



Independent University, Bangladesh

Project Title:

FoodSaver: Leftover Food Rescue System

Course: Web Applications & Internet (CSE309)

Prepared By: Shahla Sarmin Raisha

Date: 11/12/2025

Table of Contents

Table of Contents.....	2
Chapter 01: Introduction.....	2
1.1 Background.....	5
1.2 Motivation	5
1.3 Problem Statement.....	7
1.4 Objectives.....	8
1.5 Scope	9
1.6 Out of Scope	9
Chapter 02: System Overview.....	10
2.1 System Users and Their Roles.....	11
2.2 System Workflow Summary	12
2.3 System Rich Picture	14
2.4 Subsystem Breakdown.....	14
2.5 High-Level Architecture.....	15
2.6 Advantages of the System	16
2.7 System Constraints	17
2.8 Assumptions	17
2.9 Dependencies	17
Chapter 03: System Life Cycle.....	17
3.1 Agile SDLC Phases.....	18
3.2 Why Agile is Suitable for FoodSaver.....	22
3.3 Summary of Agile Workflow for FoodSaver.....	23
Chapter 04: Functional Requirements	23
4.1 Overview.....	23

4.2 List of Main Features	24
4.3 Feature Table	24
4.4 Functional Requirements Description	28
Chapter 05: System Specification Diagrams	30
5.1 Use-case Diagram	30
5.2 Use-case Narration.....	33
5.3 Data Flow Diagram	44
5.4 Activity Diagram	46
5.5 Sequence Diagram	52
5.6 Class Diagram	53
5.7 Entity Relationship Diagram.....	54
5.8 Deployment Diagram	55
Chapter 06: Non-Functional Requirements	56
6.1 Performance Requirements.....	56
6.2 Reliability Requirements	56
6.4 Usability Requirements	58
6.5 Maintainability Requirements	58
6.6 Scalability Requirements.....	59
6.7 Interoperability Requirements	59
6.8 Legal & Ethical Requirements	59
6.9 Availability Requirements	60
6.10 Portability Requirements	60
Chapter 07: Risk Analysis.....	61
7.1 SWOT Analysis	61
Chapter 08: Project Management	62
8.1 WBS	62
8.2Activity List	63
8.3 Gantt Chart	65
8.4 Network Diagram.....	66

Chapter 09: Feasibility Analysis	67
9.1 Expense Head to Implement the Project	68
9.2 Assumptions	68
9.3 Net Present Value (NPV) Table	69
9.4 Return on Investment (ROI) Chart	70
Chapter 10: Conclusion	71
Chapter 11: Reference	72

Chapter 01: Introduction

1.1 Background

Food wastage and imbalanced food distribution continue to be some of the major challenges faced globally. While a substantial part of the world, especially in developing countries like Bangladesh, faces nutrition insecurity, millions of tonnes of good, edible food are wasted each year. According to reports by FAO, nearly a third of food produced for human consumption is wasted annually. The majority of this food is wasted at the household, event, restaurant, and retail levels.

At the same time, many families, low-income earners, and daily wage workers depend on external support for food. Such disequilibrium between food surplus and food shortage requires innovative technological solutions that support the gathering, redistribution, and transparent monitoring of food donations.

The use of technology, especially web-based applications, acts as a crucial bridge in modern society. The donors, receivers, volunteers, and administrators can engage on one digital platform in a seamless ecosystem to ensure that food sharing is timely and safe. FoodSaver is designed for exactly that purpose.

1.2 Motivation

Food waste and hunger continue to coexist in developing nations, especially in urban areas with high population density. Despite the fact that much of the surplus food after events, restaurants, corporate lunches, and household dinners remains consumable, the absence of formal methods for redistribution means it goes to landfills instead of reaching hungry people. Many studies underscore that unavailability of intelligent food-sharing

digital platforms is among the important reasons for the waste. For instance, researchers mention that in developing countries, the inability to coordinate donors and beneficiaries due to inefficient manual systems, poor communication, and fragmented resources results in 25–35% of avoidable food waste every year [1]. Additionally, ICTs have already been shown to greatly decrease complexity across logistics in networks of food redistribution through the facilitation of real-time tracking and coordination of items donated [2]. Another study shows that urban food insecurity does not come about because of absolute lack, but because of ineffective means of distribution - families discard fully consumable food just because they are unaware of effective donation outlets [3].

Further, logistic inefficiencies in the delivery of donations, particularly at the last-mile transportation, lead to losses that diminish the quality of the food and discourage both donors and receivers from participating. Research in donation logistics says that coordination systems for volunteers are very important for on-time pickups and distributions of surplus food [4]. Digital platforms implementing structured workflows, such as donor posting, admin verification, volunteer matching, and delivery tracking, greatly enhance community participation, with full transparency. Lastly, research in digital platforms for social welfare has established that a community-driven approach reduces the costs of operations while enhancing engagement and long-term sustainability, with rapid improvement expected in technological access in low-income urban settings [5].

Guided by these results, FoodSaver is driven by the immediate need for closing the gap between the availability of surplus food and hunger in the community with an accessible, transparent, and smart web-based technological platform.

Supporting Points:

- ICT-enhanced food redistribution reduces wastage in urban centers through efficient communication flows. [2]
- Households often dispose of edible leftovers due to lack of awareness regarding structured redistribution systems. [3]
- Volunteer coordination remains paramount in ensuring timely collection and delivery of donations of food. [4]
- Digital platforms enhance trust, reduce donation barriers, and allow community participation. [5] Centralized systems avoid duplication, ensure transparency, and track the status of donations effectively. [1]

1.3 Problem Statement

Despite the great amount of surplus edible food produced in households, restaurants, corporate and community events, and institutional facilities daily, a centralized, coordinated, and technology-supported system does not exist to make this available for the neediest individuals and families. Many donors would like to share extra food; however, under current conditions, food redistribution activities are generally informal and unstructured, reliant upon manual methods such as telephone calls, personal networks, incidental social media postings, and ad hoc word-of-mouth agreements. All these discrete approaches are spasmodic, unreliable, and incapable of sustaining significant volumes or time-critical food redistributions-particularly perishable items necessitating timely pickups.

Because of this, informal processes often result in:

Delays in pickups and deliveries that cause food to spoil, further discouraging donors and receivers from consistent participation.

- Lack of transparency, whereby neither the donors nor receivers know what happens with the donation.
- Food safety issues because of the lack of verification, documentation, and standardized handling practices.
- Miscommunication among donors, volunteers, and receivers is common due to manual coordination and vague responsibilities.
- No tracking or accountability to confirm if food has been collected, delivered, or is wasted.
- Duplication or misappropriation of donations, as there is no system that stops multiple people from requesting the same item or claiming deliveries that are not made.

Moreover, on many occasions, receivers fail to locate food availability in time, and there is no organized platform for volunteers to view tasks, update the status of deliveries, and manage their assignments. Without any digital support, administrators also face immense hurdles in verifying donation authenticity, monitoring and preventing misuse, and observing standards that ensure food safety.

Therefore, continued reliance on informal, ad-hoc methods of communication creates inefficiencies, reduces trust in the donation process, and directly contributes to ongoing food waste and unmet community needs. Equally, the solution requires a robust, transparent, and digitally coordinated system that will ease the communication and connect all stakeholders in ensuring traceability and, consequently, safe, fast, and efficient redistribution of surplus food.

1.4 Objectives

Our goal is to have a smart, transparent, and efficient food donation ecosystem.

Core Objectives

1. Provide an easy-to-use platform whereby donors can post surplus food.
2. Allow receivers to request and track food reliably.
3. Facilitate volunteers to pick up and deliver food efficiently.
4. Enable administrators to confirm postings, handle multi-level roles, and ensure food safety.
5. Provide total transparency by way of real-time tracking and status updates.
6. Reduce food waste by connecting all stakeholders digitally.
7. Improve delivery efficiency by optimally assigning volunteers.
8. Ensure long-term sustainability and measurable community impact.

1.5 Scope

Included in FoodSaver are:

- Donor registration, login, and posting of donations
- Food post verification by admin
- Receiver request and approval workflow
- Volunteer task claiming
- Delivery status tracking Accepted → Picked → Delivered
- Volunteer-Donor-Receiver communication via dashboard
- User, post, and assignment management by admin
- Status changes (automated notifications)
- Rating and feedback system
- History viewing for donor and receiver
- Secure Data Storage and Audit Logs

1.6 Out of Scope

The following are not included in this release intentionally:

- AI-powered spoilage prediction
- Automatic route optimization algorithms
- Payment, subscription, or financial transactions
- Live chatting/messaging system
- Quality testing of food using IoT devices
- Integration of third-party logistics services
- Food pick-up scheduling calendar
- Government regulatory compliance modules
- Voice command or chatbot interfaces
- These may be included in future versions.

Chapter 02: System Overview

The FoodSaver Web Application is an online food redistribution system designed to minimize food waste by integrating donors, receivers, volunteers, and administrators on a unified digital platform. The system supports the posting, verification, requesting, assignment, and delivery tracking of surplus food. It overcomes the potential communication gaps, coordination difficulties, and lack of transparency within manual food donation networks through the automation of major workflows with role-specific dashboards.

FoodSaver has a three-tier system that encompasses:

1. Presentation Layer: User interface for donors, receivers, volunteers, and admin.
2. Application Layer – Backend logic; PHP, MySQL queries, business rules.
3. Data Layer: The database that stores users, posts, requests, assignments, history, and logs.

It is a platform that ensures surplus food is safely redistributed to the people that need it, with consideration for accountability, traceability, and food safety.

2.1 System Users and Their Roles

1. Donor

Individuals, households, restaurants, and event organizers that have excess edible food.

Key privileges:

- Create, edit and delete food donation posts
- View verification status of posts
- Track delivery of accepted posts
- View donation history
- Rate delivery quality

2. Receiver

Food insecure individuals or entities that request food donations.

Key privileges:

- View approved donation posts
- Need something? Request it!
- Track request approval
- Track delivery status
- View Request History
- Rate food quality

3. Volunteer

Registered helpers responsible for transportation and last-mile delivery.

Key privileges:

- See available tasks
- Claim pick-up assignments
- Update delivery status - Accepted → Picked Up → Delivered
- View their assignment history

4. Administrator

System manager ensuring verification, safety, and smooth operations.

Key privileges:

- Approve and verify food posts accept/reject donation requests
- Monitor delivery progress
- User management, roles, and disputes
- View system-wide reports

2.2 System Workflow Summary

FoodSaver has a very structured and traceable workflow:

1. Food post from a donor

Includes food type, quantity, pickup location, time, and expiration.

2.Admin checks the post

Admin checks whether the food is safe and suitable for donating.

3. Receiver requests approved food

Receiver places an order along with their information and delivery address.

4. Admin confirms receiver request

Ensures equitable distribution and prevents misuse.

5. Volunteers claim the delivery tasks

A volunteer takes responsibility for the collection of food from the donor.

6. Volunteer updates pickup & delivery status

Status flows as: Accepted --> Picked Up--> Delivered

7. System updates donor and receiver dashboards.

Both parties can track real-time progress.

8. Feedback & Rating

Receiver and donor each provide ratings on food quality and volunteer delivery.

2.3 System Rich Picture

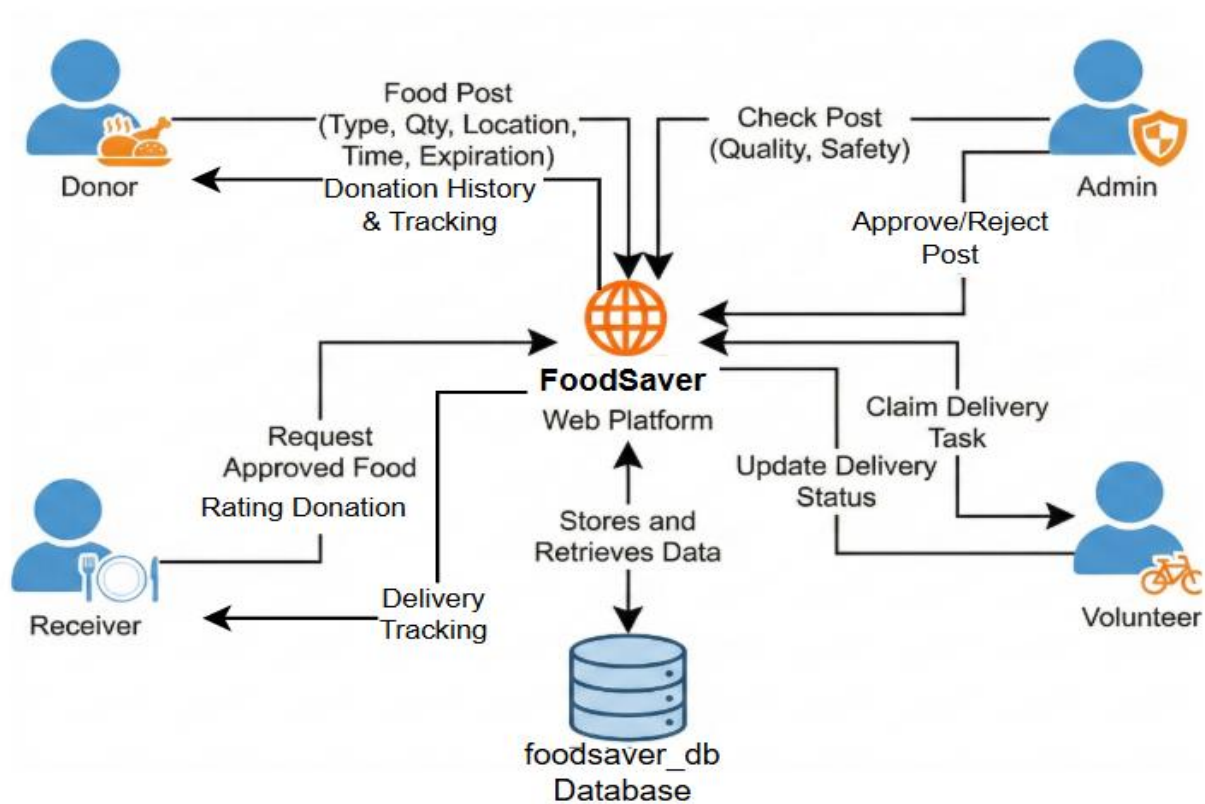


Fig: Rich Picture

2.4 Subsystem Breakdown

The three major subsystems of the system are:

A. Donation Management Subsystem

Purpose: Manage the creation, verification, and management of food donation posts.

