Blood Donation Recording System.

Software Requirements Specification Document.

*This project was developed as part of my coursework for Software Engineering I course at The British University in Egypt.

By: Mohamed Saleh Abdelsalam

Email: mohamedsaleh.ms144@gmail.com

2024

Project Title: Blood Donation Record System

A system in which data of Employee, Donee, Donor, Blood bank details would be saved and will be interrelation with each other. It manages the donation process, adds new donation, schedule appointment, recruit employee in the system, and store record of blood unit donated. It also takes request from the user for donation.

Functional Requirements:

1. Login User: The user should provide the system by the necessary information such as name, contact info, blood type, and any medical history.
2. Register Donor: The donor should submit a donation request The system should allow authorized users to have access to submit a request for donating blood.
3. Make Donation: The system shall allow registered donors to schedule donation appointments and complete pre-donation assessments.
4. Schedule Appointment: The system should allow blood donation centers to schedule appointments for donors, ensuring efficient allocation of resources and minimizing waiting times. Donors should be able to view available time slots, select a convenient appointment, and receive confirmation.
5. Track Donation record: track the donated blood units including data like donor information, blood type, available quantity, expiration dates, and storage location.
6. Give Feedback: the system shall provide a way for donors to provide feedback on their donation experience, including suggestions for improvement.
7. Request Blood: the system shall provide a way for done and blood bank to submit a blood request into the system

Non-Functional Requirements:

1. Security: The system should ensure the confidentiality, integrity, and availability of the blood donation records. It should have appropriate security measures in place to protect sensitive information, such as personal details and medical history, from unauthorized access or disclosure. Additionally, the system should comply with relevant privacy regulations, such as data protection laws, to safeguard the privacy of donors' information. It is required to have a backup of the database every 6 hours.

2.Portability: The system should be compatible with a wide range of web browsers and operating systems such as Windows, Linux, UNIX, and Macintosh to ensure accessibility for users with diverse technology preferences.

3. Reliability: The system shall have high availability, with unscheduled downtime to be 10% or less to ensure continuous access for donors, staff, and administrators.

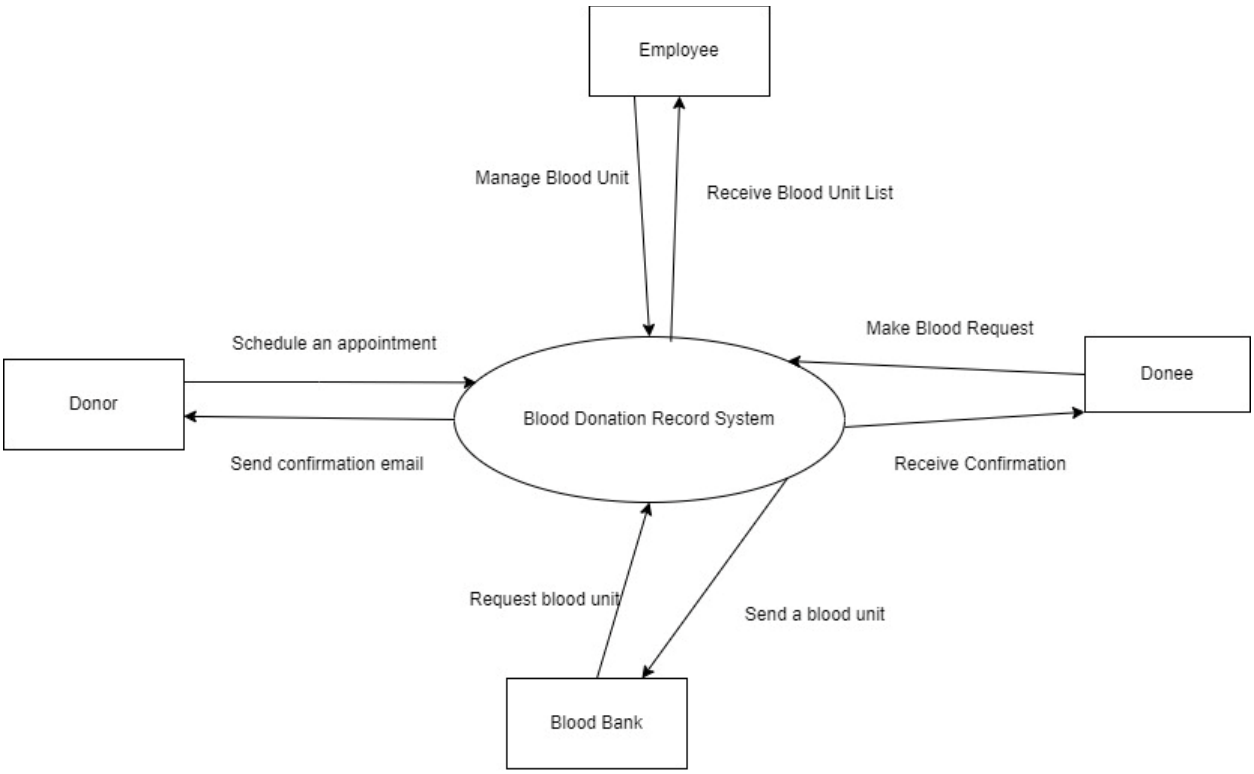
4.Speed: The system shall respond to user interactions within 2 seconds for tasks as donor registration, appointment scheduling, and donation recording.

5. Ease of use: User surveys indicate a satisfaction rate of at least 85% for the system's ease of use, with specific questions addressing the clarity of navigation, intuitiveness of workflows, and overall user experience.

