CH 7

Introduction

The Advanced Smart Inventory Stock Management System is an all-encompassing platform designed to optimize and manage key e-commerce functions, including user authentication, shopping cart management, secure payment processing, inventory control, and sales reporting. This system aims to create a seamless, secure, and efficient experience for both customers and administrators, addressing the challenges of modern e-commerce environments by providing accurate inventory tracking, real-time order updates, and reliable payment handling.

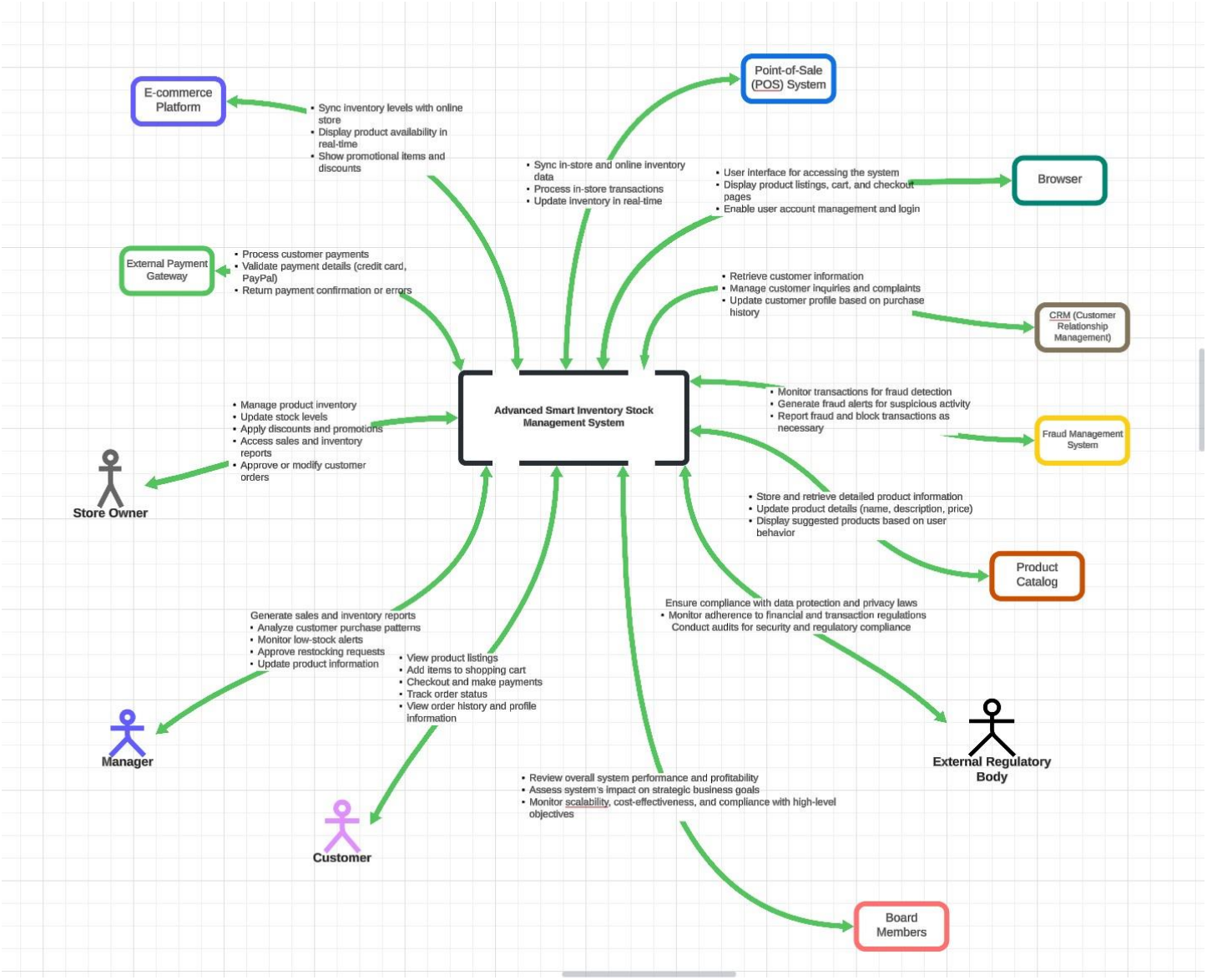
This document outlines the updates made to several core components of the system, including the System Context, Use Case Diagram, Sequence Diagram, State Diagram, and Interface Specification. Each of these updates is intended to improve the system's documentation, making it more comprehensive and easier to understand for developers, stakeholders, and users alike. By refining the design details and diagrams, this project ensures that the system architecture is well-defined, supports effective development practices, and maintains a high level of functionality and reliability.

