**Table of Contents**

[Introduction 2](#_Toc188977207)

[Background Information of the Tuck Shop 2](#_Toc188977208)

[Mission 2](#_Toc188977209)

[Vision 2](#_Toc188977210)

[Current System 2](#_Toc188977211)

[Alternative Solutions 3](#_Toc188977212)

[Proposed System 3](#_Toc188977213)

[Planning 3](#_Toc188977214)

[Key Challenges Identified: 4](#_Toc188977215)

[Non-Functional Requirements 4](#_Toc188977216)

[Software Requirements 5](#_Toc188977217)

[Use Case Diagram 5](#_Toc188977218)

[Actors 6](#_Toc188977219)

[Use Cases 6](#_Toc188977220)

[Admin Use Cases: 6](#_Toc188977221)

[Primary and Secondary Scenario 7](#_Toc188977222)

[Class Diagram 10](#_Toc188977223)

[Relationship 10](#_Toc188977224)

[CRC (Class-Responsibility-Collaboration) Cards 11](#_Toc188977225)

[Sequence And Collaboration diagram 12](#_Toc188977226)

[Sequence diagram 13](#_Toc188977227)

[Collaboration Diagram 13](#_Toc188977228)

[Activity Diagram 13](#_Toc188977229)

[Screenshot 15](#_Toc188977230)

[Time Schedule (Grantt Chart) 20](#_Toc188977231)

[Costs and Benefits Analysis 20](#_Toc188977232)

[Expected Benefits: 20](#_Toc188977233)

[Challenges and solutions 20](#_Toc188977234)

[Recommendations 21](#_Toc188977235)

[Conclusion 21](#_Toc188977236)

[References 22](#_Toc188977237)

# Introduction

Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects, which combine data and behavior. By encapsulating related data and functions, OOP fosters modularity, reusability, and scalability in software development. Central to OOP are four key principles: encapsulation, inheritance, polymorphism, and abstraction. These principles allow developers to create robust and flexible systems, enabling efficient management of complex applications. In the context of the Tuck Shop Food Ordering System, OOP provides a structured approach to modeling entities like customers, orders, menu items, and receipts, ensuring seamless interactions and system maintenance.

Background Information of the Tuck Shop  
The tuck shop is a small retail store within a school, college, or workplace that provides students, staff, and visitors with snacks, drinks, and light meals. Traditionally, orders are taken manually, which often leads to inefficiencies such as long queues, order mismanagement, and incorrect transactions. As the demand for a faster and more organized ordering system grows, there is a need to modernize the tuck shop’s operations by integrating technology.

# Mission

The mission of the Tuck Shop Food Ordering System is to enhance customer experience by providing a fast, accurate, and user-friendly digital ordering solution. By leveraging technology, the system aims to reduce waiting time, minimize human errors, and ensure smooth tuck shop operations.

Vision

The vision is to become a fully automated and efficient tuck shop service provider, offering seamless food ordering experiences through an integrated digital system that benefits both customers and shop management.

# Current System

The tuck shop currently operates on a manual order-taking and cash-based payment system. Customers place orders verbally, and staff manually records them on paper. Some of the key issues include:

> Long queues and slow service

> Increased chances of human error in order processing

> Difficulty in tracking inventory and sales

> Lack of an efficient payment and record-keeping system

Alternative Solutions

Several alternative solutions can be considered to improve the tuck shop's ordering process:

1. Mobile Application – A dedicated app where students can place orders and make payments.

2. Self-Service Kiosks – Touchscreen kiosks installed in the tuck shop for self-ordering.

3. Automated Vending Machines – Pre-packed meals and snacks dispensed through vending machines.

4. Online Website for Orders – A web-based food ordering system linked to the tuck shop inventory.

Proposed System  
  
The proposed Tuck Shop Food Ordering System is a web-based application integrated with a database and AI-driven recommendation features. It will allow students and staff to:

1. Place orders online through a web application.

2. Receive estimated preparation times.

3. Choose cashless payment options (mobile payments, digital wallets).

4. Receive personalized food recommendations based on past orders.

5. Provide order tracking and inventory management for the tuck shop.

This system will reduce errors, speed up service, and improve customer satisfaction.

