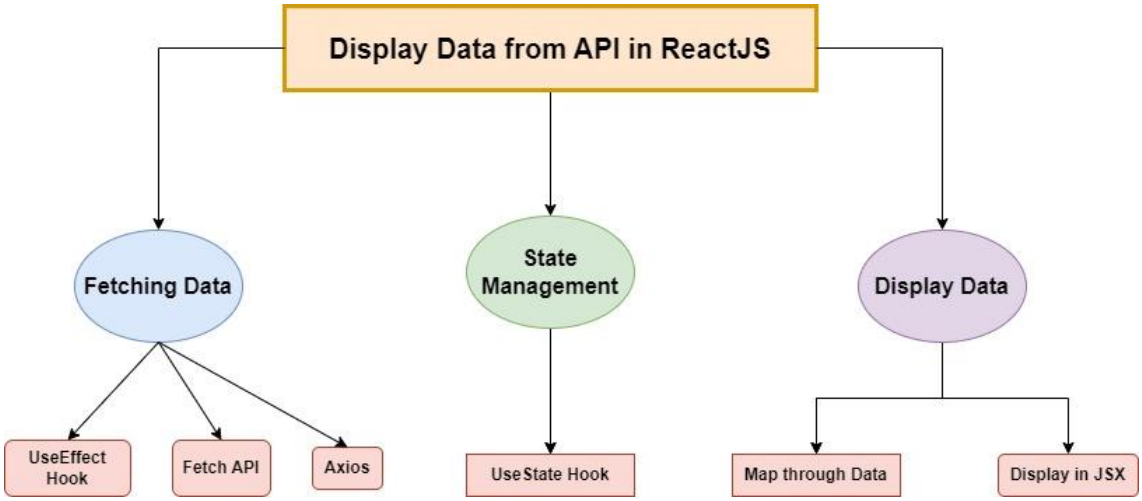

FIGURE 13. API Workflow.

Below is an overview of the API process:

An application programming interface (API) is a link that enables communication between two software programs. In a MERN stack, React is usually used to build the frontend, and JavaScript libraries or the built-in fetch function are used to fetch data. Using node.js and express.js, the backend works as a middleman between the frontend and the data source, accepting requests, analyzing them, and providing data in response.

The backend controller function executes the request, which can involve connecting with external APIs or querying a database. API routes handle kinds of requests. The JSON response to is sent back to the frontend by the backend so that it can be displayed and processed. By submitting POST, PUT, or DELETE requests to the relevant API endpoints, APIs can also manage data updates, such as adding, editing, or removing records.

5.2.5 JWT for Authentication and Authorization

JSON Web Token is a powerful tool for both authentication and authorization in web applications. Once the user's identity has been established, access to a resource is either granted or denied.

Although there are various methods for implementing authorization and authentication for online applications, using sessions to manage user status on both the client and server is the most tried-and-true method. However, a new idea for a stateless method that stores user status has been realized with a technology called JSON Web Token (JWT), which was later disclosed.

The general flow of the mechanism is explained in the following figure:

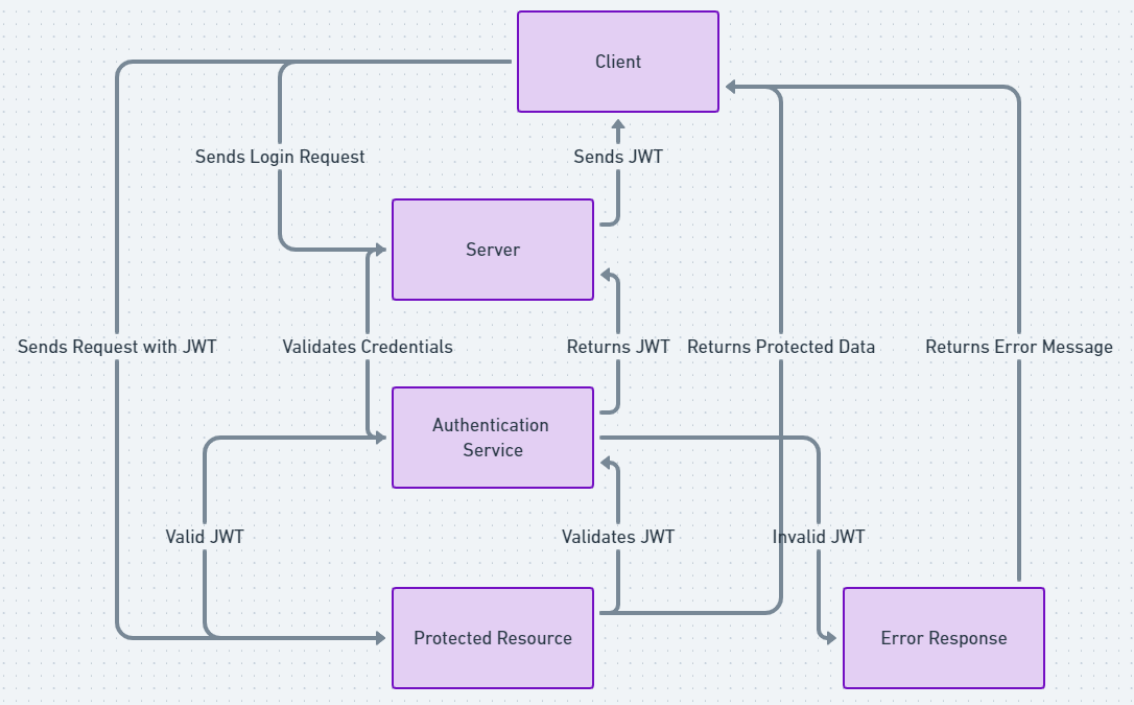

FIGURE 14. JWT Flow Diagram.

First, as shown in Figure 14, the server generates a JWT token upon user login, which is signed using a secret key and the user information supplied. After that, the client will receive the token, which they can store in a cookie or local storage, thereby assuming responsibility for user state maintenance. Upon successful login, the token has to be included to the Authorization request header with the type "Authorization: Bearer" in all requests made to protected endpoints. The

server will validate the token to see if it is valid when processing requests to protected endpoints. The user gets granted access if such is the case. If not, an error happens.

5.2.6 React Router Dom

React Router Dom is a popular widely used module for handling routing in React applications. It provides the capability to specify routes and their accompanying components, facilitating navigating between various sections of the application without the need to reload the page.

```
frontend > src > JS App.js > ...
1   import "../node_modules/bootstrap/dist/css/bootstrap.min.css";
2   import 'bootstrap/dist/css/bootstrap.min.css';
3
4   import { BrowserRouter, Routes, Route } from 'react-router-dom';
5   import './App.css';
6   import Home from './screens/Home';
7   import Donate from './screens/Donate';
8   import Signin from './screens/Signin';
9   import Signup from './screens/Signup';
10  import Receive from './screens/Receive';
11  import ContextAPI from "./components/ContextAPI";
12  import Profile from "./screens/Profile";
13  import Navbar from "./components/Navbar";
14  import Info from "./screens/Info";
15
16  function App() {
17    return (
18      <BrowserRouter>
19        <ContextAPI>
20          <Navbar />
21          <Routes>
22            <Route path="/" element={<Home />} />
23            <Route path="/donate" element={<Donate />} />
24            <Route path="/signin" element={<Signin />} />
25            <Route path="/signup" element={<Signup />} />
26            <Route path="/receive" element={<Receive />} />
27            <Route path="/profile" element={<Profile />} />
28            <Route path="/info" element={<Info />} />
29          </Routes>
30        </ContextAPI>
31      </BrowserRouter>
32    );
33  }
34
35  export default App;
```

FIGURE 15. Using React Router Dom.

The react-router-dom library is used to import the BrowserRouter, Routes, and Route components. The following views are included in the application: home, Signup, Login, Info, Donate, and Receive. To enable client-side routing, the application component is enclosed within the Router component, and the Route component is used to create each route. The landing page's URL route is set to ("/").

5.2.7 Axios

Axios is a JavaScript library used for sending HTTP requests through an online application to communicate with a server or retrieve data from an API. It offers features such as request handling, response management, error handling, interceptors, and automatic data conversion. Axios is installed by **npm install axios** command and the Axios library has to be imported as shown in Figure 21 (Axios, 2023).

FIGURE 16. Methods of using Axios.

5.2.8 React Icon

A library called React Icons offers a variety of well-known icon libraries as React components. It allows easily included icons in React applications without the need for additional setup or dependencies. React Icon is installed by npm install react-icons.

To display the Facebook icon inside a link, Figure 17 imports the FaFacebook library from react-icons/fa.