The Advanced Smart Inventory Stock Management System is essential for businesses looking to manage stock levels, process orders efficiently, and provide a seamless shopping experience for customers. This updated documentation serves as a foundational guide to the system’s structure, interactions, and functionality, setting the stage for future enhancements and scalability.

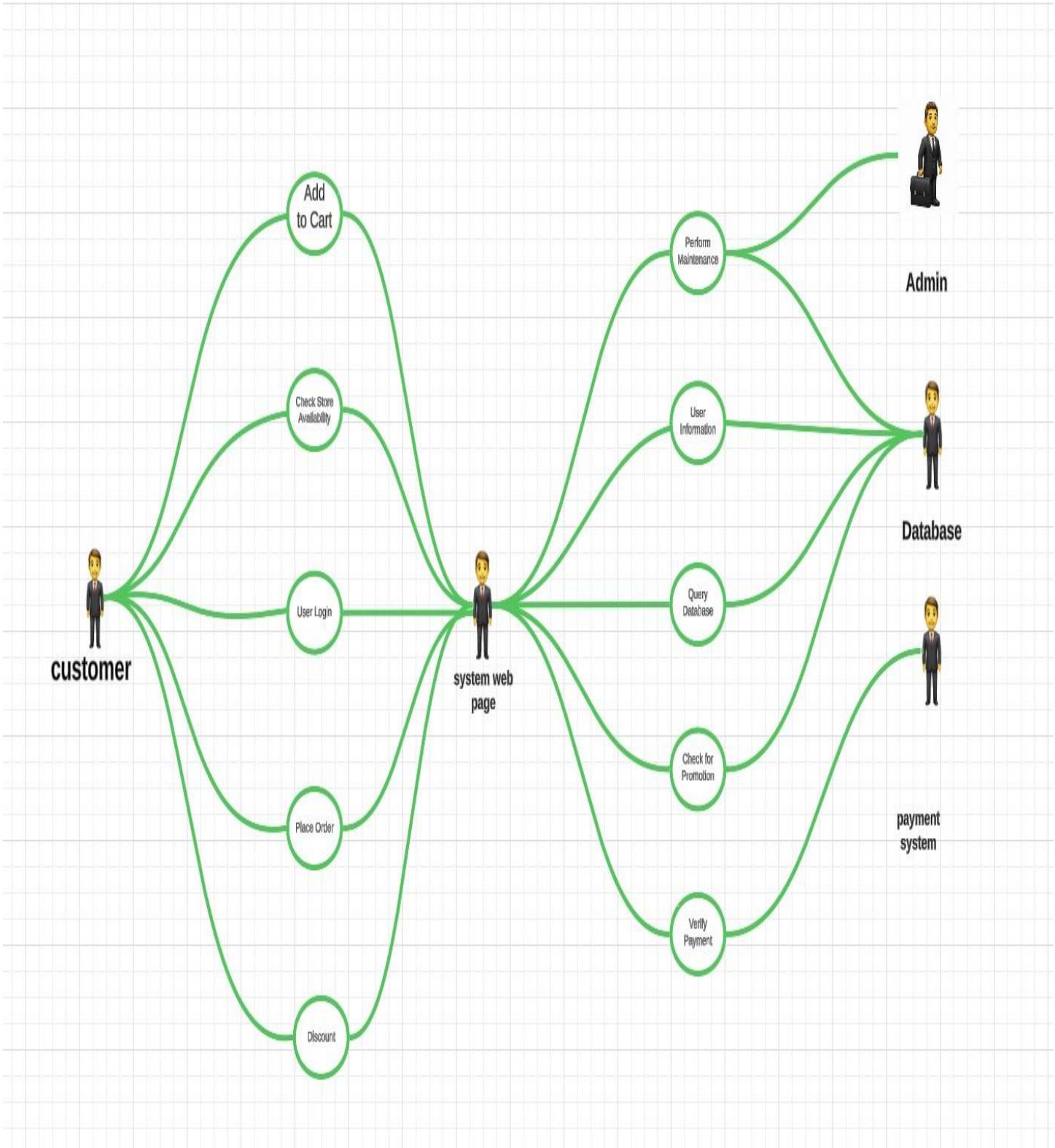
Purpose

The purpose of this week’s project work is to enhance The Advanced Smart Inventory Stock Management System by updating several key components to ensure a comprehensive and well-documented design.

