SNACK-SHELL

Console Based Food Ordering

1. Project Overview:

1.1 Project Owner / Stakeholders:

- Pratham Pawar (Product Owner)
- Restaurant Owner (Stakeholder)
- User (Stakeholder)

1.2 Team Members / Roles:

- Developers
- Product Owner
- QA
- Scrum Master

1.3 Business Goals / Objectives:

- Provide a simple and intuitive command-line interface
- Lay the foundation for scalable enhancements

1.4 Timeline / Key Milestones:

- MVP Release - 09/07/2025
- MMP Release - 11/07/2025

1.5 Tools & Technologies Used:

- Programming Language – JAVA
- Development IDE – Eclipse IDE
- Testing – Manual Testing via Console Inputs

2. Product Vision/Roadmap:

2.1 Vision Statement:

To create a console-based, modular food ordering app that streamlines the ordering, payment, and invoicing process using clean software architecture principles.

2.2 Target Users / Personas:

Persona 1:

- Name: ABC
- Role: Customer
- Needs: Quick ordering, reliable invoice, clear delivery assignment
- Pain Points: Manual billing, poor payment experience

Persona 2:

- Name: XYZ
- Role: Admin
- Needs: Easy login access to menu
- Pain Points: Can't manage orders or billing directly

2.3 High-Level Features:

- Admin and customer login
- Order placement and menu viewing
- Discount logic and invoice generation
- Payment method integration
- Delivery partner assignment

2.4 Product Roadmap:

Epic	Description	Target Date
Epic 1	Build login, menu, and order modules	July 9, 2025
Epic 2	Integrate payment, delivery, and invoice	July 11, 2025

3. Requirements (Epics & User Stories)

3.1 Product Backlog:

Story ID	User Story	Priority
US-01	As an **admin**, I want to log in so I can manage the application	High
US-02	As a **customer**, I want to log in using my name and ID so that I can place an order	High
US-03	As a **customer**, I want to view the menu so that I can choose food items to order	High
US-04	As a **customer**, I want to select multiple items and quantities so that I can customize my order	High
US-05	As a **customer**, I want the system to apply a discount if my total is over ₹500	Medium

US-06	As a **customer**, I want to choose a payment method (Cash/UPI) so that I can pay conveniently	High
US-07	As a **system**, I want to randomly assign a delivery partner (Zomato or Swiggy)	Medium
US-08	As a **customer**, I want an invoice to be generated with my name and ID	High
US-09	As an **admin**, I want to view the full menu list after login	Medium
US-10	As a **developer**, I want to apply SOLID principles so the code is modular and maintainable	High
US-11	As a **customer**, I want to see a summary of my order before confirming payment	Medium
US-12	As a **customer**, I want to cancel the order before payment if needed	Low
US-13	As a **developer**, I want to log basic user actions (like login success/failure)	Low

3.2 Acceptance Criteria:

US-01: Admin Login

User Story: As an admin, I want to log in so I can manage the application.

Acceptance Criteria:

AC1: Given valid admin credentials, when the admin logs in, then they should access the admin panel.

AC2: Given invalid credentials, when the admin tries to log in, then an error message should be shown.

US-02: Customer Login

User Story: As a customer, I want to log in using my name and ID so that I can place an order.

Acceptance Criteria:

AC1: Given valid customer credentials, when the user logs in, then their session should begin.

AC2: Given a missing name or ID, when the user logs in, then they should be prompted to enter it.

US-03: View Menu

User Story: As a customer, I want to view the menu so that I can choose food items.

Acceptance Criteria:

AC1: Given the customer is logged in, when they request the menu, then all available items with prices should be shown.

US-04: Select Items and Quantities

User Story: As a customer, I want to select multiple items and quantities so that I can customize my order.

Acceptance Criteria:

AC1: Given the menu is displayed, when the customer selects an item and quantity, then it should be added to their order.

AC2: Given the customer adds multiple items, when the order is reviewed, then all items with their quantities should be visible.

US-05: Apply Discount

User Story: As a customer, I want a discount applied if the total is over ₹500.

Acceptance Criteria:

AC1: Given the total order amount is over ₹500, when checkout is initiated, then a flat discount should be applied.

AC2: Given the total is ₹500 or less, when checkout is initiated, then no discount should be applied.

US-06: Choose Payment Mode

User Story: As a customer, I want to pay using Cash or UPI.

