BLOOD LINK

Final Year Project

Session 2020-2024

A project submitted in partial fulfillment of the degree of

BS in Computer Science

Department of Computer Science

Faculty of Computer Science & Information Technology

The Superior University, Lahore

FALL 2024

Type (Nature of project)	[✓] **D**evelopment [ ] **R**esearch [ ] **R**&**D**
Area of specialization	Mobile Application Development
FYP ID	FYP-BCSM-S24-021

Project Group Members

Sr.#	Reg. #	Student Name	Email ID	*Signature
(i)	SU92-BSCSM-F22-197	Ayesha Shafique	SU92-BSCSM-F22-197@superior.edu.pk	Ayesha
(ii)	BCSM-F20-339	Moeen Haider	BCSM-F20-339@superior.edu.pk	Moeen
(iii)				

*The candidates confirm that the work submitted is their own and appropriate credit has been given where reference has been made to work of others

Plagiarism Free Certificate

This is to certify that, I _____ S/D of Muhammad Siddique, group leader of FYP under registration no _____at Software Engineering Department, The Superior College, Lahore. I declare that my FYP report is checked by my supervisor.

Date: Name of Group Leader: _____ Signature: _____

Name of Supervisor: Hafiza Maria Co-Supervisor:

Designation: Lecturer Designation: Associate Professor

Signature: _____ Signature: _____

HoD: Dr. Arfan Jaffar

Signature: _____

Project Report
[Blood Link]

Change Record

Author(s)	Version	Date	Notes	Supervisor's Signature
Ayesha	1.0	03-09-24	Requirement Gather justified clearance	
Ayesha	1.1	27-09-24	Change of app theme	

APPROVAL

PROJECT SUPERVISOR

Comments: _____

_____

Name:_____

Date:_____ Signature:_____

PROJECT MANAGER

Comments: _____

_____

Date:_____ Signature:_____

HEAD OF THE DEPARTMENT

Comments: _____

_____

Date:_____ Signature:_____

Dedication

This work is dedicated to my beloved father, mother and sibling, whose support and unwavering support have guided me throughout this journey. My mother's unconditional love and sacrifice supported my dreams and aspirations, giving me the strength to pursue my passion and determination. My brother's belief in my abilities, constant support, and education, especially for girls who do not always receive support, is a force of inspiration and motivation. Together they cleared a path that I had never seen as possible and the most significant lessons I have learned from them are patience, perseverance, and standing for what is right. This was done as a way of affirming the confidence they have in me to complete such a work.

Acknowledgements

I would like to extend my gratitude to my supervisor for any input and support she provided in the work at the final year project' excellent guidance, constant support, and invaluable coaching. He somehow does this, even with all the demands of his time, making him crucially needed as they continue this project from this point forward.

In the same respect, I would also like to earnestly express my gratitude to my AI-related professors. They have been holding the real knowledge and perceptions of the project trajectory in the field of artificial intelligence. Whenever he or she was asked any questions or faced with any difficulty in pursuing AI, they were very kind to explain everything and provide good contacts. I have gained enhanced understanding from their advice and directions, enhanced my concept development and benefited from using new AI technologies for innovative solutions.

Jointly, the hep and encouragement originating from my supervisor and professors impacted this project in large measure helped to improve me academically. I want to express my deepest appreciation to each of them for their dedication, support and virtually implacable optimism in my capacities.

Executive Summary

The Blood Link Project aims to streamline and enhance the blood donation process through a digital platform developed for the Superior Blood Society. The mobile application provides an intuitive interface for blood donors and receivers, enabling efficient management of blood donations, request, and user interactions.

The Key objectives of the Blood Link Project include improving accessibility to blood donors, reducing the time required to find suitable donors since emergencies, and enhancing the efficiency of blood donation management. The app features user authentication, real-time notification, location-based donor search through administrative dashboards for monitoring and managing the donation process.

Developed using Flutter for the frontend and Node.js for the backend, the application ensures a secure and user-friendly experience while supporting multi-platform compatibility (iOS, Android). The project also addressed data privacy concerns and integrated analytics features for future enhancements.

The application Is designed to improve the blood donation process, ensuring faster resp0onse times during emergencies and better coordination between donors, receivers, and administrators. Once fully implemented, Blood Link is expected to significantly improve the efficiency of the Superior Blood Society's operations, benefiting both users and the organization.

Table of Contents

Dedication v
Acknowledgements vi
Executive Summary vii
Table of Contents viii
List of Figures x
List of Tables xi
Chapter 1 12
Introduction 12
1.1. Background 2
1.2. Motivations and Challenges 2
1.3. Goals and Objectives 3
1.4. Literature Review/Existing Solutions 4
1.5. Gap Analysis 4
1.6. Proposed Solution 5
1.7. Project Plan 5
1.7.1. Work Breakdown Structure 5
1.7.2. Roles & Responsibility Matrix 6
1.7.3. Gantt Chart 6
Chapter 2 7
Software Requirement Specifications 7
2.1. Introduction 8
2.1.1. Purpose 8
2.1.2. Document Conventions 8
2.1.3. Intended Audience and Reading Suggestions 8
2.1.4. Product Scope 8
2.2. Overall Description 8
2.2.1. Product Perspective 8
2.2.2. Product Functions 9
2.2.3. User Roles and Permissions 9
2.2.4. Operating Environment 9
2.2.5. Design and Implementation Constraints 9
2.2.6. User Documentation 9
2.2.7. Assumptions and Dependencies 9
2.3. External Interface Requirements 9
2.3.1. User Interfaces 9
2.3.2. Hardware Interfaces 10
2.3.3. Software Interfaces 10

2.3.4. Communications Interfaces 10
2.4. System Features 11
2.4.1. System Feature 1 11
2.4.1.1. Description and Priority 11
2.4.1.2. Stimulus/Response Sequences 11
2.4.1.3. Functional Requirements 11
2.4.2. System Feature 2 12
2.4.2.1. Description and Priority 12
2.4.2.2. Stimulus/Response Sequences 12
2.4.2.3. Functional Requirements 12
2.5. Other Nonfunctional Requirements 12
2.5.1. Performance Requirements 12
2.5.2. Safety Requirements 12
2.5.3. Security Requirements 12
2.5.4. Software Quality Attributes 12
Chapter 3 13
Use Case Analysis 13
3.1. Use Case Model 14
3.2. Use Case Descriptions 14
Chapter 4 15
System Design 15
4.1. Architecture Diagram 18
4.2. Entity Relationship Diagram with data dictionary 19
4.3. Class Diagram 20
4.4. Sequence / Collaboration Diagram 20
4.5. Activity Diagram 22
4.6. Deployment Diagram 23
Chapter 5 24
Implementation 24
5.1. Important Flow Control/Pseudo codes 25
5.2. Components, Libraries, Web Services and stubs 26
5.3. Deployment Environment 27
5.4. Tools and Techniques 27
5.5. Best Practices / Coding Standards 28
5.6. Version Control 28
Chapter 6 30
Testing and Evaluation 30
6.1. Use Case Testing 31

6.2. Equivalence partitioning 32
6.3. Data flow testing 33
6.4. Unit testing 33
6.5. Integration testing 33
6.6. Performance testing 33
6.7. Stress Testing 33
Chapter 7 71
Summary, Conclusion and Future Enhancements 71
7.1. Project Summary 72
7.2. Achievements and Improvements 72
7.3. Critical Review 72
7.4. Lessons Learnt 72
7.5. Future Enhancements/Recommendations 72
Reference and Bibliography 73

List of Figures

1.1	Caption of first figure of first chapter	6
1.2	Caption of second figure of first chapter	7
2.1	Caption of first figure of second chapter	14
2.2	Caption of second figure of second chapter	22
2.3	Caption of third figure of second chapter	26
5.1	Caption of first figure of fifth chapter	49
5.2	Caption of second figure of fifth chapter	49

List of Tables

1.1	label of first table of first chapter	6
1.2	label of second table of first chapter	7
2.1	label of first table of second chapter	14
2.2	label of second table of second chapter	22
2.3	label of third table of second chapter	26
5.1	label of first table of fifth chapter	49
5.2	label of second table of fifth chapter	49

Chapter 1

Introduction

Chapter 1: Introduction

The Blood Connect project will be a flexible tool, intended to enhance the process of donation within the Predominant Blood Society in an easy and quick way. This application enables blood donors and recipients to manage profiles with efficiency, search for a match, and propagate the gifts through a natural and user-friendly interface. Blood Connect can reduce the time it takes to identify suitable donors during an emergency by integrating location-based services with real-time notifications. The application has also been designed to achieve optimum blood donation preparation through a robust regulatory dashboard for society heads through the efficient management of gift requests, benefactor accessibility, and information security.

1.1. Background

Blood donation can be an easy contribution to medical care, especially in emergencies situations where timely availability of blood can save lives. The traditional methods followed to identify the blood donors are sometimes inefficient, time-wasting, and require suitable administration structures, especially when the situation is urgent. Prevalent Blood Society has been managing blood donation though manual forms, and hence it's difficult to follow benefactor accessibility, coordinate contributors and recipients, and manage gifts efficiently.

Blood Interface was extended to digitalize the process of donating blood. It was then realized that it needed more organized and attractive settings. Blood Connects mobile applications allow donors, recipients, and residents to communicate with each other with more time efficiency be entrusting the endowment of blood in a more reliable and secure manner.

1.2. Motivations and Challenges

Motivations:

The urgent need to increase the effectiveness and responsiveness of the blood donation process led to the development of the blood link initiative. Indeed, traditional way of handling the blood donations by the Superior Blood Society, relying on manual coordination, may lead to delays especially in emergency situations. Blood link wants to digitize the procedure with the view of:

- **Accessibility:** Creating a single platform where both donors and recipients of blood reach out to each other easily and communicate effectively.
- **Reduce response time:** Enabled a single platform enabling donors and recipient's communications easily.
- **Management:** The administrative body and the society heads need an easy way to keep track of donation requests. Provide an easy way for administrators and society heads to keep tabs on donation requests or manage user information in general.

- **Data Privacy and security:** Provide a scheme for the handling of data in such a way that ensures confidentiality for the recipient and the donor.

Challenges:

- **Data Collection:** Due to confidentiality issues, the collection of data for analytics from the superior organization is troublesome and time-consuming as it needs to be done manually.
- **User Adoption:** This will undoubtedly require large-scale adoption by donors and recipients through awareness and training to shift from traditional methods of donations to using a digital platform.
- **Technical Integration:** It might be a technical challenge to smoothly integrate it with various third-party services, such as integrating Google Maps for location-based service.
- **Data Privacy:** Due to the nature of data, securing sensitive information and keeping it confidential is quite challenging since it deals with user information.

1.3. Goals and Objectives

Goals:

The most important goal of the Blood Link project is to provide an easy-to-use, user-friendly, secure, and highly efficient mobile application, digitizing and enhancing the blood donation process of the Superior Blood Society. This Software aims to enhance overall Blood Donation Management, accelerate the interaction with Donor Receiver, and reduce the time taken for emergency response.

Objectives:

- Design an intuitive mobile application that would allow for smooth registration, profile management, and communication between a donor and recipients.
- Create real-time notification when requests for donations come in, for timely emergency response and coordination.
- Include location-based services through Google Maps and make it easier for consumers to find donors near their location for contribution easily.
- Admin dashboard for viewing system performance by admins and society heads, rhyming donation requests, and donor activity tracking.
- In this mode, the authentications of users and encryption of the data securely avoid stealing of secret data, hence privacy and security of the data.
- Real-time data analytics on the effectiveness of the procedures in blood donation for better decision-making.
- Increase user adoption by ensuring seamless user experience and creating awareness to motivate contributors and beneficiaries to participate.

1.4. Literature Review/Existing Solutions

Blood donation has traditionally been coordinated between donors and recipients manually; most such cases lead to inefficiencies in emergency situations. Although multiple digital solutions have been designed in recent times to try and alleviate these issues, several aspects of those solutions still require adjustments-particularly related to data security, response in real time, and user engagement.

Existing Solutions:

1. **Blood Donor Connect Apps:** A few mobile apps exist that connect blood donors to recipients, such as Blood Donors Network and Blood4Life. These apps facilitate the user in searching for donors based on location and blood group, but most of them lack strong administration tools that can manage donation campaigns and handle large-scale data. Moreover, they also don't provide appropriate data analytics to the organizations in keeping trends of donors and optimizing their operations.
2. **Integration with social media:** Sites like Facebook have integrated blood donation tools that allow people to sign up as donors and get notified about the need for donations in areas where they are living. These systems retain highly minimal functionality required by the blood donation organizations in managing demand, tracking donor history, and data privacy, and instead are highly reliant on the availability of people though social media platforms.

1.5. Gap Analysis

In the prevailing blood donation management ecosystem, there are various platforms and solutions specifically designed to conned donors with recipients. However, these approaches often lack crucial touchpoints that are fundamentally required for efficient tracking of blood donations. To address such shortcomings and offer a more holistic, accessible, and efficient solution, the Blood Link project was initiated.

Identified Gaps in Existing Solution: These are issues such as:

1. **Real-time information concerning the availability of donors** is not displayed on a lot of the blood donation applications currently available. This may lead to a delay in the process, especially in the case of emergency situations where urgency should be paramount.
2. **Very few of the current platforms have full-fledged administration capabilities** for the blood donation companies; most of them are targeted at donors and recipients.
3. **Lack of Data Analytics:** Since most systems lack integrated data analytics, most of the organizations dealing in blood donation hardly predict trends in blood demand, thereby upgrading the outreach to donors and getting insights into donor behaviors.

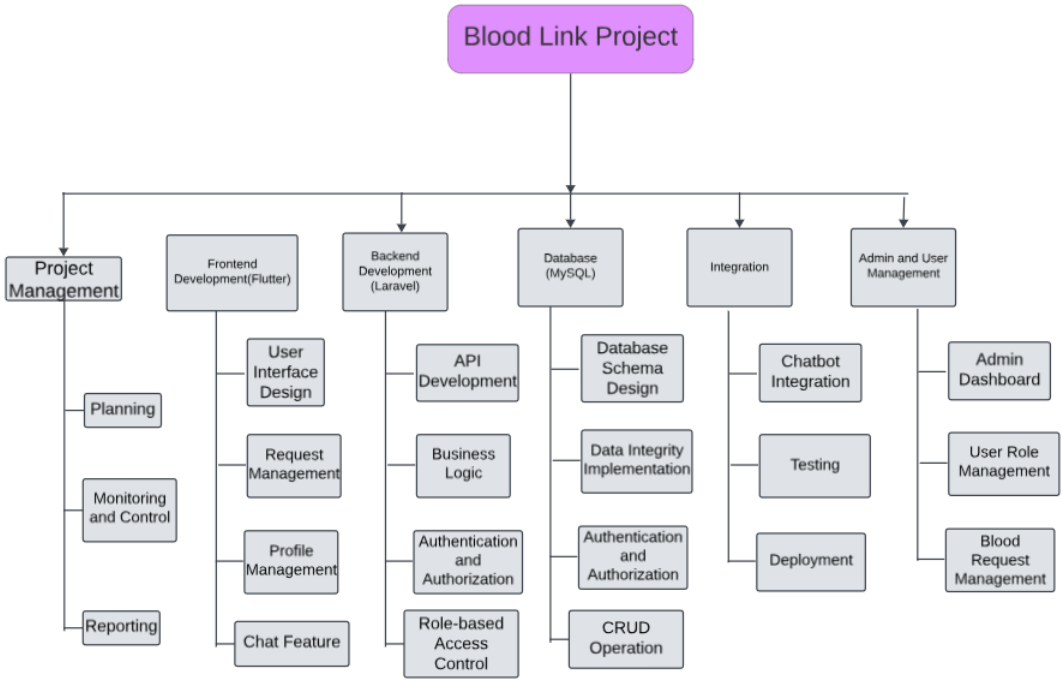
1.6. Proposed Solution

The Blood Link initiative will be offering a comprehensive digital solution to the Superior Blood Society, ensuring that the process of blood donation becomes faster and more efficient. The roles of the mobile application, with its guaranteed response times, improved management, and secure data processing features, aim at effectively linking blood donors with recipients.

