So

CS5002NI Software Engineering

35% Individual Coursework

AY 2024-2025

Credit: 30

Student Name:

Londonmet ID: Nikhil Bhandari

College ID: 23049041

Assignment Due Date: 12th May 2025

Assignment Submission Date: 12th May 2025

Submitted to: Rubin Thapa

Table of Contents

Tables of Figures

1. Introduction

Global Tech Corporation is developing a new and improved Inventory Management System (IMS) for warehouses in Nepal after their first attempt failed due to poor system design. The previous system lacked proper structure, making it difficult to maintain and scale. This time, they are focusing on Object-Oriented Analysis and Design (OOAD) to create a more organized and efficient system. The new IMS will include important features like secure login for admins and buyers, purchase order tracking with price comparison, automated sales and purchase reports, order and delivery management, product inventory control, and secure payment processing. By fixing past mistakes, this upgraded system aims to streamline warehouse operations, reduce costs, and improve customer satisfaction. The goal is to make inventory management smoother, faster, and more reliable for businesses in Nepal.

In this project we use UML diagram because it provides a clear way to visualize and design software systems before development begins. It helps teams communicate complex ideas through standardized diagrams, ensuring everyone understands the system's structure and behavior. We use Sequence diagram, activity diagram and Use case diagram in this project.

2. Work Breakdown Structure

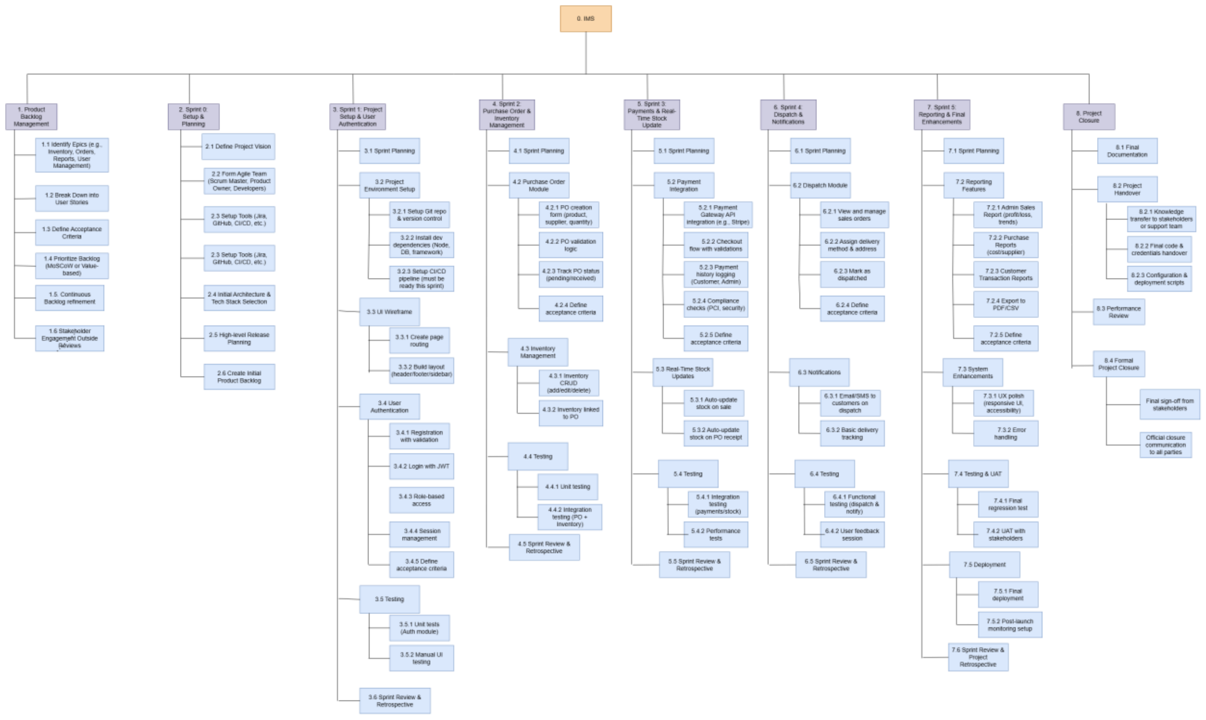
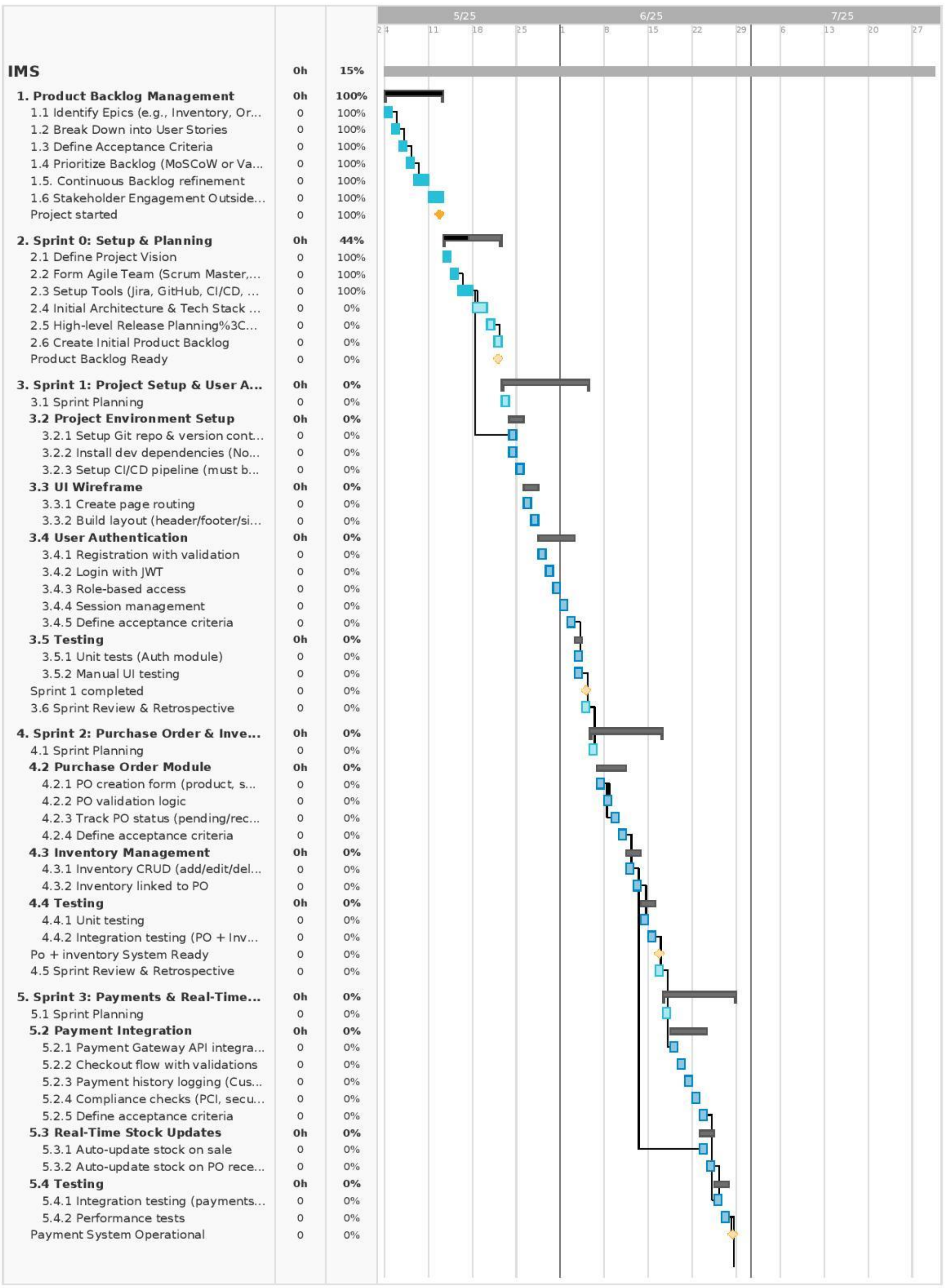

0. IMS
1. Product Backlog Management
1.1 Identify Epics (e.g., Inventory, Orders, Reports, User Management)
1.2 Break Down into User Stories
1.3 Define Acceptance Criteria
1.4 Prioritize Backlog (MoSCoW or Value-based)
1.5. Continuous Backlog refinement
1.6 Stakeholder Engagement Outside Reviews
2. Sprint 0: Setup & Planning
2.1 Define Project Vision
2.2 Form Agile Team (Scrum Master, Product Owner, Developers)
2.3 Setup Tools (Jira, GitHub, CI/CD, etc.)
2.3 Setup Tools (Jira, GitHub, CI/CD, etc.)
2.4 Initial Architecture & Tech Stack Selection
2.5 High-level Release Planning
2.6 Create Initial Product Backlog
3. Sprint 1: Project Setup & User Authentication
3.1 Sprint Planning
3.2 Project Environment Setup
3.2.1 Setup Git repo & version control
3.2.2 Install dev dependencies (Node, DB, framework)
3.2.3 Setup CI/CD pipeline (must be ready this sprint)
3.3 UI Wireframe
3.3.1 Create page routing
3.3.2 Build layout (header/footer/sidebar)
3.4 User Authentication
3.4.1 Registration with validation
3.4.2 Login with JWT
3.4.3 Role-based access
3.4.4 Session management
3.4.5 Define acceptance criteria
3.5 Testing
3.5.1 Unit tests (Auth module)
3.5.2 Manual UI testing
3.6 Sprint Review & Retrospective
4. Sprint 2: Purchase Order & Inventory Management
4.1 Sprint Planning
4.2 Purchase Order Module
4.2.1 PO creation form (product, supplier, quantity)
4.2.2 PO validation logic
4.2.3 Track PO status (pending/received)
4.2.4 Define acceptance criteria
4.3 Inventory Management
4.3.1 Inventory CRUD (add/edit/delete)
4.3.2 Inventory linked to PO
4.4 Testing
4.4.1 Unit testing
4.4.2 Integration testing (PO + Inventory)
4.5 Sprint Review & Retrospective
5. Sprint 3: Payments & Real-Time Stock Update
5.1 Sprint Planning
5.2 Payment Integration
5.2.1 Payment Gateway API integration (e.g., Stripe)
5.2.2 Checkout flow with validations
5.2.3 Payment history logging (Customer, Admin)
5.2.4 Compliance checks (PCI, security)
5.2.5 Define acceptance criteria
5.3 Real-Time Stock Updates
5.3.1 Auto-update stock on sale
5.3.2 Auto-update stock on PO receipt
5.4 Testing
5.4.1 Integration testing (payments/stock)
5.4.2 Performance tests
5.5 Sprint Review & Retrospective
6. Sprint 4: Dispatch & Notifications
6.1 Sprint Planning
6.2 Dispatch Module
6.2.1 View and manage sales orders
6.2.2 Assign delivery method & address
6.2.3 Mark as dispatched
6.2.4 Define acceptance criteria
6.3 Notifications
6.3.1 Email/SMS to customers on dispatch
6.3.2 Basic delivery tracking
6.4 Testing
6.4.1 Functional testing (dispatch & notify)
6.4.2 User feedback session
6.5 Sprint Review & Retrospective
7. Sprint 5: Reporting & Final Enhancements
7.1 Sprint Planning
7.2 Reporting Features
7.2.1 Admin Sales Report (profit/loss, trends)
7.2.2 Purchase Reports (cost/supplier)
7.2.3 Customer Transaction Reports
7.2.4 Export to PDF/CSV
7.2.5 Define acceptance criteria
7.3 System Enhancements
7.3.1 UX polish (responsive UI, accessibility)
7.3.2 Error handling
7.4 Testing & UAT
7.4.1 Final regression test
7.4.2 UAT with stakeholders
7.5 Deployment
7.5.1 Final deployment
7.5.2 Post-launch monitoring setup
7.6 Sprint Review & Project Retrospective
8. Project Closure
8.1 Final Documentation
8.2 Project Handover
8.2.1 Knowledge transfer to stakeholders or support team
8.2.2 Final code & credentials handover
8.2.3 Configuration & deployment scripts
8.3 Performance Review
8.4 Formal Project Closure
Final sign-off from stakeholders
Official closure communication to all parties

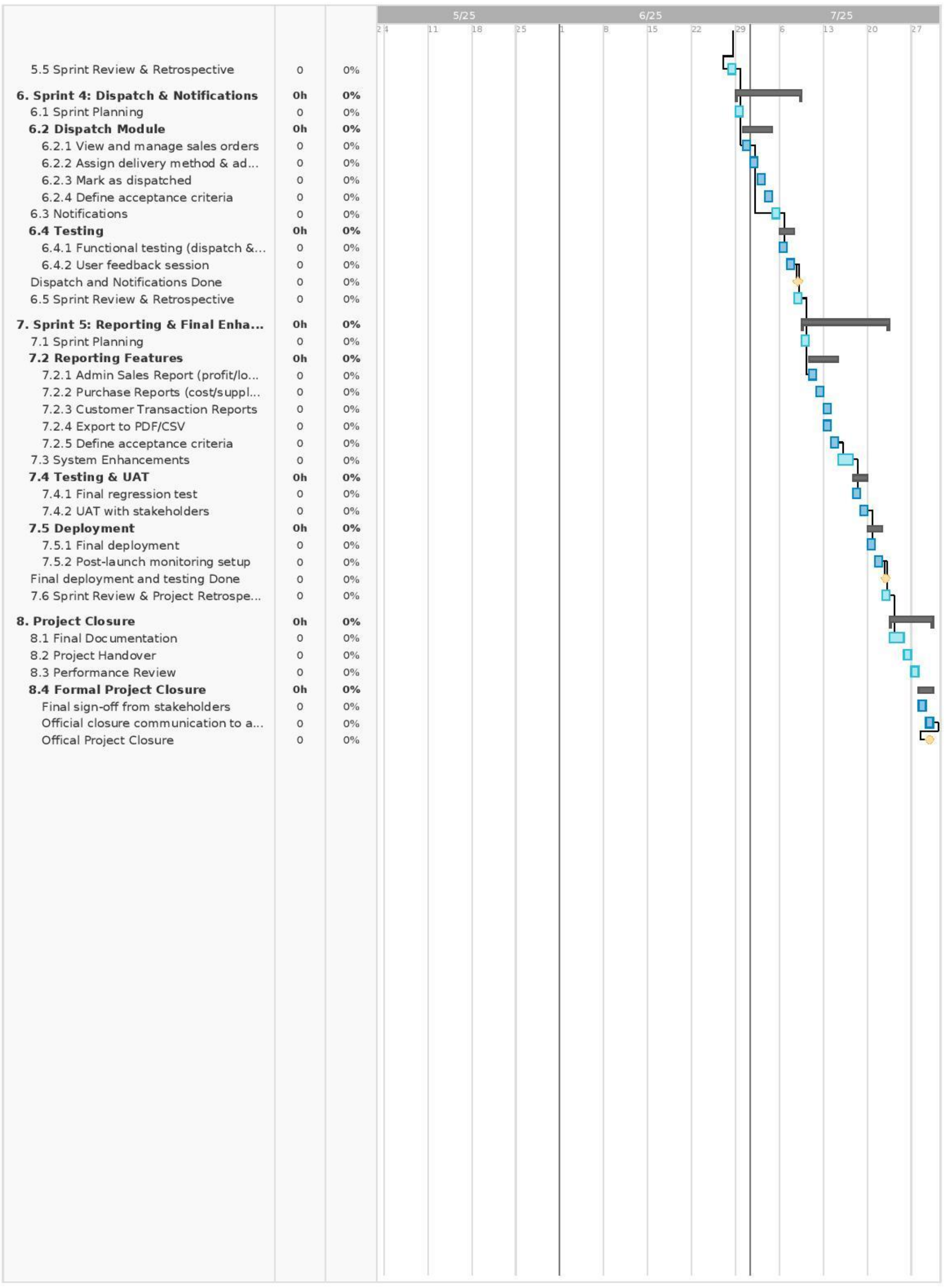
Figure 1: Work break down structure of IMS