Acceptance Criteria:

AC1: Given the customer is checking out, when they choose UPI, then UPI payment flow should be executed.

AC2: Given the customer chooses Cash, then cash payment should be acknowledged.

US-07: Assign Delivery Partner

User Story: As a system, I want to randomly assign a delivery partner.

Acceptance Criteria:

AC1: Given the payment is successful, when a delivery partner is assigned, then it should randomly select from the available options (Zomato or Swiggy).

AC2: The selected partner name should appear in the invoice.

US-08: Generate Invoice with Name & ID

User Story: As a customer, I want an invoice with my name and ID.

Acceptance Criteria:

AC1: Given the customer has placed an order and completed payment, when the invoice is printed, then it should display their name and ID.

AC2: The invoice should also include itemized list, total, discount, payment mode, and delivery partner.

US-09: Admin View Menu

User Story: As an admin, I want to view the full menu list after login.

Acceptance Criteria:

AC1: Given the admin is logged in, when they select "View Menu," then the full menu should be displayed.

US-10: Apply SOLID Principles

User Story: As a developer, I want to apply SOLID principles, so the code is modular and maintainable.

Acceptance Criteria:

AC1: Given the application is implemented, each class should follow a single responsibility.

AC2: Open/closed, interface segregation, and dependency inversion should be demonstrated through the design.

US-11: Order Summary

User Story: As a customer, I want to see a summary of my order before confirming payment.

Acceptance Criteria:

AC1: Given the customer has added items to cart, when they proceed to checkout, a summary of items and total amount should be displayed.

AC2: The customer should be able to go back and modify the order if needed.

US-12: Cancel Order

User Story: As a customer, I want to cancel the order before payment if needed.

Acceptance Criteria:

AC1: Given the customer has added items, when they choose to cancel before payment, then the order should be discarded.

AC2: A confirmation message should be displayed upon cancellation.

US-13: Log User Actions

User Story: As a developer, I want to log basic user actions (like login success/failure).

Acceptance Criteria:

AC1: Given a user attempts login, when credentials are submitted, then success/failure should be logged in console.

AC2: Logs should be informative and help trace user interactions.

3.3 Sprint Plan Overview

- **Sprint Duration:** Monday to Friday
- **Sprint Goal:** Deliver a working, console-based Mini Food Ordering App with login, ordering, discount, payment, invoice, and SOLID architecture.

3.3.1 Sprint Breakdown (Day-wise Plan):

Day	Focus Area	Deliverables
Monday	🔹 Finalize Requirements + Plan Sprint 🔹 Set up basic project structure	Product Backlog, Class Skeletons, Flow Diagram
Tuesday	🔹 Implement Login Module (Admin & Customer) 🔹 Menu Display Module	LoginService, User classes, Menu/MenuItem
Wednesday	🔹 Implement Order Logic + Discount Application 🔹 Add Order & OrderItem handling	Order, DiscountCalculator, ID/Customer link
Thursday	🔹 Payment Strategy + Delivery Assignment 🔹 Invoice Printer setup	PaymentProcessor, DeliveryService, InvoicePrinter

Friday	◆ Testing & Debugging ◆ Documentation & Final Submission	User testing, screenshots, Agile Report + Acceptance Criteria

4. Design & Architecture

4.1 UI/UX Designs:

Console-based. Menu options listed via CLI.

4.2 System Architecture:

4.2.1MAIN FEATURES (Functional Scope)

1. **Login System**
    a. Admin login (view/add menu – optional)
    b. Customer login (place order)
2. **Menu Display**
    a. Fixed list of food items (modifiable by Admin)
3. **Order Placement**
    a. Customer selects items and quantity
4. **Discount System**
    a. Apply flat discount if total > ₹500
5. **Payment Processing**
    a. Choose between Cash or UPI
6. **Delivery Assignment**
    a. Randomly assign from two partners
7. **Invoice Generation**
    a. Print invoice with:
        i. Item details
        ii. Total
        iii. Discount
        iv. Final amount
        v. Payment mode
        vi. Delivery partner
        vii. Customer Name & ID

4.2.2 COMPLETE CLASS LIST

1. User (interface)

- String getUsername()
- String getPassword()

2. Admin (implements User)

- Fields: username, password
- Used for future admin tasks (like menu management)

3. Customer (implements User)

- Fields: username, password, customerId, fullName
- Used in login and invoice

4. LoginService

- User login(String username, String password)
- Authenticates and returns Admin or Customer