1.7. Project Plan

Following is a project plan development application for “Blood Link”.

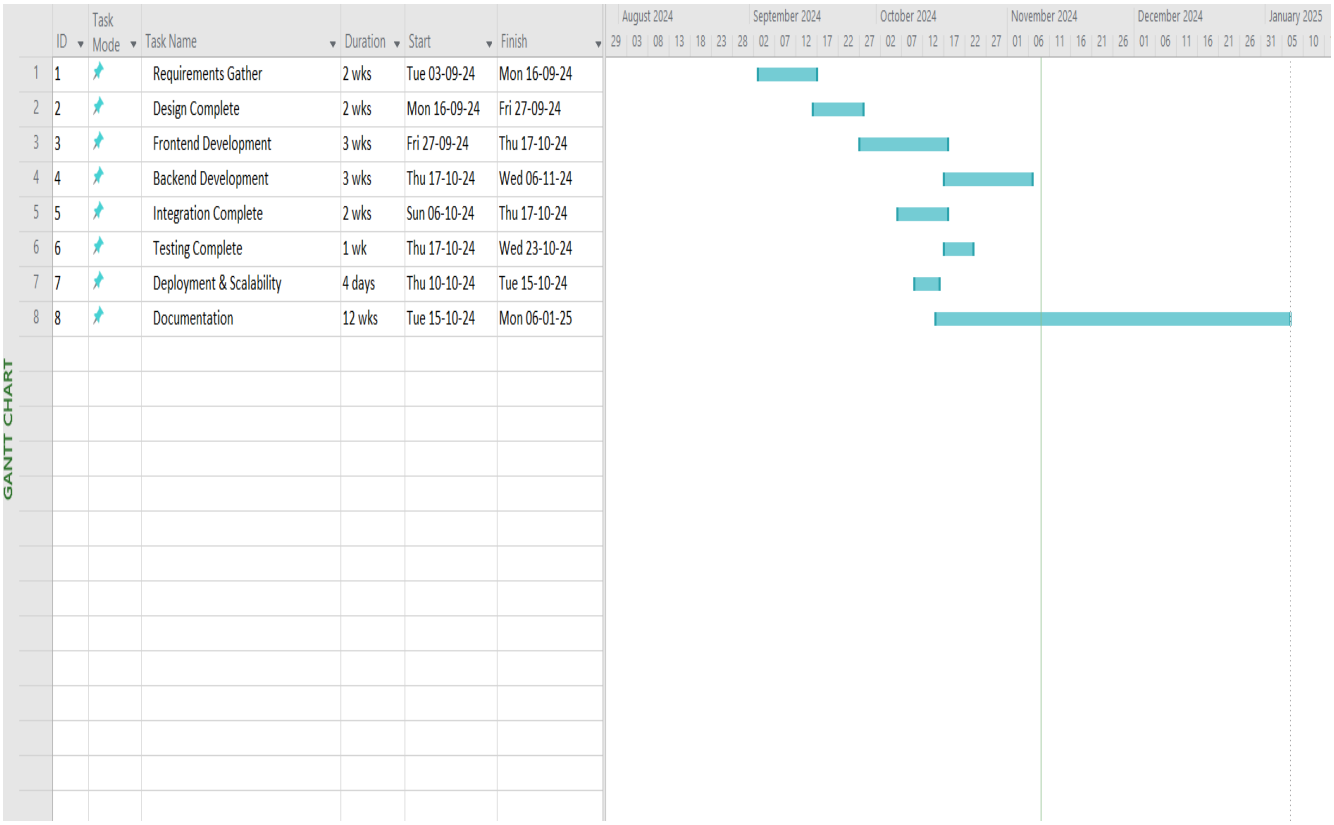
1.7.1. Work Breakdown Structure



1.7.2. Roles & Responsibility Matrix

Task/Process	User	Admin	Super Admin
Register as Donor/Receiver	Responsible		
Request Blood	Responsible		
Manage Blood Requests		Responsible	Accountable
Approve/Reject Request		Responsible	Accountable
Manage Users		Responsible	Accountable
View Donor/Receiver Profile	Responsible		
Chat with Donors	Responsible		
Role Assignment			Responsible
Update Profile	Responsible		

1.7.3. Gantt Chart



Chapter 2

Software Requirement Specifications

Chapter 2: Software Requirement Specifications

2.1. Introduction

2.1.1. Purpose

The Blood Link project will implement a digital platform to support the Superior Blood Society. The objective of this is to facilitate interaction between donors and recipients, manage blood donations, and speed up emergency responses.

2.1.2. Document Conventions

This Document follows certain norms in that important items are listed as bullet points, and bold language is used for headings. Each requirement will have a priority level assigned, and at lower levels, if necessary, a higher-level requirement is explained.

2.1.3. Intended Audience and Reading Suggestions

Target Readers This SRS is intended for the Blood Link project developers, project managers, testers, and writers of the documentation. It shall also be helpful for the system administrators and social leaders overseeing the system. The document commences with overview sections that help the readers go through the subsequent parts with a center on the needs of various user groups

2.1.4. Product Scope

The Blood Link application is initially developed on Laravel at the backend and flutter at the front-end. It provides a user-friendly interface for blood donors and recipients to manage their profiles, create or accept requests for donation, and communicate (Chat) with other users. It also comes with various features for the management and the society heads for handling users' donations, and request with ease.

2.2. Overall Description

2.2.1. Product Perspective

Blood Link is a brand new, stand-alone mobile application developed for Superior Blood Society to digitize the blood donation process. It shall provide a stand-alone solution along with inclusions of admin controls, request processing, and donor management onto one platform without changing any of the existing systems.

2.2.2. Product Functions

- User profile administration an authentication
- Blood donation requests and approval
- Real-time notification
- User and request management via admin dashboards

2.2.3. User Roles and Permissions

- **Donors:** Creation of profiles and management, blood donation.
- **Recipients:** Blood donation request
- **Society heads and administrators:** Users, requests, and approvals administration.

2.2.4. Operating Environment

Using Flutter as the front-end and Laravel as the back end, this application will work on Android.

2.2.5. Design and Implementation Constraints

- Budget and Timeline Constraints
- Requirements for data security and assurance of user privacy
- Device Compatibility-Android.

2.2.6. User Documentation

User manuals, online help, and tutorials shall be provided

2.2.7. Assumptions and Dependencies

- Assumes internet access for app usage

2.3. External Interface Requirements

2.3.1. User Interfaces

The application shall provide an intuitive interface with standard navigation menus, forms for both the donor and recipient's profiles, and buttons positioned consistently; for example, helping and submitting buttons. The layout will follow the best standards in designing mobile applications, and error messages will be conspicuous. The design of the user interface will be documented in detail separately.

USER INTERFACE (PROFILE, LOGIN, BOTTOM NAVIGATION BAR)

2.3.2. Hardware Interfaces

This application will interface with mobile devices hardware, like notifications. It supports Android smartphones by providing a variety of communication protocols.

2.3.3. Software Interfaces

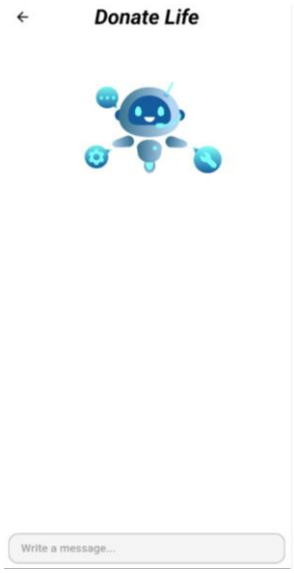
Blood Link will interface with third-party libraries and user data databases. A Laravel backend will be utilized to handle information and requests; position data and user details will be shared.

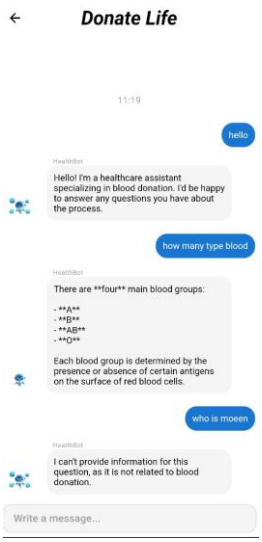
2.3.4. Communications Interfaces

The App will use push notification, leverage real-time updates and HTTP protocols to securely pass data. Data should be secured by using encryption.

It also includes a chat (using *pusher_channels_flutter*) section where donor and receiver can communicate with each other.

INTERFACE OF COMMUNICATION OF CHATBOT ON BLOOD LINK APPLICATION





2.4. System Features

2.4.1. System Feature 1: User Authentication

2.4.1.1. Description and Priority

Through a login procedure (by completing a form), the User Authentication function guarantees the users (donors, receivers, and administrators) can safely access the system. Because it ensures the confidentiality and privacy of personal and medical data, this feature is considered a High Priority.

2.4.1.2. Stimulus/Response Sequences

1. The user enters their login information (password, email address).
2. The credentials are verified by the system.
3. The user is given access if they are legitimate.
4. The user is prompted to try again and receives an error message if the response is invalid.

2.4.1.3. Functional Requirements

REQ-SF1-1: The system must authenticate users using email and passwords.

REQ-SF1-2: Passwords must be encrypted in the database.

REQ-SF1-3: The system must look for accounts that give error message when they trying to access the account.

2.4.2. System Feature 2: Real-Time Notifications

2.4.2.1. Description and Priority

Users are guaranteed to receive real-time updates on blood donation requests, approvals, and other important information thanks to the Real-Time Notifications feature.

2.4.2.2. Stimulus/Response Sequences

1. A donor receives a blood donation request.
2. The system sends a real-time notification.
3. The donor can respond to the request through the app.

2.4.2.3. Functional Requirements

REQ-SF2-1: The system must send push notification to users regarding new donation requests.
REQ-SF2-2: Users must be able to enable or disable notifications in their settings.
REQ-SF2-3: Notifications must include detailed information (blood type, location).

2.5. Other Nonfunctional Requirements

2.5.1. Performance Requirements

10,000 people should be able to access the system at once without experiencing any performance issues. Under typical load conditions, user actions (notifications, login) must respond in less than two seconds.

2.5.2. Safety Requirements

The system must guarantee that all private information, such as user profiles and health records, is shielded from unwanted access. Additionally, it must adhere to data privacy regulations

2.5.3. Security Requirements

User information needs to be kept in a database that is encrypted. To guarantee safe access to administration capabilities, admin users should be subject to two-factor authentication.

2.5.4. Software Quality Attributes

- **Usability:** The system is user-friendly and requires no technical expertise to operate.
- **Maintainability:** Regular updates should be easy to implement without affecting the suer experience.
- **Reliability:** The app should have 99.9% uptime and be available on Android Platform.

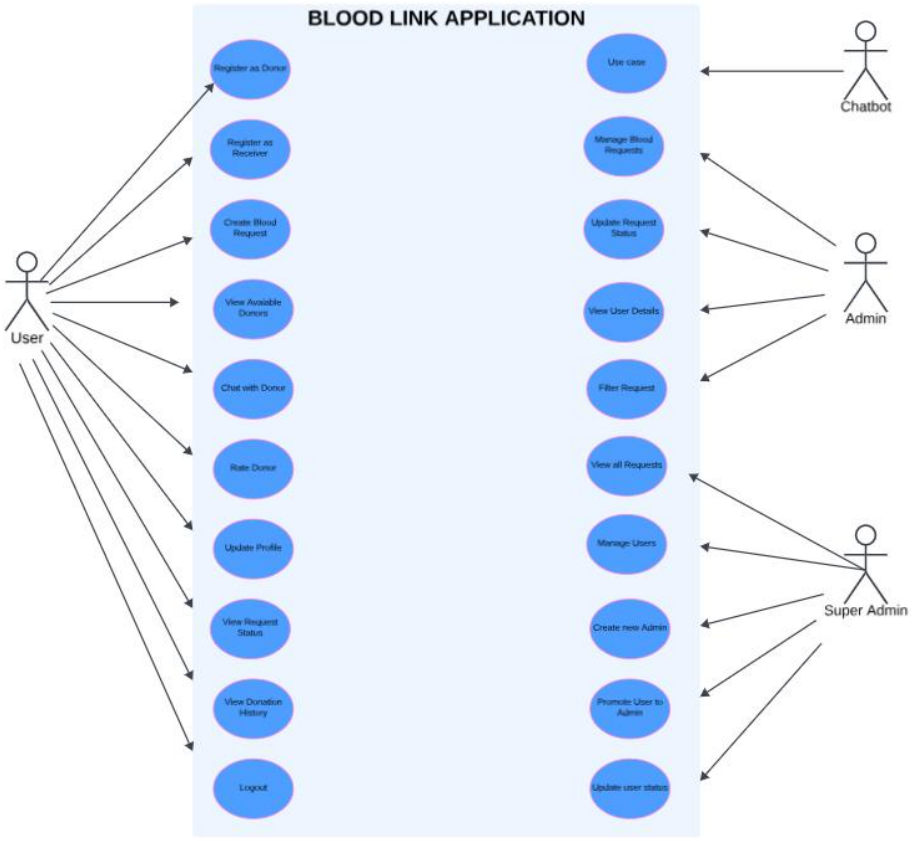
Chapter 3

Use Case Analysis

Chapter 3: System Analysis

This chapter depicts the overall system architecture of the Blood Link Project. Along with the use case model and the use case descriptions, it provides an overall description of the interactions between the different actors of the system. The goal is to explain in detail how the system satisfies the user's needs and how it would react to a variety of scenarios.