3. Gantt Chart



IMS	**0h**	**15%**
1. Product Backlog Management	**0h**	**100%**
1.1 Identify Epics (e.g., Inventory, Or...	0	100%
1.2 Break Down into User Stories	0	100%
1.3 Define Acceptance Criteria	0	100%
1.4 Prioritize Backlog (MoSCoW or Va...	0	100%
1.5. Continuous Backlog refinement	0	100%
1.6 Stakeholder Engagement Outside...	0	100%
Project started	0	100%
2. Sprint 0: Setup & Planning	**0h**	**44%**
2.1 Define Project Vision	0	100%
2.2 Form Agile Team (Scrum Master,...	0	100%
2.3 Setup Tools (Jira, GitHub, CI/CD, ...	0	100%
2.4 Initial Architecture & Tech Stack ...	0	0%
2.5 High-level Release Planning%3C...	0	0%
2.6 Create Initial Product Backlog	0	0%
Product Backlog Ready	0	0%
3. Sprint 1: Project Setup & User A...	**0h**	**0%**
3.1 Sprint Planning	0	0%
3.2 Project Environment Setup	**0h**	**0%**
3.2.1 Setup Git repo & version cont...	0	0%
3.2.2 Install dev dependencies (No...	0	0%
3.2.3 Setup CI/CD pipeline (must b...	0	0%
3.3 UI Wireframe	**0h**	**0%**
3.3.1 Create page routing	0	0%
3.3.2 Build layout (header/footer/si...	0	0%
3.4 User Authentication	**0h**	**0%**
3.4.1 Registration with validation	0	0%
3.4.2 Login with JWT	0	0%
3.4.3 Role-based access	0	0%
3.4.4 Session management	0	0%
3.4.5 Define acceptance criteria	0	0%
3.5 Testing	**0h**	**0%**
3.5.1 Unit tests (Auth module)	0	0%
3.5.2 Manual UI testing	0	0%
Sprint 1 completed	0	0%
3.6 Sprint Review & Retrospective	0	0%
4. Sprint 2: Purchase Order & Inve...	**0h**	**0%**
4.1 Sprint Planning	0	0%
4.2 Purchase Order Module	**0h**	**0%**
4.2.1 PO creation form (product, s...	0	0%
4.2.2 PO validation logic	0	0%
4.2.3 Track PO status (pending/rec...	0	0%
4.2.4 Define acceptance criteria	0	0%
4.3 Inventory Management	**0h**	**0%**
4.3.1 Inventory CRUD (add/edit/del...	0	0%
4.3.2 Inventory linked to PO	0	0%
4.4 Testing	**0h**	**0%**
4.4.1 Unit testing	0	0%
4.4.2 Integration testing (PO + Inv...	0	0%
Po + inventory System Ready	0	0%
4.5 Sprint Review & Retrospective	0	0%
5. Sprint 3: Payments & Real-Time...	**0h**	**0%**
5.1 Sprint Planning	0	0%
5.2 Payment Integration	**0h**	**0%**
5.2.1 Payment Gateway API integra...	0	0%
5.2.2 Checkout flow with validations	0	0%
5.2.3 Payment history logging (Cus...	0	0%
5.2.4 Compliance checks (PCI, secu...	0	0%
5.2.5 Define acceptance criteria	0	0%
5.3 Real-Time Stock Updates	**0h**	**0%**
5.3.1 Auto-update stock on sale	0	0%
5.3.2 Auto-update stock on PO rece...	0	0%
5.4 Testing	**0h**	**0%**
5.4.1 Integration testing (payments...	0	0%
5.4.2 Performance tests	0	0%
Payment System Operational	0	0%

Figure 2:Gantt Chart



5.5 Sprint Review & Retrospective	0	0%
6. Sprint 4: Dispatch & Notifications	**0h**	**0%**
6.1 Sprint Planning	0	0%
6.2 Dispatch Module	**0h**	**0%**
6.2.1 View and manage sales orders	0	0%
6.2.2 Assign delivery method & ad...	0	0%
6.2.3 Mark as dispatched	0	0%
6.2.4 Define acceptance criteria	0	0%
6.3 Notifications	0	0%
6.4 Testing	**0h**	**0%**
6.4.1 Functional testing (dispatch &...	0	0%
6.4.2 User feedback session	0	0%
Dispatch and Notifications Done	0	0%
6.5 Sprint Review & Retrospective	0	0%
7. Sprint 5: Reporting & Final Enha...	**0h**	**0%**
7.1 Sprint Planning	0	0%
7.2 Reporting Features	**0h**	**0%**
7.2.1 Admin Sales Report (profit/lo...	0	0%
7.2.2 Purchase Reports (cost/suppl...	0	0%
7.2.3 Customer Transaction Reports	0	0%
7.2.4 Export to PDF/CSV	0	0%
7.2.5 Define acceptance criteria	0	0%
7.3 System Enhancements	0	0%
7.4 Testing & UAT	**0h**	**0%**
7.4.1 Final regression test	0	0%
7.4.2 UAT with stakeholders	0	0%
7.5 Deployment	**0h**	**0%**
7.5.1 Final deployment	0	0%
7.5.2 Post-launch monitoring setup	0	0%
Final deployment and testing Done	0	0%
7.6 Sprint Review & Project Retrospe...	0	0%
8. Project Closure	**0h**	**0%**
8.1 Final Documentation	0	0%
8.2 Project Handover	0	0%
8.3 Performance Review	0	0%
8.4 Formal Project Closure	**0h**	**0%**
Final sign-off from stakeholders	0	0%
Official closure communication to a...	0	0%
Offical Project Closure	0	0%

Figure 3: Gantt Chart of IMS

4. Use Case Model

4.1 Use Case Diagram

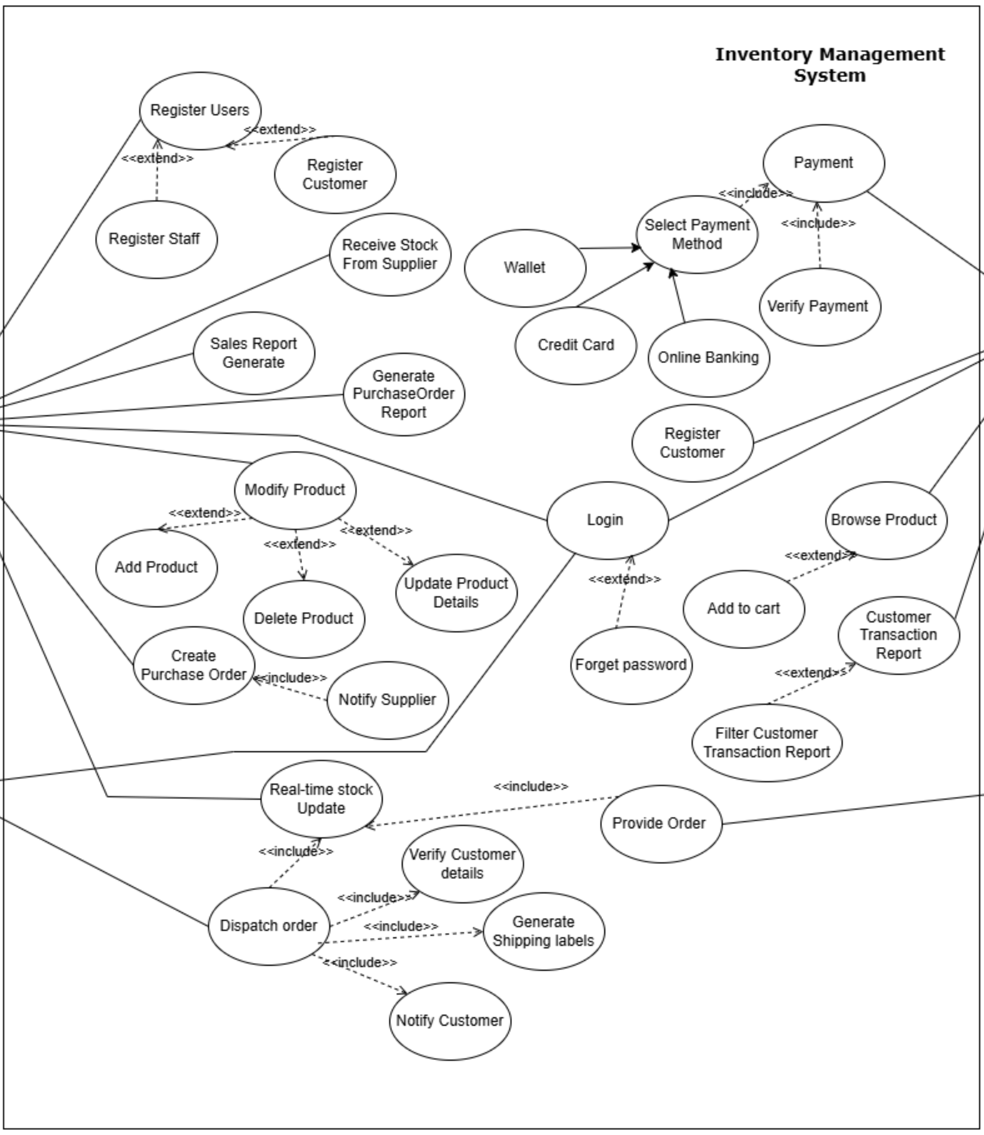


Figure 4: Use Case Diagram for IMS

4.2 High Level Description

1. Name: Register User

Actors: Admin

Description: Admin can register staff and customers manually.

2. Name: Login

Actors: Admin, Customer, staff

Description: All actors can login into the system to perform role-specific actions. Their might come login error or users can forget their password.

3. Name: Create Purchase order

Actor(s): Admin

Description: The Admin generates a purchase order for inventory restocking, specifying product details and quantities. This use case includes notifying the Supplier .

4. Name: Generate Sales Report

Actor: Admin

Description: The admin generates a report summarizing sales performance, including profit/loss calculations and trends over time.

5. Name: Receive Stock from Supplier

Actor: Admin

Description: Admin logs and manages inventory received from suppliers.

6. Name: Modify Product

Actor: Admin

Description: Admin performs product modifications, which include adding, deleting, and updating product details.

7. Name: Payment

Actor: Admin

Description: This use case allows the customer to complete a purchase by making payment. It includes **the** Select Payment Method, where the customer chooses from Wallet, Credit Card, or Online Banking, and **Verify Payment**, which confirms the success of the transaction. These steps ensure secure and validated payments before order processing continues.

8. Name: Generate Customer Transaction Report

Actor: Customer

Description: The Admin generates a report summarizing sales performance, including profit/loss calculations and trends over time.

9. Name: Dispatch order

Actor: Staff

Description: The Admin prepares and dispatches the customer's order. This includes verifying customer details and confirming payment. This also includes real-time stock update and generate shipping label.

10. Name: Real-Time Stock Up

Actor: System(automated), Admin

Description: This use case ensures that the inventory is automatically updated whenever stock is received or dispatched and admin can manually too. It includes updates triggered by **Dispatch Order** and **Provide Order**, keeping product quantities accurate in real time. This helps prevent stockouts, overstocking, and maintains operational efficiency.

11. Name: Browse product

Actor: Customer

Description: This case allows customers to view available products in the inventory system. Customers can explore product categories, descriptions, prices, and availability before making a purchase decision. It helps users interact with the system without needing to log in or make a transaction.

12. Name: Generate Purchase Order Report

Actor: Admin

Description: The Purchase Order Report provides the admin with a detailed overview of inventory acquisitions from suppliers. It includes information such as product names, quantities received, unit costs, total costs, supplier names, and order dates. This report helps track purchasing activities, monitor supplier performance, and support inventory planning and budgeting decisions.

4.3 Expanded Level Description

1. Use Case: Payment

Name : Payment

Actor: Customer

Purpose: Purchases Goods

Overview: The Customer selects items for purchase, chooses a payment method (Wallet/Credit Card/Online Banking), and completes the transaction. The system validates the payment.

Type : Primary, Essential

Action Steps:

Actor Action	System Response
1. Customer selects items and clicks Proceed to Payment.	
	2. Displays a summary of selected items, total cost, and available payment methods.
3. Customer selects a payment method(like wallet) and send the payment proof (transaction Id, account no, payment details screenshot etc).	
	4. Validates payment details (transaction Id, account no, payment details screenshot etc).
5. Customer confirms payment.	
	6. Processes the transaction via the payment gateway and generates a transaction ID.
7. Customer waits for confirmation.	
	8. Sends a payment confirmation email/SMS to the Customer with the receipt.

