# **Final Project Report: Blood Donation Management System (BDMS)**

**CS3009 - Software Engineering - Spring 2025  
Deliverable 3 - Final Deliverable  
Team Name**: ForgeCode Inc.  
**Team Members**:

* Hasnain Arshad (Team Lead, Frontend Developer)
* Ibraheem Farrukh (Frontend Developer)
* Aoun Jee (Testing and Documentation)  
  **Submission Date**: April 29, 2025

## **1. Project Introduction**

The **Blood Donation Management System (BDMS)** is a web-based application designed to streamline the process of blood donation by connecting donors and hospitals. The system facilitates donor registration, profile management, blood request creation, appointment scheduling, and donation tracking, ensuring efficient and timely blood supply management. The primary goal is to enhance accessibility, transparency, and coordination in the blood donation ecosystem, addressing critical healthcare needs.

The project was developed using an **Agile** process model, utilizing sprints to iteratively build and refine the system. The project stack includes:

* **Backend**: Spring Boot (Java)
* **Frontend**: HTML, CSS, JavaScript for a responsive user interface.
* **Database**: MySQL.
* **Tools**: Git for version control, Trello for task management, JUnit for unit testing, and Postman for API testing.

The BDMS serves donors (who register and manage profiles) and hospitals (who post blood requests), ensuring a seamless experience for all stakeholders.

## **2. Functional Requirements**

The BDMS includes the following functional requirements:

1. **User Registration and Authentication**:
   * Users can register as donors with personal details (email, password, first name, last name, blood type , date of birth, gender).
   * Users can log in and out securely.
2. **Donor Profile Management**:
   * Donors can view and update their profile (username, password etc.).
3. **Blood Request Management**:
   * Hospitals (admins) can post blood requests with details (blood type, urgency, hospital name).
   * Donors can view and respond to requests.
4. **Appointment Scheduling**:
   * Donors can schedule, reschedule, or cancel donation appointments at centers.
5. **Donation Tracking**:
   * Donors can view their donation history (date, center, amount, status).
6. **Health Tips**:
   * The system provides static health tips for donors.

## **3. Non-Functional Requirements**

1. **Performance**:
   * The system must handle up to 1,000 concurrent users with response times under 2 seconds.
2. **Security**:
   * User data must be encrypted (passwords using BCrypt) and transmitted over HTTPS.
   * Authentication must use session-based security with Spring Security.

## **4. User Stories**

Below are 15 user stories (5 per team member) that guided the development:

### **Ibraheem Farrukh (Backend Developer)**

1. As a donor, I want to register with my email, password, and blood type so that I can access the system.
2. As a donor, I want to log in securely so that I can access my dashboard.
3. As a hospital (admin), I want to post blood requests with urgency levels so that donors can respond.
4. As a donor, I want to view my donation history so that I can track my contributions.
5. As a system admin, I want user passwords to be encrypted so that data remains secure.

### **Hasnain Arshad (Frontend Developer)**

1. As a donor, I want a responsive dashboard to view my stats (blood type, last donation) so that I can stay informed.
2. As a donor, I want to update my profile details (username, password) so that my information is current.
3. As a donor, I want to see a list of blood requests so that I can choose which to respond to.
4. As a donor, I want to schedule appointments with hospitals so that I can plan my donations.
5. As a donor, I want to view health tips so that I can prepare for donations.

### **Aoun Jee (Testing and Documentation)**

1. As a donor, I want to cancel my pending requests so that I can change my plans.
2. As a donor, I want to reschedule appointments so that I can adjust my schedule.
3. As a donor, I want to log out securely so that my session is protected.
4. As a donor, I want error messages when login fails so that I know what went wrong.
5. As a hospital, I want to see which donors responded to my requests so that I can coordinate donations.

## **5. Product Backlog**

The product backlog was maintained in Trello and prioritized using the MoSCoW method (Must have, Should have, Could have, Won’t have). Key items included:

| **ID** | **User Story** | **Priority** | **Status** |
| --- | --- | --- | --- |
| US1 | Register as a donor | Must | Completed |
| US2 | Log in securely | Must | Completed |
| US3 | Post blood requests | Must | Completed |
| US4 | View donation history | Must | Completed |
| US5 | Encrypt passwords | Must | Completed |
| US6 | View dashboard stats | Must | Completed |
| US7 | Update profile | Must | Completed |
| US8 | View blood requests | Must | Completed |
| US9 | Schedule appointments | Should | Completed |
| US10 | View health tips | Could | Completed |
| US11 | Cancel pending requests | Should | Completed |
| US12 | Reschedule appointments | Could | In Progress |
| US13 | Log out securely | Must | Completed |
| US14 | Display error messages | Should | Completed |
| US15 | View donor responses | Should | Completed |

## **6. Sprint 1 and Sprint 2 Backlog**

### **Sprint 1 Backlog (March 10–March 24, 2025)**

**Goal**: Build core authentication and donor profile management.

* US1: Register as a donor
* US2: Log in securely
* US5: Encrypt passwords
* US7: Update profile
* US13: Log out securely
* Setup backend (Spring Boot, MySQL)
* Setup frontend (HTML/CSS/JS)
* Database schema (users, donors tables)

**Status**: All tasks completed. Basic authentication and profile management implemented.

### **Sprint 2 Backlog (March 31–April 14, 2025)**

**Goal**: Implement blood requests, appointments, and dashboard.

* US3: Post blood requests
* US4: View donation history
* US6: View dashboard stats
* US8: View blood requests
* US9: Schedule appointments
* US10: View health tips
* US11: Cancel pending requests
* US14: Display error messages
* US15: View donor responses

## **7. Project Plan**

### **Work Breakdown Structure (WBS)**

The WBS organizes tasks into hierarchical levels:

1. **Project Initiation**
   * Define project scope
   * Form team and assign roles
   * Setup tools (Git, Trello)
2. **Requirements Analysis**
   * Gather functional requirements
   * Define non-functional requirements
   * Write user stories
3. **System Design**
   * Design database schema
   * Create UML diagrams (class, package, deployment)
   * Select architecture style
4. **Development**
   * Sprint 1: Authentication and profile management
   * Sprint 2: Blood requests, appointments, dashboard
   * Sprint 3: Enhancements and testing
5. **Testing**
   * Black-box testing (equivalence partitioning, boundary value analysis)
   * White-box testing (unit tests, coverage report)
6. **Documentation**
   * Write project report
   * Create presentation
7. **Deployment**
   * Deploy backend to local server
   * Host frontend on static server