Includes:

- Food posting form
- Post verification by admin
- Post editing/deleting
- Status management: pending, approved, and rejected

B. Food Request & Volunteer Assignment Subsystem

Purpose: To manage the matching between receiver needs and volunteer availability.

Includes:

- Request approval
- Volunteer task claiming
- Assignment status updates
- Routing & delivery information
- Update global post status after delivery

C. Tracking & History Subsystem

Purpose: To bring transparency and accountability.

Includes:

- Track donated food
- Track request status
- Track delivery status
- View donation history
- View request history
- Rating System

2.5 High-Level Architecture

FoodSaver employs a modular, component-driven architecture:

Front-end Technologies

- HTML5

- CSS
- JavaScript
- Responsive Dashboard Layouts

Back-End Technologies

- PHP core application logic
- MySQL (database)
- PDO prepared statements - security, SQL-injection protection
- Security Features
- Role-based access control
- Input sanitization
- Session-based login system
- Hash-based password storage
- Admin-only approval systems
- Data validation for safety

2.6 Advantages of the System

- Reduces food waste.
- Creates transparent and trackable workflows.
- Ensures food safety through admin verification.
- Supports vulnerable populations.
- Uses volunteers for efficient distribution.
- Automates communication and delivery coordination.
- Generates historical insights and reports.
- Encourages active community participation through feedback mechanisms.

2.7 System Constraints

- Requires stable internet access.
- Admin availability influences approval delay.
- Delivery depends on volunteer availability.
- Cannot track real-time location as it does not integrate with GPS yet.
- Users must provide valid pickup and delivery addresses.

2.8 Assumptions

- Donors can only provide safe, edible food.
- Receivers provide precise delivery information.
- Volunteers follow the specific steps of delivery.
- Admin does verification responsibly.
- Users possess basic digital literacy.

2.9 Dependencies

- PHP Server (XAMPP).
- MySQL database.
- Browser support.
- Secure user session handling Email/SMS notification (if added in future).

Chapter 03: System Life Cycle

The FoodSaver Web Application is developed following the Agile Software Development Life Cycle because it offers flexibility, iterative refinement, and the ability to handle continuous changes in system requirements based on stakeholder feedback. Considering the fact that food redistribution workflows involve many actors, such as donors, receivers,

volunteers, and administrators, Agile is the best methodology for developing and delivering quality features incrementally in a manner that allows for adaptability to user needs and real-world operational challenges.

Agile development methodology focuses on iterative development, continuous delivery, user feedback loops, and close collaboration among developers, designers, domain experts, and stakeholders. The FoodSaver will be developed in several iterations, time-boxed in sprints, with each aimed at producing a functional and testable component of the system. Instead of delivering the system in its entirety, Agile breaks this into smaller, manageable chunks, allowing the team to re-fine requirements, incorporate the evolving expectations, and improve usability through continuous improvements.

In the case of FoodSaver, Agile SDLC will ensure that these critical functions-food posting, verification workflows, handling requests, management of volunteer tasks, status of delivery, and feedback mechanisms-are iteratively developed with continuous stakeholder involvement. This goes well with the dynamic nature of the ecosystem of food donation whereby operational processes, user behavior, and needs within the community might change with time.

3.1 Agile SDLC Phases

The following are the agile life cycle phases of the FoodSaver system:

Requirements Gathering & Analysis

During the preliminary stage, functional and non-functional requirements are gathered by the project team through stakeholders such as donors, volunteers, receivers, and administrators. Analysis includes:

- Understanding how donors post food
- Defining approval rules for administrators
- Mapping receiver request flows
- Understanding Volunteer Delivery Routing
- Identifying needs for safety, accountability, transparency

For each feature, user stories and acceptance criteria are developed. Examples:

- As a donor, I would like to post surplus food easily so that it reaches the needy people.
- As a receiver, I would like to request food in order to satisfy my current needs.
- As a volunteer, I want to update the statuses of delivery so that the system remains transparent.
- As an admin, I want to verify the posts about food for safety.

These stories form the backlog and guide the rest of the phases.

Architectural Design

During this phase, the high-level architecture of the system is drafted. Decisions include:

- Three-tier architecture: presentation, logic, data
- PHP-based backend logic
- MySQL relational database design
- Role-based access control (RBAC)
- Separation of donor, receiver, volunteer, and admin modules
- Future scalability plan (API integration, mobile app etc.)

Wireframes, ERDs, sequence diagrams, and class diagrams are iteratively developed and refined.

Sprint Planning

Before the start of each sprint, stakeholders and developers collaborate to prioritize the user stories. High-value and high-risk features are built early.

Sprint planning encompasses:

- Select 5–10 user stories per sprint
- Breaking down stories into technical tasks
- Estimating effort by Story Points
- Identifying Dependencies and Risks

The aim is to develop, at the end of every sprint, a small but fully functional module.

Iterative Development

Development is done in cycles by building, integrating, and testing features incrementally. A single sprint usually lasts from 1 to 2 weeks.

Examples of sprint-based development:

- Sprint 1: user authentication, role creation, dashboard UI
- Sprint 2: Donor food post module + admin verification
- Sprint 3: Receiver request system
- Sprint 4: Volunteer task claiming + delivery status
- Sprint 5: Rating, history pages and analytics
- Sprint 6: Performance optimization + UI polish

Each sprint delivers a working product increment ready for testing and demonstration.

Testing & Quality Assurance

With Scrum, testing is built into every sprint, not just at the end. It includes:

- Unit Testing
- Integration testing
- User-acceptance testing - UAT
- Security Testing
- Performance & usability tests

The stakeholders test new increments regularly, confirm that features exceed expectations, and detect issues in the early stages.

Review & Retrospective

Each sprint ends with:

Sprint Review

- Demonstration of New System Capabilities
- Stakeholder feedback
- Deciding which improvements or additions to make next.

Sprint Retrospective

- Team performance evaluation
- Identification of Challenges
- Improvement strategies for the upcoming sprints

This feedback loop improves product quality and development efficiency.

Deployment

After several iterations, the application is ready to be deployed. The deployment steps are as follows:

- Database setup
- Server Configuration
- Domain integration
- Live environment testing
- Security hardening
- Documentation and User Training

Deployment can happen incrementally-module by module-or as a complete release.

Maintenance & Continual Improvement

Even after its deployment, FoodSaver is continuously updated to:

- Fix bugs
- Improve User Interface
- Add new features
- Performance improvement
- Support more donors, volunteers, and receivers
- Integrate analytics dashboards

Agile ensures maintenance is not a final stage, but an ongoing activity.

3.2 Why Agile is Suitable for FoodSaver

Some reasons for choosing Agile SDLC include:

- Real world food redistribution processes frequently change.

- Need for constant communication amongst stakeholders.
- High system complexity, requiring modular iteration.
- Continuous improvement incorporating community feedback
- Rapid prototyping and early visibility to donors and admins
- Scalability for future extensions

Agile provides support for the dynamic, socially-driven nature of this FoodSaver initiative.

3.3 Summary of Agile Workflow for FoodSaver

1. Collect Requirements
2. Break into user stories
3. Prioritize features
4. Develop the system in sprints.
5. Test Continuously
6. Collect feedback
7. Improve and refine
8. Deploy final product
9. Continue enhancements

Chapter 04: Functional Requirements

4.1 Overview

The functional requirements of the FoodSaver Web Application describe the capability, behavior, and interaction a system is supposed to provide to support effective food donation, verification, processing of requests, delivery coordination, and feedback management. These requirements are based on extensive analysis of the system's goals, user needs, and operational workflows observed in the domain of food redistribution and community support services.

They were grouped based on primary user roles: Donor, Receiver, Volunteer, and Administrator, for clarity, maintainability, and modular system design. Features are identified from the operational logic expressed in the codebase, discussions among stakeholders, the user stories, and real-life challenges related to the management of surplus food.

This section organizes the functional requirements into structured feature descriptions and then presents them in a comprehensive table for ease of understanding and implementation.

4.2 List of Main Features

1. User Authentication & Role Management
2. Food Donation Post Management - Donor Module
3. Post Verification and Moderation - Admin Module
4. Request Management (Receiver Module)
5. Volunteer Module: Task Claiming and Delivery Management
6. Tracking Deliveries and Order Status
7. Rating and Feedback System
8. History Management (Donor/Receiver/Admin)

4.3 Feature Table

Feature Name	Description	Actors Involved	Inputs	Outputs	Constraints	Dependencies	Priority

User Authentication & Role Management	Allows users (donor, receiver, volunteer, admin) to register, log in, and access role-based dashboards.	Donor, Receiver, Volunteer, Admin	Email, password, role	Login success, dashboard redirection	Secure password storage, unique email	Database, session management	High
Create Food Donation Post	Donors submit food posts including category, quantity, pickup location, images, and details.	Donor	Food details, quantity, location	New post added with status "Pending"	Must follow food safety guidelines	Authentication, DB insert	High
Edit/Delete Food Post	Donors can update or remove posts before admin approval.	Donor	Updated post info	Updated/deleted post	Only allowed when status \neq Approved	Admin verification dependency	Medium

Verify Food Post	Admins review submission and approve/reject based on safety and authenticity.	Admin	Submitted post	Updated status: Approved/Rejected	Admin must provide justification for rejection	Post submission	High
Request Food Post	Receivers request approved posts by providing reason, delivery address, and preferences.	Receiver	Request details	Request confirmation	Only available for Approved posts	Admin approval (optional)	High
Claim Delivery Task	Volunteers claim available tasks for approved receiver requests.	Volunteer	Task selection	Task assigned to volunteer	One volunteer per task	Receiver request approval	High

Update Delivery Status	Volunteers update statuses: Accepted → Picked Up → Delivered.	Volunteer	Assignment ID, status	Delivery update	Cannot skip status sequence	Task claiming	High
Track Donated Food	Donors track progress of their posted items.	Donor	Track request	Status timeline	Requires active posts	Delivery status updates	Medium
Track Delivery	Receivers track volunteer's delivery progress.	Receiver	Request ID	Delivery timeline	Requires assigned volunteer	Update delivery status	Medium
Rate & Review Donation	Receivers provide rating and feedback for received food.	Receiver	Rating, comments	Stored feedback	One rating per delivery	Delivery completion	Medium
View Donation History	Donors view their past donated posts and outcomes.	Donor	None	History logs	N/A	Post lifecycle	Low

View Request History	Receivers view previous requests and delivery outcomes.	Receiver	None	Request logs	N/A	Delivery completion	Low
-----------------------------	---	----------	------	--------------	-----	---------------------	-----

4.4 Functional Requirements Description

A. User Authentication & Role Management

The system shall provide a means of secure user registration and authentication. Each user, upon registration, shall select a role: Donor, Receiver, Volunteer, or Administrator. The system shall ensure that each role accesses only the functionality permitted. Password encryption and session validation upon login are required to prevent unauthorized access.

B. Food Donation Post Management

Donors should be able to create food posts that clearly detail the type of food, estimated quantity, location for pickup, how it is preserved, conditions where the food can expire, and any additional notes. The system shall store the post with a default “Pending” status. A donor should be allowed to edit or delete the post until reviewed by the administrator.

C. Post Verification and Moderation

The administrators shall validate every posted submission for food safety. The admin may approve it or reject with comments. Rejected posts will be returned to the donor stating the reasons.

D. Request Management

Receivers can request approved food posts by providing a delivery address and the reason of the request. They should be able to view the status of the request and edit it, but only before approval.

E. Volunteer Task Assignment

Volunteers claim tasks from available requests. Once a task is accepted, other volunteers cannot view or claim that particular task. Volunteers will update the delivery status as they progress.

F. Delivery Tracking

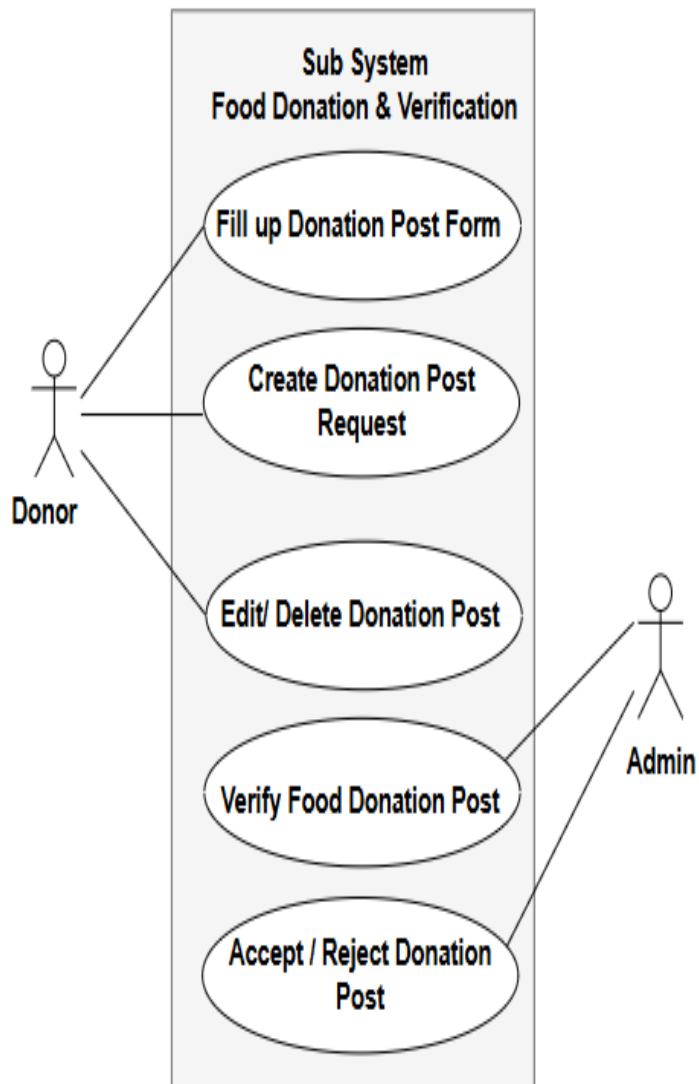
Both the donor and receiver will monitor their order of food delivery using real-time status updates by volunteers.

