# High Level Design

## Version History (Should include Version #, Name of the Author and Date)

1. Version Control:

Version 1.0: Initial draft created by Group 8 members on July 17 2024.

Version 1.1: Initial draft updated by Group 8 members on July 21 2024.

Version 2.0: Initial draft updated by Group 8 members on August 11, 2024.

1. Name of Author(Group 8):

Tang, Yuan(041128167)

Yuan, Lishu(041076188)

Zu, Haoran(041119896)

Ye, Yanni(041107526)

## Introduction (About the system and what does the document is about)

1. About the System

The Food Waste Reduction Platform (FWRP) is an innovative solution designed to address the global challenge of food waste by facilitating the efficient redistribution of surplus food. This platform connects food retailers, consumers, and charitable organizations, fostering collaboration across the food supply chain to promote sustainability, reduce hunger, and build resilient food ecosystems.

1. What does the document is about

This document serves as a comprehensive guide to the design and implementation of the Food Waste Reduction Platform. It includes:

* **High-Level Design**:
  + Solution architecture
  + Technologies used
  + UML diagrams
  + Database model
  + Deployment model
* **Functional Requirements**:
  + User registration
  + Inventory management for retailers
  + Surplus food identification
  + Alert systems for consumers and charitable organizations

## Targeted Audience (Who are the targeted audience for this document)

This document is intended for project stakeholders, developers, and testers involved in the development and deployment of the Food Waste Reduction Platform.