5. Menu

- void displayMenu()
- List<MenuItem> getAvailableItems()

6. MenuItem

- Fields: String name, double price

7. Order

- void addItem(MenuItem item, int quantity)
- double getTotalAmountBeforeDiscount()
- List<OrderItem> getOrderItems()

8. OrderItem

- Fields: MenuItem item, int quantity
- double getItemTotal()

9. DiscountCalculator

- Field: IDiscount discountStrategy
- double applyDiscount(Order order)

10. PaymentProcessor

- Field: IPayment paymentMethod
- void processPayment(double amount)

11. DeliveryService

- IDeliveryPartner assignPartner()

12. InvoicePrinter

- void printInvoice(Order order, double discount, double totalAmount, String paymentMode, String deliveryPartnerName, Customer customer)

4.2.3 COMPLETE INTERFACE LIST

1. IPayment

- void pay(double amount)

a. CashPayment

- Implements IPayment

b. UPIPayment

- Implements IPayment

2. IDiscount

- double applyDiscount(double total)

a. FlatDiscount

- Implements IDiscount

b. NoDiscount

- Implements IDiscount

3. IDeliveryPartner

- String getName()

a. ZomatoPartner

- Implements IDeliveryPartner

b. SwiggyPartner

- Implements IDeliveryPartner

4.3 Folder Architecture:

Com.aurionpro.SnackShell_app

```
│
├── Main.java
├── model
│   ├── MenuItem.java
│   ├── OrderItem.java
│   └── Order.java
│
├── user
│   ├── User.java (interface)
│   ├── Admin.java
│   └── Customer.java
│
├── service
│   ├── LoginService.java
│   ├── Menu.java
│   ├── PaymentProcessor.java
│   ├── DiscountCalculator.java
│   ├── DeliveryService.java
│   └── InvoicePrinter.java
│
├── payment
│   ├── IPayment.java
│   ├── CashPayment.java
```

```
│   └── UPIPayment.java
│
├── discount
│   ├── IDiscount.java
│   ├── FlatDiscount.java
│   └── NoDiscount.java
│
└── delivery
    ├── IDelivery.java
    ├── ZomatoPartner.java
    └── SwiggyPartner.java
```

5.Testing & Quality

Test Strategy:

- Manual testing of flows
- Console output validation
- Unit testing for discount and delivery assignment

Test Cases:

TC-ID	Description	Expected Result
TC-01	Login with valid creds	Access granted
TC-02	Order > ₹500	Discount applied
TC-03	UPI payment selected	Payment processed correctly
TC-04	Invoice generated	Shows name, ID, total

Definition of Done:

- Code works end to end
- Invoice includes required data
- Test cases passed

Release Notes:

Version	Features Released	Date
v1.0	Full food order flow, invoice	July 12, 2025

Meeting Cadence:

- Daily: Standup (self-check-in)
- Sprint Planning: Monday
- Sprint Review & Retro: Friday

6.Lessons Learned & Retrospectives

Sprint #: 1

What went well:

- Design-first approach made coding easier
- Testing scenarios helped uncover edge cases

What didn’t go well:

- No persistence or GUI
- No time for cancel order feature

Action Items:

- Add GUI/DB in future version

Self Feedback:

- Clean architecture made future improvements easier

7. End Summary/Project Conclusion:

The Mini Food Ordering Console Application was successfully designed, developed, and delivered within a 1-week sprint using Agile methodology. The project simulated a real-world food ordering workflow, incorporating essential features such as:

- Admin and Customer Login
- Dynamic Menu Display and Order Management
- Discount Calculation for Orders above ₹500
- Payment Mode Selection (Cash/UPI)
- Random Assignment of Delivery Partner (Zomato/Swiggy)
- Invoice Generation with Customer Name and ID

By applying SOLID design principles and clean object-oriented architecture, the system was made modular, extendable, and easy to maintain. Interfaces were used to abstract payment and delivery logic, enabling future enhancements without modifying existing code.

All core user stories were implemented and verified through manual test cases, ensuring functional correctness and a smooth user experience. While advanced features like persistent storage, GUI integration, and admin dashboard were out of this sprint's scope, they have been identified as potential improvements for the next iteration.

Overall, the project successfully met its objectives, serving as a strong demonstration of Agile delivery, Java development best practices, and architectural clarity.