Planning

The project will be executed in six phases:

1. Requirement Gathering – Identify user needs and system specifications.

2. Design – Develop UI/UX designs and system architecture.

3. Development – Implement front-end, back-end, and database systems.

4. Testing – Conduct functionality and security tests.

5. Deployment – Launch the system for real-time use.

6. Maintenance – Regular updates and support.

Analysis and Requirement Gathering  
  
Stakeholders such as students, tuck shop employees, and management were consulted to understand their needs.  
  
Key Functional Requirements:

- User authentication and role-based access control.

- Order placement, modification, and tracking.

- Payment integration with mobile money services.

- Inventory management for tuck shop staff.

Key Challenges Identified:

- Internet connectivity issues

- Training staff on the new system

- Ensuring data security for transactions

Non-Functional Requirements

Scalability – The system should support multiple users without lag.

Security – User authentication, encrypted transactions, and role-based access.

Usability – Easy-to-navigate UI for both customers and shop staff.

Reliability – System uptime should be at least 99%.

# Software Requirements

Front-End – HTML, CSS, JavaScript (React.js or Vue.js)

Back-End – Node.js, Python (Django/Flask)

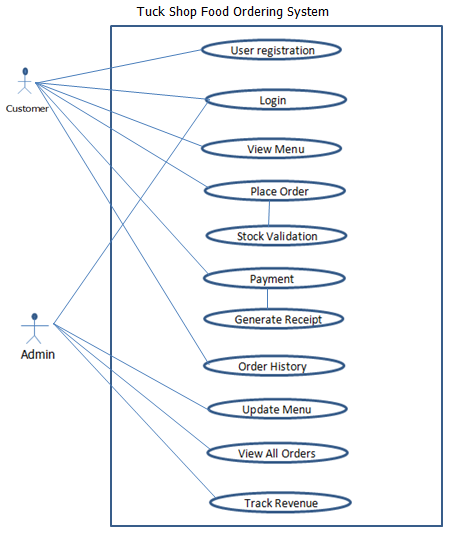
Database – MySQL or Firebase

AI Module – Python (Scikit-learn, TensorFlow for recommendations)

Hosting – Cloud-based deployment (AWS, Firebase, or Heroku)

# Use Case Diagram

The Use Case acts as a high-level blueprint for the Tuck Shop Food Ordering System by detailing interactions between actors (users) and the system’s functionalities. It defines the specific goals that both Customers and Admins aim to achieve while interacting with the system.



# Actors

1. Customer: A person who interacts with the system to register, log in, place orders, and view order history.

2. Admin: A person responsible for managing menu items, tracking revenue, and viewing orders.

3. System: The Tuck Shop Food Ordering System processes the user’s requests.

Use Cases

1. User Registration: Allows new customers to register an account.

2. Login: Authenticates registered customers and admin users.

3. View Menu: Displays the menu with items, prices, and stock.

4. Update Menu (Admin): Enables admins to add, edit, or remove menu items.

5. Place Order: Customers select items, specify quantities, and the system calculates the total cost.

6. Stock Validation: Validates the availability of items before confirming an order.

7. Payment: Processes payments through cash or card, and generates a receipt.

8. Order History: Allows customers to view their previous orders.

9. View All Orders (Admin): Allows admins to view all orders placed by customers.

10. Track Revenue (Admin): Allows admins to calculate and view total revenue.

# Admin Use Cases:

Login:

Admins log in using unique credentials to access advanced system functionalities. The purpose is to protect sensitive admin operations by restricting acess to authorized user

Update Menu:

Admins can add new menu items, update existing item details (price and stock) or remove items. Its purpose is to ensure that menu reflects real-time stock and price changes, improving customer satisfaction.

View All orders:

Admins can view details of all orders placed by customers. Its purpose is to help admins monitor customer activity, resolve disputes and maintain a record of business transactions.

Track Revenue:

The system aggregates revenue from all completed orders and provides reports for admins. The purpose is to enable admins to monitor the financial performance of the tuck shop.

# Primary and Secondary Scenario

**User Registration and Login**

Primary Scenarios:

1. The customer accesses the system.

2. The system prompts for a username, password, a

3. The customer enters valid credentials and registers successfully.

4. The customer uses the registered credentials to log in.

5. The system validates the credentials and grants access to the dashboard.

Secondary Scenarios:

1. The customer provides incomplete or invalid credentials during registration.

The system displays an error message and prompts the user to re-enter the details.