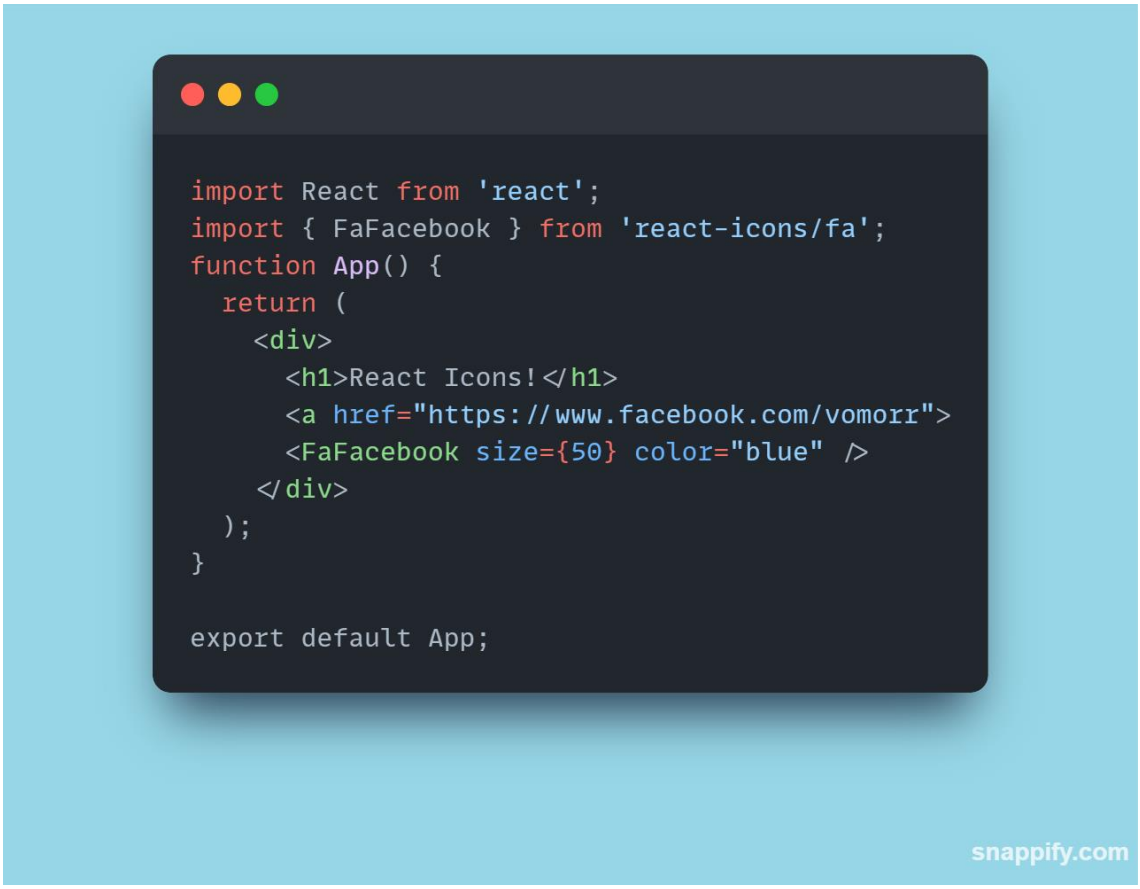

FIGURE 17. Using React Icon.

To display the Facebook icon inside a link, the FaFacebook library is imported in Figure 17 from react-icons/fa.

5.2.9 Bootstrap

Bootstrap is a popular front-end framework used for designing and creating responsive and mobile-first websites and web applications. It provides pre-designed templates, CSS styles, and JavaScript components to facilitate the development process and ensure consistency across different devices and screen sizes.

FIGURE 18. Using Bootstarp.

5.3 Backend Development

The installation and configuration of different packages and tools are covered in this section. For the installation and management of dependencies for this project, NPM (Node Package Manager) was used.

5.3.1 Installing Node.js and Express.js

Node.js version 20.09.0 was installed to configure the runtime environment for server-side JavaScript in this project. The following step involved starting the project using the "**npm init**" command in the project root directory after Node.js had been installed. The "**package.json**" file that was generated by this command is the project's configuration file. Developers can create and manage different dependencies with the help of this file, which is essential for tasks like creating and running the application (Simplilearn.com).

Express.js is used to integrate server-side functionality into the application after Node.js has been installed. To do this, use the command **"npm install express**" in the project root directory. Version 4.18.2 of Express was selected, and it was integrated into the project. Express.js was configured and prepared for usage in the application due to this command.

To add Mongoose as a project dependency, type "**npm install mongoose**" into the console. Use the command "**npm install nodemon**" to install Nodemon worldwide. During development, Nodemon is incredibly helpful since it can automatically restart the server when code is saved after a modification and monitor files for changes. Everything is ready to work on the project with Mongoose and Nodemon installed, especially when working with databases (Mongoose) and server development (Nodemon).

To use npm to install bcrypt.js as a project dependency, use the command "**npm install bcryptjs**". It is commonly used for hashing and securely storing passwords in applications.

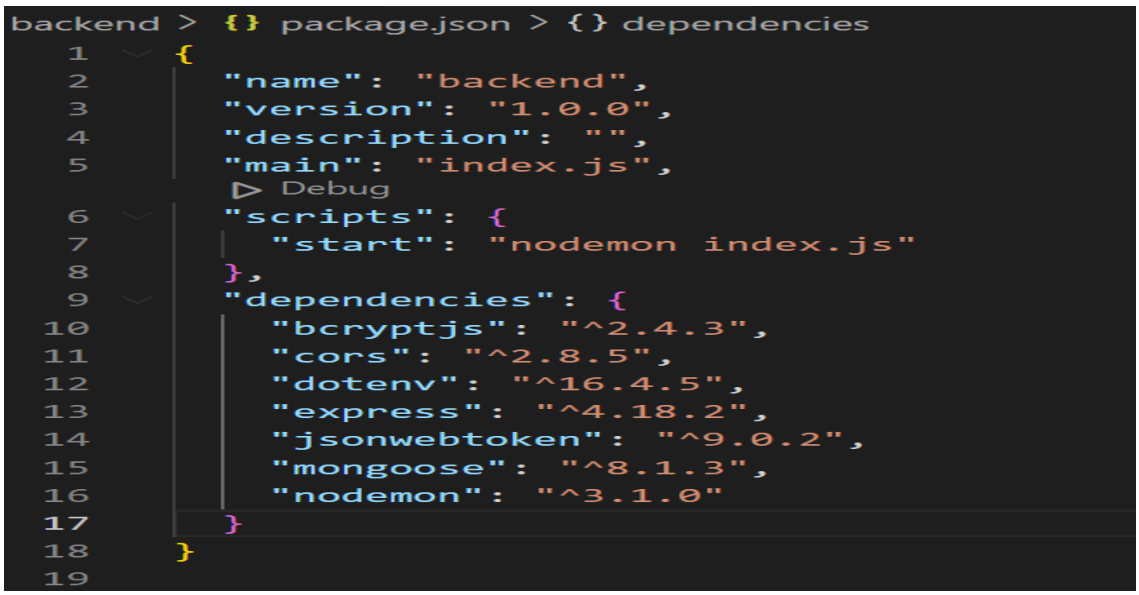

FIGURE 19. List of the backend package.json file.

5.3.2 Folder Structure

The clean architecture involves organizing a project into different layers, with each layer serving a specific role and being of importance. It aims to create systems that are easy to maintain, understand, and modify by separating concerns into different layers.

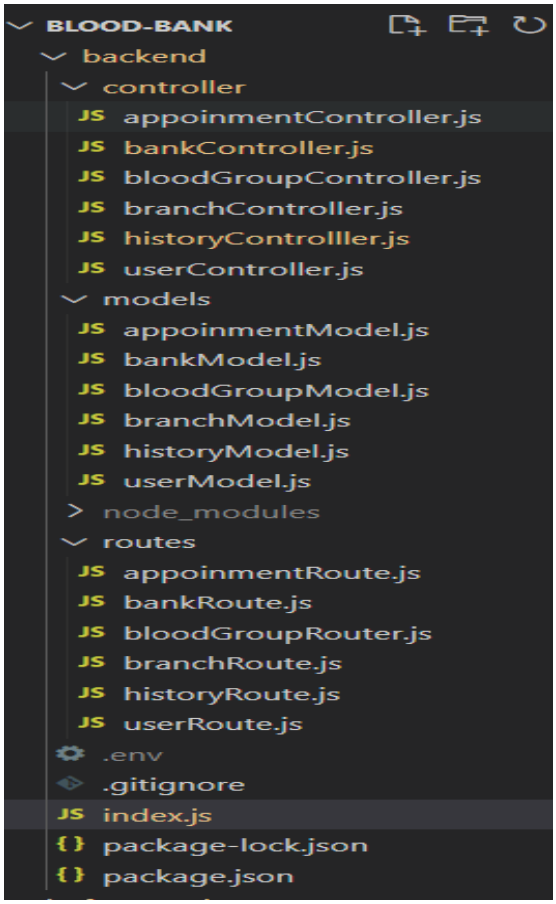

FIGURE 20. Folder structure.