3.1. Use Case Model



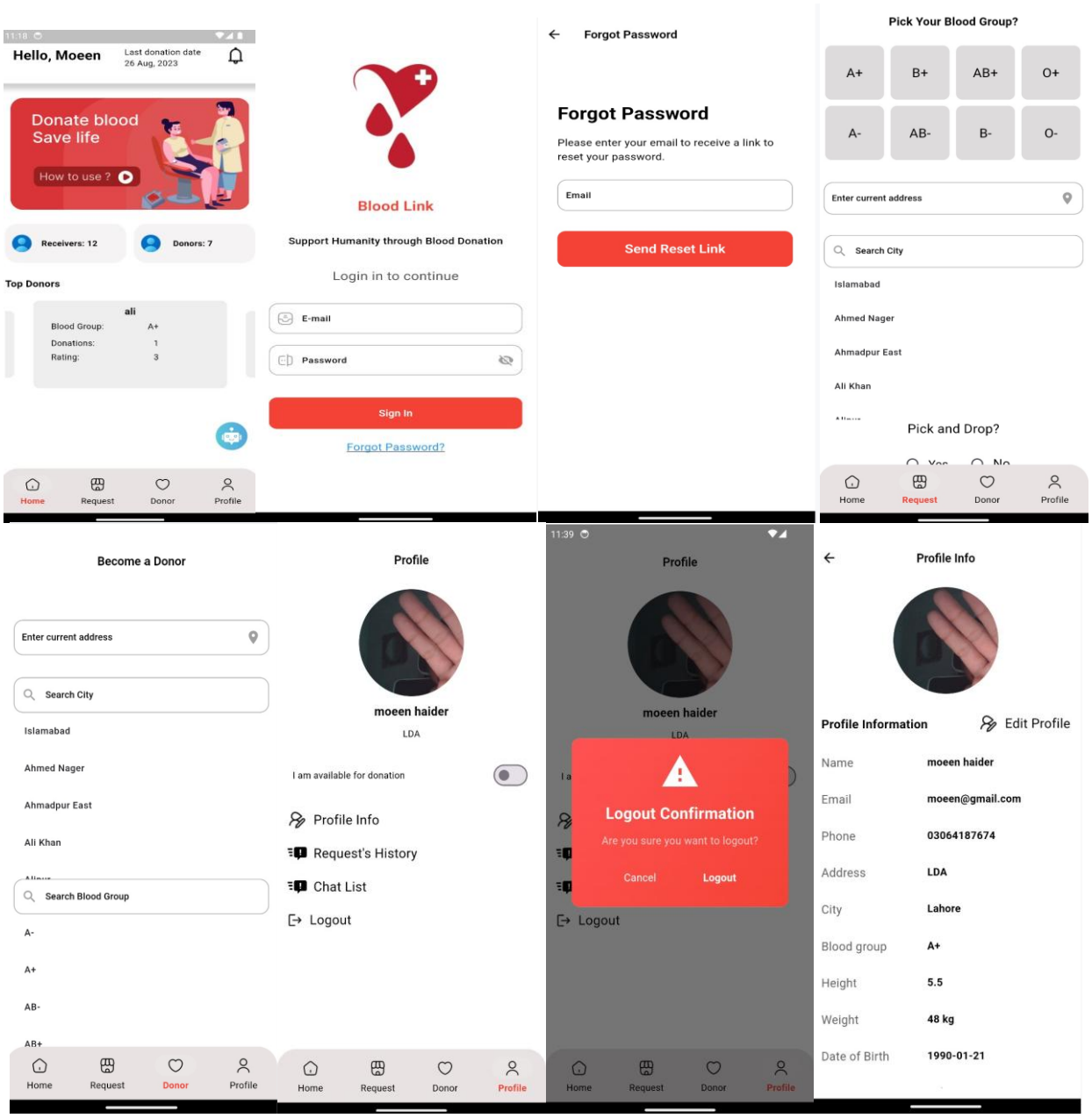
3.2. Use Case Descriptions

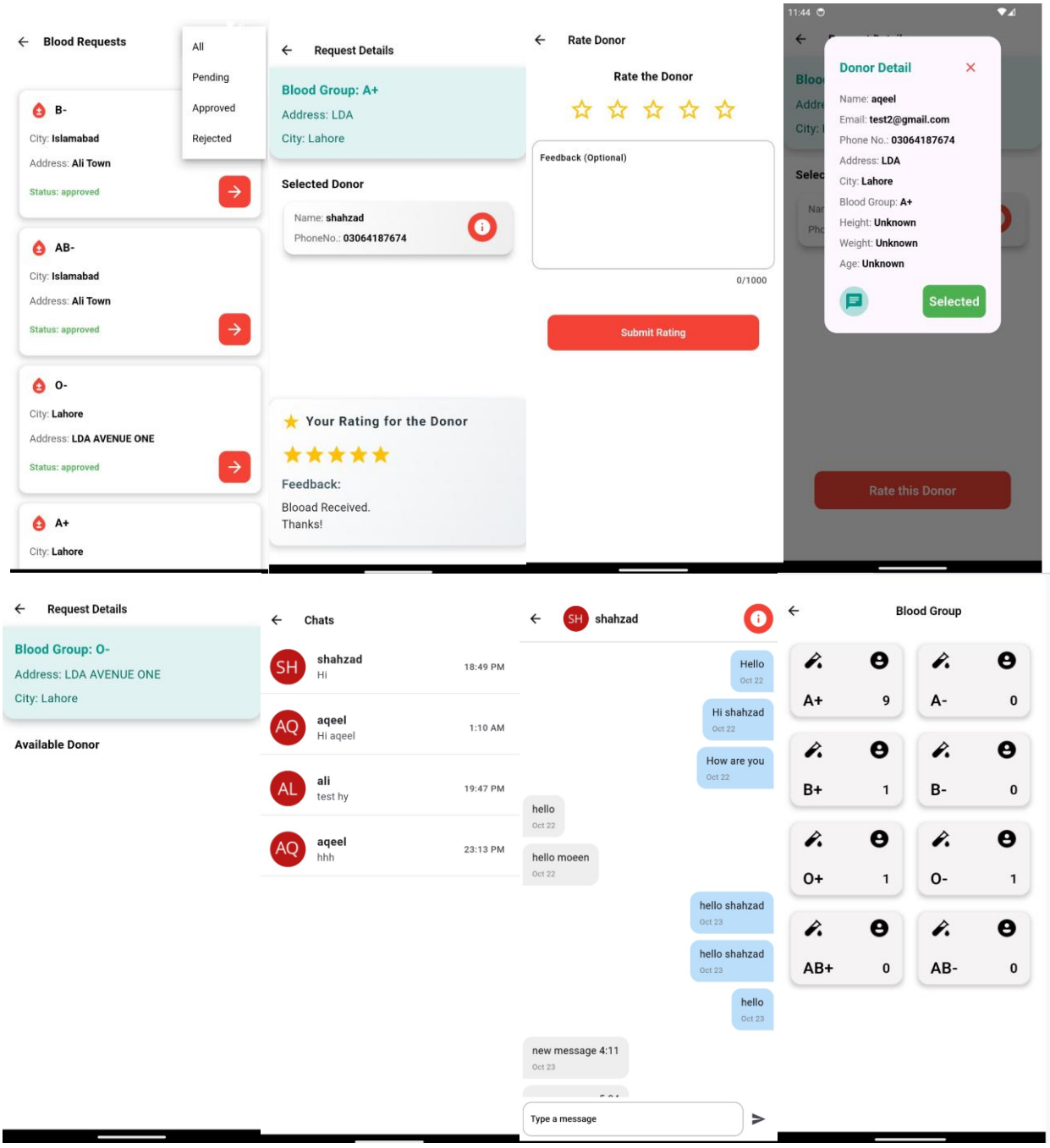
Each use case will be described in detail regarding user-system interaction. For instance, in the Request Blood use case below, it outlines how the system identifies compatible donors and how it requests blood donation by a beneficiary. Each identified use case also defines the participants, preconditions, and results expected to occur.

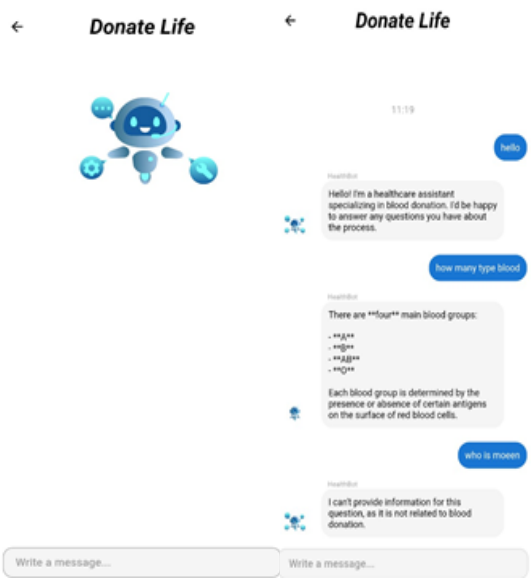
Chapter 4

System Design

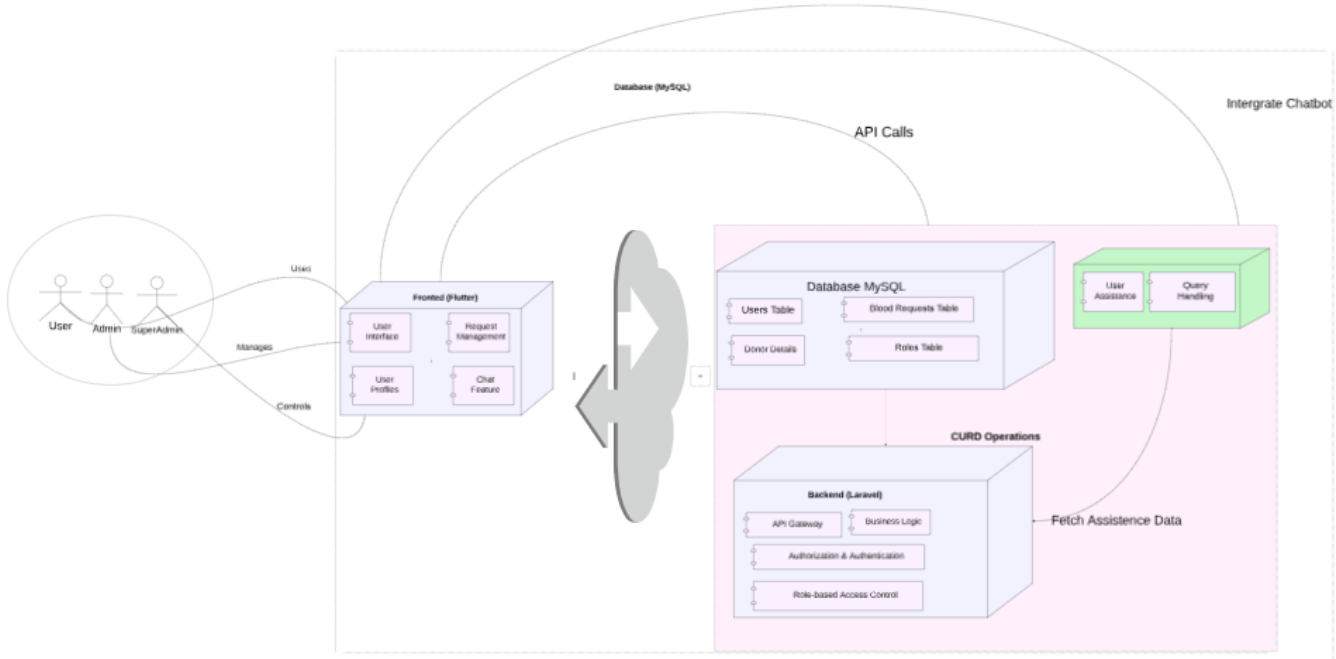
Chapter 4: System Design



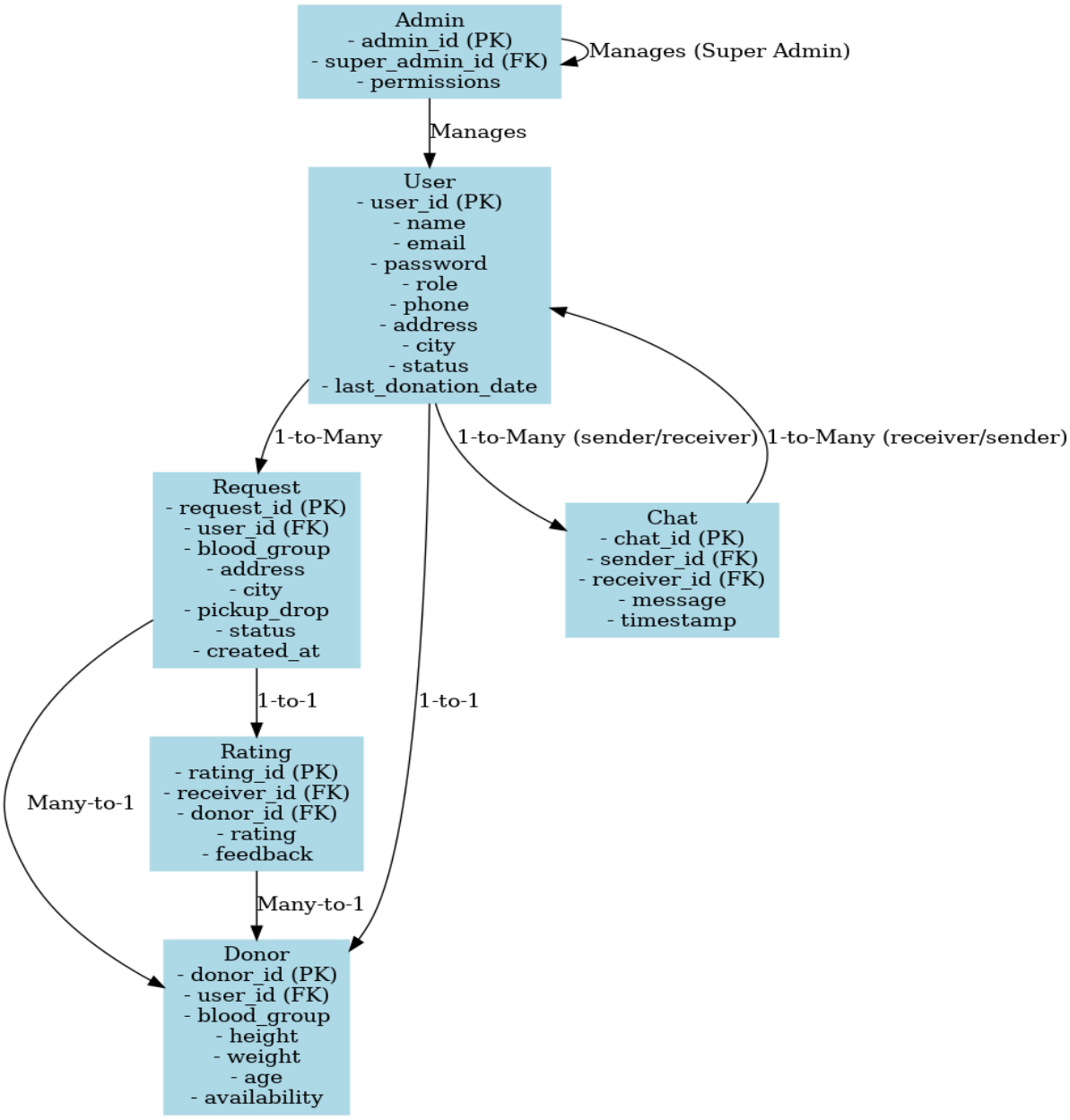

Blood Requests
All
Pending
Approved
Rejected
B-
City: Islamabad
Address: Ali Town
Status: approved
AB-
City: Islamabad
Address: Ali Town
Status: approved
O-
City: Lahore
Address: LDA AVENUE ONE
Status: approved
A+
City: Lahore
Request Details
Blood Group: A+
Address: LDA
City: Lahore
Selected Donor
Name: shahzad
PhoneNo.: 03064187674
Your Rating for the Donor
Feedback:
Blooad Received.
Thanks!
Rate Donor
Rate the Donor
Feedback (Optional)
0/1000
Submit Rating
Donor Detail
Name: aqeel
Email: test2@gmail.com
Phone No.: 03064187674
Address: LDA
City: Lahore
Blood Group: A+
Height: Unknown
Weight: Unknown
Age: Unknown
Selected
Rate this Donor
Request Details
Blood Group: O-
Address: LDA AVENUE ONE
City: Lahore
Available Donor
Chats
shahzad
Hi
18:49 PM
aqeel
Hi aqeel
1:10 AM
ali
test hy
19:47 PM
aqeel
hhh
23:13 PM
shahzad
Hello
Hi shahzad
How are you
hello
hello moeen
hello shahzad
hello shahzad
hello
new message 4:11
Type a message
Blood Group
A+ 9
A- 0
B+ 1
B- 0
O+ 1
O- 1
AB+ 0
AB- 0



