A close-up of a logo

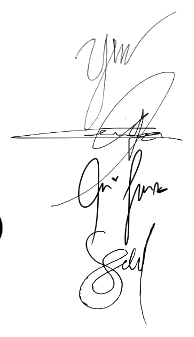
Description automatically generated

**University of Mindanao**

Department of College of Computing Education

Course: BSCS

**Mini Online Grocery System**



Yosh Batula 544580 October 9, 2024

Mheil Andrei Cenita 545045 October 9, 2024

Annika Dumalogdog 547692 October 9, 2024

Kent Leonel Sevellino 545000 October 9, 2024

Instructor:

Prof. Modesto C. Tarrazona

Submission Date:

December 18, 2024

**Table of Contents**

1.Introduction

2.Objectives

3.System Architecture

4.Class Diagram

5.Implementation Details

6.Key Features

7.Testing and Results

8.Conclusion

9.Future Enhancements

10.References

### **Introduction**

#### 1. 1 Overview

The Mini Online Grocery System is a compact e-commerce application designed to simulate the functionalities of a modern grocery store, complete with an integrated delivery system. This system allows users to browse, select, and purchase essential grocery items such as fruits, vegetables, beverages, and laundry supplies, all from the convenience of their devices. It aims to replicate the user experience of a physical grocery store while addressing the limitations associated with traditional shopping.

The inspiration for this project stems from the growing need for convenient shopping solutions in today’s fast-paced world. With increasingly hectic schedules, many individuals find it challenging to allocate time for grocery shopping. Long queues, restricted store hours, and the inconvenience of commuting to and from stores exacerbate the problem, often leaving consumers frustrated or unable to complete their errands effectively.

#### 1.2 Motivation

This project is important because it addresses these pain points by offering a streamlined, time-saving alternative to traditional grocery shopping. By implementing a system that allows users to shop at any time and have their orders delivered directly to their doorsteps, the Mini Online Grocery System significantly reduces the effort and time required to purchase essential goods.

#### 1.3 Problem Statement

The problem it solves is the inconvenience and inefficiency of traditional grocery shopping, which often involves significant physical effort and time investment. The system improves on this by offering features such as category-based browsing, real-time inventory tracking, and customizable delivery options. Customers can make informed purchasing decisions, modify their cart easily, and complete transactions seamlessly. With this approach, the Mini Online Grocery System ensures that users can prioritize their daily responsibilities without compromising on their grocery needs, making essential shopping both accessible and stress-free.

### **Objectives**

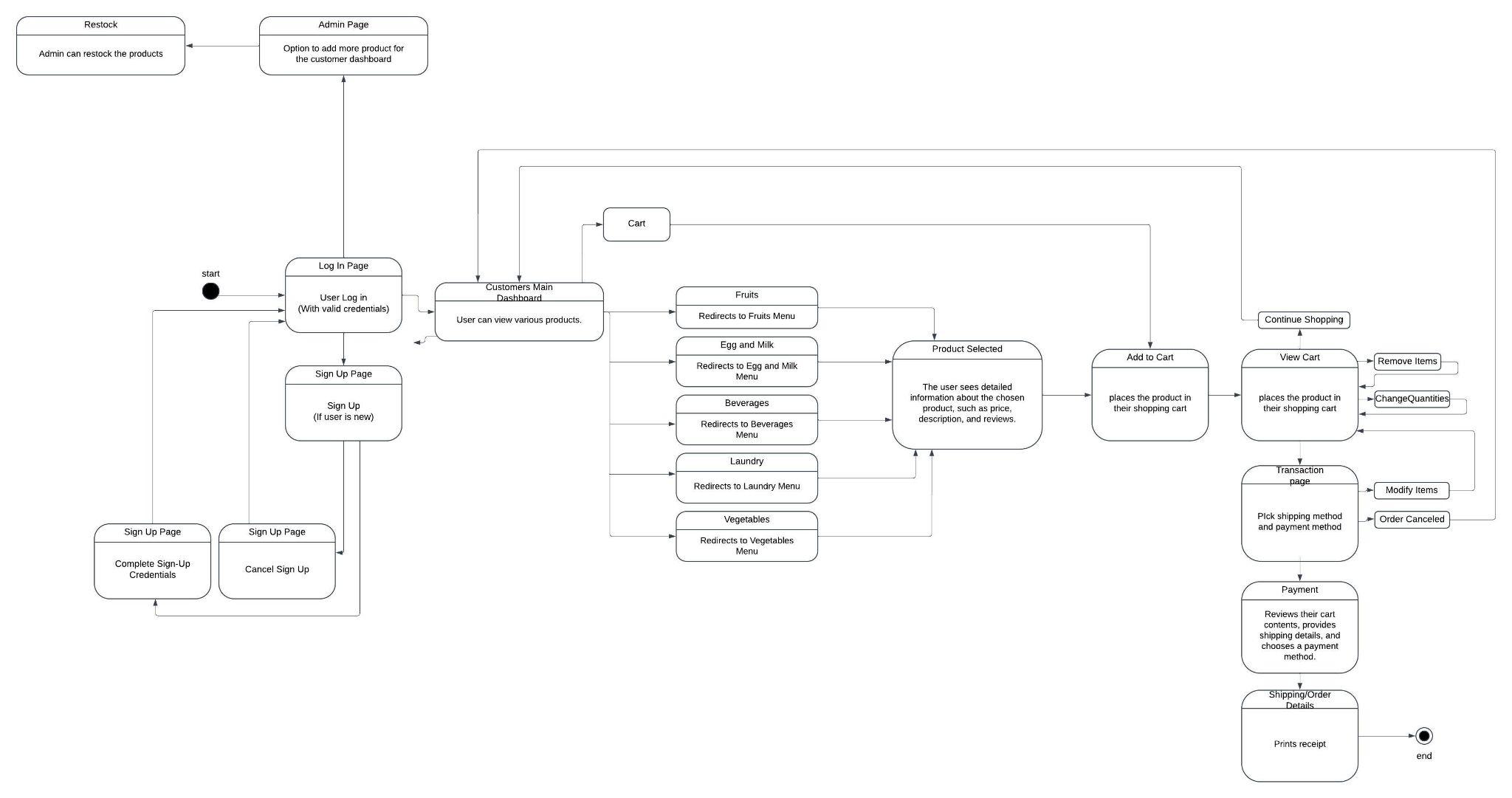
#### 2. 1 Primary Objective

The primary objective of this system is to enhance the grocery shopping experience by providing a user-friendly platform that allows customers to browse products, manage their shopping carts, and complete transactions with ease. The system aims to reduce the time and effort traditionally associated with grocery shopping by offering a streamlined and efficient process. Through its intuitive design and functionality, the Mini Online Grocery System simplifies the entire shopping experience, making it accessible and stress-free for users.

#### 2.2 Secondary Objectives

In addition to the primary goal, the system is guided by three key secondary objectives: helpfulness, convenience, and accessibility. (1) The system strives to be helpful by incorporating features such as a well-organized product catalog, robust shopping cart management, and secure payment options, ensuring that users can complete their purchases smoothly and efficiently. (2) Convenience is a key in this system, allowing customers to shop from the comfort of their homes, manage orders effortlessly, and even track their purchases in real time.(3)Lastly, the system prioritizes accessibility by being compatible with various devices and offering a user-friendly interface that caters to a diverse range of customers. Together, these objectives create a seamless and enjoyable shopping experience that is both practical and inclusive.

