B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

OOMD Mini Project Report

Food Donation and Redistribution App

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:
Bhavya Goyal(1BM23CS063)
Harbakhshish Singh Arora(1BM23CS104)
Biswajeet Behera(1BM23CS069)
Bala K(1BM23CS062)

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2025-26

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, Bhavya Goyal(1BM23CS063), Harbakhshish Singh(1BM23CS104),Biswajeet Behera(1BM23CS069), Bala K(1BM23CS062) students of 5th Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this OOMD Mini Project entitled "Food Donation and Redistribution App" has been carried out in Department of CSE, B.M.S. College of Engineering, Bangalore during the academic semester August 2025- December 2025. I also declare that to the best of our knowledge and belief, the OOMD mini Project report is not from part of any other report by any other students.

Signature of the Candidate

Bhavya Goyal(1BM23CS063)
Harbakhshish Singh Arora(1BM23CS104)
Biswajeet Behera(1BM23CS069)
Bala K(1BM23CS062)

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOMD Mini Project titled "Food Donation and Redistribution App" has been carried out by Bhavya Goyal(1BM23CS063), Harbakhshish Singh(1BM23CS104),Biswajeet Behera(1BM23CS069), Bala K(1BM23CS062) during the academic year 2025-2026.

Signature of the Faculty in Charge

Dr Roopashree S
Associate Professor
Department of Computer Science and Engineering B.M.S.

Table of Contents

Chapter 1: Problem Statement

Food waste is a significant issue in urban areas, where large quantities of edible food from households, restaurants, and events are discarded daily. At the same time, millions of people struggle with hunger and lack access to nutritious meals. The absence of an efficient, transparent, and real-time system to connect food donors with NGOs, food banks, and volunteers results in missed opportunities for redistribution and contributes to unnecessary food wastage.

Currently, food donation processes are mostly manual, unorganized, and location-dependent. Donors often do not know *where* to donate, NGOs are unaware of *available food donations*, and volunteers lack proper coordination for pickup and delivery. Issues such as delayed communication, lack of tracking, and limited visibility further reduce the effectiveness of donation networks.

Therefore, there is a need for a centralized digital platform that can seamlessly connect donors, NGOs, and volunteers, automate communication, provide real-time location-based matching, and ensure transparency throughout the donation lifecycle.

The *Food Donation and Redistribution App* aims to solve this problem by enabling users to easily donate food, helping NGOs efficiently manage donations, and allowing volunteers to coordinate pickups and deliveries through real-time tracking and notifications. This system will significantly reduce food wastage, streamline redistribution, and strengthen hunger-relief efforts in a sustainable and scalable manner.

Chapter 2: Software Requirement Specification

Food Donation and Redistribution App

Introduction

Purpose

The purpose of this document is to describe the functional and non-functional requirements of the Food Donation and Redistribution App. The app aims to connect food donors (restaurants, households, events, etc.) with NGOs, food banks, and individuals in need to reduce food wastage and support hunger relief.

Scope

The app will:

1. Allow users to donate excess food easily.
2. Notify nearby NGOs or volunteers for collection and redistribution.
3. Provide real-time tracking of donation pickups.
4. Maintain transparency between donors, NGOs, and recipients.
5. Encourage sustainability and social responsibility.

The system will include:

1. Donor module
2. NGO/Receiver module 3. Admin dashboard
4. Volunteer management

Definitions, Acronyms, and Abbreviations

Term Meaning:

1. NGO: Non-Governmental Organization
2. GPS: Global Positioning System
3. API: Application Programming Interface
4. UI: User Interface

References

1. IEEE SRS Standard (IEEE 830-1998)
2. FAO Food Waste Management Guidelines
3. Government Food Safety Regulations (India, FSSAI)

Overall Description

Product Perspective

The Food Donation App is an independent mobile application that interacts with:

Google Maps API for location tracking.

Cloud database (e.g., Firebase) for storing user and donation data.

Notification services for real-time updates.

Product Features

1. User Registration &amp; Authentication:
   - Donor, NGO, and Volunteer login using email/phone or social accounts.
2. Food Donation Request:
   - Donors can post details: food type, quantity, pickup location, and time.
3. Donation Matching:
   - System automatically matches donations with nearby NGOs/volunteers.
4. Tracking &amp; Notifications:
   - Real-time GPS tracking of pickups and delivery updates.
5. History &amp; Analytics:
   - View previous donations, impact statistics, and reports.

6. Admin Controls:

- Monitor user activities, verify NGOs, and manage reports.

User Classes and Characteristics

1. User Type Description Privileges
2. Donor Restaurant, event, or household donating food Create donation requests
3. NGO/Receiver Organization receiving food donations Accept and schedule pickups
4. Volunteer Person collecting and delivering donations Access assigned pickups
5. Admin System operator Manage users, donations, and reports

Operating Environment

- Platform: Android &amp; iOS
- Backend: Firebase / Node.js server
- Database: Firestore / MySQL
- APIs: Google Maps API, SMS/Email services

Design and Implementation Constraints

- Must comply with food safety standards.
- Internet connection required.
- User location permissions must be enabled.

Assumptions and Dependencies -

Donors provide edible food only.

- NGOs have capacity for redistribution.
- Volunteers are verified and trained.

System Features

Donor Module

Description: Enables users to donate food.

Functional Requirements:

FR1: Donor can create a donation post with details.

FR2: Donor can upload food images.

FR3: Donor receives confirmation once NGO accepts.

NGO/Receiver Module

Description: Receives and manages donation offers.

Functional Requirements:

FR4: NGO can view available donations nearby.

FR5: NGO can accept or decline requests.

FR6: NGO can mark donation as received.

Volunteer Module

Description: Handles transportation and delivery.

Functional Requirements:

FR7: Volunteer receives assigned pickups.

FR8: Volunteer updates status (“Picked Up”, “Delivered”).

Admin Module

Description: Controls and monitors the entire system.

Functional Requirements:

FR9: Admin verifies NGOs and volunteers.

FR10: Admin generates reports on donations and impact.

Non - Functional Requirements

Category Description

- Performance App should handle 1000 concurrent users with minimal lag.
- Security All data must be encrypted; authentication via secure API tokens.
- Usability Simple UI for non-technical users; multilingual support.
- Reliability 99% uptime with automatic data backup.
- Scalability Must support future expansion to multiple cities.
- Maintainability Modular code structure for easy updates.

System Design (High-Level Overview)

Architecture:

Client-side (Mobile App) ↔ Server-side (Backend API) ↔ Database ↔ Map/Notification Services

Data Flow:

1. Donor posts donation →
2. Server matches NGO/volunteer →
3. Volunteer pickup →
4. NGO confirms receipt →
5. Data stored and visible in dashboards.
6. Future Enhancements

AI-based food quantity estimation using image recognition.

Integration with government food waste reduction initiatives.

Donation reward points or certificates for donors.

Cold chain tracking for perishable items.