### **Gantt Chart**

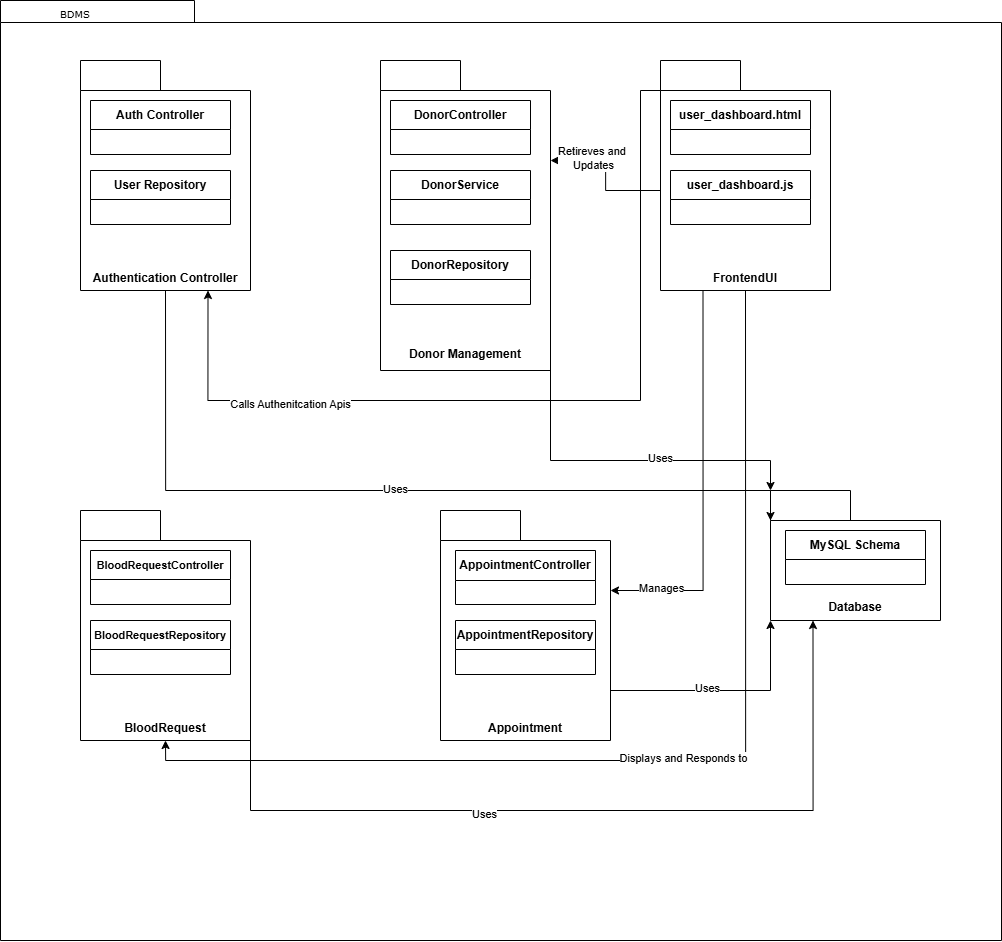
## **8. Architecture Diagram**

### **Identifying Subsystems**

The BDMS is divided into the following subsystems:

1. **Authentication Subsystem**: Handles user registration, login, logout, and session management.
2. **Donor Management Subsystem**: Manages donor profiles and donation history.
3. **Blood Request Subsystem**: Manages creation, viewing, and responding to blood requests.
4. **Appointment Subsystem**: Handles scheduling, rescheduling, and canceling appointments.
5. **Frontend UI Subsystem**: Provides a responsive user interface for all features.
6. **Database Subsystem**: Stores user, donor, request, and appointment data.