This project consists of 3 main folders, each responsible for different tasks. The model handles data and rules, the view displays what users see, and the controller manages user actions and communication between the model and the view.
Routes: Routing is a system that directs user requests to the appropriate location in the backend code. It determines which part of the code should handle the request and forwards it to the controller. The controller interacts with the model to perform tasks and selects the correct view to display the requested information. Each router has many endpoints. Each endpoint has one function. Each function working activities on Controller (Codecademy. n.d.).

Model: We keep all of the data and the rules that control its behavior in this project. This part is in capable of handling rules, maintaining track of the application's data, and replying to queries regarding the status of the application. This project uses seven models: appointment, history, bloodGroup, bank, branch, news, and user. Every model has predetermined pathways.

Controller: The controller manages user actions and communication between the model and view. In this project, there are 6 controllers named userController, bankController, bloodGroupController, appointmentController, branchController, and historyController.

5.3.3 Server

To connect the server "**npm run start**" is used to start the server in development mode. Subsequently, the program opens the index.js file, which has the server startup logic (Figure 21).

```
app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
    databaseConnection();
});
```

FIGURE 21. Server configuration.

The program logs "Server is running on port 4000" from the console after starting the server that connects to port 4000. To connect to the database, it also uses the **databaseConnection** function.

5.3.4 Database Connection

MongoDB Atlas was selected for the creation of this application. A new cluster was established, and all required information was provided after an account was created.

```
const databaseConnection = async () => {
    const URL = process.env.Database_RUL;
    try {
        await mongoose.connect(URL);
        console.log('Database connection successful');
    } catch (error) {
        console.log(error);
    }
};
```

FIGURE 22. Database connection.

In this project, the **databaseConnection** function is connected with MongoDB database. It uses the connection URL from an environment variable (**Database_RUL**). On a successful connection, it logs a success message; otherwise, it logs the error.

5.3.5 Model Connection

Below is Figure 23, which provides the details of the user model. An instance of Mongoose schema is created and named "user". It contains the schema definition and business logic required for developing the user model.

```
const mongoose = require('mongoose')

const User = new mongoose.Schema(
    {
        name: { type: String, required: true },
        email: { type: String, required: true, unique: true },
        password: { type: String, required: true },
        image: {type: String},
        gender: { type: String, required: true },
        blood_group: { type: String, required: true },
        role: { type: String, required: true , default: "user"},
        date_of_birth: { type: String, required: true },
    },
    {
        timestamps: true
    },
    { collection: 'User' }
)
const model = mongoose.model('User', User)

module.exports = model
```

FIGURE 23. User Model Connection

A new Schema instance for the User model has been created according to name, email, password, image, gender, blood group, role, and date of birth. Email addresses must be unique to ensure that no two people have the same email address. Setting the "timestamps" second argument to true in the Mongoose schema allows for the automatic generation of createdAt and updatedAt fields. This feature is convenient for easily tracking when data is modified or created if needed.

const model = mongoose.model('User', User);

This line creates a model from the schema, allowing it to interact with the User collection in MongoDB using this model. Then exports the model so it can be used in other parts of the application.

5.3.6 Middleware Setup

A vital part of Express that effectively handles requests and responses is middleware. A Node server will normally listen for incoming requests and process each one in turn within a single process. The useful tools known as middleware functions help to simplify difficult operations into smaller, more manageable steps. They can carry out a variety of duties, including changing requests and answers and smoothly moving the process to the subsequent middleware function so that it can be worked on further. Arranging and simplifying the server's request handling procedure effectively. To manage the processing of incoming JSON requests and organize the data in the req.body, be sure to add express.json() as middleware.

```
const registerUser = async (req, res) => {}
const loginUser = async (req, res) => {}
const getUser = async (req, res) => {}
const updateUser = async (req, res) => {}
```

FIGURE 24. Some Middleware function requests.

5.3.7 Routing and APIs

For improved organization of router-level middleware, all the router-level middleware in this application is stored in the "**routes**" folder and defined to **index.js** file. Here is a detailed explanation of how router-level middleware are used in the application, as shown in Figure 25:

```
app.use(cors())
app.use(express.json())
app.use(express.urlencoded({ extended: true }));

const userRoute = require("./routes/userRoute")
app.use("/api/user", userRoute)

const bloodGroupRoute = require("./routes/bloodGroupRouter")
app.use("/api/bloodGroup", bloodGroupRoute)

const branchRoute = require("./routes/branchRoute")
app.use("/api/branch", branchRoute)

const appoinmentRoute = require("./routes/appoinmentRoute")
app.use("/api/appoinment", appoinmentRoute)

const bloodbankRoute = require("./routes/bankRoute")
app.use("/api/bloodBank", bloodbankRoute)

const historyRoute = require("./routes/historyRoute")
app.use("/api/history", historyRoute)

const newsRoute = require('./routes/NewsRoute')
app.use("/api/news", newsRoute)
```

FIGURE 25. Understanding Routes and importing router-level middleware.

The above picture displays the main routes of the application, including user, bloodgroup, blood-bank, branch, news, appointment, and history. Every route contains numerous sub-routes within it. Sub routes can handle various types of requests, such as GET, POST, PUT, or DELETE. Additionally, they can be secured to restrict access to authorized users only. As an IT project manager, it's important to consider the security measures for different routes in the system. One example is the route "/api/user/profile", which is a sub route of "/api/user". This route is only accessible to logged-in users, so we have implemented the "protected" middleware to ensure its security.

5.3.8 User Router Connection

In this project, 7 routes have been defined or imported on index.js file. Each of the routes has many endpoints. userRoute handles user management. bankRoute, branchRoute, and bloodGroupRoute deals with blood-related operations. appointmentRoute manages appointments for both sides donate and receive. And historyRoutes deals history related activities.

```
const express = require("express")

const router = express.Router()

router.use(express.json())

const {
    registerUser,
    loginUser,
    getUser,
    updateUser
} = require('../controller/userController')

router.post('/register', registerUser)
router.post('/login', loginUser)
router.get('/getUser', getUser)
router.put('/(:id)', updateUser)


module.exports = router;
```

FIGURE 26. User routes.

const router = express.Router(); This line creates a new router instance using Express's Router function. The router is used to define routes for handling HTTP requests. **router.use(express.json());** adds middleware to the router that parses incoming JSON requests. Then import the functions **(registerUser, loginUser, getUser, updateUser)** from the userController module. After that, the post route, get route and put route called the **registerUser, loginUser, getUser**, and **updateUser** function. Then, Export the router for use in other parts of the application, like the main server file.

5.3.9 Controller

In the Mongoose framework, controller functions are invoked by routes to retrieve the desired data from the related model. This activates the view renderer and allows for control of the relevant model (Codecademy. n.d.).

FIGURE 27. Relation with Router, Controller, and Model.

On routes, each endpoint has a function. It imports functions from a module called userController that likely contains code for handling user-related actions like registration, login, and retrieving user information. Other functions are also used in same way in this project.
It has been used to add, read, update, and remove functions in the controller.

```
const newUser = await User.create({
    name: req.body.name,
    email: req.body.email,
    password: encriptedPassword,
    image: req.body.image,
    gender: req.body.gender,
    blood_group: req.body.blood_group,
    role: req.body.role,
    date_of_birth: req.body.date_of_birth
})
```

FIGURE 28. Create method.

Here, the hashed password and other given information are used to create a new user in the database using the create method. This project's document fields are filled with information from the request body, which improves working with MongoDB by offering tools for managing and creating documents.

```
const user = await User.findOne({
    email: req.body.email
})
```

FIGURE 29. findOne functions.

To get a single document from a collection that matches the specified query conditions, use MongoDB's finfOne method.

const user = await User.findOne({email: req.body.email }), this line searches for a user in the database by email.