### **System Architecture**

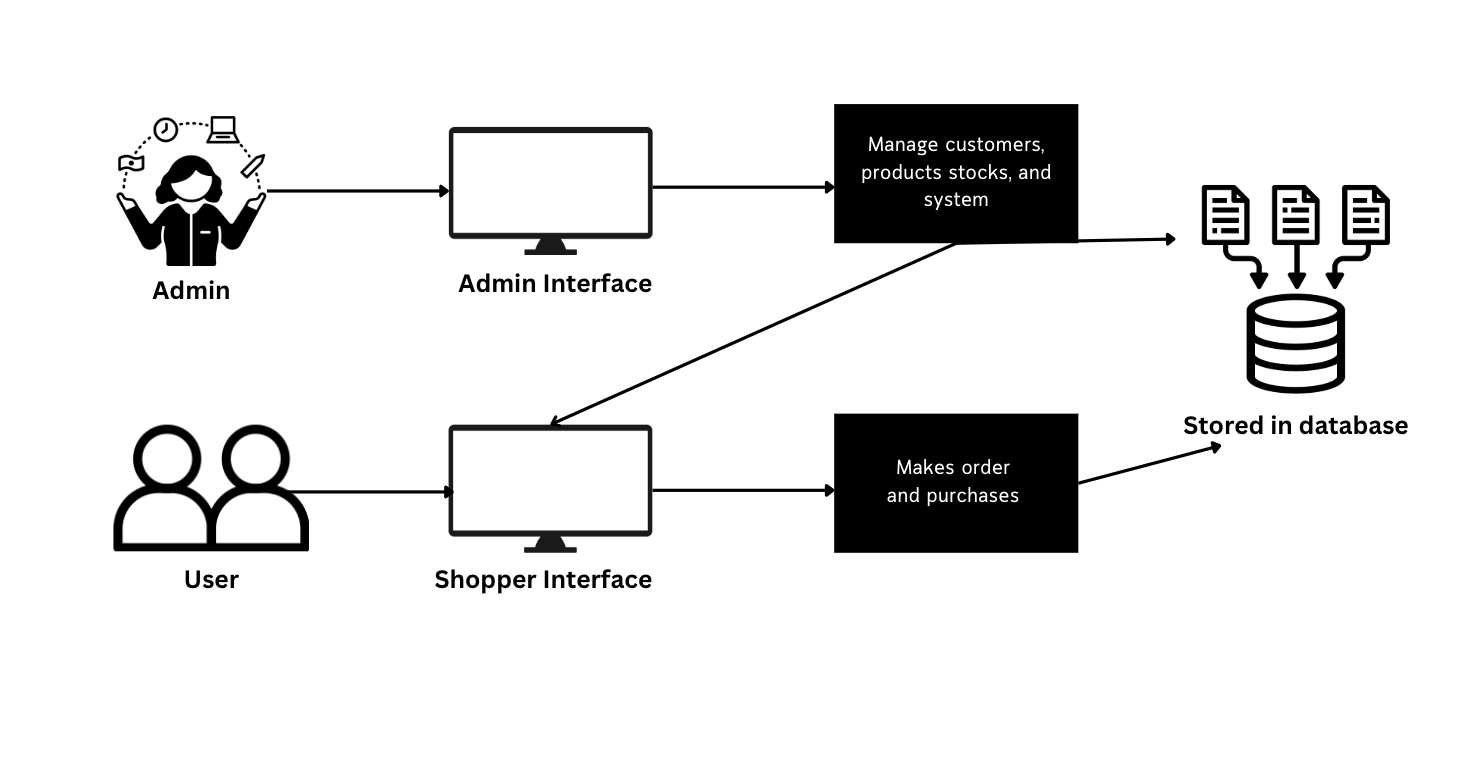


The system allows users to log in either as an admin or as a customer. Customers who do not have existing credentials in the database can sign up by creating an account. After successfully registering, they can log in and are redirected to the main dashboard, which features various product categories. These categories organize the items for easier browsing and shopping. Each product is displayed in a small panel that includes a product name, image, stock quantity, price, and an "Add to Cart" button. Customers can browse through the categories, select items they wish to purchase, and add them to their cart by clicking the button.

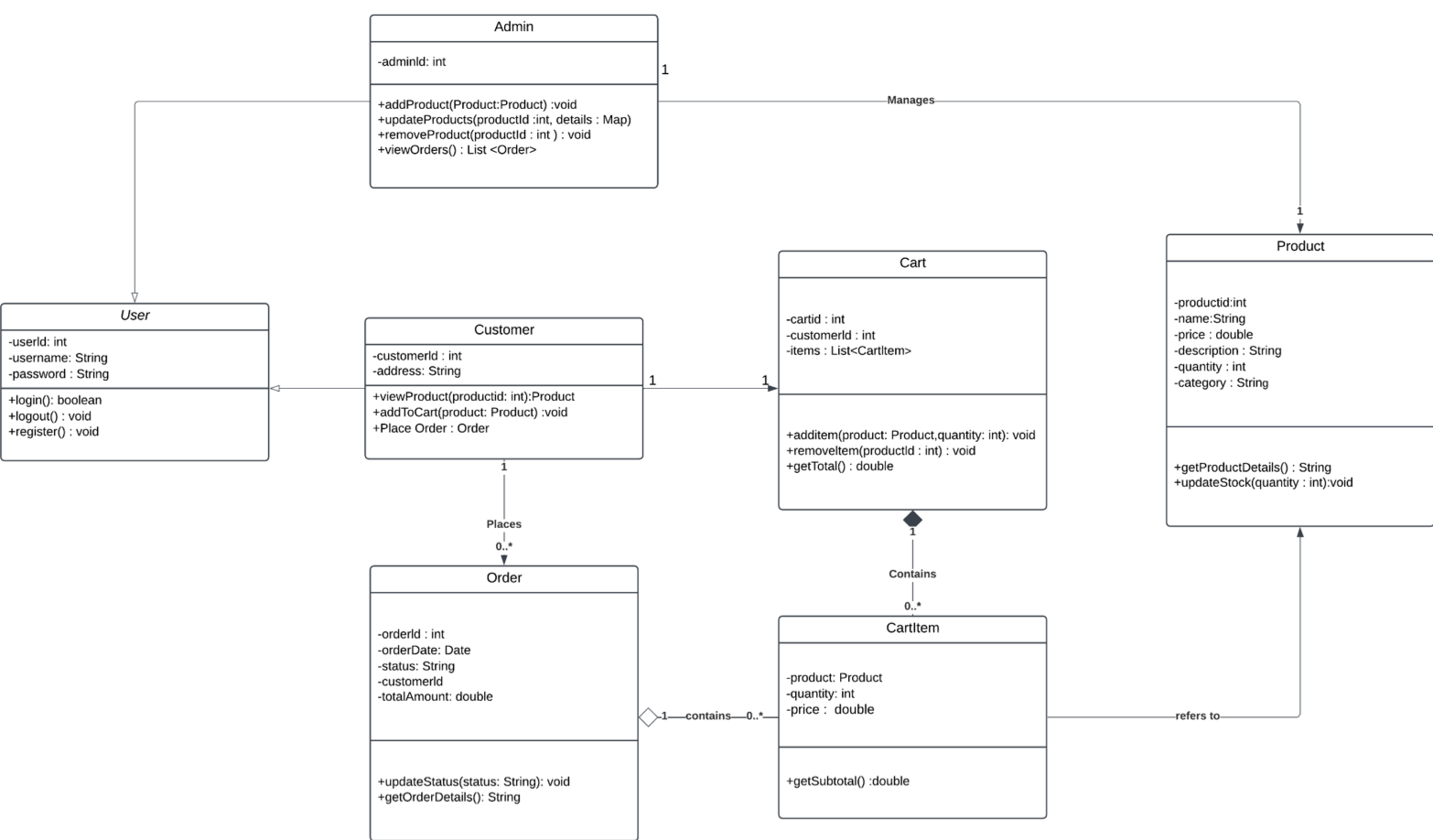
The cart allows users to visually review the items they have added, with the option to modify their selection. Customers can update quantities, add more items, or remove existing ones. The cart also dynamically updates the total price as changes are made, providing a clear overview of the ongoing transaction. Once the user is satisfied with their selection, they proceed to the transaction page.