2. The customer forgets the login credentials.

The system provides a password reset option.

3. Multiple failed login attempts lock the account temporarily.

The system notifies the user of account lockout and provides a recovery option.

**Menu Management**

Primary Scenarios:

1. The customer requests to view the menu.

2. The system fetches and displays the list of items, prices, and stock levels.

3. The admin updates the stock levels or prices as needed.

4. The system reflects the updated information in real-time.

Secondary Scenarios:

1. The menu is unavailable due to a server issue.

The system notifies the customer of technical difficulties.

2. An admin provides invalid input while updating stock or prices.

The system rejects the update and prompts for valid input.

3. An item is out of stock.

The system notifies the customer and disables ordering for that item.

**Order Placement**

Primary Scenarios:

1. The customer selects items from the menu and specifies quantities.

2. The system checks stock availability for each item.

3. If stock is sufficient, the items are added to the order, and the total price is calculated.

4. The customer confirms the order, and the system processes it.

Secondary Scenarios:

1. The customer selects an invalid quantity (e.g., exceeding stock or zero quantity).

The system displays an error and prompts the customer to adjust the quantity.

2. Stock becomes unavailable during the ordering process.

The system removes the unavailable item from the order and notifies the customer.

3. The customer cancels the order before confirmation.

The system discards the order without making any changes.

**Payment Management**

Primary Scenarios:

1. The customer selects a payment method (cash or card).

2. The system processes the payment.

3. Upon successful payment, the system generates and displays a receipt.

Secondary Scenarios:

1. The payment fails due to card issues (e.g., insufficient funds, invalid card details).

The system notifies the customer of the issue and prompts for an alternative payment method.

2. The customer cancels the payment.

The system aborts the transaction and resets the order status.

3. Cash payment exceeds the total price (e.g., overpayment).  
The system calculates the change and notifies the customer.

**Order History**

Primary Scenarios:

1. The customer requests to view their order history.

2. The system retrieves and displays all past orders, including dates, items, quantities, and total prices.

Secondary Scenarios:

1. The customer has no past orders.

The system displays a message indicating no history is available.

2. A specific order's details fail to load due to a system error.

The system notifies the customer and prompts them to retry later.

**Admin Functionality**

Primary Scenarios:

1. The admin logs in using admin credentials.

2. The system grants access to admin-specific features.

3. The admin adds new items to the menu, updates prices, or manages stock.

4. The admin views all customer orders and tracks the total revenue.

Secondary Scenarios:

1. An admin fails to log in due to invalid credentials.

- The system prompts the admin to re-enter the credentials or reset the password.

2. An invalid or duplicate menu item is added.

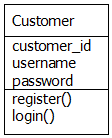
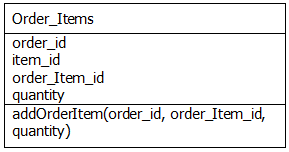
- The system rejects the addition and prompts for unique item details.

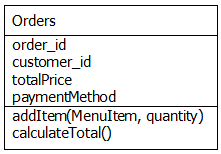
3. The revenue tracking report fails to load due to a system error.

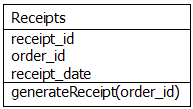
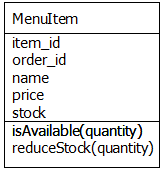
- The system notifies the admin and logs the issue for troubleshooting.

Primary scenarios represent the normal workflow of the system when all inputs are valid, and processes execute successfully. Secondary scenarios, on the other hand, cover exceptions, errors, or alternative paths that ensure robustness and improve the user experience. By handling both types of scenarios, the Tuck Shop Food Ordering System ensures a comprehensive and resilient design.

