

A Project

“Blood Donation System – RedAid”



## **CSE 100: Software Development Project I**

### **S U B M I T T E D   B Y**

Name	ID	Intake
Nowshin Ashrafi	20245103166	53
Mashfi Rejoan Saikat	20245103183	53
Md. Akif Uz Zaman	20245103190	53

### **S U P E R V I S E D   B Y :**

Ms. Iffat Tamanna  
Assistant Professor  
Department of Computer Science and Engineering

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY**  
**(BUBT)**

*1<sup>st</sup> January 2025*

## Abstract

The idea for this project was inspired by the challenges often encountered in blood donation during emergencies. Frequently, individuals rely on messaging groups to seek blood donors, a process that is inefficient and prone to delays, especially in critical situations. To address this issue, we developed **RedAid**, a blood donation system designed to streamline the search for donors and facilitate efficient communication between donors and recipients.

In its current form, RedAid is implemented in raw C++ with a file-based storage system. The system includes a signup/login dashboard, enabling users to register, search for blood donors based on blood group, and join as donors themselves. Donors can also update their donation history. User data is stored in a CSV-like format, and retrieval is managed through structured file parsing. While functional, the current implementation serves as a prototype and is not deployable in its current state.

Looking ahead, our goal is to transform RedAid into a comprehensive digital health ecosystem. Future plans include integrating location-based blood search, ambulance services, emergency response systems, doctor consultations, vaccination notifications, elderly care features, and more. By leveraging scalable technologies and innovative solutions, RedAid aims to centralize health services, reduce response times, and improve accessibility, ultimately creating a unified platform for healthcare needs.

# Table of Contents

<b><u>Abstract</u></b> .....	i
<b><u>List of Figures</u></b> .....	ii
1. Chapter: 1 <b>Introduction</b> .....	1-2
<hr/>	
o 1.1 Problem Specification/Statement	
o 1.2 Objectives	
o 1.3 Flow of the Project	
o 1.4 Organization of Project Report	
2. Chapter: 2 <b>Background</b> .....	3-4
<hr/>	
o 2.1 Existing System Analysis	
o 2.2 Supporting Literatures	
3. Chapter: 3 <b>System Analysis &amp; Design</b> .....	5-8
<hr/>	
o 3.1 Technology & Tools	
o 3.2 Model & Diagram	
4. Chapter: 4 <b>Implementation</b> .....	9-29
<hr/>	
o 4.1 Interface Design/Front-End	
o 4.2 Back-End	
o 4.3 Modules	
5. Chapter: 5 <b>User Manual</b> .....	30
<hr/>	
o 5.1 System Requirements	
o 5.2 User Interfaces	
6. Chapter: 6 <b>Conclusion</b> .....	31-32
<hr/>	
o 6.1 Findings	
o 6.2 Limitations	
o 6.3 Future Works	
7. <b>References</b> .....	33

# Chapter 1

## INTRODUCTION

---

### 1.1 Problem Specification

In critical medical situations, finding compatible blood donors quickly and efficiently can mean the difference between life and death. However, the current reliance on informal networks such as social media groups or word of mouth is highly inefficient, leading to delays and potential inaccuracies in information. This manual and unstructured approach to blood donation management is unsuitable for emergency situations.

This project aims to address these issues by developing an automated blood donation system, initially built with raw C++ and file handling techniques, which centralizes and streamlines donor management. The system categorizes donors by blood groups and maintains a searchable database for instant retrieval of donor information. By leveraging this solution, blood donation processes can become significantly more organized and responsive, enabling quicker access to lifesaving resources.

### 1.2 Objectives

The objectives of this project include:

- **Streamlining donor registration and activation** to ensure an organized onboarding process.
- **Maintaining a structured database** of available donors categorized by blood group.
- **Facilitating efficient donor searches** based on blood group, availability, and last donation date.
- **Simplifying user account management** with an easy-to-navigate interface for login, registration, and updates.
- **Ensuring data security** through modular design and robust file handling techniques.
- **Enhancing reliability** by storing donor details in separate files categorized by blood groups, ensuring data consistency and integrity.

### 1.3 Scope

This system provides:

- **User-friendly interfaces** to manage donor information efficiently.
- **File-based storage and retrieval mechanisms**, ensuring donor availability is maintained by blood group.
- **Search functionalities** to quickly identify donors based on specific criteria such as blood group and donation history.

- **Data modularity and consistency** through the use of CSV-formatted files for data storage and retrieval.
- **Future expansion capabilities** to include features like doctor consultations, ambulance services, elderly care, vaccination campaign notifications, and emergency management.

## 1.4 Organization of Project Report

The report is organized as follows:

- **Chapter 2: Existing Systems and Theoretical Background**  
Discusses the limitations of current manual systems and provides an overview of relevant theoretical concepts.
- **Chapter 3: Technologies and System Design**  
Details the technological stack used, system architecture, and design diagrams.
- **Chapter 4: Implementation**  
Explains the development process, focusing on both front-end and back-end modules.
- **Chapter 5: User Manual**  
Includes system requirements, installation steps, and guidance for using the system interface.
- **Chapter 6: Conclusion and Future Work**  
Summarizes the findings, highlights system limitations, and proposes future enhancements to broaden the scope of the project.

# Chapter 2

## INTRODUCTION

---

### 2.1 Existing System Analysis

In the current landscape, blood donation management is often handled through manual or informal methods. Social media groups, such as those on Facebook or WhatsApp, are commonly used for blood requests. These groups rely on individuals posting requests with details like blood type, location, and urgency, hoping to find a willing donor within the group. While these platforms provide wide reach, they come with several limitations:

#### Pros:

- Wide accessibility and immediate broadcasting of requests.
- Minimal technical infrastructure required.

#### Cons:

- Lack of structured and verified data on donors.
- Delayed responses due to the need for manual coordination.
- High dependency on active group members at a specific time.
- Difficulty in locating donors by specific criteria such as availability or recent donation history.

Some digital systems exist in the form of mobile apps or websites offering blood donation services, but many of these systems are region-specific and lack comprehensive features such as real-time donor availability tracking or integration with emergency services.

The proposed system, RedAid, seeks to address these shortcomings by introducing an automated and centralized solution for blood donor management. It ensures that data is well-organized, easily searchable, and secure, significantly improving response times in emergencies.