On the transaction page, users can choose their preferred payment method and delivery speed. They are then directed to a summary page that reiterates the details of their order, including the items purchased, selected payment method, delivery preferences, and the final total price. To complete the process, the system generates a receipt summarizing the transaction, ensuring the customer has a record of their purchase.

For admin users, after their login credentials are verified against the database, they are redirected to the admin dashboard. This dashboard provides administrative tools for managing the products displayed on the customer dashboard. Admins can restock items or add new products, ensuring the system's inventory remains up-to-date and available for customers. This functionality supports smooth operations and allows for efficient management of the system’s offerings.



### **Class Diagram**



The UML diagram serves as a blueprint for the structure of our system, modeling the core classes and their relationships. The primary classes include User, Admin, Customer, Cart, CartItem, Order, and Product, each playing a distinct role in the system. These classes are depicted in the UML diagram with three sections: the class name at the top, attributes in the middle, and methods at the bottom. The attributes represent the characteristics of the class, such as `username` or `price`, and are marked private (denoted by a dash “-”) to ensure data encapsulation. The methods list the actions or operations a class can perform, such as `addToCart()` or `updateStock()`, showcasing how the classes interact with data to fulfill their specific roles.

The relationships between the classes are designed to reflect object-oriented programming principles, emphasizing inheritance, association, and composition. The Admin and Customer classes inherit attributes and methods from the User class through an inheritance relationship, ensuring a shared structure for common functionality like logging in. This hierarchy eliminates redundancy and allows each subclass to extend its capabilities with unique functionalities. For instance, the Admin class is associated with the Product class in a one-to-one relationship, enabling the admin to manage product inventory directly. Meanwhile, the Customer class is associated with the Order class in a one-to-many relationship, allowing customers to place multiple orders over time.

The composition relationship between Cart and CartItem reflects the user experience of adding items to a cart. Each Cart object contains multiple CartItems, which, in turn, are associated with Order objects in a zero-to-many relationship. This design ensures that individual items can be tracked accurately within orders, and changes to the cart, such as adding or removing items, dynamically update the total price. Additionally, attributes like `totalPrice` in the Cart class and methods like `calculateTotal()` ensure seamless functionality and user-friendly interactions.

These relationships were deliberately designed to replicate real-world interactions and provide a clear, modular structure to the system. By utilizing inheritance, code reusability is maximized, and the logical hierarchy of users is maintained. Associations between classes, such as Admin-Product and Customer-Order, establish clear and functional boundaries, ensuring smooth workflows. The composition between Cart and CartItem further enhances user convenience by enabling intuitive management of cart items and seamless transitions to the checkout process.

In summary, the UML diagram not only illustrates the structural design of the system but also emphasizes its adherence to object-oriented principles. The relationships between the classes support core functionalities such as user role management, product inventory updates, and transaction workflows. This robust design ensures scalability, modularity, and efficient management of the system's components, ultimately delivering a user-friendly and reliable online grocery platform.

### **Implementation Details**

### 5.1 Key Classes and Objects

CoverPage:   
**o Purpose:** To manage the application's cover page, providing navigation to login and sign-up pages, and displaying a background image.

**o Attributes:** loginButton, signupButton, image  
**o Methods:** initialize(), goToLogin(), goToSignUp()

SignUp:   
**o Purpose:** Sign up credentials  
**o Attributes:** EmailTXT, PasswordTXT, loginBTN, signupBTN, usernameTXT  
**o Methods:** goToWelcomePage(), goToLoginPage()

Login:

**o Purpose:** Login Credentials already present in the database  
**o Attributes:** LogInBTN, SignUpBTN, passwordTxt, usernameTxt  
**o Methods:** goToSignUpPage(),goToWelcomePage()

WelcomePage:

**o Purpose:** Loading Page to welcome users before main dashboard

**o Attributes:** ImageBackground, WelcomeBTN, labelName, labelNameS  
**o Methods:** initialize(), setUsername(String username)

Payment:  
**o Purpose:** To manage the payment and delivery selection process, validate user input, and navigate to the payment summary page.

**o Attributes:** PMethodBTN, PMethodBTN111, PMethodBTN1111, PMethodBTN11111, PMethodBTN111111, PlaceOrder, methodPayment, methodPayment, termsCheckBox, selectedDeliveryMethod  
**o Methods:** handleExit(), handleMethodDelivery(ActionEvent event), handleMethodPayment(ActionEvent event), initialize(), handlePlaceOrder()

Payment2:   
**o Purpose:** To manage the user interface for displaying the delivery and payment details, along with animations in the payment summary page.

**o Attributes:** DeliveryText, PaymentText, TotalText, animatedImage  
**o Methods:** initialize(), setDetails(String deliveryMethod, String paymentMethod), animateImage()

User:  
**o Purpose:** Th User class serves as an abstract base class representing a user with encapsulated details such as username, password, address, email, and contact number. It provides getter and setter methods to access and modify these attributes.

**o Attributes:** userName, .password, address, email, contactnumber

**o Methods:** initialize(), setDetails(String deliveryMethod, String paymentMethod), animateImage()  
  
Beverages:   
**o Purpose:** The Beverages class represents a product in the Beverages category, including functionality to manage size information and provide product details.

**o Attributes:**size

**o Methods:** Beverage(String name, double price, String rating, String imageURL, String size), Beverage(), getSize(), setSize(String size), setData(String name, double price, String rating, String imageURL, int stock), isLargeSize() displayDetails()

Fruits:   
**o Purpose:** The Fruits class represents a fruit product with seasonal availability, providing methods to manage and check its details.

**o Attributes:** Season

**o Methods:** Fruits(), Fruits(String name, double price, String rating, String imageURL, String season), getSeason(), setSeason(String season), setData(String name, double price, String rating, String imageURL, int stock), isInSeason(String currentSeason)

Laundry:   
**o Purpose:** The Laundry class represents a laundry product, managing brand information and providing product details.