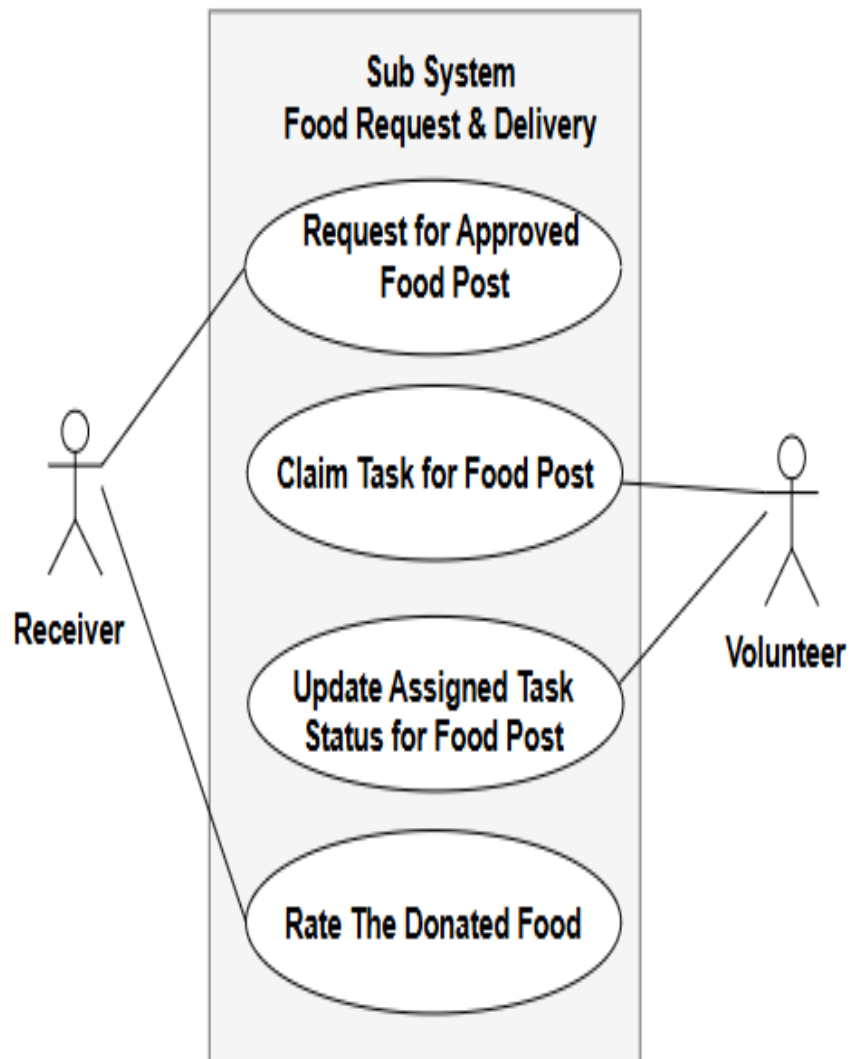
G. Rating & Feedback system

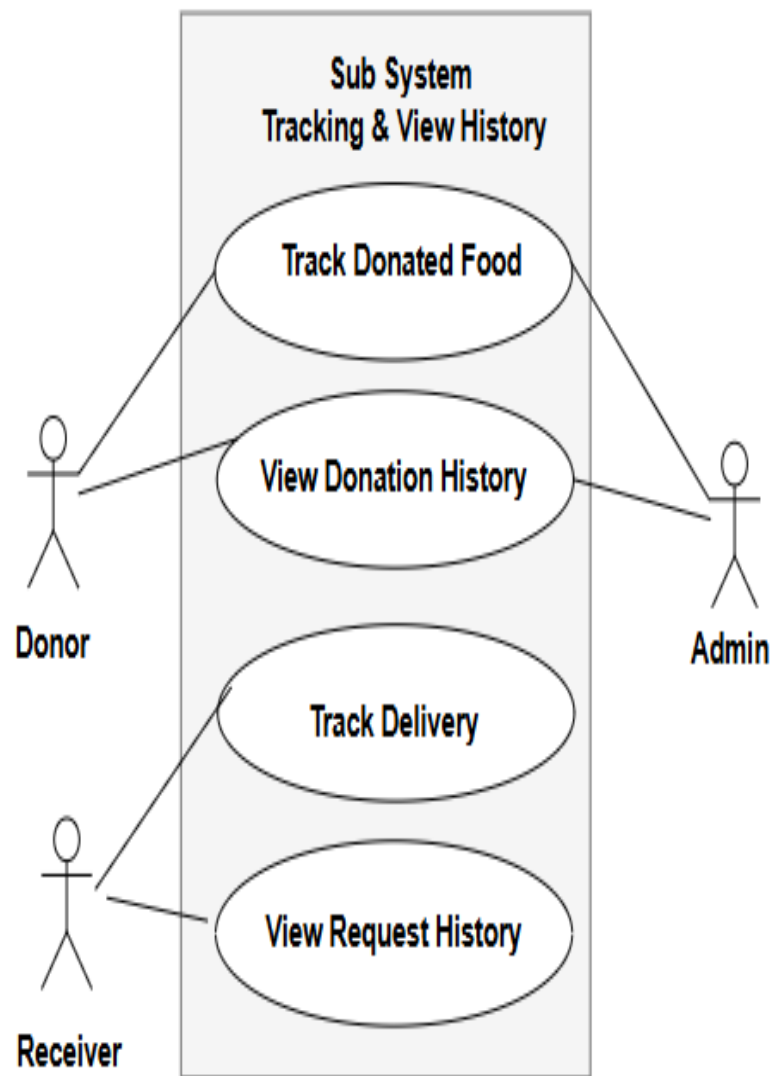
The receivers shall rate the received food through stars, rating from 1 to 5, and comments upon receipt for boosting transparency and follow-through quality.

Chapter 05: System Specification Diagrams

5.1 Use-case Diagram







5.2 Use-case Narration

Fill Up Donation Post Form

Actors:	Donor
Description:	Donor fills up donation form with required food details.
Preconditions:	Donor is logged in.
Triggers:	Donor selects 'Add New Donation'.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Opens donation form • Step2: Enters required details • Step3: Submits the form <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Loads donation form • Step2: Validates inputs • Step3: Stores data in database • Step4: Confirms submission
Alternate Actions:	<ul style="list-style-type: none"> • If required fields are missing → system shows error messages.
Postconditions:	Donation post is submitted and stored.

Create Donation Post Request

Actors:	Donor
Description:	Donor submits a donation post for approval.
Preconditions:	Donor is logged in.
Triggers:	Donor clicks 'Submit Donation'.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Selects donation post • Step2: Confirms submission <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Saves request • Step2: Marks post as pending • Step3: Notifies admin
Alternate Actions:	<ul style="list-style-type: none"> • System rejects invalid or incomplete data.
Postconditions:	Donation request is sent for admin review.

Edit/Delete Donation Post

Actors:	Donor
Description:	Donor edits or deletes an existing pending donation post.
Preconditions:	Donation post exists.
Triggers:	Donor clicks Edit/Delete.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Selects a post • Step2: Chooses edit or delete • Step3: Saves changes <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Fetches post data • Step2: Updates/deletes record • Step3: Refreshes list
Alternate Actions:	<ul style="list-style-type: none"> • If trying to edit an approved post → system denies action.
Postconditions:	Post updated or removed.

Verify Food Donation Post

Actors:	Admin
Description:	Admin verifies food donation posts and checks details.
Preconditions:	Admin is logged in.
Triggers:	Admin selects pending posts.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Reviews post details • Step2: Watch food video <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Loads post • Step2: Validates data
Postconditions:	Post verification completed.

Accept/Reject Donation Post

Actors:	Admin
Description:	Admin approves or rejects donation post after checking validity.
Preconditions:	Post is verified.
Triggers:	Admin chooses Accept/Reject.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Opens post • Step2: Selects accept or reject <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Updates status • Step2: Notifies donor
Alternate Actions:	<ul style="list-style-type: none"> • Admin may delay decision.
Postconditions:	Donation post accepted or rejected.

Request for Approved Food Post

Actors:	Receiver
Description:	Receiver requests food from approved donation posts.
Preconditions:	Receiver is logged in; post is approved.
Triggers:	Receiver clicks 'Request Food'.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Selects food post • Step2: Confirms request <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Records request • Step2: Notifies donor and volunteer
Alternate Actions:	<ul style="list-style-type: none"> • System denies request if post already requested by another user.
Postconditions:	Food request successfully placed.

Claim Task for Food Post

Actors:	Volunteer
Description:	Volunteer claims the delivery task.
Preconditions:	Task is available.
Triggers:	Volunteer selects 'Claim Task'.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Opens task list • Step2: Claims a task <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Assigns volunteer • Step2: Updates delivery table
Postconditions:	Task assigned to volunteer.

Update Assigned Task Status

Actors:	Volunteer
Description:	Volunteer updates delivery progress.
Preconditions:	Task is assigned.
Triggers:	Volunteer clicks Update Status.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Opens task • Step2: Updates status <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Saves status update • Step2: Updates timeline
Postconditions:	Delivery status updated.

Rate the Donated Food

Actors:	Receiver
Description:	Receiver gives a rating after receiving food.
Preconditions:	Delivery is completed.
Triggers:	Receiver selects 'Rate Donation'.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Opens rating form • Step2: Submits rating <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Validates rating • Step2: Stores rating
Alternate Actions:	<ul style="list-style-type: none"> • Receiver skips rating.
Postconditions:	Rating stored successfully.

Track Donated Food

Actors:	Donor
Description:	Donor tracks real-time status of their donated food.
Preconditions:	Donor has donation posts.
Triggers:	Donor selects Track Donations.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Selects a post • Step2: Views status <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Fetches donation records • Step2: Shows timestamps and volunteer info
Alternate Actions:	<ul style="list-style-type: none"> • If donor has no donation posts → system displays “No Donations Found.”
Postconditions:	Donor sees current donation status.

View Donation History

Actors:	Donor, Admin
Description:	User views past donations and records.
Preconditions:	Historical data exists.
Triggers:	User selects Donation History.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Opens history page • Step2: Scrolls records <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Retrieves old entries • Step2: Displays all details
Alternate Actions:	<ul style="list-style-type: none"> • If no historical data exists → system displays an empty list.
Postconditions:	Donation history displayed.

Track Delivery

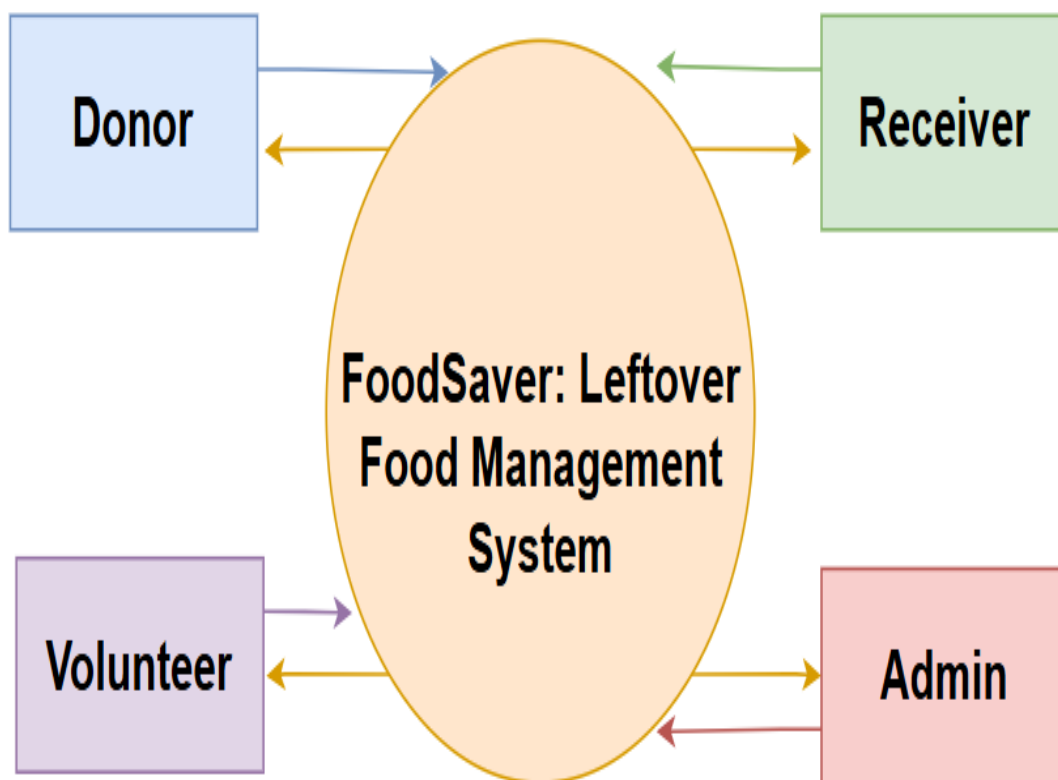
Actors:	Receiver, Admin
Description:	Tracks delivery progress for food requests.
Preconditions:	Delivery task assigned.
Triggers:	User selects Track Delivery.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Chooses a request • Step2: Views timeline <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Loads delivery details • Step2: Updates progress in real-time
Postconditions:	Delivery tracking completed.

