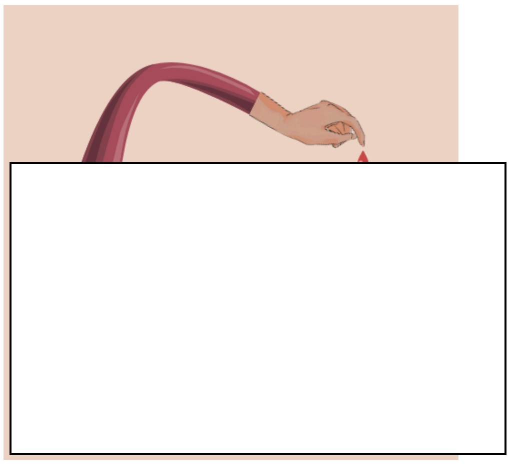
PROJECT TEMPLATE



<Plasma>

Version 0.4
Status: Draft

Date	Version	Description	Authors
	0.4	Initial Draft for Project Plasma V4	Abdulmalik Naghi, Anas Al-Mehmadi, Mohammed Al-Ghamdi, Omar Madani

Phase 1: Project Description

1.1 Introduction

Currently, the blood donation industry is suffering from two main problems: wastage of voluntary donations and inefficient communication for replacement donations. As a result, there exists a lack of efficient and effective blood supply especially in emergency situations and with rarer blood types. Consequently, messages get lost in the never-ending networks of social media, lives are sometimes lost, hospitals suffer from blood bank instability, families undergo immense stress, and ambulance authorities as well as hospitals' resources get occupied for longer times.

Having studied these problems, we concluded that this is an actual problem that needs to be addressed and solved, however, to further back our concerns we have looked at current statistics regarding the matter. A study made in 2020 in Dammam which housed 3300 people in total recorded that deferrals of blood donations was 11.7%, mostly due to non-viral factors. Furthermore, another study made in 2020 by MOH found that out of 375000 donations, 8.7% was unaccepted due to them containing viral infections, most of which are sexual disease. With regards to wastages - especially in platelets - wastages ranged from 12% to up to 95% were found by a study conducted in 2022.

All of the aforementioned studies have sealed and confirmed our concerns leading to us taking a final decision to tackle the problem, especially since the latest measure of blood donors in Saudi Arabia done in 2024 reported that only 2.3% of the whole population have donated blood within the year. In the following, we will analyse the problem further, and build up a solution to it.

1.2 Project description and objectives

Although we don't want to put a huge focus on the solution we are presenting rather than analysing the problem, it is integral that we give a brief description about how we will tackle this dilemma. Project Plasma is a platform that is envisioned to be integrated with a well know application such as Sehaty. This step will make the solution easy to reach and know about, as well as have a great potential in terms of functionality given that it will be connected which MOH systems. Both aforementioned points are lacking in today's alternatives which we will discuss later in section 1.5.

The platform will benefit hospitals and donors as end-users, with the ability of cross-hospital communication for better inventory management. It will allow for replacement-based donations as well as voluntary donations, however, both types will be managed depending on the demand to further reduce wastage. These are the main crutches of the platform requirements, however, further details will be discussed in section 1.6.

Project Plasma's Objectives:

- Raising the percentage of overall donors by 0.1% annually.
- Reduce deferral rates to below 5% and maintain it there.
- Reduction of wastages overall and especially wastages to below 10% of overall donations.

It is important to further elaborate and say that these numbers are mainly based on sensible approximations made on our end. The rates expressed in our objectives may be subject to change upon application and recording of new statistics based on information gathered from the platform.

1.3 Project team

Since all project members are studying the principles of software engineering while concurrently conducting this project, task designation for each member wasn't finalized yet, however, we are able to mention the full task designation until V4. So here is the team behind Plasma:

- Abdulmalik Naghi: Project Founder, Supervisor, Contributor.

  Tasks: Problem Description, Objectives, Goals, Stakeholders, FR, DGT, System Comparison, SD, SAD.
- Anas Al-Mehmadi: Project Contributor.

  Tasks: Sources of Domain Analysis, Risk Analysis, UC Descriptions, CD, SD.
- Mohammed Al-Ghamdi & Omar Madani: Project Contributors.

  Tasks: Project Scope and Features, NFR, UC Design, CD, AD, Testing.

Goals and Scope

1.4 Project Goals

- Reduce overall wastage and utilize voluntary donations optimally.
- Decrease searching and matching times for eligible replacement donors.
- Reduce overall time of aid in emergency situations by introducing better pre-screening indicators.
- Decrease overall wastage by introducing cross-communication between hospitals.

1.5 Scope and Sources of Domain Analysis Information

To explore the problem, we gathered information that helped us develop a deeper understanding of the challenges, current practices and improvement opportunities. The sources we explored are existing applications, medical literature, regulations and stakeholders.

In existing applications we observed The Blood Donor App, Ehsan and Wateen. Blood Donor App which is backed by the American Red Cross, is mostly for donors, as it helps donors book blood donation appointments and find local blood centers.

As to local applications, both Ehsan and Wateen focus on voluntary donations whilst the need for replacement donations is neglected. Furthermore, Wateen's approach does not limit donors from donating even if a certain hospital has enough in its inventory which is a huge contributor to wastage. Whilst Ehsan does show requests based on specific blood inventory needs, it still lacks a full suite of solutions to counter the current dilemma; a characteristic it shares with Wateen.

To elaborate on the previous point, both applications lack pre-screening indicators at least from medical records, cross hospital communication, and real time emergency notifications. All of the aforementioned consequences lead to poor management of inventory and donors, leading to higher wastage and slower response to emergencies.

Regarding medical literature, we researched the preconditions related to the donor, including age, health condition and blood type. This information will help build an optimized system with proper filtration. Furthermore, we viewed the local statistical studies regarding wastage and deferral rates in the past years.

This system works with personal health, which is regarded as sensitive data by the Saudi Personal Law. This means that our system must adhere to the regulations of the law in collecting, storing, analyzing and transferring of sensitive data.

Stakeholders include recipients, donors and healthcare providers. Recipients want a fast and reliable method of finding a donor. Donors want convenience and ease of use. Healthcare providers require eligibility checking, adherence to medical standards and blood bank stability.

Full stakeholders are as follows:
Directly involved stakeholders: Recipients, Donors, Hospitals, Tech Provider.
Indirectly involved stakeholders: Community, Ambulance Authorities.

According to what we mentioned above, here is the scope of Plasma:

1.6.1 Included

Blood Type Matching: To quickly identify the right donor for a patient and save time.

Location-Based Matching: Helps locate the nearest donor or hospital, reducing search time.

In-App Map: Makes it easier for donors to navigate to the patients, especially in cases where applications like Google Maps are suffering.

Notifications & Messaging: Ensures requests reach the right people at the right time.

Verified Donor Profiles: Builds trust and safety when selecting donors especially in high emergency situations.

Hospital-to-Hospital Communication: Allows hospitals to share blood supplies and reduce waste from expired units.

Donation History: Helps both donors and hospitals track donations and plan better.

Achievement-Based Badges: Motivates donors to donate regularly.

1.6.2 Excluded

Standalone Mobile Application: Our system will be integrated into the existing Sehhaty application instead of creating a separate app.

Physical Transportation of Blood: The project does not cover the delivery of blood, this remains the responsibility of hospitals.

Full Integration with Patient Health Records: The system will only utilize essential donor and patient information necessary for matching, without accessing complete health records.

Emergency Prioritization: While the placement of this factor may be subject to change in the near future depending on further studying of whether it is achievable in a timely manner or not, we mainly placed it here due to the fact that it could hinder the ease of integration since many hospitals may be against the notion of an app controlling any flow of work happening in their bounds.

Phase 2: Business Requirements Specifications

2.1 System Stakeholders and Actors

Firstly, let's define a stakeholder. A stakeholder is any party that is directly or indirectly affected by the system. Furthermore, a stakeholder may be a party who has a direct interest in the outcomes of the system, or might be a beneficiary without even interacting with it. Since Plasma is an aid to many entities, here is a list of all parties that Plasma affects whether directly or indirectly:

1- Government: Plasma aids in reducing main negative rates, which affects the government's reputation.

2- Ambulance authorities: Plasma allows for a widespread availability of blood in several locations which can make ambulance operations much more efficient and exclude the possibility of the need to relocate the patient to a different hospital.

3- Community: As Plasma raises the positive factors and lowers the negative ones, the community may feel more safe and comfortable rather than anxious in cases of emergencies.

4- Hospitals & their staff: Better blood management results in less hospital resources being occupied and much more efficient operations flow.

5- Patients & their families: One of the main direct beneficiaries from Plasma as they are the main entity that is negatively effected in emergencies due to the current rates of shortages and wastages. Their families also suffer from immense stress which shall not exist anymore given that Plasma is a beacon of hope to connect them correctly with a donor.

6- Donors: Often times people want to donate blood but get put off by not knowing how to effectively donate blood in a manner that ensures their donation will be useful. With Plasma, donors can easily know donation opportunities.

7- Software Engineers: Those are the main builders of the system. They might maintain it for a long period of time, or even benefit from it financially.

Next is actors. Actors are those that directly interact with the system. In some cases Actors may be referred to or understood as end-users, however, they can be a piece of software or hardware. Regardless, the best characteristic of actors that make them stand out is the fact that they are the ones that trigger use cases. Here are Plasma's Actors:

1- **Hospital Coordination** (blood bank staff, EMR staff)
2- **Donor** (voluntary or replacement)
3- **Hospital Admin** (approves staff accounts, manages inventory visibility)
4- **System Integration Service** (Sehaty + MOH auth, notification gateway, maps fallback)
5- **Platform Admin** (trust & safety/audit)

2.2 Domain Analysis

As mentioned previously in section 1.5, the main sources of analysis with regards to existing systems were Wateen and Ehsan since they are the main and only locally available approaches to this dilemma. The Red Crescent app (USA based) was visited though it was to mainly build a background and understanding to different aspects of the platform such as general legislations as well as general features.

To begin with, Wateen is the main platform that is dedicated to blood donations. The main issue of Wateen is that it completely disregards the service of replacement donations, which is the core focus of Plasma. Furthermore, even though Wateen solely focuses on voluntary donations, it doesn't take wastage-reducing measures such as allowing users to donate to a certain hospital or centre only depending on the inventory need, which might make room for overspilling and increasing the rates of wastages. Another approach that is unique to Plasma is cross-hospital communication to reduce wastages. It isn't logical or plausible that the blood of donor becomes somewhat owned and controlled only by one entity. This has an opposite effect of the purpose of blood donation in general which is providing aid to people as a whole. With the lack of cross-hospital communication, the capacity of people that the donated blood may provide aid in becomes much smaller and specific only to that hospital's patients. To counteract that, we have introduced a feature of enabling hospitals and coordinate blood-bank needs to further improve blood availability as well as reduce the rate of wastages. Finally, the most pronounced issue we found in Wateen has nothing to do with its system design or functionalities. The most problematic part of Wateen is that it is a dedicated application which decreases its popularity due to the general public not knowing about it, and therefore putting an extra burden towards making it easy for people to know how or where to donate.

Aside from Wateen, Ehsan is the remaining available platform that offers blood donation opportunities. Ehsan shares with Wateen the factor of difficulty of reach since it is a standalone application. Furthermore, Ehsan isn't widely known as a platform for blood donation rather than charitable opportunities. Another factor Ehsan shares with Wateen is that it is only for voluntary donations, neglecting emergency situations. To add to that, although Ehsan does show voluntary donation needs across hospitals depending on the inventory needs (limited amount), the reporting of hospitals that are in need isn't always in real-time. The author of this project once received a message on Whatsapp of a patient in need in one of the hospitals in Jeddah, yet when he checked the reported shortages on Ehsan, the hospital that was mentioned in the message was not shown to be having a shortage in its blood-bank.

For an in depth comparison between Plasma and the two aforementioned platforms, kindly check page **11**. Page **11** will include comparisons based on **the allowed type of donations, differences in mechanisms of the donations and its basis, difference in the end-users of communication and measures taken to counteract wastages and deferrals.**

2.3.1 Functional Requirements

Before diving into the requirements, we will display a structured definition and introduction of the platform through which we will define the functional requirements:

The proposed system, Plasma, is a blood donation coordination platform designed to integrate seamlessly with Sehaty and the Ministry of Health infrastructure in Saudi Arabia. Plasma shall act as a bridge between hospitals, donors, ensuring that blood components are matched efficiently, wastage is minimized, and donor participation is encouraged.