## Scope (What is in and out of the scope of this document)

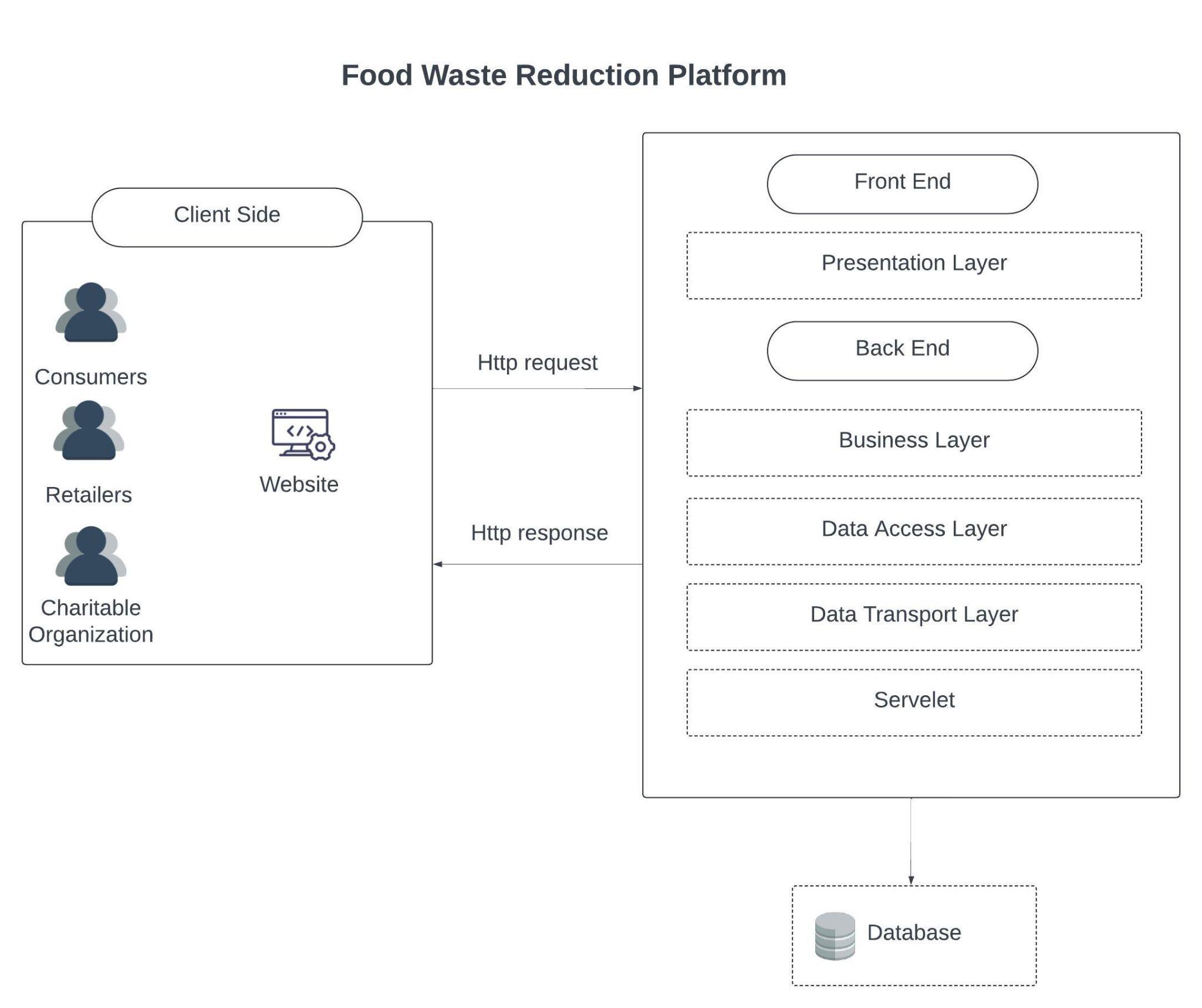
1. In the scope of this document

* **Functional Requirements:**
  + User Registration and Authentication: user registration and login/logout functionalities.
  + Retailer Capabilities: Inventory management including adding, updating, and setting expiration dates for food items. Identification and listing of surplus food items for donation or sale.
  + Charitable Organization Capabilities: Claiming donated food items; Inventory updates post-claim.
  + Consumer Capabilities: Purchasing surplus food items at discounted rates; Inventory updates post-purchase.
  + Surplus Food Alerts: communication method, and food preferences. Automatic notifications to users when new surplus items are listed.
  + Bonus: Implement a security check through user filtering. Limitation and validation for user input data.
* **Database Design**
  + Entity-Relationship Diagram (ERD) and schema for the FWRP database.
  + Sample datasets to illustrate the database structure and functionalities.
* **Solution Architecture:**
  + Description of the three-tier architecture including presentation layer, business layer, and database layer.
  + Detailed explanation of the technologies used in each layer.
* **Version Control:**
  + GitHub for version control.
* **Testing:**
  + Implementation of JUnit tests for codes.
* **Design Patterns:**
  + Explanation and usage of relevant design patterns in the solution.