### 2.2 Supporting Literatures

The development of this project leverages both theoretical and technological knowledge. Below is an overview of the applied methodologies, tools, and their significance:

#### Theoretical Foundations

##### 1. Data Structures:

- Use of structured data storage and retrieval (e.g., CSV files) to ensure efficient management of donor information categorized by blood group.

- File handling techniques ensure data consistency and modularity.
- 2. **Algorithms:**
  - Implementation of search algorithms for quick retrieval of donor details based on blood group and availability criteria.
  - Sorting algorithms for organizing donor records to enhance readability and usability.

## **Mathematical Concepts**

- Use of date calculations to determine donor eligibility based on the last donation date.
- Validation techniques for ensuring data integrity, such as verifying phone numbers and email addresses using regex.

## **Tools and Technologies**

1. **Programming Language:**
  - C++ was chosen due to its robustness, efficiency, and suitability for handling file-based operations. Its modular design capabilities support the secure and scalable development of the project.
2. **Development Tools:**
  - **Integrated Development Environment (IDE):** Visual Studio Code for code writing and debugging.
  - **File Management:** CSV files are used as a simple yet effective database for storing and retrieving donor information.
3. **Validation Techniques:**
  - Regular expressions (regex) for verifying user inputs such as email addresses and phone numbers.

## **Why These Tools and Techniques?**

- The modular design enabled by C++ and file handling techniques aligns with the project's initial focus on simplicity and efficiency.
- The use of CSV files provides a lightweight, easily manageable database solution suitable for prototyping and early-stage development.
- Validation through regex ensures data quality, reducing errors and enhancing reliability.

By combining these theoretical and technological approaches, the RedAid system establishes a foundation for a reliable and scalable blood donation management solution, ready for future expansion into broader healthcare services.

# Chapter 3

## SYSTEM ANALYSIS & DESIGN

---

### 3.1 Technology & Tools

#### Technologies Used:

1. **Programming Language:**
  - **C++:** The project is implemented using C++ due to its efficiency and ability to manage file-based data effectively. It supports modular programming, ensuring maintainability and scalability.
2. **File Management:**
  - CSV files are used to store donor information, categorized by blood groups. This choice allows for lightweight, structured data storage without requiring a full-fledged database.

#### Software Requirements:

- **Operating System:** Windows 10 or later.
- **IDE:** Visual Studio Code with extensions for C++ development.
- **Version Control:** Git for tracking changes and collaborating on code.

### 3.2 Model & Diagram

#### 3.2.1 Model (SDLC Model)

The **Waterfall Model** was chosen for this project due to its structured and sequential approach, which aligns with the project's predefined requirements and objectives. The following phases were followed:

- **Requirement Analysis:** Gathering details about donor management needs.
- **Design:** Planning the system architecture, data storage, and user interface.
- **Implementation:** Writing modular C++ code and integrating file handling for data management.
- **Testing:** Ensuring the system functions as expected with both valid and invalid inputs.
- **Deployment:** Setting up the system for use.
- **Maintenance:** Adding future features like doctor consultations or ambulance tracking.

#### Benefits:

- Ensures each phase is completed before the next begins.
- Provides clear documentation and progress tracking.



### 3.2.2 System Architecture

The system architecture outlines the components and their interactions:

#### 1. User Interface (UI):

- Provides donor registration, login, donor search, register as donor, update donor information functionality.

#### 2. File Management System:

- Handles data storage, retrieval, and updates in CSV files categorized by blood group.

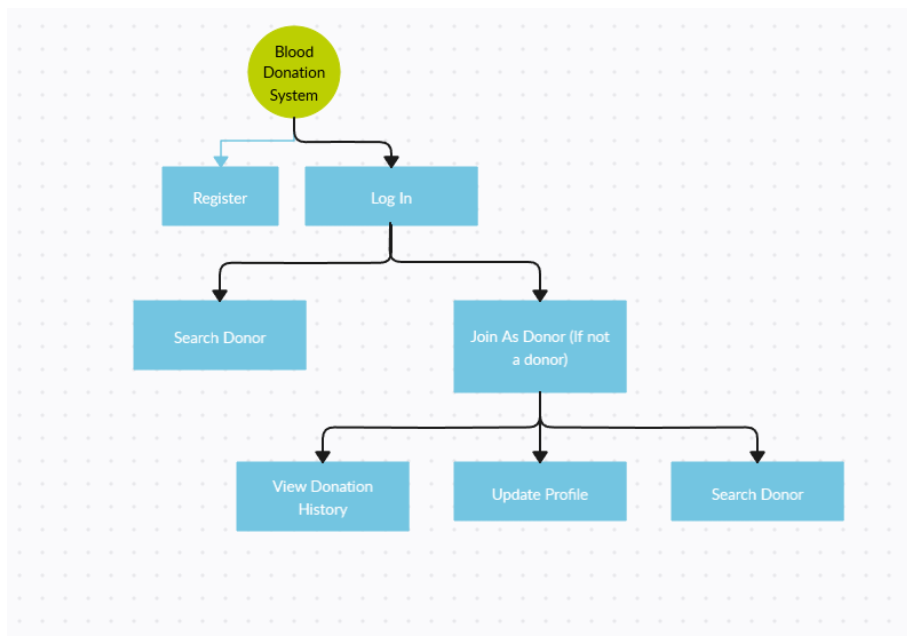
#### 3. Search and Validation Module:

- Implements search algorithms and validates user inputs.

#### 4. Core Logic:

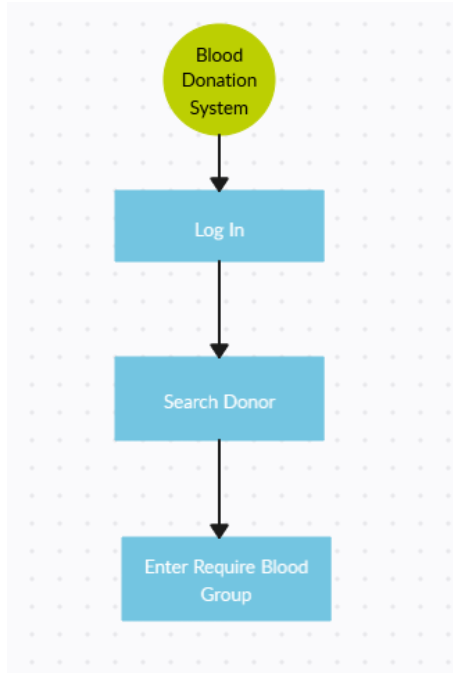
- Manages donor registration, availability checks, and eligibility criteria.