The system shall authenticate both hospitals and donors through MOH/Sehaty accounts, enforcing role-based access to ensure that requests and donor data remain secure and appropriately visible. Hospitals shall be able to create and manage blood component requests, which the system shall validate for medical accuracy and track through well-defined lifecycle statuses. Plasma shall maintain structured donor profiles, capturing only minimal eligibility metadata, and shall automatically manage donation cooldowns to prevent unsafe practices.
When a hospital posts a request, the system shall perform intelligent donor matching based on compatibility, proximity, urgency, and eligibility criteria. Donors shall be notified in structured batches and can accept or decline requests, with confirmations and QR-based check-ins ensuring transparency and accountability. Plasma shall further enhance efficiency by enabling hospitals to publish near-expiry blood components to a regional board, allowing other hospitals to claim and utilize them before expiry, thus reducing wastage, especially for short-lived products such as platelets.

To improve donor experience and engagement, the system shall provide in-app navigation, assign recognition badges based on donor activity, and deliver motivational or informational messages. Plasma shall also compute real-time analytics and KPIs—including responsiveness, fill times, and wastage metrics—to provide actionable insights for hospitals and policymakers. Finally, the system shall uphold strict compliance with privacy and data protection requirements by ensuring explicit donor consent, enabling data export or deletion, enforcing purpose limitation, and maintaining auditable logs for hospital actions.

In summary, Plasma shall provide a secure, efficient, and motivational blood donation ecosystem, addressing both urgent patient needs and systemic inefficiencies while aligning with national standards for healthcare data privacy and operational integrity.

Taking the above introduction into account, here is a further analysed structure of the functional requirements:

A. Identity, roles, and access

1. The system shall authenticate **hospital users** via MOH/Sehaty SSO and assign roles (Coordinator, Admin).

2. The system shall authenticate **donors** via Sehaty account and **explicit consent** for processing donor data (opt-in/out).

3. The system shall restrict **request visibility** to (a) the posting hospital, (b) matched donors, and (c) authorized cross-hospital viewers for redistribution.

B. Request creation & verification (hospital side)

1. A hospital coordinator shall be able to create a **blood component request** with: component type (RBC/Platelets/Plasma), ABO/Rh, units, location, urgency window, notes, and “replacement vs voluntary” flag.

2. The system shall **validate** requests against allowed component/blood-type schemas and capture an **expiry window** (esp. for platelets).

3. The system shall support **request status** transitions: Draft > Active > Paused > Filled > Cancelled > Expired.

C. Donor profile & eligibility metadata

1. The system shall maintain **donor profiles** with: ABO/Rh, approximate geocell, last donation date, known deferral state (Y/N + reason + until date), contact preferences, and donation history count/badges.

2. The system shall auto-compute **donor cooldown** windows by component (e.g., whole blood vs apheresis) and suppress notifications accordingly.

D. Matching & notifications

1. The system shall compute matches using: ABO/Rh compatibility, distance, donor availability window, cooldown/deferral state.

2. The system shall **batch-notify** the top N donors first; if unfilled after T minutes, widen radius / relax thresholds to the next batch.

3. Donors shall be able to **Accept** (with ETA) or **Decline** from the notification; acceptance locks a **slot** that expires after a configurable time if no check-in occurs.

4. The system shall provide **live fill status** (units requested, units pledged) to the posting hospital in the case of inventory based requests.

E. Check-in & completion

1. The system shall issue a **QR check-in** for accepted donors; hospitals can scan to confirm arrival.

2. Upon donation completion, the hospital shall mark **request/unit(s) completed**, updating donor **last donation date** and **history count**; failed attempts can be marked with reason (e.g., deferral → sets deferral-until date).

F. Cross-hospital coordination (anti-wastage)

1. Hospital admins shall be able to **publish near-expiry component availability** (esp. platelets) to a **regional board** visible to nearby hospitals.

2. The system shall allow a receiving hospital to **claim** near-expiry units and coordinate transfer (note: **physical logistics excluded**; we only record the intent and timestamps).

3. The system shall trigger **near-expiry alerts** (e.g., platelets D-2, D-1) to the owning hospital.

G. Navigation

1. Donors who accept shall receive **in-app directions** to the hospital.

2. The system shall store only **coarse donor location** until Accept; after Accept, donor can share **precise route** for ETA.

H. Donor motivation & messaging

1. The system shall award **badges/tiers** based on lifetime donations and reliability (accepted→showed rate), visible in donor profile.

2. The system shall support **templated messages** (thank you, milestone reached, eligibility again).

I. Analytics & KPIs

1. The system shall compute per-request metrics: time-to-first-accept, time-to-fill, no-show rate, notification-to-accept rate.
2. The system shall compute anti-wastage metrics: near-expiry alerts issued, inter-hospital claims, **avoided expiry** count.
3. The system shall compute donor-pool health: active donors by type/region, gender participation, replacement vs voluntary mix (without exposing personal health details).

J. Compliance, consent, and audit

1. The system shall capture **explicit consent** from donors for (a) notifications, (b) geolocation for routing, (c) processing minimal eligibility metadata.
2. The system shall provide **export/delete my data** for donors and maintain **audit logs** for hospital actions (create/edit/close requests, view near-expiry board).

2.3.2 Non-Functional Requirements

Performance

- The system should match donor-patient requests within 4 seconds at most.

- The system should be able to support at least 10k concurrent users without a decrease in performance.

Availability

- The system should be available 99.99% of the time.

- The system should have backup servers across the country.

Reliability

- Critical operations should retry automatically if failed.

- The system should maintain redundant services to avoid single points of failure.

Security & Privacy

- Donor and patient data must be encrypted both in transit and at rest.

- The system should follow the MoH data privacy regulations.

- The system will not access patients' health records.

- The system shall implement purpose limitation (use data only for donor-hospital matching, anti-wastage coordination, and data protected analysis and research).

Usability

- The system should be easy to use and intuitive.

- The system’s interface must be bilingual (Arabic and English).

- The system should meet accessibility standards (screen reader support for visually impaired users).

Scalability

- The system should be designed to cover major cities in Saudi Arabia.

Portability

- Since it’s integrated with Sehhaty, the system should be compatible with iOS and Android devices.

- The system should be able to adapt to future platforms.

2.4 Data-Gathering Techniques

1- Brainstorming:

During the analysis phase of project Plasma, we initially focused on brainstorming around the overall needs of the problem. This was the main source of new ideas and innovation regardless of the currently available solutions to the problem. The brainstorming phase contributed to most of the project outline, though the following phases acted as refinement stages to improve some aspects of Plasma and make it stand out.

2- Document Analysis:

Following brainstorming, we analyzed existing documents (mainly statistical research articles and papers) regarding the matter to further support the need for project Plasma. Document analysis mainly highlighted the issues of wastages and deferral rates, which are a few of the main culprits that are leading to the shortage and insufficient supply dilemma.

3- Case Studies:

In the final stages of the problem analysis phase, we reviewed existing solutions and studied their shortcomings and some attributes that make them insufficient. Although we reviewed an international approach to this problem, it was mainly to improve our knowledge base and make ourselves more grounded to the expected functionalities and features of the solution. However, the main focus was on the local solutions currently available. Wateen is the main platform concerned directly with blood donation, however, we noticed it has many shortcomings that make it incomparable to our proposal. Another local contributor is Ehsan which is mainly known for charitable opportunities, though it does offer a modest functionality of blood donation. In the next page, we will provide a detailed comparison between the existing solutions and Plasma.

4- Observation and Interviews:

It is also important to mention that observation (passive) and unstructured interviews (informal) did play a consistent part throughout all the above phases. We observed the current situation and lack of effective connection between primary stakeholders. Furthermore, we did conduct informal interviews by speaking with general people as well as people who are in the medical field. Both parties expressed positive feedback to our idea and did agree that there is an actual need to such an approach.

Comparison of Local Blood Donation Solutions vs Plasma

Aspect	Wateen	Ehsan	Plasma (Proposed)
Primary focus	Voluntary donations only	Voluntary donations, not primarily known for blood	Supports both voluntary and replacement donations, addressing actual hospital needs
Inventory awareness	Does not account for hospital inventory levels = can lead to over-supply and wastage	Shows hospital blood needs, but reports are not always accurate or up-to-date	Fully integrated with Sehaty + hospital systems = real-time requests based on inventory
Pre-screening of donors	No pre-screening beyond basic eligibility	No pre-screening	Uses Sehaty-linked medical history indicators (minimal flags) to pre-screen donors and reduce deferrals

Emergency response	No real-time emergency notifications	No real-time emergency notifications	Urgent requests prioritized with donor matching & instant alerts
Cross-hospital communication	No cross-hospital coordination	No cross-hospital coordination	Enables regional boards for sharing/claiming near-expiry blood (esp. platelets)
Wastage management	Risk of high wastage (no inventory check before calling donors)	Partial reporting, but no anti-wastage features	Tracks expiry windows and enables redistribution of near-expiry units
Ease of access	Standalone app; depends on popularity = limited reach	Standalone app; primarily associated with charity, not blood	Integrated with Sehaty (MOH central platform) = higher adoption, less friction
Donor motivation	Records donations, no real incentive mechanisms	General charity recognition, not donation-specific	Donor engagement via badges, milestones, recognition
Reputation & Awareness	Widely recognized for blood, but isolated	Well-known for charity, less for blood	Will leverage Sehaty's existing wide reach = trusted and visible

2.5 Use Case Descriptions

Use case 1: Blood Request for an Individual:

- **Actors**: the hospital, the system and the donor.

- Preconditions:
1. The hospital must be registered
2. Complete health information

- The main flow:
1. The hospital logs into the system.
2. The hospital submits a request, which includes, blood type, the location and urgency.
3. The system searches for eligible donors based on the factors in the request.
4. The system matches the request with donors based on the algorithm (Best match or nearest match).
5. The system notifies *All* matched eligible donors.
6. Donors accept the request.
7. The system notifies the recipient.

- Alternative flows:

If no donor is available, the system will recommend contacting other hospitals.

- Postconditions:
The blood request is successfully matched.

Use case 2: Hospital to Hospital Request

- Actors: the hospital and the system.

- **Preconditions:**
  1- the hospital must be registered.
  2- Each hospital must have a file which shows the current blood levels.

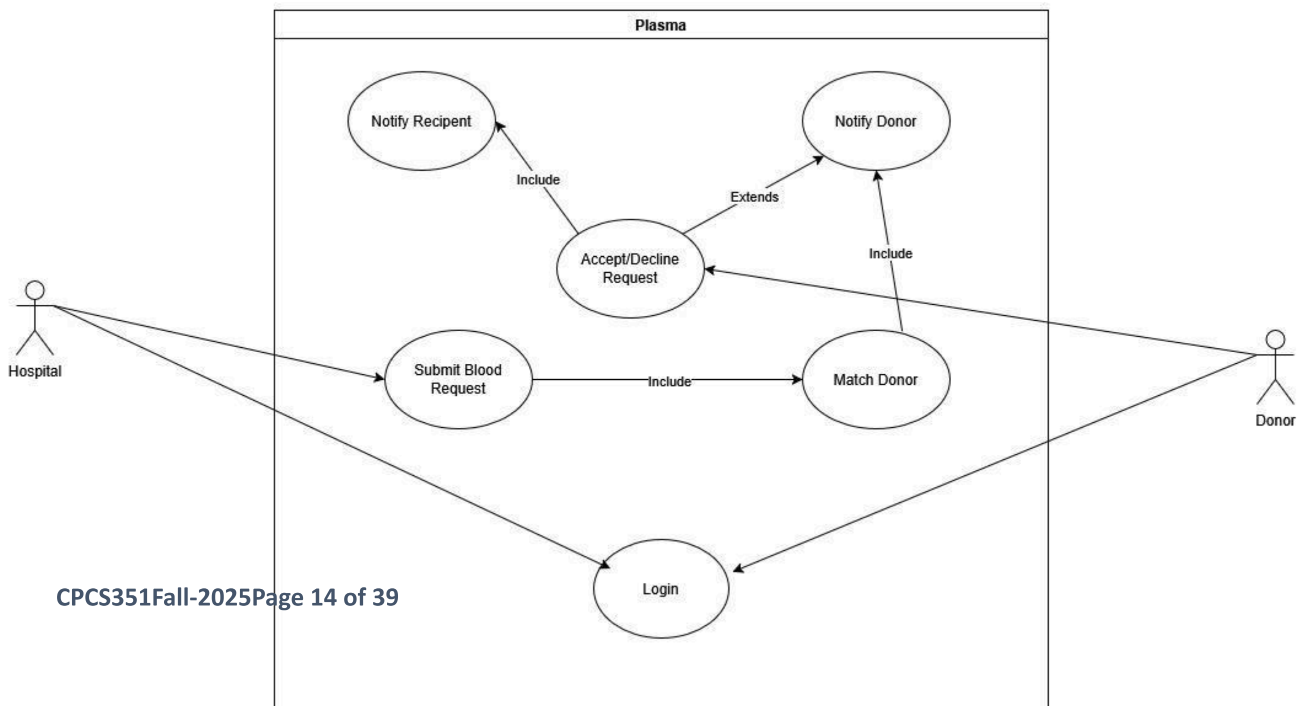
- The main flow:
1. The hospital logs into the system.
2. The hospital complete the submission form, including needed blood types and quantities.
3. The hospital submits the request.
4. The system searches for eligible hospitals based on hospitals current inventory.
5. The systems notifies selected hospitals.
6. The selected donor hospitals accept the request.
7. The system notifies the recipient hospital.