**o Attributes:** brand

**o Methods:** Laundry(), Laundry(String name, double price, String rating, String imageURL, String brand), getBrand(), setBrand(String brand), setData(String name, double price, String rating, String imageURL, int stock), displayDetails(), isPremiumBrand()

Milk and Eggs:   
**o Purpose:** The MilkAndEggs class represents a milk and eggs product, managing expiration dates and providing product details.

**o Attributes:** expirationDate

**o Methods:** MilkAndEggs(), MilkAndEggs(String name, double price, String rating, String imageURL, String expirationDate), getExpirationDate(), setExpirationDate(String expirationDate), setData(String name, double price, String rating, String imageURL, int stock), isExpired(String currentDate), displayDetails()

**Admin:** o Purpose: The Admin class inherits from User and allows administrators to manage products and system operations.  
 o Attributes: username, email, password, address, contactNumber (inherited)  
 o Methods: Admin(String username, String email, String password, String address, String contactNumber), displayAllProducts(), loadProducts(), displayInfo()

**Customer:** o Purpose: The Customer class inherits from User and represents the customer’s ability to browse products, add them to the cart, and manage purchases.  
 o Attributes: username, email, password, address, contactNumber (inherited), favorites, totalPrice  
 o Methods: Customer(String username, String email, String password, String address, String contactNumber), viewCart(), updateTotalPrice(), displayInfo()

**CartItems:** o Purpose: The CartItems class represents individual products added to a customer's shopping cart.  
 o Attributes: name, price, quantity, stock, imageURL  
 o Methods: CartItems(String name, double price, int quantity, int stock, String imageURL), getName(), getPrice(), getQuantity(), setQuantity(int quantity), getStock(), setStock(int stock)

**Product:** o Purpose: The Product class provides methods for setting and displaying product details, making it suitable for both product display and cart management.  
 o Attributes: name, price, rating, imageURL, stock  
 o Methods: setData(String name, double price, String rating, String imageURL, int stock), setData(String name, double price, String rating, String imageURL), setDataofCartItem(String name, double price, String rating, String imageURL, int stock)

**DatabaseConnection:** o Purpose: The DatabaseConnection class manages the connection between the application and the database.  
 o Attributes: DB\_URL, USERNAME, PASSWORD  
 o Methods: getConnection()

#### 5.2 Key OOP Principles Use

**1. Encapsulation**

**Encapsulation** ensures that an object's data is hidden and accessed only through controlled methods, enhancing security, maintainability, and flexibility.

**Implementation in the System**

1. **Private Fields**:  
    Key data members across classes such as Product, CartItems, Customer, and Admin are declared as private to restrict direct access.

**Example**:  
 In the CartItems class:

private String name;

private double price;

private int quantity;

private int stock;

1. **Public Getter and Setter Methods**:  
    To allow controlled access to private fields, public getters and setters are implemented.

**Example**:

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

1. **Encapsulation in Controllers**:
   * In **BLoginPageController**, the validateUser() method encapsulates database validation logic.
   * In **Admin**, methods like displayAllProducts() encapsulate product retrieval and display logic from the database.

**Example**:

private boolean validateUser(String email, String password) {

String sql = "SELECT COUNT(\*) FROM customers WHERE email = ? AND password = ?";

...}

**Benefits**:

* Prevents unauthorized access to data.
* Provides a clear interface for interacting with objects.
* Makes the code more modular and easier to maintain.

**2. Inheritance**

**Inheritance** enables one class to inherit fields and methods from another, promoting code reusability and a hierarchical structure.

**Implementation in the System**

1. **Abstract User Class**:  
    The User class serves as a blueprint for different user types. It contains common attributes and methods that both Admin and Customer classes share.

**User Class**:

public abstract class User {

protected String name;

protected String email;

public User(String name, String email) {

this.name = name;

this.email = email;

}

public abstract void displayInfo();

}

1. **Admin Class (Inherits User)**:  
    The Admin class inherits the User class and implements the abstract displayInfo() method. It also provides additional functionalities such as loading and displaying products.

**Example**:

public class Admin extends User {

public Admin(String name, String email) {

super(name, email);

}

@Override

public void displayInfo() {

System.out.println("Admin: " + name);

}

public void loadProducts() {

// Logic to load products from database

}

}

1. **Customer Class (Inherits User)**:  
    The Customer class also inherits the User class and provides customer-specific features, such as viewing the cart and updating total prices.

**Example**:

public class Customer extends User {

private double totalPrice;

public Customer(String name, String email) {

super(name, email);

}

@Override

public void displayInfo() {

System.out.println("Customer: " + name);

}

public void viewCart() {

// Logic for viewing cart

}

}

**Class Hierarchy**:

User (Abstract)

|

|-- Admin

|

|-- Customer

**Benefits**:

* Promotes code reuse by sharing common functionality.
* Simplifies the structure of the codebase.

**3. Polymorphism**

**Polymorphism** allows the same method to take multiple forms, achieved via **method overriding** and **method overloading**.

**Method Overriding**

Method overriding occurs when a subclass provides a specific implementation of a method already declared in its parent class.

**Example**:  
 Both Admin and Customer classes override the displayInfo() method defined in the User class:

@Override

public void displayInfo() {

System.out.println("Admin: " + name);

}

@Override

public void displayInfo() {

System.out.println("Customer: " + name);

}

**Method Overloading**

Method overloading occurs when multiple methods in the same class share the same name but differ in parameters.

**Example**:  
 In the **Product** class, setData() is overloaded:

public void setData(String name, double price, String rating, String imageURL, int stock) {

this.name = name;

this.price = price;

this.rating = rating;

this.imageURL = imageURL;

this.stock = stock;

}

public void setData(String name, double price, String rating, String imageURL) {

this.name = name;

this.price = price;

this.rating = rating;

this.imageURL = imageURL;

}

**Benefits**:

* Enables dynamic behavior at runtime (method overriding).
* Enhances code readability and usability (method overloading).

**4. Abstraction**

**Abstraction** hides the implementation details of a class and only exposes necessary functionalities, providing a clean and simplified interface.

**Implementation in the System**

1. **Abstract Classes**:  
    The User class is abstract and defines common properties and behaviors for all user types (Admin and Customer).

**Example**:

public abstract class User {

public abstract void displayInfo();

}

1. **Database Connection Abstraction**:  
    The DatabaseConnection class abstracts database connection logic so that other parts of the system do not need to know the implementation details.

**Example**:

public static Connection getConnection() {

try {

return DriverManager.getConnection(DB\_URL, USERNAME, PASSWORD);

} catch (SQLException e) {

System.err.println("Connection failed: " + e.getMessage());

return null;

}

}

1. **Controllers and UI Logic**:  
    Controllers such as BLoginPageController, Admin, and Customer encapsulate UI-specific logic while hiding complex operations, like database queries and cart logic.

**Benefits**:

* Reduces system complexity.
* Provides a clean and modular design.

### **Key Features**

**Easy-to-Use Menu and Search Functionality**

- Intuitive navigation to help users quickly find categories, products, and promotions.

- Advanced search options to filter results based on user preferences (e.g., price, popularity).

**User-Friendly Interface**

- Clean and modern design that enhances user experience and accessibility.