Alternative Course of Action:

Line 4: If payment details are invalid (e.g., expired card), the system flags an error, and the Customer re-enters information or selects another method.

Line 8: If stock is insufficient *after* payment (due to system lag), the Admin is notified to resolve it , and the Customer receives an apology email with a discount coupon.

2. Use Case: Dispatch Order

Name : Dispatch Order

Actor: Staff

Purpose: Prepare and ship orders to customers

Overview: Staff reviews the sales order, validates customer details (address, contact), selects a delivery method, and dispatches the order. Stock levels are reduced automatically upon dispatch.

Type : Primary, Essential

Action Steps:

Actor Action	System Response
1. Staff selects “Dispatch Order” from the dashboard.	
	2. Displays a list of paid orders awaiting dispatch.
3. Staff selects an order and reviews customer details (address, contact).	
	4. Validates the delivery address ,contact information and payment.
5. Staff selects a delivery method (e.g., Nepalcanmove).	
	6. Generates a shipping label and updates the order status to “Ready for Dispatch.”
7. Staff confirms dispatch.	
	8. Triggers Real-Time Stock Update to deduct dispatched items from inventory.
	9. Sends a dispatch notification (tracking ID, ETA) to the Customer via email.

Alternative Course of Action:

Line 4: If the address, contact no is invalid, the system gives an error, and Admin/Staff contacts the Customer to update details before proceeding.

Line 8:If stock levels are incorrect (e.g., item already out of stock), the system alerts Admin/Staff to halt dispatch and resolve the issue manually via Adjust Stock Levels.

5. Sequence Diagram

Sequence diagram are type of UML diagram that show how objects in a system or classes within a use case interact with each other. It shows interaction among objects as a two dimensional chart. Messages show the information being sent between objects. It shows interactions in the order they take place. In other words, they show the sequence of events and hence named sequence diagrams. It emphasizes on the time ordering of the messages. They are primarily used by developers and business professionals to document processes or understand the requirements of new program. It shows interaction between numerous objects within a single use case. (GeeksforGeeks, 2017)

This sequence diagram shows how customer orders get dispatched in the Inventory Management System (IMS). It all starts when a staff member clicks on the "Dispatch Order" option to check out all the orders that have already been paid for. The system then checks the customer's delivery info like their address and contact number to make sure everything is valid. If something’s wrong (like a missing digit in the phone number or an incomplete address), the system tells the staff to reach out to the customer and get the correct info. Once everything checks out, the staff chooses a delivery method (like courier or in-store pickup).

Next, the system contacts the warehouse to create a shipping label and get the tracking details. It also changes the order status to "Ready for Dispatch" and updates the stock in real time. If there’s not enough stock, the system cancels the dispatch and starts the restocking process. But if the stock is available, it gets deducted from the inventory.

Finally, the system sends confirmation to the customer with the tracking ID and the estimated delivery time.

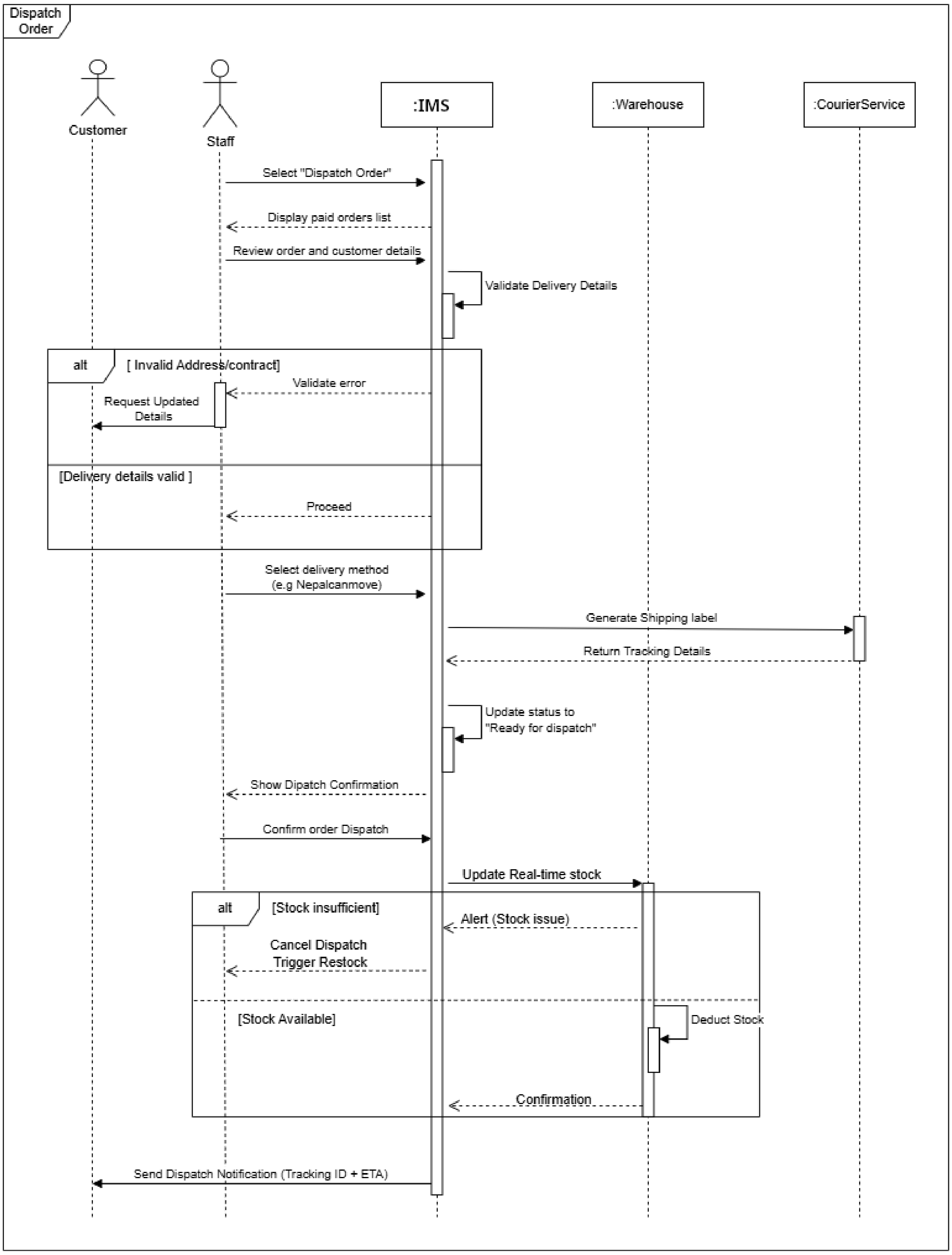


Figure 5: Sequence Diagram for Dispatch order

6. Activity Diagram

An activity diagram is a type of UML (Unified Modeling Language) behavioral diagram that visually represents the flow of control or data through a process or system. It is used to model workflows, business logic, and operational sequences within a system, highlighting the various activities, decisions, and parallel processes that occur. (GeeksforGeeks, 2017)

This activity diagram represents the customer payment process in an Ims system. The sequence begins with the customer browsing products, viewing items, and adding desired products to the cart. After checkout, the user is prompted to choose a payment method.

The diagram includes a decision node where the customer selects one of three payment modes: Wallet, Credit Card, or Online Banking. Regardless of the selected option, the next steps involve inputting payment data, uploading payment proof, and having the system validate the proof. If the proof is incorrect, the customer must re-enter the data. If the validation is successful, the system sends a payment success email with the receipt, marking the end of the transaction process.

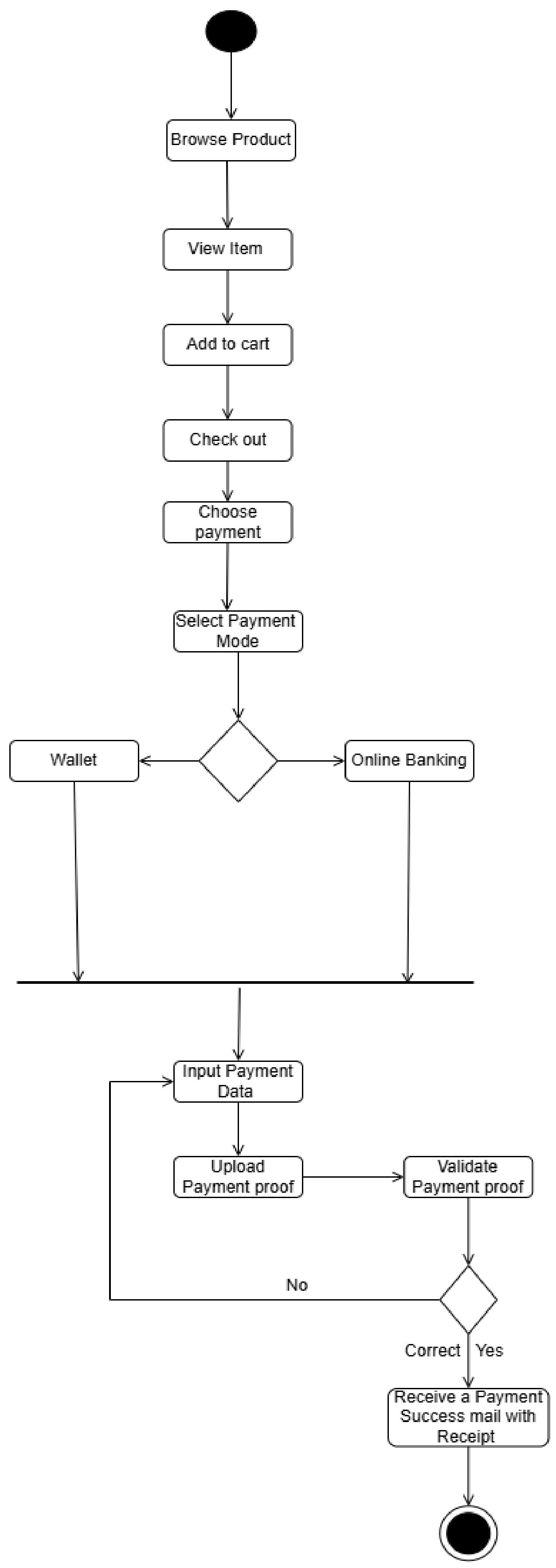


Figure 6: Activity Diagram for Payment

7. Class Diagram

A class diagram is a type of static structure diagram in UML that illustrates the system's classes, their attributes, methods, and the relationships between them. It provides a high-level overview of how different entities in the system interact and are organized. Class diagrams are essential for understanding the data model and guiding object-oriented design. They help in visualizing inheritance, associations, dependencies, and aggregations among classes.

This class diagram represents an Inventory Management System (IMS) with roles such as Admin, Customer, and Staff. It captures core functionalities like order processing, payment validation, stock updates, report generation, and notifications. Relationships between classes ensure clear responsibilities.For example, Admin manages purchase orders and reports, while Customers place orders and make payments. The system supports multiple payment types and real-time inventory updates for accurate tracking.

Use Cases	Domain Classes
Register	User, Customer, RegistrationService, Profile
Login	User, Customer, Staff, LoginService
Create Purchase Order	PurchaseOrder, Product, Supplier, OrderService
Sales Report Generate	SalesReport, ReportService
Purchase Order Report Generate	PurchaseOrderReport, ReportService
Customer Transaction Report	CustomerTransactionReport, Customer
Payment	Payment, WalletPayment, CreditCardPayment, OnlineBankingPayment, Customer
Real-time Stock Update	RealTimeStockUpdate, Inventory, Product
Dispatch Order	Order, Delivery, ShippingLabel, Customer, Staff
Verify Customer Details	Customer, VerificationService, Staff
Verify Payment	Payment, WalletPayment, CreditCardPayment, OnlineBankingPayment
Provide Order	PurchaseOrder, Supplier, Product

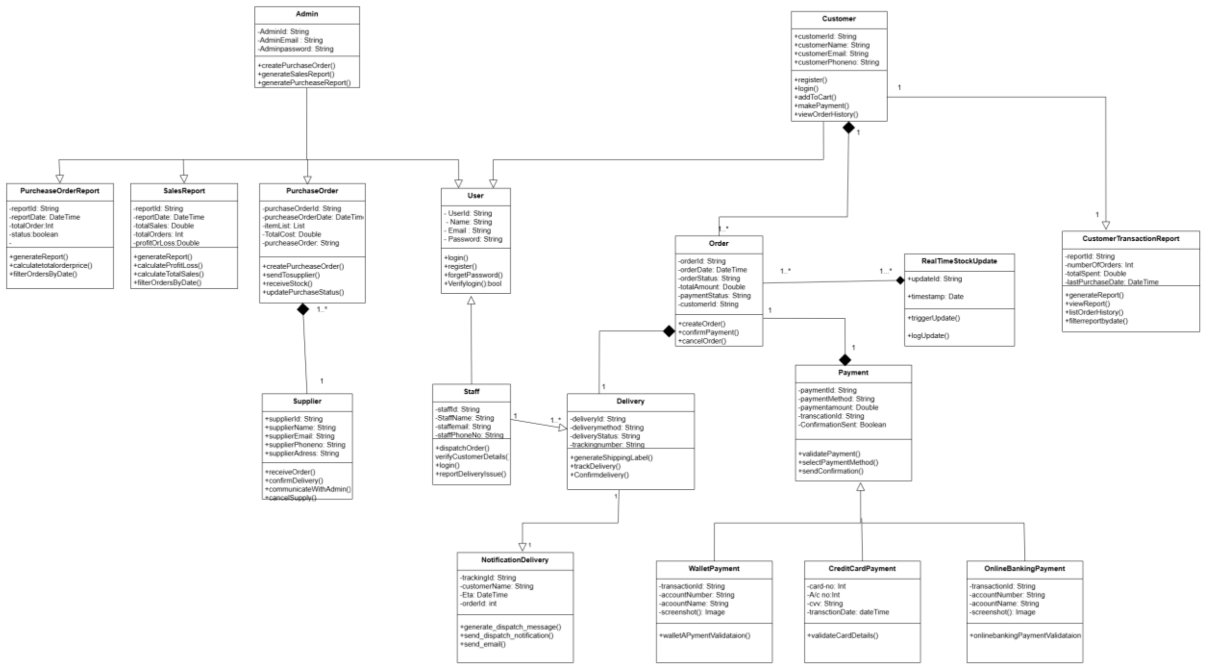


Figure 7:Class Diagram

8. Further Development

As we move forward with the project, I plan to adopt the Agile methodology, specifically the Scrum framework, to guide the entire development process. Agile is the most feasible approach for this project because it supports iterative development, quick adaptability to changing requirements, continuous stakeholder feedback, and faster delivery of working features.