# Class Diagram



# Relationship

1. Customer → Orders: A customer can place multiple orders (1-to-many relationship).

2. Orders → Order\_Items: An order contains multiple menu items (1-to-many relationship).

3. MenuItem → Order\_Items: Menu items are part of orders (many-to-many relationship resolved via Order\_Items).

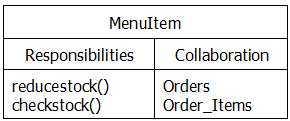
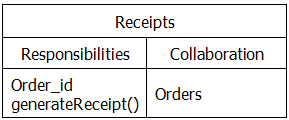
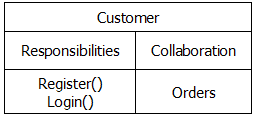
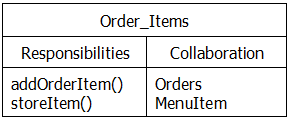
4. Orders → Receipts: Each order generates one receipt (1-to-1 relationship).

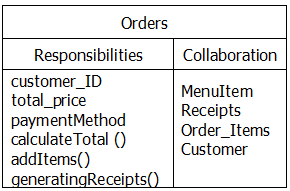
The class diagram illustrates the relationship between various entities in the system, highlighting their interactions and dependencies. The Customer class is linked to the Orders class, signifying that a customer can place multiple orders. The Orders class aggregates items from the MenuItem class through the intermediary Order\_Items class, which manages the specific items and their quantities for each order. The MenuItem class is responsible for maintaining stock levels and checking availability. Once an order is completed, the Orders class is associated with the Receipts class to generate and store payment confirmations. The Order\_Items class acts as a bridge between the Order and MenuItem classes, ensuring seamless tracking of ordered items. These relationships enable the system to function cohesively, integrating customer actions, menu management, and payment processing. Each entity collaborates effectively to support the overall functionality of the tuck shop ordering system.

# CRC (Class-Responsibility-Collaboration) Cards

**CRC Model:**

The Class-Responsibility-Collaboration (CRC) cards provide an organized view of the system's classes, responsibilities, and their collaborations with other classes. The Customer class handles user-related responsibilities, such as registering and logging in, while collaborating with the Orders class to place orders. The Receipts class focuses on generating receipts for completed orders and collaborates with the Orders class to retrieve order details. The MenuItem class manages inventory through methods for checking and reducing stock, working closely with Order\_Items to track specific item details for orders. The Order\_Items class acts as an intermediary, storing item and quantities while collaborating with Orders and MenuItem. Lastly, the Orders class is central to the system, managing customer IDs, payment methods, total price calculation, and generating receipts while collaborating with all other classes. These CRC cards highlight the system's modular design and efficient collaboration between its components.