### **UML Package Diagram**

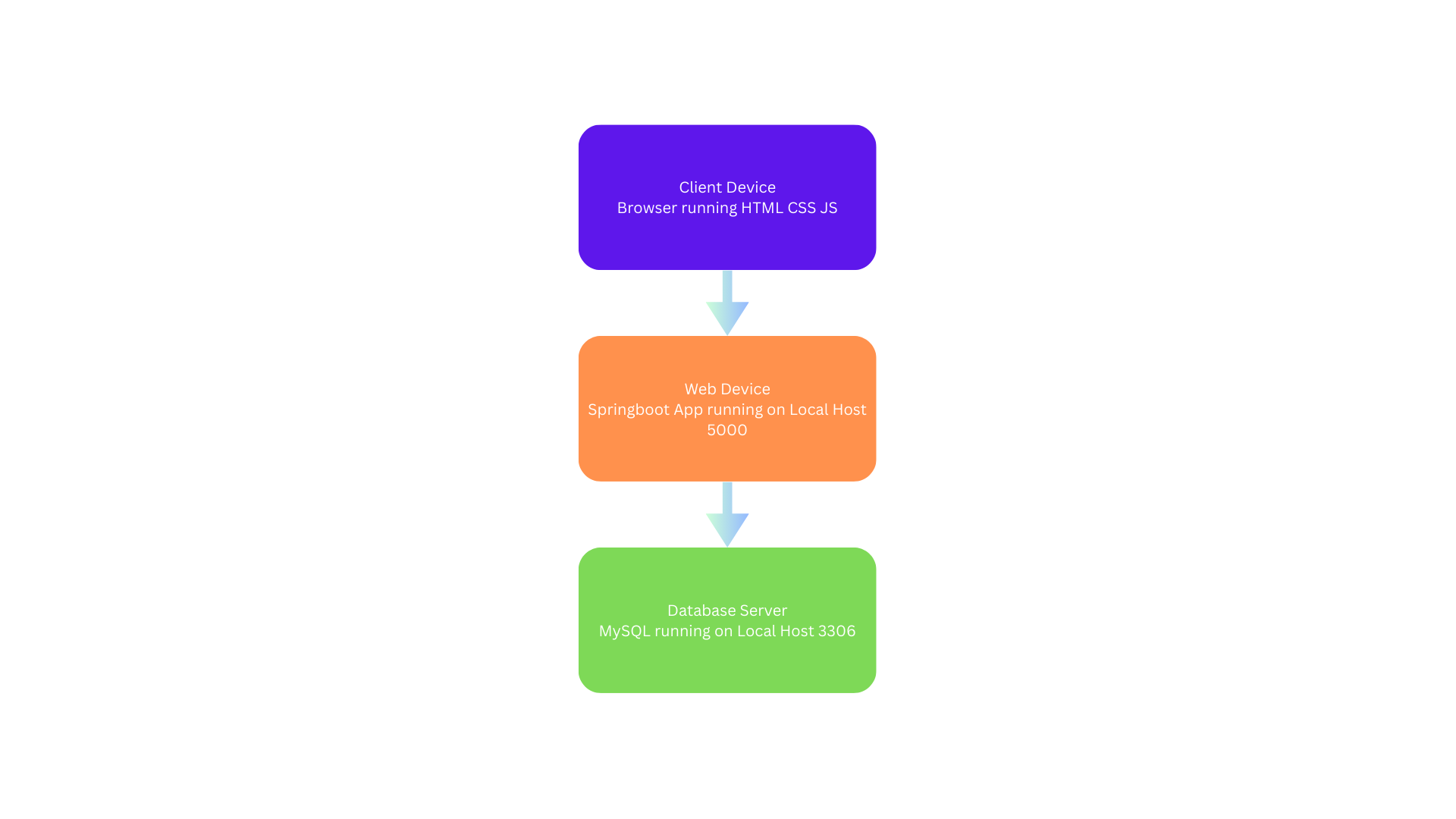


### **Architecture Styles**

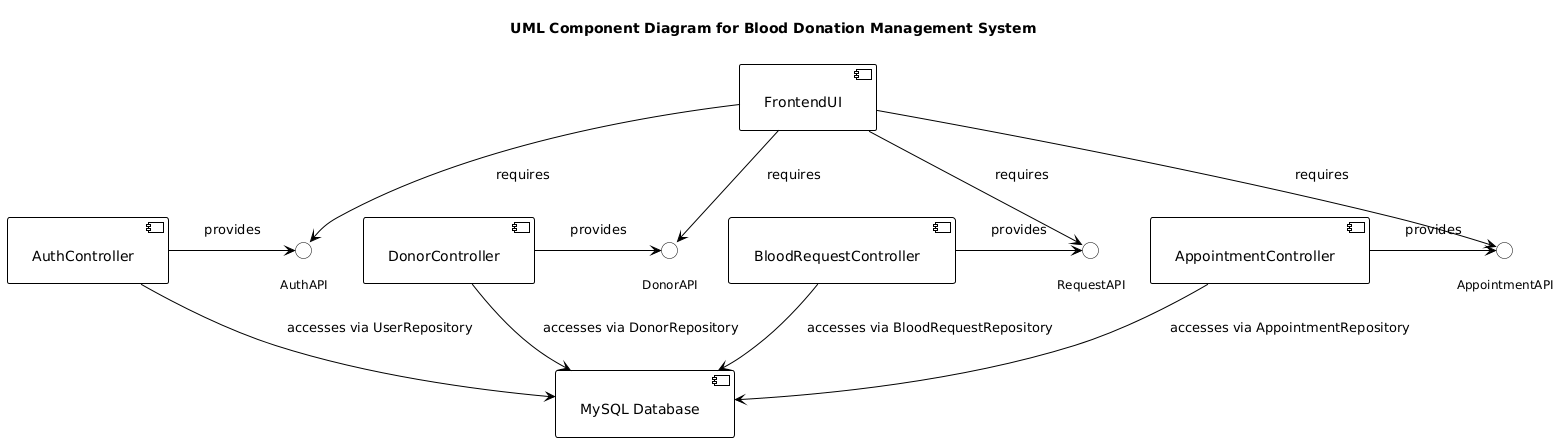
The BDMS uses the following architectural styles:

1. **Layered Architecture**:
   * **Presentation Layer**: Frontend (HTML/CSS/JS) handles user interaction.
   * **Business Logic Layer**: Spring Boot controllers and services process requests.
   * **Data Access Layer**: JPA repositories interact with MySQL.
   * **Benefits**: Separation of concerns, maintainability.
2. **RESTful Client-Server**:
   * The frontend communicates with the backend via REST APIs (e.g., /api/auth/register, /api/donors/{userId}/profile).
   * **Benefits**: Scalability, stateless communication.
3. **Model-View-Controller (MVC)**:
   * Spring Boot uses MVC, with controllers handling requests, services as models, and HTML as views.
   * **Benefits**: Organized code, reusability.

