Experiment - 4

UML Diagrams

Food Management System

Software Engineering

By

Vulasala Sujan (BU22CSEN0101959)

Meti Chaitanya (BU22CSEN0101523)

Maraka Ganesh (BU22CSEN0101803)

J Bhargav Reddy (BU22CSEN0101198)

Under the Guidance of

Kerenalli Sudarshana (700542)

Gandhi Institute of Technology and Management

(DEEMED TO BE UNIVERSITY)

BENGALURU, KARNATAKA, INDIA

Academic Year 2024-25

INDEX

Introduction

The food donation application facilitates seamless interactions between donors and recipients, aiming to reduce food wastage while addressing hunger issues. The system's design revolves around three core classes: the User Class, App Class, and System Class, each with clearly defined roles and responsibilities. Together, they form a streamlined architecture that ensures a user-friendly experience, robust backend processing, and effective communication of donation status updates.

This document explores the key functionalities of these classes, highlights the architecture's interaction flow, and provides a Unified Modeling Language (UML) representation for better visualization of the system. The UML diagrams include class diagrams and sequence diagrams to illustrate the relationships and communication patterns among the core components.

Key Classes and Their Responsibilities

1. **User Class**:
    - **Role**: Represents the user interacting with the system, such as donors or recipients.
    - **Responsibilities**:
        - Open the app (openApp()).
        - Log in or register (logIn() and register()).
        - Select food donation option (selectDonateFood()).
        - Enter food details (enterFoodDetails()).
        - Submit the donation (submitDonation()).
        - Log out of the application (logOut()).
2. **App Class**:
    - **Role**: Acts as the intermediary between the user and the backend system.
    - **Responsibilities**:

        - Validate user registration (checkRegistration()).
        - Process food donations (processDonation()).
        - Schedule pickups for claimed donations (schedulePickup()).

3. **System Class**:
    - **Role**: Handles the backend operations and notification processes.
    - **Responsibilities**:
        - Notify users about the acceptance of a donation (notifyAcceptance()).
        - Notify users about the rejection of a donation (notifyRejection()).

Architecture Highlights

- **User Interaction**: The **User** class initiates actions such as logging in, submitting food donations, and logging out. These actions are routed through the **App** class for processing.
- **App Logic**: The **App** class acts as the core of the application, validating inputs, processing donations, and communicating with the **System** class for backend operations.
- **Backend Notifications**: The **System** class is responsible for generating and sending notifications based on the status of donations, ensuring users are informed about the success or failure of their actions.

Relationships and Interactions

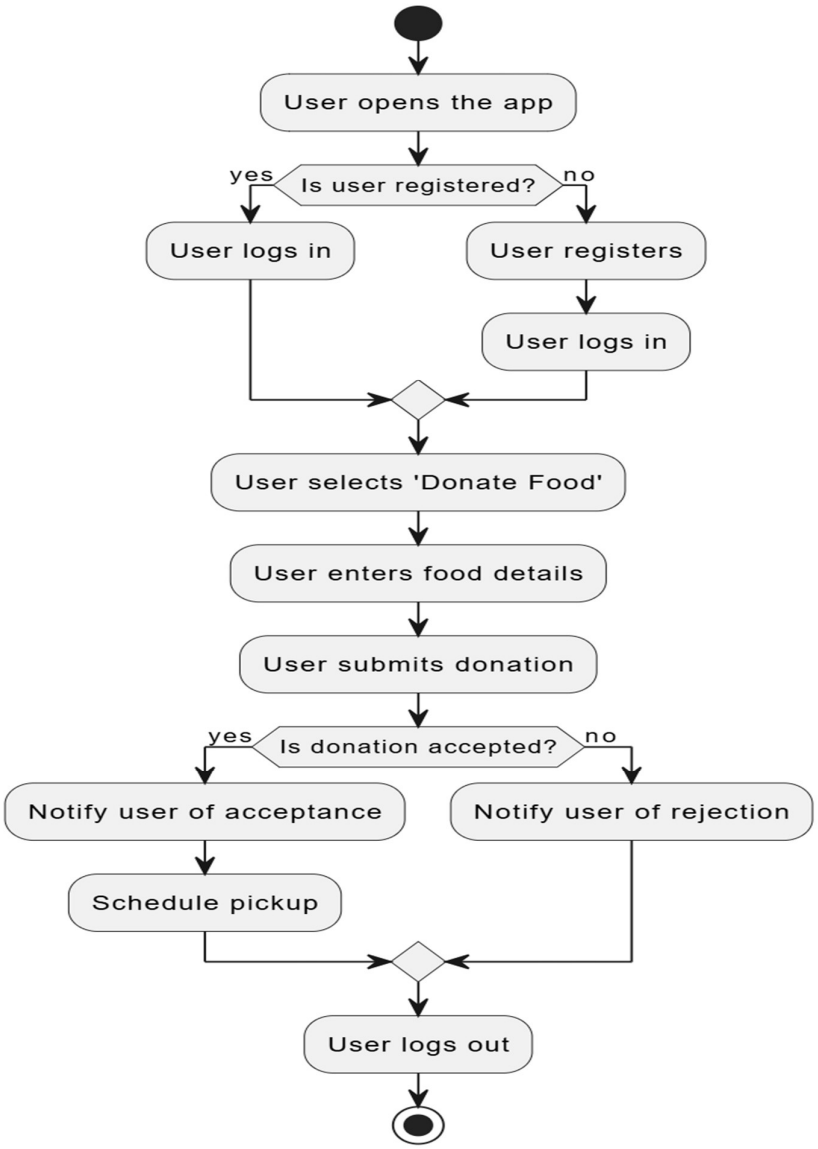
1. **User and App**:
    - The **User** interacts directly with the **App** class to perform their operations.
    - The **App** validates and processes these inputs.
2. **App and System**:

- The **App** communicates with the **System** for backend operations like scheduling pickups and managing notifications.

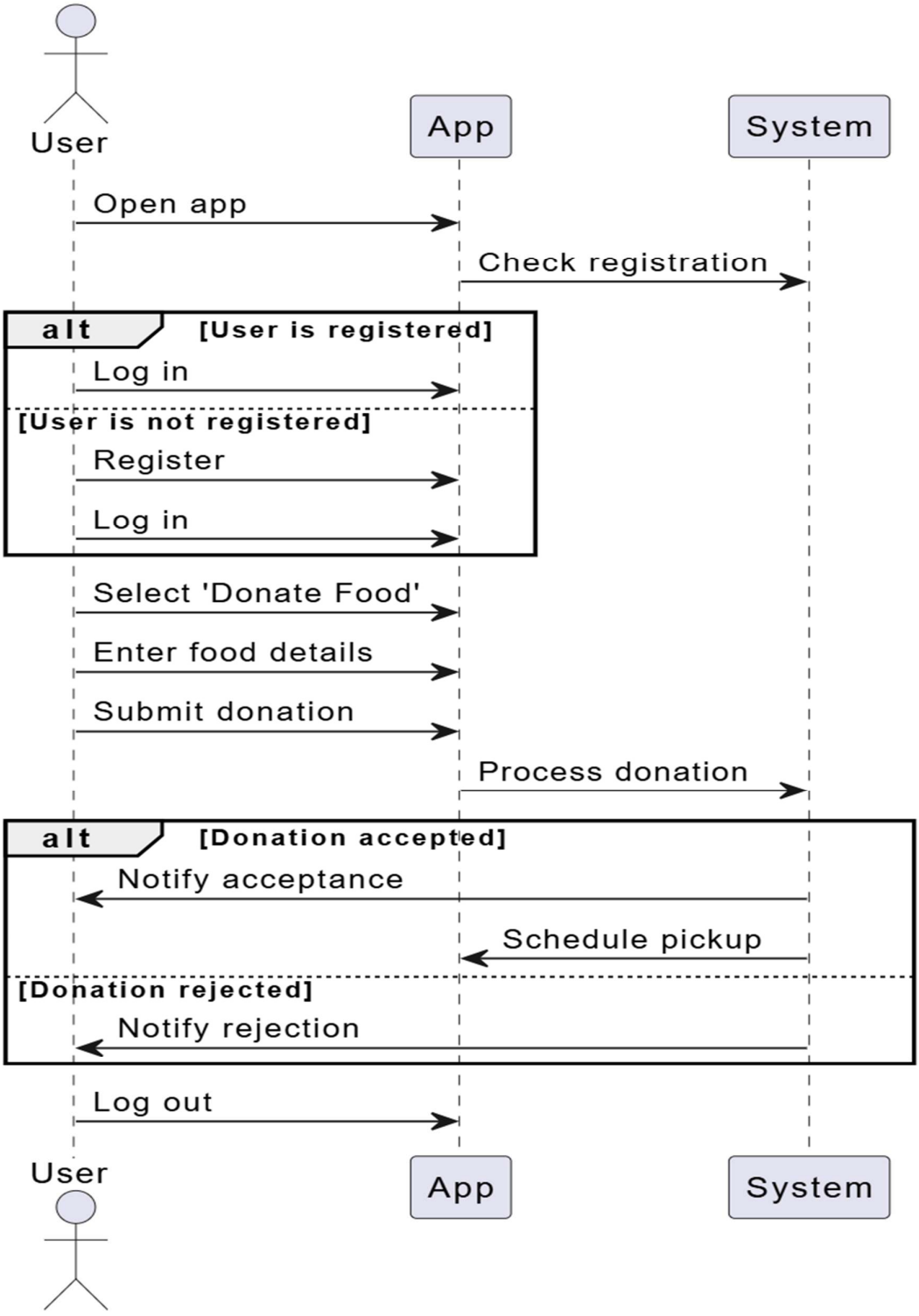
3. **System and User**:
   - The **System** sends notifications back to the **User** based on donation status updates.