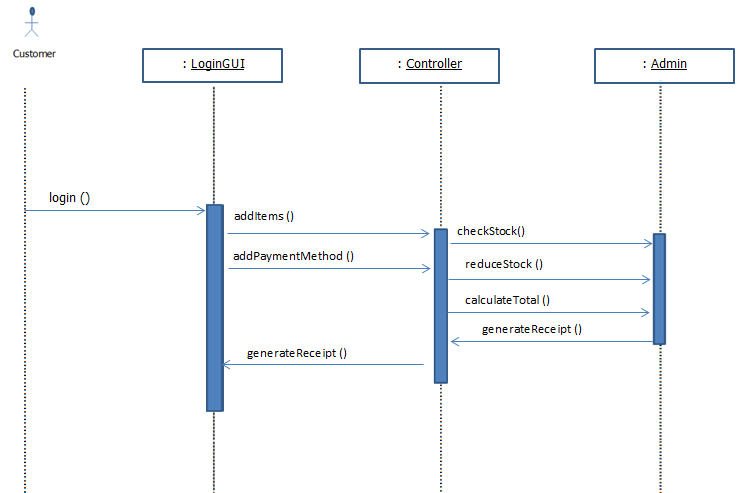




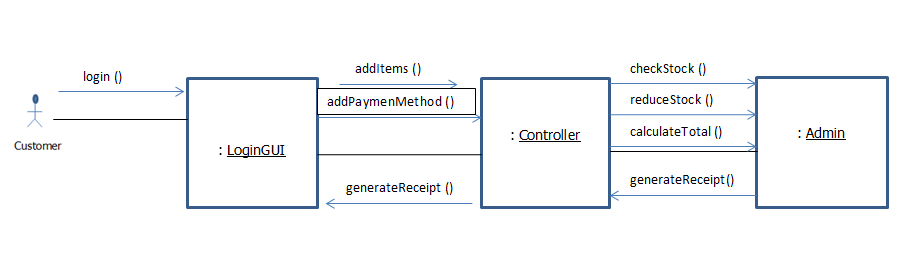
# Sequence And Collaboration diagram

This UML sequence diagram illustrates the interaction between a Customer, the LoginGUI, a Controller, and an Admin in a system, likely for an ordering or inventory management application. The process begins with the Customer logging in through the LoginGUI, which forwards actions such as adding items (addItems()) and payment methods (addPaymentMethod()) to the Controller. The Controller interacts with the Admin to manage backend operations like checking stock availability (checkStock()), reducing stock (reduceStock()), calculating the total cost (calculateTotal()), and generating a receipt (generateReceipt()). Finally, the generated receipt is sent back to the Customer via the LoginGUI, completing the transaction process. The diagrams below showcase the step-by-step flow of actions within the system.

# Sequence diagram



# Collaboration Diagram



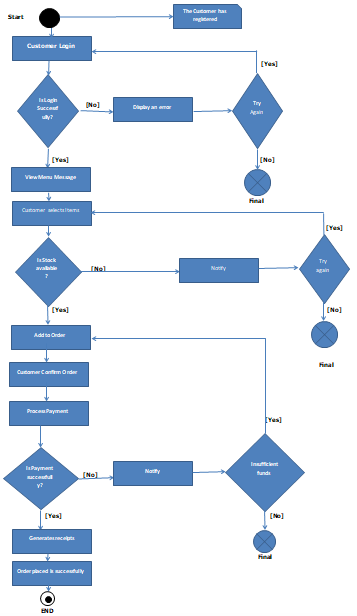
# Activity Diagram

Purpose

Activity diadram visually represents the flow of activities involved in placing an order within the Tuck Shop Food Ordering System. It identifies decision points like stock avilability and payment success, ensuring the systemhandles various scenarios effectively.