**Diagram:**



### 3.2.3 Use Case Diagram

A use case diagram highlights the interactions between the system and its users. Key actors:



### 3.2.4 Database Schema

Since the system uses CSV files instead of a database, the schema outlines the file structure:

Field Name	Data Type	Description
Name	String	Full name of the donor
Phone Number	String	Unique identifier
Blood Group	String	Blood group (e.g., A+, O-)
Last Donation	Date	Last donation date
Availability	Boolean	True if eligible to donate

### 3.2.5 Algorithms/Flowchart

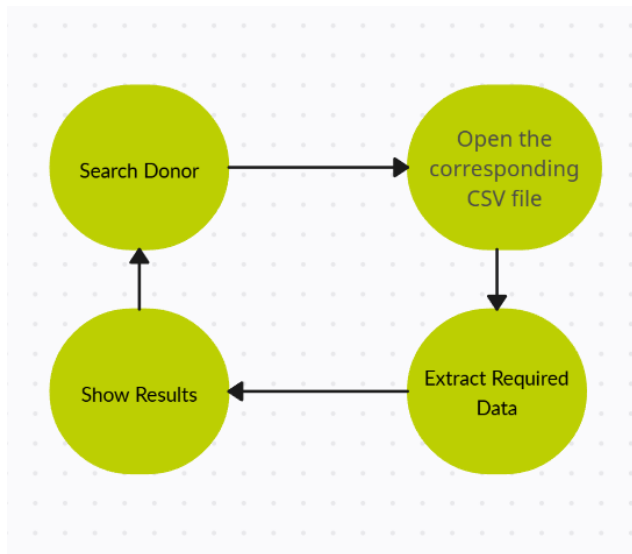
#### Algorithm: Donor Search

1. Input: Blood group.
2. Open the corresponding CSV file.
3. For each record:

Check if the donor is eligible (availability == true).

4. Display matching donors.

#### Flowchart:



This chapter provides a comprehensive overview of the technologies, tools, and design methodologies applied in the project. Let me know if you would like additional diagrams or modifications!

## Chapter 4

### IMPLEMENTATION

---

#### 4.1 Interface Design/Front-End

The **Interface Design/Front-End** of the **RedAid Blood Management System** is simple and text-based, catering to the project's goal of accessibility and functionality without reliance on advanced graphical interfaces. The interface was developed using **C++ console I/O operations**, ensuring it is lightweight, straightforward, and suitable for beginners.

#### VISUAL REPRESENTATION

##### INTRO PAGE:



## THE DASHBOARD PROVIDES INTUITIVE OPTIONS:

```
=====
|           Dashboard           |
=====

-----
|   [1] Register   |
-----

-----
|   [2] Log In    |
-----

-----
|   [3] Exit      |
-----

Enter your choice: |
```

## UNDER REGISTRATION:

```
D:\piu\RedAid-main\RedAid-1 x + v

*           Registration           *
-----

*   Select your blood group   *
-----
* No | Blood Group          *
-----
* 1 | A+                    *
* 2 | A-                    *
* 3 | B+                    *
* 4 | B-                    *
* 5 | AB+                   *
* 6 | AB-                   *
* 7 | O+                    *
* 8 | O-                    *
-----

Enter the number corresponding to your blood group: 1
Enter your name: Nowshin Ashrafi
Enter your phone (01xxxxxxxx): 01811111111
Enter password: *****
Re-enter password to confirm: *****1
```

#### AFTER SUCCESSFUL REGISTRATION:

```
*****
*      Registration Successful      *
*****

          *****
          *      Your Details      *
          *****
* Name           :      Nowshin Ashrafi      *
*                                                         *
* Phone          :      01833333333          *
*                                                         *
* Blood Group    :      A+                  *
*                                                         *
*****
```

#### UNDER THE EXIT:

```
=====
THANK YOU FOR USING BLOOD DONATION SYSTEM
=====

-----
|                                     |
|   Your contribution helps save lives!   |
|                                     |
| * One donation can save up to 3 lives  |
| * Every 2 seconds someone needs blood |
| * Be a hero - Donate blood regularly  |
|                                     |
|   Emergency Blood Service Contacts:   |
|   Contact: 0189-7911901               |
|   Email: rejoansaikat01@gmail.com     |
|                                     |
|-----|
|                                     |
|   Developed by: Mashfi Rejoan Saikat   |
|   Developed by: Md. Akif Uz Zaman     |
|   Developed by: Nowshin Ashrafi       |
|   Version: 1.0.0                      |
|-----|

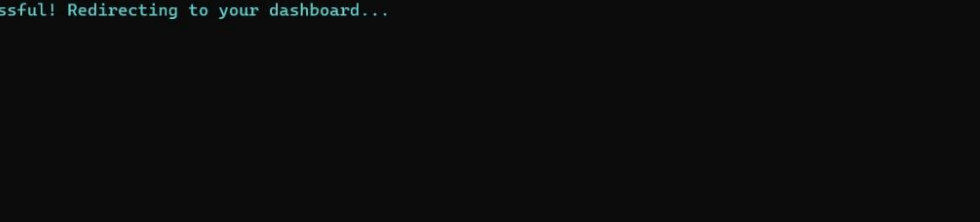
Press any key to exit...
Process returned 0 (0x0)   execution time : 5.192 s
Press any key to continue.
```

## LOG IN DASHBOARD

```
D:\piu\RedAid-main\RedAid-1 X + v
```

```
*****  
*                               Log In                               *  
*****  
  
*****  
*   Select your blood group   *  
* No | Blood Group           *  
* 1 | A+                     *  
* 2 | A-                     *  
* 3 | B+                     *  
* 4 | B-                     *  
* 5 | AB+                    *  
* 6 | AB-                    *  
* 7 | O+                     *  
* 8 | O-                     *  
*****  
Enter the number corresponding to your blood group: 1  
Enter your phone (user ID): 01873619760  
Enter your password: *****1|
```