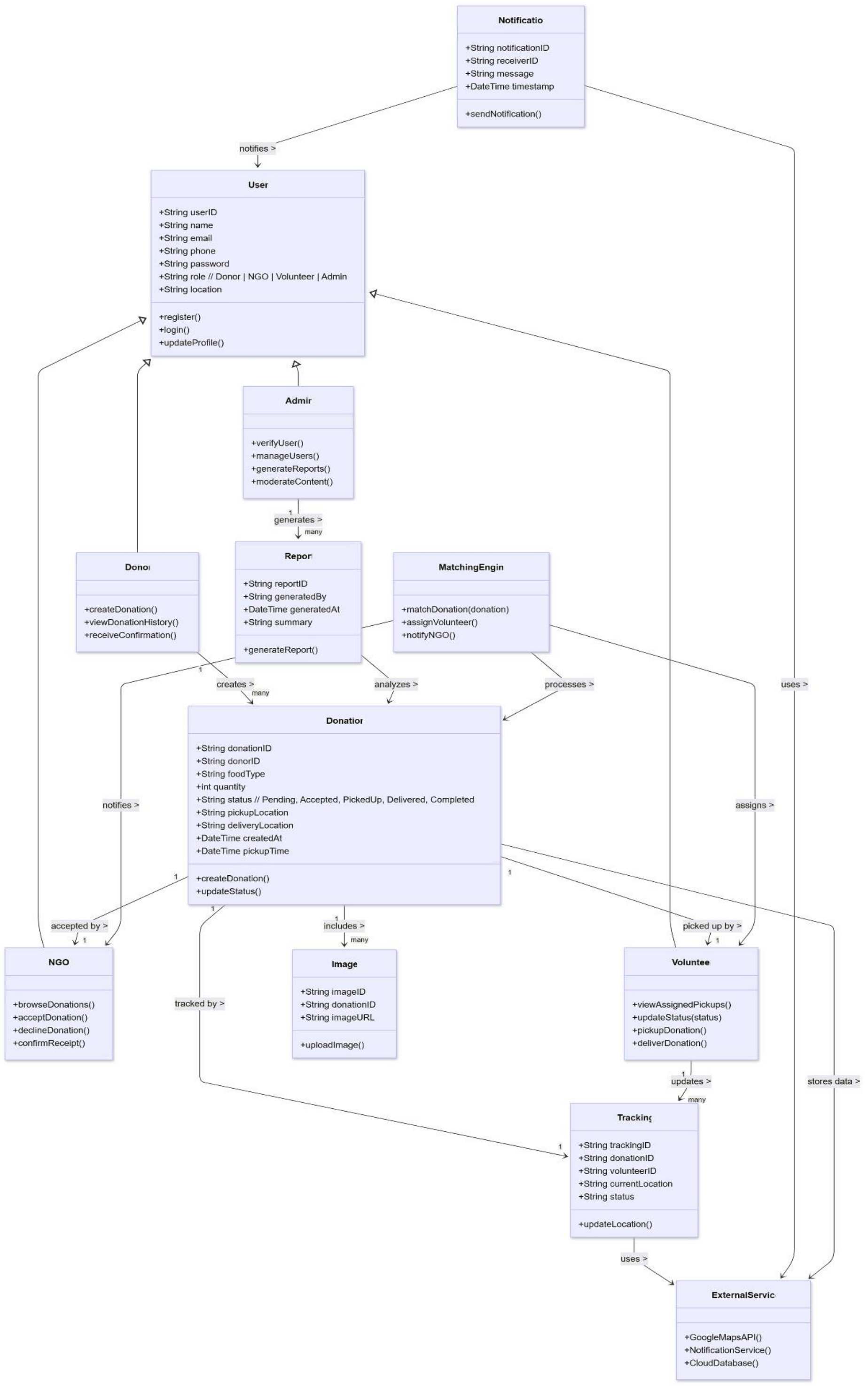
Appendix

Tools Used: Figma (UI Design), Firebase (Backend), Android Studio (Development)

Team Roles:

1. Project Manager
2. Developer
3. UI/UX Designer
4. Tester
5. NGO Liaison

Chapter 3: Class Modeling



1. User Class

Description & Purpose

The **User** class represents the parent class for all types of system users—Donors, NGOs, Volunteers, and Admins. It contains common attributes like name, email, phone, location, and credentials.

Why this class is important

- Centralizes all shared attributes and functions (register, login, updateProfile).
- Enables inheritance, reducing duplication across Donor/NGO/Volunteer/Admin.
- Provides a unified identity system for authentication and authorization.

2. Donor Class (inherits User)

Description

Represents individuals, households, restaurants, or events donating surplus food.

Relevance

- Holds operations related to donation creation and history tracking.
- Acts as the entry point for the donation process.

Key Methods

- createDonation()
- viewDonationHistory()
- receiveConfirmation()

3. NGO Class (inherits User)

Description

Represents food distribution organizations that accept and distribute food.

Relevance

-

  Critical for validating and accepting donated food.
- Provides decision-making (“accept” or “decline”).
- Confirms the final completion of donation delivery.

Key Methods

- browseDonations()
- acceptDonation()
- declineDonation()
- confirmReceipt()

4. Volunteer Class (inherits User)

Description

Responsible for the transport and delivery of food from donor to NGO.

Relevance

- Handles logistics and real-time movement of food.
- Works with tracking and notification systems.

Key Methods

- viewAssignedPickups()
- updateStatus(status)
- pickupDonation()
- deliverDonation()

5. Admin Class (inherits User)

Description

The system's top-level controller responsible for monitoring and maintaining the platform.

Relevance

- Ensures platform integrity and user verification.
- Generates system-wide analytical reports.

Key Methods

- verifyUser()
- manageUsers()
- generateReports()
- moderateContent()

6. Donation Class

Description

Represents a donation record created by a donor. Acts as the **central entity** connecting donors, NGOs, volunteers, images, and tracking.

Relevance

- Core object around which the entire workflow revolves.
- Stores essential data: type of food, quantity, status, timestamps, locations.
- Used by MatchingEngine to assign NGOs/volunteers.

●

Key Features

Maintains full lifecycle: Pending → Accepted → PickedUp → Delivered → Completed.

- Supports multiple images.
- Tracks pickup and delivery time.

7. Image Class

Description

Stores image metadata uploaded with the donation (e.g., food photos).

Relevance

- Enhances trust and transparency.
- Helps NGOs verify food quality before accepting.

Key Method

- uploadImage()

8. Tracking Class

Description

Manages real-time tracking of volunteer movement during pickup and delivery.

Relevance

- Provides live status of delivery progress.
- Enhances accountability and transparency.
- Useful for analytics and optimizing volunteer routes.

Key Attributes

- current_location
- status (on the way, picked up, delayed, delivered)

Key Method

- updateLocation()

9. Notification Class

Description

Handles creation and sending of system notifications.

Relevance

- Ensures all stakeholders are updated instantly.
- Used across donors, NGOs, volunteers, and admin.
- Sends messages like donation accepted, volunteer assigned, donation delivered, etc.