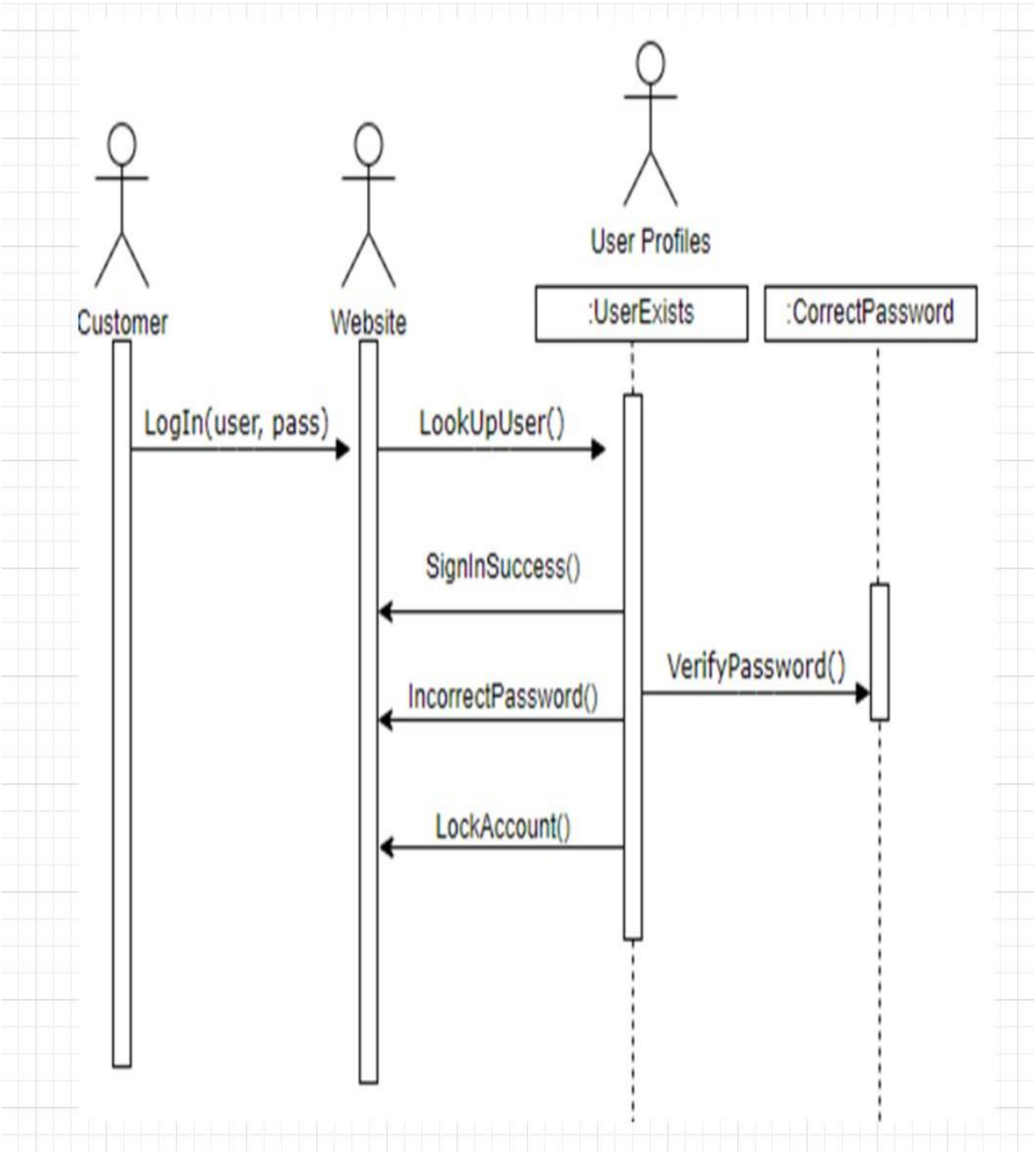
System context details and diagram



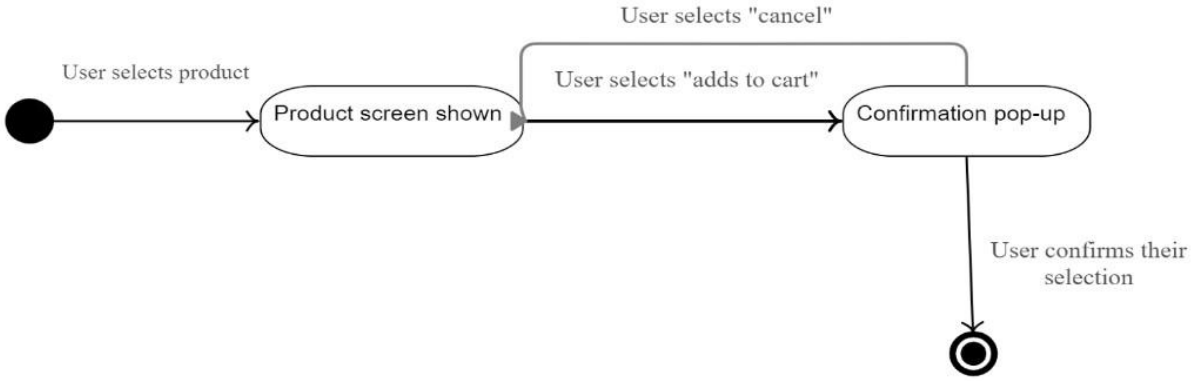
Updated use case diagram



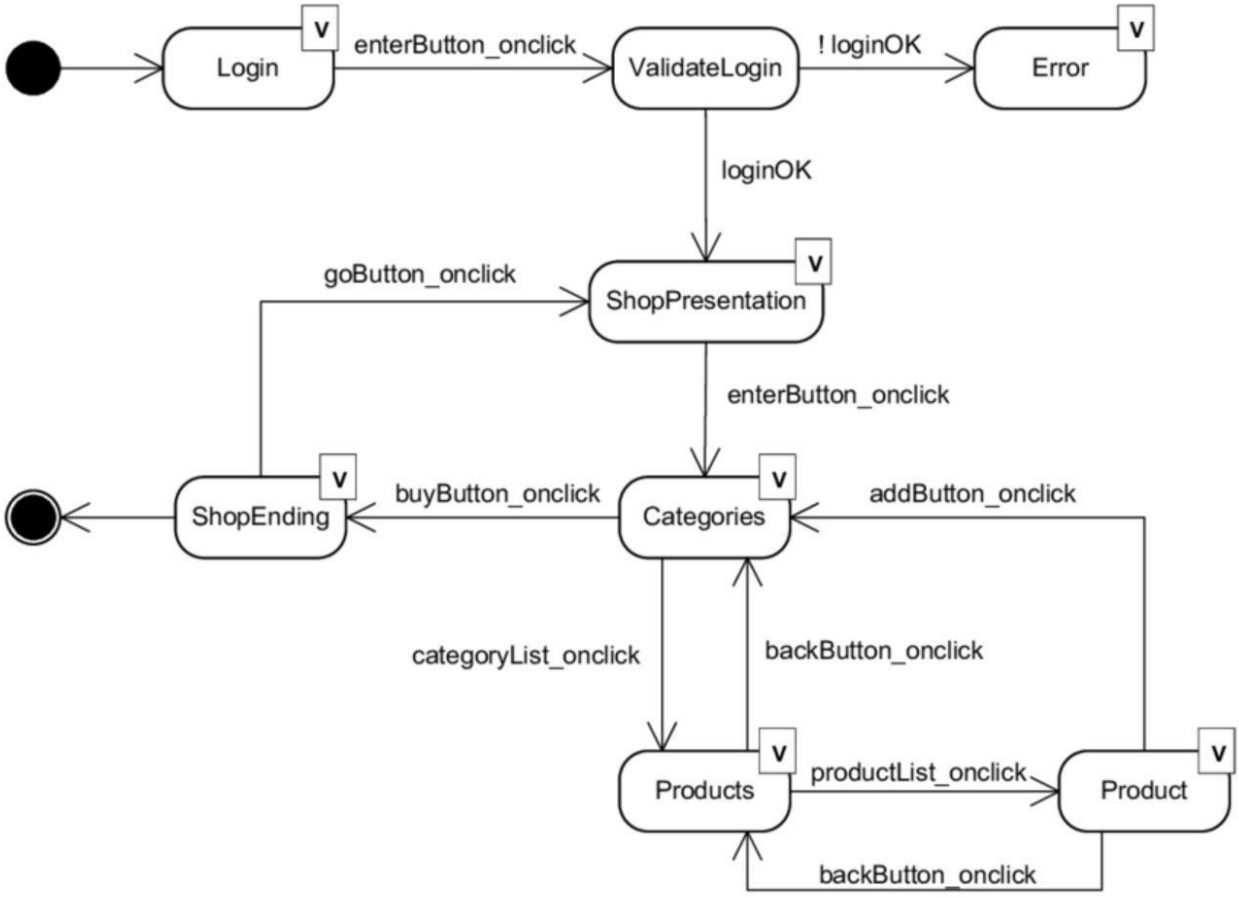
Login Sequence Diagram



Add to Cart State Diagram



User Navigation and Checkout State Diagram



User Account

Method	login(username, password)
Description	Authenticates user credentials for account access.
Parameters	username (String): The user's account name. password (String): The user's account password.
Return Value	Boolean: Returns true if credentials are valid, otherwise false.
Error Handling	InvalidCredentialsException: Raised if credentials are incorrect.
Example	login('john_doe', 'password123') → true