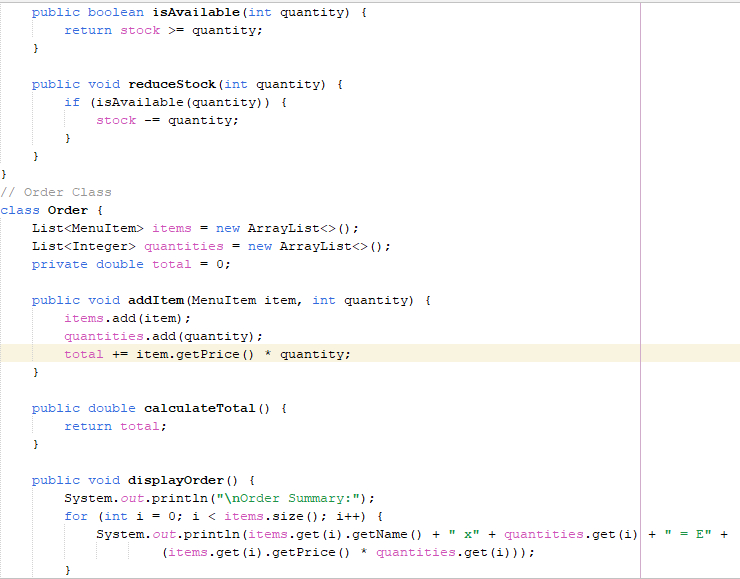
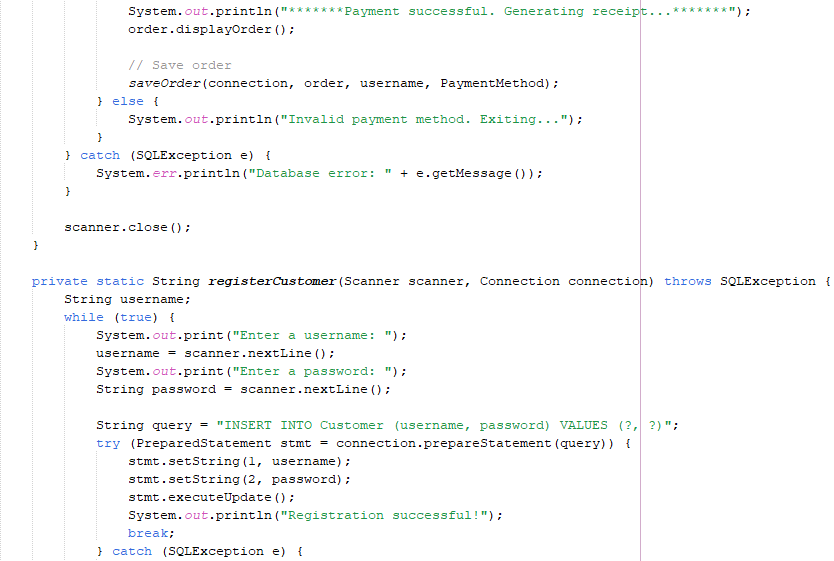
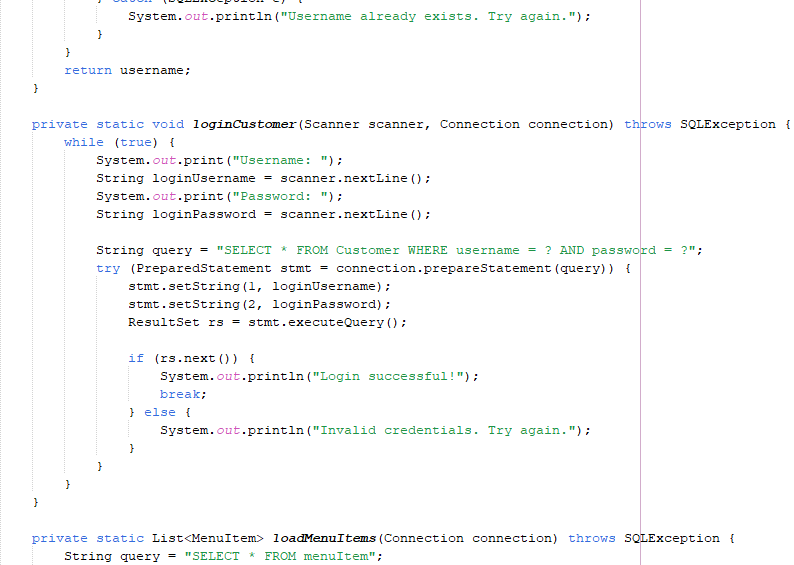
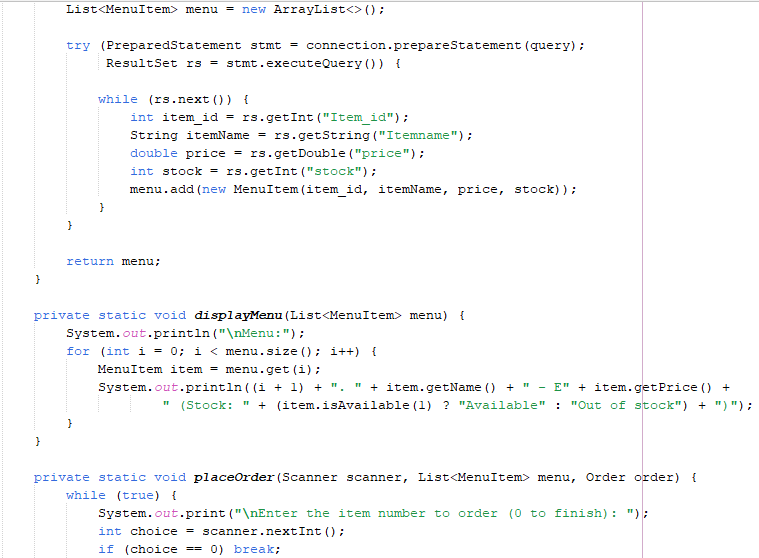