8.1 Architectural Choice

For the development of this system, I will adopt a Layered (N-Tier) Architecture. The reason for choosing a layered architecture is its ability to make the system highly scalable, secure, and easy to maintain. Changes can be made to one layer without heavily impacting others; for instance, frontend updates can occur without altering backend logic, and database changes can happen independently of user interface adjustments. Additionally, this structure promotes better security, as sensitive operations like payment validation and stock management are confined within protected backend services, away from direct client access. It also improves testability, allowing unit and integration testing to be conducted at each individual layer.

The frontend will communicate with the backend using RESTful APIs, sending requests for operations like registration, login, order placement, and payment processing. The backend will process these requests according to business rules, then interact with the database to store, retrieve, or update records accordingly. This clean and modular flow will help ensure the system remains robust, flexible, and ready to scale in the future.

This architecture will divide the application into three distinct layers, each with its own responsibility:

Presentation Layer: Handles the user interface and user interaction (e.g., login page, purchase order form, payment interface).

Business Logic Layer: Processes core business logic and application rules (e.g., validating payment, managing orders, generating reports).

Data Access Layer: Manages interaction with the database (e.g., saving user data, retrieving order details, updating stock).

8.2 Design Pattern

The primary design pattern I will implement is the Model-View-Controller (MVC) pattern. This pattern will help in structuring the application into three interconnected components: the Model (handling data and business logic), the View (managing the user interface), and the Controller (coordinating input, processing, and output). Using MVC will allow a clean separation between the UI and business logic, enabling faster development and easier maintenance when scaling or updating individual parts of the system.

Alongside MVC, I will implement the Singleton pattern for core services such as the Payment Gateway and the Notification Service. These services must ensure a single, consistent point of interaction throughout the system, and Singleton will guarantee that only one instance exists during the application's lifetime. This will help avoid duplication errors and unnecessary overhead.

Additionally, the Factory pattern will be used for generating various types of reports, such as Sales Reports and Customer Transaction Reports. This pattern provides a way to create objects without specifying the exact class of the object that will be created, making the system highly flexible and easily extensible for new types of reports in the future.

Finally, for handling real-time updates such as order dispatch notifications and stock updates, I will apply the Observer pattern. This pattern allows different parts of the system (e.g., customer notification services) to automatically receive updates when certain actions (like dispatching an order) occur. This will promote loose coupling and better real-time responsiveness across the system.

8.3 Development Plan

To move forward with the implementation of the system, I have carefully selected a technology stack and structured a priority-based development approach. The frontend of the system will be developed using **React.js**, paired with **CSS** for building a responsive and modern user interface. For the backend, I will use **Node.js** with the **Express.js** framework, providing a fast, scalable, and lightweight server environment. The system's database will be based on **MongoDB**, a flexible NoSQL database well-suited for the dynamic data structures the application requires.

In terms of resources and tools, **Git** and **GitHub** will be used for version control, ensuring proper management of code changes, collaboration, and history tracking. **Jira** will serve as the Agile project management tool, helping to plan and monitor sprint progress efficiently. For continuous integration and deployment (CI/CD), **GitHub Actions** will be configured to automate build, test, and deployment workflows. The system will eventually be deployed to **AWS** infrastructure, specifically **EC2 instances** for server hosting and **S3** buckets for managing static resources like uploaded documents and images.

The development will be carried out in prioritized sprints. Initially, **Sprint 1** will focus on setting up the project environment, creating the basic UI scaffolding, and implementing the user registration and login modules with secure JWT-based authentication. **Sprint 2** will involve developing the purchase order creation and inventory management functionalities. In **Sprint 3**, payment integration with a Payment Gateway API and the implementation of real-time stock updates will be prioritized. **Sprint 4** will cover the dispatch module, including delivery tracking and customer notification systems. **Sprint 5** will focus on building robust reporting features, including Sales Reports and Customer Transaction Reports, followed by system-wide integrations and enhancements.

This structured, iterative development approach will ensure steady progress, early feedback, and deliver functional increments at the end of each sprint, keeping the project aligned with Agile best practices.

8.4 Testing Plan

In testing Plan ,we will follow a testing strategy that starts with "testing-in-the-small" and gradually moves to "testing-in-the-large," just like how it’s done in regular and object-oriented software testing. First, we will focus on Unit Testing, where each module and class (including their attributes and operations) will be tested separately. We’ll check things like the class interfaces, local data structures, boundary conditions, independent paths, and how errors are handled. We’ll also create test drivers to help run these tests properly.

After unit testing is done for all modules, we’ll move on to Integration Testing. Instead of testing everything at once (like in the “big bang” method), we’ll integrate modules one by one. We'll use both Top-Down Integration (starting with the higher-level modules and using stubs for the ones not ready yet) and Bottom-Up Integration (starting with the lower-level modules and using drivers). This mixed method is also called Sandwich Testing and will help us make sure both the control flow and data flow are working correctly.

Once modules are integrated, we’ll keep running Regression Testing often to make sure that new changes or features don’t break anything that was already working. Every time we change the code, we’ll rerun earlier test cases to catch any unexpected bugs. On top of that, we’ll do Smoke Testing for every new build. This will help us catch major issues early and save time before doing deeper testing.

For the actual testing methods, we’ll mostly use Black-Box Testing techniques like Equivalence Partitioning and Boundary Value Analysis. This way, we’ll focus on testing the inputs and outputs without worrying about the internal code. For features like image or file uploads, we’ll prepare detailed test cases to check things like valid file paths, allowed file types, file size limits, error handling for unsupported files, upload progress, and what happens with duplicate or oversized files.

By combining unit testing, integration testing, regression testing, smoke testing, and black-box testing, we’ll make sure the project is reliable, errors are found quickly, and the system will be stable and ready for deployment.

8.5 Maintenance Plan

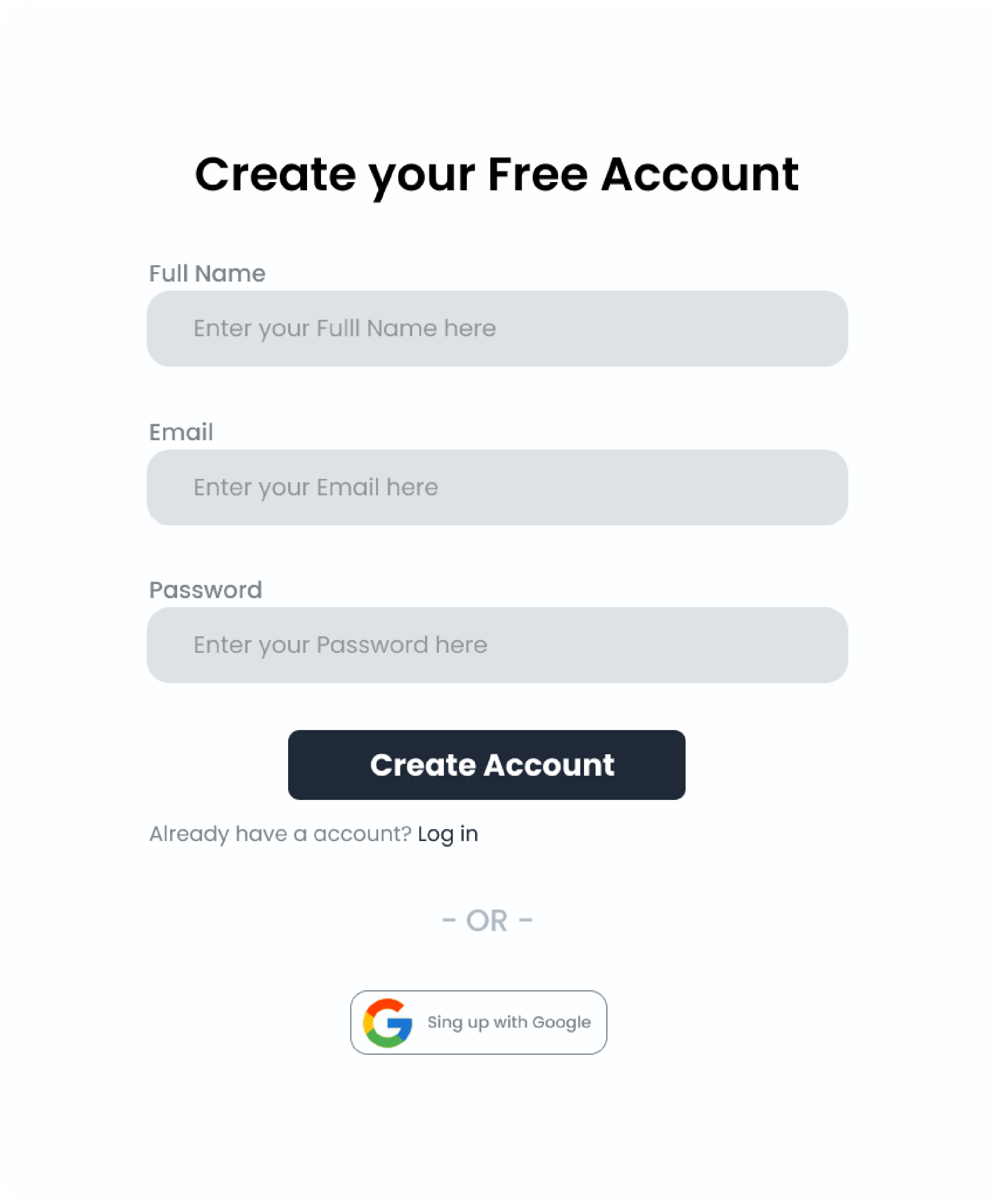
After the system is successfully deployed, we will set up a structured maintenance plan to keep everything running smoothly, securely, and up-to-date. The maintenance will fall into a few categories: **corrective**, **adaptive**, **perfective**, and **preventive**.

- **Corrective maintenance** will be for fixing bugs or errors that users report after the system is live. We'll quickly address these through hotfixes or patch releases to make sure everything keeps working.
- **Adaptive maintenance** will happen when we need to update the system due to changes in the environment, like when third-party services are upgraded or the deployment platform itself gets updated.
- **Perfective maintenance** will be about improving the system over time, like boosting performance, improving the user interface, and making the system more user-friendly based on feedback and any new requirements that come up.
- **Preventive maintenance** will focus on doing things to stop problems before they happen, like refactoring code, updating dependencies, and optimizing the system to avoid future issues.

To help with all of this, we'll use monitoring tools like AWS CloudWatch to keep track of server performance, database health, and error logs in real-time. We'll also set up daily backups of the MongoDB database to make sure data is safe, and we'll have a recovery plan in place in case of any system failures or cyber-attacks, so we can get things back up and running quickly.

We'll plan small releases every two months to incorporate improvements, fix bugs, and enhance the system's performance. We'll also keep a continuous feedback loop with stakeholders and users to ensure the system evolves and gets better even after the initial launch.