Key Method

- sendNotification()

10. Report Class

Description

Represents statistical and analytical reports generated by Admin.

Relevance

- Helps measure impact: total donations, food delivered, volunteer hours, etc.

Supports admin decision-making and external audits.

Key Method

- generateReport()

11. MatchingEngine Class

Description

An intelligent service responsible for processing new donations and assigning them to the best NGO or volunteer based on location, capacity, and availability.

Relevance

- Automates decision-making for the system.
- Implements logic such as nearest-NGO matching and load balancing among volunteers.
- Reduces manual intervention and speeds up response time.

Key Methods

- matchDonation(donation)
- assignVolunteer()
- notifyNGO()

Advanced Features Included

- Could use **GPS-based proximity algorithms**
- Could support **AI-based smart matching** using historical data
- Can optimize assignments based on traffic, volunteer workload, and NGO capacity

●

12. ExternalService Class

Description

Acts as an integration layer for all external APIs.

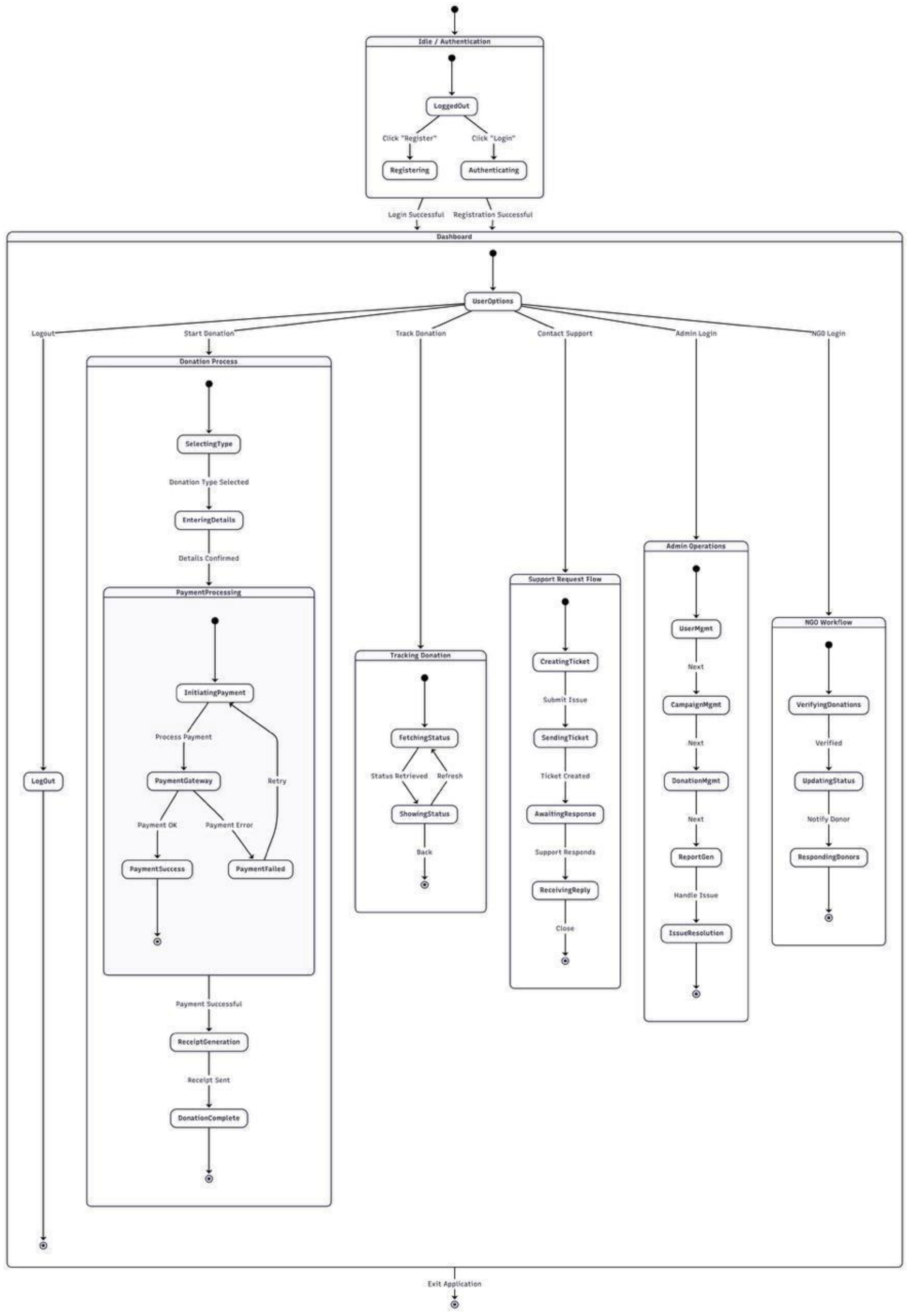
Relevance

- Improves modularity: API implementation stays separate from core logic.
- Supports Google Maps, Notification services, cloud storage, and more.

Key Services

- GoogleMapsAPI()
- NotificationService()
- CloudDatabase()

Chapter 4: State Modeling



1. Idle / Authentication Super-State

States

1. **LoggedOut**
    - **Meaning:** User has not yet authenticated; only login/register options are visible.
    - **Relevance:** This is the *entry point* of the system and enforces that no secure features (donation, tracking, admin, NGO work) are used without authentication.
2. **Registering**
    - **Meaning:** User is currently filling registration details and submitting them.
    - **Relevance:** Handles validation, account creation, role selection (Donor/NGO/Volunteer/Admin). It enables future features like role-based dashboards.
3. **Authenticating**
    - **Meaning:** User has entered login credentials and the system is verifying them.
    - **Relevance:** Protects access to dashboards and data; can include advanced checks like OTP, multi-factor auth, or account lockout on repeated failures.

Events

- **Click "Register"**
    - **Triggers:** LoggedOut → Registering
    - **Relevance:** User initiates account creation; starts the registration flow.
- **Click "Login"**
    - **Triggers:** LoggedOut → Authenticating
    - **Relevance:** Starts credential verification and session creation.
- **Registration Successful**
    - **Triggers:** Registering → Dashboard/UserOptions

- ○

  Relevance: After user details are validated and stored, user is taken to the main app, showing a seamless onboarding.

- **Login Successful**
  - **Triggers:** Authenticating → Dashboard/UserOptions
  - **Relevance:** Only after successful authentication can the user access donation, tracking, admin, or NGO workflows.

2. Dashboard / UserOptions Super-State

This is the big central box: once logged in, the user is in **Dashboard / UserOptions**.

State: UserOptions (Dashboard)

- **Meaning:** Main menu/state where the user chooses what to do: start a donation, track a donation, contact support, perform admin/NGO tasks, or log out.
- **Relevance:** It acts as a **hub** connecting all major flows; supports different user roles via different options.