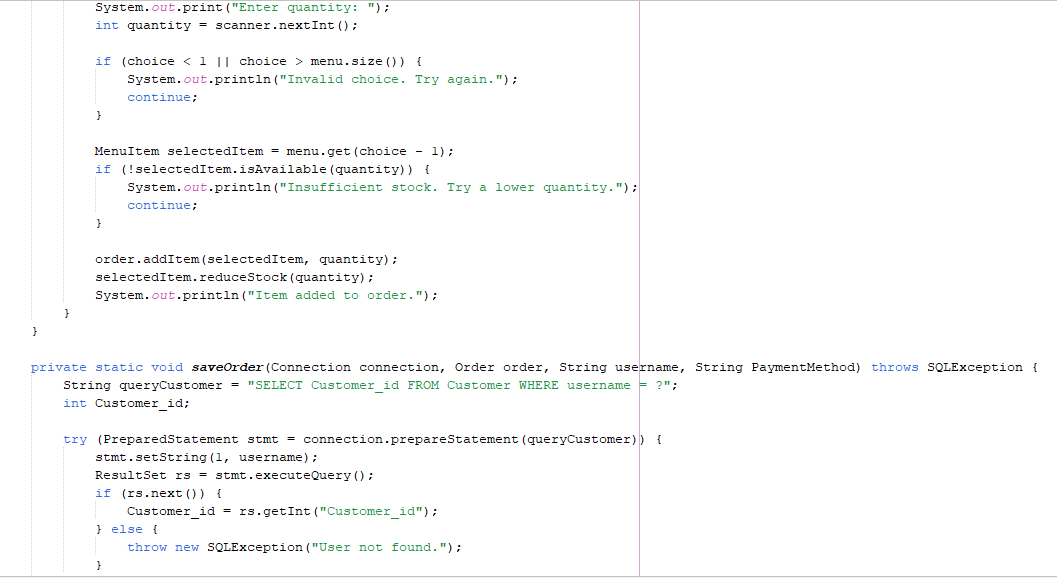
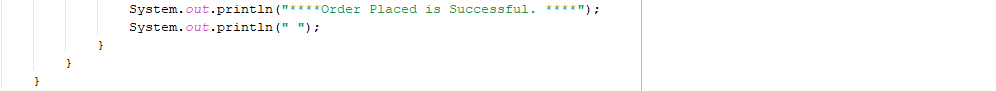
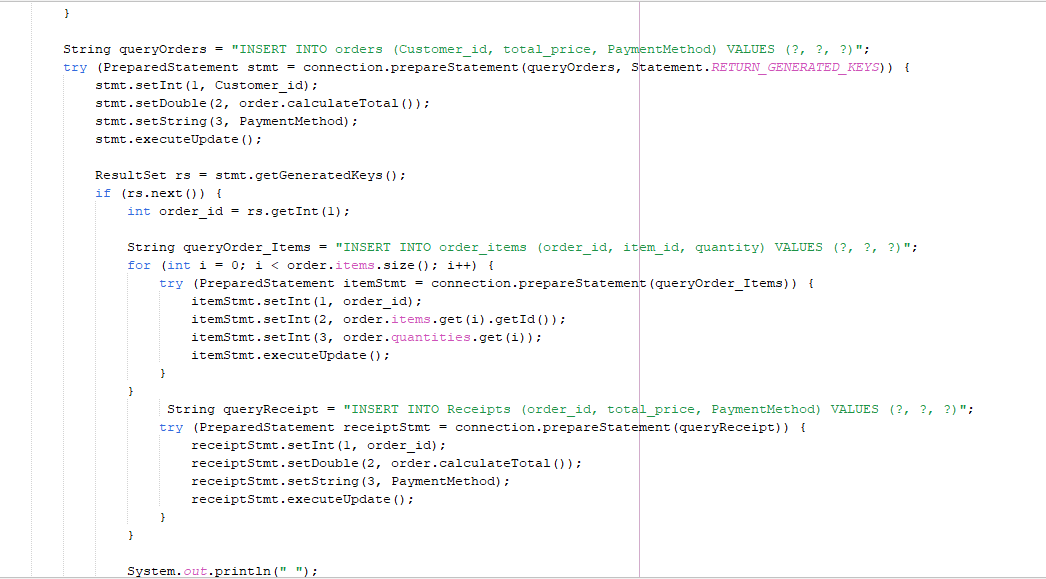
Flow