**SUCCESSFUL LOGIN:**



The screenshot shows a terminal window with a dark background. The title bar at the top indicates the file path 'D:\piu\RedAid-main\RedAid-1' and includes standard window controls (minimize, maximize, close). The terminal output displays the message 'Login successful! Redirecting to your dashboard...' in a light blue/cyan monospaced font. A cursor is visible on the line following the message.

```
D:\piu\RedAid-main\RedAid-1 x + v
Login successful! Redirecting to your dashboard...
|
```

## MAIN DASHBORD:

```
D:\piu\RedAid-main\RedAid-1 x + -
=====
|| Dashboard ||
=====

| [1] View Donation |
| History           |
|-----|
| [2] Update Profile |
|-----|
| [3] Search Blood   |
|-----|
| [4] Logout         |
|-----|

Enter your choice: 1|
```

## UNDER DONATION HISTORY:

```
=====
DONATION HISTORY
=====

-----
Personal Information
-----
Name       : Nowshin Ashrafi
Blood Group : nowshinashrafi@gmail.com
Age        : 22
Weight     : 55 kg
-----

-----
Donation Information
-----
First Time Donor : No
Last Donation    : 2024-03-02
Next Eligible Date: 2024-06-02
-----

-----
Health Status
-----
Medical Conditions: Eligible
-----

Press any key to return to dashboard...|
```



## UNDER UPDATE PROFILE:

```
-----
*           Current Information           *
-----

Name: Nowshin Ashrafi
Phone: 01873619760
Blood Group: nowshinashrafi@gmail.com

What would you like to update?
1. Email
2. Age
3. Weight
4. Last Donation Date
5. Back to Dashboard
Enter your choice: 5
Enter new eligibility status (Yes/No): YES|
```

## UNDER SEARCH DONER:

```
-----
*           Search Blood Donors           *
-----

-----
*   Select the Blood Group   *
-----
*  No  |   Blood Group   *
-----
*  1   |   A+           *
*  2   |   A-           *
*  3   |   B+           *
*  4   |   B-           *
*  5   |   AB+          *
*  6   |   AB-          *
*  7   |   O+           *
*  8   |   O-           *
-----

Enter the number corresponding to the blood group: 1|
```

## RESULT OF SEARCH DONER:

```
Name      : 01318377455
Phone     : MD.Mashiur Rhaman Dipto
Email     : rahmandipto149@gmail.com
Status    : Eligible
-----
Donor Details
Name      : 01876519855
Phone     : Feroza Akter Eti
Email     : ferozaaktereti@gmail.com
Status    : Eligible
-----
Donor Details
Name      : 01612781670
Phone     : MD.MUNABBER HUSSAIN TURZA
Email     : Turza
Status    : Eligible
-----
Donor Details
Name      : 01518935692
Phone     : Shahedul Islam Redwan
Email     : Shahedulislam1007@gmail.com
Status    : Eligible
-----
Donor Details
Name      : 01812622330
Phone     : Monika rani siddha
Email     : Monika rani siddha
Status    : Eligible

Press any key to return to dashboard...P|
```

## UNDER LOGOUT:

```
D:\piu\RedAid-main\RedAid-1 x + v
Logging out... Please wait.
```

## 4.2 Back-End

The back-end of the Blood Management System is designed to efficiently manage donor data by utilizing text files. The data for each donor is stored in plain text files, categorized by blood groups. This system ensures that the data is easy to retrieve and manage while maintaining simplicity and clarity in its design.

# Welcome!!!!

The back-end includes a visually engaging welcome page displayed to users when the system is launched. This feature leverages Windows API functions to add color and delays, making the interaction more dynamic and appealing. Below is the function definition for the welcome page:

[illegible]

This function displays a formatted welcome message with the following elements:

1. **Styled Welcome Banner:** The banner introduces the platform as "RedAid: A Centralized Blood Solution Platform."
2. **Tagline:** The tagline, "Bound by Blood, Driven by Compassion," reflects the system's mission.
3. **Authors List:** Displays the names of the developers, personalizing the system for its users.
4. **Dynamic Effects:** The use of `HANDLE`, `SetConsoleTextAttribute`, and `Sleep` functions adds color and pauses to the output for a polished, professional experience.
5. **Transition:** Calls `refresh_screen()` and `entry_dashboard()` to guide users into the system.

## Screen Refresh Function

```
void refresh_screen()
{
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}
```

To ensure cross-platform compatibility, the back-end includes a `refresh_screen` function for clearing the console screen. This function checks the operating system and uses the appropriate command to clear the screen:

This function is called in the welcome page and other parts of the application where refreshing the console is necessary, maintaining a clean and user-friendly interface regardless of the user's platform

## Entry Dashboard

The **Entry Dashboard** is implemented as a console-based menu system. It displays navigation options in a visually structured format and allows users to proceed based on their input. Below is the function definition for the dashboard:

[illegible]

**1. Dynamic Interaction:**

- The dashboard captures the user's input and invokes the corresponding functions:
  - `register_user()` for new user registration.
  - `login_user()` for user authentication.
  - `exit_message()` for a graceful exit from the program.

**2. Error Handling:**

- The system ensures valid input by prompting the user again if they enter an invalid option.

**3. Platform Independence:**

- The dashboard refreshes the screen after every input using the `refresh_screen()` function, ensuring compatibility across Windows and Unix-based systems.

**4. Centralized Navigation:**

- The dashboard acts as the central hub for user interaction, linking to the registration and login functionalities seamlessly.

The **login\_user** function enables users to log in based on their blood group and unique credentials. Below is the function definition:

```

string stored_phone = line.substr(0, pos1);

size_t pos2 = line.find(" ", pos1 + 1);
string stored_password = line.substr(pos1 + 1, pos2 - pos1 - 1);

size_t pos3 = line.find(" ", pos2 + 1);
string stored_name = line.substr(pos2 + 1, pos3 - pos2 - 1);

if (stored_phone == phone && stored_password == password)
{
    refresh_screen();
    cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t|                               Welcome                               |\n";
    cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\tWelcome back, " << stored_name << "\n\n";
    user_found = true;
    break;
}
}

file.close();

if (user_found)
{
    refresh_screen();
    cout << "Login successfull Redirecting to your dashboard...\n";
    Sleep(2000);
    refresh_screen();
    login_dashboard(blood_group, phone);
}
else
{
    cout << "Invalid phone number or password. Please try again.\n";
    Sleep(2000);
    goto label3; // Redirect back to the login
}
}

```

- 19

## 5. Platform Independence:

- The `refresh_screen()` function ensures that the interface is cleared and refreshed for both Windows and Unix-based systems.