9. Prototype


Create your Free Account

Full Name

Enter your Fulll Name here

Email

Enter your Email here

Password

Enter your Password here

Figure 8: Registration Page

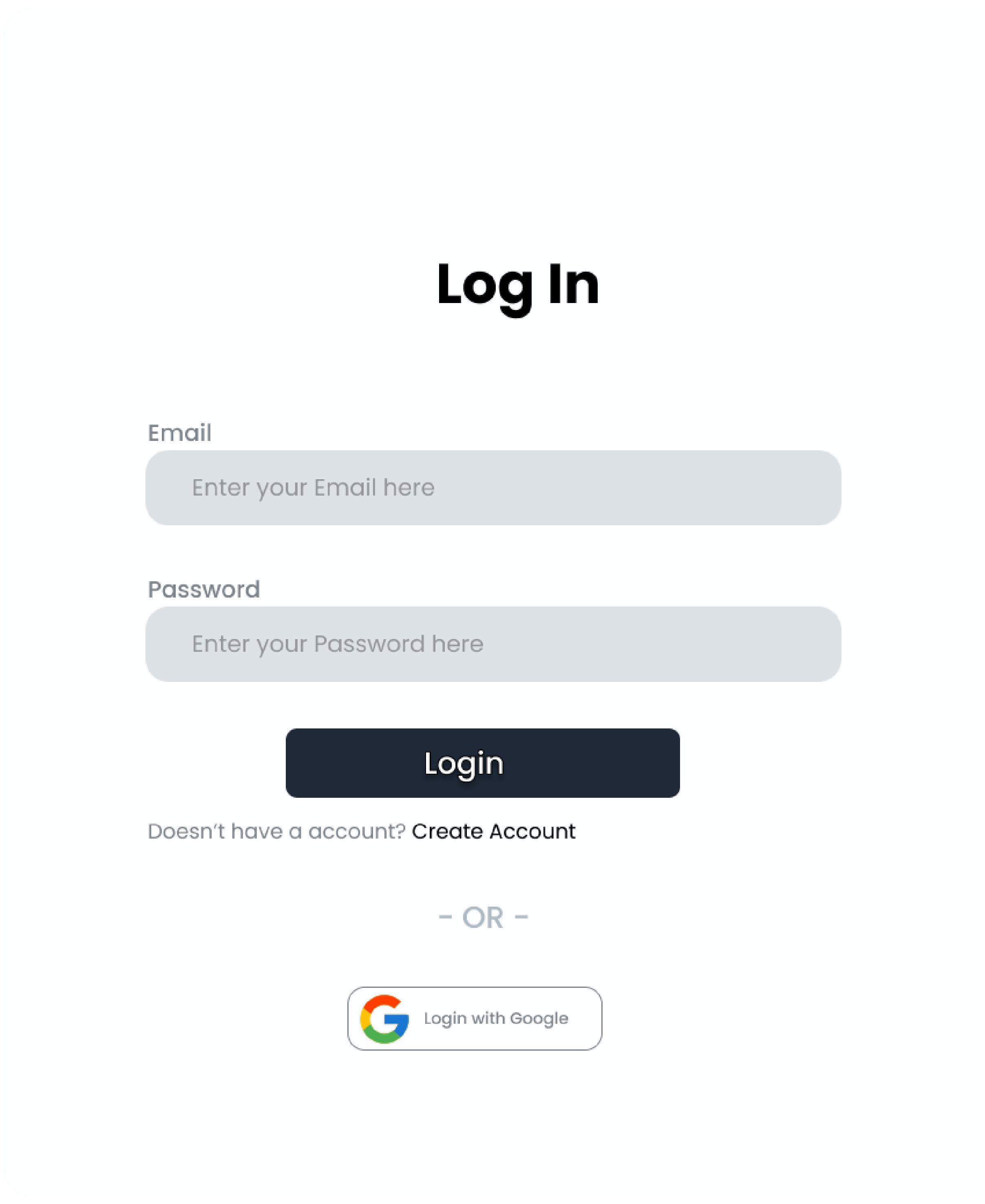


Figure 9:Login Page

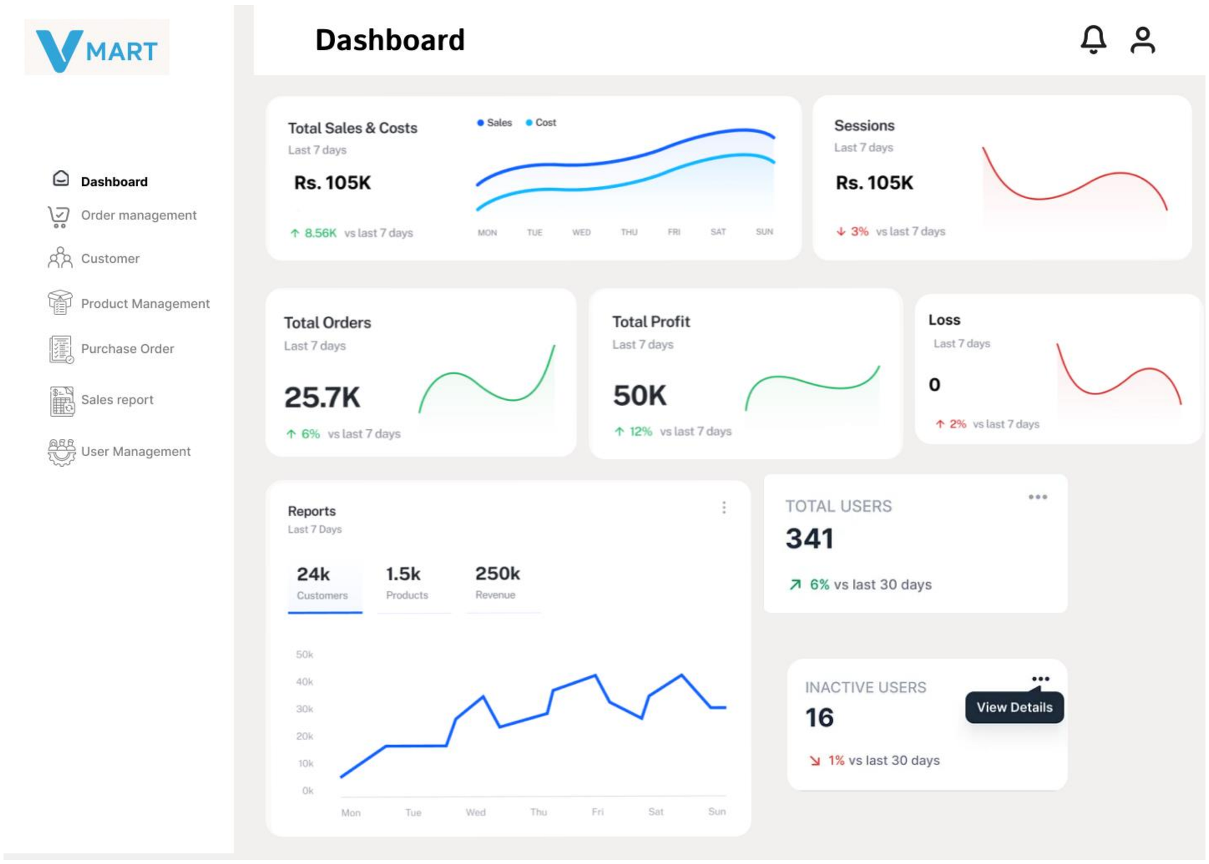
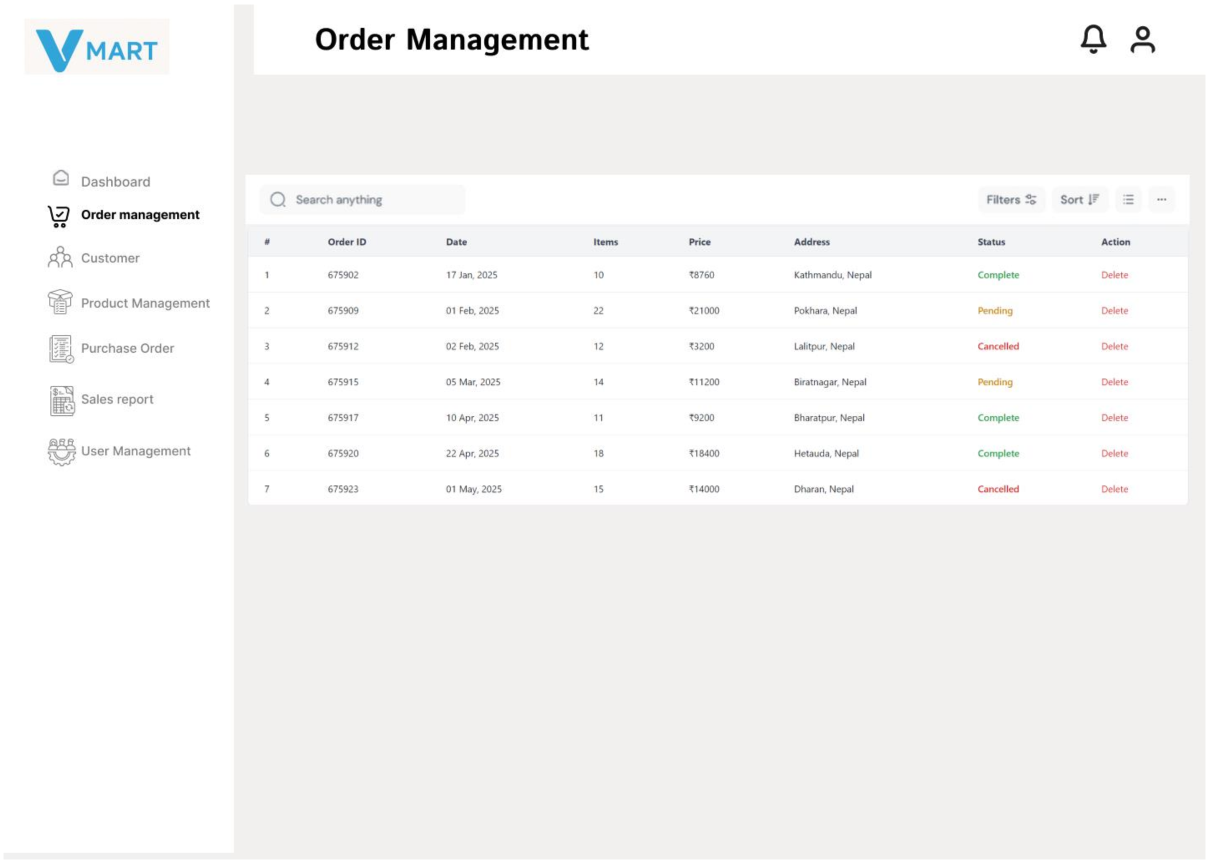
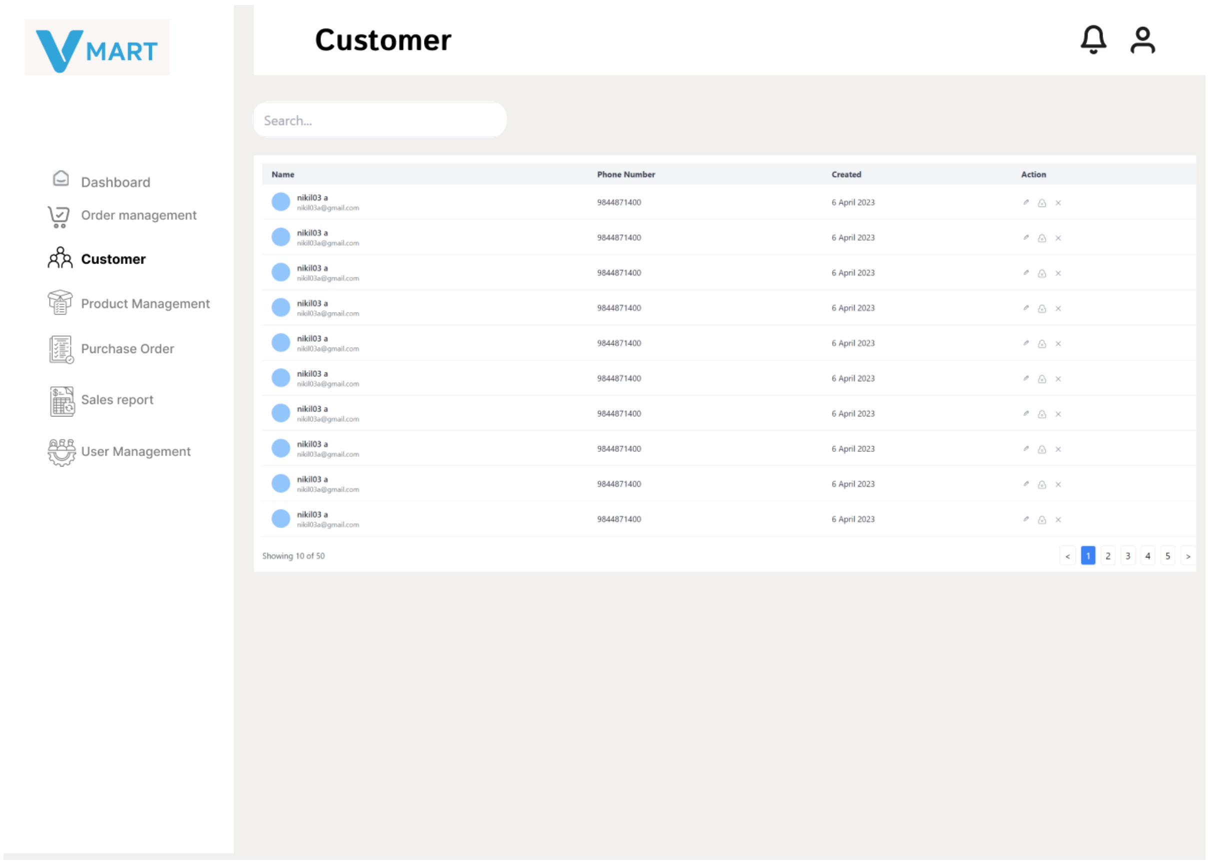


Figure 10:Dashboard



#	Order ID	Date	Items	Price	Address	Status	Action
1	675902	17 Jan, 2025	10	₹8760	Kathmandu, Nepal	Complete	Delete
2	675909	01 Feb, 2025	22	₹21000	Pokhara, Nepal	Pending	Delete
3	675912	02 Feb, 2025	12	₹3200	Lalitpur, Nepal	Cancelled	Delete
4	675915	05 Mar, 2025	14	₹11200	Biratnagar, Nepal	Pending	Delete
5	675917	10 Apr, 2025	11	₹9200	Bharatpur, Nepal	Complete	Delete
6	675920	22 Apr, 2025	18	₹18400	Hetauda, Nepal	Complete	Delete
7	675923	01 May, 2025	15	₹14000	Dharan, Nepal	Cancelled	Delete