Events from Dashboard

- **Start Donation** → enters **DonationProcess**
- **Track Donation** → enters **Tracking Donation**
- **Contact Support** → enters **Support Request Flow**
- **Admin Login** → enters **Admin Operations**
- **NGO Login** → enters **NGO Workflow**
- **Logout** → goes back to **LoggedOut**

- **Relevance:** These events represent **role-based navigation** and enable different sub-flows without re-authenticating.

3. Donation Process Super-State

States

1. **SelectingType**
   - **Meaning:** Donor chooses type/category of donation (e.g., cooked food, packaged food, monetary donation).
   - **Relevance:** Allows categorization and later optimization in MatchingEngine/NGO handling.
2. **EnteringDetails**
   - **Meaning:** Donor fills in quantity, pickup location, timing, images, etc.
   - **Relevance:** Captures all necessary metadata for matching NGOs/volunteers and tracking.
3. **PaymentProcessing (sub-state)**
   Advanced feature for **monetary donations** or service fees.
   - **InitiatingPayment**
     - Starts payment transaction with gateway.
   - **PaymentGateway**
     - Interaction with external payment API.
   - **PaymentSuccess**
     - Payment verified and stored.
   - **PaymentFailed**
     - Payment rejected; donor can retry.

◦

4. **ReceiptGeneration**

    ◦ **Meaning:** System generates a receipt/acknowledgment (possibly with tax-benefit info).

      Relevance: Supports transparency, legal compliance, and donor satisfaction.

5. **DonationComplete**

    ◦ **Meaning:** Donation request successfully created (and payment done if needed).

    ◦ **Relevance:** Marks end of donation-creation flow and starting point for matching, notifications, and tracking.

Events in Donation Flow

- **Donation Type Selected**

    ◦ SelectingType → EnteringDetails

    ◦ Ensures the UI adapts to the chosen type (fields, validations).

- **Details Confirmed**

    ◦ EnteringDetails → PaymentProcessing

    ◦ Confirms correctness before committing to database or payment.

- **Process Payment**

    ◦ InitiatingPayment → PaymentGateway

    ◦ Starts interaction with external payment service.

- **Payment OK**

    ◦ PaymentGateway → PaymentSuccess

    ◦ Successful transaction; triggers receipt and donation activation.

- **Payment Error**

    - PaymentGateway → PaymentFailed
    - Allows error handling: card failure, gateway down, etc.
- **Retry**

○

PaymentFailed → InitiatingPayment

○ Advanced usability feature letting user try again without restarting whole donation.

- **Payment Successful**
  - PaymentSuccess → ReceiptGeneration
  - Moves to acknowledgement and record creation.

4. Tracking Donation Super-State

States

1. **FetchingStatus**
   - **Meaning:** System queries latest status from database/tracking service (Volunteer/NGO updates).
   - **Relevance:** Allows real-time or near-real-time visibility of where the donation is in the pipeline.
2. **ShowingStatus**
   - **Meaning:** The status (Pending, Accepted, PickedUp, Delivered, Completed) is displayed to the user.
   - **Relevance:** Increases transparency and trust in the system.

Events

- **Status Retrieved**
  - FetchingStatus → ShowingStatus
  - Indicates data successfully fetched.
- **Refresh**

ShowingStatus → FetchingStatus

  - Lets users manually refresh, useful when volunteers update frequently.

- **Back**
  - ShowingStatus → UserOptions
  - Returns to main dashboard, ending tracking session.

5. Support Request Flow Super-State

States

1. **CreatingTicket**
   - User enters problem (e.g., issue with donation, NGO not responding).
2. **SendingTicket**
   - Ticket is submitted to server; ID generated.
3. **AwaitingResponse**
   - System waits for admin/support staff to respond.
4. **ReceivingReply**
   - User sees answer/solution from support.

Relevance

- Implements an **in-app support/ticketing system**, which is an advanced feature for large-scale apps.
- Helps handle disputes, technical problems, and feedback systematically.

Events

  -

- **Submit Issue**
  - CreatingTicket → SendingTicket
- **Ticket Created**
  - SendingTicket → AwaitingResponse
- **Support Responds**
  - AwaitingResponse → ReceivingReply
- **Close**
  - ReceivingReply → UserOptions

  ○ Closes ticket view and returns to dashboard.

6. Admin Operations Super-State

States

1. **UserMgmt**
   - Managing users: verification, suspension, role assignment.
2. **CampaignMgmt**
   - Creating awareness campaigns, events, or special drives.
3. **DonationMgmt**
   - Monitoring donation flow, resolving conflicts, manual reassignment.
4. **ReportGen**
   - Generating analytics (number of donations, cities, NGOs, etc.).
5. **IssueResolution**

   Closing support tickets, resolving flagged problems.

Relevance

- Concentrates all **administrative and analytical functions** in one flow.
- Enables advanced features like **impact analytics**, **campaign management**, and **central moderation**.

Events

- **Next**
  - Moves between admin tasks (UserMgmt → CampaignMgmt → DonationMgmt → ReportGen → IssueResolution).
  - Represents sequential navigation through admin panels.

7. NGO Workflow Super-State

States

1. **VerifyingDonations**
   - NGO checks details: quantity, type, location, timing.
2. **UpdatingStatus**
   - NGO updates status (Accepted/Declined, Received, etc.).
3. **RespondingDonors**
   - NGO sends acknowledgments, feedback, or follow-up requests.

Relevance

- Models **NGO-specific behaviour** separate from donors & volunteers.
- Ensures that donation acceptance and confirmation are traceable and auditable.

    ○

Events

- **Verified**
    - VerifyingDonations → UpdatingStatus
- **Notify Donor**
    - UpdatingStatus → RespondingDonors
    - Triggers notifications to donors about the NGO's action (accepted/received).

