CS5002NI Software Engineering

MAIN-SIT 35% Coursework

AY 2024-25

Credit: 30

Student Name: Aarya Paudel

London Met ID: 23048670

College ID: np01ai4a230109

Assignment Due Date: 12th May, 2025

Assignment Submission Date: 11th May, 2025

Word Count: 5430 words

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

23048670_Aarya_Paudel2.docx

Islington College,Nepal

Document Details

Submission ID
trn:oid:::3618:95359344

Submission Date
May 11, 2025, 10:01 PM GMT+5:45

Download Date
May 11, 2025, 10:02 PM GMT+5:45

File Name
23048670_Aarya_Paudel2.docx

File Size
31.7 KB

33 Pages

4,791 Words

26,347 Characters

6% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

26 Not Cited or Quoted 6%
Matches with neither in-text citation nor quotation marks

0 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

1% Internet sources

0% Publications

5% Submitted works (Student Papers)

Table of Contents

Table of Figures

Introduction

Global Tech Corporation has instigated the development of a revolutionized Inventory Management System within its facilities in Nepal. The system will facilitate optimization of warehouse operations. A significant driver for this development was the gaps identified in the previous system, which was the root cause of operational inefficiency, delays, and low customer satisfaction. Pressed by the felt need of a reliable inventory system, the present project tries to develop a lean and thorough designing and implementation strategy based on well-tested principles of Object-Oriented Analysis and Design.

The new system will ease a number of inventory management functions, such as user registration, product management, processing of purchases and sales, reporting, and secure payment handling. Both parties, mainly the admins and the users who are customers, are supplied with a new interface for better intuition, usability, and responsiveness in inventory management.

The report describes the design and development process of the system with a critical analysis towards the major architectural decisions, development methodologies, iterative refinements, and the testing strategies implemented. The system is aimed to be a robust and maintainable solution that eliminates all current known problems and faces yet to be defined new organization needs, so its design should clearly reflect the emphasis on usability, modularity, and performance.

Work Breakdown Structure (WBS)

The Work Breakdown Structure (WBS) follows an Iterative Waterfall Model, where each phase contains multiple iterations to refine and improve the system progressively.