- Alternative flows:
If the donor hospitals decline the request, the system will send notifications to eligible individuals donors.

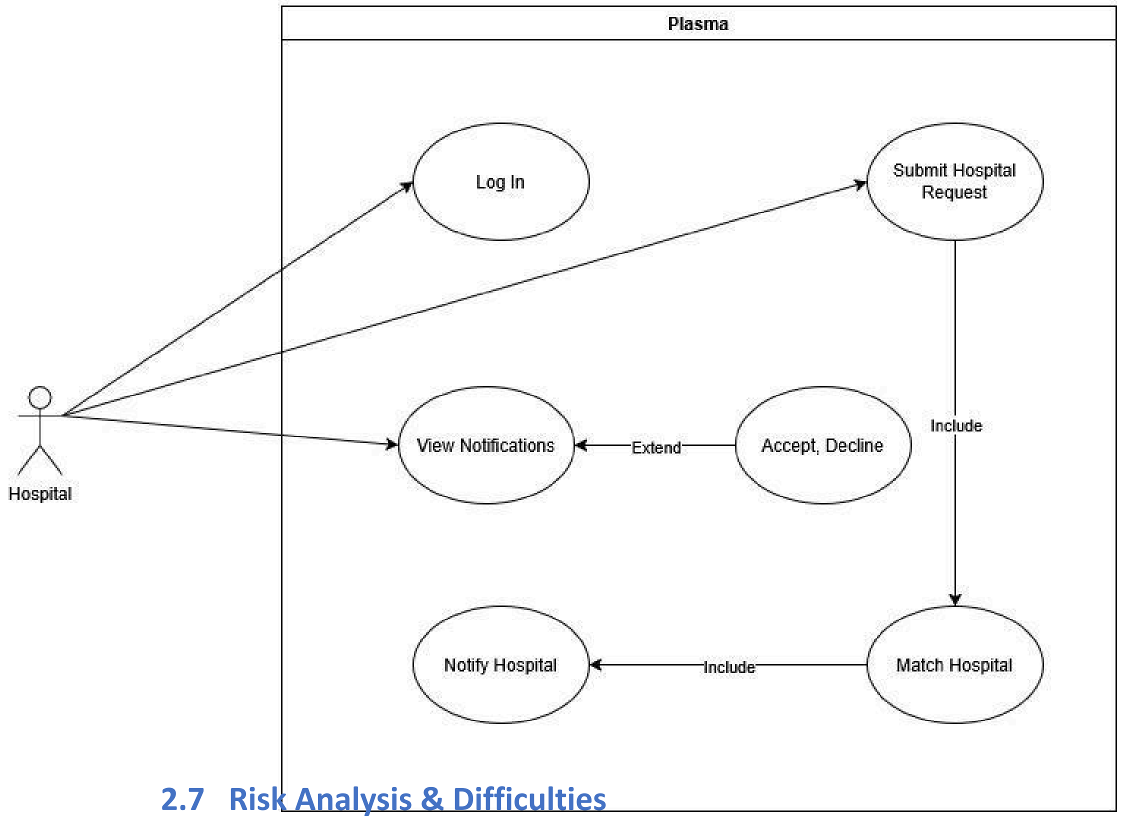
Postconditions:
- The Hospital to Hospital request is completed.

2.6 Use Case Diagrams

Use Case 1: Hospital Request Blood



Use Case 2: Hospital-To-Hospital Rrequest



2.7 Risk Analysis & Difficulties

Difficulties

1. Donor Eligibility Verification: It's possible that some donors may be unaware of medical conditions that may affect their eligibility to donate blood. This may cause wastage of time and effort, which the system aims to minimise.

2. Low Adoption Rate: The effectiveness of system relies on the community to achieve its goals, low adoption rate would threaten the feasibility of the system.

3. Notification Algorithm: Designing the logic for notifying the donors is a challenge. What are the factors that must be taken into account when notifying donors of need for donation? Would it be matching-based, the best matched blood type would be notified? or location-based, the nearest eligible donor would be notified? Deciding on those factors is a major difficulty of this system.

4. Maintaining Donors Engagement: Donors may register in the system but lose interest or forget about the system, before getting the chance to donate.

5. Communication difficulties: Expatriate make a substantial percentage of the Saudi community, which require supporting multiple languages to ensure covering this category of potential donors. This add a layer of complexity to the system design.

Risks

1. Privacy Breach: The system handles Health-related data, which are sensitive. Any breach on any level would pose a significant risks on both the users and the system.

2. Donor Availability: Even with many registered donor, there is no guarantee that an eligible donor will be available at the time of need.

3. Integration with Sehaty: Integrating with Sehaty is crucial to verify users information and to achieve the system's goals. A refusal from the related authorities would jeopardise the system's success.

4. System Shutdown: Since the systems may be used for urgent and life-threatening cases, any downtime would have severe consequences on the recipients.

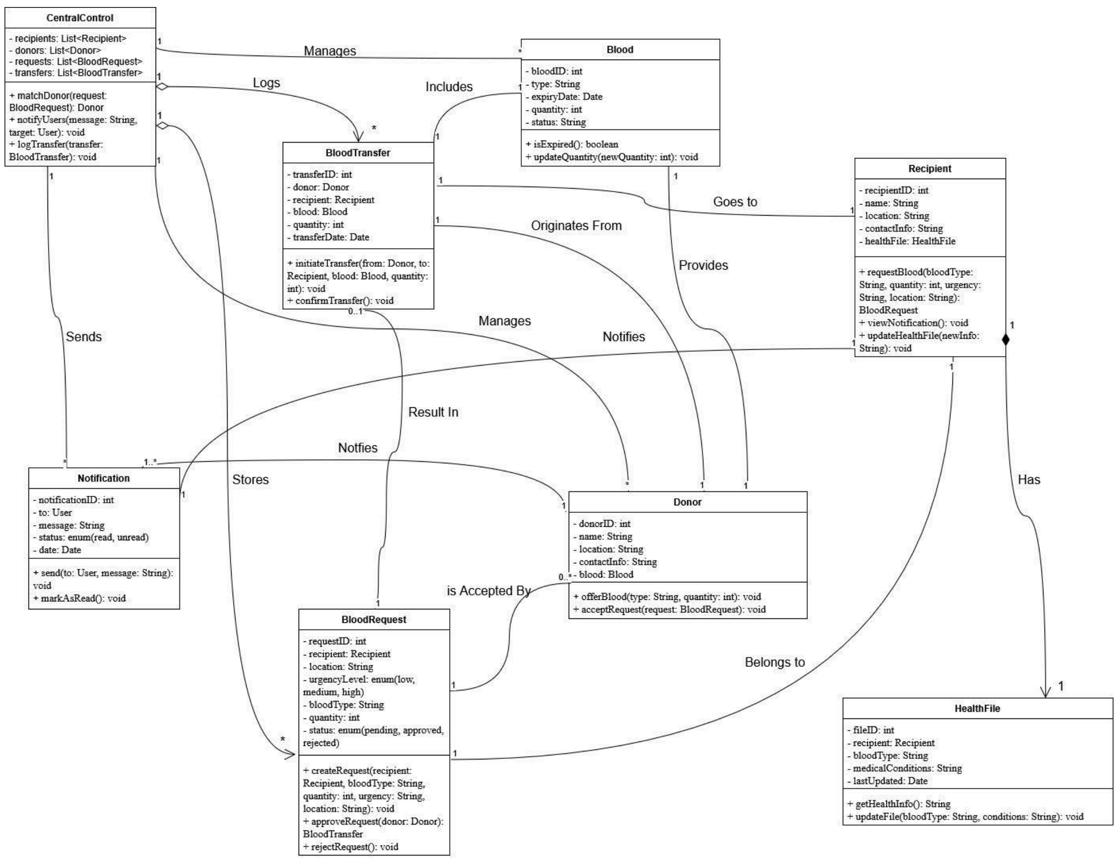
Phase 3: Design and Structuring

In the following, we will gradually showcase the conversion of the use-cases to class diagrams, and applying refinements and combining them into one UML class diagram. Following that, we will also showcase examples of objects created from these classes to enhance the understanding of them and their importance.

3.1 Converting use cases to class diagram

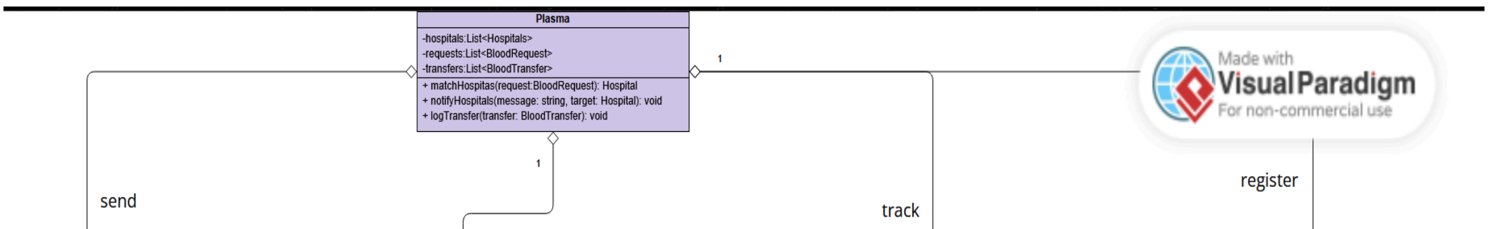
Conversion of use-case 1:

Use case 1 is concerned with the scenario of a replacement donation. By looking at the use-case diagram, main entities include: Hospitals, Donors, Recipients, Blood and Blood Request. Submission of and response to replacement donations are by donors and hospitals. Furthermore, donors and hospitals receive requests through notifications which must be a class of its own since it has unique identifying attributes such as date of sending and status. A blood request may be accepted, which would then be considered a blood transfer. The distinction between the two is integral since their states and points of failure can be different. Each donation/ transfer is concerned with a specific blood of a certain type with an expiry date. Both the donor and the recipient have a brief health file which is composed from the Sehaty API. The health file entity is important to better the filtration process when choosing a donor and keep track of donation histories, health conditions and impactful disease. It is also a point which would be further updated upon blood transfer. All these transactions and operations are logged and recorded in a central point which most entities interact with as well.