8. Logout and Exit

States / Events

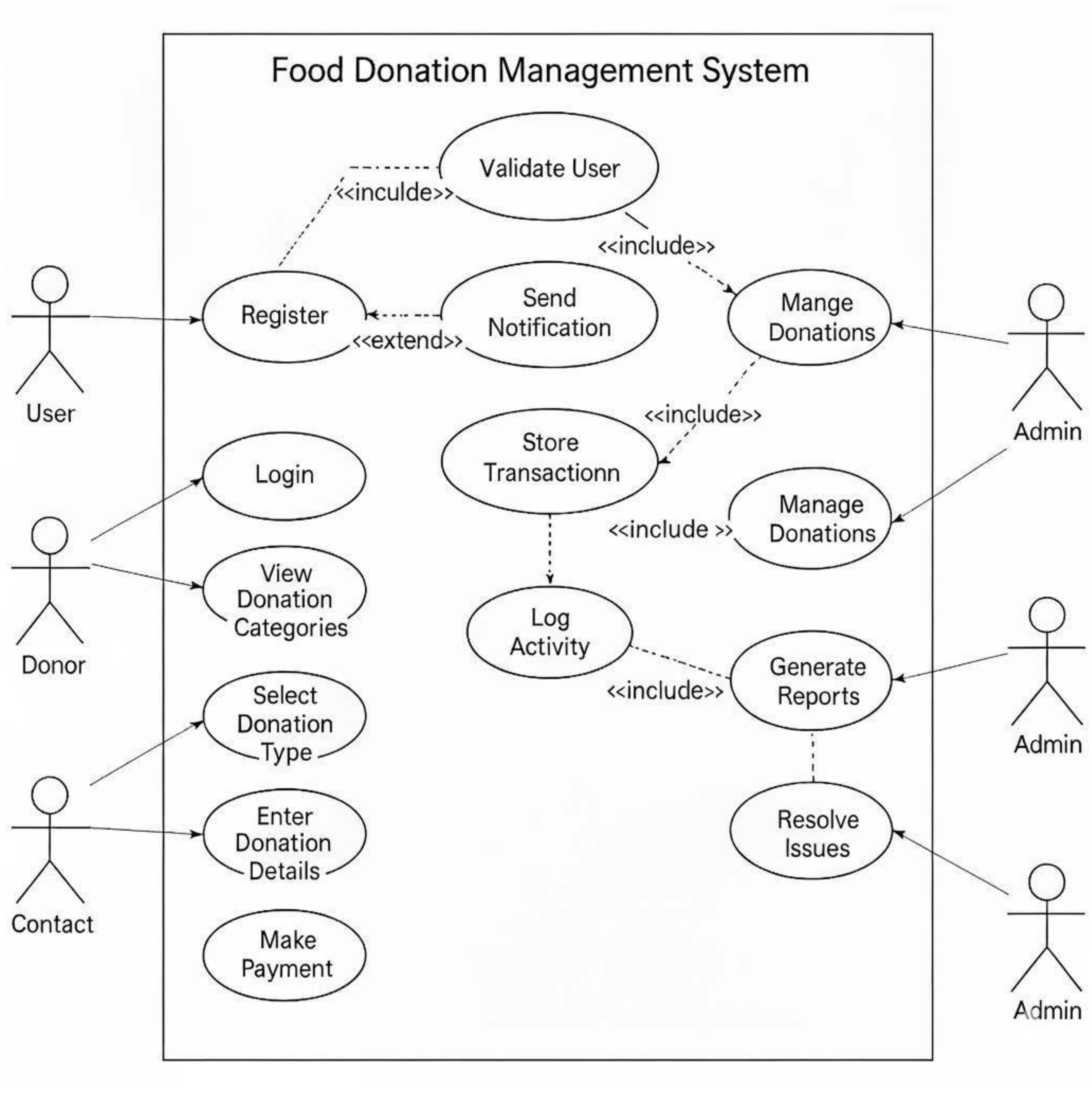
- **Logout (event)**
    - From UserOptions back to **LoggedOut**.
    - Ends current session but keeps app running.
- **Exit Application (final state)**
    - Represents closing the application process entirely.
    - Ends all active flows.

Relevance

- Properly models **session management** and **application lifecycle**, important for security and resource management.

Chapter 5: Interaction Modeling

Use Case Diagram:



Actors and Their Relevance

1. User (General User)

Relevance:
The User represents anyone interacting with the system at the entry level. This actor can register, log in, and access general functions available before selecting a specific role (donor/admin).

- Ensures the system supports basic onboarding.
- Represents common operations applicable to all user types.

2. Donor

Relevance:
This actor donates food through the system. They perform role-specific actions such as selecting donation type, entering details, and making payments if needed.

- Central to the system because donations originate from donors.
- Enables tracking of donation flow, categories, and transactions.

3. Contact (Support User / Help-Seeker)

Relevance:
This actor interacts with the system for support or issue resolution. They may submit complaints, request help, or follow up on donation-related issues.

- Ensures that the system handles user queries and grievances.

- Helps in maintaining service quality and reliability.

4. Admin

Relevance:
This is the system administrator responsible for managing donations, validating users, generating reports, and resolving issues.

- Ensures system integrity and coordination.
- Critical for monitoring activity and maintaining data accuracy.
- Handles verification and supervision tasks that ensure smooth operational workflow.

Use Cases and Their Relevance

1. Register (User)

Relevance:
Allows a new user to create an account in the system.
This is the entry point for all other system services. Includes:

- User data validation Extends:
- Sending notifications

Enables role-based access control after registration.

2. Login (User / Donor)

Relevance:
Allows an existing user to access the system by verifying credentials.

- Ensures secure access to donor or admin features.
- Maintains personal and transactional data privacy.

3. View Donation Categories (Donor)

Relevance:
Allows donors to view different categories such as cooked food, packaged food, groceries, etc.

- Helps donors classify their donations properly.
- Improves NGO matching accuracy.

4. Select Donation Type (Donor

Relevance:
Allows donors to choose the type/category of donation (e.g., cooked food, leftover food, raw groceries).

- Helps system pre-fill or adjust donation requirements.
- Ensures clarity in donor → NGO communication.

5. Enter Donation Details (Donor)

Relevance:
The donor enters specific details such as quantity, pickup address, pickup time, images, etc.

- Essential for NGOs to verify quality and availability.
- Core input for MatchingEngine and logistics.

6. Make Payment (Donor)

Relevance:
Used in case monetary donations or service fees are involved.

- Ensures that payment details and receipts are recorded.
- Supports optional financial operations for certain donation types.

7. Validate User (Admin / System)

Relevance:
Checks user credentials or profile information during registration or login. Included in Register.

- Ensures only legitimate users enter the system.
- Prevents fraud and maintains data security.

8. Send Notification (System)

Relevance:
Sends notifications for registration, verification, donation updates, etc. Triggered by multiple use cases (e.g., Register, Manage Donations).

- Improves communication across donors, NGOs, admins.
- Enhances user engagement and transparency.

9. Store Transaction (System / Donor)

Relevance:
Stores information about donations, payments, and user activities. Included in various actions related to donations.

- Ensures traceability and accountability.
- Provides a basis for analytics and reporting.

10. Log Activity (System)

Relevance:
Maintains activity logs for auditing, tracking, and system health. Included in store transaction, manage donations, handle issues.

- Helps admin understand user patterns.
- Supports security audits and troubleshooting.

11. Manage Donations (Admin)

Relevance:
Admin monitors, updates, and supervises all donations. Examples:

- Approve/decline donation
- Assign volunteers
- Update donation status

Included in many workflows for end-to-end donation management.

12. Generate Reports (Admin)

Relevance:
Admin creates analytical reports about:

- Donation volume
- Donor contributions
- NGO performance
- System usage

Included in many administrative operations.
Supports decision-making and performance improvement.

13. Resolve Issues (Admin)

Relevance:
Admin handles complaints, donation failures, frauds, mismatches, etc.
Helps maintain trust and reliability of the system. Included
in support flow and report analysis.