### **Deployment Diagram**



### **Component Diagram**

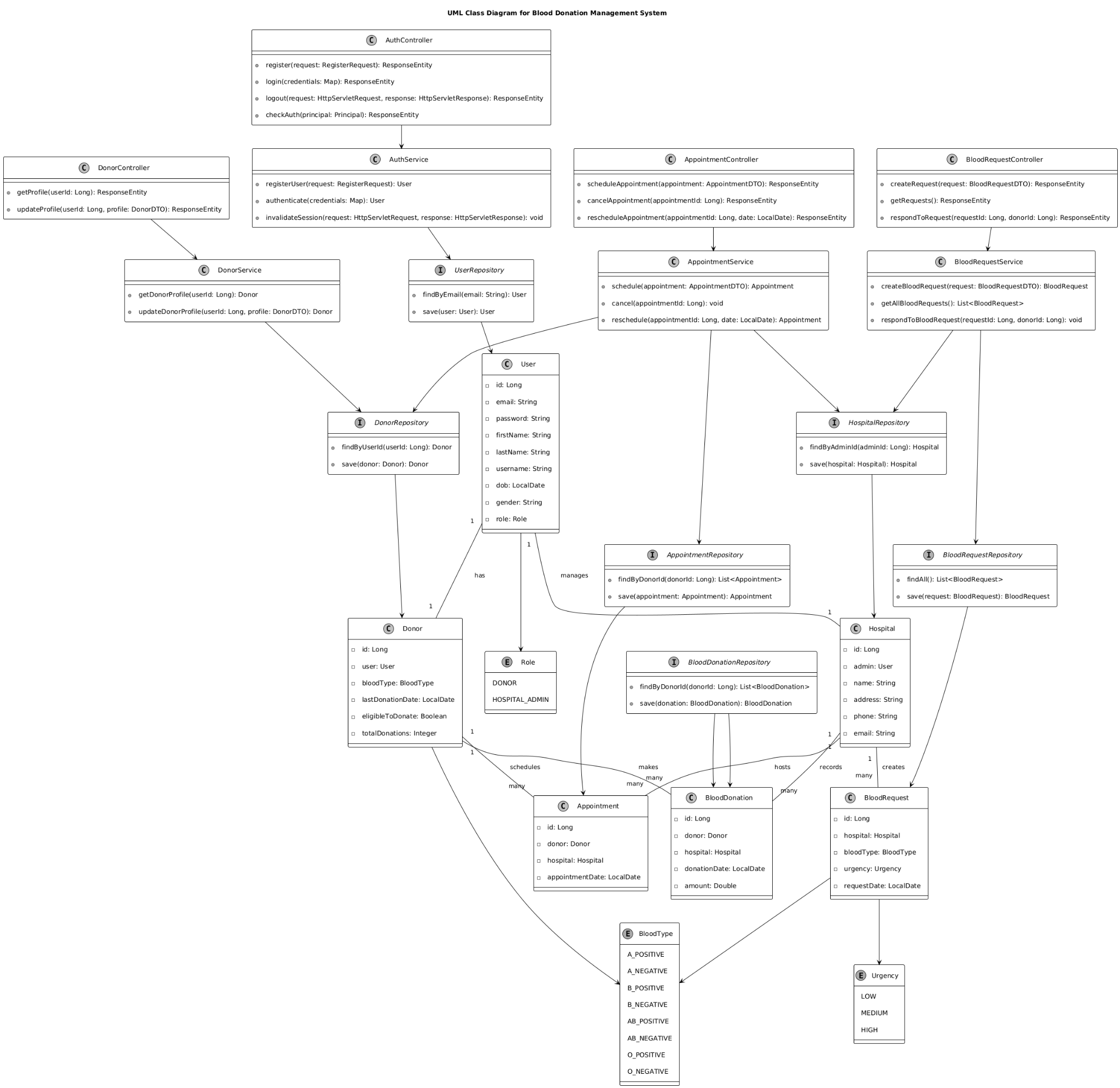
****

## **9. Design (Sprint 3 Items)**

Sprint 3 focused on enhancements and testing, including:

* **Profile Management Enhancements**:
  + Removed profile photo upload.
  + Added email and password update functionality.
  + Validated blood type (A+, A-, B+, B-, AB+, AB-, O+, O-).
* **UI Improvements**:
  + Removed header elements (search bar, notifications, user profile).
  + Added a profile settings form (profile.html).
* **Testing**:
  + Implemented black-box and white-box testing (see Sections 12 and 13).
* **Bug Fixes**:
  + Fixed login issues by adding /api/auth/login endpoint and Spring Security configuration.