1. Out of the scope of this document

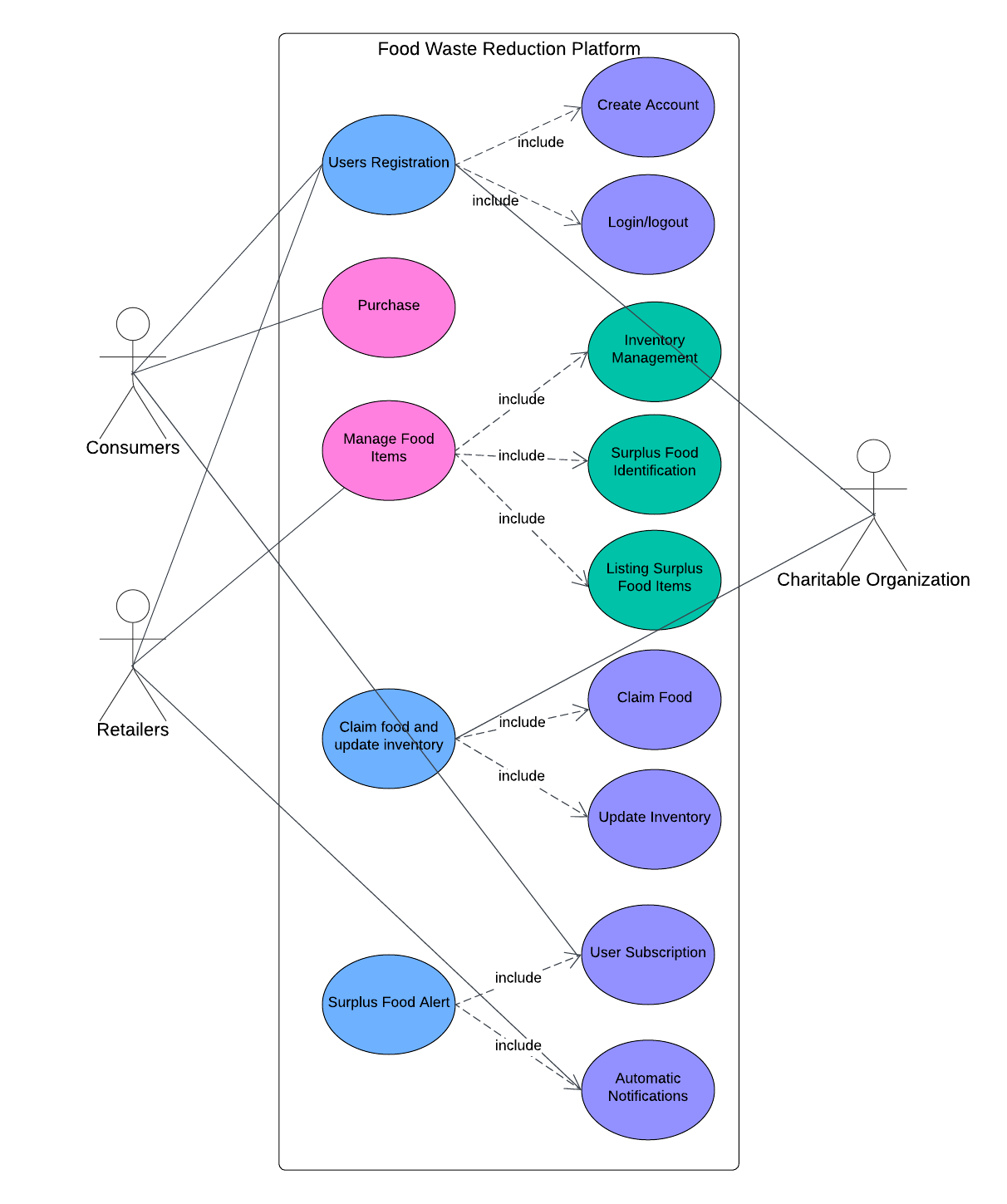
* **Real-time Transactions:**
  + Actual financial transactions for purchasing food items are simulated and not implemented as real transactions.
* **Security Measures Beyond Authentication:**
  + Advanced security features such as encryption, advanced user role management, and data protection beyond basic authentication are not included.
* **Post-Deployment Maintenance:**
  + Ongoing maintenance, updates, and long-term support for the platform are not within the scope of this project.

## Application Architecture (High level architecture/overview of entire system/ main component etc.)

**

Link:https://lucid.app/lucidchart/bdaead6d-d587-446f-9890-ffe25ff99bd4/edit?invitationId=inv\_d205d479-cdfa-40d6-8624-ff38748bdff3

## Business Architecture (Use Case diagrams along with the description)

****

* Consumers: Can register as new user; purchase items; receive food subscription message.
* Retailer: Can register as new user; manage food items (include Inventory Management, read Surplus Food Identification, Listing Surplus Food Items).
* Charitable Organization: Can register as new user; claim food and update inventory.

Link:https://lucid.app/lucidchart/419e07e2-7ca2-4193-b156-f737efff3be0/edit?invitationId=inv\_862989bf-ffe5-472a-aaf7-235b06804cd7

## Detailed Design (Class diagrams, Component diagrams etc.)

This UML diagram represents MVC architecture for a Food Waste Reduction Platform.