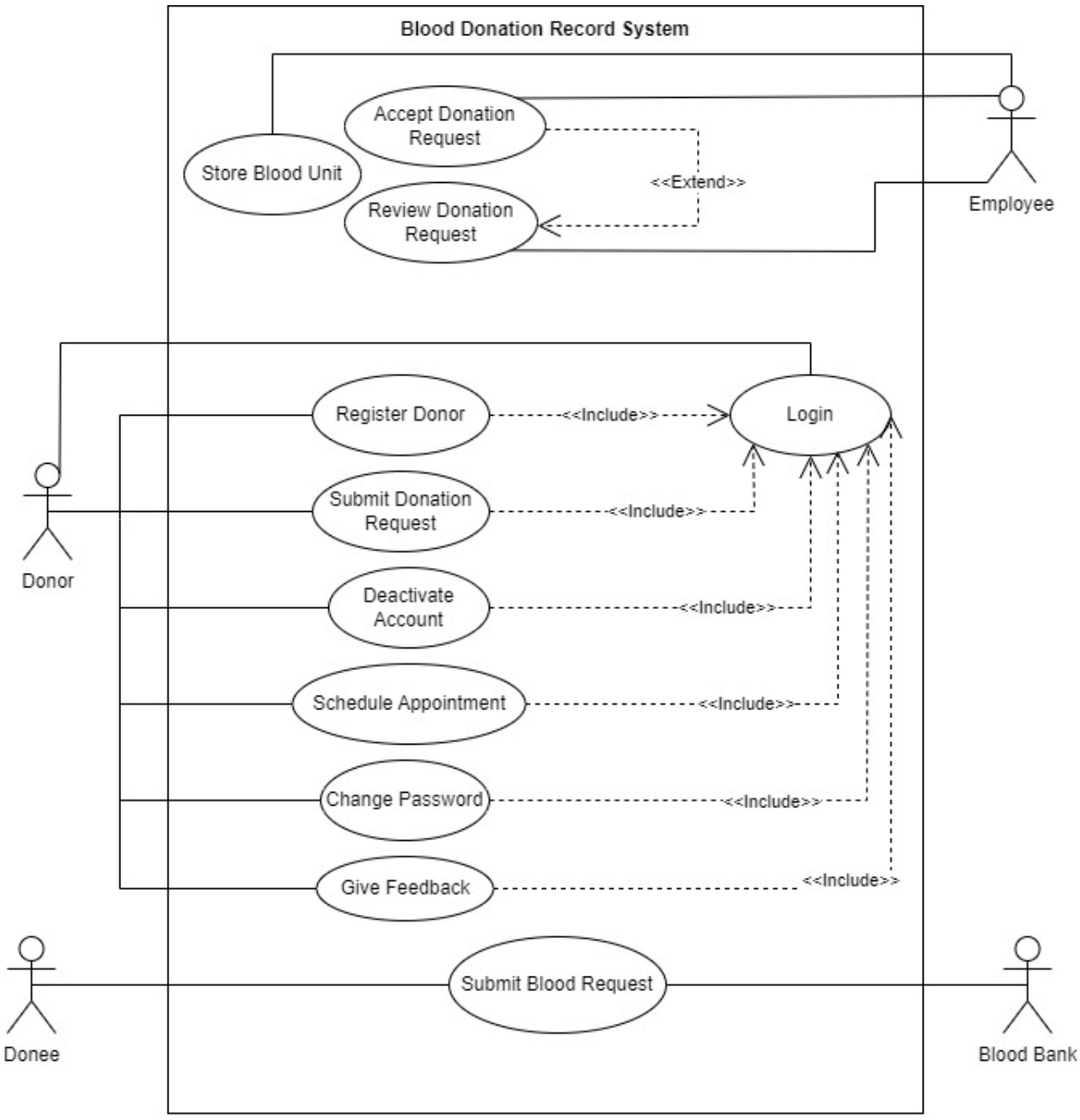
6. Maintainability: The system should be designed and developed in a modular and maintainable manner, where the average time required to resolve a high-priority software defect shall not exceed 4 hours, measured from the time the defect is reported to its resolution, with performance tracked monthly.

7. Performance: The system shall support a minimum throughput of 1000 concurrent users without experiencing degradation in response time, as measured by the system's ability to handle peak load scenarios, with load testing conducted quarterly.

Context Diagram:



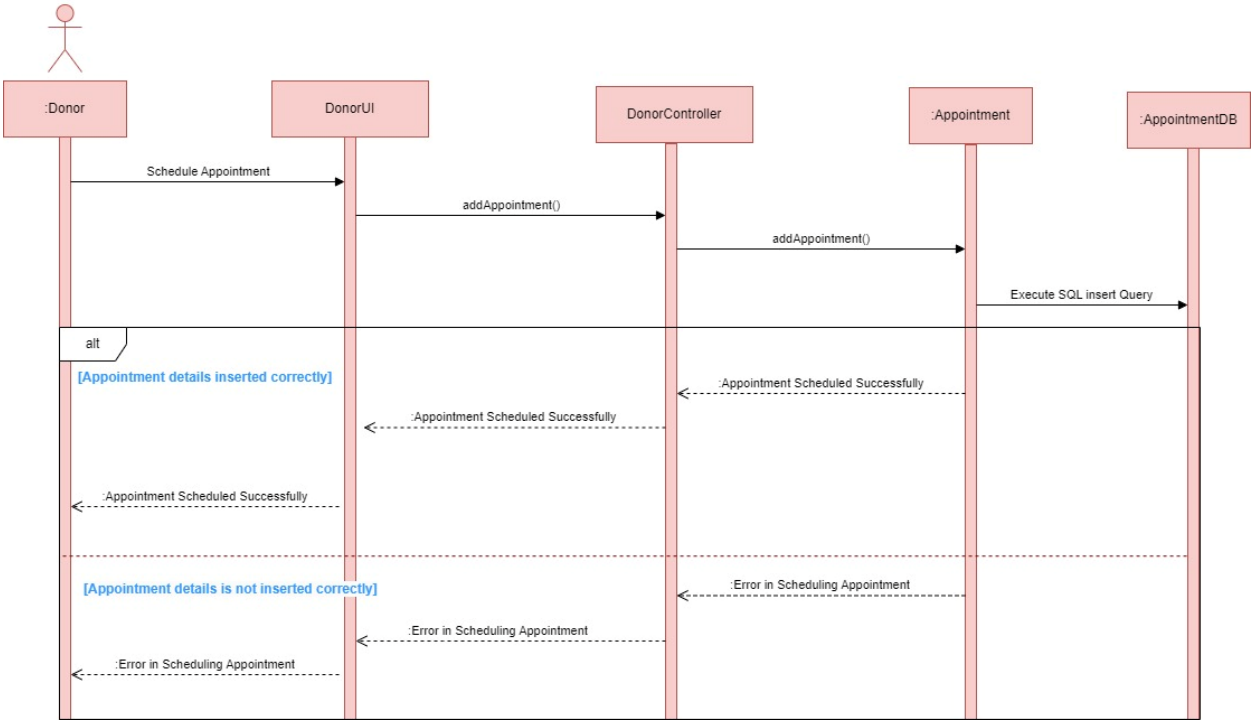
Use Case Diagram:



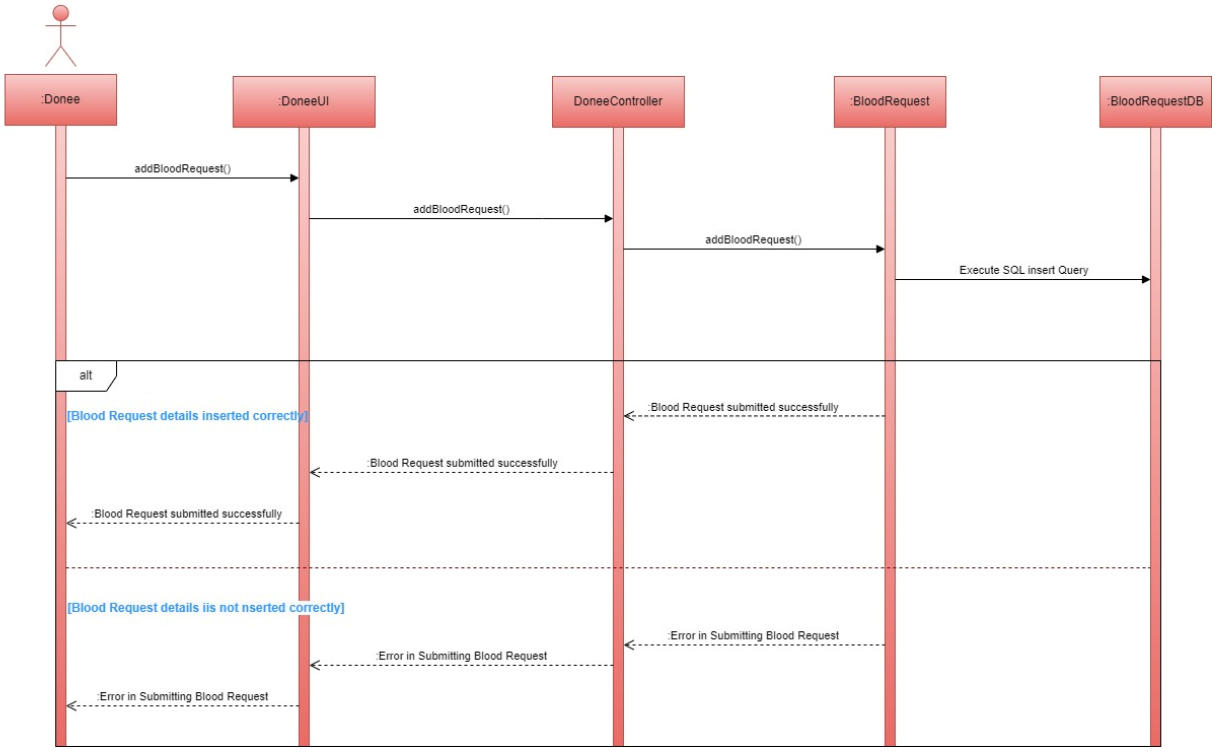
Use Case Scenario:

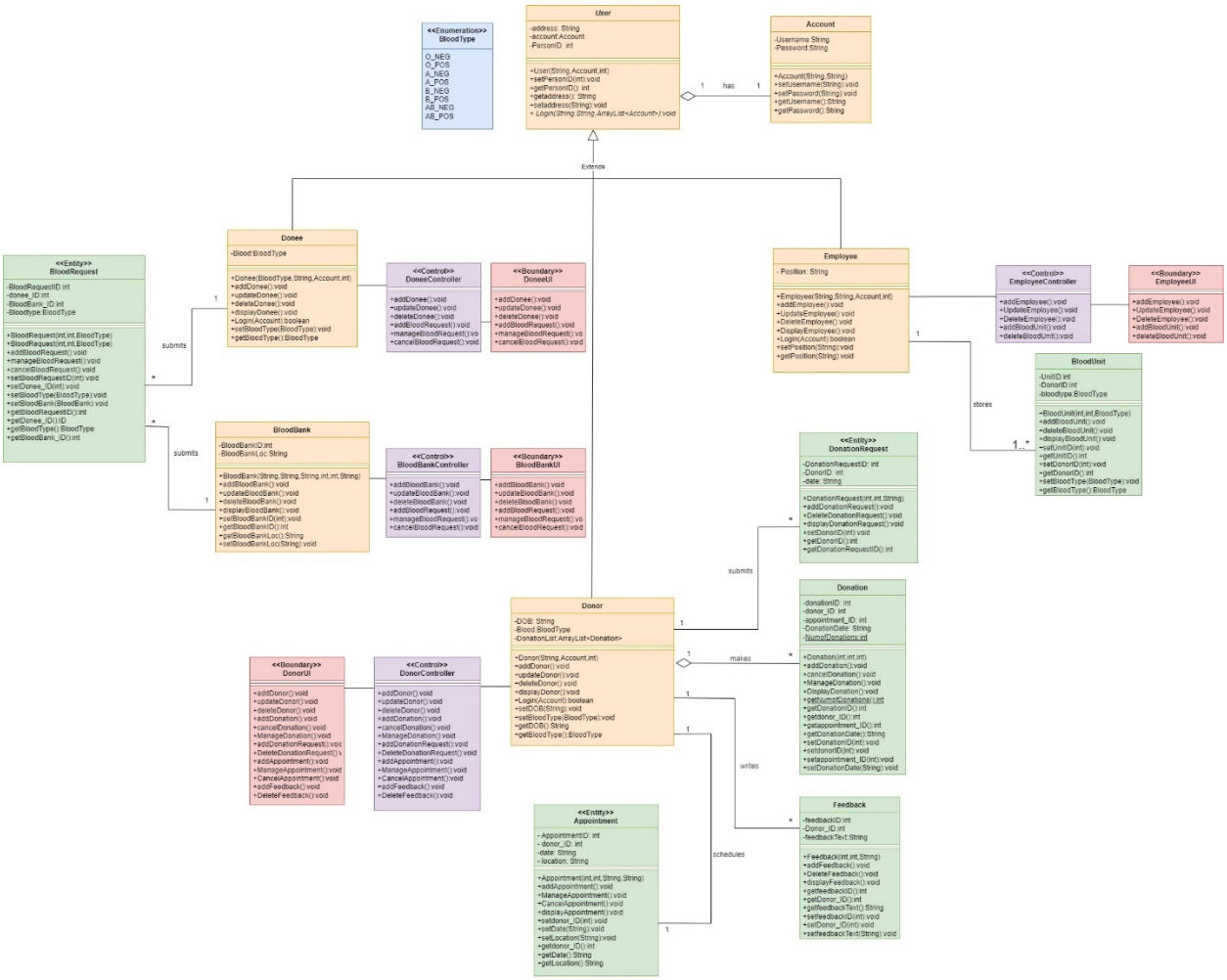
ID	UC_01
Title	Register Donor
Description	The Donor initiates the process of registering as a blood donor in the system.
Priority	High
Primary Actor	Donor
Secondary Actor	N/A
Preconditions	1.The user has data such as (user name and password) to enter the system, the Donor has account.
Post-conditions	1. The user successfully performs the requested action within the system. 2. Account changes (deactivation, password change) are reflected in the system's database.
Main Success Scenario	1. Log in: - The user enters the login page for the blood donation system. - The user enters his username and password. 2. Deactivate account: - The user goes to the account settings or profile section of the system. - The user selects the option to deactivate his account. 3. Change password: - The user enters the account settings or profile section of the system. - The user chooses the option to change his password. - After entering the required information, the system updates the user's password accordingly. 4. Blood request: - The user goes to the "Blood Requests" section of the system. - The user chooses the "Request new blood" option. - The system prompts the user to provide details about the blood request, including: - Type of blood required (whole blood, plasma, platelets, etc.). - The amount of blood required. - Reason for requesting blood (medical treatment, emergency, etc.). - Level of urgency (normal, urgent, emergency). - The user enters the necessary information accurately into the system. - Upon submission, the system records the blood request and initiates the workflow for processing.
Extensions	1a. If the user enters incorrect login credentials, the system will prompt him to try again or start the password reset process. 2b. If the user encounters problems while deactivating the account or changing the password, he or she can contact system support for assistance. 3c. If the blood request submitted by the user requires immediate attention (urgent or emergency), the system notifies the blood bank coordinator or relevant staff to expedite processing.