## 6. Post-Login Navigation:

- Upon successful authentication, the system redirects the user to their personalized dashboard (login\_dashboard).

## Registration Function

The **register\_user** function allows new users to register in the system, associating their details with their blood group. Below is the function definition:

```

void register_user()
{
    cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t *           Registration           *\n";
    cout << "\t\t\t\t\t-----\n\n";

    string name, phone, blood_group, password, confirm_password;
    bool isValid = false;

    cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t* Select your blood group *\n";
    cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t* No | Blood Group *\n";
    cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t* 1 | * << setw(5) << "Aa" << " *\n";
    //cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t* 2 | * << setw(5) << "Ab" << " *\n";
    //cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t* 3 | * << setw(5) << "Ba" << " *\n";
    //cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t* 4 | * << setw(5) << "Bb" << " *\n";
    //cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t* 5 | * << setw(5) << "ABa" << " *\n";
    //cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t* 6 | * << setw(5) << "ABb" << " *\n";
    //cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t* 7 | * << setw(5) << "Oa" << " *\n";
    //cout << "\t\t\t\t\t-----\n";
    cout << "\t\t\t\t\t* 8 | * << setw(5) << "Ob" << " *\n";
    cout << "\t\t\t\t\t-----\n";

    int blood_group_choice;
    do
    {
        cout << "Enter the number corresponding to your blood group: ";
        cin >> blood_group_choice;
        switch (blood_group_choice)
        {
            case 1:
                blood_group = "Aa";
                break;
            case 2:
                blood_group = "Ab";
                break;
            case 3:
                blood_group = "Ba";
                break;
            case 4:
                blood_group = "Bb";
                break;
            case 5:
                blood_group = "ABa";
                break;
            case 6:
                blood_group = "ABb";
                break;
            case 7:
                blood_group = "Oa";
                break;
            case 8:
                blood_group = "Ob";
                break;
            default:
                cout << "Invalid blood_group choice. Please select a valid blood_group.\n";
        }
    } while (blood_group.empty());

    // Get name
    cout << "Enter your name: ";
    cin.ignore();
    getline(cin, name);

    // Validate phone number
    // Get phone number and check for duplicates
    do {
        cout << "Enter your phone (0xxxxxxx): ";
        cin >> phone;

        if (phone.length() != 11 || !regex_match(phone, regex("[0-9]*"))) {
            cout << "Invalid phone number. Please enter exactly 11 digits.\n";
        }
        else if (phone.substr(0, 3) != "01") {
            cout << "Phone number must start with '01'.\n";
        }
        else if (isPhoneNumberRegistered(phone, blood_group)) {
            cout << "This phone number is already registered. Please use a different number.\n";
        }
        else {
            break; // Valid and not registered
        }
    } while (true);

    // Validate password
    do {
        case 4:
            blood_group = "Aa";
            break;
        case 5:
            blood_group = "Ab";
            break;
        case 6:
            blood_group = "Ba";
            break;
        case 7:
            blood_group = "Bb";
            break;
        case 8:
            blood_group = "Oa";
            break;
        default:
            cout << "Invalid blood_group choice. Please select a valid blood_group.\n";
        }
    } while (blood_group.empty());

    // Step 1: Enter password
    password = getPassword("Enter password: ");
    // Step 2: Validate password
    if (isValidPassword(password)) {
        cout << "Invalid password! Ensure it has at least 8 characters, one uppercase, one lowercase, and one digit.\n";
        continue; // Restart the loop
    }
    // Step 3: Re-enter password
    confirm_password = getPassword("Re-enter password to confirm: ");
    // Step 4: Check if passwords match
    if (password != confirm_password) {
        cout << "Passwords do not match. Please try again." << endl;
    }
    else {
        isValid = true;
        cout << "Password confirmed and valid." << endl;
    }
}

} while (!isValid);

// Save to file
ofstream file(blood_group + ".txt", ios::app);
file << phone << ", " << password << ", " << name << ", " << blood_group << "\n";
file.close();

refresh_screen();

// Display dashboard header
cout << "\t\t\t\t\t*****\n";
cout << "\t\t\t\t\t* Your Details *\n";
cout << "\t\t\t\t\t*****\n";
cout << "\t\t\t\t\tName : " << setw(20) << left << name << " *\n";
cout << "\t\t\t\t\t*\n";
//cout << "\t\t\t\t\t*****\n";
cout << "\t\t\t\t\tPhone : " << setw(20) << left << phone << " *\n";
cout << "\t\t\t\t\t*\n";
//cout << "\t\t\t\t\t*****\n";
cout << "\t\t\t\t\tBlood Group : " << setw(20) << left << blood_group << " *\n";
cout << "\t\t\t\t\t*\n";
cout << "\t\t\t\t\t*****\n";

Sleep(4000);
refresh_screen();
entry_dashboard();
}
}

```

## Password Hiding

To improve user privacy, the **getHiddenPassword** function captures the password securely, hiding the input as asterisks (\*) while allowing users to delete characters.

```

string getHiddenPassword(const string& prompt) {
    string password;
    char ch;
    cout << prompt;

    while ((ch = _getch()) != '\n') {
        if (ch == '\b' && !password.empty()) { // Handle backspace
            password.pop_back();
            cout << "\b \b";
        } else if (ch != '\b') {
            if (!password.empty()) {
                cout << "\b*";
            }
            password += ch;
            cout << ch;
        }
    }

    if (!password.empty()) {
        cout << "\b*";
    }
    cout << endl;
    return password;
}