**View:**

* + Footer.jsp
  + Subscription.jsp
  + Indexjsplogin.jsp
  + Signup.jsp
  + CharityltemsJSP
  + RetailltemsJSP
  + ShowAddRetailerltemJSP
  + ShowCheckoutJSP
  + ShowClaimJSP
  + ShowEditRetailerltemsJSP
  + ShowltemsJSP
  + ShowSurplusJSP
  + styles.css

**Service:**

* + AlertService.java
  + AlertServicelmpl.java
  + CharityService.java
  + CharityServicelmpl.java
  + CustomerService.java
  + CustomerServicelmpl.java
  + FoodltemService.java
  + FoodltemServicelmpl.java
  + PriceTypeService.java
  + PriceTypeServicelmpl.java
  + PurchaseService.java
  + PurchaseServicelmpl.java
  + RetailerService.java
  + RetailerServicelmpl.java
  + SubscriptionService.java
  + SubscriptionServicelmpl.java

**Controller:**

* + AddRetailerltemController.java
  + CheckoutController.java
  + ClaimltemController.java
  + EditRetailerltemsController.java
  + LoginController.java
  + LogoutController.java
  + ShowCharityltemsController.java
  + ShowCheckoutController.java
  + ShowCustomerltems.java
  + ShowRetailerltemsController.java
  + SignUpController.java
  + SubscribeController.java
  + SurplusltemController.java