Blood Donation Record System: Schedule Appointment

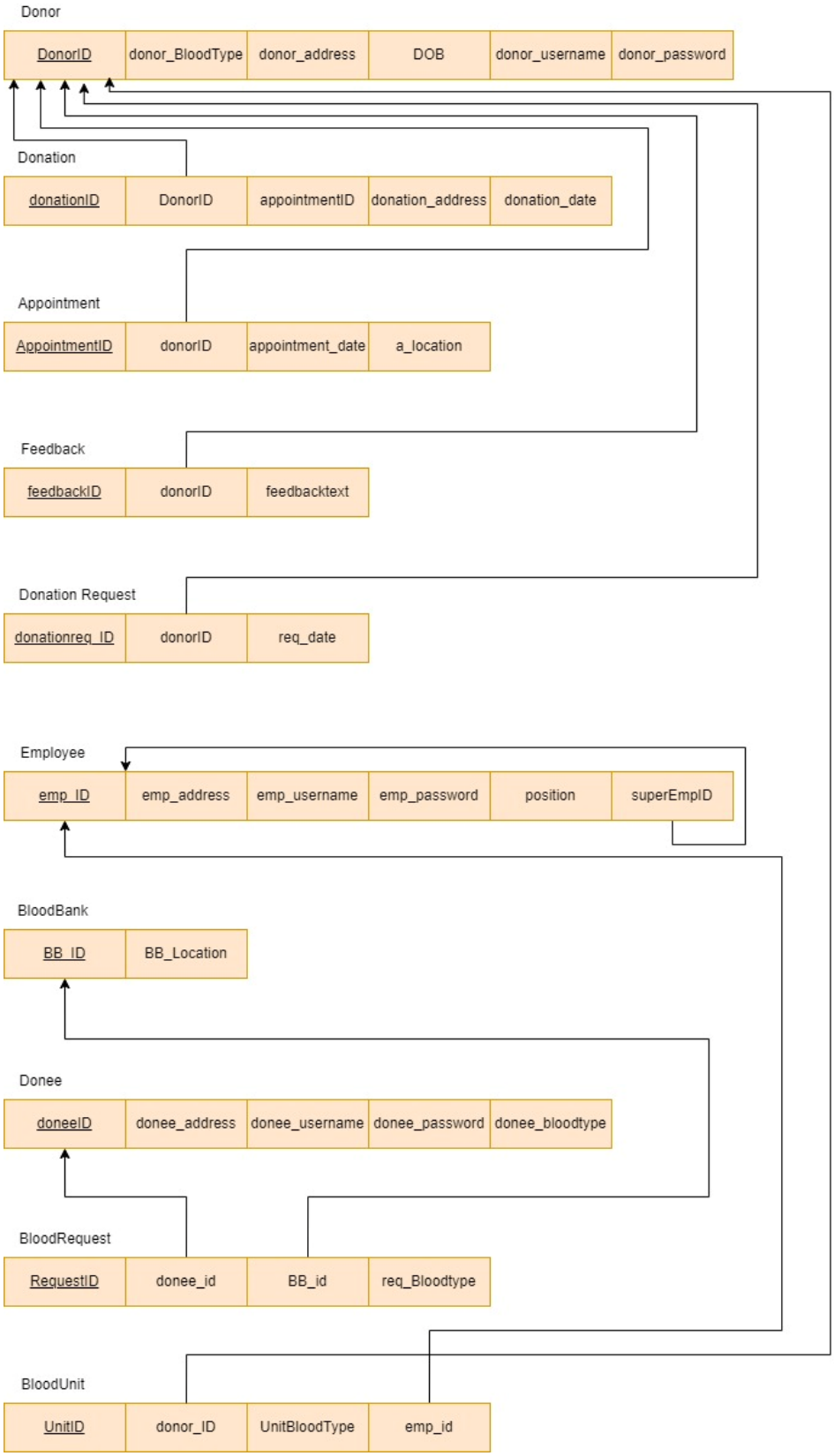


Blood Donation Record System: Submit Blood Request

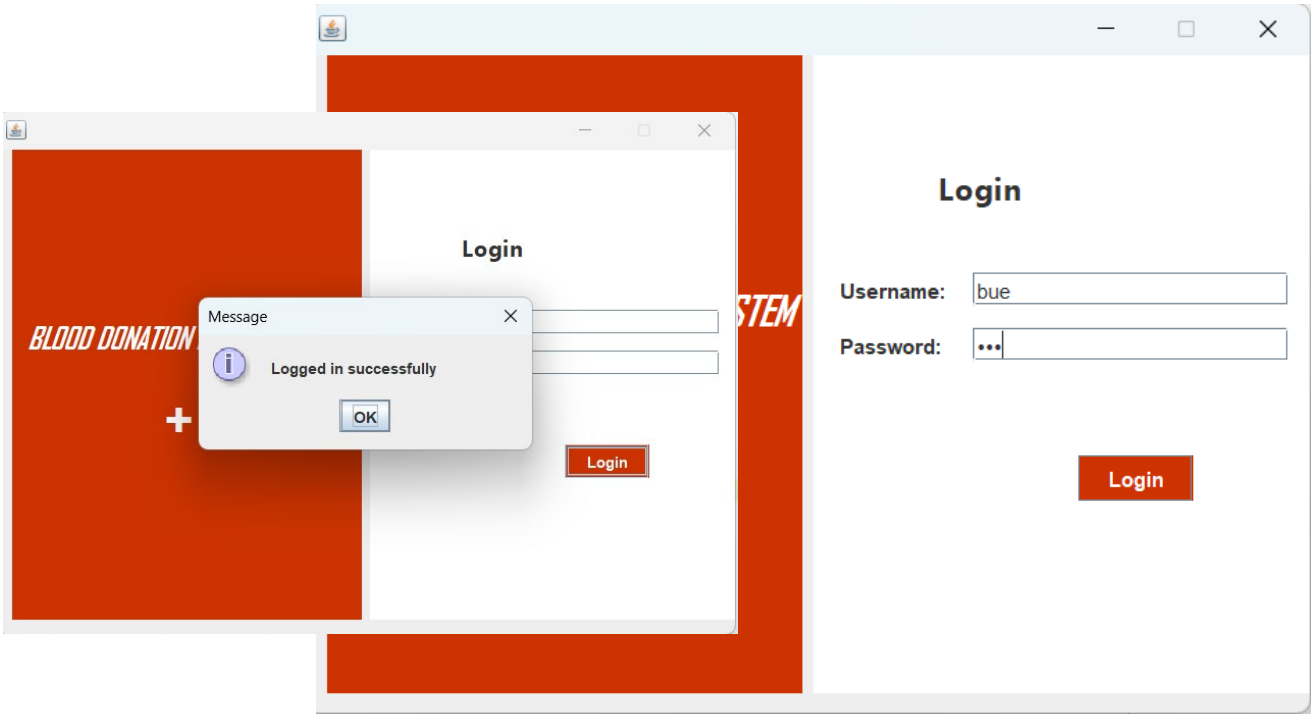


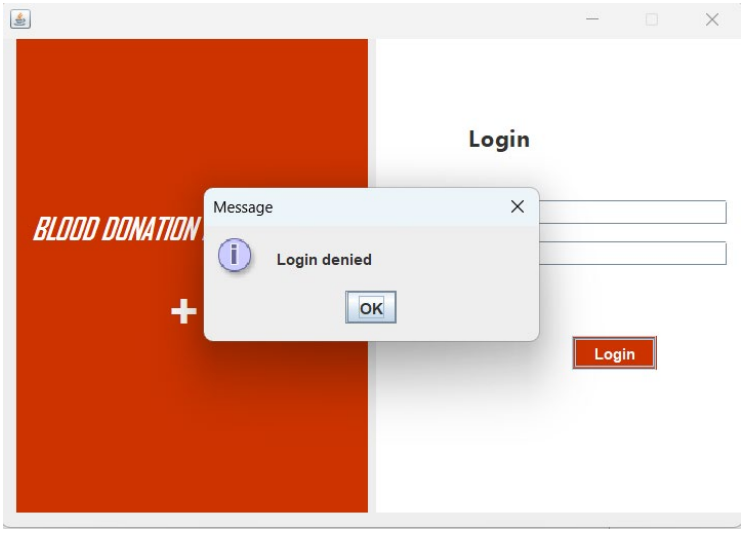

«Enumeration»
BloodType
O_NEG
O_POS
A_NEG
A_POS
B_NEG
B_POS
AB_NEG
AB_POS
User
-address: String
-account:Account
-PersonID: int
+User(String,Account,int)
+setPersonID(int):void
+getPersonID(): int
+getaddress(): String
+setaddress(String):void
+ Login(String,String,ArrayList<Account>):void
1
has
1
Account
-Username:String
-Password:String
+Account(String,String)
+setUsername(String):void
+setPassword(String):void
+getUsername():String
+getPassword():String
Extends
«Entity»
BloodRequest
-BloodRequestID:int
-donee_ID:int
-BloodBank_ID:int
-Bloodtype:BloodType
+BloodRequest(int,int,BloodType)
+BloodRequest(int,int,BloodType)
+addBloodRequest():void
+manageBloodRequest():void
+cancelBloodRequest():void
+setBloodRequestID(int):void
+setDonee_ID(int):void
+setBloodType(BloodType):void
+setBloodBank(BloodBank):void
+getBloodRequestID():int
+getDonee_ID():ID
+getBloodType():BloodType
+getBloodBank_ID():int
submits
1
*
submits
*
1
Donee
-Blood:BloodType
+Donee(BloodType,String,Account,int)
+addDonee():void
+updateDonee():void
+deleteDonee():void
+displayDonee():void
+Login(Account):boolean
+setBloodType(BloodType):void
+getBloodType():BloodType
«Control»
DoneeController
+addDonee():void
+updateDonee():void
+deleteDonee():void
+addBloodRequest():vo
+cancelBloodRequest():void
«Boundary»
DoneeUI
+addDonee():void
+updateDonee():void
+deleteDonee():void
+addBloodRequest():vo
+manageBloodRequest():vo
+cancelBloodRequest():void
BloodBank
-BloodBankID:int
-BloodBankLoc:String
+BloodBank(String,String,String,int,int,String)
+addBloodBank():void
+updateBloodBank():void
+deleteBloodBank():void
+displayBloodBank():void
+setBloodBankID(int):void
+getBloodBankID():int
+getBloodBankLoc():String
+setBloodBankLoc(String):void
«Control»
BloodBankController
+addBloodBank():void
+updateBloodBank():void
+deleteBloodBank():void
+addBloodRequest():void
+manageBloodRequest():vo
+cancelBloodRequest():void
«Boundary»
BloodBankUI
+addBloodBank():void
+updateBloodBank():void
+deleteBloodBank():void
+addBloodRequest():void
+manageBloodRequest():vo
+cancelBloodRequest():void
Employee
- Position: String
+Employee(String,String,Account,int)
+addEmployee():void
+UpdateEmployee():void
+DeleteEmployee():void
+DisplayEmployee():void
+Login(Account):boolean
+setPosition(String):void
+getPosition(String):void
«Control»
EmployeeController
+addEmployee():void
+UpdateEmployee():void
+DeleteEmployee():void
-addBloodUnit():void
+deleteBloodUnit():void
«Boundary»
EmployeeUI
+addEmployee():void