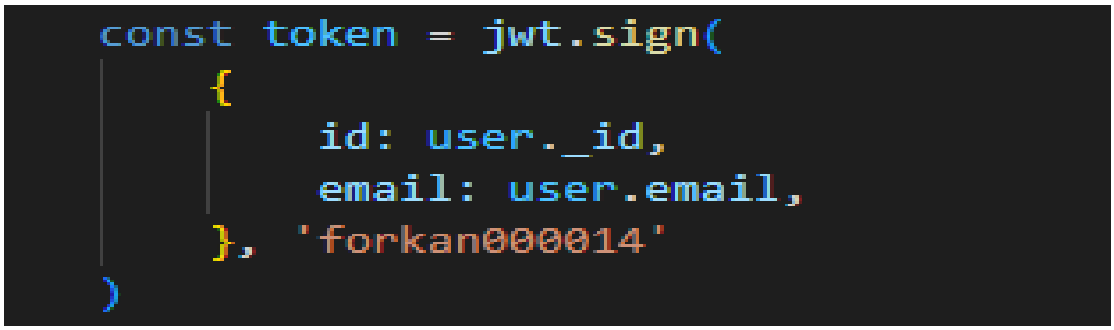

FIGURE 30. Creating JWT Token.

Generate a JSON Web Token (JWT) using the user's email address and ID. It expires in one hour and is signed using the secret from environment variables. This is frequently implemented for permission and authentication in app development.

const getUser = async (req, res) => {

This line defines an asynchronous function getUser which handles requests to get user information.

const userToken = req.headers['x-access-token'];

This line retrieves the token from the request headers.

try { const decoded = jwt.verify(userToken, process.env.JWT_SECRET);

This line verifies the token using the secret from the environment variables.

const user = await User.findOne({email: decoded.email });

This line finds the user in the database by email extracted from the decoded token.

updateOne method:

```
const updateUser = async (req, res) => {
    const userToken = req.headers['x-access-token']

    if (userToken) {
        try {
            const user_id = req.params.id
            const updatedUser = await User.updateOne({ _id: user_id }, {
                name: req.body.name,
                gender: req.body.gender,
                blood_group: req.body.blood_group,
                date_of_birth: req.body.date_of_birth
            })
            return res.json({ status: 'okay' })
```

FIGURE 31. updateOne method.

updateUser is an asynchronous function that responds to requests to update user data and gets the token from the request headers. Get the user ID from the request parameters and update the data from the request body. It then updates the user's record in the database with the updated information.

6 DESIGN AND VIEWS

6.1 Responsive

Responsive web design is a method of creating websites that adjust to different screen sizes and devices. In this application, responsiveness is handled by Bootstrap. A strong front-end framework called Bootstrap makes it easier to create mobile-first, responsive websites. With the use of a grid structure, pre-designed components, and CSS media queries, Bootstrap makes sure that the elements and layout of the website automatically adapt to various screen sizes and devices. This makes it easy to create a seamless and consistent user experience across desktops, tablets, and smartphones, enhancing both usability and aesthetic appeal.

6.2 Mobile and Desktop Views

The Digitalized Blood Bank application is mobile and desktop responsive, making it usable on any device. For large desktop screens, typically 1200px or more, layouts can be expansive and feature rich. Medium displays, ranging from 992px to 1199px, require a more compact yet still functional design. For tiny screens, spanning 768px to 991px, content must be streamlined for optimal readability and usability. Mobile devices present their own spectrum of screen sizes, with large screens at 414px and above, medium screens from 375px to 413px, and small screens ranging from 320px to 374px.

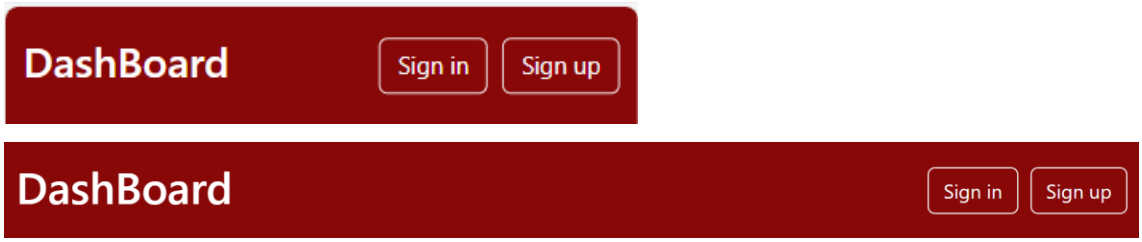

FIGURE 32. Navbar view for mobile and desktop views.

6.2.1 Home Page

When a user opens the application, the Home page is shown by default.

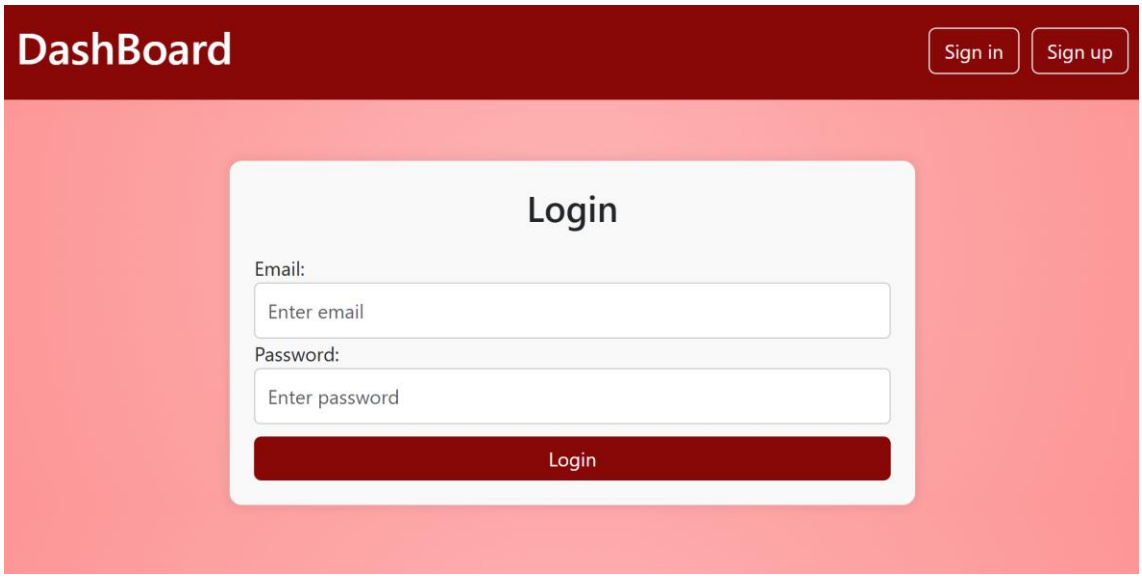

FIGURE 33. Application front page.

6.2.2 Sign Up Page

Users are required to register and log in before they can have the ability to donate or receive blood using the site.

Figure 34, which may be found below, displays the user interface given on the Sign up page:

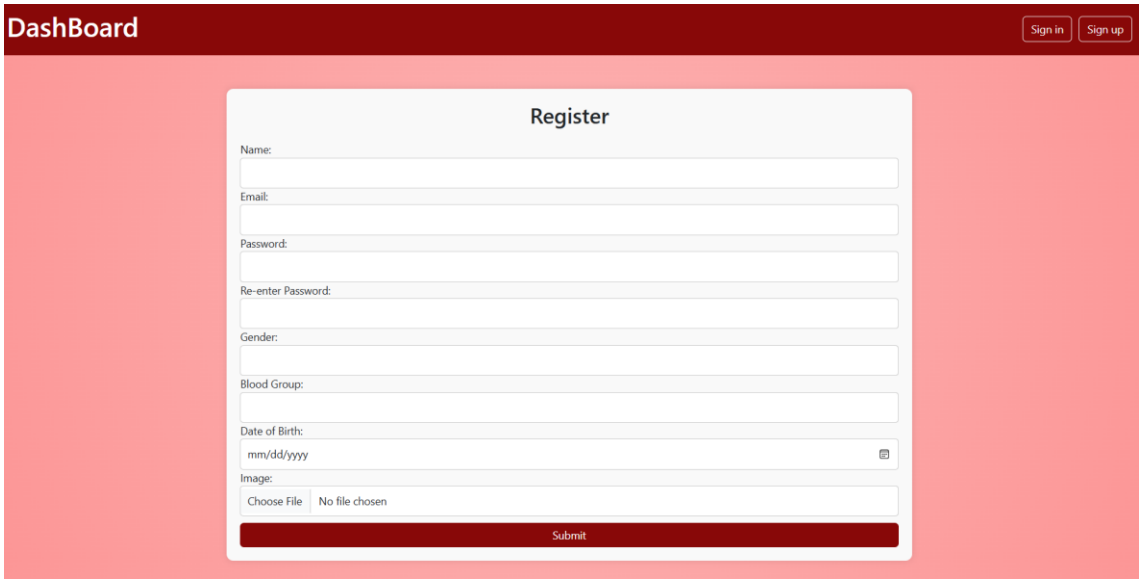

FIGURE 34. Sign up page.