Understanding Include / Extend Relationships

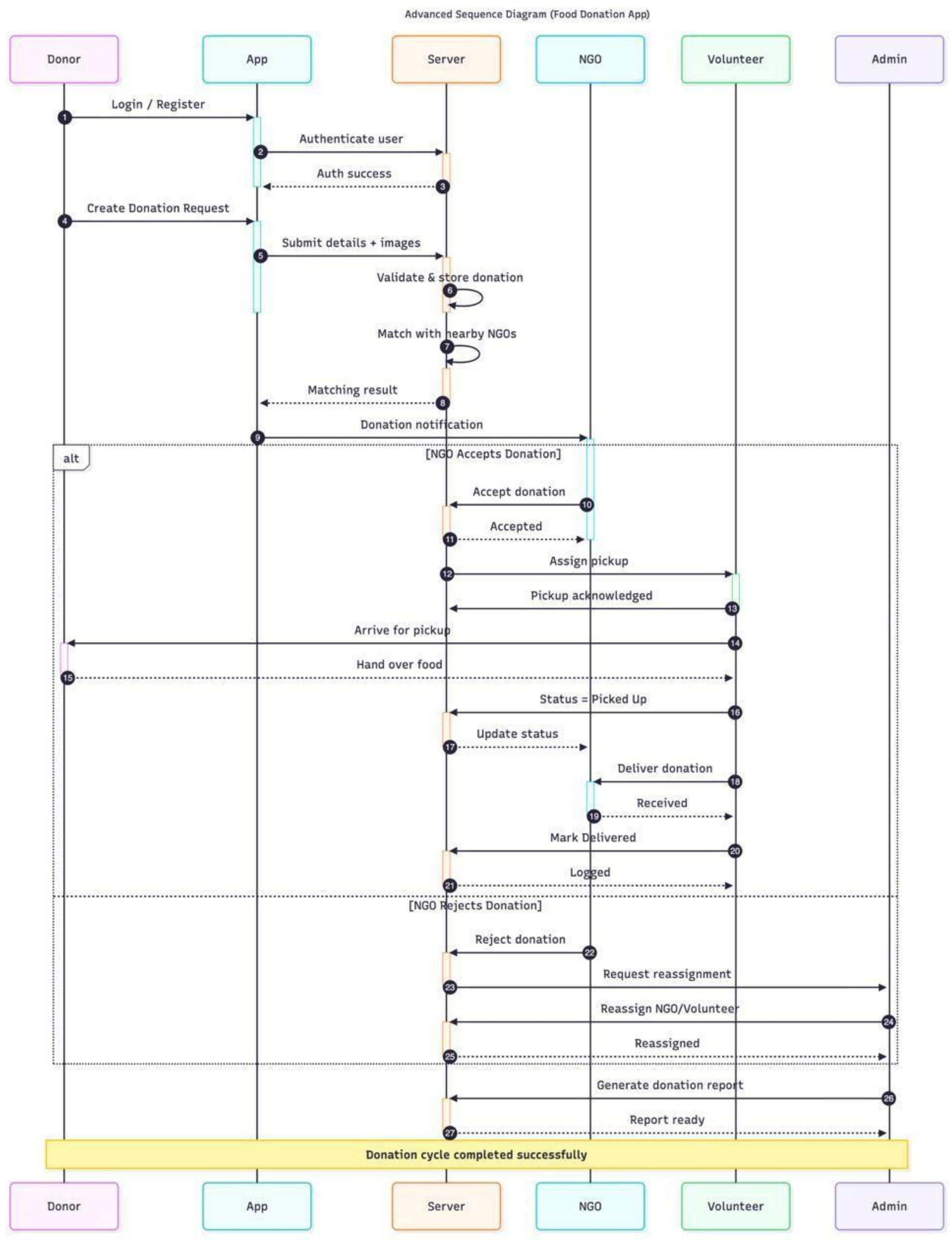
«include»

- Means the included use case is mandatory.
- Example:
    - Register → includes → Validate User
    - This means registration **always** requires validation.

«extend»

- Optional behavior added under specific conditions.
- Example:
    - Register → extends → Send Notification
    - Notification is not the main task of registration but an additional behavior triggered after success.

Sequence Diagram



This sequence diagram visualizes the **end-to-end lifecycle of a food donation**, from donor login to donation completion, involving **six key participants**:

- **Donor**
- **Mobile App**
- **Server (Backend)**
- **NGO**
- **Volunteer**
- **Admin**

1. Donor Authentication Phase

Step 1 – Donor → App: Login / Register

The donor initiates the interaction by registering or logging into the application.

Step 2 – App → Server: Authenticate User

The app sends credentials to the server for verification.

Step 3 – Server → App: Auth Success

Upon successful validation, the server returns authentication status, enabling access to system features.

Relevance

- Ensures secure access
- Supports role-based permissions
- Establishes user identity for donation tracking

2. Creating a Donation Request

Step 4 – Donor → App: Create Donation Request

The donor selects donation type, quantity, location, time, and uploads images.

Step 5 – App → Server: Submit details + images App forwards the donation data to the server.

Step 6 – Server: Validate & Store Donation

The server checks:

- Valid fields
- Safe quantity/time
- Image format

Then stores it in the database.

Relevance

- Ensures accurate data entry
- Makes the donation available for matching
- Secures metadata for NGO decisions

3. Matching the Donation with an NGO

Step 7 – Server: Match with nearby NGOs

Using GPS location, capacity, and availability, the server tries to find the best NGO.

Step 8 – Server → App: Matching Result

The matching outcome (NGO found / no NGO found) is sent back.

Step 9 – Server → NGO: Donation Notification The selected NGO is notified about a new donation.

Relevance

- Implements **smart matching algorithm**
- Uses location-based filtering
- Ensures quick response from the nearest NGO

4. NGO Decision Phase (alt block)

The diagram contains an **alt block** with two mutually exclusive flows:

A. If NGO Accepts Donation

Step 10 – NGO → Server: Accept Donation NGO accepts after reviewing details.

Step 11 – Server → App: Accepted Donor is notified about acceptance.

Step 12 – Server → Volunteer: Assign Pickup

Server assigns a volunteer based on proximity and load.

Step 13 – Volunteer → Server: Pickup Acknowledged

Volunteer accepts the pickup assignment.

Relevance

- Ensures fast allocation of resources
- Supports real-time coordination between NGO and volunteers

B. If NGO Rejects Donation (Alternative Path)

Step 20 – NGO → Server: Reject Donation

NGO rejects due to unsuitability, unavailability, or time constraints.

Step 21 – Server → App: Reject Notification Donor is informed.

Step 22 – Server → Admin: Request Reassignment Admin is alerted to find alternate NGO or volunteer.

Step 23 – Admin → Server: Reassign NGO/Volunteer Admin manually intervenes.

Step 24 – Server → App/NGO/Volunteer: Reassigned New assignment is delivered to all relevant actors.

Relevance

- Ensures system robustness
- Allows human intervention in exceptional cases
- Prevents donation wastage

5. Pickup and Handover Phase

Step 14 – Volunteer → Donor: Arrive for Pickup Volunteer reaches donor's address.

Step 15 – Donor → Volunteer: Hand Over Food Physical transfer of food takes place.