Method	registerUser(userInfo)
Description	Registers a new user in the system.
Parameters	userInfo (Object): Contains user details such as name, email, password, and address.
Return Value	String: Returns a success message or failure reason.
Error Handling	DuplicateUserException: Raised if the username or email already exists.
Example	registerUser({name: 'John', email: 'john@example.com', ...}) → 'Registration successful'

Shopping Cart

Method	addItem(itemID, quantity)
Description	Adds a specified item to the user's shopping cart.
Parameters	itemID (Integer): Unique identifier for the product. quantity (Integer): Number of items to add.
Return Value	String: Success message if the item is added.
Error Handling	OutOfStockException: Thrown if requested quantity exceeds available stock.
Example	addItem(101, 2) → 'Item added to cart successfully.'

Method	removeItem(itemID)
Description	Removes a specified item from the cart.
Parameters	itemID (Integer): Unique identifier for the product.
Return Value	String: Confirmation message on successful removal.
Error Handling	ItemNotFoundException: Thrown if the item does not exist in the cart.
Example	removeItem(101) → 'Item removed from cart successfully.'

Payment system

Method	processPayment(cartTotal, paymentDetails)
Description	Processes payment using the provided details.
Parameters	cartTotal (Decimal): Total amount to be paid. paymentDetails (Object): Contains payment method, card info, etc.
Return Value	Boolean: Returns true if payment is successful, otherwise false.
Error Handling	PaymentFailedException: Raised if payment could not be completed.
Example	processPayment(150.00, {method: 'Credit Card', ...}) → true

Method	refundPayment(orderID)
Description	Initiates a refund for the specified order.
Parameters	orderID (Integer): Unique identifier for the order.
Return Value	String: Confirmation of successful refund initiation.
Error Handling	InvalidOrderException: Thrown if the order ID is not valid.
Example	refundPayment(1001) → 'Refund initiated successfully.'

Inventory Control

Method	updateStock(itemID, quantity)
Description	Updates the stock level for a specified item.
Parameters	itemID (Integer): Unique identifier for the product. quantity (Integer): New stock level to set.
Return Value	String: Confirmation message after update.
Error Handling	DatabaseException: Thrown if there's an issue updating the stock in the database.
Example	updateStock(101, 50) → 'Stock updated successfully.'

Method	getStockLevel(itemID)
Description	Retrieves the current stock level for a specified item.
Parameters	itemID (Integer): Unique identifier for the product.
Return Value	Integer: Current stock level.
Error Handling	ItemNotFoundException: Raised if the item does not exist in inventory.
Example	getStockLevel(101) → 20

Sales Report

Method	generateReport(reportType, startDate, endDate)
Description	Generates sales or inventory report for a given period.
Parameters	reportType (String): Type of report (e.g., 'sales', 'inventory'). startDate (Date): Start of the reporting period. endDate (Date): End of the reporting period.
Return Value	Object: Report data, typically structured as a list or table.
Error Handling	ReportGenerationException: Raised if there's an issue generating the report.
Example	generateReport('sales', '2024-01-01', '2024-01-31') → ReportObject

Method	exportReport(reportID, format)
Description	Exports the specified report in the requested format (e.g., PDF, CSV).
Parameters	reportID (Integer): Unique identifier for the report. format (String): Format to export to (e.g., 'PDF', 'CSV').
Return Value	String: File path or download link to the exported report.
Error Handling	ExportException: Raised if the export fails.
Example	exportReport(100, 'PDF') → 'path/to/report.pdf'

Conclusion

In conclusion, the updates made to The Advanced Smart Inventory Stock Management System have enhanced the clarity, structure, and functionality of the system's documentation and design. By refining the System Context, Use Case Diagram, Sequence Diagram, State Diagram, and Interface Specification, this project has provided a more detailed and accurate representation of the system's architecture and interactions.

These improvements ensure that the platform is well-equipped to handle the complexities of e-commerce operations, such as user authentication, shopping cart management, payment processing, inventory control, and sales reporting. With a clear understanding of each component's role and interaction, developers can more easily implement and maintain the system, while administrators and stakeholders can rely on a robust, user-friendly solution for managing inventory and processing orders.