A form with eight different input items may be seen on the Signup page shown in Figure 34. These components are as follows: Name, Email, Password, and Re-

enter-Password, Gender, Blood Group, Date of Birth, and Image. To successfully submit the sign-up request, it is necessary for the password and the re-enter password to be the same.

When users send in a valid password, a POST request with account information in JSON format will be sent to the "/api/user" route on the backend. If there is already a user with the same email address, the computer will send back an error message that says the user already exists. This message will be shown on the page. Otherwise, the user will be added to the database correctly. The frontend will sign the person in if the sign-up goes well.

6.2.3 Sign in Page

On the Login page, a user can log in to the system to perform actions that are restricted to users who are already logged in.

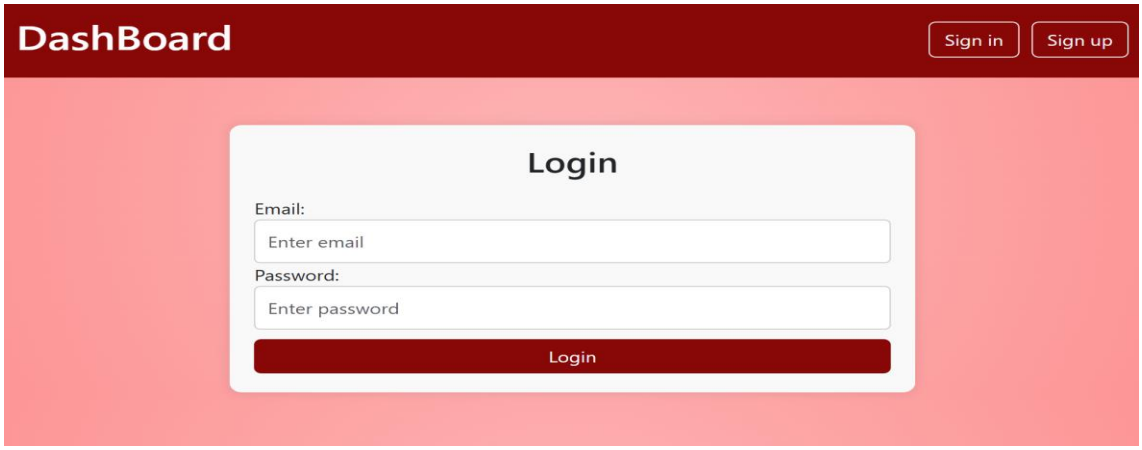

FIGURE 35. The application's login interface.

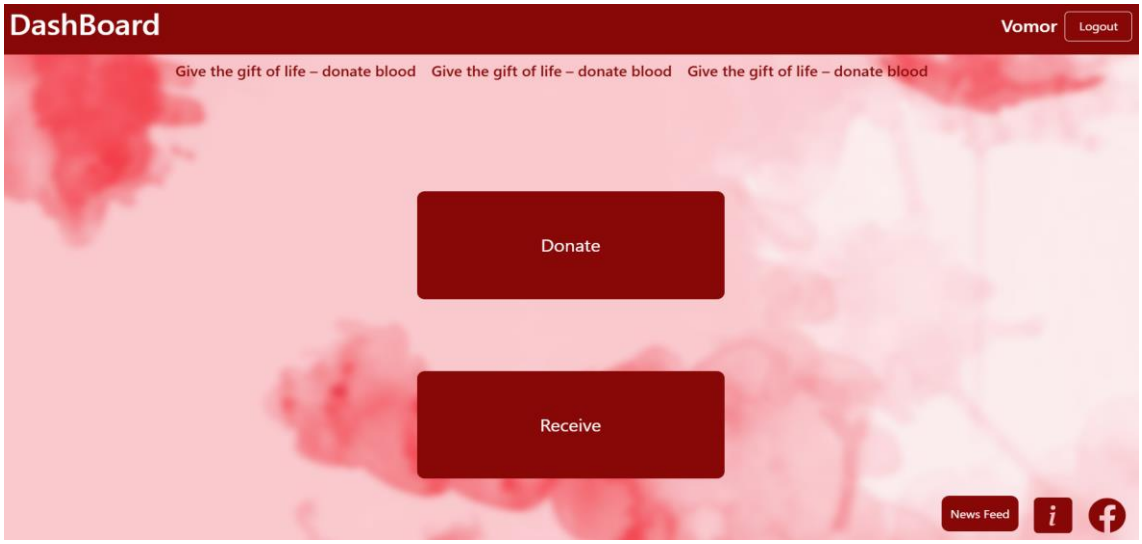

FIGURE 36. Home page view after Sign in.

If users wish to donate blood, they need to book an appointment with a particular branch and specify the date and time of their appointment.

Once an appointment has been scheduled, the user has the option to either finish the donation, cancel the appointment, or make changes to the appointment.

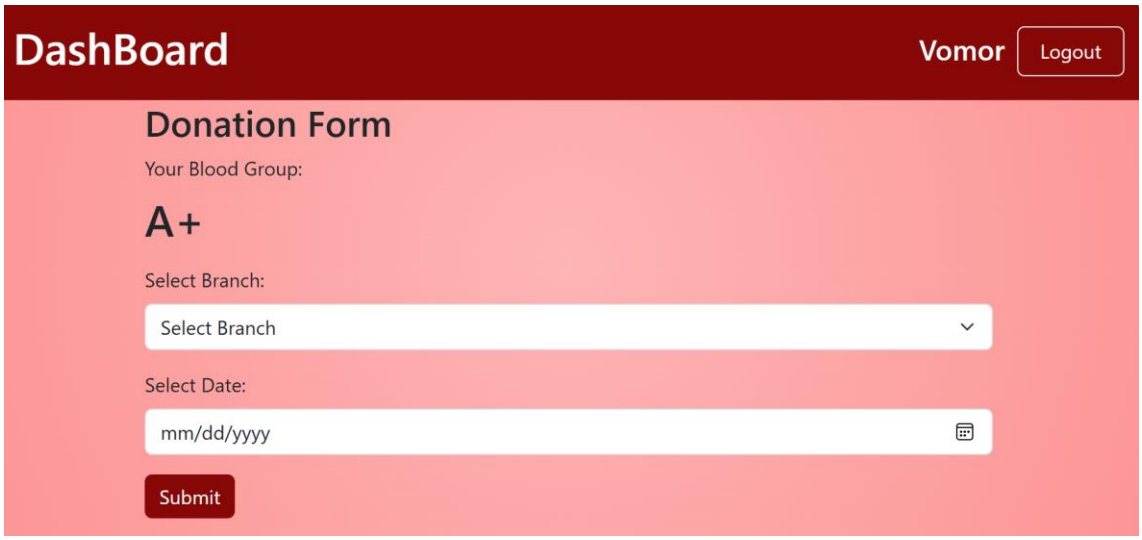

FIGURE 37. Appointment form for blood donation.

6.2.4 Blood Bank

If the users wish to collect blood, they must first go to the Blood Bank and select the blood bag that they need by making use of the many search options that are available during their visit.

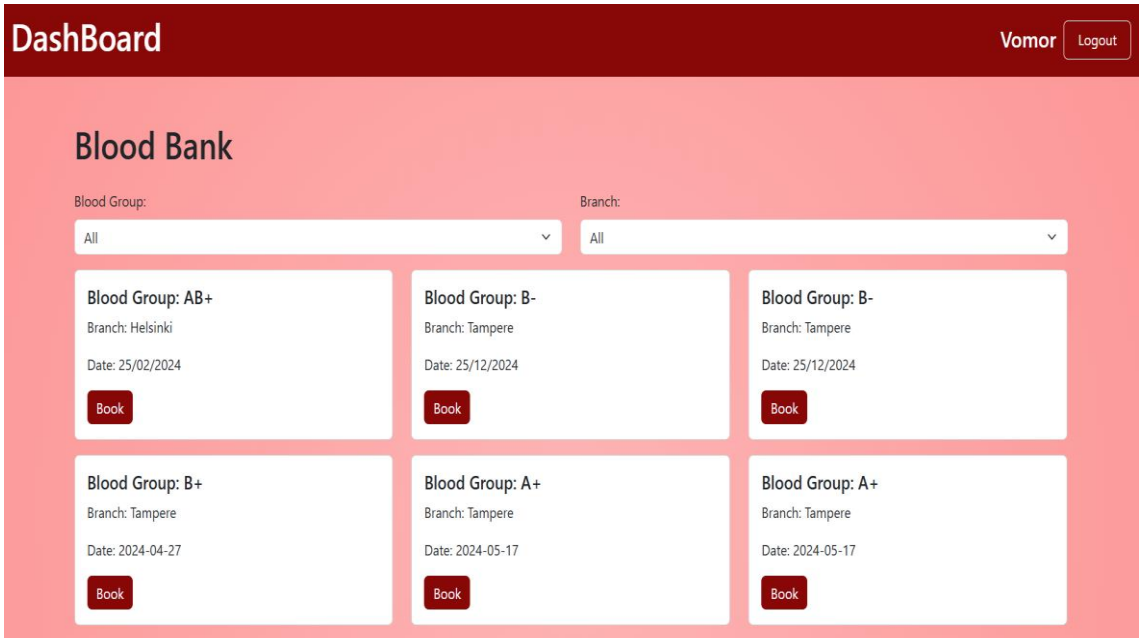

FIGURE 38. Blood Bank views.