View Request History

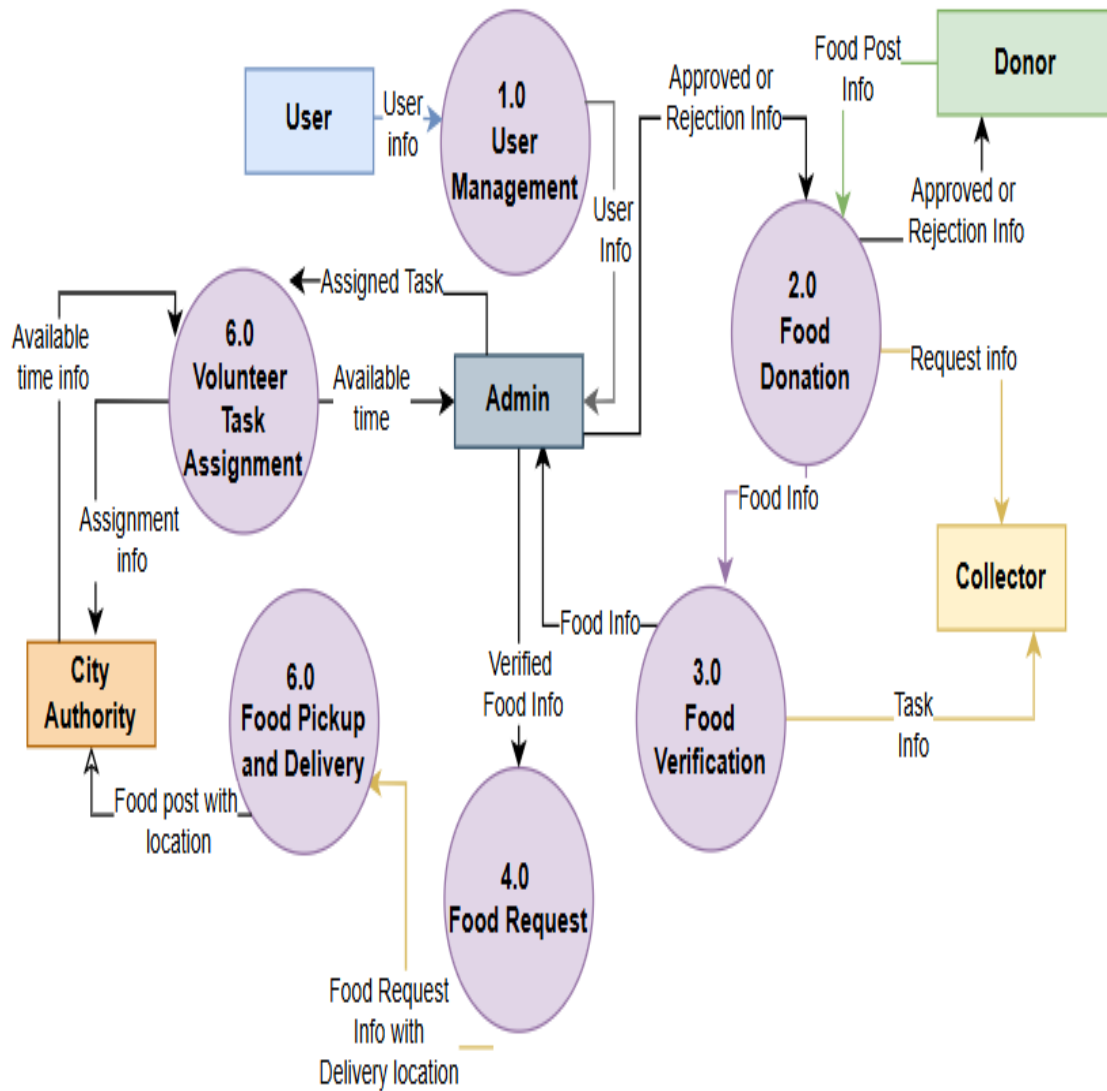
Actors:	Receiver
Description:	Receiver views previous food requests.
Preconditions:	Receiver has made requests.
Triggers:	Receiver selects Request History.
Typical Actions:	<p>Actor Actions:</p> <ul style="list-style-type: none"> • Step1: Opens history • Step2: Reviews entries <p>System Actions:</p> <ul style="list-style-type: none"> • Step1: Fetches request history • Step2: Displays final status
Alternate Actions	<ul style="list-style-type: none"> • If no previous requests exist → system displays “No Request History.”
Postconditions:	Request history displayed.