Relevance

- Marks start of logistics journey
- Links donor and volunteer logs

6. Delivery and Completion Phase

Step 16 – Volunteer → Server: Status = Picked Up

Volunteer updates status.

Step 17 – Server → NGO: Deliver Donation Server

notifies NGO that donation is en route.

Step 18 – NGO → Server: Received NGO

confirms receiving the donation.

Step 19 – Server: Mark Delivered

Backend marks the donation as completed.

Relevance

- Enables real-time tracking
- Creates auditable transaction logs
- Completes the food movement chain

7. Administrative & Reporting Phase

Step 25 – Server → Admin: Generate Donation Report

Server compiles delivery data and logs.

Step 26 – Admin → Server: Report Ready Admin

receives finalized analytics/report.

Relevance

- Helps track NGO efficiency, donor contribution, and volunteer performance
- Supports transparency and government reporting

- Enables future improvements and insights

8. Final State

Donation cycle completed successfully

Represents a fully completed cycle:

- Food uploaded
- Matched
- Accepted
- Picked up
- Delivered
- Logged
- Report generated

Advanced Features Demonstrated in This Sequence Diagram

1. Smart NGO Matching Algorithm

- Uses location, availability, and capacity filters
- Dynamic matching loop shown with “Matching result”

2. Real-Time Notifications

- Donor notified after each stage
- NGO receives immediate alerts

3. Role-Based Actions

● Donor, NGO, Volunteer, Admin all have distinct but coordinated responsibilities

4. Manual Reassignment (Error Handling)

● Admin override shown in alternate rejection path

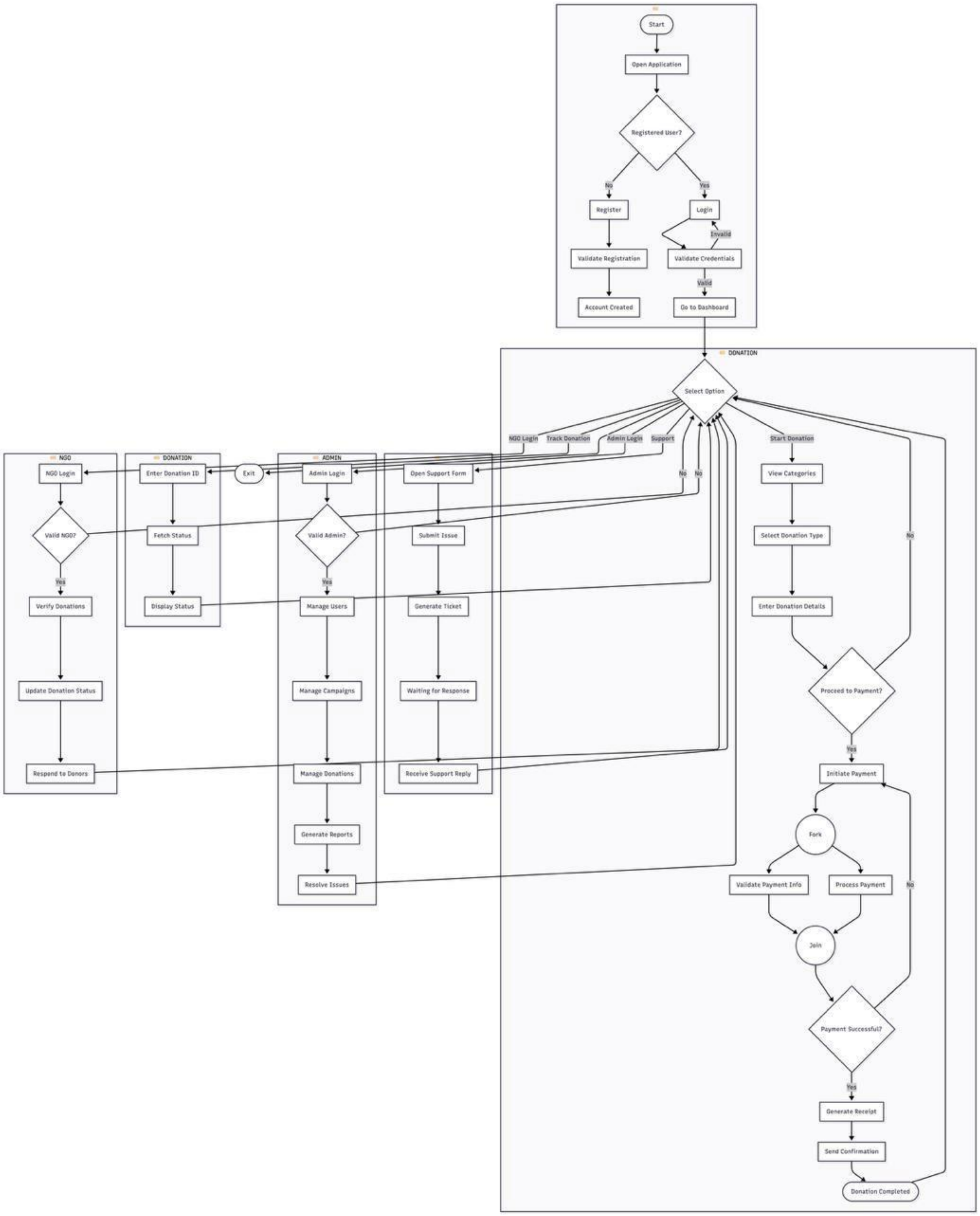
5. Real-Time Status Updating

● Volunteer updates "Picked Up," "Delivered," etc.

6. Complete Audit Trail

- Server logs each step
- Easy for Admin to generate reports

Activity Diagram


Start
Open Application
Registered User?
No
Yes
Register
Login
Invalid
Validate Registration
Validate Credentials
Valid
Account Created
Go to Dashboard
DONATION
Select Option
NGO Login
Track Donation
Admin Login
Support
Start Donation
NGO
DONATION
ADMIN
NGO Login
Enter Donation ID
Exit
Admin Login
Open Support Form
No
No
View Categories
Valid NGO?
Fetch Status
Valid Admin?
Submit Issue
Select Donation Type
No
Yes
Yes
Verify Donations
Display Status
Manage Users
Generate Ticket
Enter Donation Details
Update Donation Status
Manage Campaigns
Waiting for Response
Proceed to Payment?
Yes
Respond to Donors
Manage Donations
Receive Support Reply
Initiate Payment
Generate Reports
Fork
Resolve Issues
Validate Payment Info
Process Payment
No
Join
Payment Successful?
Yes
Generate Receipt
Send Confirmation
Donation Completed

Swimlanes

Swimlane 1: User / Donor / Contact

This lane contains activities done directly by people using the app.

- Register – new user creates an account.
- Login – existing user logs into the system.
- View Donation Categories – donor browses available categories.
- Select Donation Type – donor chooses type (cooked food, packaged, etc.).
- Enter Donation Details – donor enters quantity, address, time, etc.
- Make Payment – donor performs monetary donation/payment if required.
- Contact (implicitly) – user/contact actor may later trigger support-related actions.