CentralControl
- recipients: List<Recipient>
- donors: List<Donor>
- requests: List<BloodRequest>
- transfers: List<BloodTransfer>
+ matchDonor(request: BloodRequest): Donor
+ notifyUsers(message: String, target: User): void
+ logTransfer(transfer: BloodTransfer): void
Blood
- bloodID: int
- type: String
- expiryDate: Date
- quantity: int
- status: String
+ isExpired(): boolean
+ updateQuantity(newQuantity: int): void
BloodTransfer
- transferID: int
- donor: Donor
- recipient: Recipient
- blood: Blood
- quantity: int
- transferDate: Date
+ initiateTransfer(from: Donor, to: Recipient, blood: Blood, quantity: int): void
+ confirmTransfer(): void
Recipient
- recipientID: int
- name: String
- location: String
- contactInfo: String
- healthFile: HealthFile
+ requestBlood(bloodType: String, quantity: int, urgency: String, location: String): BloodRequest
+ viewNotification(): void
+ updateHealthFile(newInfo: String): void
Notification
- notificationID: int
- to: User
- message: String
- status: enum(read, unread)
- date: Date
+ send(to: User, message: String): void
+ markAsRead(): void
Donor
- donorID: int
- name: String
- location: String
- contactInfo: String
- blood: Blood
+ offerBlood(type: String, quantity: int): void
+ acceptRequest(request: BloodRequest): void
BloodRequest
- requestID: int
- recipient: Recipient
- location: String
- urgencyLevel: enum(low, medium, high)
- bloodType: String
- quantity: int
- status: enum(pending, approved, rejected)
+ createRequest(recipient: Recipient, bloodType: String, quantity: int, urgency: String, location: String): void
+ approveRequest(donor: Donor): BloodTransfer
+ rejectRequest(): void
HealthFile
- fileID: int
- recipient: Recipient
- bloodType: String
- medicalConditions: String
- lastUpdated: Date
+ getHealthInfo(): String
+ updateFile(bloodType: String, conditions: String): void
Manages
Logs
Includes
Goes to
Originates From
Provides
Sends
Manages
Notifies
Result In
Notfies
Stores
Has
is Accepted By
Belongs to
1
*
0..1
1..*
0..*

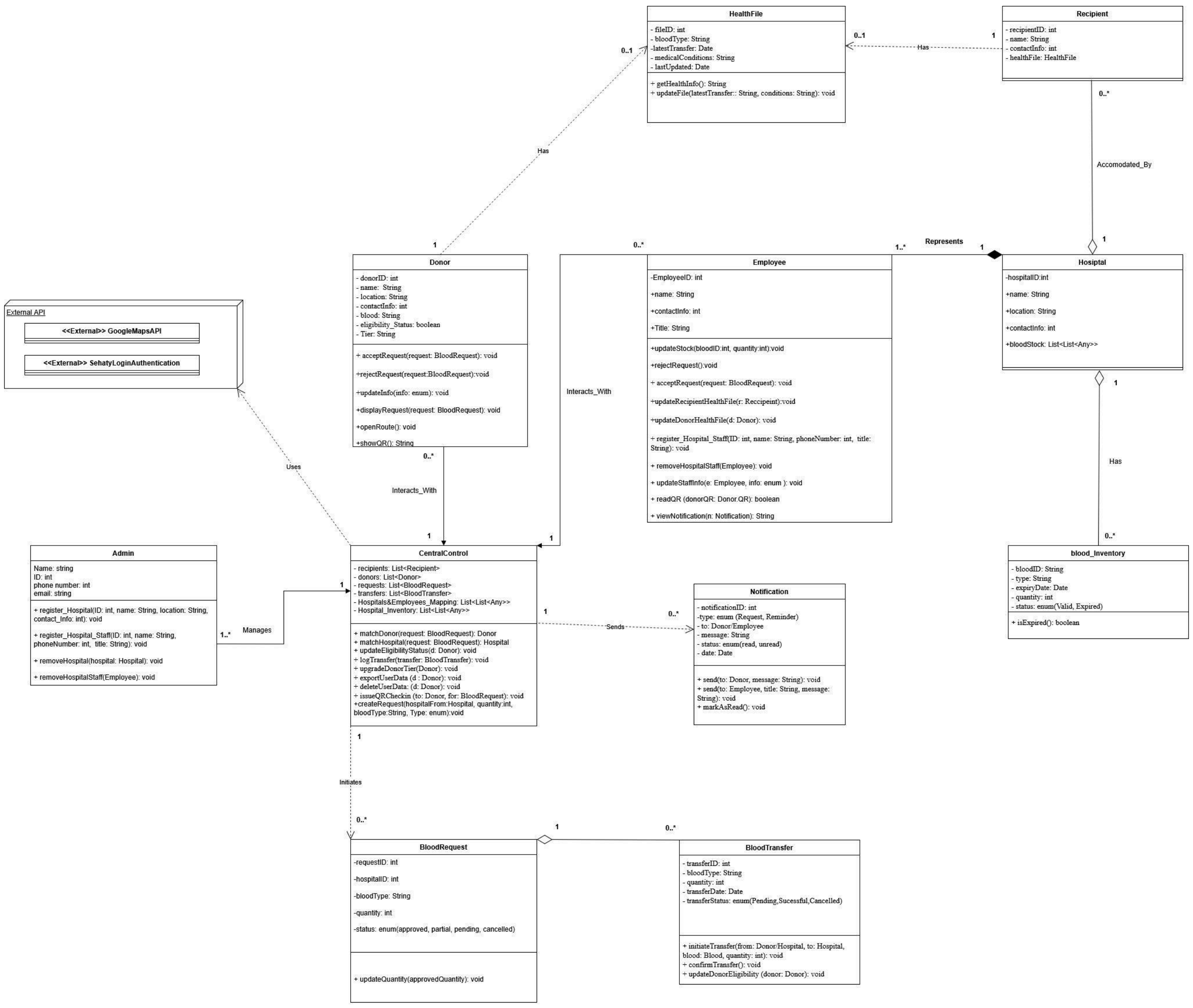
Conversion of use-case 2:

Use case 2 depicts the process of inter-communication between hospitals for efficient exchange of blood units. It starts by a hospital submitting a request for a certain blood with unique attributes. Following that, the central system which is named here “Plasma” sends a notification to eligible hospitals which may or may not respond. Once a hospital accepts the request, the request is then considered a transfer similar to the circumstances of the previous diagram. Both ends interact with a central point of contact which is named as Plasma here, though it has the same purpose of what was named as “CentralControl” in the previous diagram.


Plasma
-hospitals:List<Hospitals>
-requests:List<BloodRequest>
-transfers:List<BloodTransfer>
+ matchHospitas(request:BloodRequest): Hospital
+ notifyHospitals(message: string, target: Hospital): void
+ logTransfer(transfer: BloodTransfer): void
1
1
send
track
register
Made with Visual Paradigm For non-commercial use

Having created individual conversions and initial designs, we then combined both designs, though we also revised existing design components including class existence, class naming, relationship types, multiplicities, and even attributes and functions withing each module. In the following, we will showcase the final class diagram as well as design justifications to improve the overall understanding of he diagram.

Final UML class diagram:


HealthFile
- fileID: int
- bloodType: String
-latestTransfer: Date
- medicalConditions: String
- lastUpdated: Date
+ getHealthInfo(): String
+ updateFile(latestTransfer:: String, conditions: String): void
Recipient
- recipientID: int
- name: String
- contactInfo: int
- healthFile: HealthFile
Has
0..1
1
0..1
Has
1
0..*
Accomodated_By
1
Represents
1..*
1
0..*
Donor
- donorID: int
- name: String
- location: String
- contactInfo: int
- blood: String
- eligibility_Status: boolean
- Tier: String
+ acceptRequest(request: BloodRequest): void
+rejectRequest(request:BloodRequest):void
+updateInfo(info: enum): void
+displayRequest(request: BloodRequest): void
+openRoute(): void
+showQR(): String
Employee
-EmployeeID: int
+name: String
+contactInfo: int
+Title: String
+updateStock(bloodID:int, quantity:int):void
+rejectRequest():void
+ acceptRequest(request: BloodRequest): void
+updateRecipientHealthFile(r: Reccipeint):void
+updateDonorHealthFile(d: Donor): void
+ register_Hospital_Staff(ID: int, name: String, phoneNumber: int, title: String): void
+ removeHospitalStaff(Employee): void
+ updateStaffInfo(e: Employee, info: enum): void
+ readQR (donorQR: Donor.QR): boolean
+ viewNotification(n: Notification): String
Hosipital
-hospitalID:int
+name: String
+location: String
+contactInfo: int
+bloodStock: List<List<Any>>
1
Has
0..*
External API
<<External>> GoogleMapsAPI
<<External>> SehatyLoginAuthentication
Uses
Interacts_With
0..*
Interacts_With
1
1
Admin
Name: string
ID: int
phone number: int
email: string
+ register_Hospital(ID: int, name: String, location: String, contact_Info: int): void
+ register_Hospital_Staff(ID: int, name: String, phoneNumber: int, title: String): void
+ removeHospital(hospital: Hospital): void
+ removeHospitalStaff(Employee): void
1..*
Manages
1
CentralControl
- recipients: List<Recipient>
- donors: List<Donor>
- requests: List<BloodRequest>
- transfers: List<BloodTransfer>
- Hospitals&Employees_Mapping: List<List<Any>>
- Hospital_Inventory: List<List<Any>>
+ matchDonor(request: BloodRequest): Donor
+ matchHospital(request: BloodRequest): Hospital
+ updateEligibilityStatus(d: Donor): void
+ logTransfer(transfer: BloodTransfer): void
+ upgradeDonorTier(Donor): void
+ exportUserData (d : Donor): void
+ deleteUserData: (d: Donor): void
+ issueQRCheckin (to: Donor, for: BloodRequest): void
+createRequest(hospitalFrom:Hospital, quantity:int, bloodType:String, Type: enum):void
1
Sends
0..*
Notification
- notificationID: int
-type: enum (Request, Reminder)
- to: Donor/Employee
- message: String
- status: enum(read, unread)
- date: Date
+ send(to: Donor, message: String): void
+ send(to: Employee, title: String, message: String): void
+ markAsRead(): void
blood_Inventory
- bloodID: String
- type: String
- expiryDate: Date
- quantity: int
- status: enum(Valid, Expired)
+ isExpired(): boolean
1
Initiates
0..*
BloodRequest
-requestID: int
-hospitalID: int
-bloodType: String
-quantity: int
-status: enum(approved, partial, pending, cancelled)
+ updateQuantity(approvedQuantity): void
1
0..*
BloodTransfer
- transferID: int
- bloodType: String
- quantity: int
- transferDate: Date
- transferStatus: enum(Pending,Sucessful,Cancelled)
+ initiateTransfer(from: Donor/Hospital, to: Hospital, blood: Blood, quantity: int): void
+ confirmTransfer(): void
+ updateDonorEligibility (donor: Donor): void

Brief description of differences and refinements:

Main entities that were retained are BloodRequest, BloodTransfer, Hospital, Donor, Recipient, CentralControl, Notifications, and HealthFile. However, even with these modules, relationships and multiplicities were refined which will be discussed further later on. The role of the recipient was watered down where now the recipient is just responsible for arriving at the hospital, whilst the hospital is responsible for submitting requests and interacting with the system. Still, the presence of the recipient entity is of major importance to update their health record. A hospital in of itself must have an employee to represent it when interacting with the system, hence, the creation of the employee class was brought to life with the employee having all the actions that used to be in the Hospital class. An admin of the platform is introduced to better align with the functional requirements mentioned in section 2.3.1. Another factor of the functional requirements which was added here is the use of external APIs such as for navigation and authentication, though they were represented superficially due to the conceptual nature of the diagram as well as its purpose to depict the internal structure of the system. Unnecessary relationships with entities like BloodRequest, and Donor were excluded to reduce coupling and improve flexibility to future changes. The blood entity was converted to represent the blood available at the inventory whilst any other existence of blood details especially in replacement scenarios were considered as attributes within classes.

Design Justifications (Relationships and Multiplicities):

CentralControl group of relationships:

- All multiplicities on the side of this module were marked as 1 since only a singular object of this module will exist.
- **Admin-CentralControl:** at least one admin must exist to manage this module especially since it contains information that the system admin must introduce including the registration of new hospitals and initial employees.
- **CentralControl-Notification**: In the lifetime of the system, it will send many notifications to different receiving ends. However, 0..* multiplicity is important here since at the startup and introduction of the platform, it must function correctly even if it still didn't send any notifications.
- **CentralControl-BloodRequest**: Similar to its relationship with notifications, it will initiate many requests during its lifetime, though it must be functional in the beginning at 0 requests.
- **CentralControl-Donor**: All Donors will interact with the central control for various functionalities, though the central control must be functional with no donors registered.
- **CentralControl-Employee**: Similarly to its relationship with donor, this module must work correctly even with no hospitals being registered as of yet. Employees interact with this module mainly to request blood and respond to requests, though also a QR functionality is of use to them.
- Finally the module also relies on external APIs for authentication and navigation. Most relationships are Association relationships since they are generally independent of each other, however, notifications' and blood requests' existence relies on the behavior and actions of the central control which forms a dependency (uses) relationship.