```

## Unique ID Validation

To ensure ID uniqueness, the **isPhoneNumberRegistered** function checks if a phone number (used as a unique ID) already exists in the blood group file.

```

bool isPhoneNumberRegistered(const string& phone, const string& blood_group) {
    ifstream file(blood_group + ".txt");
    if (!file.is_open()) {
        return false; // If the file doesn't exist, the phone number can't be r
    }

    string line;
    while (getline(file, line)) {
        size_t pos = line.find(',');
        if (pos != string::npos) {
            string existingPhone = line.substr(0, pos);
            if (existingPhone == phone) {
                return true; // Phone number found, already registered
            }
        }
    }
    return false; // Phone number not found, can register
}

```

## Password Validation

The **isPasswordValid** function ensures the password meets the following criteria:

```

bool isPasswordValid(const string& password) {
    if (password.length() < 8) return false;

    regex hasUppercase("[A-Z]");
    regex hasLowercase("[a-z]");
    regex hasSpecialChar("[^A-Za-z0-9]");

    return regex_search(password, hasUppercase) &&
        regex_search(password, hasLowercase) &&
        regex_search(password, hasSpecialChar);
}

```

- At least 8 characters long.
- Contains at least one uppercase letter, one lowercase letter, and one special character.



## Email Validation

The **isValidEmail** function uses a regex pattern to validate email addresses.

```
bool isValidEmail(const string& email) {  
    const regex pattern("(\\w+)(\\.|_)?(\\w*)@(\\w+)(\\. (\\w+))+" );  
    return regex_match(email, pattern);  
}
```

## View Donation History

The **view\_donation\_history** function retrieves and displays the donor's details from the respective blood group file, including personal details, donation history, next donation eligibility, and health status. It also ensures the user is shown appropriate messages in case their record is not found.

## Login Dashboard:

After a user successfully logs in, they are directed to a personalized **Login Dashboard**. This dashboard offers various options depending on the user's membership status in the Red Army:

- If the user has not joined the Red Army, they are offered the option to **Join Red Army**.
- If the user is already a Red Army member, they can view their **Donation History** blood groups, or log out.

```
void login_dashboard(string blood_group, string phone) {  
    bool isRedArmyMember = hasJoinedRedArmy(blood_group, phone);  
  
    refresh_screen();  
    cout << "~~~~~\n";  
    cout << "~~~~~| Dashboard |~~~~~\n";  
    cout << "~~~~~\n";  
  
    if (!isRedArmyMember) {  
        cout << "~~~~~\n";  
        cout << "~~~~~| [1] Join Red Army |~~~~~\n";  
        cout << "~~~~~\n";  
    } else {  
        cout << "~~~~~\n";  
        cout << "~~~~~| [1] View Donation |~~~~~\n";  
        cout << "~~~~~| History |~~~~~\n";  
        cout << "~~~~~\n";  
    }  
  
    cout << "~~~~~\n";  
    cout << "~~~~~| [2] Update Profile |~~~~~\n";  
    cout << "~~~~~\n";  
  
    cout << "~~~~~\n";  
    cout << "~~~~~| [3] Search Blood |~~~~~\n";  
    cout << "~~~~~\n";  
  
    cout << "~~~~~\n";  
    cout << "~~~~~| [4] Logout |~~~~~\n";  
    cout << "~~~~~\n";  
}
```

```
cout << "~~~~~ Enter your choice: ";  
int dashboard_option;  
cin >> dashboard_option;  
refresh_screen();  
  
switch (dashboard_option) {  
    case 1:  
        if (!isRedArmyMember) {  
            join_red_army(blood_group, phone);  
        } else {  
            view_donation_history(blood_group, phone);  
        }  
        break;  
    case 2:  
        update_profile(blood_group, phone);  
        break;  
    case 3:  
        search_donor(phone);  
        break;  
    case 4:  
        cout << "~~~~~ Logging out... Please wait.~~~~~\n";  
        Sleep(2000);  
        refresh_screen();  
        entry_dashboard();  
        break;  
    default:  
        cout << "~~~~~ Invalid input. Please try again.~~~~~\n";  
        Sleep(2000);  
        login_dashboard(blood_group, phone);  
}
```

Here's how the **Login Dashboard** function is structured:

## Join Red Army

The `join_red_army` function allows registered users to join as blood donors by providing

```
void join_red_army(string blood_group, string phone) {
    string email;
    int age;
    int weight;
    bool isFirstTime;
    string lastDonationTime = "N/A";
    bool hasDisease;

    do {
        cout << "\t\t\t\t\tEnter your email address: ";
        cin >> email;

        if (!isValidEmail(email)) {
            cout << "\t\t\t\t\tInvalid email format. Please try again";
        } else {
            ifstream checkFile(blood_group + ".txt");
            string line;
            bool emailExists = false;

            while (getline(checkFile, line)) {
                if (line.find(email) != string::npos) {
                    emailExists = true;
                    break;
                }
            }
            checkFile.close();

            if (emailExists) {
                cout << "\t\t\t\t\tThis email is already registered";
                continue;
            }
        }
    } while (true);

    do {
        cout << "\t\t\t\t\tEnter your age: ";
        cin >> age;

        if (cin.fail() || age < 18 || age > 65) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "\t\t\t\t\tInvalid age. You must be between 18 and 65 years old";
        } else {
            break;
        }
    } while (true);

    do {
        cout << "\t\t\t\t\tEnter your weight (in kg): ";
        cin >> weight;

        if (cin.fail() || weight < 50) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "\t\t\t\t\tInvalid weight. You must weigh at least 50 kg to donate blood.";
        } else {
            break;
        }
    } while (true);

    cout << "\t\t\t\t\tIs this your first donation?\n";
    isFirstTime = choice_menu();
    cin.ignore();

    if (!isFirstTime) {
        bool validDate;
        do {
            cout << "\t\t\t\t\tWhen was your last donation? (YYYY-MM-DD): ";
            getline(cin, lastDonationTime);

            regex datePattern("\\d{4}-\\d{2}-\\d{2}");
            validDate = regex_match(lastDonationTime, datePattern);

            if (!validDate) {
                cout << "\t\t\t\t\tInvalid date format. Please use YYYY-MM-DD format.\n";
            }
        } while (!validDate);
    }

    cout << "\t\t\t\t\tPlease answer the following health questions:\n\n";

    bool hasAnyDisease = false;

    cout << "\t\t\t\t\tDo you have or have you ever had any of the following:\n";
    cout << "\t\t\t\t\tHIV/AIDS\n";
    hasAnyDisease |= choice_menu();

    cout << "\t\t\t\t\tHepatitis B or C\n";
    hasAnyDisease |= choice_menu();

    cout << "\t\t\t\t\tAny blood disorders or blood-related diseases\n";
    hasAnyDisease |= choice_menu();

    cout << "\t\t\t\t\tAny chronic diseases\n";
    hasAnyDisease |= choice_menu();

    if (hasAnyDisease) {
        cout << "\t\t\t\t\tYou are not eligible to join the Red Army due to health reasons. Please consult with a healthcare provider for more information.\n";
        return;
    }

    ofstream updateFile(blood_group + ".txt", ios::app);
    updateFile << email << "\n";

    ofstream updateFile(blood_group + ".txt", ios::app);
    updateFile << lastDonationTime << "\n";

    ofstream updateFile(blood_group + ".txt", ios::app);
    updateFile << weight << "\n";

    ofstream updateFile(blood_group + ".txt", ios::app);
    updateFile << age << "\n";

    ofstream updateFile(blood_group + ".txt", ios::app);
    updateFile << hasAnyDisease << "\n";

    cout << "\t\t\t\t\tCongratulations! You have successfully joined the Red Army!\n";
    cout << "\t\t\t\t\tThank you for your commitment to making blood.\n";
    cout << "\t\t\t\t\tYour next donation date will be calculated based on your last donation date.\n";

    return;
}
```