5.3 Data Flow Diagram

Context Level DFD

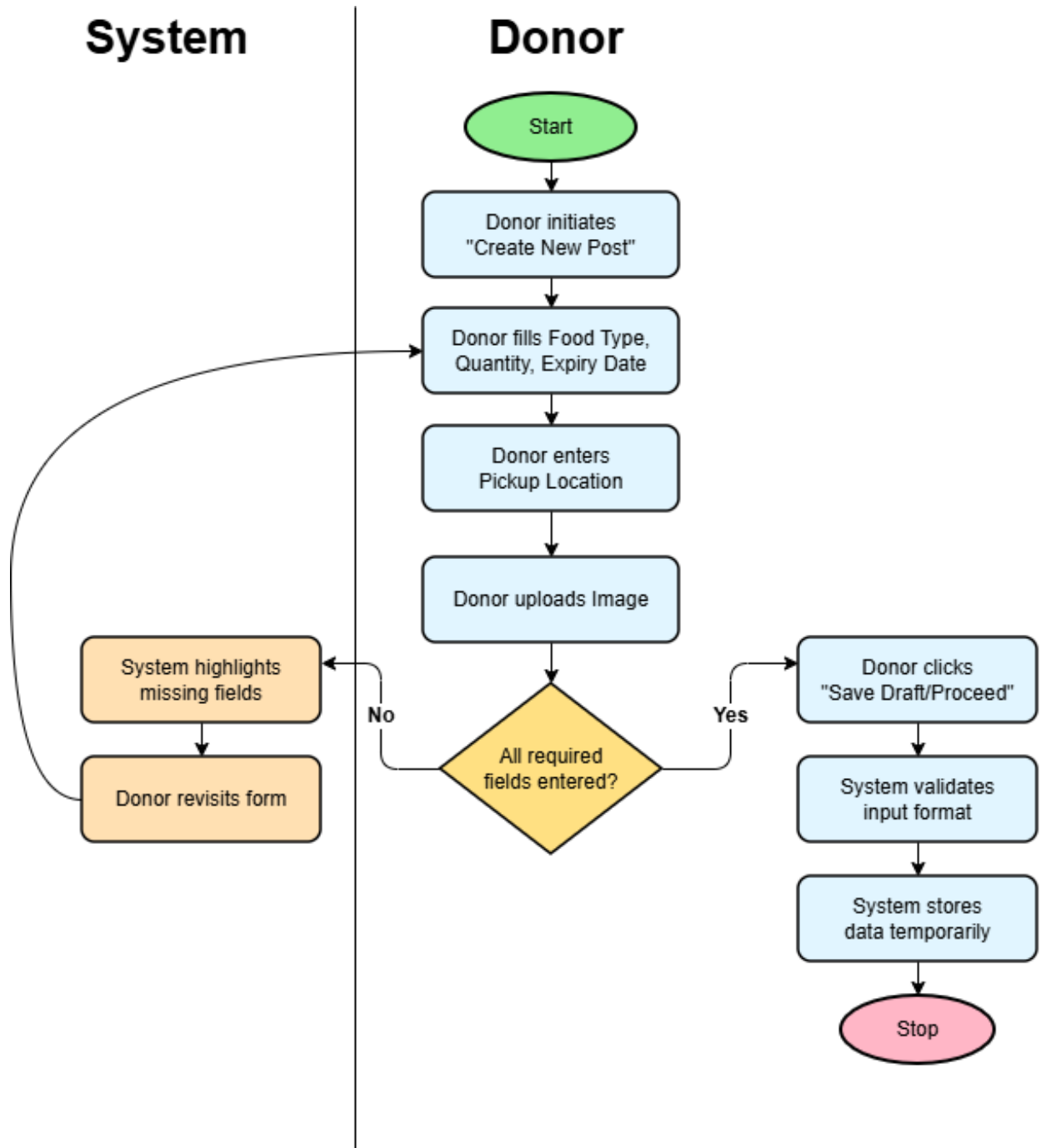


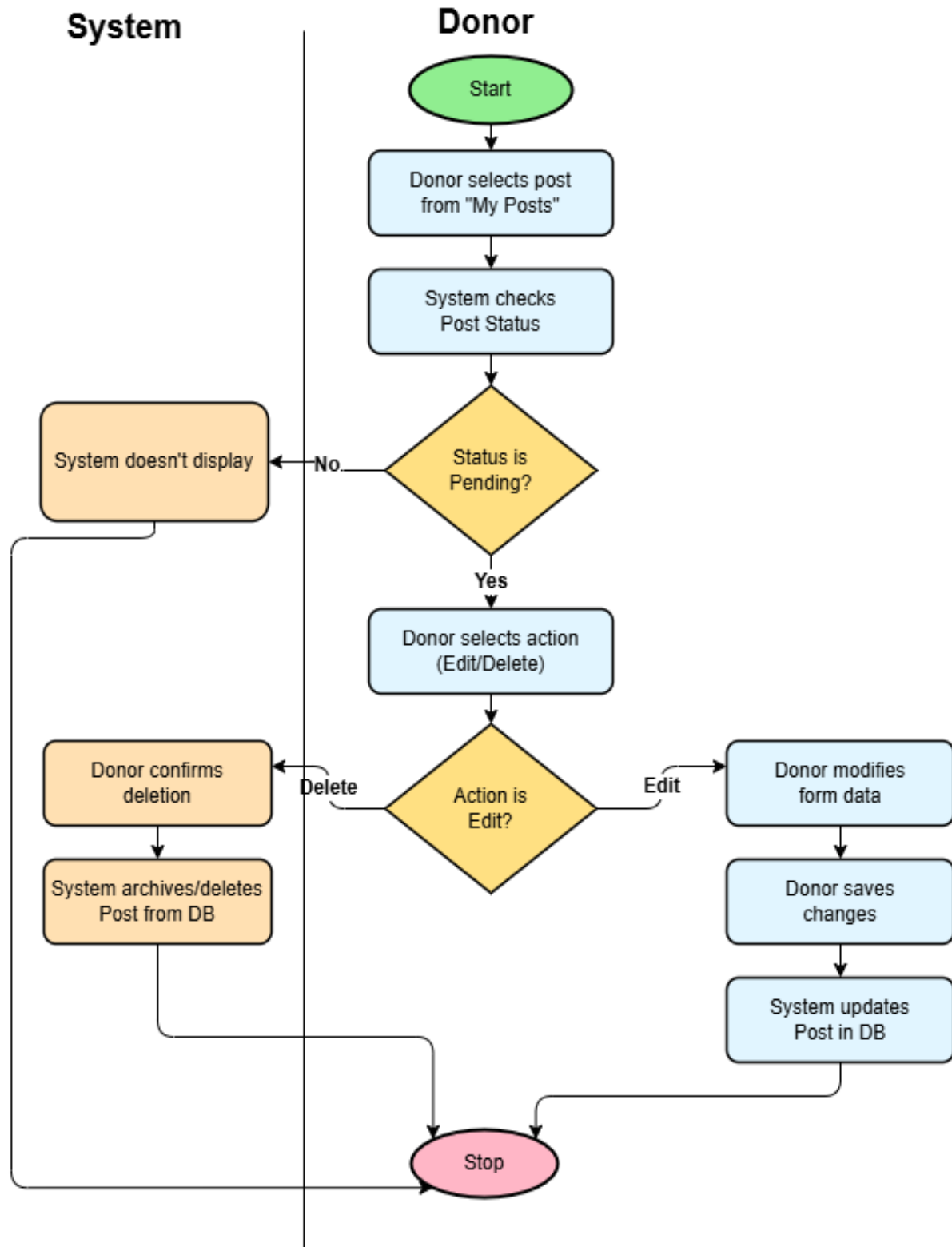
System Level DFD



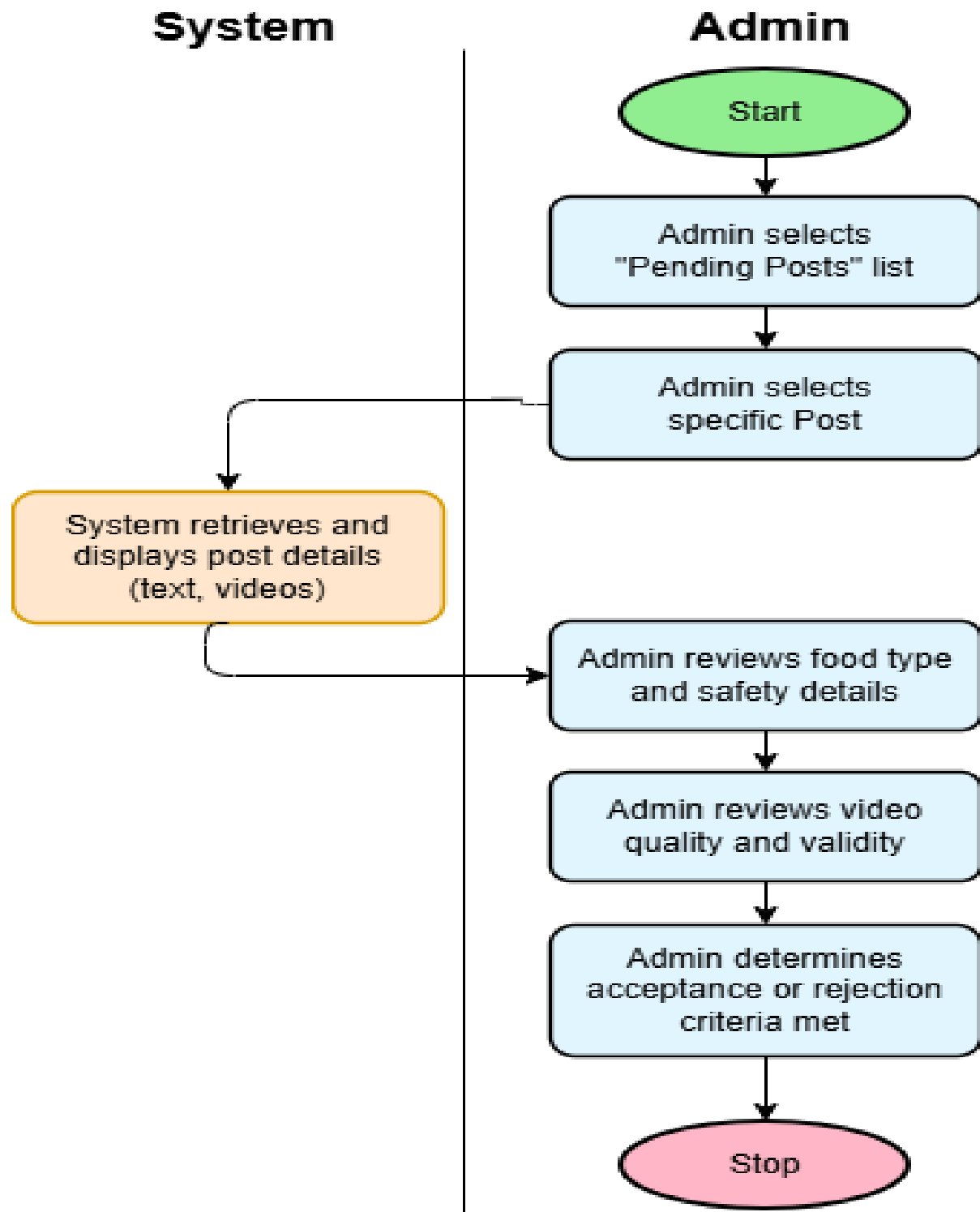
5.4 Activity Diagram

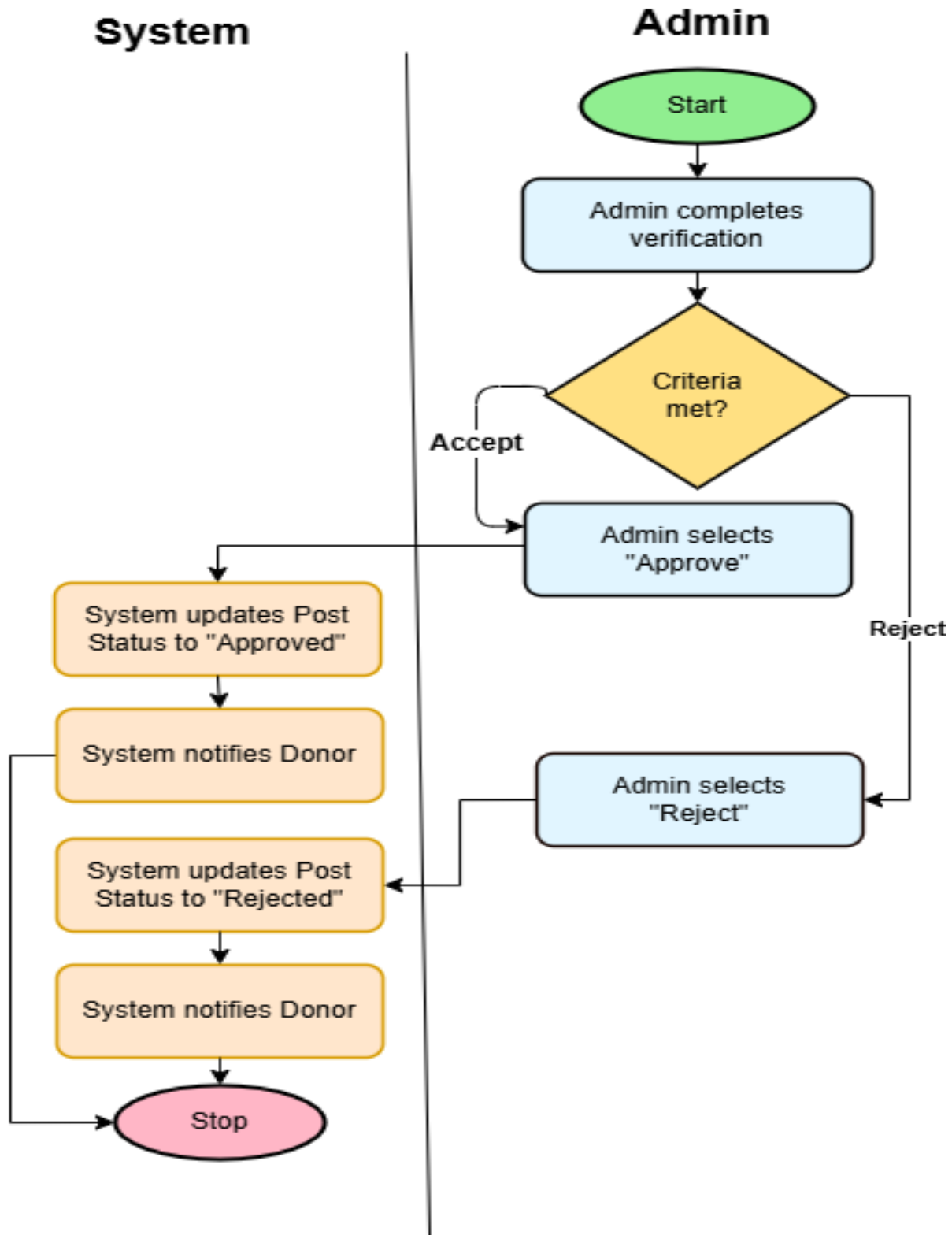
A. Donor Creates Post:

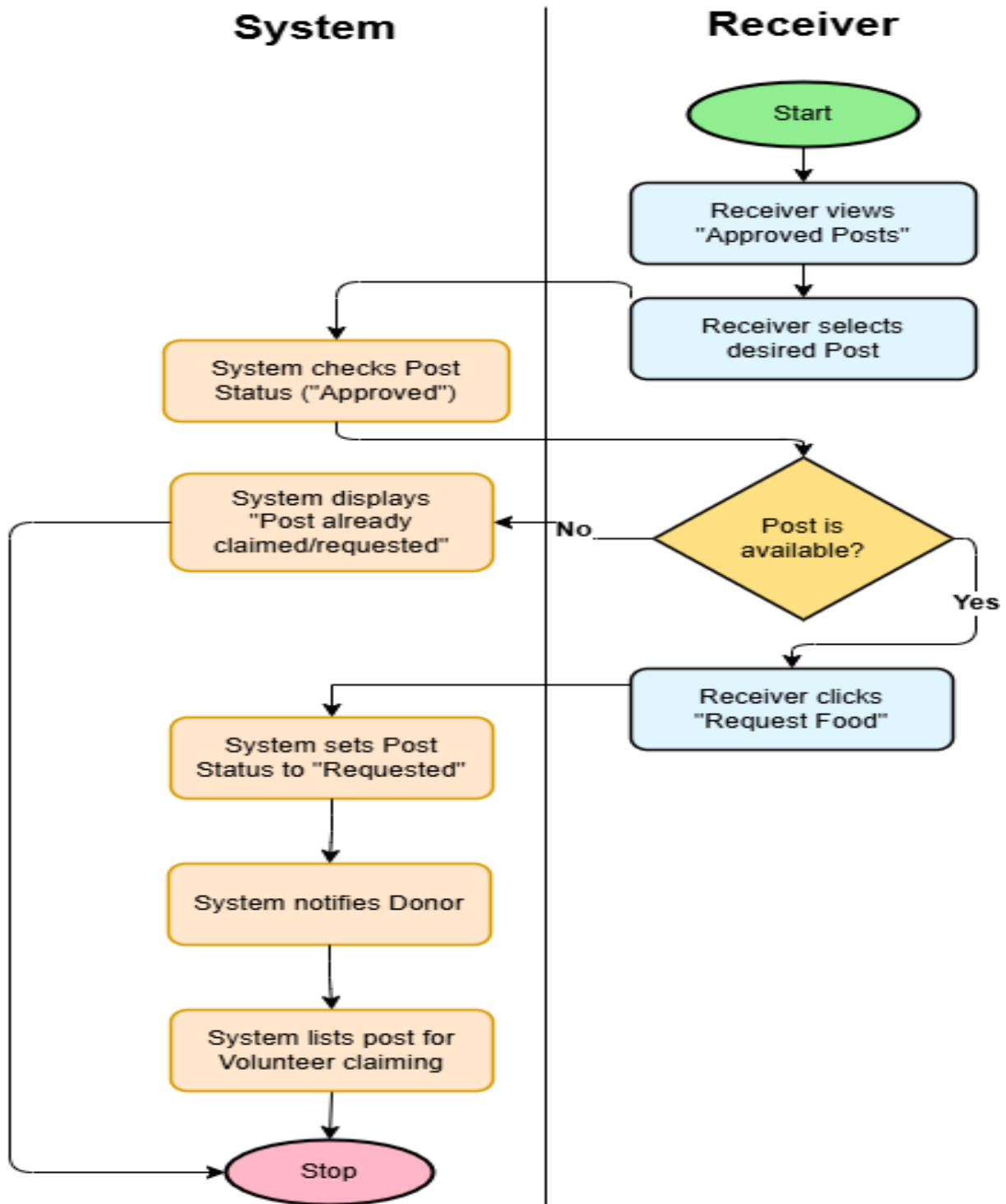


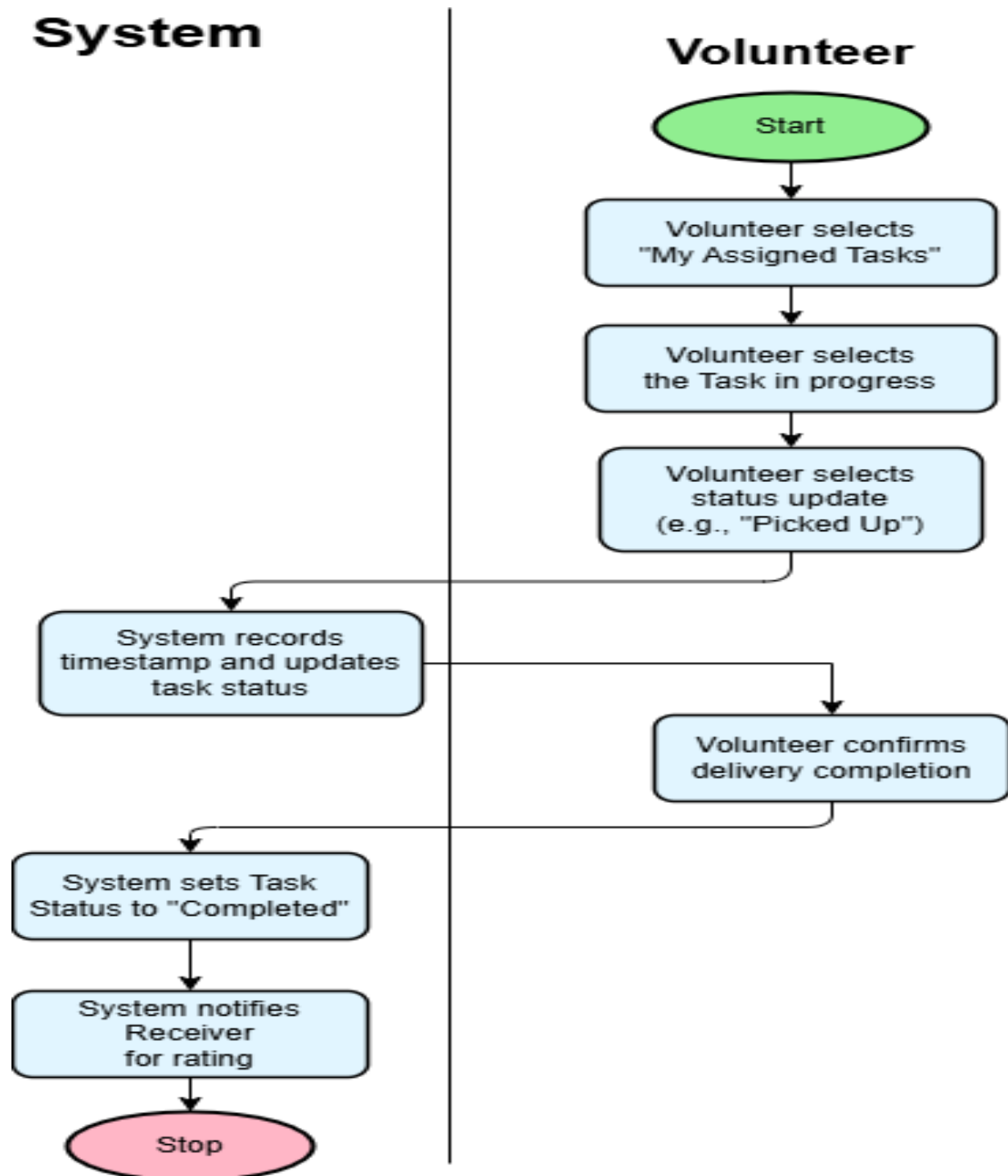
B.DonorEditPost:

C. Admin Verify Post:

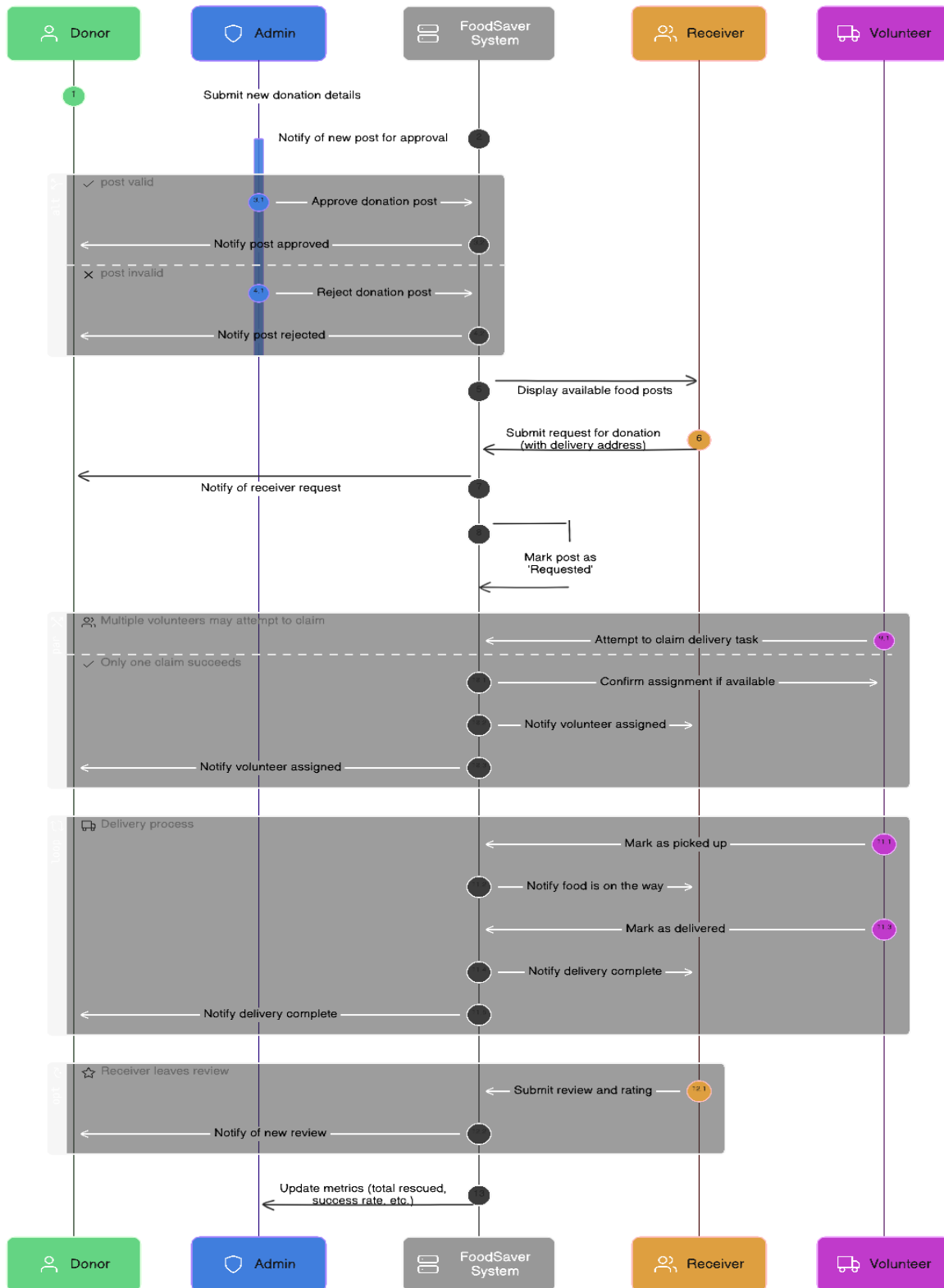


D.AdminApprove/RejectPost:

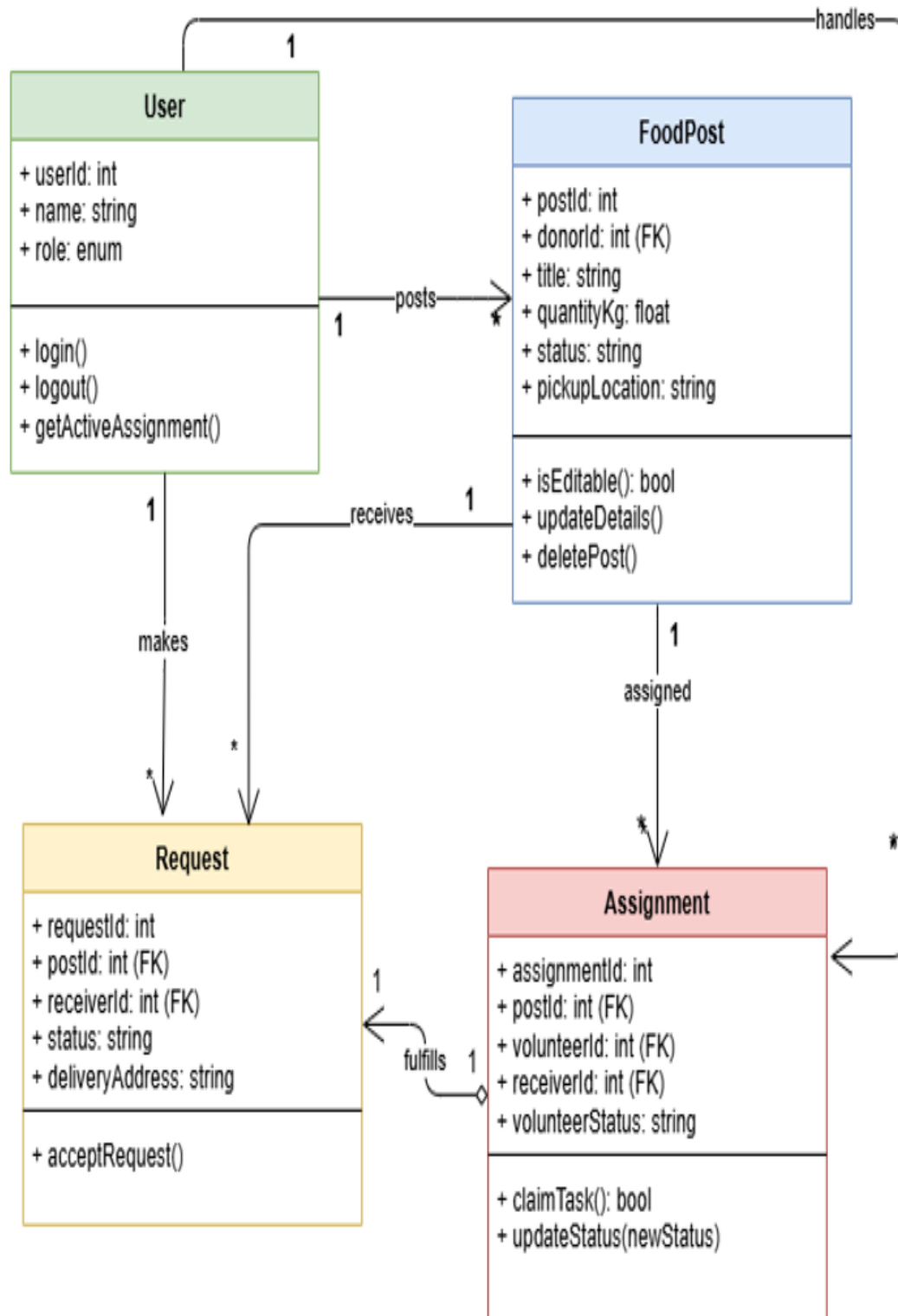
E. Receiver Request for Posted Food:

F.Volunteer Update Assigned Task Status:

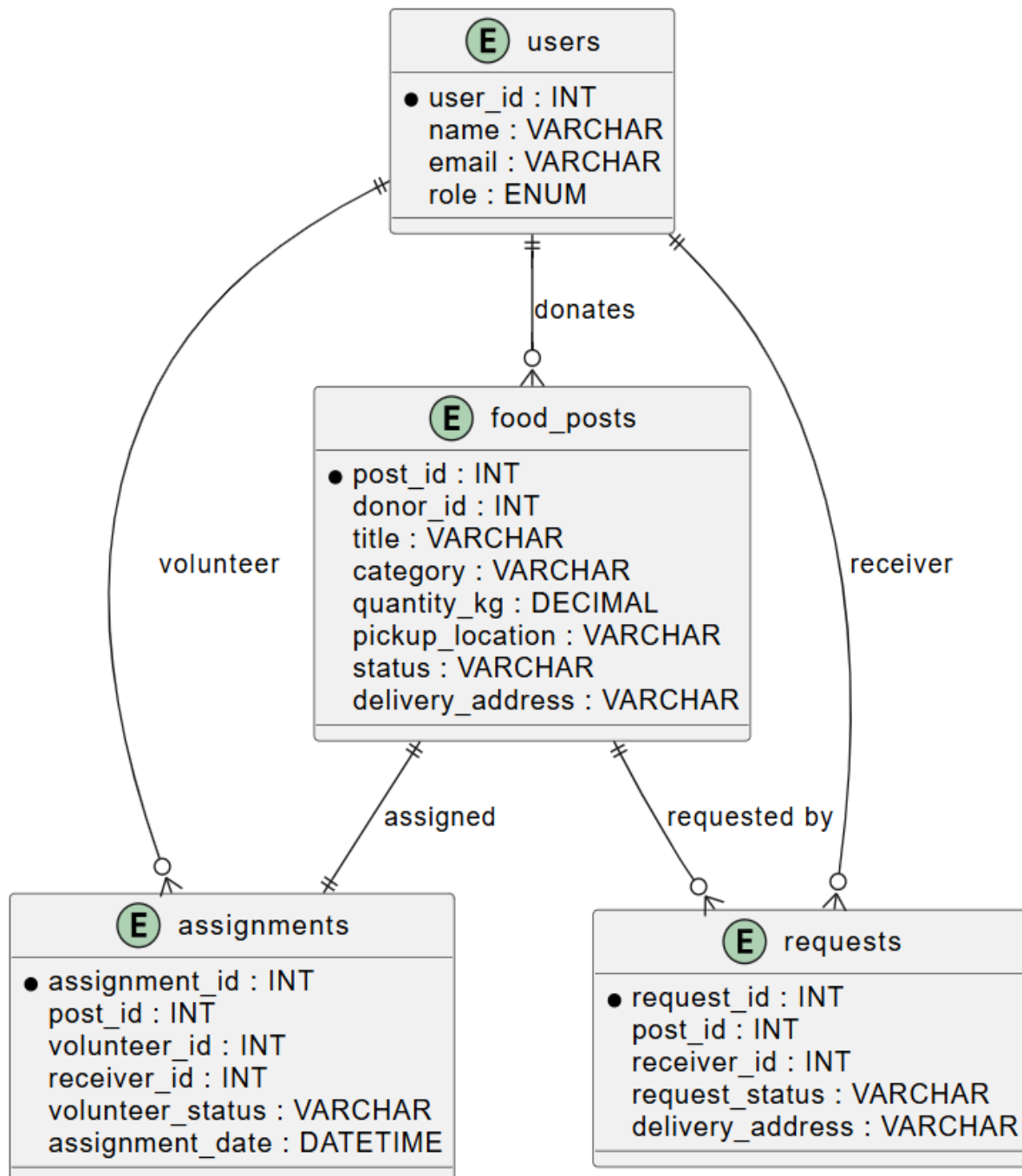
5.5 Sequence Diagram



5.6 Class Diagram

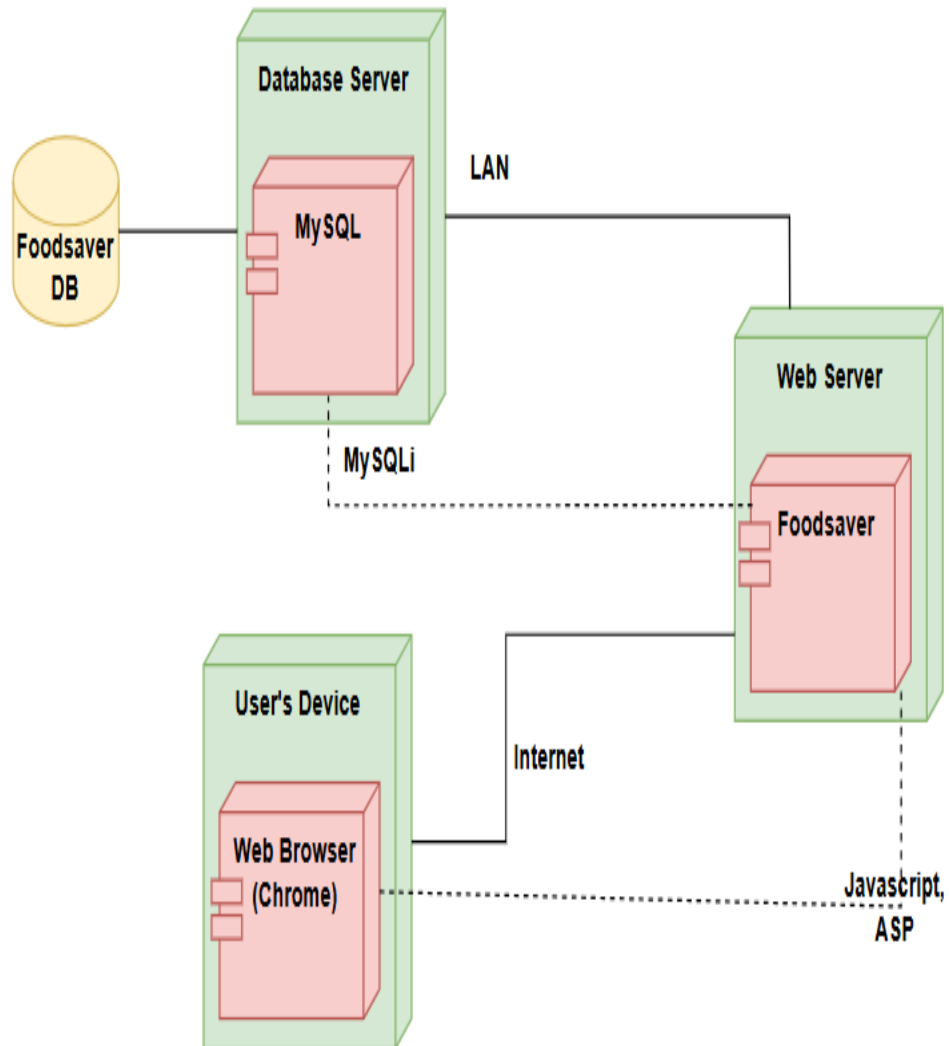


5.7 Entity Relationship Diagram



5.8 Deployment Diagram

Deployment Diagram



Chapter 06: Non-Functional Requirements

Non-Functional Requirements represent the quality attributes, constraints, and performance standards with which the FoodSaver Web Application must comply to ensure reliability, usability, and long-term sustainability. These requirements are essential in establishing the acceptance criteria that ensure that the system meets both user expectations and industrial benchmarks.