BloodRequest—BloodTrasfer:

- The nature of the relationship is an aggregation since a the existence of a blood transfer refers to a certain blood request which has triggered it, and its existence depends on the existence of a blood request.
- With regards to multiplicity a single blood request may not be accepted, hence the possibility of 0 blood transfers. Furthermore, if a request is for a large quantity of blood, we may need many blood transfers to accommodate it.

Hospital group of relationships:

- **Hospital-Employee**: There must be at least one employee to represent the hospital for it to exist in the system. This is a strong dependency of existence where also the existence of an employee object will terminate shall the existence of the hospital dies. Which is why it was depicted as a composition relationship. All employee must refer to a single hospital.
- **Hospital-Recipient**: The existence of a recipient depends on the existence of the hospital accommodating them. However, a hospital can exist with zero recipients for the purpose of initial startup or voluntary donations. So it is an aggregation relationship with 0 or more patients referring to a singular hospital.
- **Hospital-BloodInventory**: The blood stock is a part of a hospital though at the initial startup of the system, a hospital may not have blood stock and still be able to function well. An aggregation relationship was appointed with zero or more blood batches possible all referring to a single hospital.

HealthFile group of relationships:

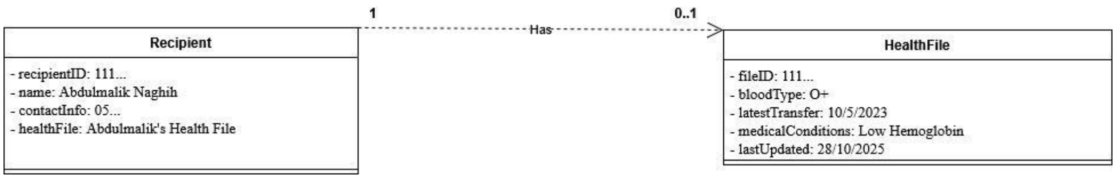
- Relationship type: Both the recipient and the donor may have a health file though it is not right to say that the module of the health file is part of a donor/recipient rather than it referring to them. Therefore, a dependency (uses) relationship was drawn.
- Multiplicities: a single donor/recipient may or may not have a health file especially if it was someone coming from abroad and not have been registered in the Saudi health system.

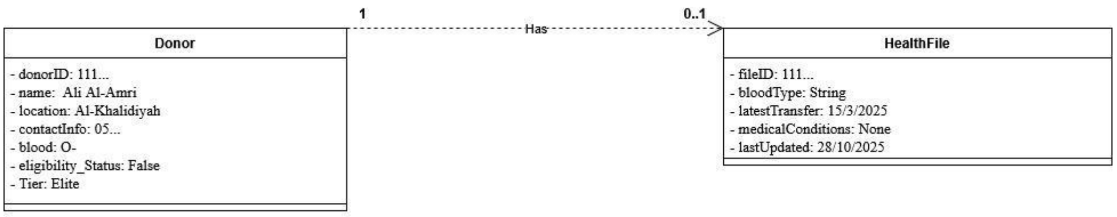
Future improvements:

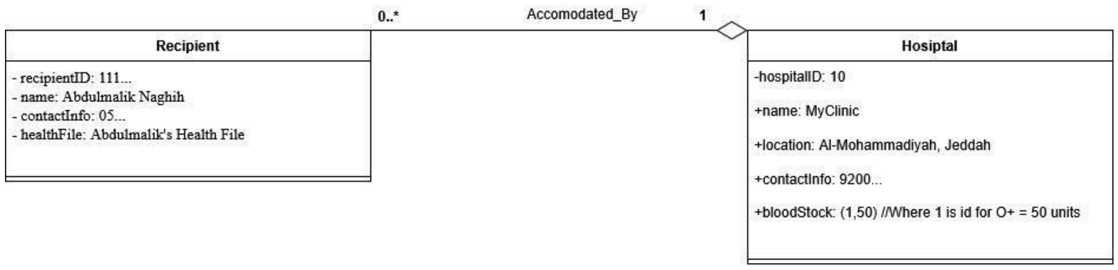
One main point to be refined is CentralControl. It is a module that is serving many purposes such as being a point of interaction, management, and a trigger for other modules like notifications. This lessens the flexibility and modularity of the system and may be a reliability threat in future maintenance sessions especially since the CentralControl being a central and single point of failure. It is planned to break down the CentralControl into various modules serving more specific purposes to reduce bottlenecking the system in.

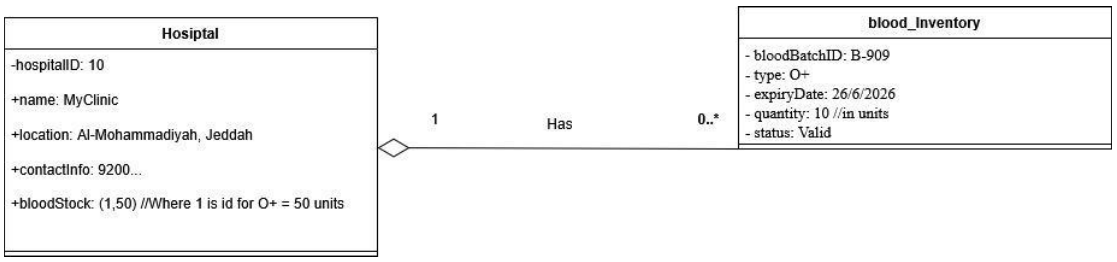
3.2 Object Diagrams:

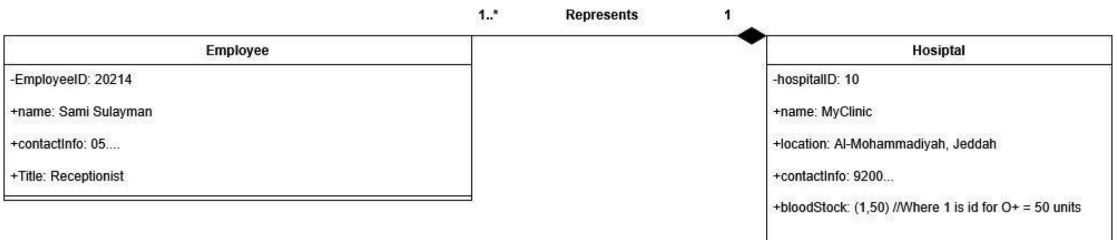
In the following, instances of the classes that were designed are represented:

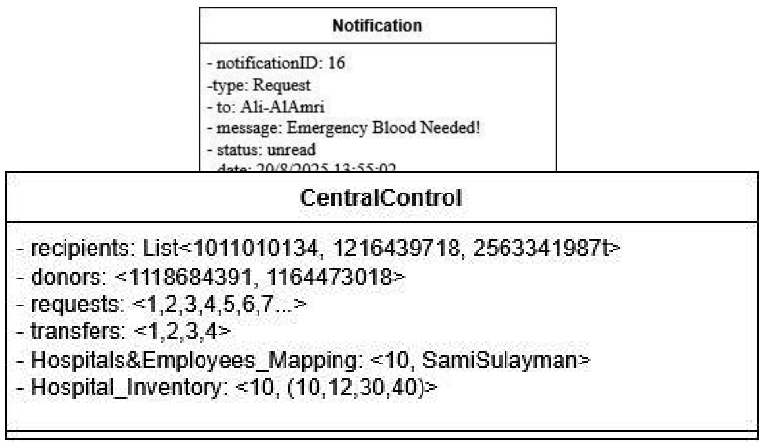

1
Has
0..1
Recipient
- recipientID: 111...
- name: Abdulmalik Naghih
- contactInfo: 05...
- healthFile: Abdulmalik's Health File
HealthFile
- fileID: 111...
- bloodType: O+
- latestTransfer: 10/5/2023
- medicalConditions: Low Hemoglobin
- lastUpdated: 28/10/2025




0..*
Accomodated_By
1
Recipient
- recipientID: 111...
- name: Abdulmalik Naghih
- contactInfo: 05...
- healthFile: Abdulmalik's Health File
Hosiptal
-hospitalID: 10
+name: MyClinic
+location: Al-Mohammadiyah, Jeddah
+contactInfo: 9200...
+bloodStock: (1,50) //Where 1 is id for O+ = 50 units





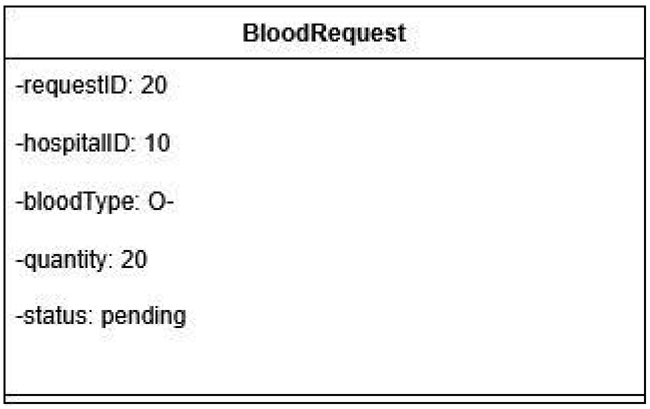


Notification

- notificationID: 16
-type: Request
- to: Ali-AlAmri
- message: Emergency Blood Needed!
- status: unread
- date: 20/8/2025 13:55:02

CentralControl

- recipients: List<1011010134, 1216439718, 2563341987t>
- donors: <1118684391, 1164473018>
- requests: <1,2,3,4,5,6,7...>
- transfers: <1,2,3,4>
- Hospitals&Employees_Mapping: <10, SamiSulayman>
- Hospital_Inventory: <10, (10,12,30,40)>



BloodRequest

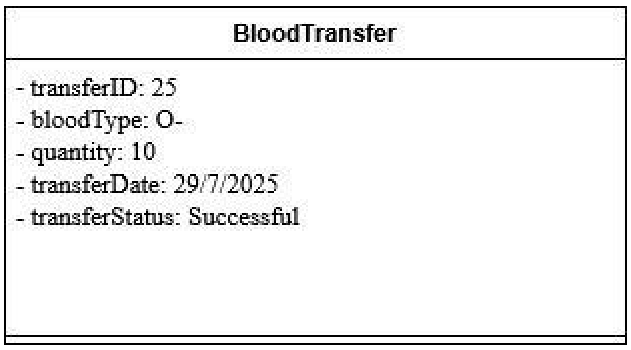
-requestID: 20

-hospitalID: 10

-bloodType: O-

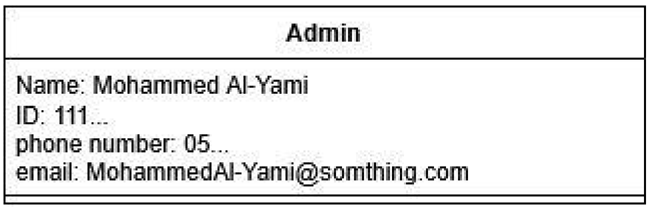
-quantity: 20

-status: pending



BloodTransfer

- transferID: 25
- bloodType: O-
- quantity: 10
- transferDate: 29/7/2025
- transferStatus: Successful



Phase 4: Modelling, Interaction & Behaviour

4.1 Interaction diagrams

4.1.1 Sequence Diagrams:

The following two diagrams represent two main functionalities/scenarios of the system. The first sequence diagram represents a hospital submitting a request for a replacement donation. The second diagram showcases the process of a hospital requesting blood from another hospital.