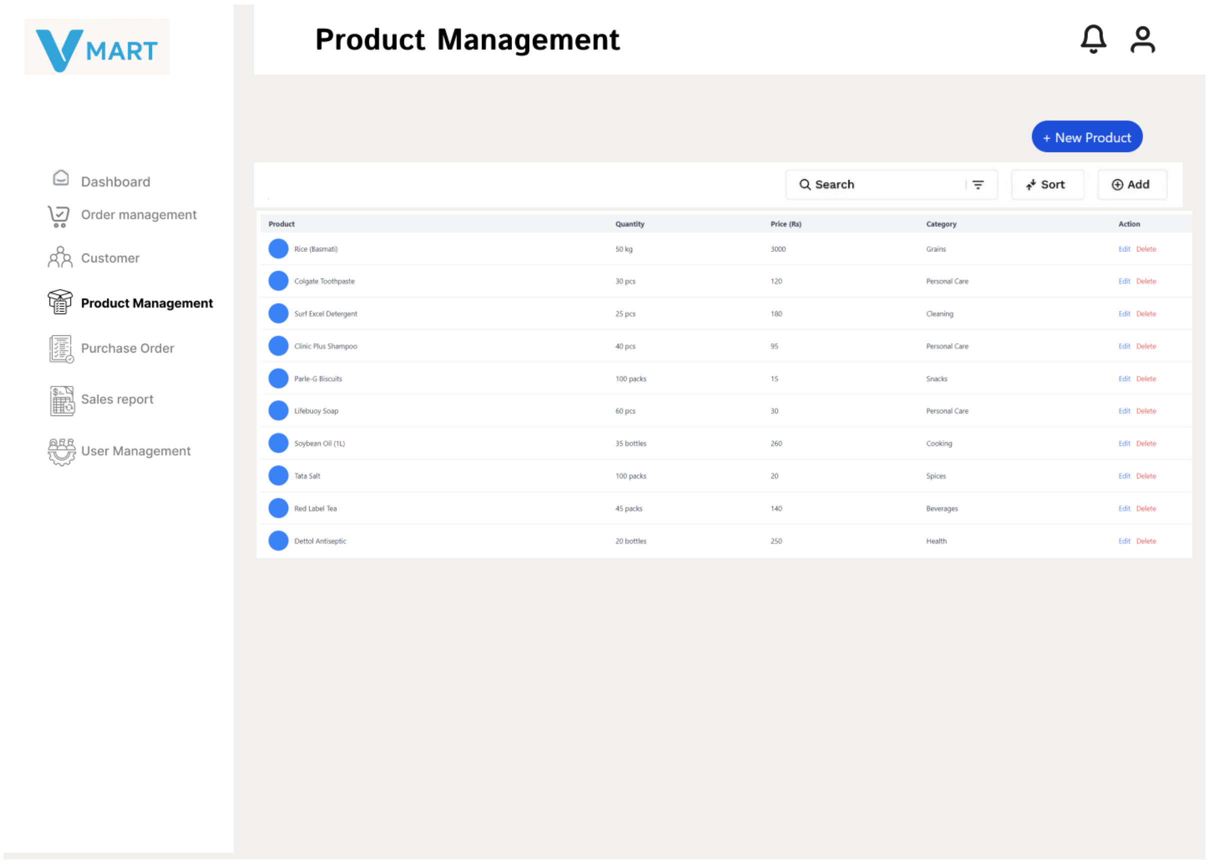
Figure 11:Order Management



Customer

Name	Phone Number	Created	Action
nikil03 a nikil03a@gmail.com	9844871400	6 April 2023	
nikil03 a nikil03a@gmail.com	9844871400	6 April 2023	
nikil03 a nikil03a@gmail.com	9844871400	6 April 2023	
nikil03 a nikil03a@gmail.com	9844871400	6 April 2023	
nikil03 a nikil03a@gmail.com	9844871400	6 April 2023	
nikil03 a nikil03a@gmail.com	9844871400	6 April 2023	
nikil03 a nikil03a@gmail.com	9844871400	6 April 2023	
nikil03 a nikil03a@gmail.com	9844871400	6 April 2023	
nikil03 a nikil03a@gmail.com	9844871400	6 April 2023	
nikil03 a nikil03a@gmail.com	9844871400	6 April 2023	

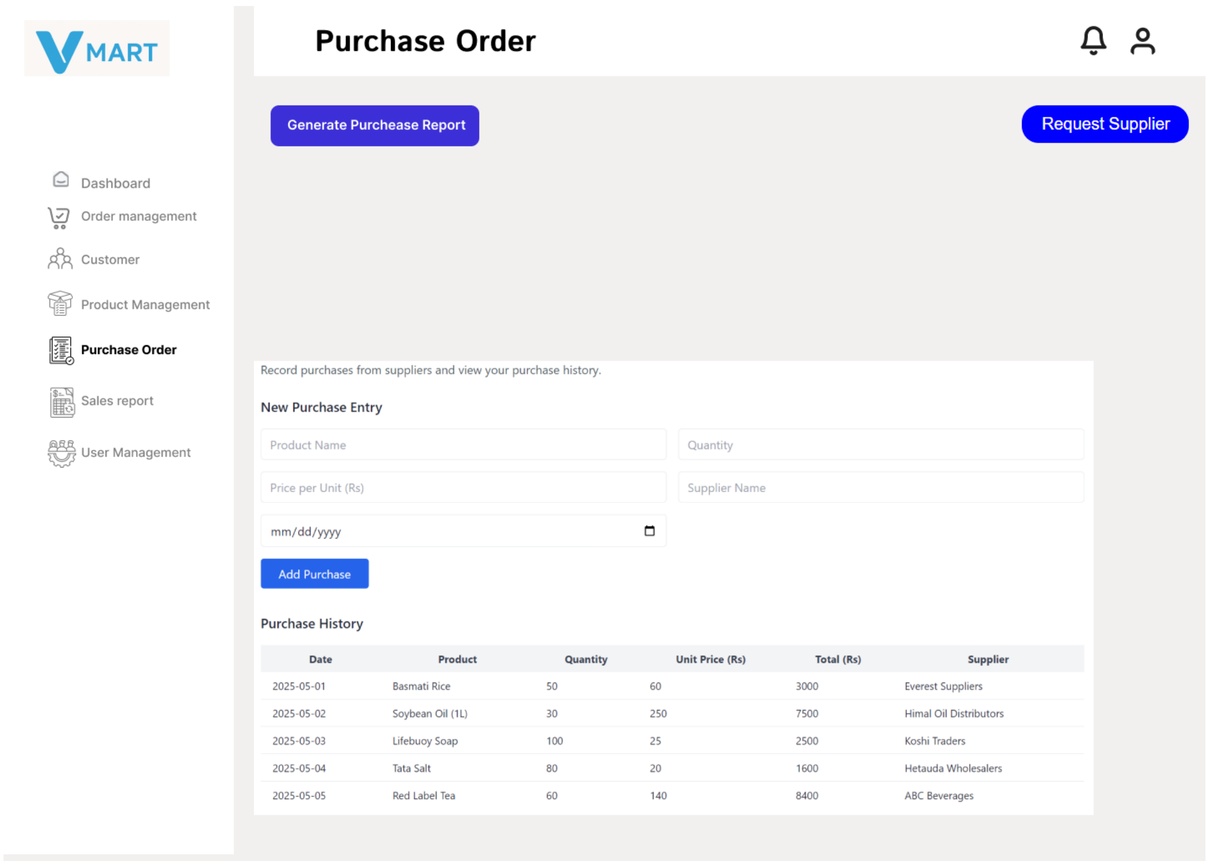
Figure 12:Customer page



Product Management

Product	Quantity	Price (Rs)	Category	Action
Rice (Basmati)	50 kg	3000	Grains	Edit Delete
Colgate Toothpaste	30 pcs	120	Personal Care	Edit Delete
Surf Excel Detergent	25 pcs	180	Cleaning	Edit Delete
Clinic Plus Shampoo	40 pcs	95	Personal Care	Edit Delete
Parle-G Biscuits	100 packs	15	Snacks	Edit Delete
Lifebuoy Soap	60 pcs	30	Personal Care	Edit Delete
Soybean Oil (1L)	35 bottles	260	Cooking	Edit Delete
Tata Salt	100 packs	20	Spices	Edit Delete
Red Label Tea	45 packs	140	Beverages	Edit Delete
Dettol Antiseptic	20 bottles	250	Health	Edit Delete

Figure 13:Product Management



Purchase Order

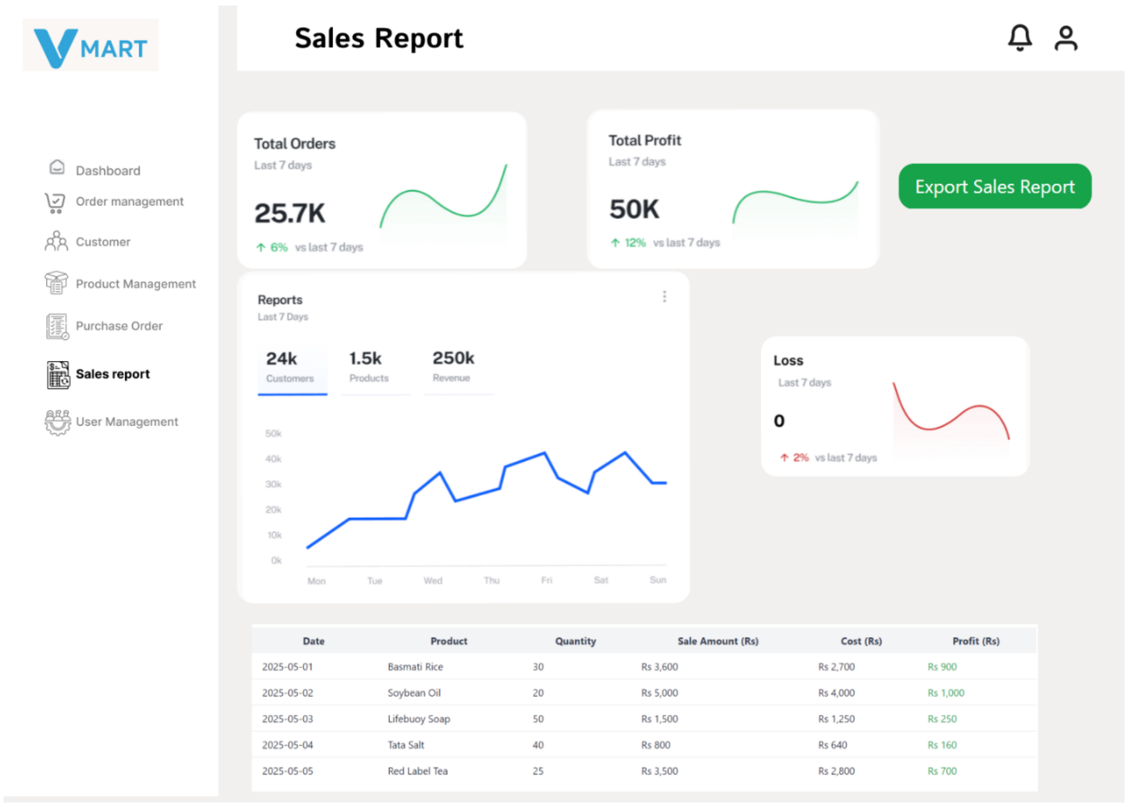
Record purchases from suppliers and view your purchase history.

New Purchase Entry

Purchase History

Date	Product	Quantity	Unit Price (Rs)	Total (Rs)	Supplier
2025-05-01	Basmati Rice	50	60	3000	Everest Suppliers
2025-05-02	Soybean Oil (1L)	30	250	7500	Himal Oil Distributors
2025-05-03	Lifebuoy Soap	100	25	2500	Koshi Traders
2025-05-04	Tata Salt	80	20	1600	Hetauda Wholesalers
2025-05-05	Red Label Tea	60	140	8400	ABC Beverages

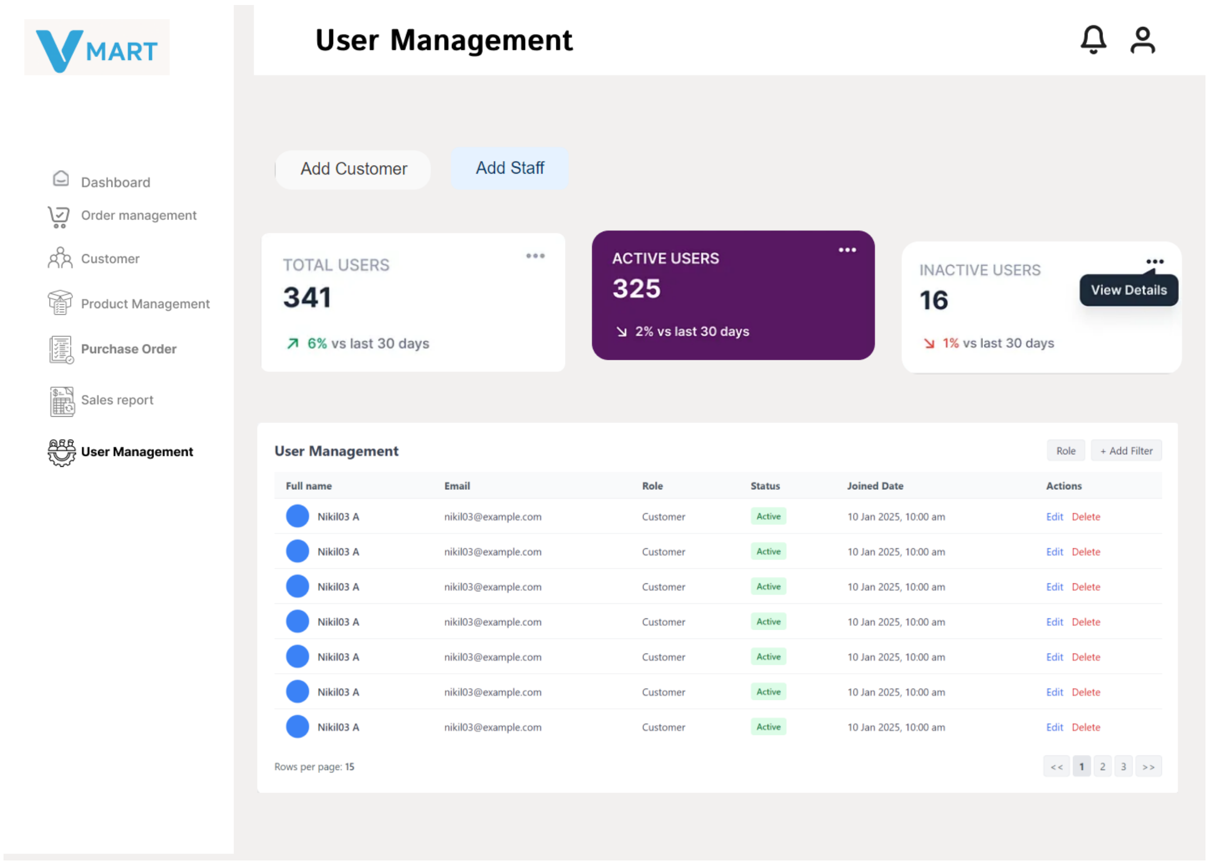
Figure 14: Purchase Order Page



Sales Report

Date	Product	Quantity	Sale Amount (Rs)	Cost (Rs)	Profit (Rs)
2025-05-01	Basmati Rice	30	Rs 3,600	Rs 2,700	Rs 900
2025-05-02	Soybean Oil	20	Rs 5,000	Rs 4,000	Rs 1,000
2025-05-03	Lifebuoy Soap	50	Rs 1,500	Rs 1,250	Rs 250
2025-05-04	Tata Salt	40	Rs 800	Rs 640	Rs 160
2025-05-05	Red Label Tea	25	Rs 3,500	Rs 2,800	Rs 700

Figure 15:Sales Report Page



VMART

User Management

- Dashboard
- Order management
- Customer
- Product Management
- Purchase Order
- Sales report
- **User Management**