The process starts when the customer logs in. If login is successful, the customer views the menu, selects items and confirms the order. The system check stock availabilty, adds items to the order and processes payment. Depending on payment success, the system generates a receipt and confirms the order or notifies the customer of failure.

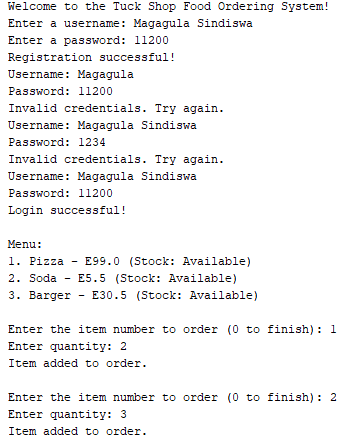
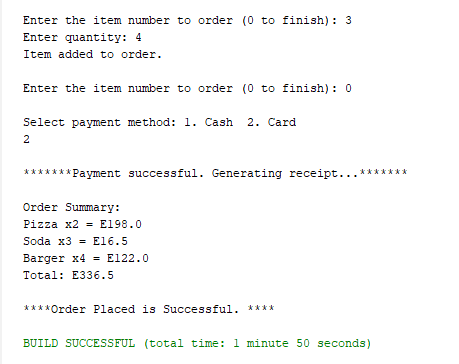


# Screenshot

**Code**

**Output**

# Time Schedule (Grantt Chart)

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Start Date | End Date | Duration |
| Requirement Analysis | Jan-12-2025 | Jan-14-2025 | 2 days |
| UI/UX Design | Jan-15-2025 | Jan-16-2025 | 1 day |
| System Development | Jan-17-2025 | Jan-25-2025 | 8 days |
| Testing | Jan-26-2025 | Jan-28-2025 | 2 days |
| Deployment | Jan-29-2025 | Jan-30-2025 | 2 days |

Costs and Benefits Analysis

Estimated Costs:

- Software Development: E500

- Hosting & Domain: E100

- Hardware (POS, Computers): E800

- Staff Training: E300

- Maintenance & Updates: E200

Expected Benefits:

- Increased Efficiency – Faster order processing reduces waiting time.

- Better Customer Experience – Personalized recommendations improve user satisfaction.

- Revenue Growth – Faster processing and AI-driven promotions increase sales.

- Reduced Errors – Automated system minimizes human mistakes.

# Challenges and solutions

1. Internet Connectivity

**Challenge:**

- The system relies on a stable internet connection to process orders in real time

- Poor network coverage or downtime can lead to delays in placing order and processing payments

**Solutions:**

- Implement an offline mode where users can place order, which will sync automatically once the internet is restore

- Optimize the system to consume the minimal bandwidth, making it functional even in areas with weak connection.

2. Inventory Management Errors

**Challenges:**

- Managing real-time inventory updates is crucial to avoid overselling or stock shortage

- Without proper tracking, customer may place orders for out-of-stock items

**Solutions:**

- Implement automated inventory tracking that updates stock levels immediately after an order is placed.

- Allow manual adjustments for unexpected stock changes

- Enable low-stock alerts to notify admins when items need restocking.

# Recommendations

To enhance the Tuck Shop Food Ordering System, several key recommendations should be implemented. Security can be strengthened by incorporating two-factor authentication (2FA) and regular system updates to protect user data. Integrating AI-based recommendations will improve customer experience by suggesting food items based on preferences and purchase history. Additionally, optimizing the payment process by offering multiple payment options and ensuring secure transactions will increase user trust. Improving inventory management with real-time stock tracking and low-stock alerts will help prevent shortages. Implementing these recommendations will ensure a secure, efficient, and user-friendly tuck shop ordering system.

# Conclusion

In conclusion, Object-Oriented Programming (OOP) is a powerful paradigm for designing and implementing scalable and maintainable software systems. By leveraging its principles, developers can create modular and reusable code that simplifies the complexity of real-world applications. For the Tuck Shop Food Ordering System, OOP facilitates the development of a robust system where objects like customers, menu items, and orders interact seamlessly. This approach not only improves code readability and maintainability but also enhances the overall efficiency and reliability of the system, making it adaptable to future requirements.

# References

1. Booch, G. (2007). Object-Oriented Analysis and Design with Applications. Addison-Wesley.

2. Balagurusamy, E. (2018). Object-Oriented Programming with C++. McGraw-Hill.

3. Lafore, R. (2002). Object-Oriented Programming in C++. Sams Publishing.

4. Stroustrup, B. (2013). The C++ Programming Language. Addison-Wesley.

5. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). Object-Oriented Modeling and Design. Prentice Hall.