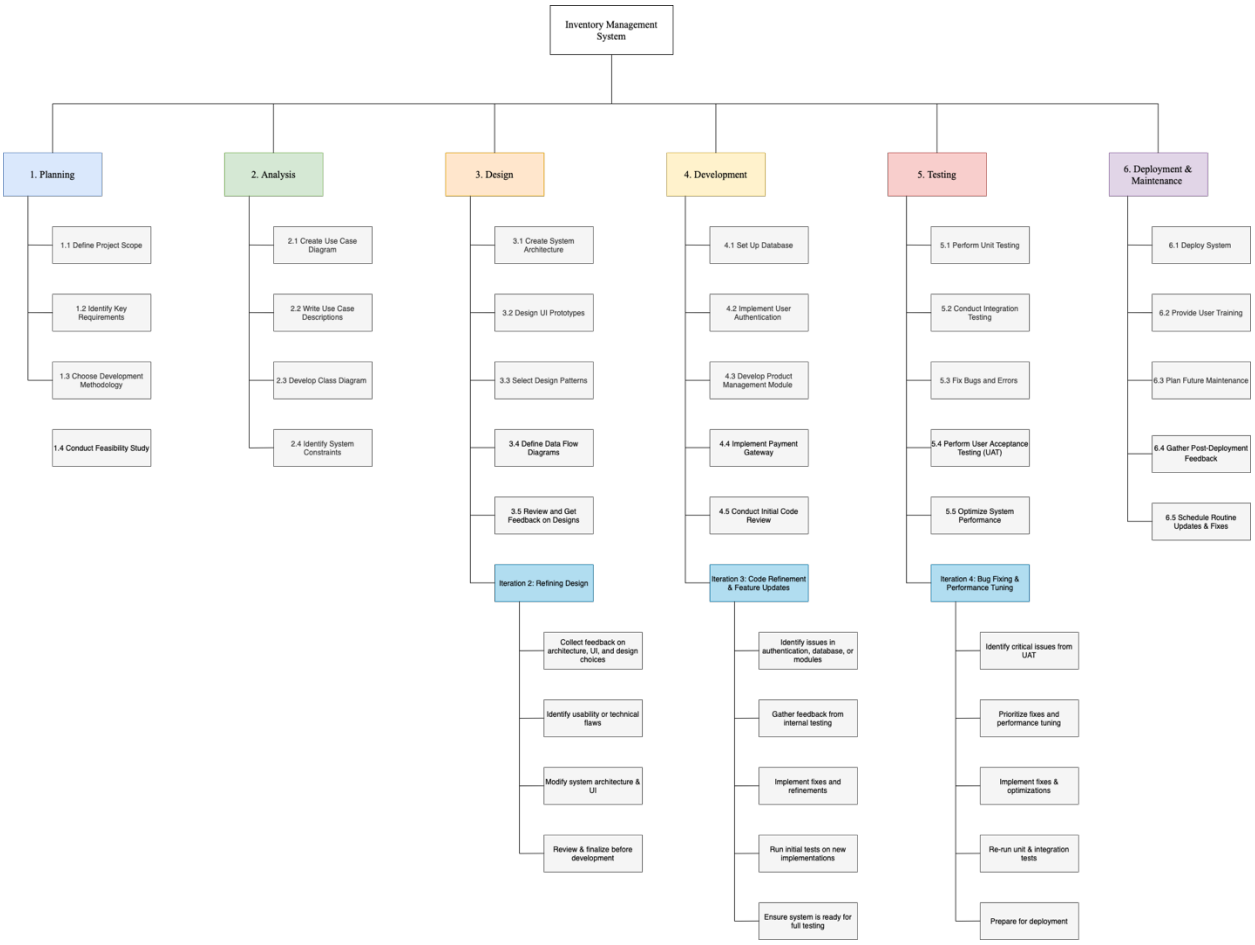

Figure 1: WBS

The inventory management system settlement plan provides an orderly, hierarchical decomposition of all work required at six main phases—that of a sequential, waterfall-like flow with embedded iterative elements.

1. Planning: The primary formative phase is very significant for establishing the foundation of a project. It is instrumental in finding direction and also in alignment. Major Informative Components:

    1.1 Defining Project Scope: This encapsulates a very clear definition of the boundary and objectives of the system. It eliminates scope creep and helps keep the project on course.

    1.2 Identify Key Requirements: This brings out the basic concept of gathering and documenting a set of key functionalities and features that the system has to offer. This will automatically lead to the needs for the subsequent phases.

    1.3 Choose a development methodology: The development methodology followed for this project is iterative waterfall. Implemented to set a context about how the project is going to be executed and managed.

    1.4 Feasibility Study Conduct: This refers to the highest level of probing into project viability with respect to technology, economy, and operations.

2. Analyze: This is another important phase dedicated to understanding the problem domain and the users who are there. Some informative points include:

2.1 Create Use Case Diagram: It is a kind of diagram that captures the interaction of system users with the system, giving a picture of different possible scenarios under which the system is going to be applied.

2.2 Specify Use Case: This can be detailed by checking whether each use case—its steps or high-level behaviors, preconditions, postconditions, or even what-if variations of the flow—can be described in depth to portray a real understanding of the given system's behavior.

2.3 Develop Class Diagram: Provides a structure that demonstrates entities within the system, their properties, and relationships—essentially a model of what entities are possible to be within the system.

2.4 Specify System Constraints: Limitations or restrictions on movement of data are identified in this phase at the early part of the cycle to give a direction to the design and development decisions much later.

3. Design: If requirements are the blueprint for a system, then consider this phase the generation of the blueprint of the blueprint. Major Informative Elements:

3.1 Create the System Architecture: Definition of the topmost structure with its parts and how they work together. Will help make the system scalable and maintainable.

3.2 Design the UI Prototypes: Making a visual approach to the user interface will again reduce the rate of leading to success fast and easily, ensuring that the system is user-friendly.

3.3 Select Design Patterns: This often increases the quality of the system at hand, maintainability, and even efficiency.

3.4 Outline Data Flow Diagrams: These are going to help show the path data will be moving within the system; this will be very helpful in understanding details of data processing and storage.

Iteration 1: Design Refinement: This will involve the iteration aspect itself showing where some kind of initial design was reviewed, feedback collected for rework, and then reworked design produced. This is more of an indication that improvement remains an ongoing process for progress.

4. Development: Coding and implementation are going to follow this. Major Informative Elements:

4.1 Setup of the Database: The part is going to be established at the very start, relating to setting the base for the storage part of the data.

4.2 Implement User Authentication: It will be one of the most important features to be implemented for the secured login to the system.

4.3 Product Management: This is one of the core areas of functionality represented within the inventory system.

4.4 Payment Gateway Integration: This is very important in the case where there are transactions involved in the system.

4.5 Initial code review: As is probably usual, this way in the initial stages it can help reveal and correct issues to increase the quality of the code.

Iteration 2: Code Refinement & Feature Addition: This also includes further iterations in improving the existing code and integrating new features as required.

5. Testing: It is a process to validate that the system is working correctly and meeting its specified requirements. Major Informative Points:

5.1 Unit Testing: Here, the individual components of the system are tested in isolation in order to verify that each works correctly.

5.2 Integration Tests: This demonstrates whether each part of the system collaborates.

5.3 Fixing Bugs and Errors: This is part of the testing process to ensure that the system is safe and reliable.

5.4 User Acceptance Testing: Testing done by the end-users it is tested, ensuring that it fulfills the requirements and is effective.