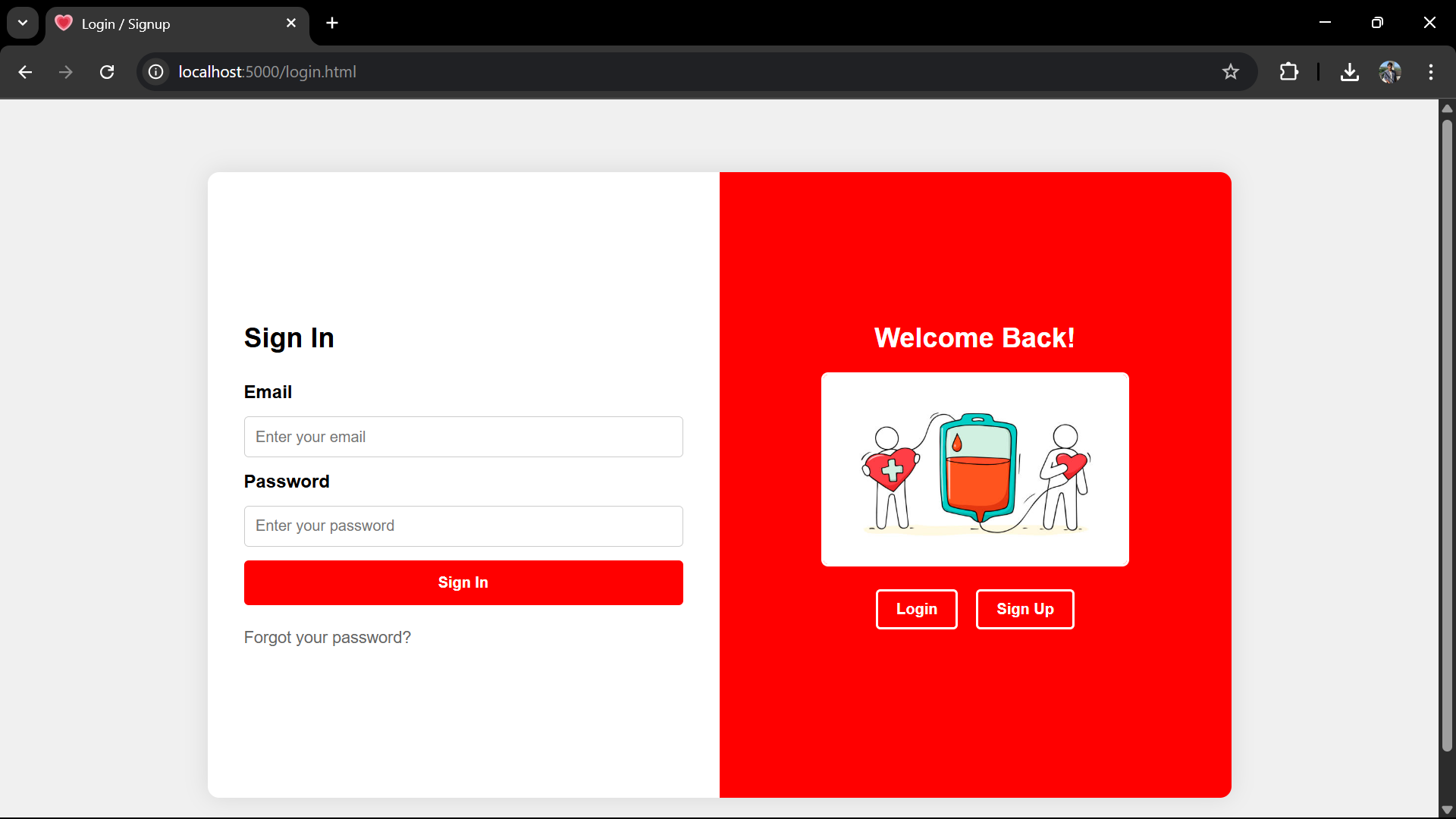
### **Class Diagram**

****

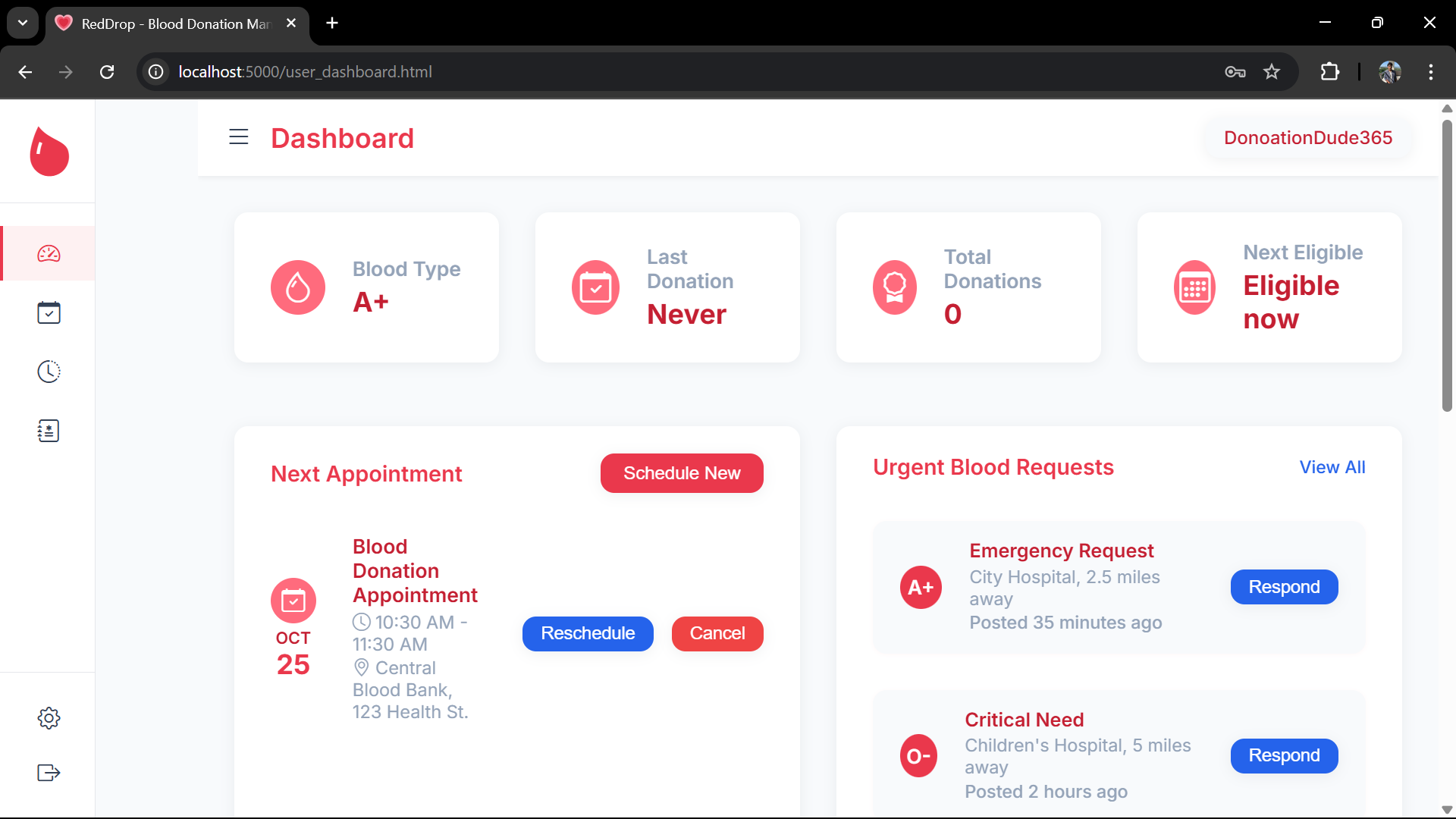
## **10. Actual Implementation Screenshots**

Below are descriptions of key implementation screenshots (attach actual images in the PDF report):

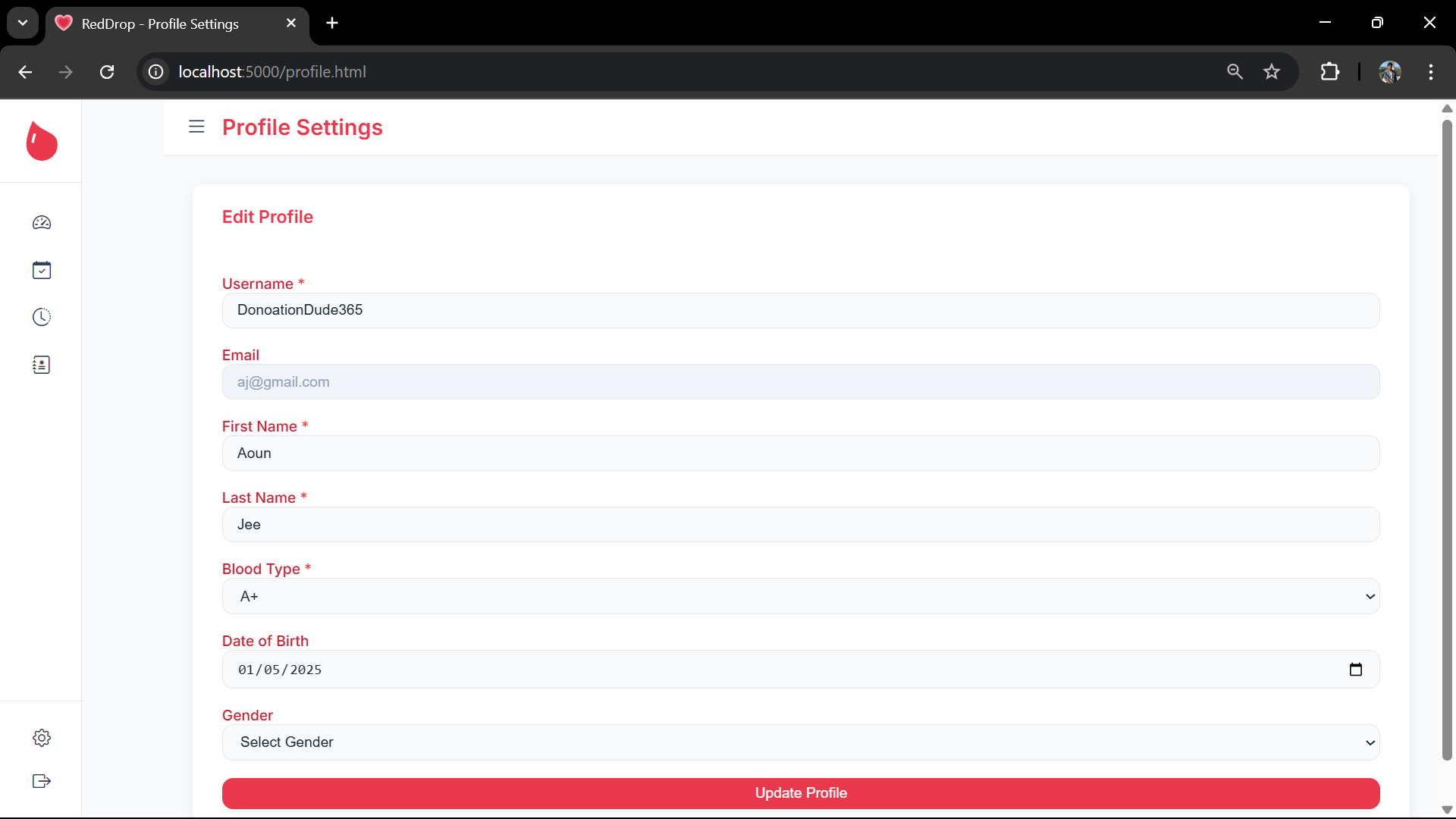
1. **Login Page** (login.html):