- Responsive layout that adjusts seamlessly across different devices (mobile, tablet, desktop).

**Product Categories**

- Clear categorization of products (e.g., fruits, vegetables, dairy, snacks) for easy browsing.

- Attractive visuals and detailed descriptions to enhance product discovery.

**Shopping Cart Overview**

- User-friendly cart interface displaying selected items, quantities, and total price.

- Options to modify item quantities or remove items easily.

**User Registration and Login**

- Simple sign-up process with secure login options for user accounts.

- Password recovery features to assist users in regaining access to their accounts.

**Multiple Payment Options**

- Support for various payment methods, including credit/debit cards, digital wallets, and cash on delivery.

- Secure payment processing to ensure user financial information is protected.

**Delivery Scheduling**

- Options for users to select preferred delivery times and methods, such as express or scheduled deliveries.

- Real-time tracking of delivery status to keep users informed.

**Order Management**

- Easy order placement, modification, and cancellation processes.

- Ability for users to track their order statuses and view order history.

**Real-Time Stock Updates**

- Notifications for product availability to keep users informed about stock levels.

- Automatic updates to prevent users from ordering out-of-stock items.

**Customer Page**

- Dedicated page for users to manage their profiles, view order history, and update preferences.

- Access to personalized recommendations based on past purchases.

**Admin Page**

- Administrative interface for managing products, orders, and user accounts.

- Tools for monitoring sales performance and inventory levels.

**Object-Oriented Programming (OOP)**

- Implementation of OOP principles to enhance code organization, maintainability, and reusability.

- Use of classes and objects to represent system components, promoting a modular architecture.

**Graphical User Interface (GUI)**

- Interactive GUI elements that facilitate user engagement and navigation.

- Visual feedback (e.g., animations, alerts) to improve user interaction and experience.

### **Testing and Results**

#### **7.1. Testing Methodology**

Manual testing is a valuable methodology that complements automated testing, especially for applications that require human judgment, flexibility, and a focus on user experience. By choosing manual testing for our Mini Online Grocery System, we can ensure a thorough evaluation of its functionality, usability, and overall quality before deployment. Manual testing is a software testing technique where testers execute test cases manually, without the assistance of automation tools. In this process, testers design and perform tests to identify defects, validate functionality, and ensure the software meets specified user requirements. Manual testing allows for a hands-on evaluation of the system's performance and usability, ensuring that the software is reliable, aligns with user needs, and operates as intended. Ultimately, this approach contributes to enhancing the overall quality of the software.

The testing methodology we did was handing out surveys. We made a Likert scale survey, rating their answers on a scale of 1-5, namely from strongly disagree to strongly agree. The researchers had 10 questions to gather quantitative data and another 3 questions to ask the respondents for suggestions for our qualitative data.

#### **7.2. Test Cases**

| Test Case ID | Test Case Description | Precondition | Input | Steps to Execute | Expected Result |
| --- | --- | --- | --- | --- | --- |
| TC-001 | User Registration | User is on the registration page. | Username: testuser  Email: testuser@example.com  Password: SecurePass123  Confirm Password: SecurePass123 | 1. Open the application.  2. Click "Sign Up".  3. Fill in the registration form.  4. Click "Sing Up". | User receives a confirmation message and is redirected to the login page. |
| TC-002 | User Login | User has a valid account. | Username: testuser  Password: SecurePass123 | 1. . Open the application.  2. Click "Login".  3. Enter username and password.  4. Click "Submit". | User is redirected to the homepage. |
| TC-003 | Browse Products | User is logged in. | N/A | 1. Navigate to the product catalog. | All products are displayed with correct details. |
| TC-004 | Add Item to Cart | User is browsing products. | Product Name: Apple | 1. Select a product.  2. Click ‘+’ icon | 1. Select a product.  2. Click "Add to Cart". |
| TC-005 | View Cart | Items are in the cart. | N/A | 1. Navigate to the shopping cart. | The cart displays all selected items. |
| TC-006 | Checkout Process | User has items in the cart. | Payment Method: Credit Card  Delivery Method: Standard Shipping | 1. Click on "Checkout".  2. Fill in payment details.  3. Select delivery method.  4. Click "Confirm Order". | User receives an order confirmation message. |