6.1 Performance Requirements

- The system shall support, at least, 1,000 concurrent users-both donors and receivers-together with volunteers and admins-without compromising the performance.
- Page load time should be no more than 3 seconds at average traffic and up to 5 seconds at peak loads.
- Database transactions-all CRUD operations on Donation Posts, Requests, and Assignments-should complete within 2 seconds.
- This system should efficiently handle at least 5,000 donation records, 10,000 delivery records, and 20,000 request logs.
- The alert system should send notifications such as task assignments, approval updates, delivery status, etc. within 5 seconds of the event being triggered.
- System response time in user authentication must not take more than 2 seconds.
- The backend should ensure 99.5% uptime excluding scheduled maintenance.

6.2 Reliability Requirements

- The system shall guarantee zero data loss, by doing periodical database backups every 24 hours.
- All critical operations like donation posting, request approval, and delivery updates must be transaction-safe with appropriate rollback mechanisms in case of failure.
- The server must handle failures gracefully using automatic recovery and redundancy features.
- The system should process correctly any sequence of updates of the status of volunteers, while maintaining consistent delivery records.
- Error logs should be generated automatically for transactions that have failed, and securely stored for 90 days.
- The MTBF (Mean Time Between Failures) target of the system is at least 720 hours or 30 days.

6.3 Security Requirements

- Passwords must be securely hashed for donor, volunteer, receiver, and admin user accounts.
- Session management should be strictly followed with secure cookie and session timeout rules.
- Role-based access control shall prevent unauthorized activity:
- Volunteers cannot directly edit donation posts.
- Receivers cannot claim tasks.
- Donors cannot approve posts.
- All APIs need to be secured against:
- SQL Injection
- Man-in-the-middle attacks
- Sensitive data include passwords, volunteer location, delivery address, and shall be transmitted in encrypted form using HTTPS.
- Two-factor authentication is required for admin users.

- Failed login attempts should be locked after 5 tries.

6.4 Usability Requirements

- This system should have a clean user interface based on accessibility of users.
- All major actions should not require more than 3 clicks: create post, request food, claim task, update delivery.
- Forms should contain validation messages and tool tips.
- Colors, icons, and layout should be user-friendly and responsive for all devices.
- Volunteers should be able to view tasks and update statuses with minimum scrolling.
- There should be a consistent navigation sidebar on every dashboard.
- Receivers should have the ability to check the real-time status of delivery from a simple interface.

6.5 Maintainability Requirements

- The system shall be designed to be modular, using the MVC architecture.
- The system should feature developer documentation for each module.
- All modules, such as donor, receiver, volunteer, admin, should be independently updatable.
- Bug fixes must be deployed without downtime.
- Comments in code are necessary for readability.
- The database schema should be normalized at least to 3NF.
- Versions must be controlled, with the proper branching strategy on Git.

6.6 Scalability Requirements

- The back-end should support horizontal scaling (more servers) and vertical scaling (more RAM/ CPU).
- The database should be able to be migrated to cloud services such as AWS RDS or Azure SQL without redesign.
- The notification module should be extensible to SMS, email, or mobile push notifications.
- System shall allow the addition of new actor types in the future.
- It should allow for future integration with food safety IoT devices such as temperature sensors.

6.7 Interoperability Requirements

- System should provide support for integrating APIs of delivery services or NGO management systems.
- Exporting data should allow CSV, Excel, and JSON formats.
- The platform needs to be integrated with the Google Maps API to route volunteers.
- System should follow REST API standards for future integrations with mobile apps.

6.8 Legal & Ethical Requirements

- All food safety guidelines shall be based on Bangladesh National Food Safety Standards.
- Personal data must follow privacy compliance similar to GDPR guidelines.
- All donors must agree to a disclaimer stating that donated food is safe and handled properly.

- Volunteers cannot pass receiver information to any third party.

6.9 Availability Requirements

- The system shall be available 24/7 unless scheduled maintenance is announced 24 hours in advance.
- Crash recovery system shall restore services within 15 minutes.
- Admin users should be able to temporarily disable new submissions in case of system updates.

6.10 Portability Requirements

The application should work seamlessly on:

- Chrome
- Firefox
- Safari
- Mobile browsers

The backend must be deployable on:

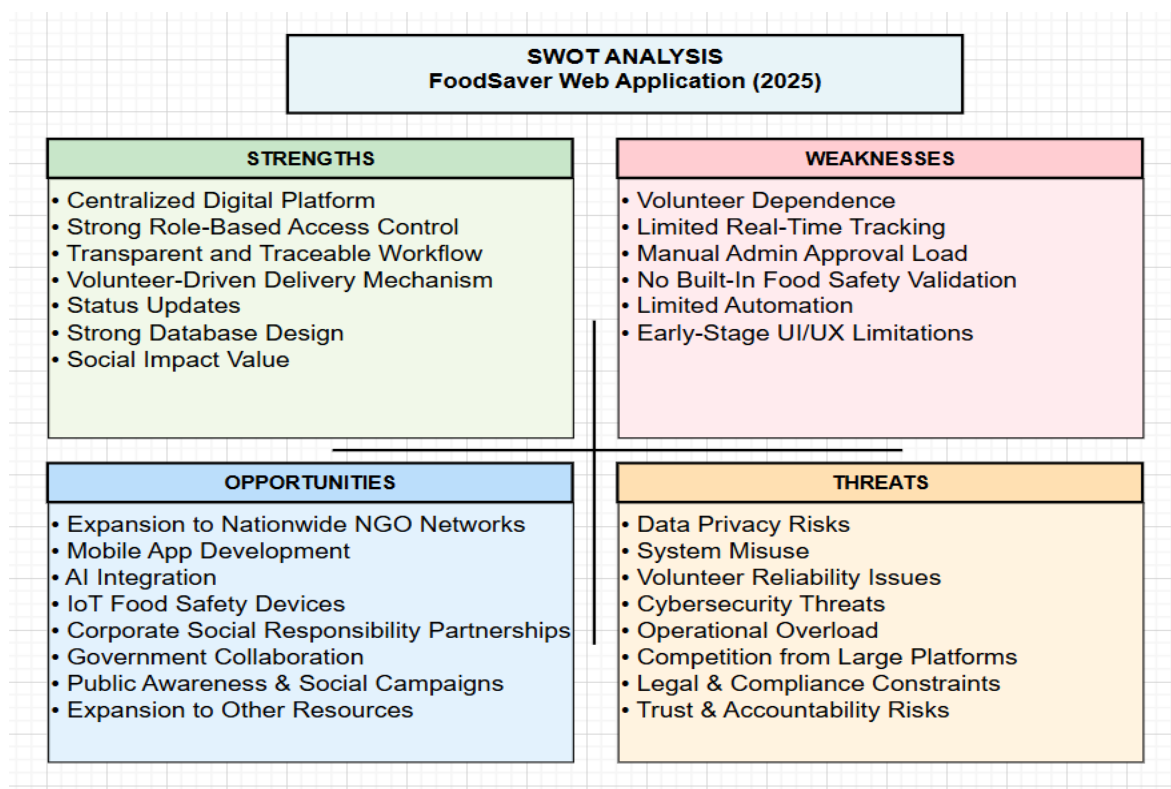
- Linux
- Windows Server

Database portability:

It should allow easy migration from MySQL to PostgreSQL with minimal changes.

Chapter 07: Risk Analysis

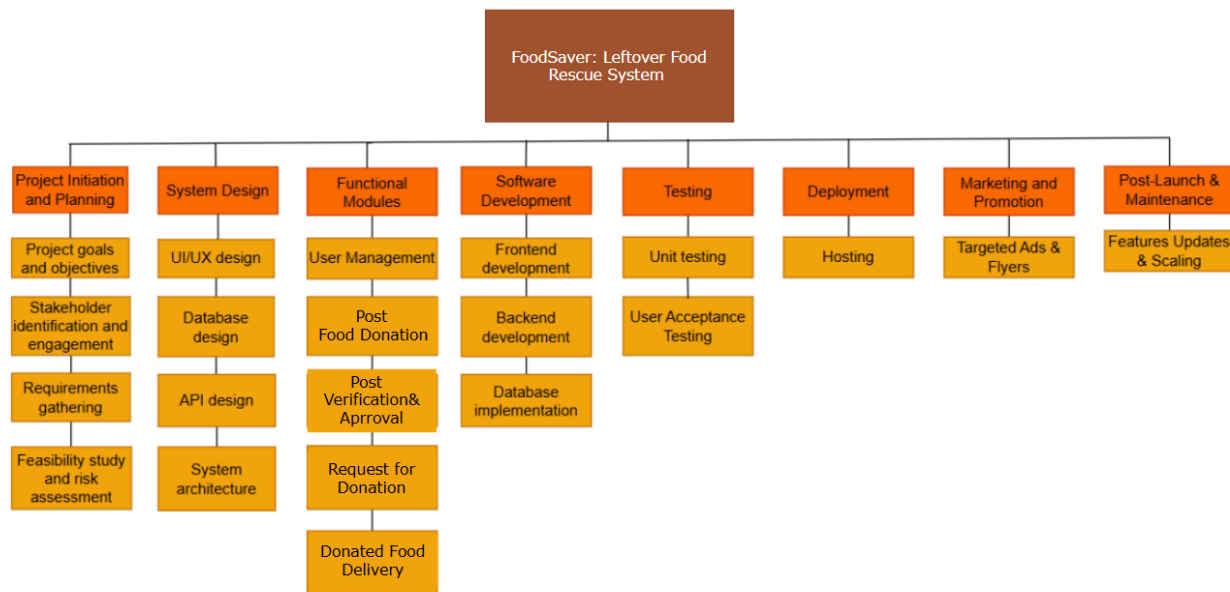
7.1 SWOT Analysis



The FoodSaver Web Application operates in an environment that is socially impacting but operationally complex. Its strengths arise from structured workflows, transparency, and community collaboration, while its weaknesses entail dependency on volunteers and manual administration. The strong opportunities include government partnerships, NGO expansion, and technological upgrades, such as AI and IoT. Still, the system must be very proactive in managing privacy, operational, and cybersecurity threats to maintain trust and reliability.

Chapter 08: Project Management

8.1 Work Breakdown Structure (WBS)



FoodSaver, the leftover food rescue system starts by defining project goals, identifying stakeholders, and gathering system requirements. A feasibility study assesses risks and checks if the project is viable. During the design phase, the UI/UX layout is created, system architecture is planned, and the database structure is set up to support features like users, food post, requests, assignments. Next is software development. The front-end interface and back-end logic are created. Key modules are built, such as user management, post food donation, verification and approval, request for donation, delivery, history and tracking, rating and analytics. Once development is complete, all components are integrated and tested to confirm the system works properly.

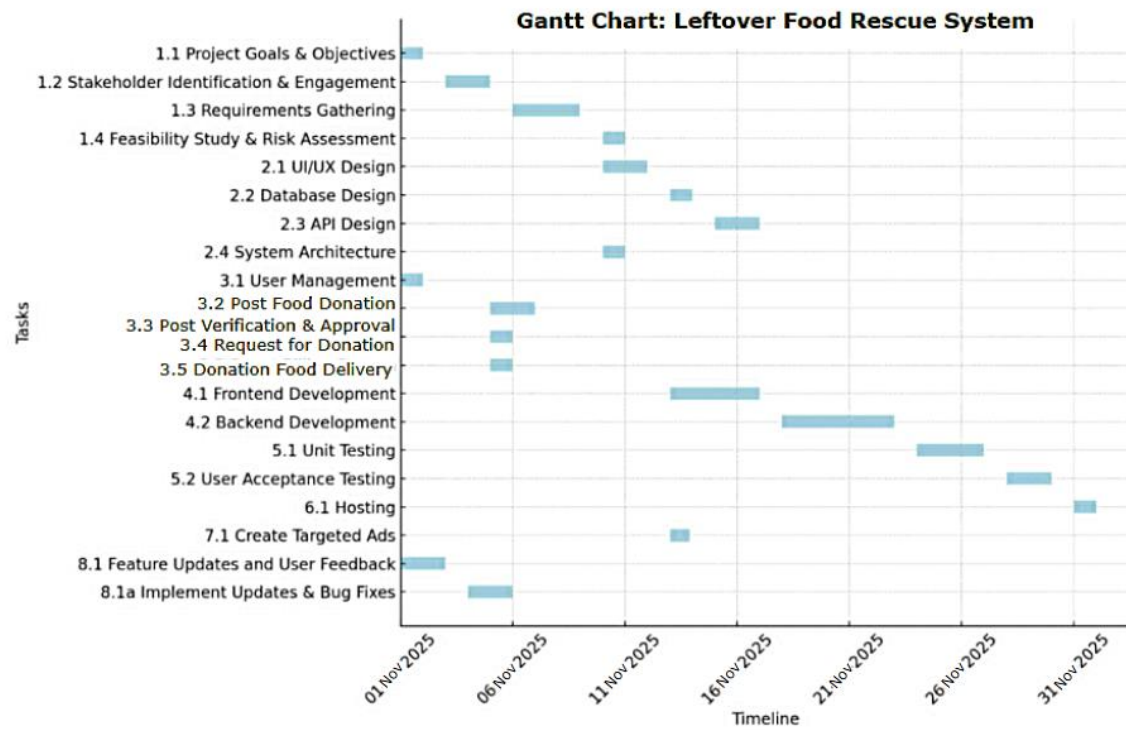
8.2Activity List

SI	Activity	Duration (days)	Dependencies	Resources	Cost (BDT)
1	Project Initiation and Planning				
1.1	Project goals and objectives	2		Project Manager	24,000
1.2	Stakeholder identification and engagement	3	1.1	Project Manager, Analyst	48,000
1.3	Requirements gathering	4	1.2	Analyst, Developer	60,000
1.4	Feasibility study and risk assessment	2	1.3	Project Manager	30,000
2	System Design				
2.1	UI/UX design	3	1.3	UI Designer	36,000
2.2	Database design	2	2.1	Database Engineer	36,000