additional details. The function enforces strict validation and eligibility criteria to ensure

compliance with safety protocols.

- Email Validation:**
  - Ensures valid email format and uniqueness.
- Age and Weight Criteria:**
  - Confirms the donor is between 18 and 65 years old and weighs at least 50 kg.
- Health Screening:**
  - Verifies eligibility through health-related questions.
- Data Integration:**
  - Updates user records in the respective blood group file.
- User Feedback:**
  - Provides confirmation messages or detailed reasons for ineligibility.

## 6. Safety Compliance:

- Adheres to medical safety standards for blood donation.

## 1. Data Storage and File Organization:

Donor records are stored in text files corresponding to each blood group (e.g., A+.txt, O-.txt). Each record in the file is stored in a **comma-separated values (CSV)** format with the following fields:

- **ID:** Unique identifier (phone number or user ID)
- **Password:** User login password
- **Name:** Full name of the donor
- **Email:** Contact email address
- **Age:** Age of the donor
- **Weight:** Weight of the donor
- **First-time Donor:** Indicator (Yes/No) if this is the donor's first donation
- **Last Donation Date:** Date of the last donation (in YYYY-MM-DD format)
- **Has Any Disease:** Indicator (Yes/No) if the donor has any diseases

Example:

```
01768640957,password123,Tahia Nazim Ahona,nazimtahia@gmail.com,22,55,No,2024-03-17,No
```

## 2. Active Status Calculation:

The active status of a donor is determined based on the date of their last donation. A donor is considered **inactive** if the difference between the current date and their last donation date exceeds **120 days**. If the difference is less than or equal to 120 days, the donor is marked as **active**.

The system checks this by calculating the number of days between the last donation date and the current date. The active status is determined dynamically during the search and update processes.

Here's how the active status calculation works:

- The **last donation date** is stored in the donor's record.
- The **current date** is calculated based on the system's date.
- The difference in days between the two dates is calculated.

- If the difference is greater than **120 days**, the donor is marked **inactive**; otherwise, the donor is marked **active**.

A sample C++ code snippet for this calculation:

```
bool isEligibleDonor(const string& last_donation_date) {  
  
    int year, month, day;  
    char dash1, dash2;  
    stringstream date_stream(last_donation_date);  
    date_stream >> year >> dash1 >> month >> dash2 >> day;  
  
    tm last_donation_tm = {};  
    last_donation_tm.tm_year = year - 1900;  
    last_donation_tm.tm_mon = month - 1;  
    last_donation_tm.tm_mday = day;  
  
    time_t last_donation_time = mktime(&last_donation_tm);  
    time_t current_time = time(nullptr);  
    double seconds_diff = difftime(current_time, last_donation_time);  
    int days_diff = seconds_diff / (60 * 60 * 24);  
  
    return days_diff >= 120;  
}
```

### Update Profile Function:

This function allows users to update their profile details, such as email, age, weight, and last donation date. The function reads the existing profile from a file, prompts the user for the desired update, validates the input, and then saves the updated data back to the file.



```

if (update_successful) {

    ifstream read_file(blood_group + ".txt");
    vector<string> updated_lines;
    while (getline(read_file, line)) {
        if (line.find(phone + ",") == 0) {

            string updated_line = "";
            for (size_t i = 0; i < current_info.size(); i++) {
                updated_line += current_info[i];
                if (i < current_info.size() - 1) {
                    updated_line += ",";
                }
            }
            updated_lines.push_back(updated_line);
        } else {
            updated_lines.push_back(line);
        }
    }
    read_file.close();

    ofstream write_file(blood_group + ".txt");
    for (const string& updated_line : updated_lines) {
        write_file << updated_line << endl;
    }
    write_file.close();

    cout << "\n\t\t\t\t\tProfile updated successfully!\n";
    Sleep(2000);

    login_dashboard(blood_group, phone);
}

```

### Key Points of Functionality:

1. **User Data Retrieval:** The function opens the file corresponding to the user's blood group and searches for the user's data based on their phone number.
2. **Profile Display:** It displays the current information such as name, phone number, and blood group.
3. **Update Options:** The user can choose to update their email, age, weight, last donation date, or eligibility status.
4. **Validation:** It validates the inputs such as email format, age range, and donation date.
5. **Data Update:** The updated data is saved back to the file.
6. **Return to Dashboard:** After the profile is updated, the user is returned to the dashboard.

### Integration with the Rest of the System:

- This function fits into the **Login Dashboard** where users can update their profile.
- It utilizes file operations to read and write user data and provides feedback on the update process.