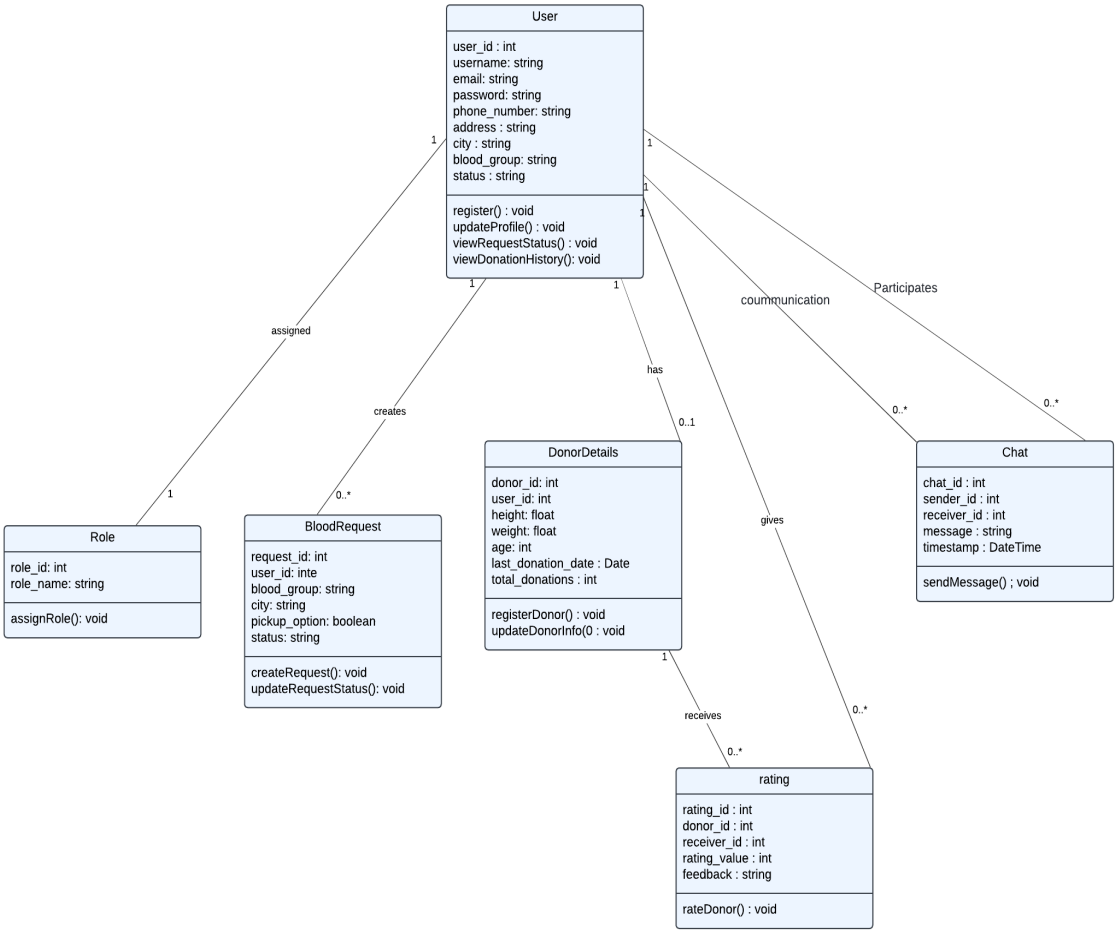
4.1. Architecture Diagram



4.2. Entity Relationship Diagram with data dictionary

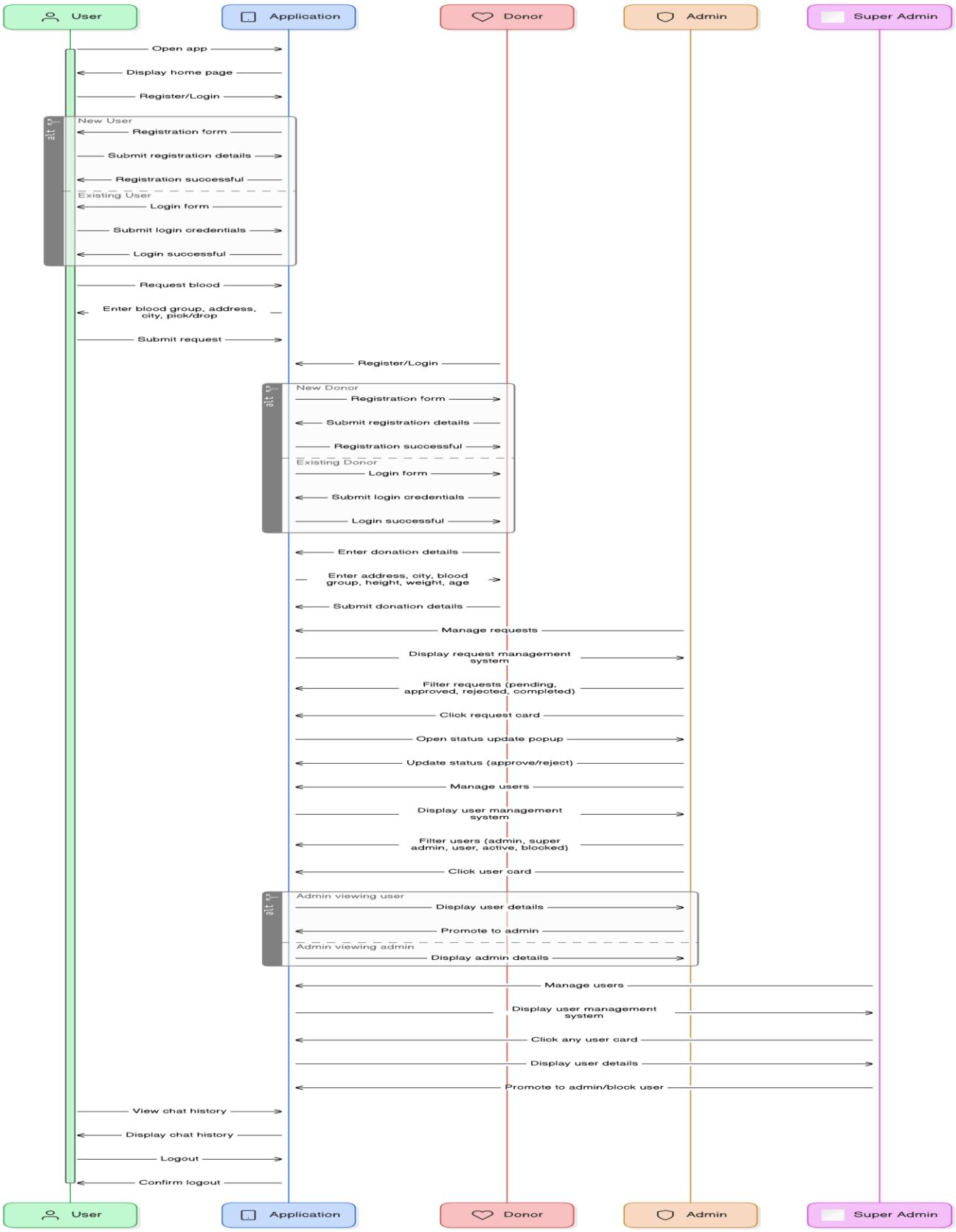


4.3. Class Diagram

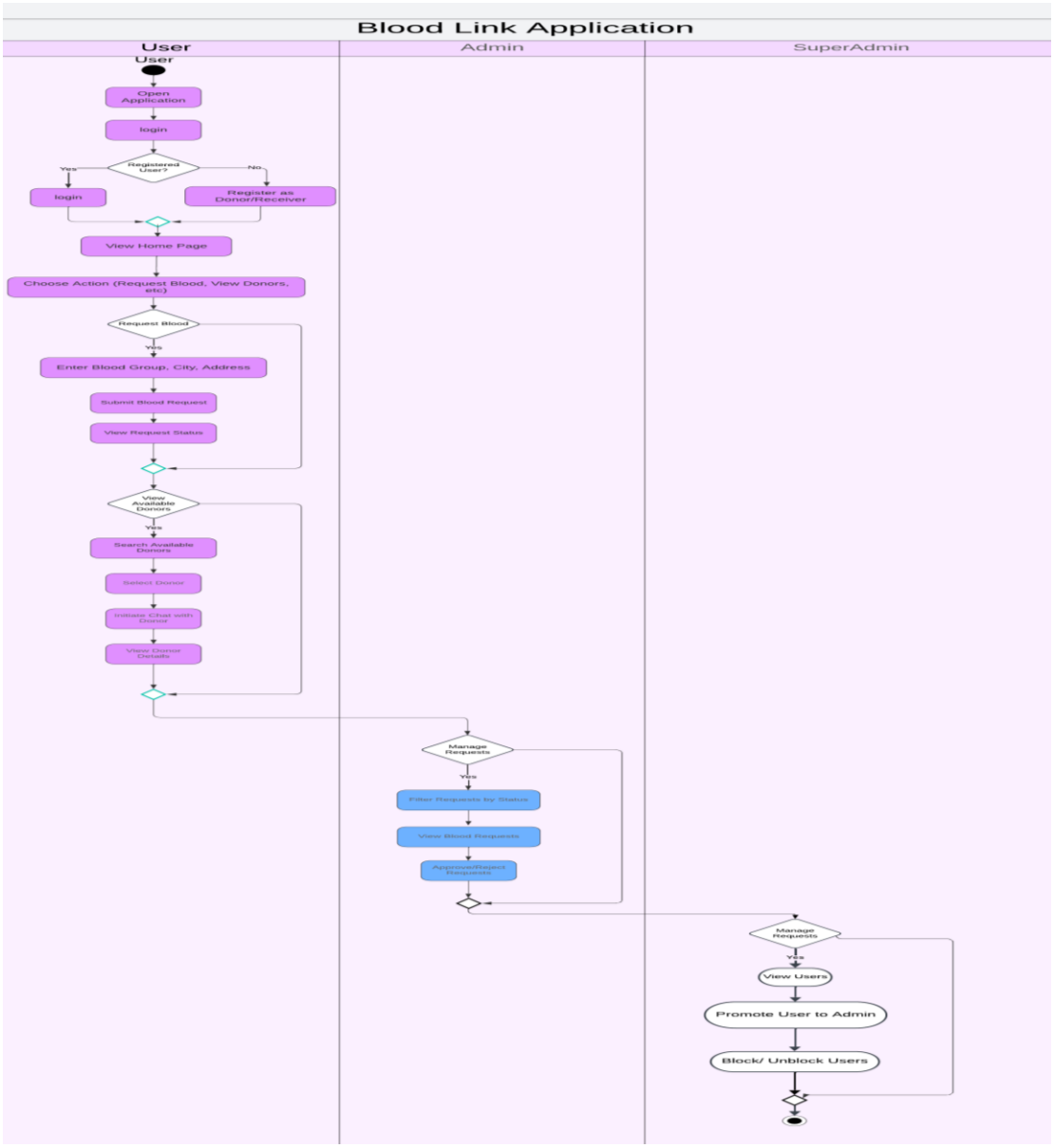


4.4. Sequence / Collaboration Diagram

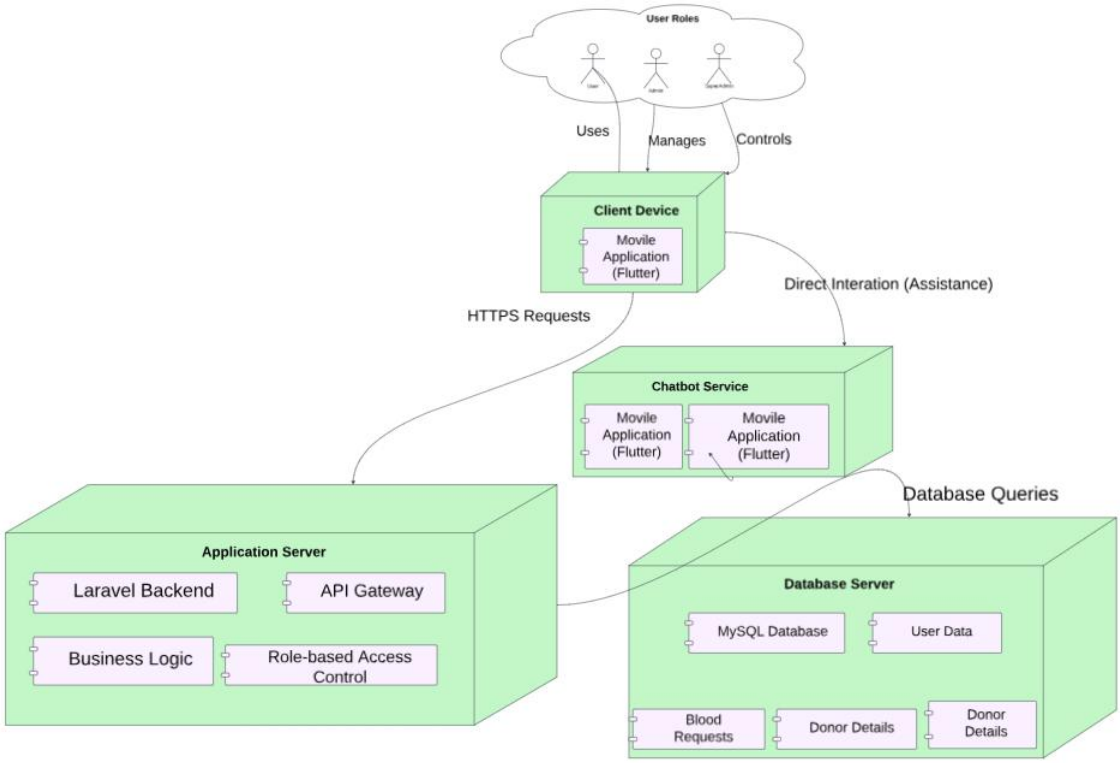
Blood Donation Application Process



4.5. Activity Diagram



4.6. Deployment Diagram



Chapter 5

Implementation

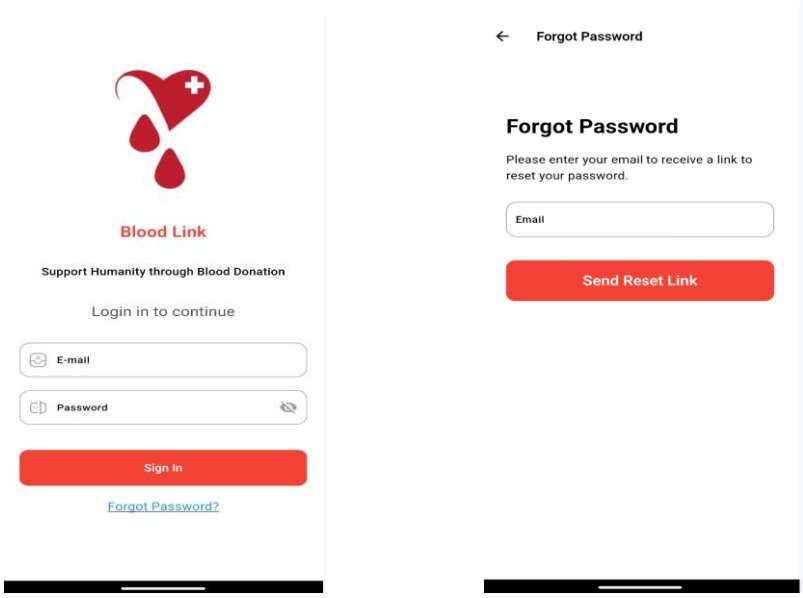
Chapter 5: Implementation

This chapter discusses in detail the procedures and actions regarding the technical execution of the Blood Link Project. The discussion will entail the environment of deployment, various components and libraries used, tools and methodologies applied, the control logic flow using pseudo codes, version control to preserve project integrity, and coding standards.

5.1. Important Flow Control/Pseudo codes

The flow of User Registration

The application manages the user registration process via a Google Form, which is linked to it directly. The information that the user will submit automatically gets saved and connected to the database of the application. Since all data collection and submission are handled through the Google Form, special coding was not made for user registration. This form will send a message to the administrator with details like the name of the user, e-mail address, and blood type for approval.



Purpose:

This widget enables users to select the city, input their current address, select the blood group, and choose whether they require pick-and-drop services through the PickBloodGroup widget. It's a user-friendly tool for requesting blood.

Key Components:

1. **State Variables:**
    a. **_selectedBloodGroup:** Selected blood group from the menu.

   b. **_selectedOption:** Boolean indicating user's pick-and-drop service preference.
   c. **currentAddress:** User's current address input managed by a text controller.

2. **Controllers Used:**
   a. **CityController:** Manages the selected state and city.
   b. **SelectBloodGroupController:** Manages the chosen blood group value.
   c. **BloodController:** Facilitates the blood request submission

3. **UI Elements:**
   - **Select Blood Group Dropdown:** Custom blood group selection widget.
   - **TextField for Address:** Field where users input their current address.
   - **City Dropdown:** Shows available cities and dynamically adjusts height based on selection.

Sr.No	Type	Give Blood To	Receive Blood From
1	A+	A+ AB+	A+ A- O+ O-
2	O+	A+ AB+ B+ O+	O+ O-
3	B+	AB+ B+	O+ O- B+ B-
4	AB+	AB+	EVERYONE
5	A-	A+ A- AB+ AB-	A- O-
6	O-	EVERYONE	O-
7	B-	B+ B- AB+ AB-	B- O-
8	AB-	AB+ AB-	AB- O- A- B-

   - **Pick-and-drop Radio Buttons:** Options for pick-and-drop services, with "Yes" or "No" choices.
   - **Submit Button:** Displays a loading indicator and initiates request submission after input validation.

4. **Interaction Flow:**
   - The app collects this data and users BloodController to submit a blood request when the submit button is tapped.

5.2. Components, Libraries, Web Services and stubs

Some of the components used in the implementation of the Blood Link Project are as follows:

The front end is done with Flutter. Laravel provides the required scalability for server-side handling, enabling real-time communication between donors and recipients through web services. Stubs are used in the initial development stage where the team is still implementing the full modules, acting as placeholders to simulate responses for processing other modules.

Main Components:

- Frontend

- Backend
- API
- Database

5.3. Deployment Environment

The deployment environment specifies the hardware, software, and network requirements needed for effective program operation. It outlines the settings necessary for maintaining system reliability and efficiency.

1. **Hardware Requirement:**
    - **Development Machine:** A current multi-code CPU, 100GB of free disk space, and at least 8GB of RAM.
    - **Mobile Device or Emulator:** An Android device, or emulator, with a minimum of 2GB of RAM is needed for testing purposes.
    - **Server:** For backend hosting, the server should have 40GB storage, 4GB of RAM, and a stable internet connection.
2. **Software Requirements:**
    - **Flutter SDK**
    - **Android Studio or Visual Studio Code**
    - **Java Development Kit (JDK)**

5.4. Tools and Techniques

1. **MVC Architecture Pattern:** This helps in organizing the structure of the project effectively by separating the user interface, business logic, and data management. This approach allowed for a cleaner, modular codebase, which is easier to maintain and extend.
2. **GetX for State Management:** Chosen because it users reactive programming and is lightweight, helping handle state management in Flutter applications way more efficiently. It eliminated the use of setState, which thus allowed for better performance and more scalable, which thus allowed for better performance and more scalable code.
3. **Laravel Back-end for API Integration:** It acted like a backend for maintaining API endpoints regarding user authentication. The integration provided smooth interaction for the client and the server through APIs in respect of login processes. APIs: used to perform real-time operations of fetching and updating user data; this makes the application more interactive and dynamic.
4. **Features of User Experience Onboarding Screen:** Implemented to provide a slick welcome for first-time users by showing the way around how to navigate the app and use its features. **Form Validation:** Extensive validation of login and registration forms along with feedback messages and user-friendly password toggle feature for good usability.
5. **Nodemailer for Email Verification:** Implemented for the increase in security by sending the 6-digit verification code at the time of registration and login. This way, the account of the user was verified and secured.

6. **Dynamic UI Component – Search Dropdown for Cities:** Dynamic component fetching sity data from API; also included a search bar for easy navigation through options. Dynamic Data Display: It displays real-time information about blood donors, organized according to blood groups for ease of understanding and application. **Toggle Between Donors and Receivers:** The flexible UI functionality of toggling between the list of donors and receivers in one interface enhances navigation and user control.

5.5. Best Practices / Coding Standards

Modular Code Structure: The project adheres to the MVC architecture, separating data management, user interface, and business logic. This maintains clen code organization and enhances scalability and maintainability.

Consistent State management: Using GetX ensures efficient state handling, offering better performance and a more reactive approach compared to setState.

Best Practices for API integration: Integration with Laravel backend APIs follows a standard format for data fetching and updates.

Enhanced user Experience: Best practices include features like a “No Internet Connection” screen and onboarding logic for smooth interactions and informative feedback.