After choosing the blood bag needed, confirmation of the selection is required.

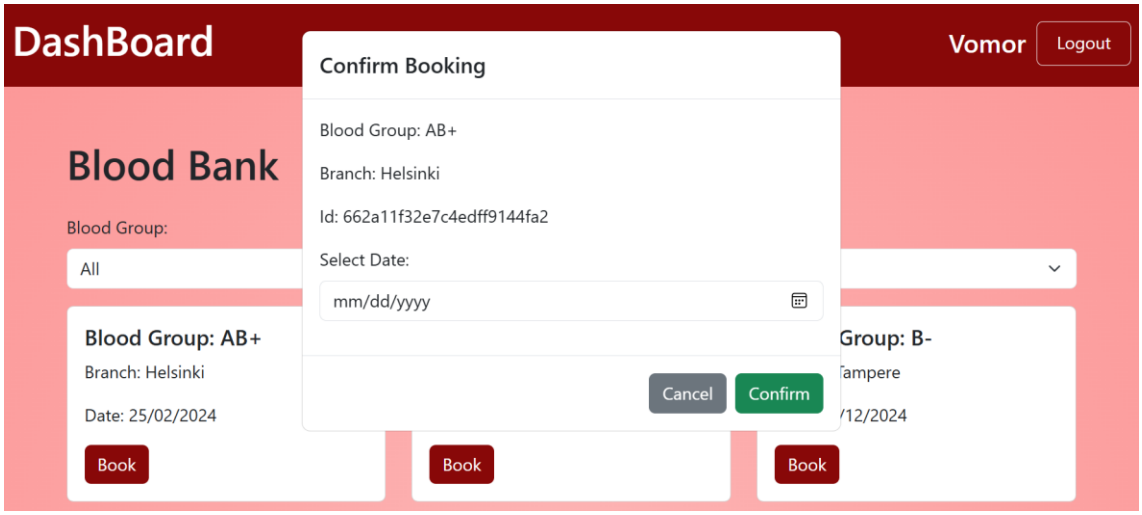

FIGURE 39. Appointment form for Receiver.

6.2.5 Profile Page

On the profile page, users can view details about themselves. The details can be edited if necessary. Additionally, once an appointment has been scheduled, the user has the option to either finish the donation, delete the appointment, or change the appointment. After the user has finished their appointment, the information of the donation or receiving will be kept as Donation History and Receiving History respectively.

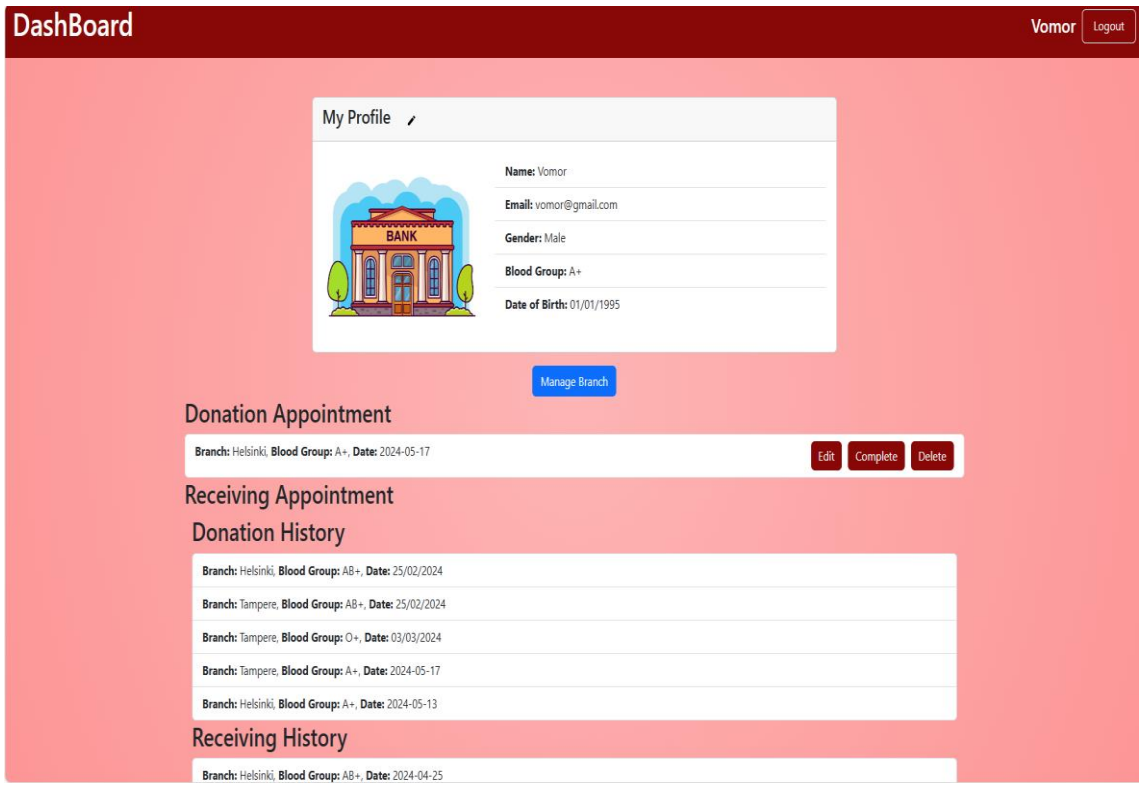

FIGURE 40. User profile page view on desktop.

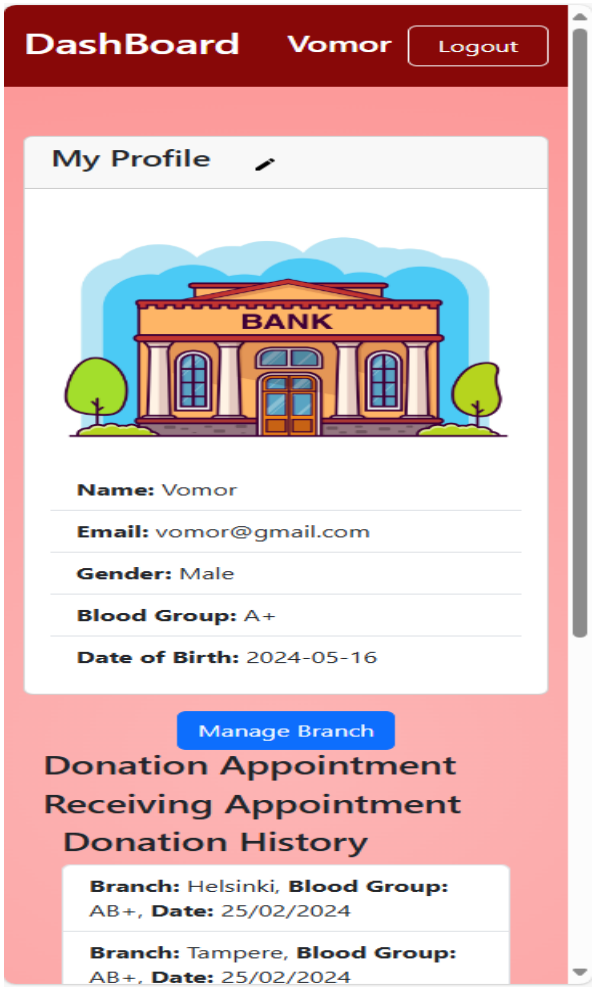

FIGURE: Profile view on Mobile.

6.2.6 Admin-specific Task Design

In this web-based application, the admin can control the blood bank (including the ability to add and edit blood bags), branches (including the ability to add a new branch, edit existing branches, and delete branches), and publish news on the news feed field.