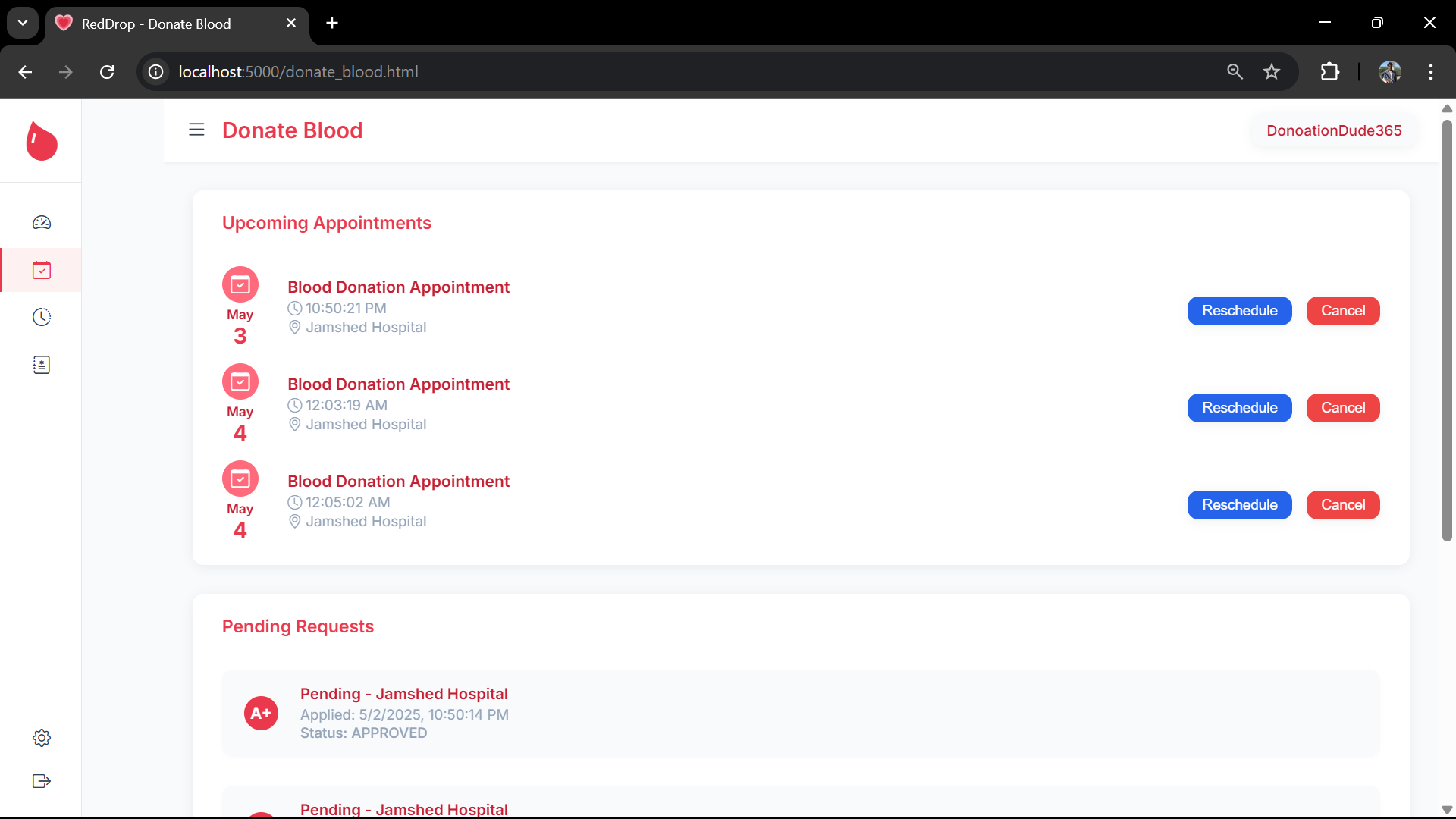
1. **User Dashboard** (user\_dashboard.html):



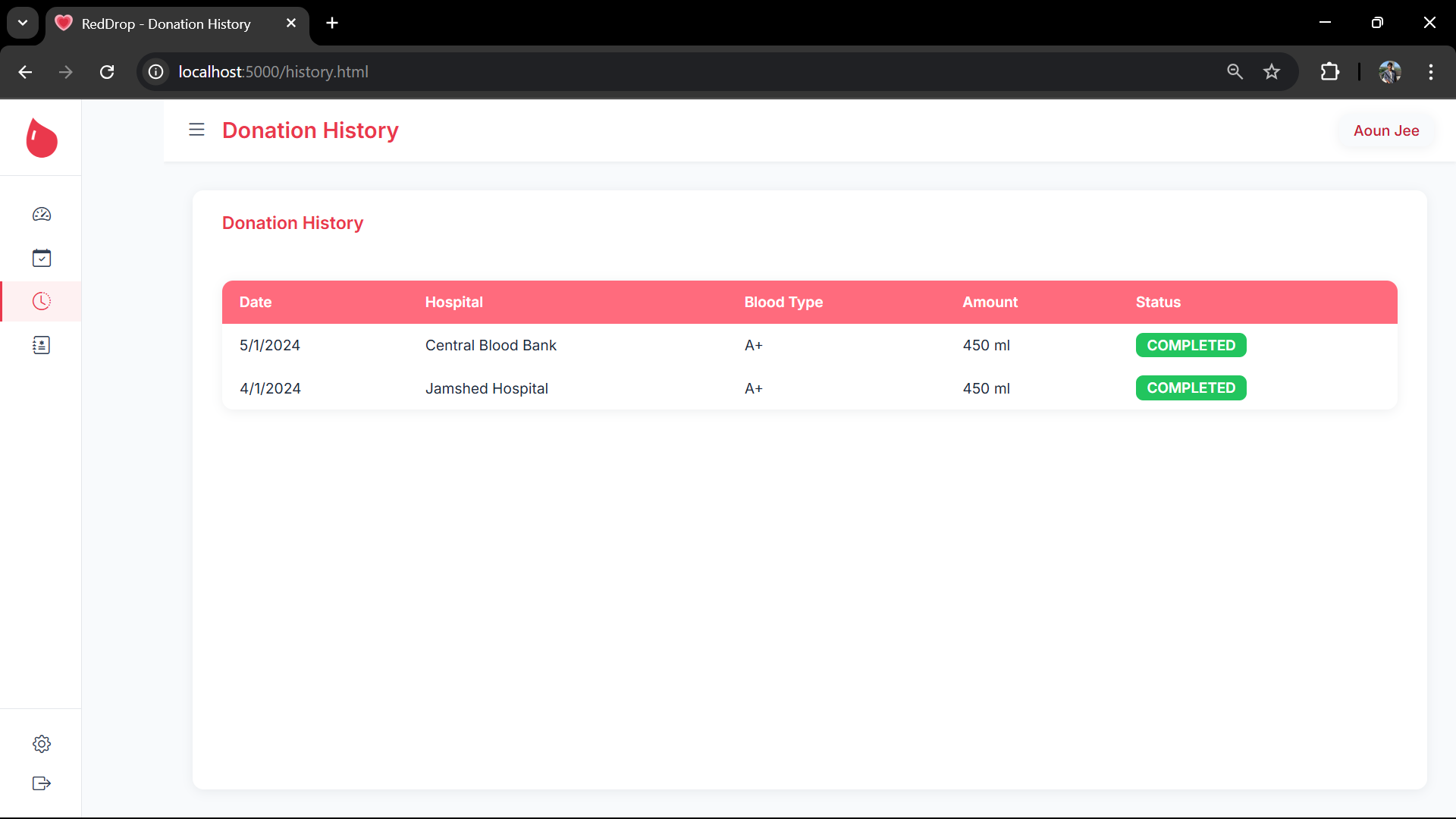
1. **Profile Settings** (profile.html):