+UpdateEmployee():void
+DeleteEmployee():void
+addBloodUnit():void
+deleteBloodUnit():void
1
stores
1..*
BloodUnit
-UnitID:int
-DonorID:int
-bloodtype:BloodType
+BloodUnit(int,int,BloodType)
+addBloodUnit():void
+deleteBloodUnit():void
+displayBloodUnit():void
+setUnitID(int):void
+getUnitID():int
+setDonorID(int):void
+getDonorID():int
+setBloodType:BloodType):void
+getBloodType():BloodType
«Entity»
DonationRequest
-DonationRequestID: int
-DonorID: int
-date: String
+DonationRequest(int,int,String)
+addDonationRequest():void
+DeleteDonationRequest():void
+displayDonationRequest():void
+setDonorID(int):void
+getDonorID():int
+getDonationRequestID():int
*
submits
1
Donor
-DOB: String
-Blood:BloodType
-DonationList:ArrayList<Donation>
+Donor(String,Account,int)
+addDonor():void
+updateDonor():void
+deleteDonor():void
+displayDonor():void
+Login(Account):boolean
+setDOB(String):void
+setBloodType(BloodType):void
+getDOB():String
+getBloodType():BloodType
1
makes
*
Donation
-donationID: int
-donor_ID: int
-appointment_ID: int
-DonationDate: String
-NumofDonations:int
+Donation(int,int,int)
+addDonation():void
+cancelDonation():void
+ManageDonation():void
+DisplayDonation():void
+getNumofDonations():int
+getDonationID():int
+getdonor_ID():int
+getappointment_ID():int
+getDonationDate():String
+setDonationID(int):void
+setdonorID(int):void
+setappointment_ID(int):void
+setDonationDate(String):void
«Boundary»
DonorUI
+addDonor():void
+updateDonor():void
+deleteDonor():void
+addDonation():void
+cancelDonation():void
+ManageDonation():void
+addDonationRequest():voi
+DeleteDonationRequest():v
+addAppointment():void
+ManageAppointment():void
+CancelAppointment():void
+addFeedback():void
+DeleteFeedback():void
«Control»
DonorController
+addDonor():void
+updateDonor():void
+deleteDonor():void
+addDonation():void
+cancelDonation():void
+ManageDonation():void
+addDonationRequest():void
+DeleteDonationRequest():void
+addAppointment():void
+ManageAppointment():void
+CancelAppointment():void
+addFeedback():void
+DeleteFeedback():void
1
writes
*
Feedback
-feedbackID:int
-Donor_ID:int
-feedbackText:String
+Feedback(int,int,String)
+addFeedback():void
+DeleteFeedback():void
+displayFeedback():void
+getfeedbackID():int
+getDonor_ID():int
+getfeedbackText():String
+setfeedbackID(int):void
+setDonor_ID(int):void
+setfeedbackText(String):void
1
schedules
«Entity»
Appointment
- AppointmentID: int
- donor_ID: int
- date: String
- location: String
+Appointment(int,int,String,String)
+addAppointment():void
+ManageAppointment():void
+CancelAppointment():void
+displayAppointment():void
+setdonor_ID(int):void
+setDate(String):void
+setLocation(String):void
+getdonor_ID():int
+getDate():String
+getLocation():String