Sequence Diagram 1: Hospital-Donor Request

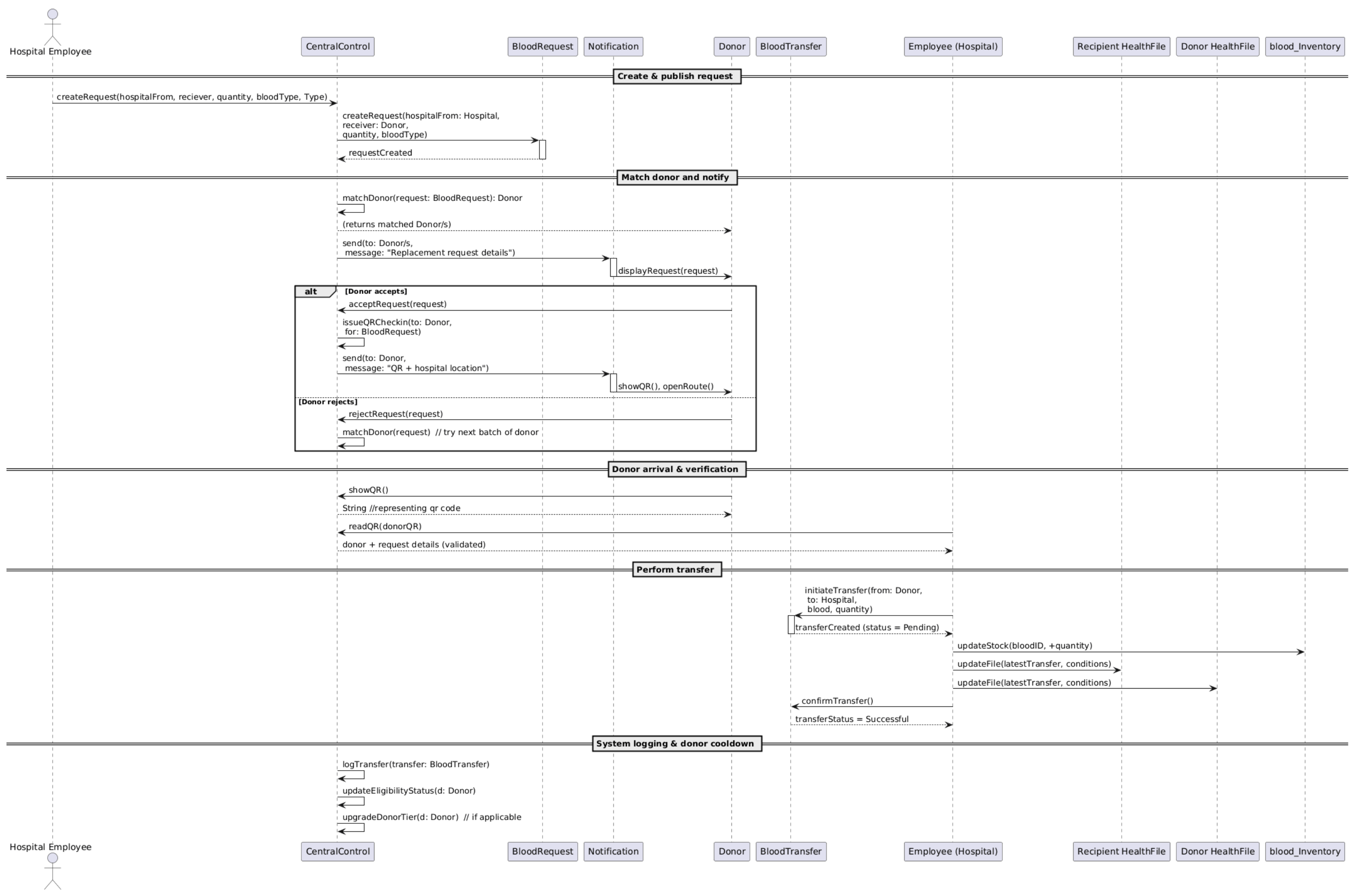

Hospital Employee
CentralControl
BloodRequest
Notification
Donor
BloodTransfer
Employee (Hospital)
Recipient HealthFile
Donor HealthFile
blood_Inventory
Create & publish request
createRequest(hospitalFrom, reciever, quantity, bloodType, Type)
createRequest(hospitalFrom: Hospital, receiver: Donor, quantity, bloodType)
requestCreated
Match donor and notify
matchDonor(request: BloodRequest): Donor
(returns matched Donor/s)
send(to: Donor/s, message: "Replacement request details")
displayRequest(request)
alt
[Donor accepts]
acceptRequest(request)
issueQRCheckin(to: Donor, for: BloodRequest)
send(to: Donor, message: "QR + hospital location")
showQR(), openRoute()
[Donor rejects]
rejectRequest(request)
matchDonor(request) // try next batch of donor
Donor arrival & verification
showQR()
String //representing qr code
readQR(donorQR)
donor + request details (validated)
Perform transfer
initiateTransfer(from: Donor, to: Hospital, blood, quantity)
transferCreated (status = Pending)
updateStock(bloodID, +quantity)
updateFile(latestTransfer, conditions)
updateFile(latestTransfer, conditions)
confirmTransfer()
transferStatus = Successful
System logging & donor cooldown
logTransfer(transfer: BloodTransfer)
updateEligibilityStatus(d: Donor)
upgradeDonorTier(d: Donor) // if applicable

Diagram 1 Description:

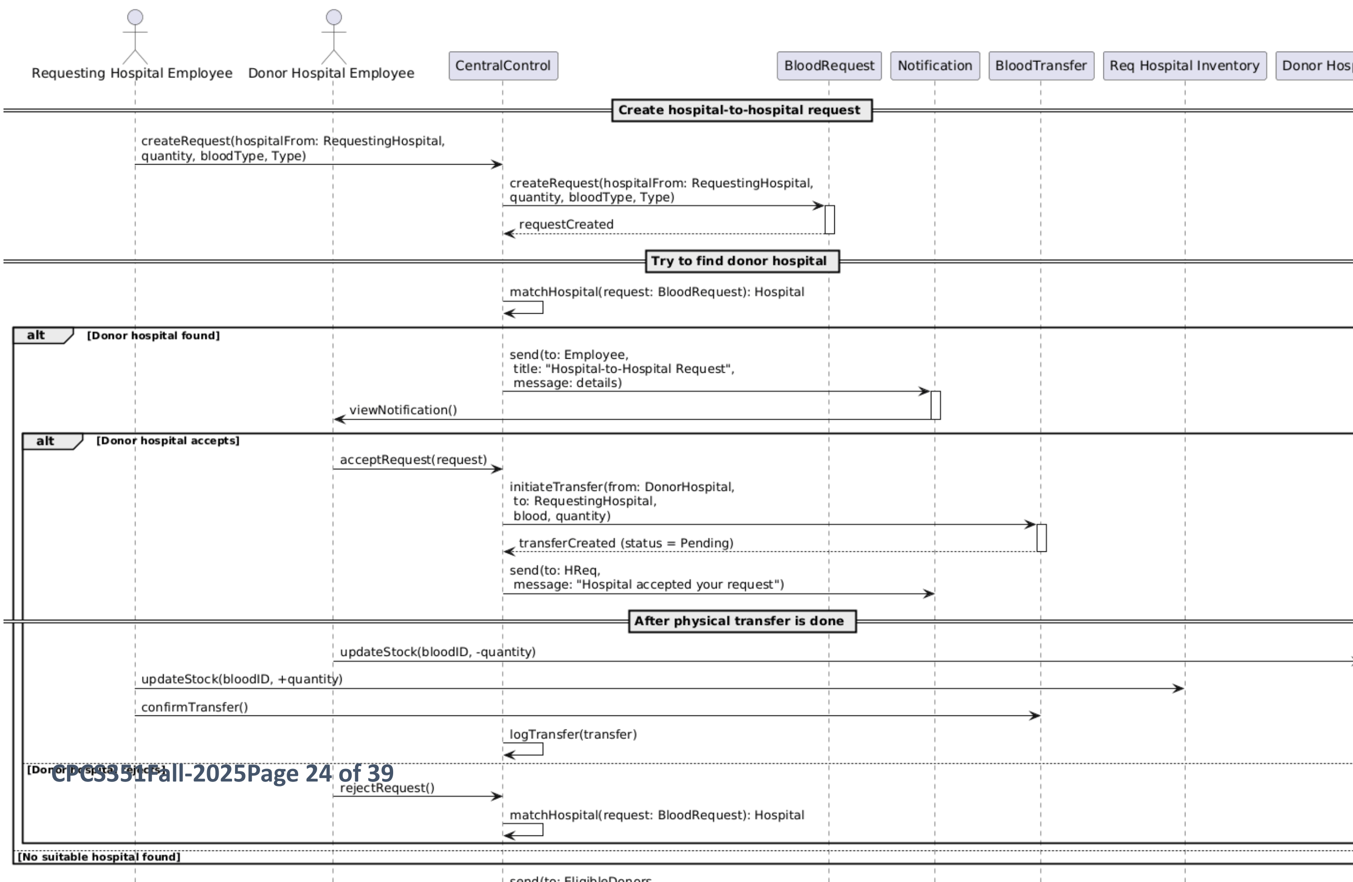
The process starts with the trigger which is the request creation. This is applied by the needing-hospital employee who is actually an employee class, though is represented as a stickman here to represent that they are the actor. The request creation method is present in the CentralControl class which itself creates the BloodRequest object in a manner similar to what's implemented in the factory or builder methodologies.

Potential donors are matched and returned to the central control which itself creates a notification object that is sent to those donors. These donors can then display these messages through their class method of display. Following that, we get into a 2-case scenario where the donor accepts or rejects the request. In the case of acceptance, a QR code is generated by CentralControl and the functionality of routing is enabled and both are sent to the donor. However, if the donor or set of donors all reject the request, the breadth of the search area is widened and the request notification is sent to the new potential donors.

For accepting donors, upon reaching the hospital, they show the check-in QR code for verification whilst the present employee reads it. Once that is done, the transfer is initiated which signifies that this instance is not just a request anymore. Once the donation process is done, the inventory and health file information are updated, and the transfer is marked as complete/successful.

Finally, the transfer information is logged, eligibility and donor tier information is all updated by the CentralControl.

Sequence Diagram 2: Hospital-Hospital Request



Requesting Hospital Employee
Donor Hospital Employee
CentralControl
BloodRequest
Notification
BloodTransfer
Req Hospital Inventory
Donor Hos
Create hospital-to-hospital request
createRequest(hospitalFrom: RequestingHospital, quantity, bloodType, Type)
createRequest(hospitalFrom: RequestingHospital, quantity, bloodType, Type)
requestCreated
Try to find donor hospital
matchHospital(request: BloodRequest): Hospital
alt
[Donor hospital found]
send(to: Employee, title: "Hospital-to-Hospital Request", message: details)
viewNotification()
alt
[Donor hospital accepts]
acceptRequest(request)
initiateTransfer(from: DonorHospital, to: RequestingHospital, blood, quantity)
transferCreated (status = Pending)
send(to: HReq, message: "Hospital accepted your request")
After physical transfer is done
updateStock(bloodID, -quantity)
updateStock(bloodID, +quantity)
confirmTransfer()
logTransfer(transfer)
[Donor hospital rejects]
rejectRequest()
matchHospital(request: BloodRequest): Hospital
[No suitable hospital found]
send(to: EligibleDonors

Diagram 2 description is listed in the following page □
Diagram 2 Description:

Similar to diagram 1, the process starts by the same trigger applied by the requesting hospital employee. CentralControl then creates the request object and computes potential matching hospitals through its internal board information.

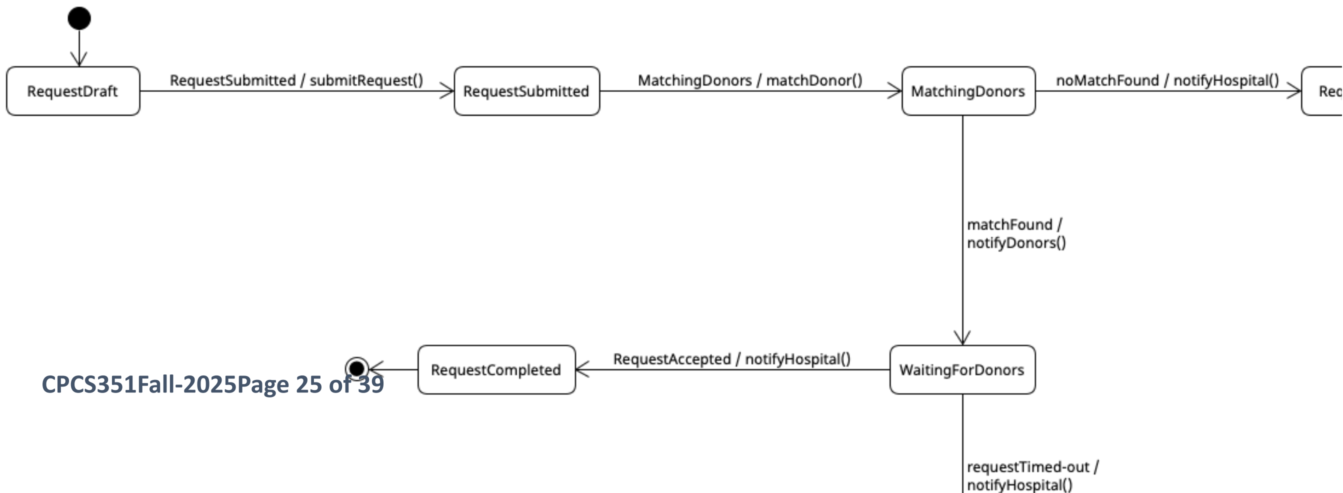
Following this, we stumble upon a 2-case situation, either a hospital is found or not. If a hospital is found, CentralControl then sends a notification to an employee or a set of suitable employees of the request through which they might accept or reject. On the other hand, if no hospital is found that has extra blood type in its inventory, we switch to the process of diagram 1 and search for matching donors.