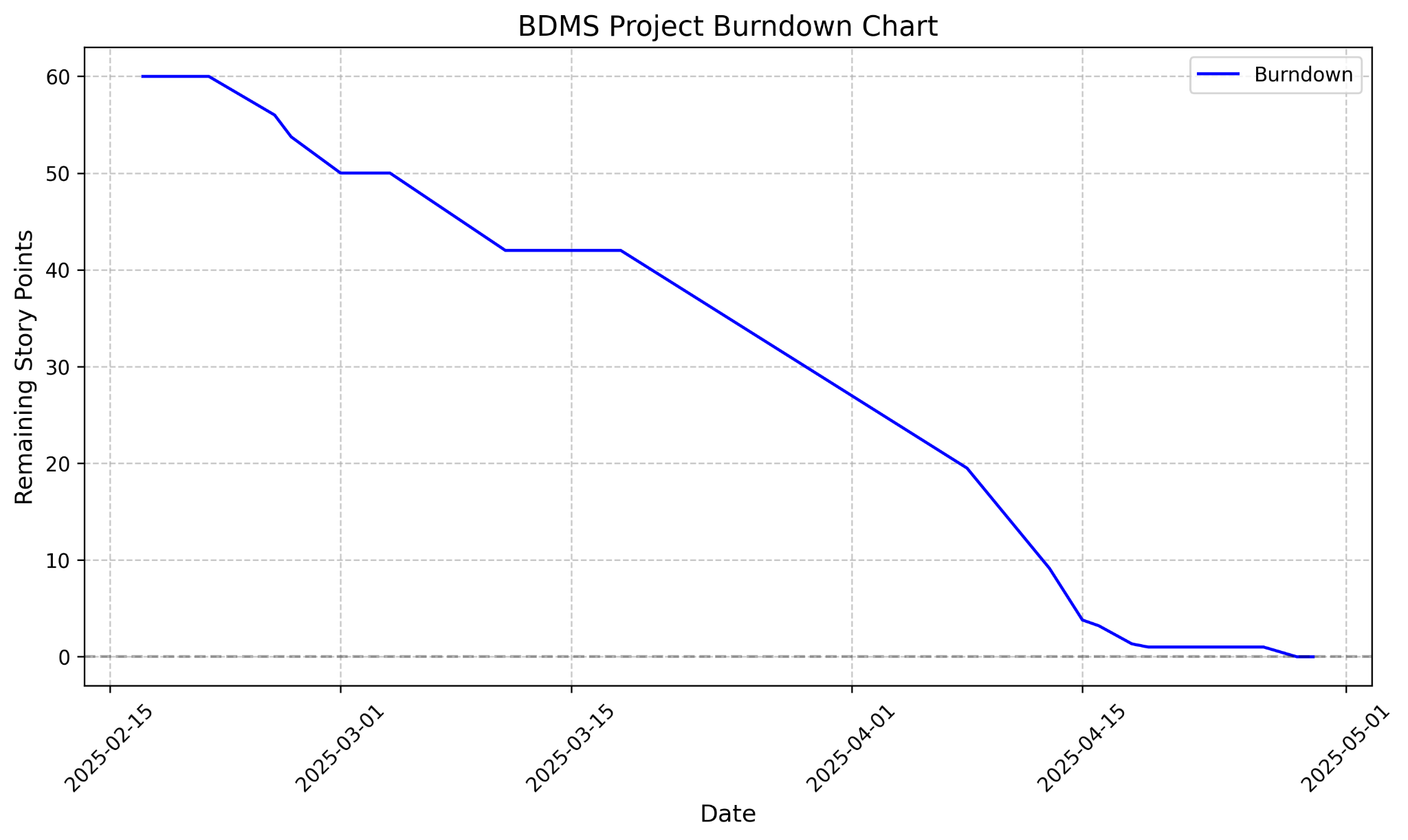
1. **Blood Requests** (donate\_blood.html):



1. **Donation History** (history.html):