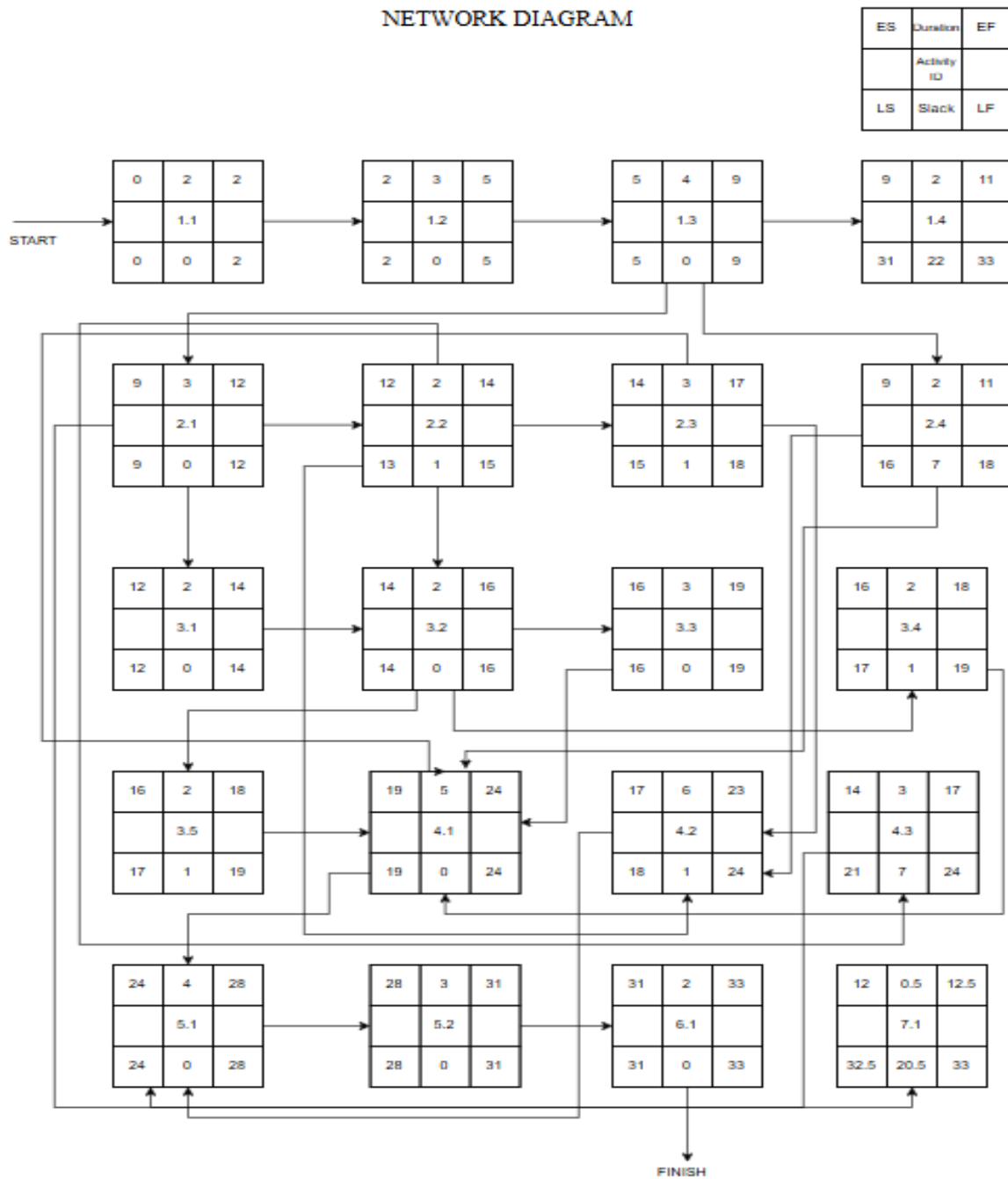
2.3	API design	3	2.2	Database Engineer	30,000
2.4	System architecture	2	1.3	System Architect	36,000
3	Functional Modules				
3.1	User Management	2		Developer	30,000
3.2	Post Food Donation	2	3.1	Developer	30,000
3.3	Post Verification & Approval	3	3.2	Developer	36,000
3.4	Request for Donation	2	3.2	Developer	30,000
3.5	Donated Food Delivery	2	3.2	Developer	30,000
4	Software	2			
4.1	Frontend development	5	2,3	Frontend Developer	72,000
4.2	Backend development	6	2.2,2.3,2.4	Backend Developer	84,000
4.3	Database implementation	3	2.2	Database Engineer	48,000
5	Testing				
5.1	Unit testing	4	4.1-4.5	Developer	48,000

5.2	User Acceptance Testing	3	5.1	Developer	48,000
6	Deployment				
6.1	Hosting	2	5.2	DevOps Engineer	36,000
7	Marketing and Promotion				
7.1	Create targeted ads (urban: social media, rural: flyers)	0.5	2.1	Marketing Managers	25,000 (Social Media & Flyers)
8	Post-Launch and Maintenance				
8.1	Feature Updates and User Feedback				
8.1a	Implement updates & bug fixes based on feedback	Ongoing	User Feedback	Customer Support	20,000 (Contingency for Scaling & SMS Costs)

8.3 Gantt Chart



8.4 Network Diagram



Chapter 09: Feasibility Analysis

9.1 Expense Head to Implement the Project

Software Development	Frontend and backend design, coding, API integration	6,00,000
Domain & Hosting	Web hosting, domain name (1 year)	1,20,000/year
Database & Cloud Storage	MySQL / MongoDB with cloud access	1,00,000/year
UI/UX Design Tools	Use of Canva, Figma, prototyping tools	70,000
Testing & Debugging	Manual testing, browser/device testing	90,000
Documentation & Reporting	User manual, admin manual, team report printing	45,000
Project Promotion & Awareness	Posters, social media mock promotion, demo site	1,30,000
Communication Tools	Team communication tools (Discord/Slack)	20,000
Miscellaneous	Backup drives, stationary, snacks during development	1,10,000
Total Estimated Cost		BDT 12,85,000

9.2 Assumptions

- **Subscription Fee:** BDT 200 per year (per individual)
- **Subscriber Base:**

Year 1: 4,000 subscribers

Year 2: 6,000 subscribers

Year 3: 10,000 subscribers

Year 4: 15,000 subscribers

Year 5: 30,000 subscribers

- **Calculation:**

Revenue = Subscription Fee × Number of Subscribers

- **Estimated Revenue:** (Estimate each year subscribers increase up to 5 years)

Year 1: $4,000 \times 200 = 8,00,000$ BDT

Year 2: $6,000 \times 200 = 12,00,000$ BDT

Year 3: $10,000 \times 200 = 20,00,000$ BDT

Year 4: $15,000 \times 200 = 30,00,000$ BDT

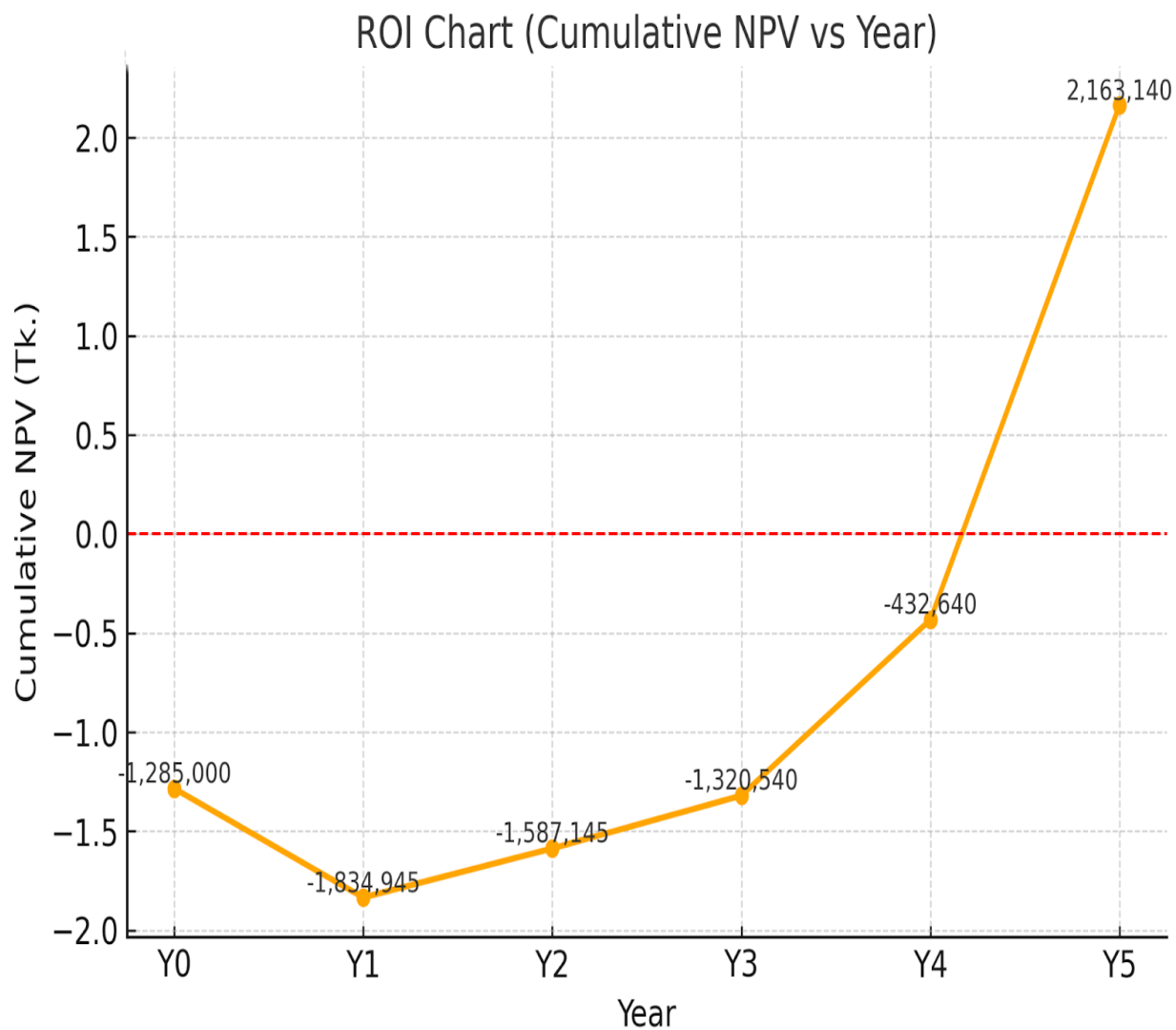
Year 5: $30,000 \times 200 = 60,00,000$ BDT

9.3 Net Present Value (NPV) Table

Year	Total Cost	Total Benefit	Net Flow	Discount Factor	Present Value
Y0	12,85,000	0	-12,85,000	1.000	-12,85,000
Y1	14,05,000	8,00,000	-6,05,000	0.909	-5,49,945
Y2	15,25,000	12,00,000	-3,00,000	0.826	2,47,800
Y3	16,45,000	20,00,000	3,55,000	0.751	2,66,605
Y4	17,00,000	30,00,000	13,00,000	0.683	8,87,900
Y5	18,20,000	60,00,000	41,80,000	0.621	25,95,780
Total					21,63,140

9.4 Return on Investment (ROI) Chart

The following chart shows the ROI based on cumulative net returns from the project over a 6-year period.



Chapter 10: Conclusion

The FoodSaver Web Application is a revolutionary solution to one of the most persistent and overlooked problems in modern society: food waste and unequal food distribution. With an enormous amount of edible surplus food produced every day in households, events, restaurants, and institutions, it has not been easy, in the absence of an organized and technology-aided approach, to distribute the surplus to people and communities with food insecurity. FoodSaver directly addresses this gap by offering a structured, transparent, and reliable digital platform that connects donors, volunteers, receivers, and administrators into one ecosystem.

This analyzed the application from different viewpoints throughout: operational, technical, functional, non-functional, economic, and organizational. The system's functional design thus supports every crucial activity: posting of donations, verification of posts, request for food, assigning delivery tasks, real-time tracking, and maintenance of complete historical records. Clearly defined user roles assure proper distribution of responsibility and secure interactions. The involvement of volunteers introduces a community-driven logistics network that substantially reduces operation costs while raising social participation.

From a technical viewpoint, the platform will be designed using widely adopted and tested technologies that ensure maintainability, scalability, and long-term sustainability. The security and non-functional requirements in this SRS have been crafted to ensure that the system can support massive user traffic without the integrity of the data, reliability of the system, and privacy of users being compromised. The feasibility study proves that FoodSaver will be not only technically viable but economically viable and operationally effective, with social benefits that considerably surpass development and maintenance costs.

In conclusion, the FoodSaver Web Application is not only a technological project but a socially responsible digital ecosystem for minimizing food waste, engaging the community, and enhancing access to food by the most vulnerable social groups. Its well-structured

workflow, modern architecture, and high scalability potential make it perfectly suitable for long-term deployment, expansion, and integration with welfare systems. This SRS thus provides a holistic basis for successful development, implementation, and further enhancements of the FoodSaver platform.

Chapter 11: Reference

1. Bakharev, V. V., Mityashin, G. Y., & Stepanova, T. V. (2023). Food security, food waste and food sharing: The conceptual analysis. *Food systems*, 6(3), 390-396.
2. Reynolds, C., Goucher, L., Quested, T., Bromley, S., Gillick, S., Wells, V. K., ... & Jackson, P. (2019). Consumption-stage food waste reduction interventions—What works and how to design better interventions. *Food policy*, 83, 7-27.
3. Cane, M., & Parra, C. (2020). Digital platforms: mapping the territory of new technologies to fight food waste. *British Food Journal*, 122(5), 1647-1669.
4. Chauhan, Y. (2020). Food waste management with technological platforms: Evidence from Indian food supply chains. *Sustainability*, 12(19), 8162.
5. Ciaghi, A., & Villafiorita, A. (2016, September). Beyond food sharing: Supporting food waste reduction with ICTs. In *2016 IEEE International Smart Cities Conference (ISC2)* (pp. 1-6). IEEE.