This function allows users to search for eligible blood donors by selecting a blood group. It retrieves donor data from a file corresponding to the selected blood group, filters out the eligible donors, and displays their details.

```
while (getline(file, line)) {  
    stringstream ss(line);  
    string item;  
    vector<string> donor_info;  
  
    while (getline(ss, item, ',')) {  
        donor_info.push_back(item);  
    }  
  
    if (donor_info.size() == 9 && donor_info[0] != "No") {  
        if (isEligibleDonor(donor_info[7])) {  
            eligible_donors.push_back(donor_info);  
        }  
    }  
}  
file.close();  
  
if (eligible_donors.empty()) {  
    refresh_screen();  
    cout << "\n\n\n\n\n\n\n\n\n\n";  
    cout << "No eligible donors found for \n";  
    cout << "\nBlood Group: " << setw(18) << left << blood_group << " \n";  
    cout << "\n\n\n\n\n\n\n\n\n\n";  
} else {  
    refresh_screen();  
    cout << "\n\n\n\n\n\n\n\n\n\n";  
    cout << "Eligible donors found for \n";  
    cout << "\nBlood Group: " << setw(18) << left << blood_group << " \n";  
    cout << "\nTotal Eligible Donors: " << eligible_donors.size() << " \n";  
    cout << "\n\n\n\n\n\n\n\n\n\n";
```

[illegible]

1. **User Input for Blood Group:** The user is prompted to choose a blood group from a list.
2. **File Reading:** The system looks for a file named after the selected blood group (e.g., A+.txt) and reads the donor data from it.
3. **Eligibility Check:** It filters out donors based on eligibility criteria (e.g., medical status and last donation).
4. **Displaying Eligible Donors:** If eligible donors are found, their details (name, phone, email, and eligibility status) are displayed in a formatted manner.
5. **Return to Dashboard:** After viewing the results, the user is given the option to return to the dashboard.

- This function integrates into the **Login Dashboard**.
- It is accessible after a user logs in, allowing them to search for blood donors by blood group.

### 4.3 Modules

- **Login & Registration:** Ensures secure user access.
- **Donor Activation:** Allows users to volunteer as donors.
- **Account Management:** Provides user-specific details.
- **Search Donor:** Enables searching for donors by blood group.
- **File Handling System:** Maintains donor records in separate files categorized by blood groups to enhance search efficiency.



# Chapter 5

## User Manual

---

### 5.1 System Requirements

#### Software Requirements:

- **Operating System:** Windows 10 or later.
- **Programming Environment:** Visual Studio Code with C++ extensions installed.
- **File Handling:** CSV files are used for data storage and retrieval.

#### Hardware Requirements:

- **Processor:** Intel i3 or above.
- **RAM:** Minimum 4 GB.
- **Storage:** At least 10 GB of free disk space.

### 5.2 Log In Credentials

Blood Group: B+ (Log In Option 3)

Phone: 01897911901

Password: @RedAid1

## Chapter 6

# CONCLUSION

---

### 6.1 Findings

The development and implementation of the blood management system have demonstrated significant improvements in the efficiency and accessibility of blood donation processes. Key findings include:

- **User Interaction:** The system facilitates easy user registration, profile updates, and seamless access to donor information through a simple yet intuitive dashboard.
- **Data Storage and Retrieval:** Efficient use of text files for storing blood donor data has proven to be effective for a small-scale application, ensuring quick access and modifications to donor records.
- **Eligibility Filtering:** The integration of eligibility criteria (such as age, weight, and last donation date) has allowed the system to ensure that only eligible donors are selected, ensuring that the blood collection process maintains a high standard.
- **Flexibility:** The system is flexible enough to adapt to various blood types and can be expanded to include more complex features like real-time data processing and integration with hospital management systems.

### 6.2 Limitations

While the blood management system provides useful features, there are certain limitations:

- **File-Based Storage:** The reliance on text files for storing user data limits the scalability of the system. This approach can be slow and error-prone as the number of records increases.
- **Manual Data Entry:** Data is input manually by the user, which could lead to errors such as incorrect phone numbers or blood group choices. A more robust validation system could be implemented to reduce such errors.
- **Lack of Real-Time Features:** The system does not include real-time features like notifications or updates, which could enhance the user experience. For example, donors could be notified about upcoming donation drives or events.
- **Security Concerns:** The system does not incorporate advanced security measures for user data protection, such as encryption or password hashing. As the system expands, implementing these features would be crucial for protecting sensitive user information.

### 6.3 Future Works

To enhance the functionality and performance of the blood management system, the following future developments are recommended:

- **Database Integration:** Moving from a file-based system to a relational database would significantly improve performance, data security, and scalability. A database management system (DBMS) like MySQL or PostgreSQL could be implemented to handle larger datasets and complex queries.
- **User Authentication:** Adding a user authentication system would improve security by ensuring that only authorized individuals can access or modify sensitive information.
- **Real-Time Notification System:** Incorporating a notification system to alert users about donation requests, blood availability, and upcoming donation events could improve user engagement and streamline the blood donation process.
- **Mobile Application:** Developing a mobile app for users to track their donation history, find nearby donation events, and receive notifications would increase the system's accessibility and usability.
- **Advanced Data Analytics:** Future versions of the system could incorporate analytics to identify trends in blood donations, such as demand surges during emergencies or specific demographic groups that are more likely to donate. This data could help optimize donation drives and resource distribution.

# REFERENCES

---

## 1. Programming Language Resources:

- Stroustrup, B. (2013). *The C++ Programming Language (4th Edition)*. Addison-Wesley Professional.
- Official C++ Documentation: <https://cplusplus.com>

## 2. Development Tools:

- Visual Studio Code: Integrated Development Environment. Available at: <https://code.visualstudio.com>
- GCC Compiler: GNU Compiler Collection for C++ development. Documentation: <https://gcc.gnu.org>

## 3. Design and Modeling Techniques:

- Use Case Diagram and Context Diagram Tutorials: Creately, <https://creately.com/guides/use-case-diagram-tutorial/>
- Data Flow Diagram Guide: Lucidchart, <https://www.lucidchart.com/pages/data-flow-diagram>

## 4. File Management and Storage Techniques:

- CSV File Handling in C++: GeeksforGeeks, <https://www.geeksforgeeks.org>

## 5. Project Management and Models:

- Waterfall Model Overview: InterviewBit, <https://www.interviewbit.com/blog/waterfall-model/>