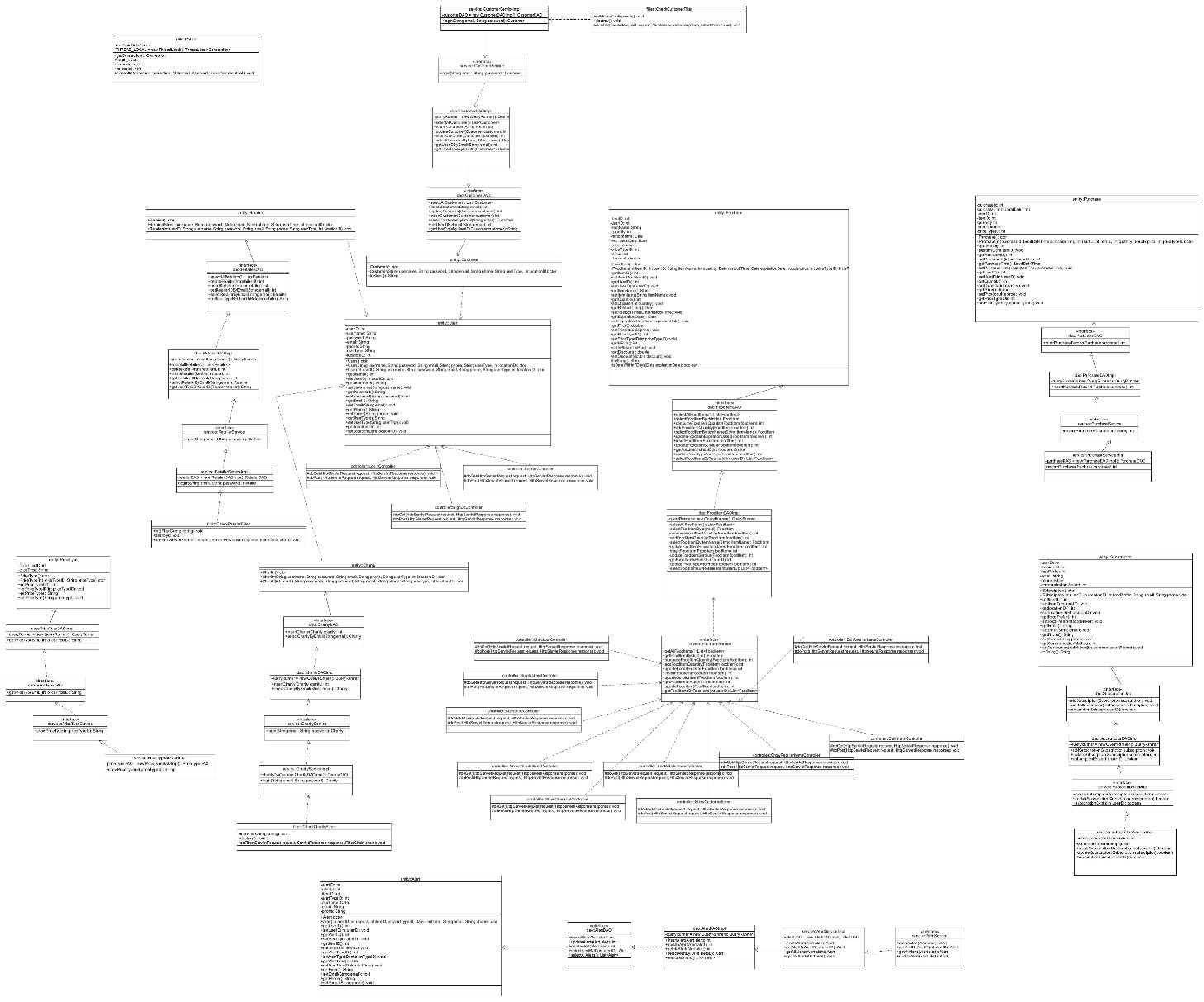
**Data Access Layer:**

* Interfaces defining data access methods:
  + AlertDAO.java
  + CharityDAO.java
  + CustomerDAO.java
  + FoodltemDAO.java
  + PriceTypeDAO.java
  + PurchaseDAO.java
  + RetailerDAO.java
  + SubscriptionDao.java
* Implementation classes:
  + AlertDAOlmpl.java
  + CharityDAOlmpl.java
  + CustomerDAOlmpl.java
  + FoodltemDAOlmpl.java
  + PriceTypeDAOlmpl.java
  + PurchaseDAOlmpl.java
  + RetailerDAOlmpl.java
  + SubscriptionDAOlmp.java

**Data Layer:**

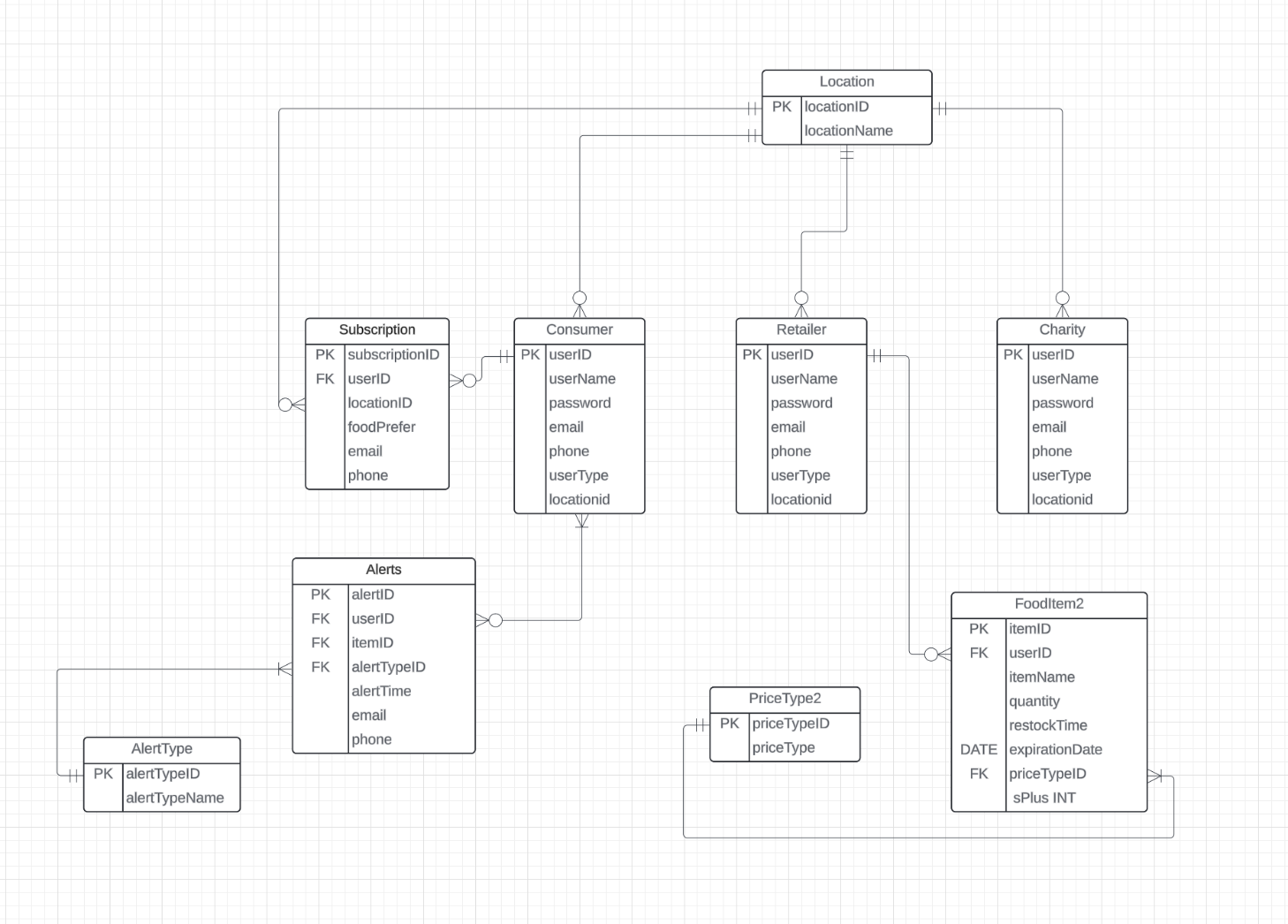
* Entity classes:
  + User
  + Retailer
  + Consumer
  + Charity
  + FoodItems
  + Purchase
  + PriceType
  + Alert
  + Subscription
* DbUtil
  + class for database connections