Swimlane 2: System (FDMS Core)

This lane contains internal system activities automatically done by the application.

- Validate User – checks credentials / registration data.
- Send Notification – sends emails/SMS/push notifications (e.g., on successful registration or donation).
- Store Transaction – saves donation/payment data into the database.
- Log Activity – records actions for auditing (included whenever store transaction runs).

Swimlane 3: Admin

This lane contains activities done by admins through the dashboard.

- Manage Donations – verify, approve, reassign, or cancel donations.
- Generate Reports – produce analytical or summary reports.
- Resolve Issues – handle complaints, disputes, or failed cases.

Splitting and Merging of Control

Splitting of Control

1. After “Register”
   - The flow splits:
     - Into Validate User (system checks details).
     - Then Send Notification (welcome mail / OTP).
   - Conceptually, a decision/fork: registration can succeed or fail.
2. After “Login”
   - Control branches based on role:
     - If Donor → goes to *View Donation Categories → Select Donation Type → Enter Donation Details → Make Payment*.
     - If Admin → goes to *Manage Donations / Generate Reports / Resolve Issues*.
   - This is a role-based decision node.
3. After “Store Transaction”
   - Flow includes “Log Activity” in parallel/sequence.
   - System may continue to Manage Donations while also logging the operation.

Merging of Control

1. Registration / Login → Normal Usage
   - Whether a user registered or logged in, they both merge into the same “normal use” flow where donors can start donation activities.
2. Admin Operations Merge

  - Admin may start with Manage Donations, jump to Generate Reports, then to Resolve Issues; when any of these complete, control merges back to the idle admin state (ready for next request).

3. Donation Completion Path

  - Multiple possible paths (different donation types, payments success/fail) eventually merge at “Store Transaction” & “Log Activity”, representing a common end of the donation operation being saved.

3. Overall Explanation

1. A User first interacts with the system by performing Register or Login.

2. During registration, the system validates the user and, on success, stores the account details, logs the activity, and sends a notification (e.g., welcome/verification message).

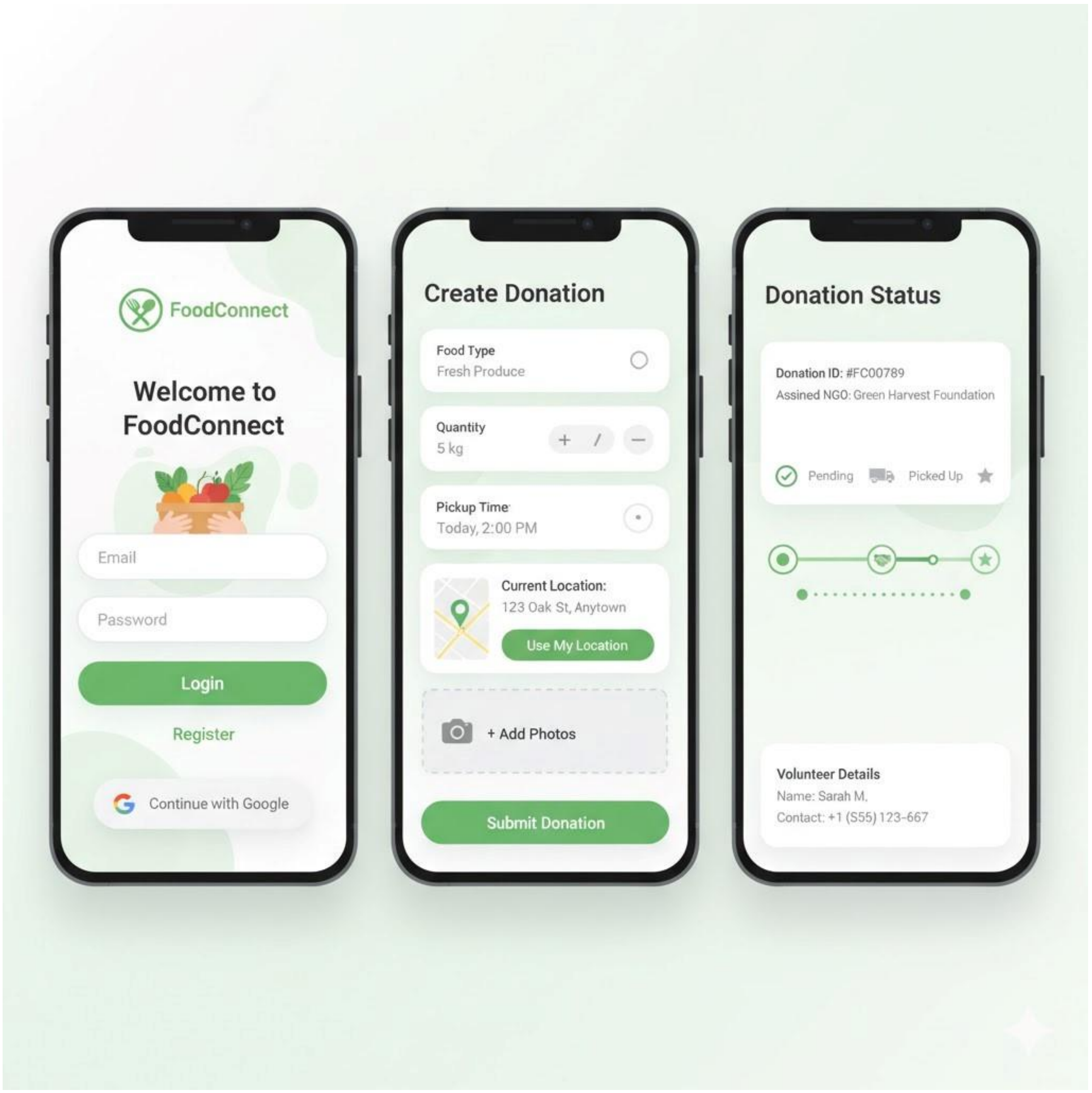
3. A logged-in Donor can then view donation categories, select a donation type, enter donation details, and optionally make a payment.

4. For each donation/payment, the system stores the transaction and logs the activity for auditing and future reporting.

5. In parallel, Admins use Manage Donations to supervise ongoing donations, Generate Reports to analyze performance and impact, and Resolve Issues raised by users or detected by the system.

6. Activities like Store Transaction, Log Activity, Send Notification, and Manage Donations are interconnected with <<include>> relationships, showing that they are reused core sub-activities whenever relevant actions occur.

Ch 6: UI Design with Screenshots


FoodConnect
Welcome to FoodConnect
Email
Password
Login
Register
Continue with Google
Create Donation
Food Type
Fresh Produce
Quantity
5 kg
Pickup Time
Today, 2:00 PM
Current Location:
123 Oak St, Anytown
Use My Location
+ Add Photos
Submit Donation
Donation Status
Donation ID: #FC00789
Assined NGO: Green Harvest Foundation
Pending
Picked Up
Volunteer Details
Name: Sarah M.
Contact: +1 (S55) 123-667