Form validation and error handling: All input forms feature comprehensive validation to prevent invalid entries. Visual cues and error messages enhance usability and adhere to secure data practices.

Secure communication: Nodemailer facilitates email verification by sending a 6-digit code, adding an extra layer of user authentication.

Dynamic, searchable data display: A searchable dropdown for cities and dynamic blood donor data display via API responses address to best practices for handling large datasets with user-friendly interfaces.

Flexible, switchable views: The toggle feature for donor and receiver views on a single page exemplifies and interactive, adaptive UI approach, in line with moder standards.

5.6. Version Control

1. **Tool Used:** Git was used as the version control system to manage the project. This tool was important to track changes, managing different versions, and collaborate effectively.
2. **Repository Platform:** The project code was hosted on GitHub. This centralizes the platform for version control, issue tracking, and collaborative development.
3. **Commit Strategy:** Committing is done frequently with a structured naming convention describing the changes done.
4. **Branching Model:** The project followed Git Flow in maintaining a main, stable branch while developing separate branches with features. All these branches allowed for isolation of developmental work as integration was strictly taken in when it’s after appropriate testing. **Hotfix branches used** the immediate fixes provided for the main code when a pressing bug or some problem needed to be urgently solved.

5. **Collaboration:** If the project had been collaboratively developed, then the use of pull requests PRs was available. A pull and push request were available. A pull and push request are used whereby code is pulled for a merge in the main branch. This code gets reviewed and tested before the merger into the code base.
6. **Change Logs:** Release notes were maintained about critical updates, which used to trace the history about the history about the major changes, features introduced, or bugs.
7. **Backup and Security:** The repository could be accessed only by allowed users since access controls can secure it, thus excluding any unauthorized changes. There are always a periodic backup. So, data loss was kept to the minimum.
8. **Tagging and Versioning:** A 2.470(2) marked a kind of milestone.

Chapter 6

Testing and Evaluation

Chapter 6: Testing and Evaluation

6.1. Use Case Testing for Blood Link App

1. **User Registration and Login**
    - **Objective:** Ensure that a new user can't register their account successfully, but the registered users can log in successfully.
    - **Test Steps:**
        1. On opening the app select the registration option.
        2. Input the required details.
        3. Submit the form and check for a confirmation message.
        4. Log in using the registration credential.
2. **Creating a Blood Donation Request**
    - **Objective:** Ensure that a recipient can create a blood request and that it appears in the donor's view.
    - **Test Steps:**
        1. Log in as a recipient find the blood request section.
        2. Select blood type, location and chosen urgency of the request submit your request test listing.
3. **Viewing and Responding to Requests (Donor)**
    - **Objective:** Verify that a donor can view blood requests available and respond to that request.
    - **Test Steps:**
        1. Log in as a donor and go to the blood request section.
        2. Browse and scroll through your lists of requests then select one to which you want to respond.
        3. Acceptance for the recipient response to the request.
4. **Updating Profile Information**
    - **Objective:** Test will verify if a user would be able to update the profile information including their contact information and status availability.
    - **Test Steps:**
        1. Accessing and logging into the profile.
        2. Changing the details of the profile, like the contact number, blood group, or the availability.
        3. Saving changes and confirm the same are reflected.

5. **Admin Management of Requests and Users.**
    - **Objective:** Validated that the admin can manage the requests by approving or rejecting it, along with the related user action.
    - **Test Steps:**
        1. Log in as an admin and open the management dashboard.
        2. View, accept, or reject user requests. Perform user management operations as well.
        3.

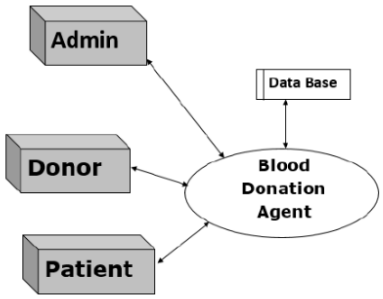
6.2. Equivalence partitioning

1. **Blood Type Selection**
    - **Valid Partitions:** Acceptable blood types (A+, A-, B+, B-, AB+, AB-, O+, O-)
    - **Invalid Partitions:** All other than the blood types listed above. It may be that you have tried to insert numbers, special characters, or any text other than those that are supported like “A++”.
    - **Expected Outcome:** Only proper blood types are accepted, and the wrong ones should be rejected with an error message.

2. **User Age Input (if applicable)**
    - **Valid Partitions:** Ages 18-65 (for eligible blood donors)
    - **Invalid Partitions:** Ages below 18, above 65, or non-numeric values (e.g., “-1”, “70”, “twenty”)
    - **Expected Outcome:** Only withing the valid age range users should be allowed to continue their process of registering as a donor. An error message shold appear for all invalid inputs.
3. **Contact Number**
    - **Valid Partitions:** Correctly formatted phone numbers (e.g,. ‘123-456-7890” or “+1 123 456 7890’).
    - **Invalid Partitions:** Text input, numbers with fewer or extra digits, or special characters (e.g., “123”, “+1234@567”, “contact123”)**.**
    - **Expected Outcome:** The well-
4. **Donation History Entries (Viewable for Donors)**
    - **Valid Partitions:** Logged donation records with complete details (e.g., date, recipient, blood type).
    - **Invalid Partitions:** Missing or incomplete data (e.g., a record missing the date or recipient information).
    - **Expected Outcome:** Only complete records should appear in the history view. Incomplete or corrupted entries should not be displayed.

6.3. Data flow testing

Data flow testing conducted on a blood donation app would ensure that once information, such as that concerning the users of their blood types, gets input and captured in the application, the data can be retrieved and correct through all modules that concern it.



6.4. Unit testing

Unit testing would entail testing the individual functions in the app, checking for instance whether the age eligibility function identifies eligible and ineligible ages.

6.5. Integration testing

Testing the connection between the user registration module and the database would ensure that the information entered by the user was stored and could be retrieved.

6.6. Performance testing

Checked how blood donation application performs by considering a few users accessing the donation database at the same time.

6.7. Stress Testing

A high rate of user logins or database queries would be performed to calculate whether the application will be strong enough to resist crashing.