Add Customer | Add Staff

TOTAL USERS **341** ↗ 6% vs last 30 days

ACTIVE USERS **325** ↘ 2% vs last 30 days

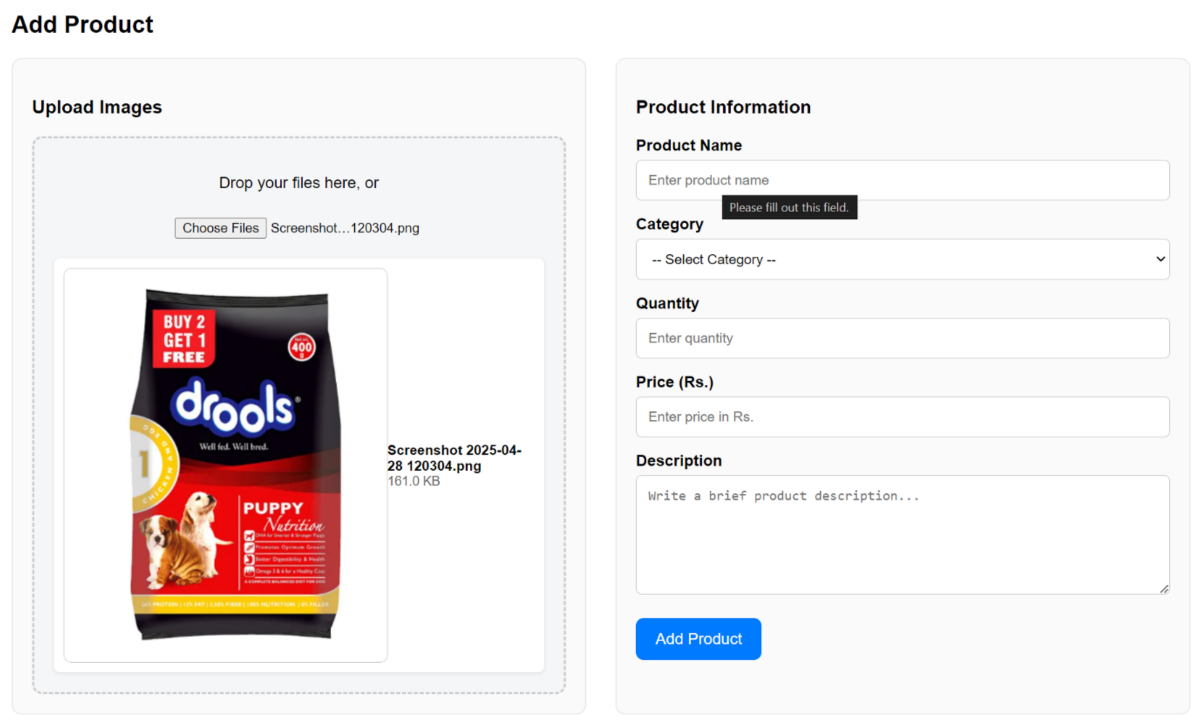
INACTIVE USERS **16** ↘ 1% vs last 30 days — View Details

User Management — Role | + Add Filter

Full name	Email	Role	Status	Joined Date	Actions
Nikil03 A	nikil03@example.com	Customer	Active	10 Jan 2025, 10:00 am	Edit Delete
Nikil03 A	nikil03@example.com	Customer	Active	10 Jan 2025, 10:00 am	Edit Delete
Nikil03 A	nikil03@example.com	Customer	Active	10 Jan 2025, 10:00 am	Edit Delete
Nikil03 A	nikil03@example.com	Customer	Active	10 Jan 2025, 10:00 am	Edit Delete
Nikil03 A	nikil03@example.com	Customer	Active	10 Jan 2025, 10:00 am	Edit Delete
Nikil03 A	nikil03@example.com	Customer	Active	10 Jan 2025, 10:00 am	Edit Delete
Nikil03 A	nikil03@example.com	Customer	Active	10 Jan 2025, 10:00 am	Edit Delete

Rows per page: 15 — << 1 2 3 >>

Figure 16:User Management Page



Add Product

Upload Images

Drop your files here, or

Choose Files Screenshot…120304.png

Screenshot 2025-04-28 120304.png
161.0 KB

Product Information

Product Name
Enter product name
Please fill out this field.

Category
-- Select Category --

Quantity
Enter quantity

Price (Rs.)
Enter price in Rs.

Description
Write a brief product description...

Add Product

Figure 17:Add product Form

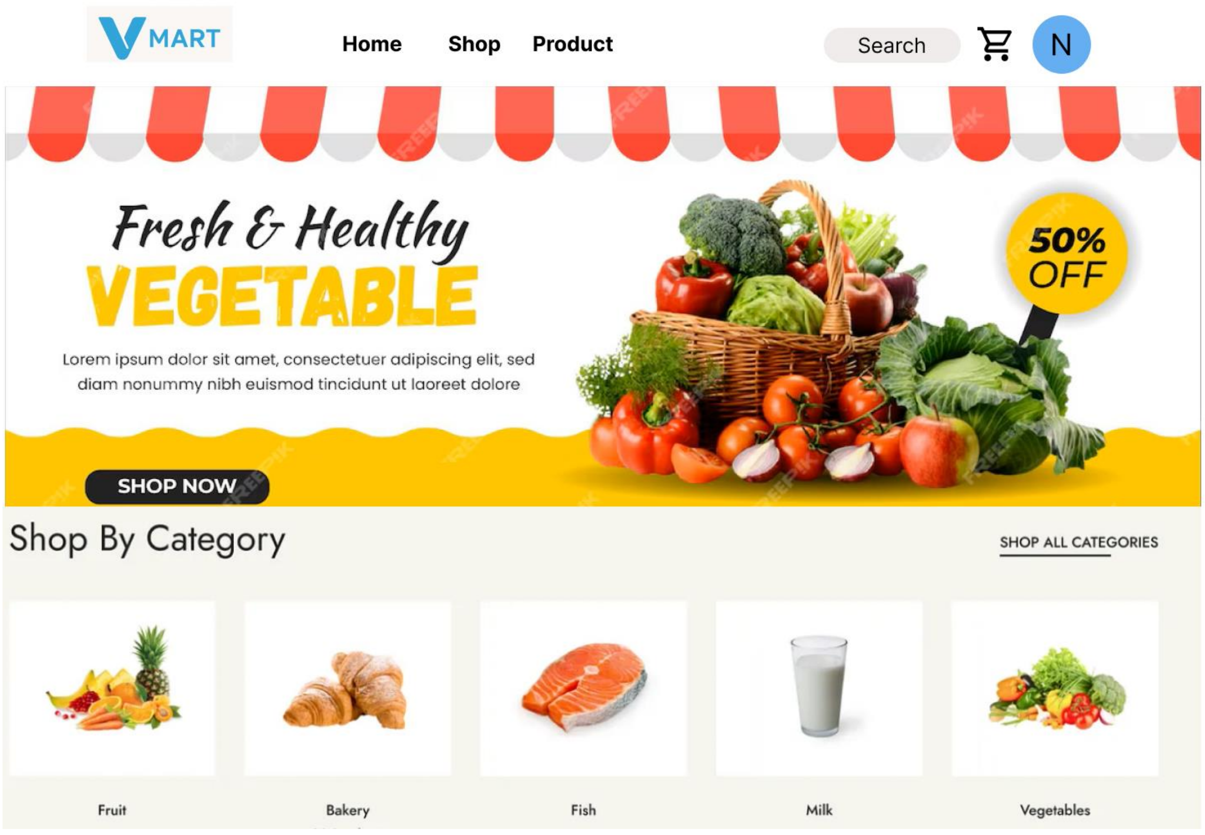


Figure 18:Home Page

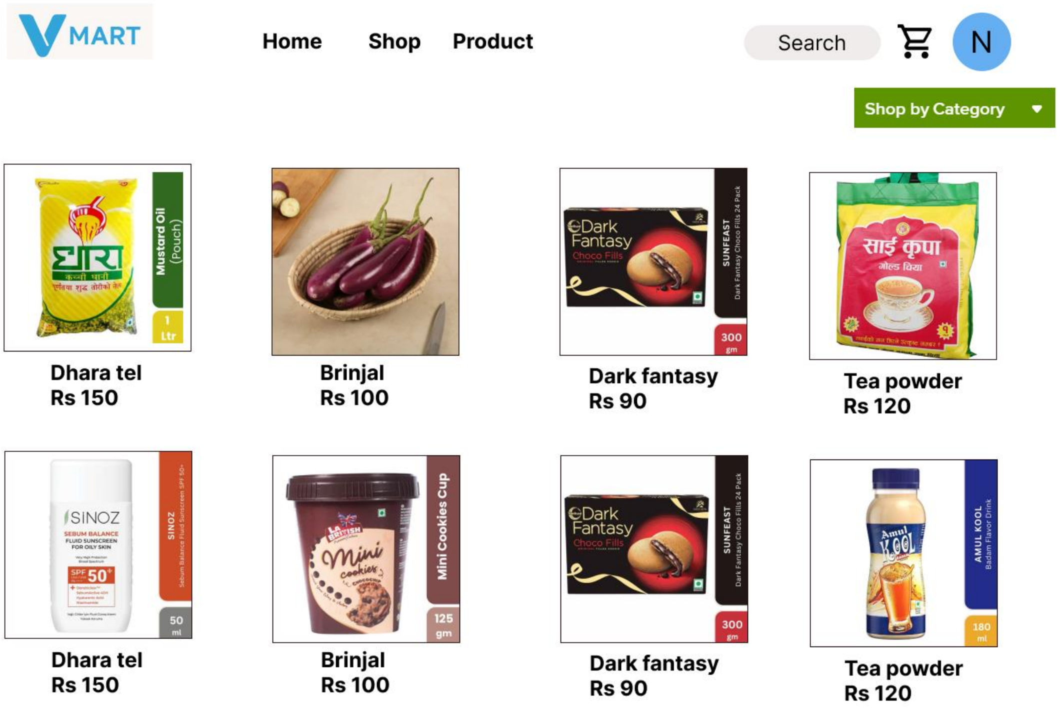
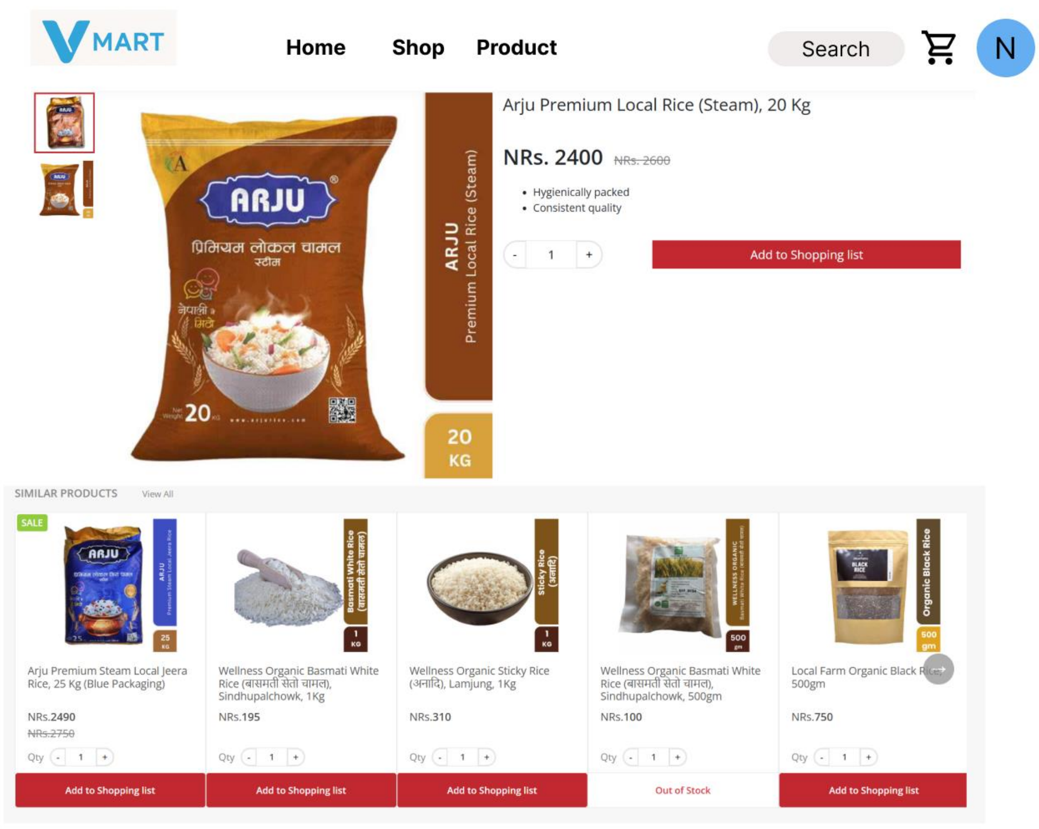


Figure 19:Shop Page



Arju Premium Local Rice (Steam), 20 Kg

NRs. 2400 ~~NRs. 2600~~

- Hygienically packed
- Consistent quality

Arju Premium Steam Local Jeera Rice, 25 Kg (Blue Packaging) NRs.2490 ~~NRs.2750~~