Database Schema:



Application Form 1: Login form

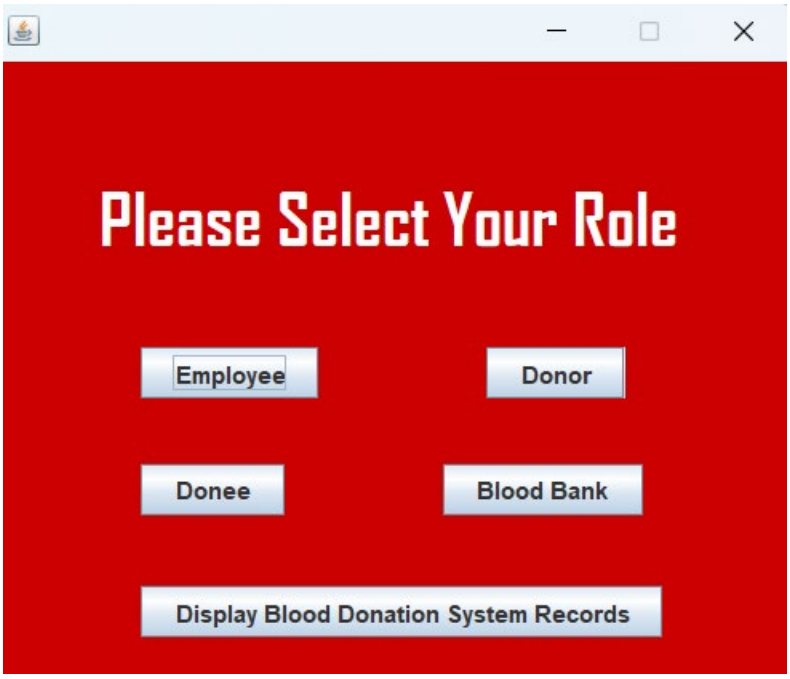




Description of Output:

At the moment of the system runs it displays Login form with username and password required, if you tried to skip it, it pop up and error. The Username and password are “bue”.

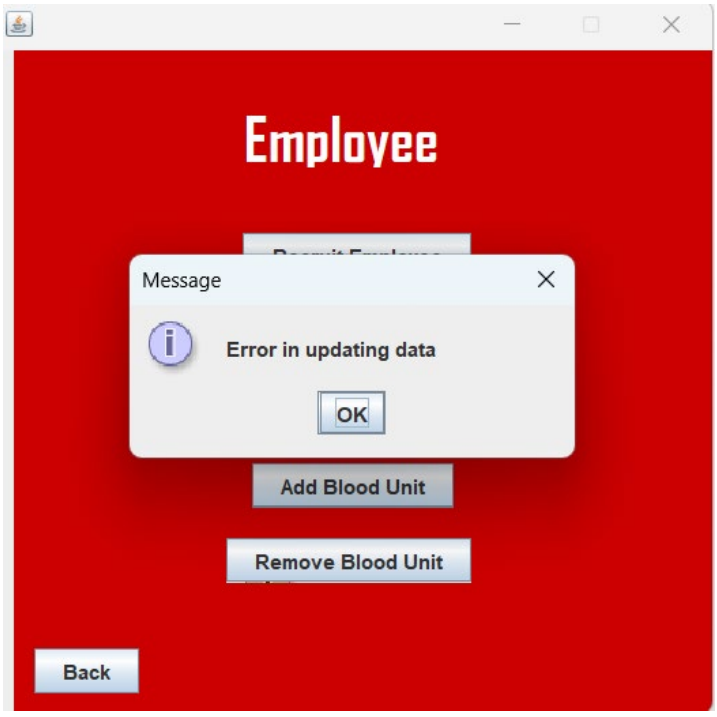
Application Form 2:Role

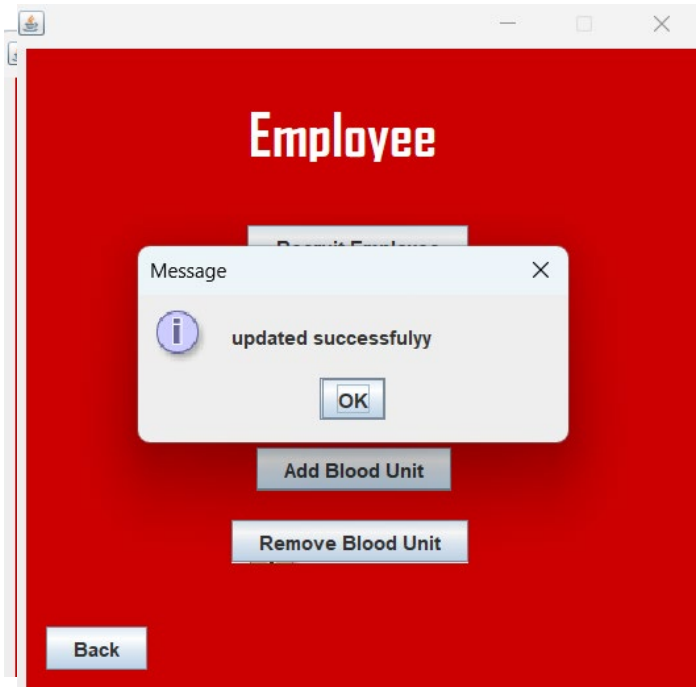


Description of Output:

After succeeding in logging in, a form that make user choose his role in the system or to display the system records.