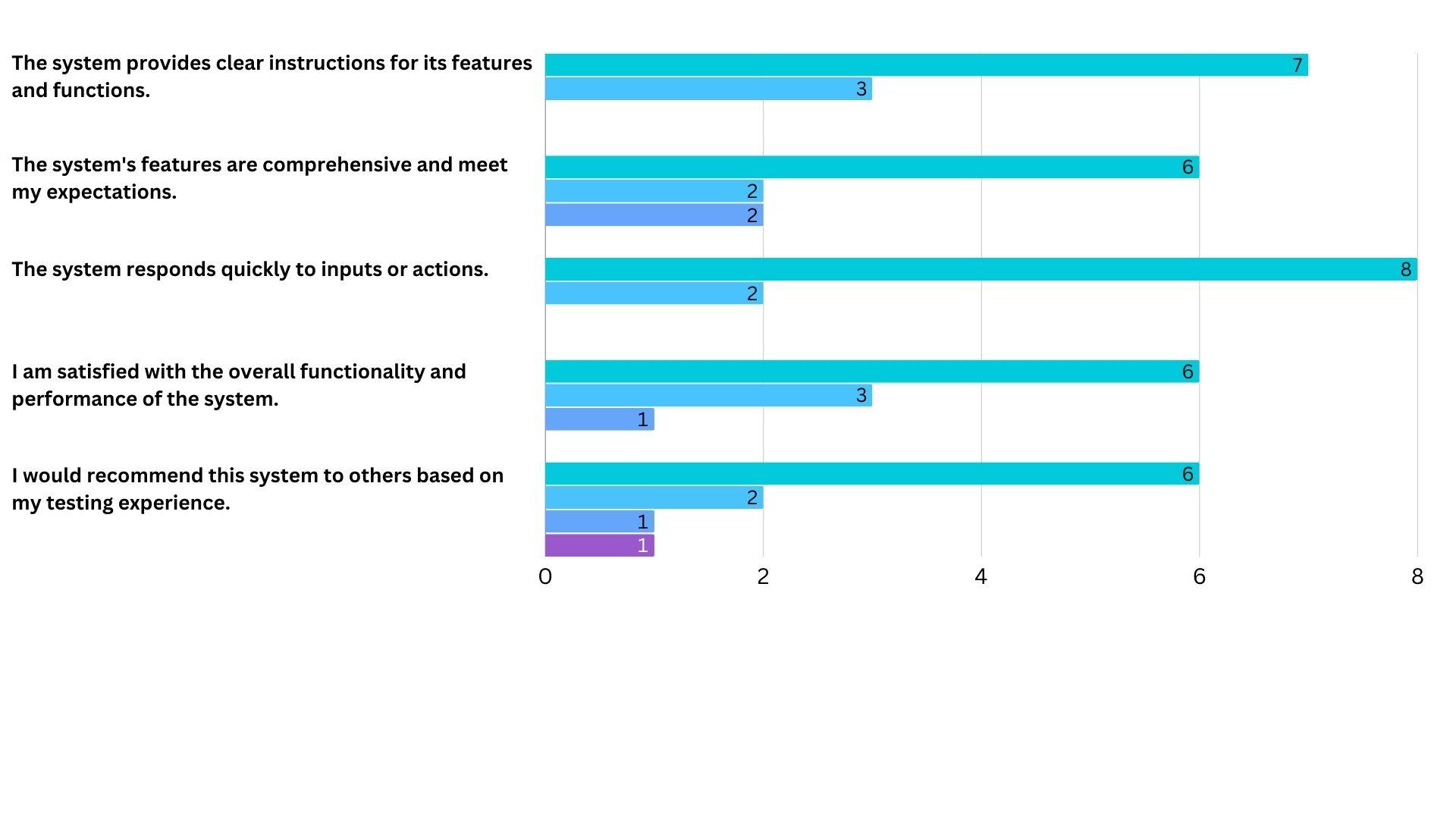
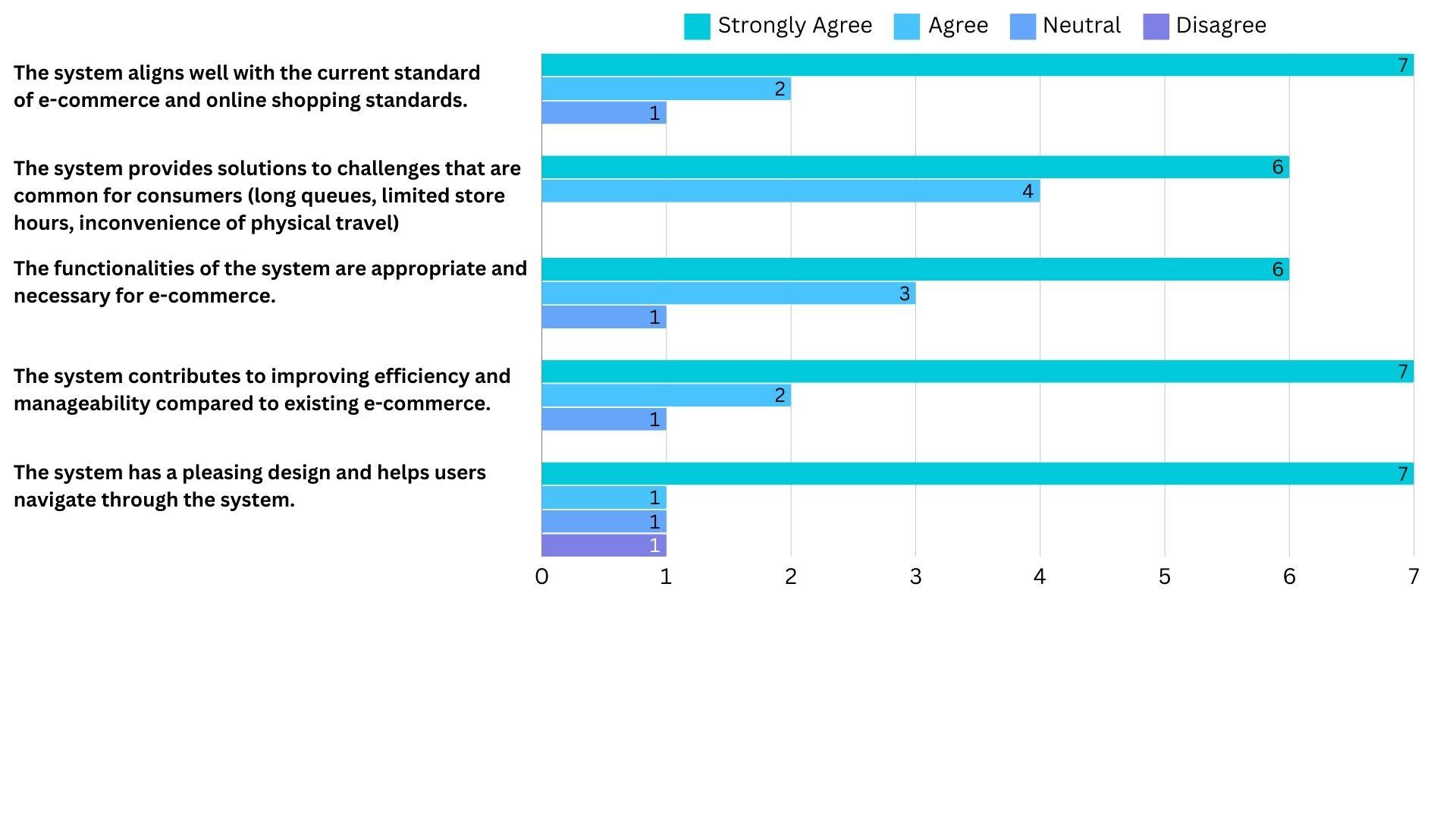
#### 