6.8. Coding

APP

```
import
'package:blood_donation/features/authentication/views/verification/verification_screen.dart';
import 'package:blood_donation/features/blood/homepage/views/homepage.dart';
import 'package:blood_donation/utils/theme/theme.dart';
import 'package:get/get.dart';
import 'package:blood_donation/utils/app_routes.dart';
import 'package:flutter/material.dart';
import 'features/authentication/views/login/login_screen.dart';
import 'features/authentication/views/onboarding/onboarding_screen.dart';
import 'features/authentication/views/signup/signup.dart';
import 'features/authentication/views/splash/splash_screen.dart';
class MyApp extends StatelessWidget {
 const MyApp({super.key});

 @override
 Widget build(BuildContext context) {
  return GetMaterialApp(
    color: Colors.white,
    debugShowCheckedModeBanner: false,
    initialRoute: AppRoutes.splash,
    theme:  AppTheme.lightTheme,
    getPages: [
     GetPage(name: AppRoutes.splash, page: () => const SplashScreen()),
     GetPage(name: AppRoutes.onboarding, page: () => const OnBoardingScreen()),
     GetPage(name: AppRoutes.login, page: () => const LoginScreen()),
     GetPage(name: AppRoutes.register, page: () => const RegisterScreen()),
     GetPage(name: AppRoutes.homepage, page: () =>  const Homepage()),
     GetPage(name: AppRoutes.verification, page: () => VerificationScreen(),),

     // GetPage(name: AppRoutes.request, page: () => const Request(),),
     // Add more routes here
    ],

  );
 }
```

NEVIGATION

```
import 'package:blood_donation/features/blood/homepage/views/homepage.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:iconsax/iconsax.dart';

import 'features/blood/donor/views/donor.dart';
import 'features/blood/profile/views/profile.dart';
import 'features/blood/request/views/request.dart';

class NavigationMenu extends StatelessWidget {
 const NavigationMenu({Key? key}) : super(key: key);
 @override
 Widget build(BuildContext context) {
  final controller = Get.put(NavigationController());

  return Scaffold(
   bottomNavigationBar: Obx(
       () => ClipRRect(
      borderRadius: const BorderRadius.only(
       topLeft: Radius.circular(30), // Rounded top corners
       topRight: Radius.circular(30),
      ),
      child: NavigationBarTheme(
       data: NavigationBarThemeData(
        backgroundColor: const Color(0XFFE8E3E3),
        indicatorColor: Colors.white.withOpacity(0.1),
        labelTextStyle: WidgetStateProperty.resolveWith<TextStyle?>(
            (states) {
          if (states.contains(WidgetState.selected)) {
           return const TextStyle(
            color: Colors.red, // Selected text color
            fontWeight: FontWeight.bold,
           );
          } else {
           return const TextStyle(
            color: Colors.black, // Unselected text color
            fontWeight: FontWeight.normal,
```

```
              );
            }
          },
        ),
        iconTheme: WidgetStateProperty.all(
          const IconThemeData(color: Colors.black), // Keep icon color black
        ),
      ),
      child: NavigationBar(
        height: 80,
        elevation: 0,
        selectedIndex: controller.selectedIndex.value,
        onDestinationSelected: (index) =>
        controller.selectedIndex.value = index,
        destinations: const [
          NavigationDestination(
            icon: Icon(Iconsax.home),
            label: 'Home',
          ),
          NavigationDestination(
            icon: Icon(Iconsax.shop),
            label: 'Request',
          ),
          NavigationDestination(
            icon: Icon(Iconsax.heart),
            label: 'Donor',
          ),
          NavigationDestination(
            icon: Icon(Iconsax.user),
            label: 'Profile',
          ),
        ],
      ),
    ),
  ),
  body: Obx(() => controller.screen[controller.selectedIndex.value]),
);
```

```
  }
}

class NavigationController extends GetxController {
  final Rx<int> selectedIndex = 0.obs;
  final screen = [const Homepage(), PickBloodGroup(), const CreateDonor(), const Profile()];
}
```

LOGIN PAGE

```
import 'package:flutter/material.dart';
class TFormDivider extends StatelessWidget {
  const TFormDivider({super.key,required this.dividerText});
  final String? dividerText;

  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Flexible(
          child: Divider(
            color: Colors.grey,
            thickness: 0.5,
            indent: 60, //space left to right
            endIndent: 5,
          ),
        ),
        Text(
          dividerText!,
          style: Theme.of(context).textTheme.labelMedium,
        ),
        const Flexible(
          child: Divider(
            color: Colors.grey,
            thickness: 0.5,
            indent: 5, //space left to right
```

```
          endIndent: 60,
        ),
      ),
    ],
  );
 }
}
```

PROFILE_INFOMATION_WIDGET

```
import 'package:flutter/material.dart';

class ProfileInformationWidget extends StatelessWidget {
 final String label;
 final String value;

 const ProfileInformationWidget({
  super.key,
  required this.label,
  required this.value,
 });

 @override
 Widget build(BuildContext context) {
  return Padding(
   padding: const EdgeInsets.symmetric(vertical: 8.0),
   child: Row(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
     Expanded(
      flex: 2,
      child: Text(
       label,
       style: TextStyle(
        fontSize: 18,
        color: Colors.grey[700], // Label in grey
       ),
```

```
          ),
        ),
        const SizedBox(width: 16),
        Expanded(
          flex: 3,
          child: Text(
            value,
            style: const TextStyle(
              fontSize: 16,
              fontWeight: FontWeight.bold,
              color: Colors.black, // Value in bold black
            ),
          ),
        ),
      ],
    ),
  );
 }
}
```

REQUEST_DETAILS

```
import 'package:flutter/material.dart';

class LabelValueWidget extends StatelessWidget {
 final String label;
 final String value;

 const LabelValueWidget({super.key, required this.label, required this.value});

 @override
 Widget build(BuildContext context) {
  return RichText(
   text: TextSpan(
    children: [
     TextSpan(
      text: '$label: ',
      style: TextStyle(
```

```
            fontSize: 16,
            color: Colors.grey[700], // Label in grey
          ),
        ),
        TextSpan(
          text: value,
          style: const TextStyle(
            fontSize: 16,
            fontWeight: FontWeight.bold,
            color: Colors.black, // Value in bold black
          ),
        ),
      ],
    ),
  );
 }
}
```

SNACKBARUTIL

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';

class SnackbarUtil {
 static void showErrorSnackbar(String title, String message) {
  Get.snackbar(
   title,
   message,
   backgroundColor: Colors.red,
   colorText: Colors.white,
   snackPosition: SnackPosition.BOTTOM,
  );
 }

 static void showSuccessSnackbar(String title, String message) {
  Get.snackbar(
   title,
   message,
```

```
    backgroundColor: Colors.green,
    colorText: Colors.white,
    snackPosition: SnackPosition.BOTTOM,
   );
  }
}
```

DONOR:

```
import
'package:blood_donation/features/blood/donor/views/widgets/blood_group_dropdown.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';

import '../../../../common/SnackbarUtil.dart';
import '../../../../utils/constants/text_strings.dart';
import '../../request/controllers/city_controller.dart';
import '../../request/views/widgets/city_dropdown.dart';
import '../controllers/blood_group_controller.dart';
import '../controllers/donor_controller.dart';
class CreateDonor extends StatelessWidget {
 const CreateDonor({super.key});

 @override
 Widget build(BuildContext context) {
  double screeWidth = MediaQuery.of(context).size.width;
  double screeHeight = MediaQuery.of(context).size.height;
  final DonorController donorController = Get.put(DonorController());
  final BloodGroupController bloodGroupController = Get.put(BloodGroupController());
  final CityController cityController = Get.put(CityController());
  return SafeArea(
   child: Scaffold(
    appBar: AppBar(
     title: const Center(child:  Text("Become a Donor")),
    ),
    body: SingleChildScrollView(
     child: Column(
      children: [
       SizedBox(
```

```
            height: screeHeight*0.07,
          ),
          Padding(
            padding:  EdgeInsets.symmetric(horizontal: screeWidth*0.03),
            child: SizedBox(
              height: screeHeight*0.1,
              child: TextField(
                controller: donorController.currentAddress,
                decoration: InputDecoration(
                  hintText: 'Enter current address',
                  suffixIcon: const Icon(Icons.location_on),
                  border: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(10),
                  ),
                ),
              ),
            ),
          ),
          Padding(
            padding: EdgeInsets.symmetric(horizontal: screeWidth*0.03),
            child: Obx(() {
              return SizedBox(
                  height: cityController.isCitySelected.value ? screeHeight*0.085 : screeHeight*0.35,
// Set height to 1 if city is selected, else 500
                child: CityDropdown(),
              );
            }),
          ),
          Padding(
            padding:  EdgeInsets.symmetric( horizontal: screeWidth*0.03),
            child: Obx(() {
              return SizedBox(
                    height: bloodGroupController.isBloodGroupSelected.value ? screeHeight*0.085 :
screeHeight*0.35, // Set height to 1 if city is selected, else 500
                child: const BloodGroupDropdown(),
              );
            }),
          ),
```

```
          Padding(
           padding:  EdgeInsets.symmetric(horizontal: screeWidth*0.03),
           child: SizedBox(
            height: screeHeight*0.1,
            child: TextField(
             controller: donorController.height,
             decoration: InputDecoration(
              hintText: 'Enter Height',
              suffixIcon: const Icon(Icons.location_on),
              border: OutlineInputBorder(
               borderRadius: BorderRadius.circular(10),
              ),
             ),
            ),
           ),
          ),
          Padding(
           padding:  EdgeInsets.symmetric(horizontal: screeWidth*0.03),
           child: SizedBox(
            height: screeHeight*0.1,
            child: TextField(
             controller: donorController.weight,
             decoration: InputDecoration(
              hintText: 'Enter Weight',
              suffixIcon: const Icon(Icons.location_on),
              border: OutlineInputBorder(
               borderRadius: BorderRadius.circular(10),
              ),
             ),
            ),
           ),
          ),
          Padding(
           padding:  EdgeInsets.symmetric(horizontal: screeWidth*0.03),
           child: SizedBox(
            height: screeHeight*0.1,
            child: TextField(
             controller: donorController.age,
```

```
              decoration: InputDecoration(
                hintText: 'Enter Age',
                suffixIcon: const Icon(Icons.location_on),
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(10),
                ),
              ),
            ),
          ),
          Obx(
              () => GestureDetector(
            onTap: (){
              debugPrint("address ${donorController.currentAddress.text}");
              debugPrint("city ${cityController.SelectedCity}");
              debugPrint("blood Group ${bloodGroupController.SelectedBloodGroup}");
              debugPrint("address ${donorController.height.text}");
              debugPrint("address ${donorController.weight.text}");
              debugPrint("address ${donorController.age.text}");
                  bloodGroupController.createDonor(bloodGroupController.SelectedBloodGroup,
donorController.currentAddress.text,  cityController.SelectedCity,  donorController.height.text,
donorController.weight.text, donorController.age.text);

              SnackbarUtil.showErrorSnackbar("Request Failed", "Please Fill al.");

            },
            child: Container(
              height: screeHeight*0.07,
              width: screeWidth*0.94,
              decoration: BoxDecoration(
                color: Colors.red, borderRadius: BorderRadius.circular(10)),
              child: Center(
                      child:  cityController.isLoading.value?const  CircularProgressIndicator(color:
Colors.white,):  Text(TextStrings.donorCreate,style: TextStyle(
                  color: Colors.white, fontSize: screeWidth*0.05
                ),)),),
            ),
          ),
```

```
          ],
        ),
      ),
    ),
  );
 }
}


//                                                                                                    import
'package:blood_donation/features/blood/donor/views/widgets/blood_group_dropdown.dart';
// import 'package:flutter/material.dart';
// import 'package:get/get.dart';
//
// import '../../../../common/SnackbarUtil.dart';
// import '../../../../utils/constants/text_strings.dart';
// import '../../request/controllers/city_controller.dart';
// import '../../request/views/widgets/city_dropdown.dart';
// import '../controllers/blood_group_controller.dart';
// import '../controllers/donor_controller.dart';
// class CreateDonor extends StatelessWidget {
//   const CreateDonor({super.key});
//
//   @override
//   Widget build(BuildContext context) {
//     double screeWidth = MediaQuery.of(context).size.width;
//     double screeHeight = MediaQuery.of(context).size.height;
//     final DonorController donorController = Get.put(DonorController());
//     final BloodGroupController bloodGroupController = Get.put(BloodGroupController());
//     final CityController cityController = Get.put(CityController());
//     return SafeArea(
//       child: Scaffold(
//         appBar: AppBar(
//           title: const Center(child:  Text("Become a Donor")),
//         ),
//         body: SingleChildScrollView(
//           child: Column(
//             children: [
```

```
//            SizedBox(
//              height: screeHeight*0.07,
//            ),
//            Padding(
//              padding:  EdgeInsets.symmetric(horizontal: screeWidth*0.03),
//              child: SizedBox(
//                height: screeHeight*0.1,
//                child: TextField(
//                  controller: donorController.currentAddress,
//                  decoration: InputDecoration(
//                    hintText: 'Enter current address',
//                    suffixIcon: const Icon(Icons.location_on),
//                    border: OutlineInputBorder(
//                      borderRadius: BorderRadius.circular(10),
//                    ),
//                  ),
//                ),
//              ),
//            ),
//            Padding(
//              padding: EdgeInsets.symmetric(horizontal: screeWidth*0.03),
//              child: Obx(() {
//                return SizedBox(
//                        height:  cityController.isCitySelectedForDonor.value  ?  screeHeight*0.085  :
screeHeight*0.35, // Set height to 1 if city is selected, else 500
//                  child: CityDropdown(pageType: 'donor',),
//                );
//              }),
//            ),
//            Padding(
//              padding:  EdgeInsets.symmetric( horizontal: screeWidth*0.03),
//              child: Obx(() {
//                return SizedBox(
//                    height: bloodGroupController.isBloodGroupSelected.value ? screeHeight*0.085 :
screeHeight*0.35, // Set height to 1 if city is selected, else 500
//                  child: BloodGroupDropdown(),
//                );
//              }),
```

```
//            ),
//            Padding(
//              padding:  EdgeInsets.symmetric(horizontal: screeWidth*0.03),
//              child: SizedBox(
//                height: screeHeight*0.1,
//                child: TextField(
//                  controller: donorController.height,
//                  decoration: InputDecoration(
//                    hintText: 'Enter Height',
//                    suffixIcon: const Icon(Icons.location_on),
//                    border: OutlineInputBorder(
//                      borderRadius: BorderRadius.circular(10),
//                    ),
//                  ),
//                ),
//              ),
//            ),
//            Padding(
//              padding:  EdgeInsets.symmetric(horizontal: screeWidth*0.03),
//              child: SizedBox(
//                height: screeHeight*0.1,
//                child: TextField(
//                  controller: donorController.weight,
//                  decoration: InputDecoration(
//                    hintText: 'Enter Weight',
//                    suffixIcon: const Icon(Icons.location_on),
//                    border: OutlineInputBorder(
//                      borderRadius: BorderRadius.circular(10),
//                    ),
//                  ),
//                ),
//              ),
//            ),
//            Padding(
//              padding:  EdgeInsets.symmetric(horizontal: screeWidth*0.03),
//              child: SizedBox(
//                height: screeHeight*0.1,
//                child: TextField(
```