If the potential donor-hospital accepts, a transfer object is created and set to pending. The object is marked as complete/successful once the receiving hospital employee confirms that they have received it. Following that the transfer is logged. However, if the potential hospital rejects, the requesting hospital is notified and then no further action is taken.

4.1.2 State Diagrams:

One core class that is being continuously used throughout the two main functionalities is the BloodRequest class, and for this sole reason we decided to use the great power of state diagrams to represent the general behaviour of a request across the two main scenarios that were depicted in the sequence diagrams. It is important to mention that the method names were modified to accommodate for easy understanding and overall brevity. The first diagram will represent the flow of a request in a Hospital-Donor situation, while the second one will depict the flow in the Hospital-Hospital communication process:

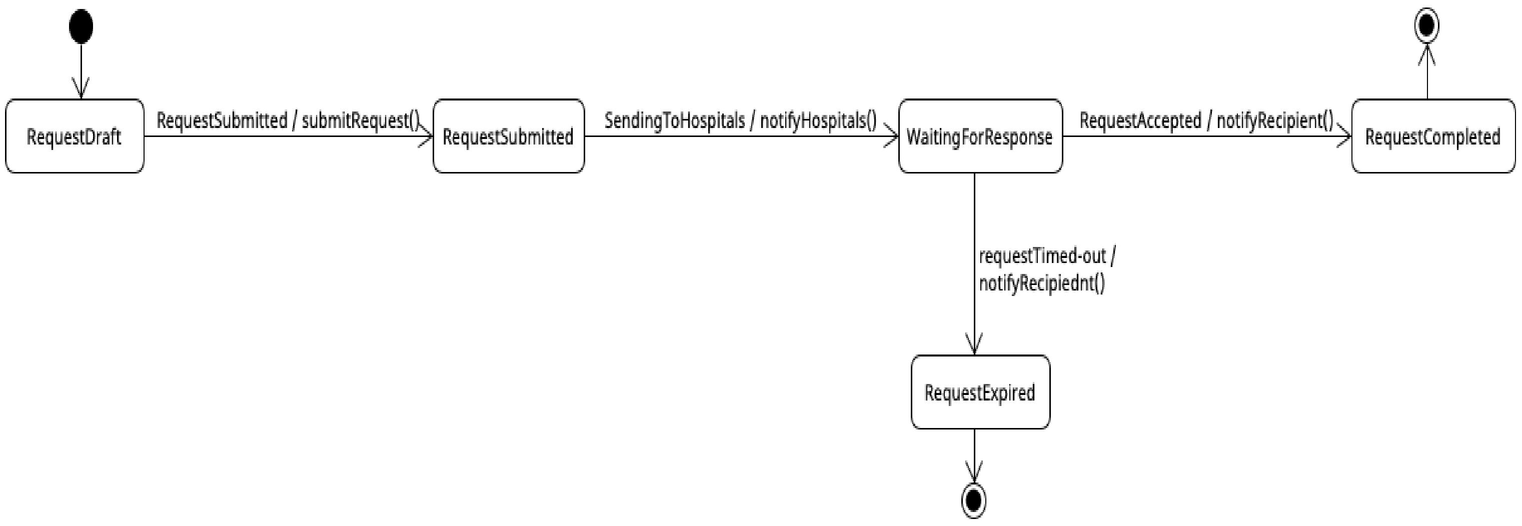
State Diagram 1: Request Behaviour in Hospital-Donor Communication



RequestDraft
RequestSubmitted / submitRequest()
RequestSubmitted
MatchingDonors / matchDonor()
MatchingDonors
noMatchFound / notifyHospital()
Rec
matchFound / notifyDonors()
WaitingForDonors
RequestAccepted / notifyHospital()
RequestCompleted
requestTimed-out / notifyHospital()

Diagram 2 is showcased in the following page

State Diagram 2: Request Behaviour in Hospital-Hospital Communication


RequestDraft
RequestSubmitted / submitRequest()
RequestSubmitted
SendingToHospitals / notifyHospitals()
WaitingForResponse
RequestAccepted / notifyRecipient()
RequestCompleted
requestTimed-out / notifyRecipiednt()
RequestExpired

4.1.3 Activity Diagrams:

Activity Diagram 1: Hospital-Donor Request

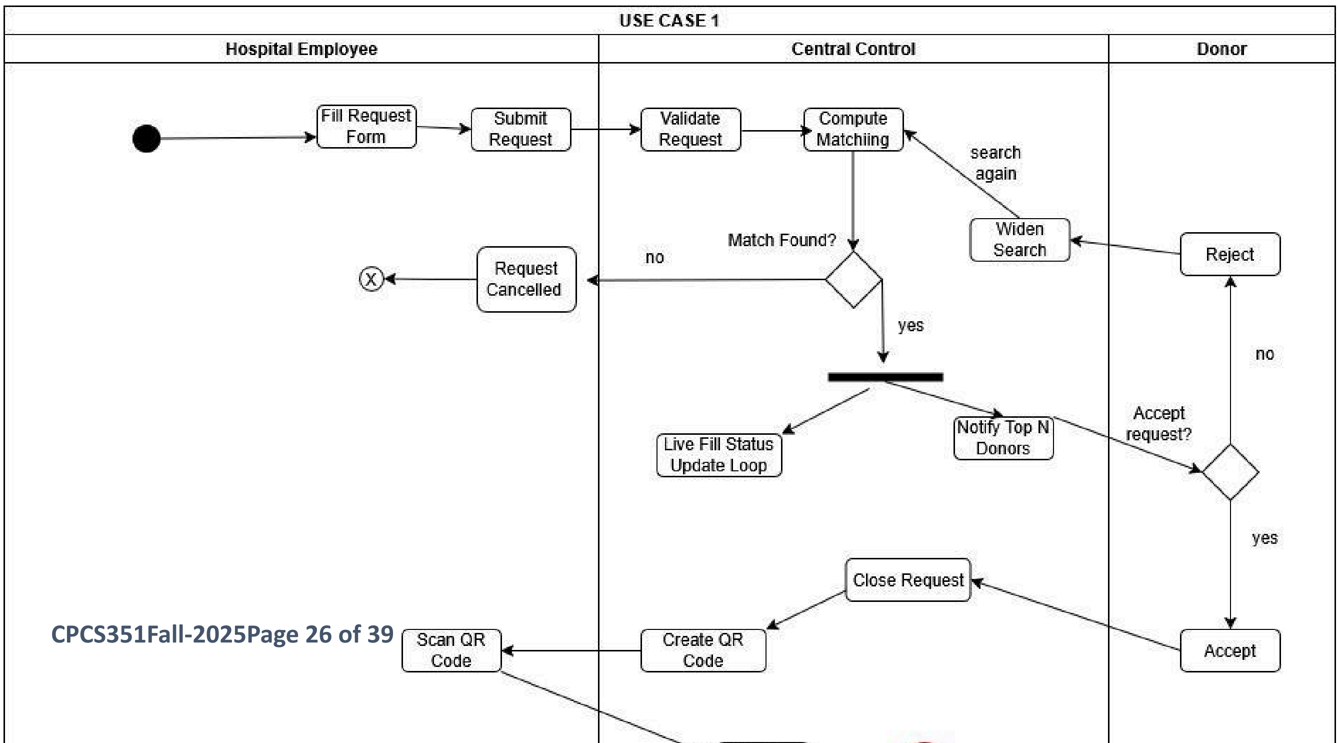


Diagram 1 description is elaborated in the following page □

Diagram 1 Description:

The activity diagram here showcases the set of actions and who they are performed by. Aligning with the sequence diagram depicted in the previous pages, the process always starts from the hospital employee who fills and submits the request.

The central control then validates and analyses the information in the request and computes matching potential donors. If none are found the request is cancelled. A good question would be well why don't we widen the search before cancelling? However, that is actually what happens and a widening process does take place to some extent before cancelling the request, however, we simplified things here to make the diagram more understandable.

If a match/s is found, the central control will notify the potential donor/s and possibly generate a QR code depending on their response. Once the QR is scanned, it means that the donor is present and therefore the central control can update their donation history (health file).

Activity Diagram 2: Hospital-Hospital Request:

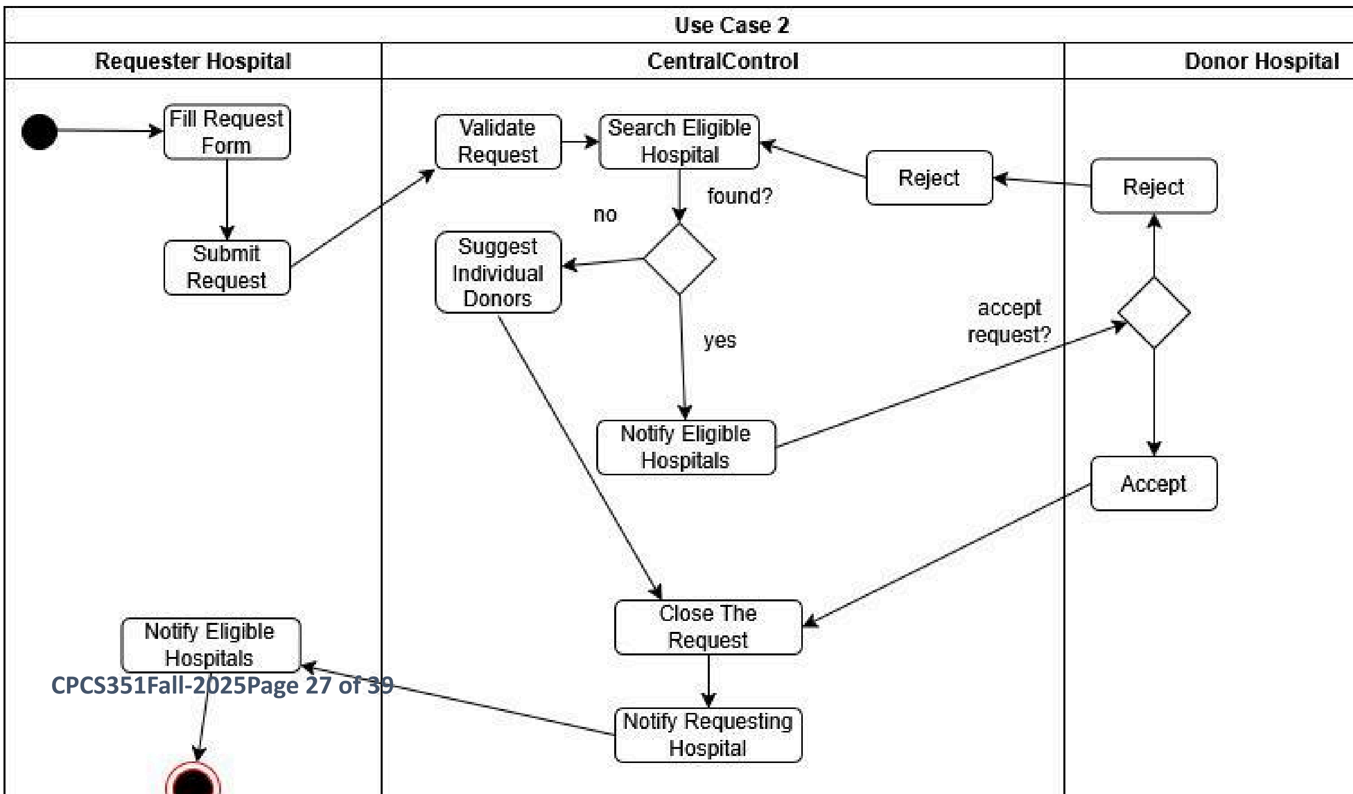


Diagram 2 description can be found in the following page □

Diagram 2 Description:

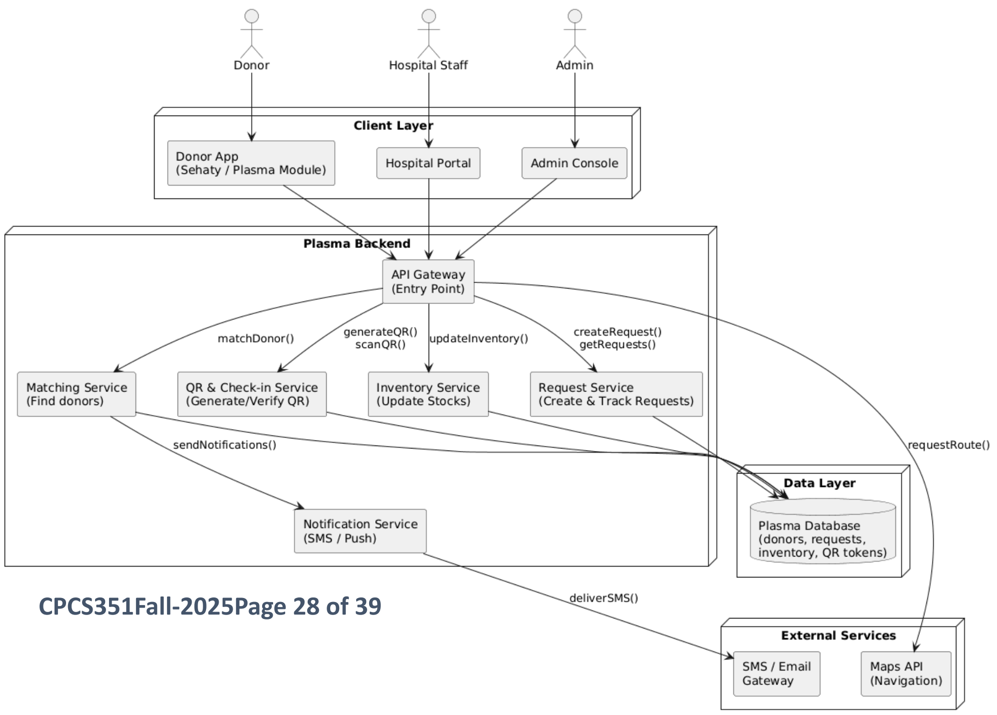
Very similar to diagram 1 in terms of set of activities and the regions to which they belong to, however, we can see here that in the case that no matching hospitals were found, the process switches to search for individual donors which would then take us to diagram 1 just like what was shown in sequence diagram 2.

To detail what's happening here, the employee of the requesting hospital fills and submits a request. The flow then is mainly at the central control which validates the request and searches for matching hospitals and notify them if they exist.

Based on the potential donating-hospital response, different actions are taken, yet in both cases the flow goes back to the central control which either further searches for other hospitals through certain metrics such as area, or switches the process to be a request to individuals rather than hospitals and closes the request that was to the hospitals. In all cases, the requesting hospital is notified and the process of hospital-hospital communication comes to an end.

Finally, though this is very lightly touched upon in the sequence and activity diagrams 2, for voluntary donations, there will a protocol to enforce inter-communication/request between hospitals before reaching out to individuals. The sole purpose of this protocol is to aid in reducing wastages.

4.1.4 System Architecture Diagram:



System Architecture Diagram Description:

Having gone through all the intricate parts of Plasma, we now zoom out and divide the system into main components. Below the main 3 human actors of the system resides what is often referred to as the front end. This is the interaction point between the system and the actors.

The front end takes in behaviour and sends it to a gateway which will contain a set of controllers that will guide where we shall then go. The flow is guided to the main components that typically will contain a group of the classes we showcased in the diagrams.

These low-level components continuously interact with the database for storing and retrieval, however, we tried to minimize the need for this type of contact since it is costly by embedding a matrix in the Central Control class for fast information retrieval. The storing process can then happen every end of day or even week.

External components are used either by the low-level components in the back-end of the system or by the guiding controllers in cases like using Nafath/Sehaty verification for example.

In the following pages, we will go over another aspect of the project which is concerned with delivering the project with good quality. We will showcase the testing objectives and strategies and go over them in detail.

Testing Objectives

Through the testing objectives plan, we aim to set the primary goals, expected outcomes of testing, and main points of testing. We aim to achieve the demonstration of the software satisfying the stated requirements (both functional and non-functional) and exclude undesired behavior that could compromise the system's effectiveness or security… basically ensuring that our system is orthogonal.

1- **Primary Testing Goals:**

   1.1- **Verify functional requirements compliance.**

   1.2- **Validate non-functional requirements.**

   1.3- **Ensure healthcare data security and privacy.**

   1.4- **Test system integration with external services.**

   1.5- **Validate critical business and logic algorithms.**

   1.6- **Identify and eliminate defects (from early on).**

   1.7- **Make sure system is ready to be deployed.**

2- **Specific Testing Objectives:**

   2.1- **Functional Testing Points:**

   - Verify blood request submission works correctly.

   - Make sure donor matching algorithm is working

   correctly.

   - Verify QR generating and reading properly function without misreads.

   - Verify donor cooldown periods are correctly calculated and limitations are enforced.

   2.2- **Performance Testing Objectives:**

   - Ensure donor matching responds within 4 seconds under normal load.

   - Ensure system maintains 99% uptime and availability with appropriate failover approaches.

   - Ensure system can handle 10,000 concurrent users without performance dips.

2.3- Security Testing Objectives:

- Verify that all health data is encrypted using industry-standard algorithms.

- Ensure authentication system prevents unauthorized access.

- Ensure role-based access properly restricts different user types to their allowed methods.

- Ensure the system is protected from common web-based attacks like SQL-injections.

2.4- Usability Testing Objectives:

- Ensure user interface is intuitive and requires minimal training.

- Ensure multi-lingual support, and that this feature functions correctly across different actions.

- Ensure all functions properly work across different operating systems and overall responsiveness is equal.

3- Testing Success Criteria:

- No critical or high-severity defects exist/remain.

- Unit test code coverage reaches minimum 80% threshold.

- All performance benchmarks are met.

- Security testing reveals no major vulnerabilities.

- System is stable over an extended time without crashes.

- All test cases are executed and results are recorded.

Testing strategy and techniques

Overview

This will be the process of evaluating Plasma to verify it satisfies specified requirements and to identify any gaps , errors, or missing requirements are strategies cover both functional and non functional testing requirements

Testing levels

unit testing

Test individual components (classes/methods) in isolation

Components to test :

- Doner eligibility validation methods
- Blood type matching algorithm
- Request validation functions
- Notifications generation logic

Method used : white box testing

Who : software developers

Integration testing :

Testing interactions between integrated modules

Integration points

- Central Control □□ notification System
- Sehaty/moh api □□healthFile module
- Hospital □□blood inventory
- booldRequst □□doner matching

method : combination of white box & black box

who : software testers & developers

System testing

Testing the complete integrated system against requirements

Includes:

- functional testing (features working as specified)
- Non functional testing (performance , security , usability)

Who : QA Team

Method : black box testing

Acceptance testing

Validating the system meets the stakeholder needs

Participants :

- Hospital staff (testing request workflows)
- Donors (test notification and acceptance)
- End User verify System readiness

Testing methods

Black box testing

Testing without knowledge of internal workings

Application: Testing system functionality by providing inputs and examining outputs

Used For:

- Use case validation
- Functional requirements verification
- User interface testing

White-Box Testing

Definition: Testing with knowledge of internal code structure

Application: Testing internal logic and code paths

Used For :

- Unit testing
- Code coverage analysis
- Algorithm validation (matching logic)

Grey-Box Testing

Definition: Testing with limited knowledge of internal workings

Application: Integration testing where testers know module interfaces but not full implementation

Functional Testing

Purpose: Verify the system performs intended functions correctly

- Test Steps:
- Determine functionality the application should perform
- Create test data based on specifications
- Define expected output based on specifications
- Write test scenarios and execute test cases
- Compare actual vs expected results

Key Functional Areas:

- Blood request submission (Use Case 1 & 2)
- Donor matching algorithm
- Notification system
- Cross-hospital coordination
- Authentication and authorization

Non-Functional Testing

Performance Testing

Focus: Speed, capacity, stability, scalability

Test Aspects:

- Response Time: Donor matching < 4 seconds
- Data rendering speed
- Network delay impact
- Database transaction processing

Sub-types:

- Load Testing: Test with 10,000 concurrent users
- Stress Testing: Test beyond normal load limits

Security Testing

Purpose : Identify security vulnerabilities and flaws

Test Areas:

- confidentiality of health data
- Data integrity
- Authentication (Sehaty/MOH integration)
- Authorization (role based access)
- Input validation
- SQL injection prevention
- Session management

Portability Testing

Purpose: Verify system works across different platforms

- Test Strategy:
- iOS and Android compatibility (Sehaty integration)
- Different hospital system environments
- Cross-browser compatibility

Black box testing techniques

Equivalence Partitioning

Definition: Dividing input data into valid and invalid partitions

Example for Blood Type Input:

  - Valid Partition: {A+, A-, B+, B-, AB+, AB-, O+, O-}
  - Invalid Partition: {XY, C+, null, numbers}

Test one value from each partition

Boundary Value Analysis

Definition: Testing at boundaries of input ranges

Example for Blood Quantity:

- Minimum boundary: 0, 1
- Maximum boundary: 49, 50 (if max is 50)

- Invalid: -1, 51

Test scenarios (use case)

Blood Request for Individual (Use Case 1)

Test Case 1: Valid Request

- Input: Blood type O+, 2 units, urgent, Hospital ID: 10
- Expected: System matches eligible donors, sends notifications
- Technique: Equivalence partitioning (valid partition)

Test Case 2: Invalid Blood Type

- Input: Blood type "XY", 2 units
- Expected: System rejects with error message
- Technique: Equivalence partitioning (invalid partition)

Test Case 3: Boundary Quantity

- Input: Blood type A+, 0 units
- Expected: System rejects (minimum boundary violation)
- Technique: Boundary value analysis

Hospital-to-Hospital Request (Use Case 2)

Test Case 4: Successful Transfer

- Input: Hospital A requests 5 units O-, Hospital B has 10 units
- Expected: System notifies Hospital B, transfer recorded
- Technique: Functional testing

Test Case 5: No Available Hospitals

- Input: Request for rare type, no hospitals have inventory
- Expected: System notifies requesting hospital, suggests individual donors

Technique: Cause-effect testing

Testing Priorities

Critical (Must Test First):

- Authentication & authorization

- Donor-patient matching algorithm
- Data encryption and security
- Emergency notifications

High Priority:

- Cross-hospital coordination
- Performance (response time)

Medium Priority:

- Inventory management
- Donor history tracking

Testing Tools

Manual Testing:

- Test cases executed by QA team
- Exploratory testing by testers

Automation Testing:

- Unit Tests: JUnit (Java) / pytest (Python)
- API Testing: Postman
- Performance: JMeter
- Security: OWASP ZAP

References

[1] H. Alsulami, A. Aljobran, S. Safhi, H. Alsulami, A. Aljobran, and S. Safhi, “Determination of rate, causes and cost of wastage of blood and its products in two blood donation centers in Riyadh,” *GSC Advanced Research and Reviews*, vol. 14, no. 1, pp. 055–059, 2023, doi: https://doi.org/10.30574/gscarr.2023.14.1.0015.

[2]
J. Alsughayyir *et al.*, “Demography and blood donation trends in Saudi Arabia: A nationwide retrospective, cross-sectional study,” *Saudi Journal of Biological Sciences*, vol. 29, no. 12, p. 103450, Dec. 2022, doi: https://doi.org/10.1016/j.sjbs.2022.103450.

[3]
E. SH, “Demographical Pattern of Blood Donors and Pre-Donation Deferral Causes in Dhahran, Saudi Arabia,” *DOAJ (DOAJ: Directory of Open Access Journals)*, Jul. 2020.

[4]

D. Almalki *et al.*, “Blood Donor Characteristics at a Hospital Blood Bank in Saudi Arabia: A Trend-analysis,” *Journal of Applied Hematology*, vol. 15, no. 3, pp. 169–175, Jul. 2024, doi: https://doi.org/10.4103/joah.joah_74_24.

[5]

A. A. English, “Saudi Crown Prince launches national blood donation campaign,” *Al Arabiya English*, Aug. 21, 2025. https://english.alarabiya.net/News/saudi-arabia/2025/08/22/saudi-crown-prince-launches-national-blood-donation-campaign (accessed Sep. 09, 2025).