Relations illustrate dependencies and implementations between the classes, interfaces, and business logic components. This structure ensures modularity, separation of concerns, and ease of maintenance.



Link: https://github.com/LishuYuan0512/JavaProject/blob/main/umlVersion5.jpgf

## Data Architecture (Database structures, ERD, Physical/Logical Data Model)

**

**Entities and Attributes**

• Consumer Table

* userID (PK): Unique identifier for each consumer.
* userName: Name of the consumer.
* password: Password for consumer login.
* email: Email address of the consumer.
* phone: Phone number of the consumer.
* userType: Type of user (specific to consumer).
* locationID (FK): References the locationID in the Location table.

• Retailer Table

* userID (PK): Unique identifier for each retailer.
* userName: Name of the retailer.
* password: Password for retailer login.
* email: Email address of the retailer.
* phone: Phone number of the retailer.
* userType: Type of user (specific to retailer).
* locationID (FK): References the locationID in the Location table.

• Charity Table

* userID (PK): Unique identifier for each charity.
* userName: Name of the charity.
* password: Password for charity login.
* email: Email address of the charity.
* phone: Phone number of the charity.
* userType: Type of user (specific to charity).
* locationID (FK): References the locationID in the Location table.

• Location Table

* locationID (PK): Unique identifier for each location.
* locationName: Name or description of the location.

• Subscription Table

* subscriptionID (PK): Unique identifier for each subscription.
* userID (FK): References the userID in the Consumer table.
* locationID (FK): References the locationID in the Location table.
* foodPrefer: Preferred food type for the subscription.
* email: Email address associated with the subscription.
* phone: Phone number associated with the subscription.

• FoodItem2 Table

* itemID (PK): Unique identifier for each food item.
* userID (FK): References the userID in the Retailer table.
* itemName: Name of the food item.
* quantity: Quantity of the food item.
* restockTime: Time when the food item was restocked.
* expirationDate: Expiration date of the food item.
* priceTypeID (FK): References the priceTypeID in the PriceType2 table.
* sPlus: Surplus quantity available.

• Alerts Table

* alertID (PK): Unique identifier for each alert.
* userID (FK): References the userID in the Consumer table.
* itemID (FK): References the itemID in the FoodItem2 table.
* alertTypeID (FK): References the alertTypeID in the AlertType table.
* alertTime: Time when the alert was triggered.
* email: Email address associated with the alert.
* phone: Phone number associated with the alert.

• AlertType Table

* alertTypeID (PK): Unique identifier for each alert type.
* alertTypeName: Name or description of the alert type.

• PriceType2 Table

* priceTypeID (PK): Unique identifier for each price type.
* priceType: Name or description of the price type (e.g., regular, discounted).