UML Diagrams:

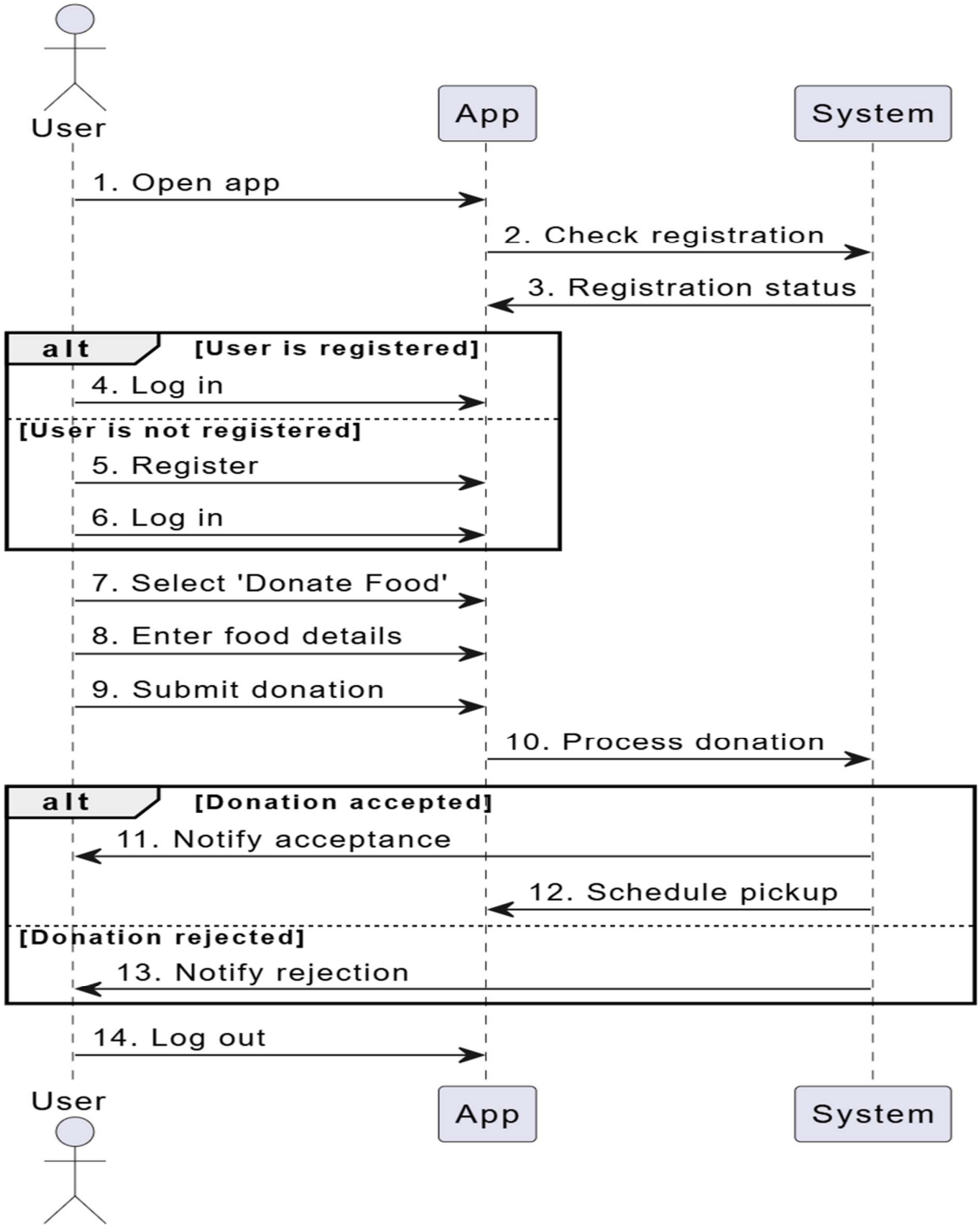
1. **Activity Diagram:**



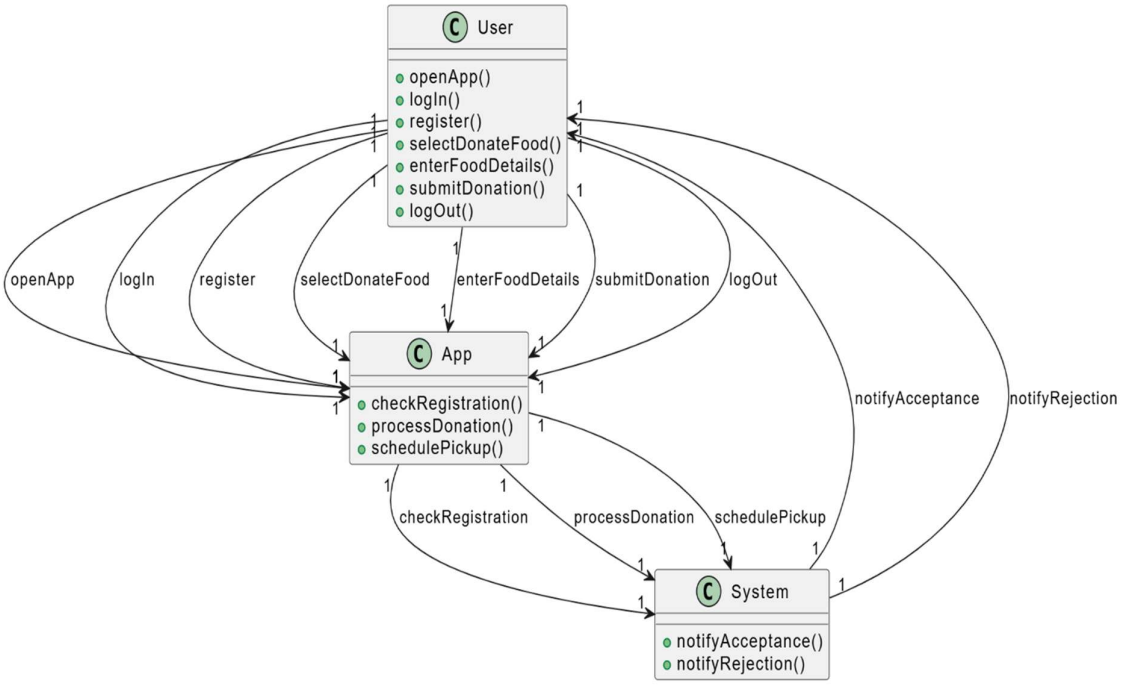
2. **Sequence Diagram:**



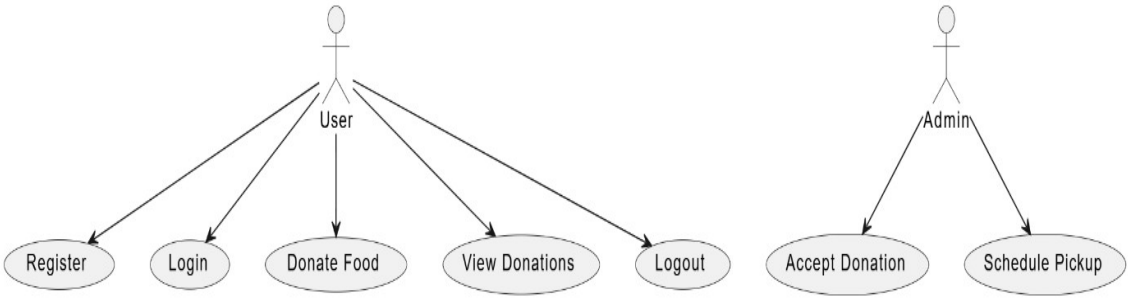
3. **Collaboration Diagram:**



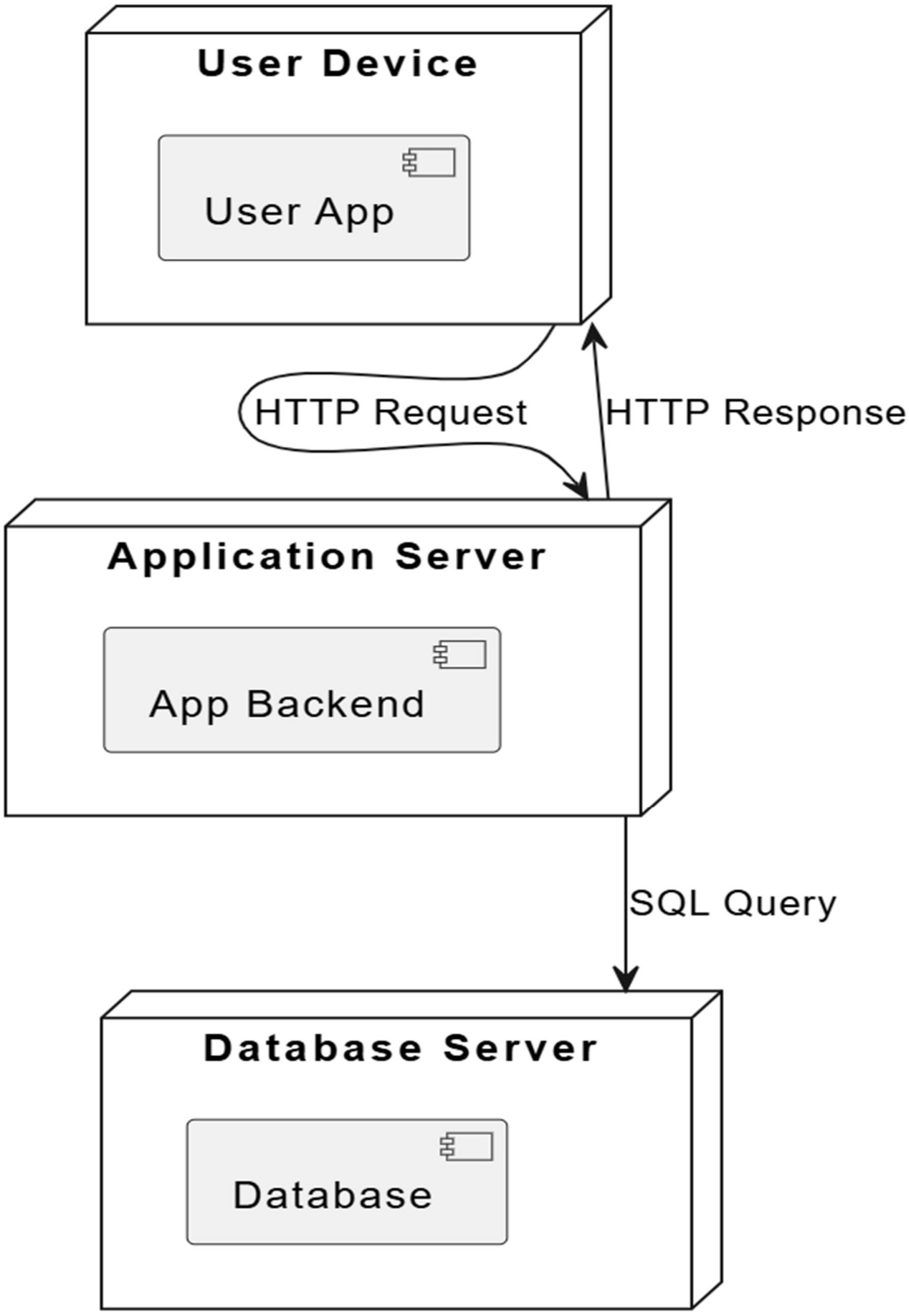
4. Class Diagram:



5. Use-Case Diagram:



6. Deployment Diagram:



CONCLUSION:

The food donation application exemplifies a well-structured design, emphasizing user interaction, efficient backend processing, and timely notifications. By clearly defining roles and responsibilities across the User, App, and System classes, the application ensures seamless coordination between donors, recipients, and the backend system. This modular and scalable architecture not only addresses current functional requirements but also provides a foundation for future enhancements. The accompanying UML diagrams further clarify the system's structure and operation, supporting developers and stakeholders in understanding and extending the application effectively.