Application Form 3: Employee form

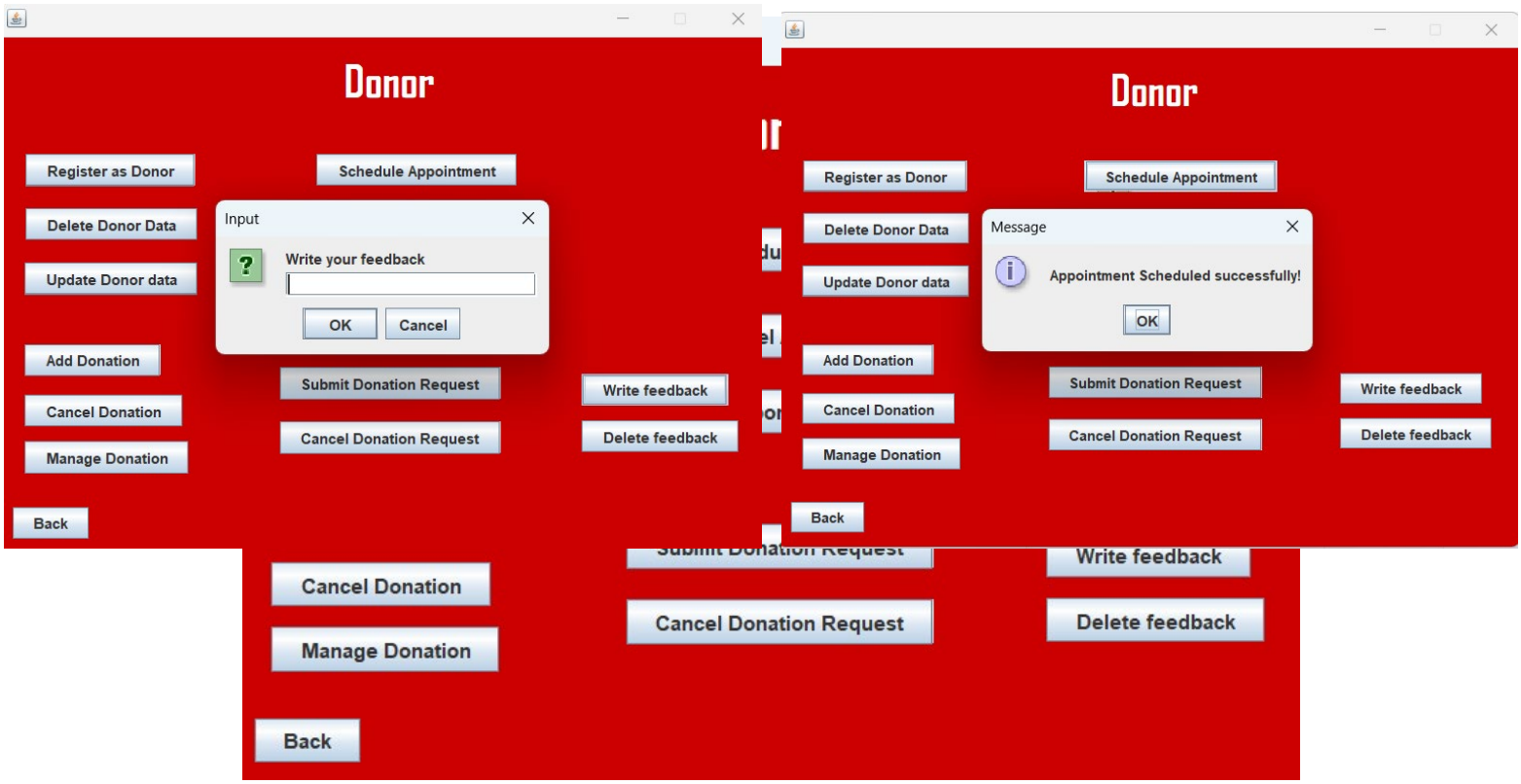




Description of Output:

After selecting Employee role the buttons describe functionalities that the employee can do and every button trigger a built-in function in the system and ask user for data to input then this data is inserted in the system's database, exceptions will be handled if data insertion in data base didn't go as it should.