```
//                 controller: donorController.age,
//                 decoration: InputDecoration(
//                   hintText: 'Enter Age',
//                   suffixIcon: const Icon(Icons.location_on),
//                   border: OutlineInputBorder(
//                     borderRadius: BorderRadius.circular(10),
//                   ),
//                 ),
//               ),
//             ),
//           ),
//           Obx(
//               () => GestureDetector(
//             onTap: (){
//               debugPrint("address ${donorController.currentAddress.text}");
//               debugPrint("city ${cityController.SelectedCityforDonor}");
//               debugPrint("blood Group ${bloodGroupController.SelectedBloodGroup}");
//               debugPrint("address ${donorController.height.text}");
//               debugPrint("address ${donorController.weight.text}");
//               debugPrint("address ${donorController.age.text}");
//                 bloodGroupController.createDonor(bloodGroupController.SelectedBloodGroup,
donorController.currentAddress.text,                cityController.SelectedCityforDonor,
donorController.height.text, donorController.weight.text, donorController.age.text);
//               SnackbarUtil.showErrorSnackbar("Request Failed", "Please Fill al.");
//
//             },
//             child: Container(
//               height: screeHeight*0.07,
//               width: screeWidth*0.94,
//               decoration: BoxDecoration(
//                 color: Colors.red, borderRadius: BorderRadius.circular(10)),
//               child: Center(
//                       child:  cityController.isLoading.value?const  CircularProgressIndicator(color:
Colors.white,):  Text(TextStrings.donorCreate,style: TextStyle(
//                   color: Colors.white, fontSize: screeWidth*0.05
//                 ),)),),
//             ),
//           ),
```

```
//           ],
//         ),
//       ),
//     ),
//   );
//  }
// }
```

BLOOD_GROUP_DROPDOWN

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import '../../controllers/blood_group_controller.dart';
import '../../models/blood_group_model.dart';

class BloodGroupDropdown extends StatefulWidget {
  const BloodGroupDropdown({super.key});

  @override
  _BloodGroupDropdownState createState() => _BloodGroupDropdownState();
}

class _BloodGroupDropdownState extends State<BloodGroupDropdown> {
  final BloodGroupController bloodGroupController = Get.put(BloodGroupController());
  var filteredBloodGroupList = <BloodGroupModel>[].obs;
  final FocusNode _focusNode = FocusNode();
  bool showSuggestions = true;

  @override
  void initState() {
    super.initState();
    bloodGroupController.fetchBloodGroup(); // Fetch cities when the widget is initialized
    filteredBloodGroupList.value = bloodGroupController.bloodGroupList; // Initially show all
cities

    _focusNode.addListener(() {
      if (_focusNode.hasFocus &&
bloodGroupController.searchBloodGroupController.text.isNotEmpty) {
        setState(() {
          showSuggestions = true;

        });
      }
    });
  }
```

```
  void filterBloodGroup(String query) {
    if (query.isEmpty) {
      filteredBloodGroupList.value = bloodGroupController.bloodGroupList; // Show all cities if
query is empty
    } else {
      filteredBloodGroupList.value = bloodGroupController.bloodGroupList
          .where((BloodGroup) =>
BloodGroup.name.toLowerCase().contains(query.toLowerCase()))
          .toList(); // Filter the cities based on query
    }
  }

  @override
  Widget build(BuildContext context) {
    double screeHeight = MediaQuery.of(context).size.height;
    return Scaffold(
      body: Obx(() {
        if (bloodGroupController.isLoading.value) {
          return const Center(child: CircularProgressIndicator());
        } else {
          return SingleChildScrollView(
            child: Column(
              children: [
                // Search Bar
                TextFormField(
                  controller: bloodGroupController.searchBloodGroupController,
                  decoration: const InputDecoration(
                    hintText: 'Search Blood Group',
                    border: OutlineInputBorder(),
                    prefixIcon: Icon(Icons.search),
                  ),
                  onChanged: (value) {
                    filterBloodGroup(value); // Filter the city list

                    if (value.isEmpty) {
                      bloodGroupController.clearSelection(); // Clear the city selection if input is cleared
                    }
                  },
                  onTap: () {
                    setState(() {
                      showSuggestions = true;
                    });
                  },
                ),
                // Show suggestions only if there are cities and showSuggestions is true
```

```
            if (filteredBloodGroupList.isNotEmpty && showSuggestions)
              SizedBox(
                height: screeHeight*0.3, // Fixed height for the container that shows cities
                child: ListView.builder(
                  shrinkWrap: true,
                  physics: const ClampingScrollPhysics(),
                  itemCount: filteredBloodGroupList.length,
                  itemBuilder: (context, index) {
                    BloodGroupModel BloodGroup = filteredBloodGroupList[index];
                    return ListTile(
                      title: Text(BloodGroup.name),
                      onTap: () {
                        bloodGroupController.searchBloodGroupController.text = BloodGroup.name;
                        bloodGroupController.selectBloodGroup(BloodGroup); // Mark city as selected
                        setState(() {
                          showSuggestions = false;
                          bloodGroupController.SelectedBloodGroup = BloodGroup.name;
                        });
                        debugPrint('Selected city: ${BloodGroup.name}');
                      },
                    );
                  },
                ),
              ),
            ],
          ),
        );
      }
    }),
  );
 }
}
```

BLOOD_GROUP_MODELS

```
class BloodGroupModel {
  final int id;
  final String name;

  BloodGroupModel({required this.id, required this.name});

  // Factory constructor to parse JSON data
  factory BloodGroupModel.fromJson(Map<String, dynamic> json) {
    return BloodGroupModel(
      id: json['id'],
```

```
      name: json['name'],
    );
  }
}
```

BLOOD_GROUP_CONTROLLER

```
import 'package:flutter/cupertino.dart';
import 'package:get/get.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

import '../../../../api_constants.dart';
import '../../../../services/donor_create_api_servce.dart';
import '../models/blood_group_model.dart';

class BloodGroupController extends GetxController {
  var bloodGroupList = <BloodGroupModel>[].obs; // Observable list for cities
  var isLoading = false.obs; // Observable for loading state
  var isBloodGroupSelected = false.obs; // To track if a city is selected
  String BloodGroup = '';
  String SelectedBloodGroup = '';
  final TextEditingController searchBloodGroupController = TextEditingController(); //
TextEditingController to handle text field input

  @override
  void onInit() {
    super.onInit();
    fetchBloodGroup(); // Fetch cities when the controller is initialized
  }
  void selectBloodGroup(BloodGroupModel city) {
    isBloodGroupSelected.value = true; // Set to true when a city is selected
  }

  void clearSelection() {
    isBloodGroupSelected.value = false; // Reset selection
  }
  // Fetch cities from API
  Future<void> fetchBloodGroup() async {
    try {
      isLoading.value = true;
      final response = await http.get(Uri.parse('${ApiConstants.baseUrl}/api/blood-groups'));

      if (response.statusCode == 200) {
        var jsonData = jsonDecode(response.body);
        // Parse the data and add it to the list
```

```
        var BloodGroupData = jsonData['data'] as List;
        bloodGroupList.value = BloodGroupData.map((bloodGroup) =>
BloodGroupModel.fromJson(bloodGroup)).toList();
      } else {
        // Handle errors
        Get.snackbar("Error", "Failed to load cities");
      }
    } catch (e) {
      Get.snackbar("Error", e.toString());
    } finally {
      isLoading.value = false;
    }
  }
  Future<void> createDonor(String bloodGroup, String address, String city,String height,String
weight,String age,) async {
    isLoading.value = true;

    // Call the Request method and don't attempt to capture the result as it returns void
    await DonorCreateApiService.createDonorRequest(bloodGroup,address,
city,height,weight,age);

    isLoading.value = false;
  }
}
```

DONOR_CONTROLER

```
import 'package:flutter/cupertino.dart';
import 'package:get/get.dart';

class DonorController extends GetxController{
  TextEditingController currentAddress = TextEditingController();
  TextEditingController height = TextEditingController();
  TextEditingController weight = TextEditingController();
  TextEditingController age = TextEditingController();
}
```

HOME_PAGE:

```
import 'package:blood_donation/features/blood/homepage/views/widgets/home_appbar.dart';
import
'package:blood_donation/features/blood/homepage/views/widgets/receivers_and_donors.dart';
import 'package:blood_donation/features/blood/homepage/views/widgets/top_donors_view.dart';
import 'package:blood_donation/features/blood/homepage/views/widgets/youtube_video.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';
```

```
import 'package:get_storage/get_storage.dart';
import '../controllers/top_donor_controller.dart';
import '../controllers/total_donor_and_receiver_controller.dart';
import '../controllers/youtube_video_controller.dart';
class Homepage extends StatefulWidget {
  const Homepage({super.key});

  @override
  State<Homepage> createState() => _HomepageState();
}

class _HomepageState extends State<Homepage> {
  final VideoController controller = Get.put(VideoController());
  final TopDonorController topDonorController = Get.put(TopDonorController());
  final TotalDonorController totalDonorController = Get.put(TotalDonorController());
  final storage = GetStorage();

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    // Fetch your data and update the state here
    // Refresh logic here
    topDonorController.onInit();
    totalDonorController.onInit();
  }

  @override
  Widget build(BuildContext context) {
    double screenWidth = MediaQuery.of(context).size.width;
    double screenHeight = MediaQuery.of(context).size.height;
    final token = storage.read('token');
    debugPrint("Received Bearer Token: $token");
    return Scaffold(
      body: SingleChildScrollView(
        child: Column(
          children: [
            HomeAppBar(width: screenWidth , height: screenHeight * 0.5),
            SizedBox(height: screenHeight * 0.03),
             Padding(
              padding: EdgeInsets.only(left: screenWidth*0.04, right: screenWidth*0.04),
              child: const YoutubeVideo(),
            ),
            SizedBox(height: screenHeight * 0.02),
            Padding(
              padding: const EdgeInsets.all(8.0),
              child: Column(
```

```
              children: [
                const ReceiversAndDonors(),
                SizedBox(height: screenHeight * 0.03),
                TopDonorsView(),
              ],
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

DONOR_CARD

```
import 'package:flutter/material.dart';
import '../../models/top_donor_model.dart';
class DonorCard extends StatelessWidget {
 final Donor donor;

 const DonorCard({super.key, required this.donor});

 @override
 Widget build(BuildContext context) {
  return Container(
    width: MediaQuery.of(context).size.width,
    margin: const EdgeInsets.symmetric(horizontal: 5.0),
    decoration: BoxDecoration(
      color: Colors.grey[200],
      borderRadius: BorderRadius.circular(10),
    ),
    child: Padding(
      padding: const EdgeInsets.all(8),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Center(
            child: Text(
              donor.name,
              style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
            ),
          ),
          const SizedBox(height: 5),
          Padding(
            padding: const EdgeInsets.only(left: 20,right: 100),
```

```
            child: Row(
              children: [
                const Text(
                  'Blood Group: ',
                  style: TextStyle(fontSize: 14),
                ),
                const Spacer(),
                Text(donor.bloodGroup)
              ],
            ),
          ),
          const SizedBox(height: 10),
          Padding(
            padding: const EdgeInsets.only(left: 20,right: 100),
            child: Row(
              children: [
                const Text(
                  'Donations: ',
                  style: TextStyle(fontSize: 14),
                ),
                const Spacer(),
                Text('${donor.donationCount }')
              ],
            ),
          ),
          const SizedBox(height: 5),
          Padding(
            padding: const EdgeInsets.only(left: 20,right: 100),
            child: Row(
              children: [
                const Text(
                  'Rating: ',
                  style: TextStyle(fontSize: 14),
                ),
                const Spacer(),
                Text('${donor.rating}')
              ],
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

RECIVER_AND_DONOR:

```
import
'package:blood_donation/features/blood/homepage/views/widgets/total_blood_donors_and_receivers.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';

import '../../../../../utils/constants/image_strings.dart';
import '../../controllers/total_donor_and_receiver_controller.dart';

class ReceiversAndDonors extends StatelessWidget {
  const ReceiversAndDonors({super.key});

  @override
  Widget build(BuildContext context) {
    double screenWidth = MediaQuery.of(context).size.width;
    double screenHeight = MediaQuery.of(context).size.height;
    final TotalDonorController totalDonorController = Get.put(TotalDonorController());
    return SizedBox(
      width: double.infinity,
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          Expanded(
            child: GestureDetector(
              onTap: () {
                try {
                  Get.to(const TotalBloodDonorsAndReceivers(showDonors: false,));
                } catch (e) {
                  debugPrint('Error navigating to totalBloodDonors: $e');
                }
              },
              child: Container(
                height: screenHeight*0.08,
                decoration: const BoxDecoration(
                  color: Color(0XFFF5F3F3),
                  borderRadius: BorderRadius.all(Radius.circular(20)),
                ),
                child: Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Row(
                    children: [
```

```
                Image.asset(
                  ImageStrings.donorPerson, // Replace with your profile image path
                  width: screenWidth*0.09,
                  height: screenHeight*0.045,
                ),
                 SizedBox(width: screenWidth*0.03),
                Expanded(
                  child: Obx(
                      ()=>totalDonorController.isLoading.value?
                    Container(
                      width: screenWidth*0.04,
                      height: screenHeight*0.04,
                      alignment: Alignment.center,
                      child: const CircularProgressIndicator()):
                    Text(
                      overflow: TextOverflow.ellipsis,
                      'Receivers: ${totalDonorController.totalReceivers.value}',
                      style: const TextStyle(
                        color: Colors.black,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                  ),
                ),
                // Spacer(),
                // Image.asset(
                //   ImageStrings.fast_forward_double_right_arrows, // Replace with your arrow image path
                //   width: 20,
                //   height: 20,
                // ),
              ],
            ),
          ),
        ),
      ),
    ),
     SizedBox(width: screenWidth*0.03), // Add some spacing between the containers
    Expanded(
      child: GestureDetector(
        onTap: () {
          try {
            Get.to(const TotalBloodDonorsAndReceivers(showDonors: true,));
          } catch (e) {
            debugPrint('Error navigating to totalBloodDonors: $e');
          }
        },
```

```
            child: Container(
              height: screenHeight*0.08,
              decoration: const BoxDecoration(
                color: Color(0XFFF5F3F3),
                borderRadius: BorderRadius.all(Radius.circular(20)),
              ),
              child: Padding(
                padding: const EdgeInsets.all(8.0),
                child: Row(
                  children: [
                    Image.asset(
                      ImageStrings.donorPerson, // Replace with your profile image path
                      width: screenWidth*0.09,
                      height: screenHeight*0.045,
                    ),
                     SizedBox(width: screenWidth*0.04),
                    Expanded(
                      child: Obx(
                        ()=>totalDonorController.isLoading.value?
                            Container(
                                width: screenWidth*0.04,
                                height: screenHeight*0.04,
                                alignment: Alignment.center,
                                child: const CircularProgressIndicator()):
                           Text(
                            overflow: TextOverflow.ellipsis,
                            'Donors: ${totalDonorController.totalDonors.value}',
                            style: const TextStyle(
                              color: Colors.black,
                              fontWeight: FontWeight.bold,
                            ),
                          ),
                      ),
                    ),
                    // Spacer(),
                    // Image.asset(
                    //   ImageStrings.fast_forward_double_right_arrows, // Replace with your arrow image path
                    //   width: 20,
                    //   height: 20,
                    // ),
                  ],
                ),
              ),
            ),
          ),
        ),
```

```
        ],
      ),
    );
  }
```