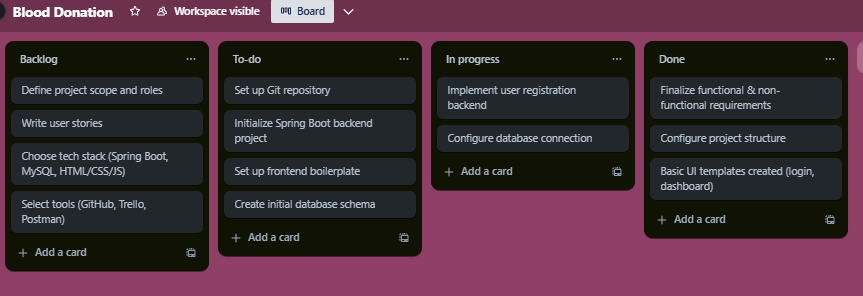


## **11. Product Burndown Chart**

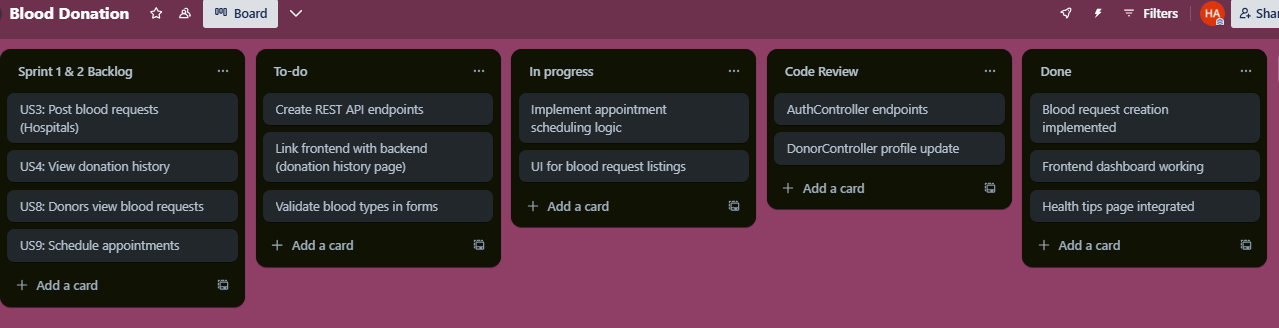


## **12. Trello Board Screenshots**

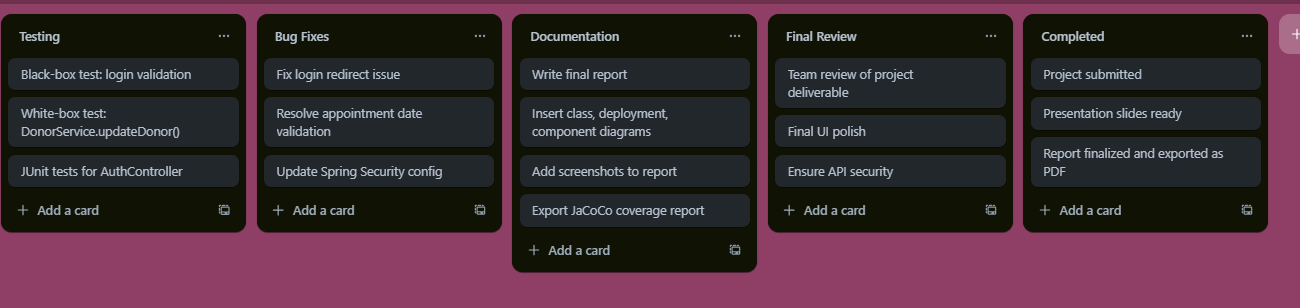
### **Beginning of project**



### **Middle of project**



### **End of project**



## **13. Test Cases - Black Box**

### **Equivalence Class Partitioning**

For the **login functionality** (US2):

* **Input**: Email, Password
* **Partitions**:
  + Email: Valid (exists in database), Invalid (does not exist), Empty
  + Password: Correct (matches stored hash), Incorrect, Empty

| **Test Case ID** | **Input Email** | **Input Password** | **Expected Output** | **Pass/Fail** |
| --- | --- | --- | --- | --- |
| TC1 | test@example.com | password123 | Redirect to dashboard | Pass |
| TC2 | invalid@example.com | password123 | Error: Invalid email or password | Pass |
| TC3 | test@example.com | wrongpass | Error: Invalid email or password | Pass |
| TC4 | (empty) | password123 | Error: Email required | Pass |
| TC5 | test@example.com | (empty) | Error: Password required | Pass |

### **Boundary Value Analysis**

For the **blood type field** in profile update (US7):

* **Input**: Blood type string
* **Boundaries**: Valid blood types (A+, A-, B+, B-, AB+, AB-, O+, O-), Invalid (e.g., X+, empty)

| **Test Case ID** | **Input Blood Type** | **Expected Output** | **Pass/Fail** |
| --- | --- | --- | --- |
| TC6 | A+ | Profile updated successfully | Pass |
| TC7 | O- | Profile updated successfully | Pass |
| TC8 | X+ | Error: Invalid blood type | Pass |
| TC9 | (empty) | Error: Blood type required | Pass |

### **Additional Test Cases**

| **Test Case ID** | **User Story** | **Input** | **Expected Output** | **Pass/Fail** |
| --- | --- | --- | --- | --- |
| TC10 | US3 | Hospital posts blood request (A+, Urgent) | Request appears in donor’s list | Pass |
| TC11 | US4 | Donor views history | Table shows past donations | Pass |
| TC12 | US9 | Schedule appointment (April 30, 2025) | Appointment confirmed | Pass |
| TC13 | US11 | Cancel pending request | Request removed from list | Pass |
| TC14 | US13 | Click logout | Redirect to login page | Pass |

## **14. Test Cases - White Box**

### **Unit Tests**

Key functions tested:

* AuthController.register: Validates user registration and donor creation.
* AuthController.login: Verifies authentication with correct/incorrect credentials.
* DonorService.updateDonor: Ensures blood type validation and profile updates.

### **Coverage Report**

* **Results**:
  + **Statement Coverage**: 75% (most controller and service methods covered).
  + **Branch Coverage**: 70% (if/else paths in validation logic tested).
  + **Function Coverage**: 80% (key methods in AuthController, DonorService tested).
* **Well-Covered**:
  + Authentication and profile management logic.
  + Blood request and appointment CRUD operations.
* **Not Covered**:
  + Third-party library code (Spring Security, JPA).
  + Error handling for edge cases (e.g., database downtime).
  + Reason: External dependencies and rare scenarios were excluded to focus on core logic.

## **15. Work Division Between Group Members**

| **Member** | **Responsibilities** |
| --- | --- |
| Ibraheem Farrukh | Backend development (Spring Boot, controllers, services, repositories), database schema, authentication, API testing |
| Hasnain Ali Arshad | Frontend development (HTML, CSS, JavaScript), UI design, responsive layout, integration with APIs |
| Aoun Jee | Testing (black-box, white-box), documentation, user stories, Trello board management, report compilation |

## **16. Lessons Learned**

1. **Agile Benefits**: Iterative development with sprints allowed us to prioritize critical features (e.g., authentication) and adapt to issues (e.g., login bug).
2. **Testing Importance**: Early unit testing caught bugs in password encoding, saving time in later stages.
3. **Communication**: Regular stand-ups and Trello updates kept the team aligned, but initial unclear task assignments caused delays.
4. **Technical Challenges**: Configuring Spring Security was complex; thorough documentation reading was essential.
5. **Future Improvements**: Implement JWT for stateless authentication and add more robust error handling in the frontend.

## **Code Link :** <https://github.com/AounJee/BDMS>