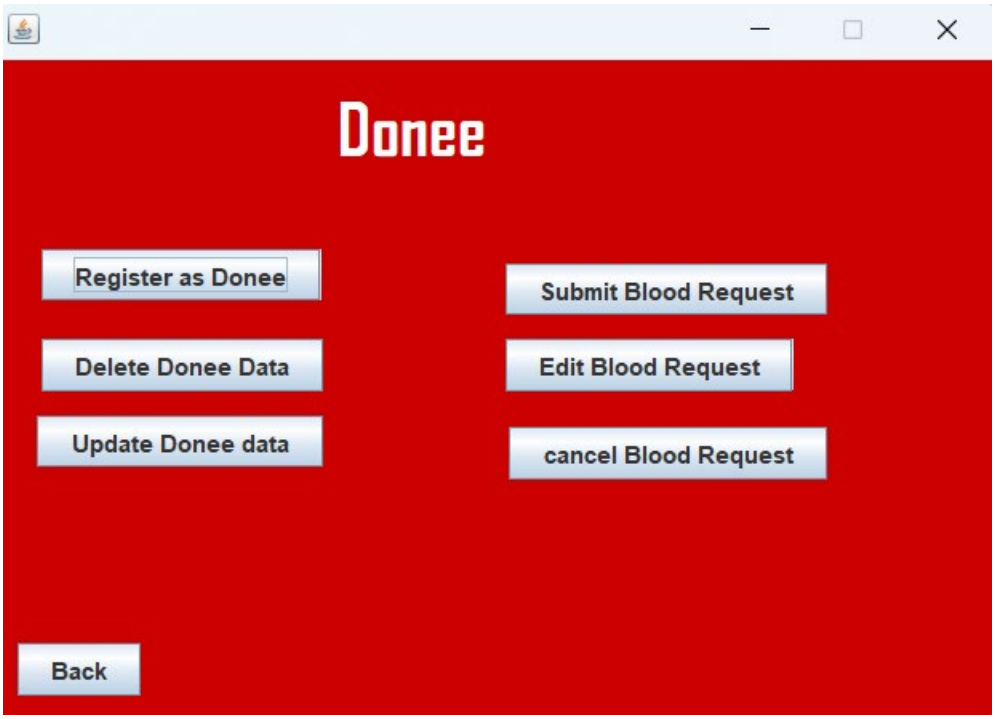
Application Form 4: Donor form

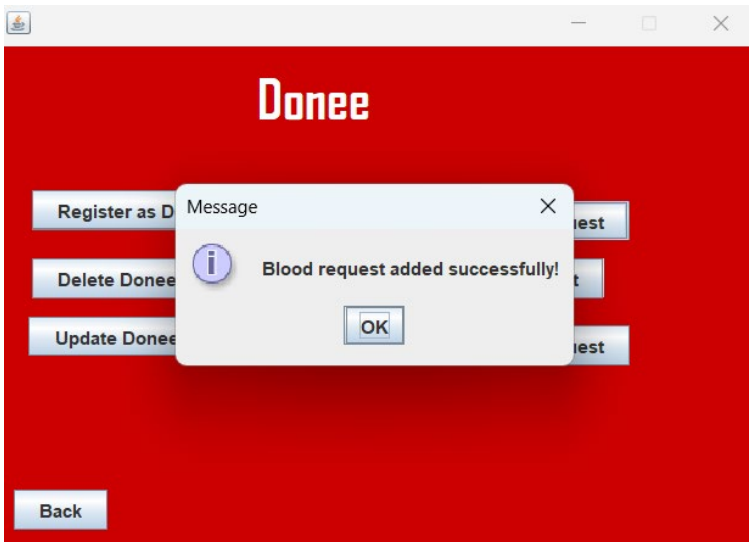


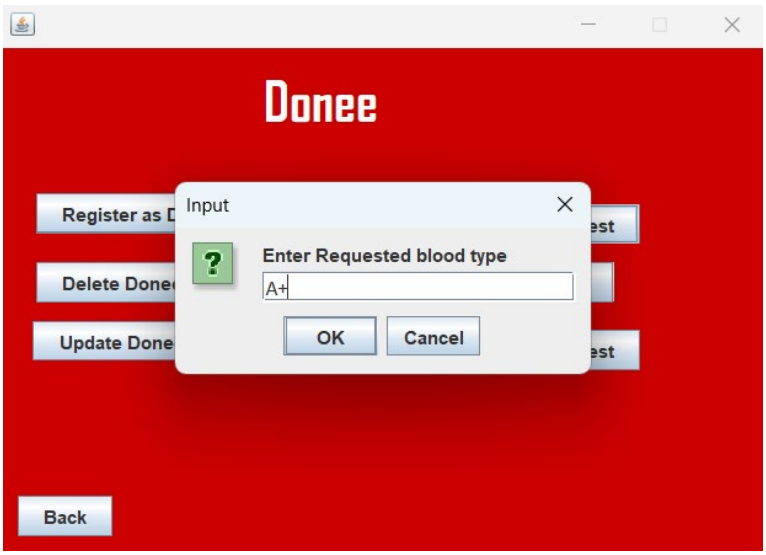
Description of Output:

After selecting Donor role the buttons describe functionalities that the donor can do and every button trigger a built-in function in the system and ask user for data to input then this data is inserted or manipulated in the system's database, exceptions will be handled if data insertion in data base didn't go as it should.

Application Form 5: Donee form



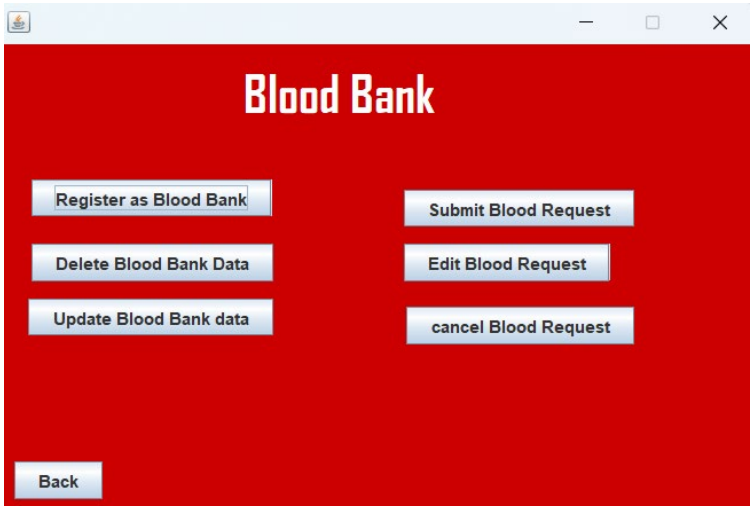




Description of Output:

After selecting the role donee a set of buttons will show the functionalities that the donee can do which is registration and managing donee data and submit blood request and all the data taken from user will be stored in the system database.

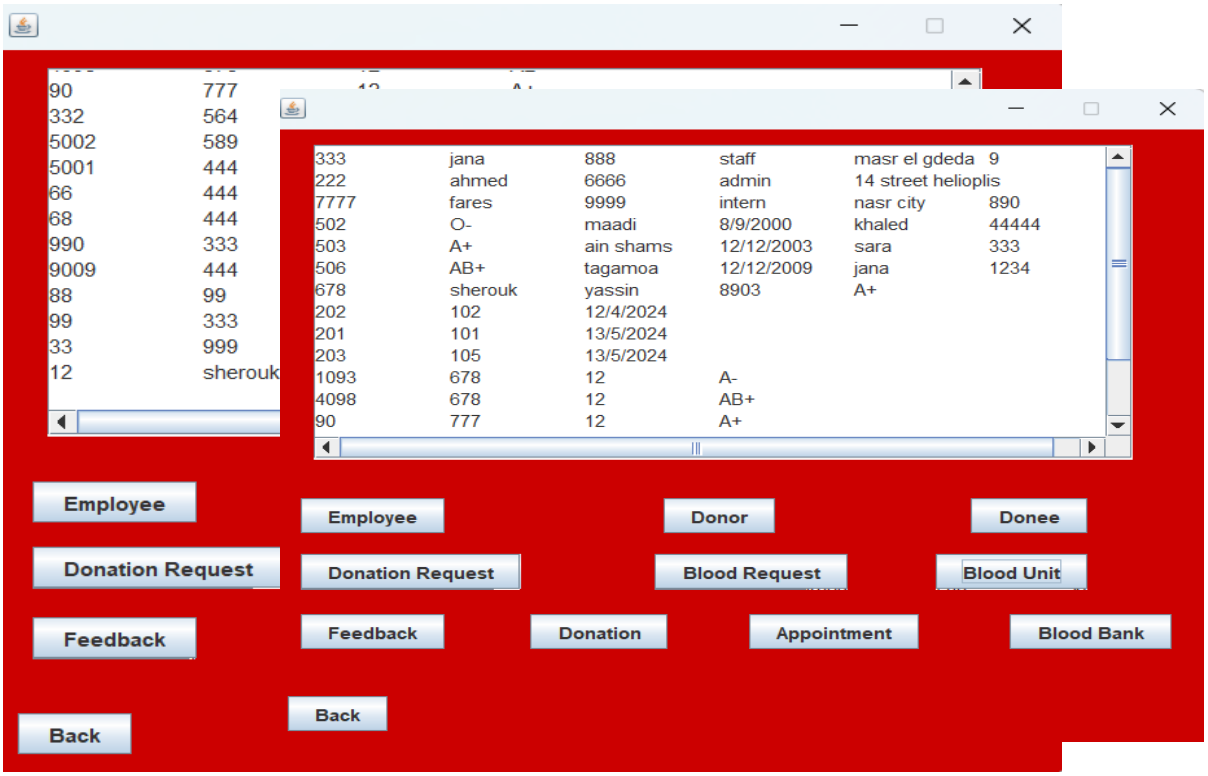
Application Form 6: Blood Bank form



Description of Output:

After selecting the role Blood Bank a set of buttons will show the functionalities that the Blood Bank can do which is registration and managing blood bank data and submit blood request and all the data taken from user will be stored in the system database.

Application Form 7: Display system records form



Description of Output:

After pressing the display records button from the second form a text area will show any type of data the user inserted in the system like Employee, Donor, Blood bank, etc. every button triggers a retrieve data query that selects all the records and show it in the text area.