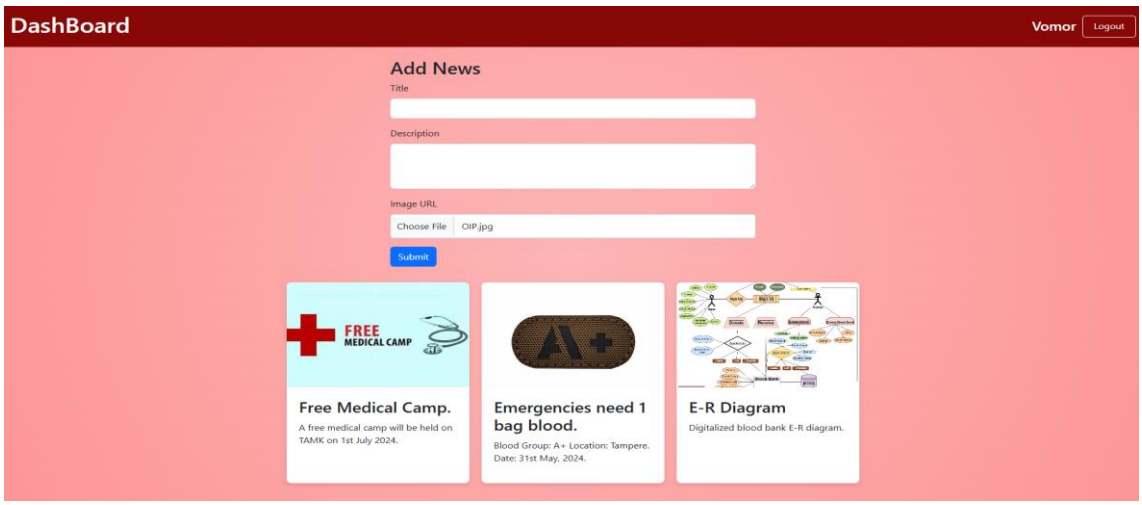

FIGURE 41. News Feed View.

7 TESTING

A vital part of every software development process is testing. Testing gives users confidence in the product's quality. The test results show how well the code is written and offer suggestions for making the codebase better.

Postman has been used in this project for testing. Postman provides an environment for developing APIs that cover every step of the software development lifecycle, including developing, mocking, evaluating, maintaining, and tracking. It makes it possible to test the same request using environment-specific variables against various environments. When starting the creation of the user interface, the login and registration endpoints were checked and verified. By simply stating that GET/POST requests to /api/user/login and /api/user/register, respectively, the login endpoints required authentication, while the register endpoints were tested without the need for any authentication.

.

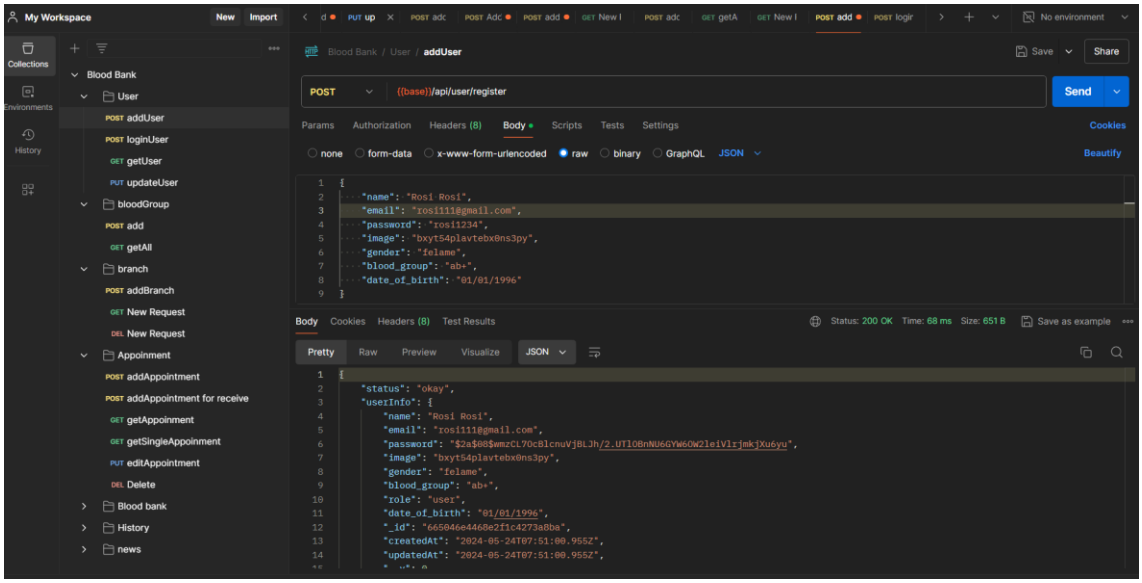

FIGURE 42. API testing with Postman.

8 DEPLOYMENT

In this development phase, the application is run on the local machine. A web server should be used to deploy the application so that users can access it. It is vital to perform routine maintenance and monitoring to provide the ideal experience for users.

The cloud hosting service Render has been used for this thesis application. Render can be used as both a hosting service and a virtual machine. The backend and frontend projects have been released through GitHub integration. In render, there is a free plan that gives you 0.1 code CPU and 512 MB of storage.

Setting up the backend was the first step. Thus, in a new web host, the location is in "/backend." "**npm install**" is the command for building, and "**npm run start**" is the command for starting. With the backend project's secret key for the database URL, it was necessary to include a secret variable for that URL. The backend process is shown in Figure 42.

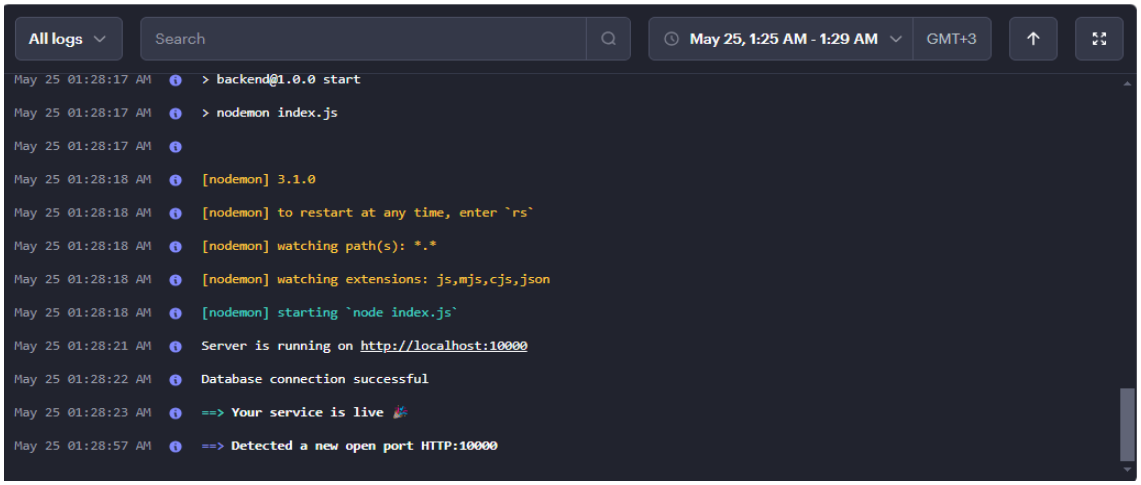

FIGURE 42. Backend Pipeline.

Using the deployed server address in place of localhost in the frontend is the second step. One example of using the deployed server address in the frontend is shown in Figure 43.

```
try {
    const response = await axios.post('https://blood-bank-server-mkn0.onrender.com/api/user/login', {
        email,
        password
    });
```

FIGURE 43. Frontend Server Address.

After sending the code to GitHub, the next step involves launching the frontend project as a static site. "**npm install; npm run build**" is the build command, and the directory is "/frontend". The command "build" is used to run the code. The frontend deployment pipeline is shown in Figure 44.