#### **7.3. Results**

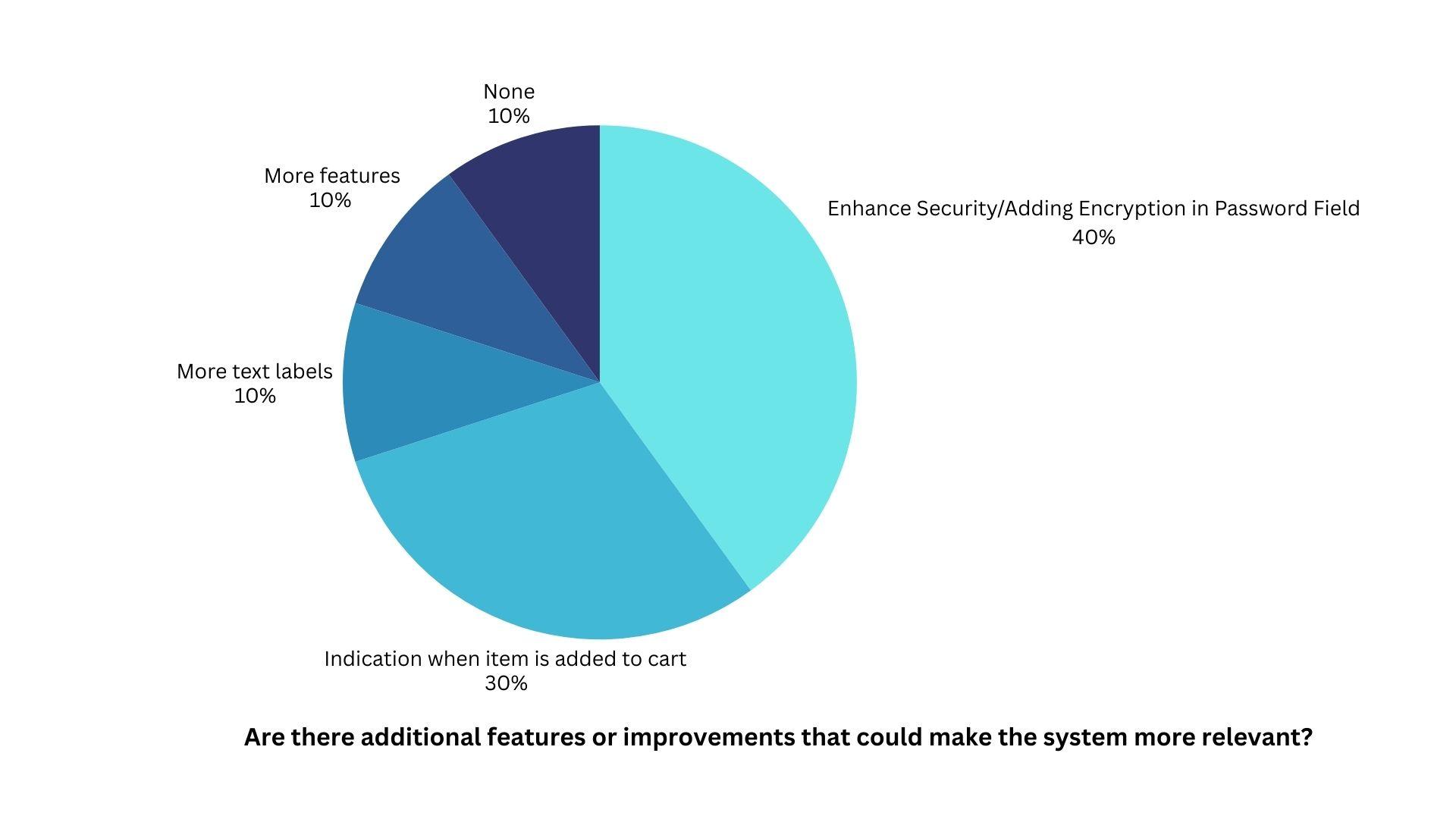
#### 

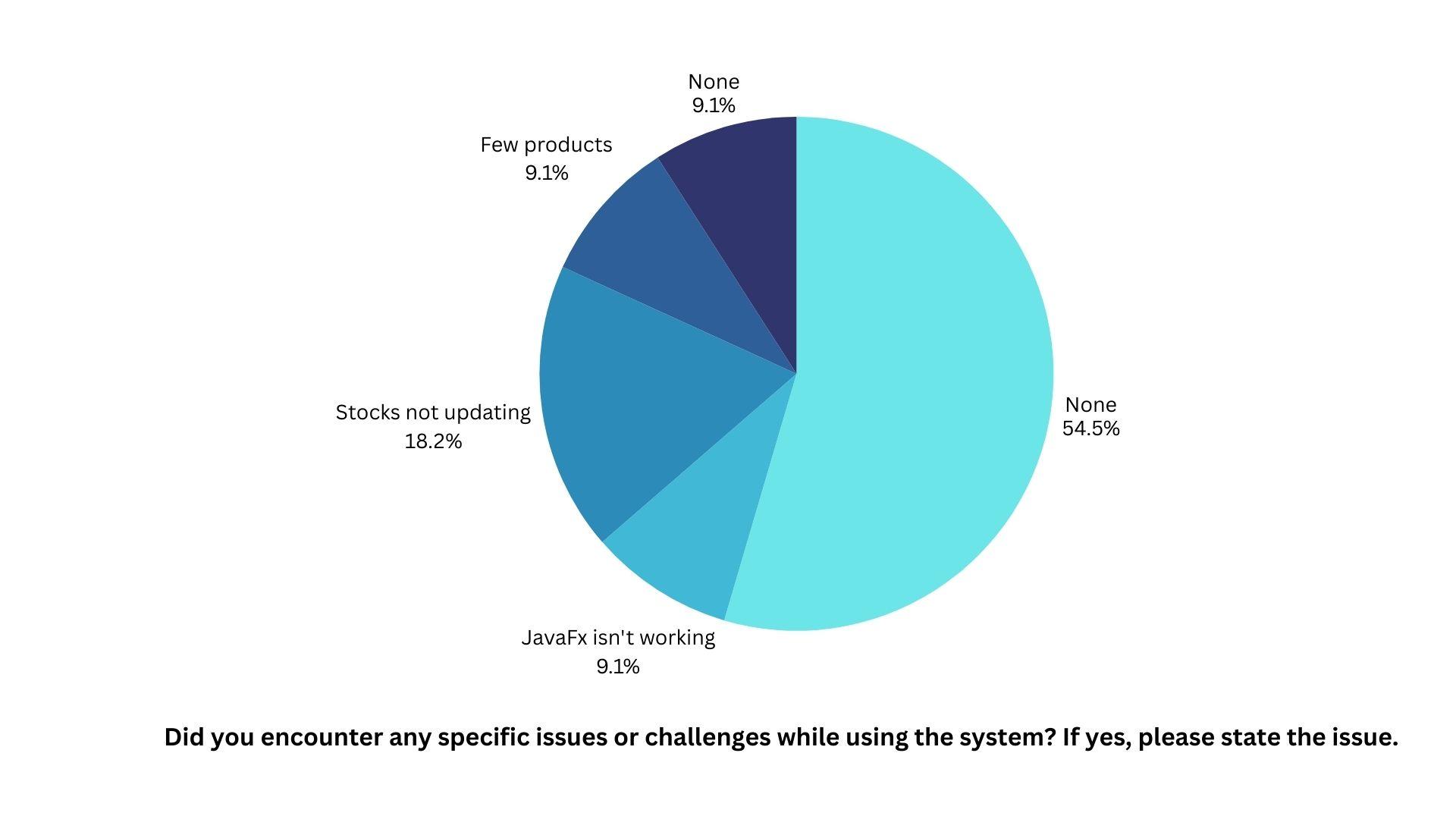
| **Test Case ID** | **Description** | **Status** | **Comments** |
| --- | --- | --- | --- |
| TC-001 | User Registration | Passed | Registration successful. |
| TC-002 | User Login | Passed | | User redirected to the homepage. | | --- | |
| TC-003 | Browse Products | Passed | | All products displayed correctly. | | --- | |
| TC-004 | Add Item to Cart | Passed | | Item added to the cart successfully. | | --- | |
| TC-005 | View Cart | Passed | | Cart displays correct items. | | --- | |
| TC-006 | Checkout Process | Passed | Order Confirmation Received and Printing of Receipt |