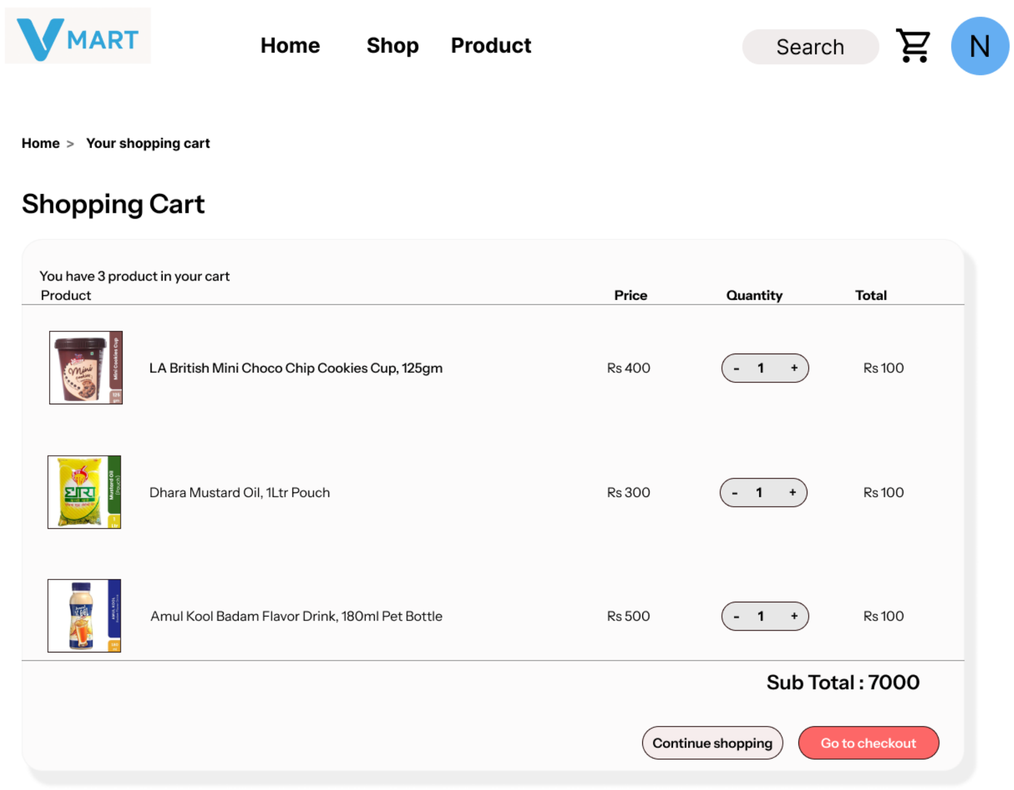
Wellness Organic Basmati White Rice (बासमती सेतो चामल), Sindhupalchowk, 1Kg NRs.195

Wellness Organic Sticky Rice (अनादि), Lamjung, 1Kg NRs.310

Wellness Organic Basmati White Rice (बासमती सेतो चामल), Sindhupalchowk, 500gm NRs.100

Local Farm Organic Black Rice, 500gm NRs.750

Figure 20:Product Details Page



Shopping Cart

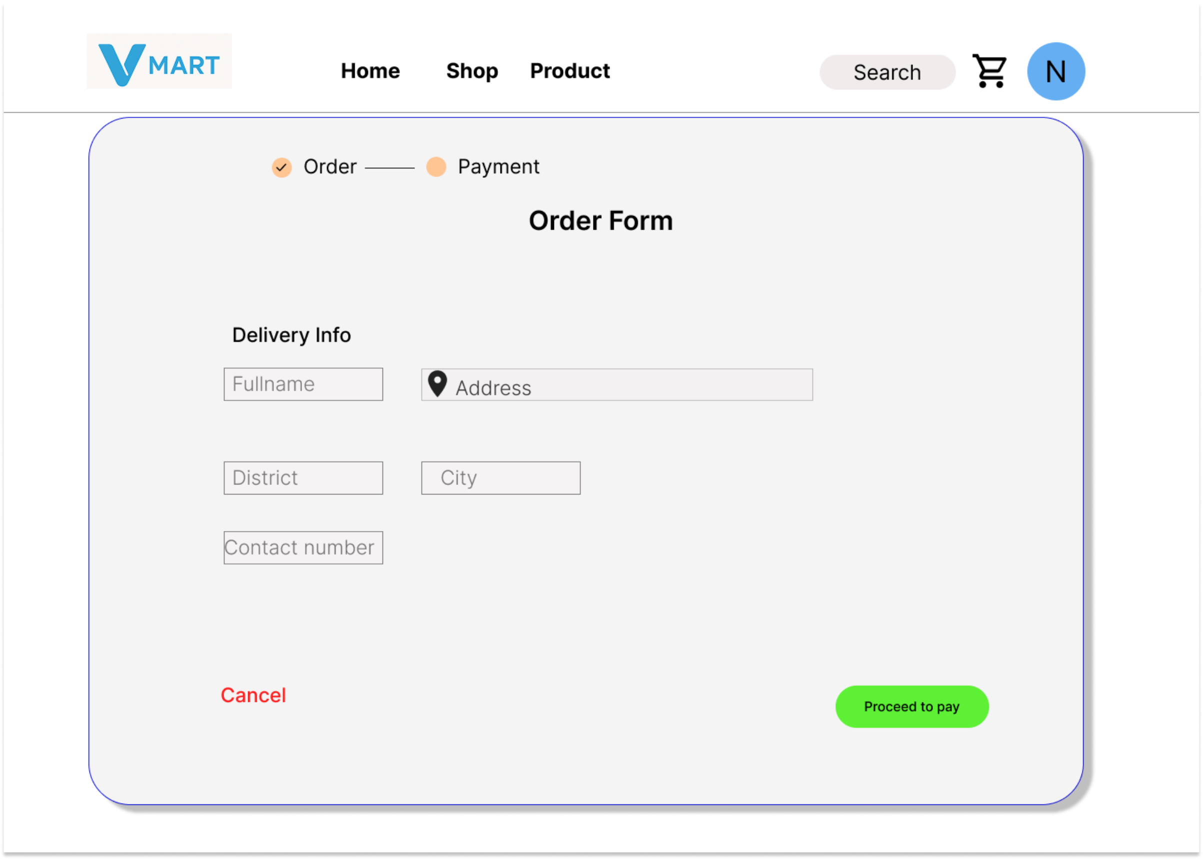
You have 3 product in your cart

Product	Price	Quantity	Total
LA British Mini Choco Chip Cookies Cup, 125gm	Rs 400	1	Rs 100
Dhara Mustard Oil, 1Ltr Pouch	Rs 300	1	Rs 100
Amul Kool Badam Flavor Drink, 180ml Pet Bottle	Rs 500	1	Rs 100

Sub Total : 7000

Figure 21: Cart Page

Figure 22:Payment Page



Order Form

Delivery Info

Fullname

Address

District

City

Contact number

Figure 231: Delivery Info Page

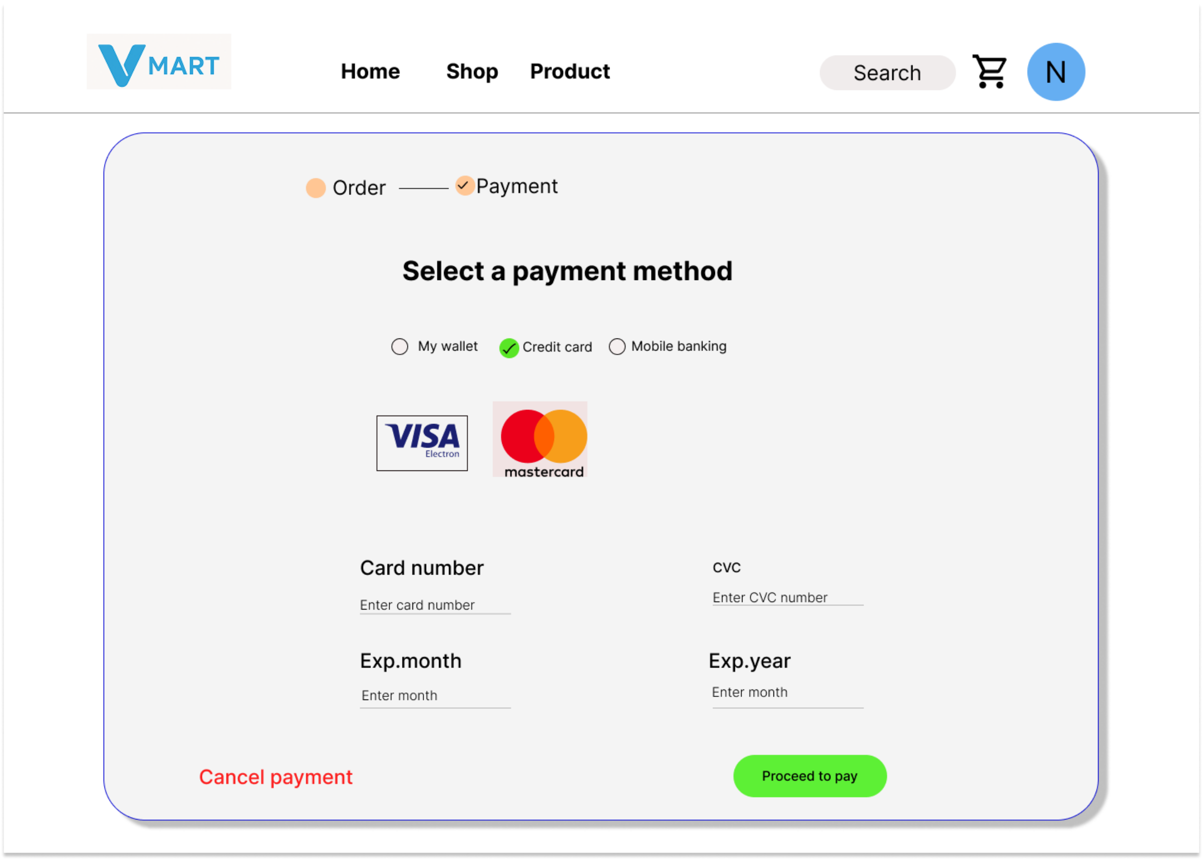
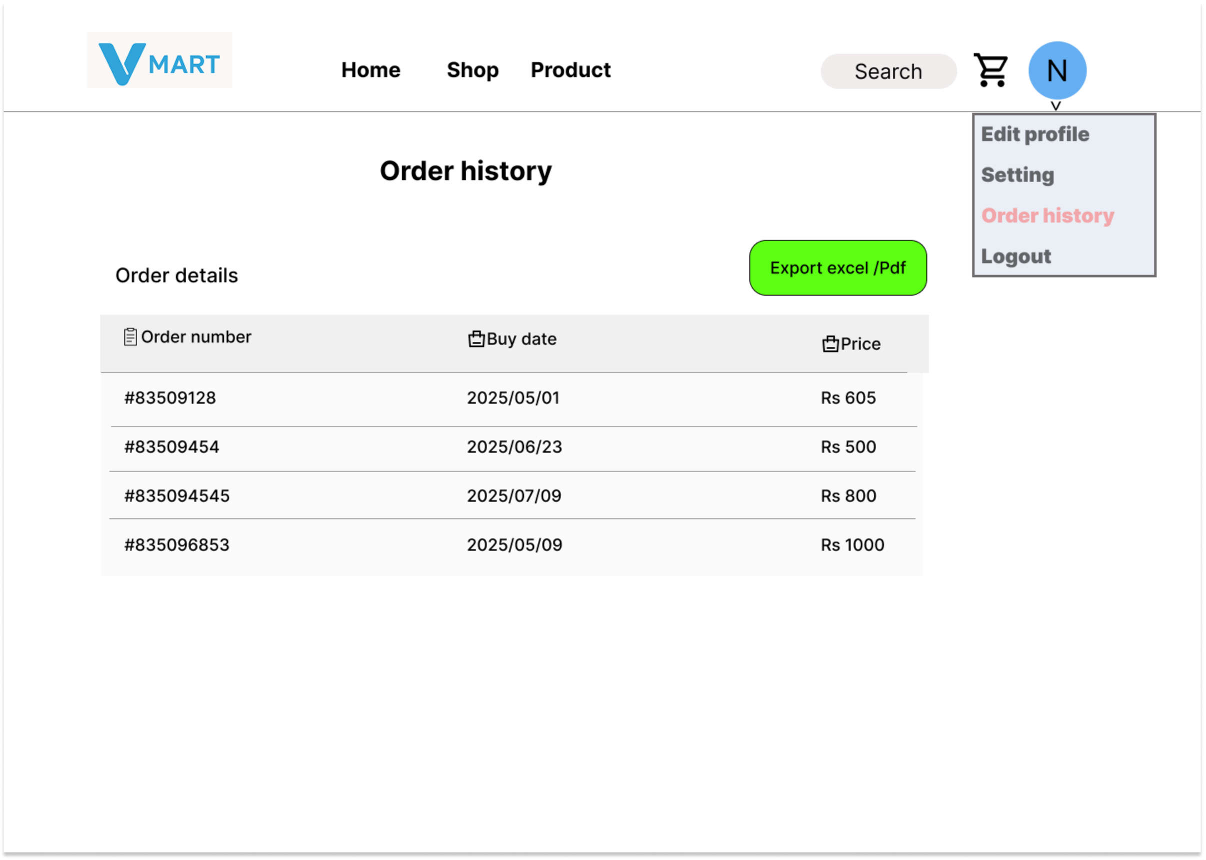


Figure 22:Payment Page



Order number	Buy date	Price
#83509128	2025/05/01	Rs 605
#83509454	2025/06/23	Rs 500
#835094545	2025/07/09	Rs 800
#835096853	2025/05/09	Rs 1000

Figure 24: Order Histor

10. Conclusion

This coursework describes the development of an Inventory Management System (IMS) for Global Tech Corporation, aimed at fixing the issues in their old system using a structured approach based on Object-Oriented Analysis and Design (OOAD). The project used UML diagrams like Use Case, Sequence, Activity, and Class diagrams to map out the system's behavior, interactions, and overall structure. We used Agile methods, particularly the Scrum framework, to work in iterative cycles, get constant feedback from stakeholders, and be flexible enough to adapt to changes as the project progressed. To make the system scalable, secure, and easy to maintain, we chose a Layered Architecture and the MVC design pattern. The project was built using technologies like React.js, Node.js, and MongoDB, which provided a solid base for the frontend, backend, and database. We also integrated Restful APIs, used Singleton and Observer patterns for improved efficiency, and set up automated CI/CD pipelines to help with real-time updates and seamless deployment.

For testing, we followed a rigorous strategy that included unit, integration, and regression testing to make sure the system was stable and reliable. A detailed maintenance plan was also created to handle future updates, bug fixes, and other necessary changes to keep the system running smoothly over time. The goal of this IMS is to improve the inventory management process in Nepal's warehouse operations, offering a solution that is scalable, user-friendly, and capable of increasing inventory accuracy, reducing costs, and improving customer satisfaction. This project not only provides a practical solution but also serves as a model for future software development projects.

References

GeeksforGeeks (2017). *Sequence Diagrams Unified Modeling Language (UML)*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/ [Accessed 12 May 2025].