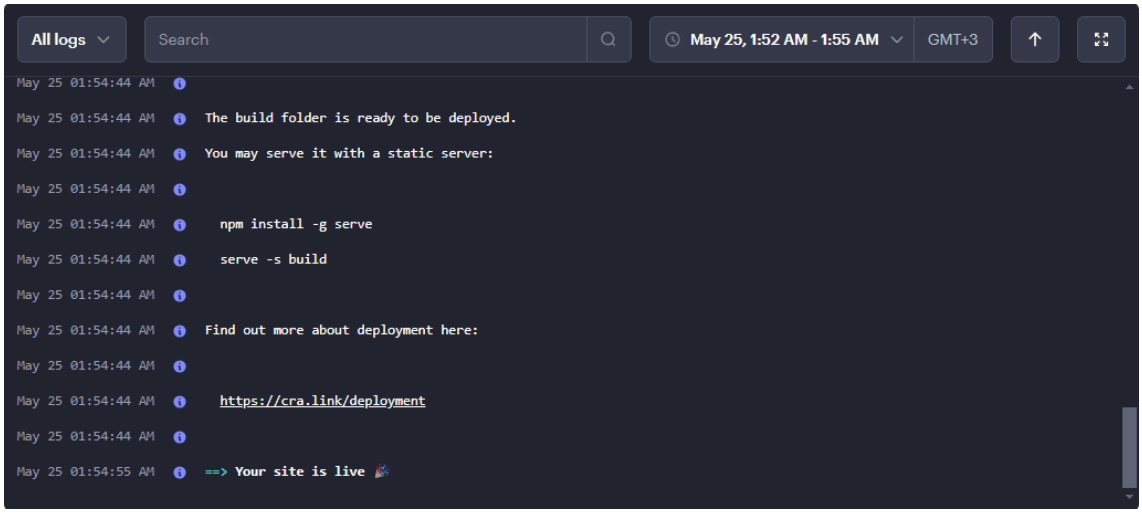

FIGURE 44. Frontend Pipeline.

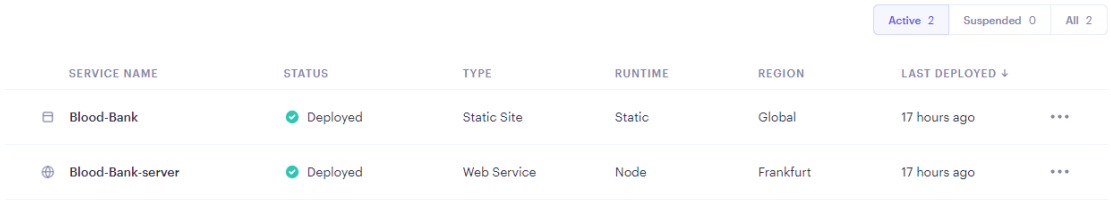

FIGURE 45. Rendering the Application.

"**Blood-Bank**" indicates that the application's front end has deployed successfully, and "**Blood-Bank-server**" web services indicate that the app's backend is operating properly.

Access to the Render application that has been installed can be gained through browsing: https://blood-bank-y9og.onrender.com

9 RESULT AND DISCUSSION

The Digitalized Blood Bank application was successfully developed using the MERN stack technologies. This proves that the MERN stack can be used to create complicated full-stack apps. The program used a variety of tools and packages, including MongoDB, Express, Node, and React. The thesis gives detailed descriptions of these technologies and tools.

The implementation of this project has gone through several phases, including planning, design, development, and testing of the application. The use of MERN technologies has created a platform that is strong, flexible, and efficient. The project has reached the goals that were set at the start of the development phase and resulting in a fully functional application capable of meeting user demands.

Users, including donors, recipients, and admin can register and log in securely, and update their profiles with relevant information such as blood type, and contact details. Similarly, users can schedule appointments for blood donation or receive blood based on various search criteria such as location, blood type availability, and date. They can also view their donation or receiving history on their profile.

The application provides easy access to blood donation services, making it convenient for users to find donate, and receive blood. Also, the system streamlines the blood bank process, making it more efficient for donors and recipients. Furthermore, digitalized records reduce the risk of errors associated with manual record-keeping.

Admin users can change details about users, manage the blood bank, control branches, and make events. The final app is easy to understand and use in general. Any user can finish their blood donation tasks with just a few steps.

The application can be enhanced with additional features. This includes adding push notifications for mobile apps to keep users informed about donation reminders, urgent requests, and upcoming events. Integrating WhatsApp or SMS alerts can enable more immediate communication. Predicting blood demand trends us-

ing machine learning algorithms can help in better inventory management. Additionally, implementing AI chatbots can assist users with inquiries and provide instant support.

Furthermore, lots of modern methods and ideas could be used to improve the application's multiple features even more. First off, the application's "create-react-app" package offers server-side rendering as an alternative to the more traditional client-side rendering. Second, to enforce high-quality well-known testing tools like Jest, react-testing-library, and Cypress could be used to execute a variety of testing methods, such as complete, unit, and integration testing.

10 CONCLUSION

The Digitalized Blood Bank application, developed using the MERN stack, successfully meets the initial goals and user demands. It offers a user-friendly, intuitive interface, making blood donation activities straightforward for all users. The project's success demonstrates the capability of the MERN stack to build complex, full-stack applications effectively.

The application allows users to register, log in, update profiles, schedule appointments, and view donations or receiving history. The app makes the blood bank process easier, which cuts down on mistakes and increases productivity. Admin users can manage the app, control branches, and publish news and events.

As a complete guide to the MERN stack, the thesis is useful for those who wish to learn more about the MERN stack's development. The application met all the standards that had been set from the beginning of the project.

REFERENCES

Axios (2023). Getting Started | Axios Docs. Read on: 13.02.2024. Available at: https://axios-http.com/docs/intro

Codecademy. (n.d.). MVC: Model, View, Controller. Read on: 25.02.2024. Available at: https://www.codecademy.com/article/mvc

Data Management. (n.d.). What is MongoDB? Features and how it works. Read on: 20.02.2024. Available at: https://www.techtarget.com/searchdatamanagement/definition/MongoDB#:~:text=MongoDB%20is%20a%20schema%2Dless.

Duggal, N. (2021). What is the MERN Stack? Introduction & Examples. Read on: 17.02.2024. Available at: https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mern-stack-introduction-and-examples

Emmanuel, A. (2023). Using Design Patterns in Full-Stack Development. Read on: 28.02.2024. Available at: https://medium.com/@aboyejiemmanuel07/using-design-patterns-in-full-stack-development-b3a39df8c346

Fullstackopen.com. (n.d.). Fullstack part1. Read on: 03.03.2024. Available at: https://fullstackopen.com/en/part1/introduction_to_react

FreeCodeCamp.org. (2023). How to Fetch API Data in React. Read on: 13.03.2024. Available at: https://www.freecodecamp.org/news/how-to-fetch-api-data-in-react/

Geeksforgeeks (2019). MERN Stack. Read on: 30.01.2024. Available at: https://www.geeksforgeeks.org/mern-stack/

GeeksforGeeks. (2020). ER diagram of Bank Management System. Read on: 21.04.2024. Available at: https://www.geeksforgeeks.org/er-diagram-of-bank-management-system/

Hoque, S. (2020). Full-Stack React Projects. Packt Publishing Ltd. Read on: 19.02.2024. Available at: http://books.google.ie/books?id=097dDwAAQBAJ&printsec=frontcover&dq=mern+stack+web+app&hl=&cd=2&source=gbs_api

Heroku (2020). Cloud Application Platform | Heroku. Read on: 27.03.2024. Available at: https://www.heroku.com/.

LifeBank, H. (n.d.). Digitization Guidelines for a Blood Bank. Read on: 19.01.2024. Available at: https://hamrolifebank.com/digitization-guidelines-for-a-blood-bank.

Microsoft (2022). Database Design Basics. Read on: 24.02.2024. Available at: https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5.

Matina, S. (2023). What is the 3-Tier Architecture? Read on: 03.01.2024. Available at: https://medium.com/@shrestha.matina.20/what-is-the-3-tier-architecture-4520522e0720

Node.js. (n.d.). Introduction to Node.js. Read on: 09.02.2024. Available at: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

OpenAI (2024). ChatGPT. Read on: 17.01.2024. chatgpt.com. Available at: https://chatgpt.com/

Pabbly. (n.d.). ExpressJS Overview. [online] Available at: https://www.pabbly.com/tutorials/expressjs-overview/

Sazib, M. (2022). Mongoose Discusses Schema and Model Work. Read on: 27.02.2024. Available at: https://medium.com/@musasazib/mongoose-discusses-schema-and-model-work-81c5b4b73975

Scribd. (n.d.). Blood Bank Management System | PDF | Use Case | Visual Impairment. Read on: 17.01.2024. Available at: https://www.scribd.com/document/432397432/Blood-Bank-Management-System

Simplilearn.com. (n.d.). How to Install Express JS using NPM (Step-By-Step). Read on: 15.02.2024. Available at: https://www.simplilearn.com/tutorials/nodejs-tutorial/introduction-to-install-express-js

Tutorials Point (2019). JavaScript - Overview – Tutorialspoint. Read on: 14.02.2024. Available at: https://www.tutorialspoint.com/javascript/javascript_overview.htm

World Health Organization (2022). World Health Organization. Read On: 09.01.2024. Available at: https://www.who.int/