## Security Architecture (What are the security consideration in your designs)

1. Authentication and Authorization:

* **Authentication:**
  + Secure user authentication is implemented using industry standards such as OAuth 2.0 and JWT (JSON Web Tokens) for session management.
  + Passwords are stored securely using bcrypt hashing algorithm to ensure they are protected.
  + Multi-factor authentication (MFA) is implemented to enhance security during the login process.
* **Authorization:**
  + Role-Based Access Control (RBAC) is used to ensure users have access only to the resources and actions permitted for their role (e.g., Retailer, Consumer, Charitable Organization).
  + Fine-grained access control policies are enforced to restrict access to sensitive data and operations.

1. Data Protection:

* **Encryption:**
  + All data in transit is encrypted using HTTPS (HTTP Secure) to protect against interception and man-in-the-middle attacks.
  + Sensitive data at rest, such as user credentials and personal information, is encrypted using AES-256 encryption.
* **Data Masking:**
  + Sensitive data such as email addresses and phone numbers are masked when displayed to users who do not have necessary permissions.

1. Input Validation and Sanitization:

* **Input Validation:**
  + All user inputs are validated on the client side and server side to ensure they meet expected formats and constraints.
  + Regular expressions and validation libraries are used to enforce input constraints and prevent malicious inputs.
* **Sanitization:**
  + Inputs are sanitized to remove any potentially harmful content, protecting against SQL injection, cross-site scripting (XSS), and other injection attacks.
  + Parameterized queries and prepared statements are used for database access to prevent SQL injection attacks.

1. Audit Logging:

* **Audit Logs:**
  + Comprehensive audit logs are maintained to track important actions and access attempts, including user logins, data modifications, and system errors.
  + Logs are securely stored and protected from tampering to ensure they can be used for forensic analysis and compliance reporting.

1. Session Management:

* **Secure Session Management:**
  + Secure cookies are used to store session tokens, with attributes such as HttpOnly and Secure to prevent client-side access and transmission over unencrypted connections.
  + Session timeouts and inactivity timeouts are enforced to minimize the risk of session hijacking.

1. Backup and Recovery:

* **Regular Backups:**
  + Regular backups of the database are performed to ensure data can be restored in case of data loss or corruption.
  + Backups are encrypted and stored securely to prevent unauthorized access.
* **Disaster Recovery Plan:**
  + A comprehensive disaster recovery plan is in place to ensure the system can be quickly restored in the event of a catastrophic failure or data breach.

1. Security Testing:

* **Vulnerability Assessments:**
  + Regular vulnerability assessments are conducted to identify and address security weaknesses in the application and infrastructure.
  + Automated security testing tools are used to scan for common vulnerabilities such as SQL injection, XSS, and insecure configurations.
* **Penetration Testing:**
  + Security professionals conduct periodic penetration testing to simulate real-world attacks and identify potential security gaps.
  + Findings from penetration tests are addressed promptly to enhance the overall security posture of the platform.
  + Regular security testing, including vulnerability assessments and penetration testing, is conducted to identify and address security weaknesses.

## Deployment Architecture (Infrastructure, deployment model etc.)

Link: <https://online.visual-paradigm.com/community/share/deployment-1u29u18f3k>

* Client Interaction: Users interact with the platformthrough their web browsers, sending HTTP requests to perform various actions.
* Webserver Handling: The webserver receives these requests, serves static content.
* Application Processing: The application server processes the business logic, interacts with the database server for data operations, and returns responses to the webserver.
* Database Operations: The database server performs the necessary data operations and ensures data integrity and availability.
* Version Control: GitHub manages the source code, enabling collaborative development, code reviews.

## Testing Model (How are you testing your application. Junit, API testing etc)

* **Junit Testing**

Unit testing focuses on verifying the functionality of individual components of the application. By isolating each unit, we can ensure that every part of the code performs as expected.

Scope: Testing individual methods and classes within the application

* **API testing**

API testing ensures that the API endpoints of the application return the expected results and handle various input scenarios correctly.

Scope: Testing the RESTful API endpoints of the application

## References

AWS Documentation: <https://docs.aws.amazon.com/>

## Acronyms/Abbreviation

* FWRP: The Food Waste Reduction Platform
* ERD: Entity-Relationship Diagram