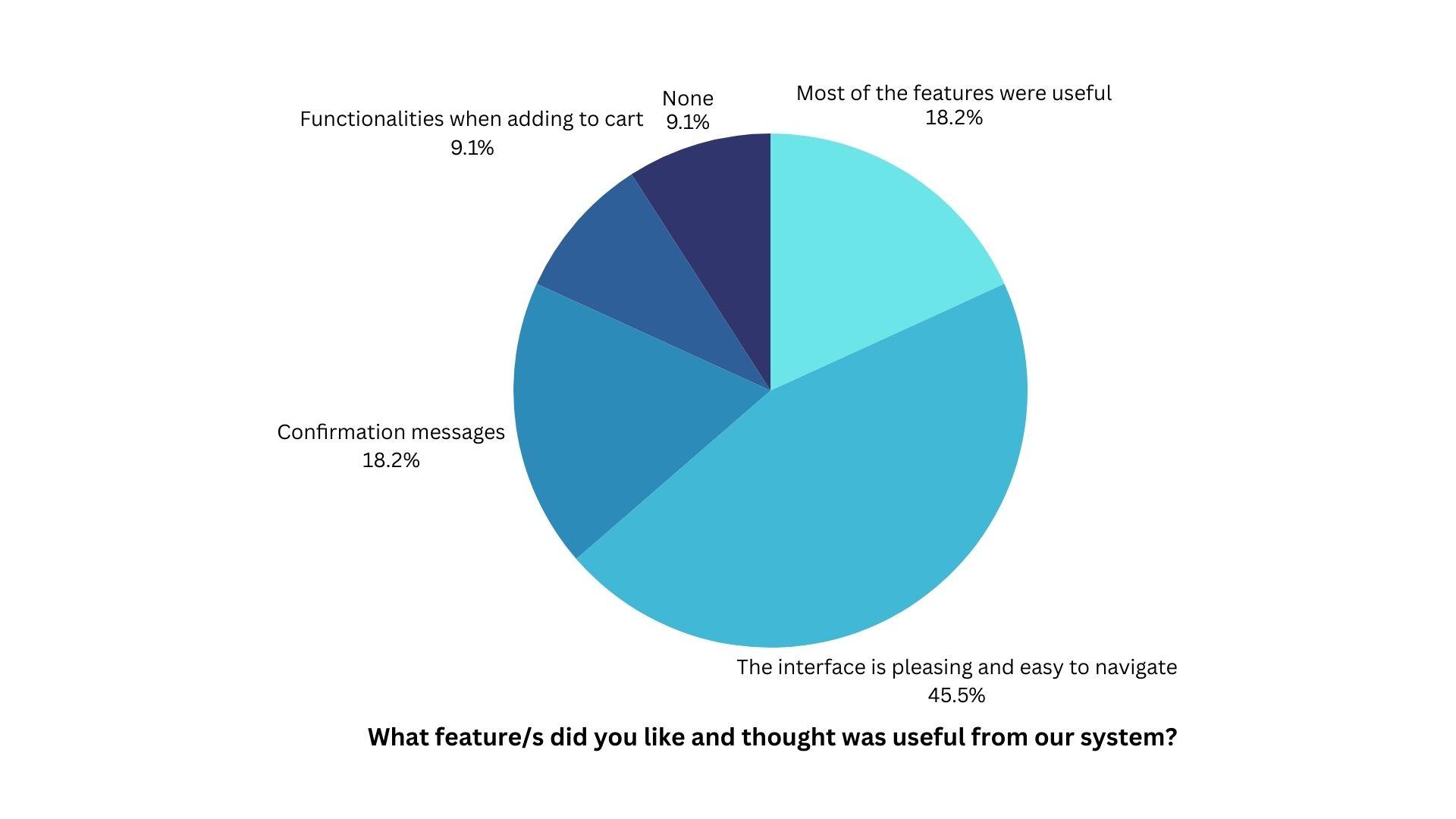
#### 



The analysis of user experience with the e-commerce system reveals a generally positive reception, with specific areas needing improvement. The system is well-aligned with current e-commerce standards and provides solutions to common consumer challenges such as long queues and limited store hours. The platform's functionalities are deemed appropriate and necessary, contributing to improved efficiency and manageability compared to existing e-commerce options. Users also find the system's design pleasing and easy to navigate, with clear instructions for its features. The system is responsive and comprehensive, meeting user expectations and leading to overall satisfaction and a willingness to recommend it. These aspects indicate that the platform is well-designed, functional, and user-friendly.

The user feedback on the e-commerce system, as shown in the sources, can be broken down into distinct categories based on the data presentation. First, the bar graph data suggests areas for improvement, with security(specifically, adding password encryption) being the most prominent concern at 40%. Following this, users would like an indication when an item is added to the cart (30%). These two issues seem to be the biggest areas for improvement, whereas users also suggest adding more text labels and more features (both 10%).



The pie chart data is a mix of user feedback regarding additional features/improvements and issues encountered. In terms of improvements, the largest percentage (54.5%) of users had no suggestions for improvements, whereas 18.2% reported stocks not updating as an issue and 9.1% had issues with JavaFX and another 9.1% noted that there were few products. In terms of features that users liked, 45.5% noted that the interface was pleasing and easy to navigate, 18.2% reported that most of the features were useful and another 18.2% liked confirmation messages. These responses suggest that the system has a generally positive user experience, but some specific issues (stock updating, technical problems) need attention. Overall, this data indicates a generally positive user experience but highlights key areas, such as security and technical stability, that should be addressed to improve user satisfaction.

### **Conclusion**

### **Future Enhancements**

Based on user feedback data, several key areas for future enhancement of the e-commerce system have been identified. The most pressing issue is enhancing security by adding encryption to the password field, as suggested by 40% of users. This significant concern must be addressed immediately to build user trust and confidence in the platform. Additionally, a substantial number of users (30%) expressed a desire for an indication when an item is added to the shopping cart. Improving feedback mechanisms for user actions will enhance the overall shopping experience.

Furthermore, 18.2% of users reported issues with stock levels not updating, indicating that the system needs to be updated to ensure real-time inventory management. Addressing the JavaFX issues noted by 9.1% of users is also important for improving the technical stability and overall performance of the system. Additionally, some users (9.1%) mentioned that there were too few products available, suggesting that expanding the product range could improve user engagement.

While not as critical, there were also minor suggestions from users, including the need for more text labels (10%) and additional features (10%). In summary, the data reveals that while the system is generally well-received and has many positive attributes, future development efforts should prioritize enhancing security with password encryption, adding an indication when items are added to the cart, ensuring accurate stock updates, and improving technical stability. Addressing these areas will lead to a more robust, user-friendly, and reliable e-commerce system, while implementing minor improvements in text labels and new features would further enhance the platform.

### **References**

Ayoob, M. (2021). ABC Grocery: A case study on designing an online grocery shopping experience. Medium. https://jocro010.medium.com/abc-grocery-a-case-study-on-designing-an-online-grocery-shopping-experience-23ac1a2a01a1

El-Masri, M. (2011). E-grocer strategies: A case study. ResearchGate. https://www.researchgate.net/publication/221409636\_E-Grocer\_Strategies\_A\_Case\_Study

TechTarget. (n.d.). Object-oriented programming (OOP). https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

Bowers, A. (2023). What is object-oriented programming? freeCodeCamp. https://www.freecodecamp.org/news/what-is-object-oriented-programming/

JavaTpoint. (n.d.). JavaFX tutorial. https://www.javatpoint.com/javafx-tutorial

### Curriculum Vitae

### **Personal Information:**

Robert Jhon D. Aracena

Tawantawan, Baguio District,

Davao City, 8000

09912288994

r.aracena.545985@umindanao.edu.ph

**Educational Background:**

BSCS in University of Mindanao, Davao City  
2027 March

### 



**Personal Information:**

Annika Lois R. Dumalogdog

Elenita Heights Catalunan Grande Phase 2

Davao City, 8000

09274879391

a.dumalogdog.547692.umindanao.edu.ph

**Educational Background:**

BSCS in University of Mindanao, Davao City  
2027 March



**Personal Information:**

Jexel Lomarda

Elenita Heights Catalunan Grande Phase 2

Davao City, 8000

09274879391

j.lomarda.---.umindanao.edu.ph

**Educational Background:**

BSCS in University of Mindanao, Davao City  
2027 March

### 

### 

### 

### 

### 