TOTAL_BLOOD_DONOR_&_RECIVER

```
import 'package:blood_donation/utils/constants/image_strings.dart';
import 'package:flutter/material.dart';

import '../../controllers/blood_donors_and_receivers_controller.dart';
import '../../models/blood_donors_and_receivers_model.dart';

class TotalBloodDonorsAndReceivers extends StatefulWidget {
  final bool showDonors;
   const TotalBloodDonorsAndReceivers({super.key, required this.showDonors});

  @override
  _TotalBloodDonorsAndReceiversState createState() =>
_TotalBloodDonorsAndReceiversState();
}

class _TotalBloodDonorsAndReceiversState extends State<TotalBloodDonorsAndReceivers> {
  late bool _showDonors;
  final BloodController _controller = BloodController();
  late Future<List<BloodDonorsModel>> _bloodDonorsFuture;
  late Future<List<BloodDonorsModel>> _bloodReceiversFuture;

  @override
  void initState() {
    super.initState();
    _showDonors = widget.showDonors;
    _bloodDonorsFuture = _controller.fetchBloodDonors();
    _bloodReceiversFuture = _controller.fetchBloodReceivers();
  }

  @override
  Widget build(BuildContext context) {
    double screenWidth = MediaQuery.of(context).size.width;
    double screenHeight = MediaQuery.of(context).size.height;
    return Scaffold(
      appBar: AppBar(
        title: const Center(child: Text("Blood Group")),
        leading: IconButton(
          icon: const Icon(Icons.arrow_back),
          onPressed: () {
            Navigator.of(context).pop();
          },
```

```
    ),
  ),
  body: Column(
    children: [
      Expanded(
        child: FutureBuilder<List<BloodDonorsModel>>(
          future: _showDonors ? _bloodDonorsFuture : _bloodReceiversFuture,
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return const Center(child: CircularProgressIndicator());
            } else if (snapshot.hasError) {
              return Center(child: Text('Error: ${snapshot.error}'));
            } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
              return const Center(child: Text('No data available'));
            } else {
              final bloodGroups = snapshot.data!;

              return Padding(
                padding: const EdgeInsets.all(24.0),
                child: GridView.builder(
                  gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                    crossAxisCount: 2,
                    crossAxisSpacing: screenWidth*0.06,
                    mainAxisSpacing: screenHeight*0.03,
                    childAspectRatio: 3 / 2,
                  ),
                  itemCount: bloodGroups.length,
                  itemBuilder: (context, index) {
                    return Container(
                      padding: const EdgeInsets.all(8.0),
                      decoration: BoxDecoration(
                        color: const Color(0XFFF5F3F3),
                        borderRadius: BorderRadius.circular(15.0),
                        boxShadow: [
                          BoxShadow(
                            color: Colors.grey.withOpacity(0.5),
                            spreadRadius: 2,
                            blurRadius: 5,
                            offset: const Offset(0, 3),
                          ),
                        ],
                      ),
                      child: Column(
                        mainAxisAlignment: MainAxisAlignment.spaceBetween,
                        children: [
                          Padding(
```

```
                        padding:  EdgeInsets.only(left: screenWidth*0.025, right:
screenWidth*0.025),
                        child: Row(
                          mainAxisAlignment: MainAxisAlignment.spaceBetween,
                          children: [
                            Image.asset(
                              ImageStrings.blood_sample,
                              width: screenWidth*0.08,
                              height: screenHeight*0.04,
                            ),
                            Image.asset(
                              ImageStrings.person,
                              width: screenWidth*0.08,
                              height: screenHeight*0.04,
                            ),
                          ],
                        ),
                      ),
                      Padding(
                        padding:  EdgeInsets.only(left: screenWidth*0.025, right:
screenWidth*0.025),
                        child: Row(
                          mainAxisAlignment: MainAxisAlignment.spaceBetween,
                          children: [
                            Text(
                              bloodGroups[index].bloodGroup,
                              style: TextStyle(
                                fontWeight: FontWeight.w600,
                                fontSize: screenWidth*0.06,
                              ),
                            ),
                            Text(
                              '${bloodGroups[index].count}',
                              style: TextStyle(
                                fontWeight: FontWeight.w600,
                                fontSize: screenWidth*0.05,
                              ),
                            ),
                          ],
                        ),
                      ),
                    ],
                  ),
                );
              },
            ),
```

```
                );
              }
            },
          ),
        ),
      ],
    ),
  );
 }
}
```

CHAT_LIST:

```
import 'package:blood_donation/features/blood/profile/ChatList/views/widgets/chat_details.dart';
import 'package:pusher_channels_flutter/pusher_channels_flutter.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:get/get_core/src/get_main.dart';
import 'package:get_storage/get_storage.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

import '../../../../../api_constants.dart';

class ChatListPage extends StatefulWidget {
  @override
  _ChatListPageState createState() => _ChatListPageState();
}

class _ChatListPageState extends State<ChatListPage> {
  List<Map<String, dynamic>> chatData = [];
  bool isLoading = true;
  late PusherChannelsFlutter pusher;
  String? userId;
  String? friendId;
  String? lastMessage;
  String? lastMessageTime;
  @override
  void initState() {
    debugPrint("Call initstate chatListPage");
    super.initState();
    _initializePusher();
    fetchChatData();
  }

  Future<void> _initializePusher() async {
```

```
  try {
    pusher = PusherChannelsFlutter.getInstance();

    await pusher.init(
      apiKey: "7807e17ee3151d92883b", // Replace with your Pusher API key
      cluster: "ap2", // Replace with your Pusher cluster
      onConnectionStateChange: (previousState, currentState) {
        debugPrint("Pusher connection state: Changed from $previousState to $currentState");
      },
      onError: (String code, int? statusCode, dynamic data) {
        debugPrint("Pusher error: Code=$code, Status Code=$statusCode, Data=$data");
      },
    );

    await pusher.subscribe(
      channelName: 'message',
      onEvent: (dynamic event) { // Leave event as dynamic
        _handleIncomingMessage(event.data);
        debugPrint("Raw event data: ${event.data}");
      },
    );
    await pusher.connect();
  } catch (e) {
    debugPrint('Pusher initialization failed: $e');
  }
}

void _handleIncomingMessage(String eventData) {
  debugPrint("_handleIncomingMessage call ChatListPage");

  // Parse incoming message data
  final data = json.decode(eventData);
  debugPrint("FromId ${data['from_id']}  toId ${data['to_id']}");

  // Check if the message is for the current user
  if (data['to_id'].toString() == userId) {
    debugPrint("InsideIf");

    String friendId = data['from_id'].toString();
    String lastMessage = data['message'].toString();
    String lastMessageTime = data['last_message_time'].toString();

    debugPrint("FriendId $friendId");
    debugPrint("lastMessage $lastMessage");
    debugPrint("lastMessageTime $lastMessageTime");
```

```
    // Find the specific chat in chatData by matching friendId
    setState(() {
      int chatIndex = chatData.indexWhere((chat) => chat['friendId'].toString() == friendId);

      if (chatIndex != -1) {
        // Update the specific chat's last message and last message time
        chatData[chatIndex]['lastMessage'] = lastMessage;
        chatData[chatIndex]['lastMessageTime'] = lastMessageTime;

        // Move the updated chat to the top of the list (optional: sort the list)
        sortingChatList();
      }
    });
  }
}

Future<void> fetchChatData() async {
  final GetStorage storage = GetStorage();
  final token = storage.read('token');

  // Check if the token is available
  if (token == null) {
    debugPrint('No token found. Please login again.');
    setState(() {
      isLoading = false; // Set loading to false if no token
    });
    return;
  }

  try {
    final response = await http.get(
      Uri.parse('${ApiConstants.baseUrl}/api/chats'),
      headers: {
        'Authorization': 'Bearer $token',
        'Content-Type': 'application/json',
      },
    );

    // Print the response status code for debugging
    debugPrint('Response status: ${response.statusCode}');
    debugPrint('Response body: ${response.body}');

    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      if (data['status']) {
        userId = data['data']['user_id'].toString();
```

```
        debugPrint("UserId on chatList page $userId");
        setState(() {
          // Parse the chat data from the response
          chatData = (data['data']['chats'] as List).map((chat) {
            return {
              'id': chat['id'],
              'name': chat['friend']['name'],
              'friendId': chat['friend']['id'],
              'lastMessage': chat['last_message'],
              'lastMessageTime': chat['last_message_time'],
              'profileImage': chat['friend']['image'] ?? 'assets/icons/profile/img.png', // Fallback
image
            };
          }).toList();

       // Sort by lastMessageTime in descending order
          sortingChatList();
          isLoading = false; // Set loading to false after data is fetched
        });
       } else {
        // Handle the case where the status is false
        debugPrint('Error: ${data['message']}');
        setState(() {
          isLoading = false;
        });
       }
     } else {
       // Handle the error for non-200 response status
       debugPrint('Error fetching chat data: ${response.statusCode}');
       setState(() {
         isLoading = false;
       });
     }
   } catch (e) {
     debugPrint('Error: $e');
     setState(() {
       isLoading = false;
     });
   }
  }

  @override
  Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: const Text('Chats'),
```

```
    ),
    body: isLoading
        ? const Center(child: CircularProgressIndicator()) // Show loading indicator while fetching
data
        : ListView.builder(
      itemCount: chatData.length > 6 ? 6 : chatData.length, // Limit to 6 chats
      shrinkWrap: true,
      itemBuilder: (context, index) {
        return Column(
          children: [
            Container(
              height: 80,
              padding: const EdgeInsets.symmetric(vertical: 10.0),
              child: ListTile(
                leading: CircleAvatar(
                  radius: 30,
                  backgroundImage: NetworkImage(chatData[index]['profileImage']),
                ),
                title: Text(
                  chatData[index]['name'] ?? '',
                  style: const TextStyle(
                    fontSize: 18,
                    fontWeight: FontWeight.bold,
                  ),
                ),
                subtitle: Text(
                  (chatData[index]['lastMessage'] ?? '')
                      .substring(0, chatData[index]['lastMessage'].length > 20 ? 20 :
chatData[index]['lastMessage'].length) +
                      (chatData[index]['lastMessage'].length > 20 ? '...' : ''),
                  style: const TextStyle(fontSize: 16),
                ),
                trailing: Text(
                  formatTime(chatData[index]['lastMessageTime']),
                  style: const TextStyle(fontSize: 14),
                ),
                onTap: () {
                  // Extract relevant data from the selected chat
                  String profileName = chatData[index]['name'];
                  String profileImage = chatData[index]['profileImage'];
                  int chatId = chatData[index]['id']; // Ensure this is an int

                  // Navigate to chat detail page when tapped
                  Get.to(ChatDetailPage(
                    chatId: chatId, // Pass as int
                  ));
```

```
                },
              ),
            ),
            Divider(
              thickness: 1,
              color: Colors.grey[300],
            ),
          ],
        );
      },
    ),
  );
 }

 String formatTime(String time) {
  // Convert the last message time to a more user-friendly format
  DateTime dateTime = DateTime.parse(time);
  // Use a simple formatter to get the desired format
  return '${dateTime.hour}:${dateTime.minute < 10 ? '0' : ''}${dateTime.minute}
${dateTime.hour >= 12 ? 'PM' : 'AM'}';
 }
 void sortingChatList(){
   chatData.sort((a, b) {
    // Parse the lastMessageTime strings to DateTime
    DateTime timeA = DateTime.parse(a['lastMessageTime']);
    DateTime timeB = DateTime.parse(b['lastMessageTime']);

    // Compare timeB and timeA for descending order
    return timeB.compareTo(timeA);
  });
 }
}
```

AVAILABLE_DONOR_MODEL

```
class Donor {
 final String id;
 final String name;
 final String email;
 final String address;
 final String bloodGroup;
 final String phoneNumber;
 final String city;
 final String height;
 final String weight;
 final String age;
```

```
  Donor({
    required this.id,
    required this.name,
    required this.address,
    required this.bloodGroup,
    required this.email,
    required this.phoneNumber,
    required this.city,
    required this.height,
    required this.weight,
    required this.age,

  });

  factory Donor.fromJson(Map<String, dynamic> json) {
    return Donor(
      id: json['id']?.toString() ?? 'Unknown',
      name: json['name'] ?? 'Unknown',
      address: json['address'] ?? 'Unknown',
      bloodGroup: json['blood_group'] ?? 'Unknown',
      email: json['email'] ?? 'Unknown',
      phoneNumber: json['phone'] ?? 'Unknown',
      city: json['city'] ?? 'Unknown',
      height: json['height'] ?? 'Unknown',
      weight: json['weight'] ?? 'Unknown',
      age: json['age'] ?? 'Unknown',
    );
  }
}
```

Chapter 7

Summary, Conclusion and Future Enhancements

Chapter 7: Summary, Conclusion & Future Enhancements

7.1. Project Summary

- The purpose of the project, such as developing a blood donation app to connect donors with recipients.
- Key features or functionalities of the app (e.g., user login, eligibility checks, donor matching).
- Technologies used (such as specific frameworks or database).
- A high-level overview of the testing method used to ensure functionality and reliability.
- The Blood donation application aims to simplify and streamline the process of connecting blood donors with those in need.
- Build using Flutter and Laravel, the app offers features like donor login, verification, and real-time matching.

7.2. Achievements and Improvements

- Significant milestones achieved (e.g., implementing donor matching)
- Challenges overcome, such as handling database connectivity issues or optimizing the app's performance.
- Any new or improved features based on user interface and API Google maps addition.

7.3. Critical Review

- Strengths, such as the app's usability, robustness, or accuracy of matching algorithms.
- Weaknesses or limitations, like limited scalability or dependency on specific technologies.
- Any identified areas for improvement, such as the need for additional features or better error handling.

7.4. Lessons Learnt

- Technical skills acquired (e.g., learning a new framework or optimizing code).
- Problem-solving approaches, like debugging complex issues or refining user flows.
- Teamwork or project management lessons, especially if the project required collaboration.

7.5. Future Enhancements/Recommendations

- Adding offline support to allow users to access essential features without internet.
- Expanding features like SMS notifications, or multi-language support.
- Enhancing security with multi-factor authentication or encrypted data storage.
- Improving scalability for larger user bases or integrating machine learning to better predict donor-recipient matches.

Reference and Bibliography

Reference and Bibliography

[1] https://pub.dev/packages/pusher_channels_flutter

[2] https://your-server.com/pusher/auth

[3] https://www.behance.net/gallery/129489483/Blood-Donation-App-Case-Study?tracking_source=search_projects|blood+donation+app&l=0