5.5 Performance testing: This is done through studying how quick and stable the system can be when subjected to partial or full user loads.

Iteration 3: Bug Fixing & Performance Tuning: Iteration number three, therefore, also focuses on fixing whatever issues there were found during testing and tuning, so the application would perform better.

6. Deployment and Maintenance: Deployment of the system into an operational environment calls for continued software support. Major Informative Aspects:

6.1 Deploy the system: The process of making it presentable to the end user.

6.2 Train users: Ensuring that the new system is well known to the users.

6.3 Maintenance planning: Planning for support, updates, and possible bug fixing.

6.4 Get Feedback after Deployment: Get quantified user experience data in the system for improvement.

6.5 Schedule Maintenance and Fixes: Arrange a timetable in order for the system to operate continuously. The iterative mechanisms in the Design, Development, and Testing phases are predominately informative as an indication of a commitment instilled within the development life cycle to listen to feedback and improve thereon. This slightly reduces the risk associated with a purely linear waterfall model. At the end of the day, the WBS is an informative structural breakdown of a project into manageable components that also gives the reader a peek into the major activities, deliverables, and the development process of this Inventory Management System.

Gantt chart

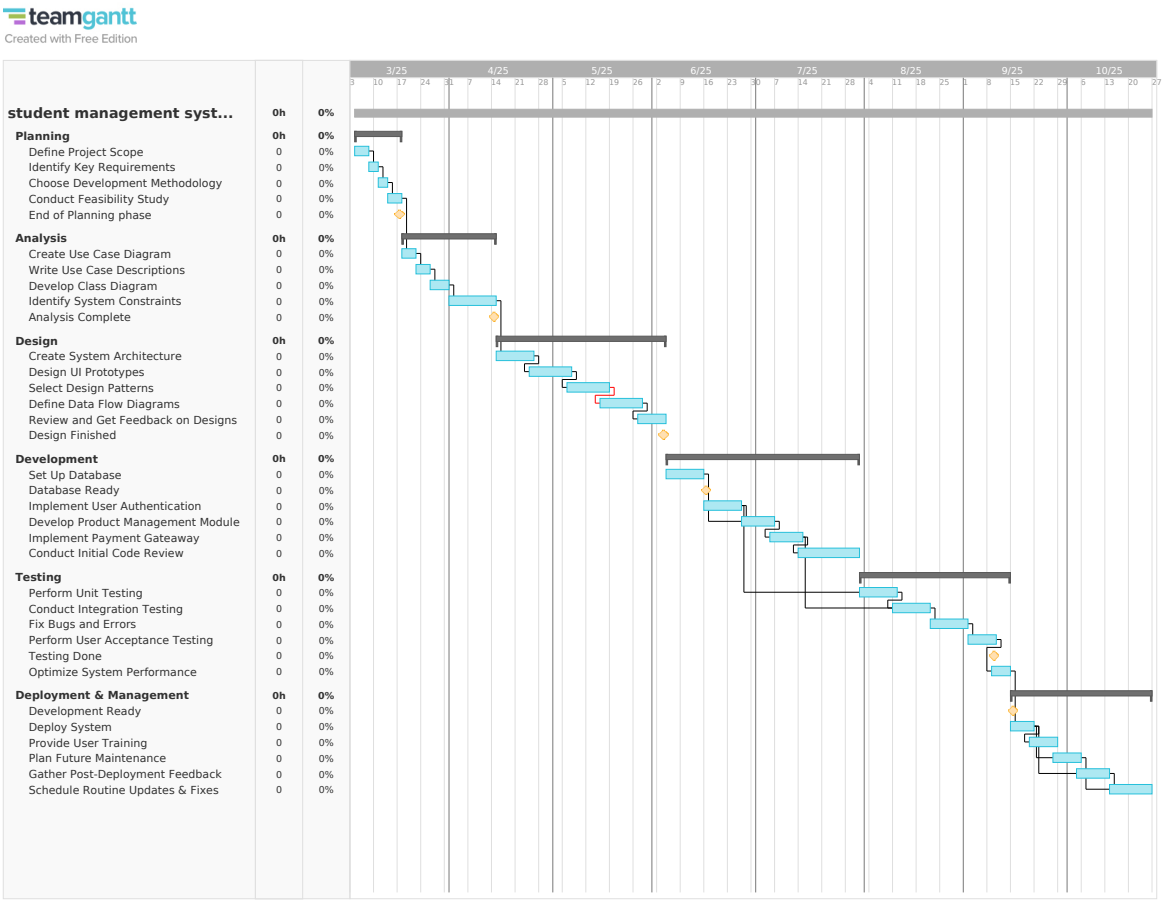

Figure 2: Gantt Chart

Use Case Diagram

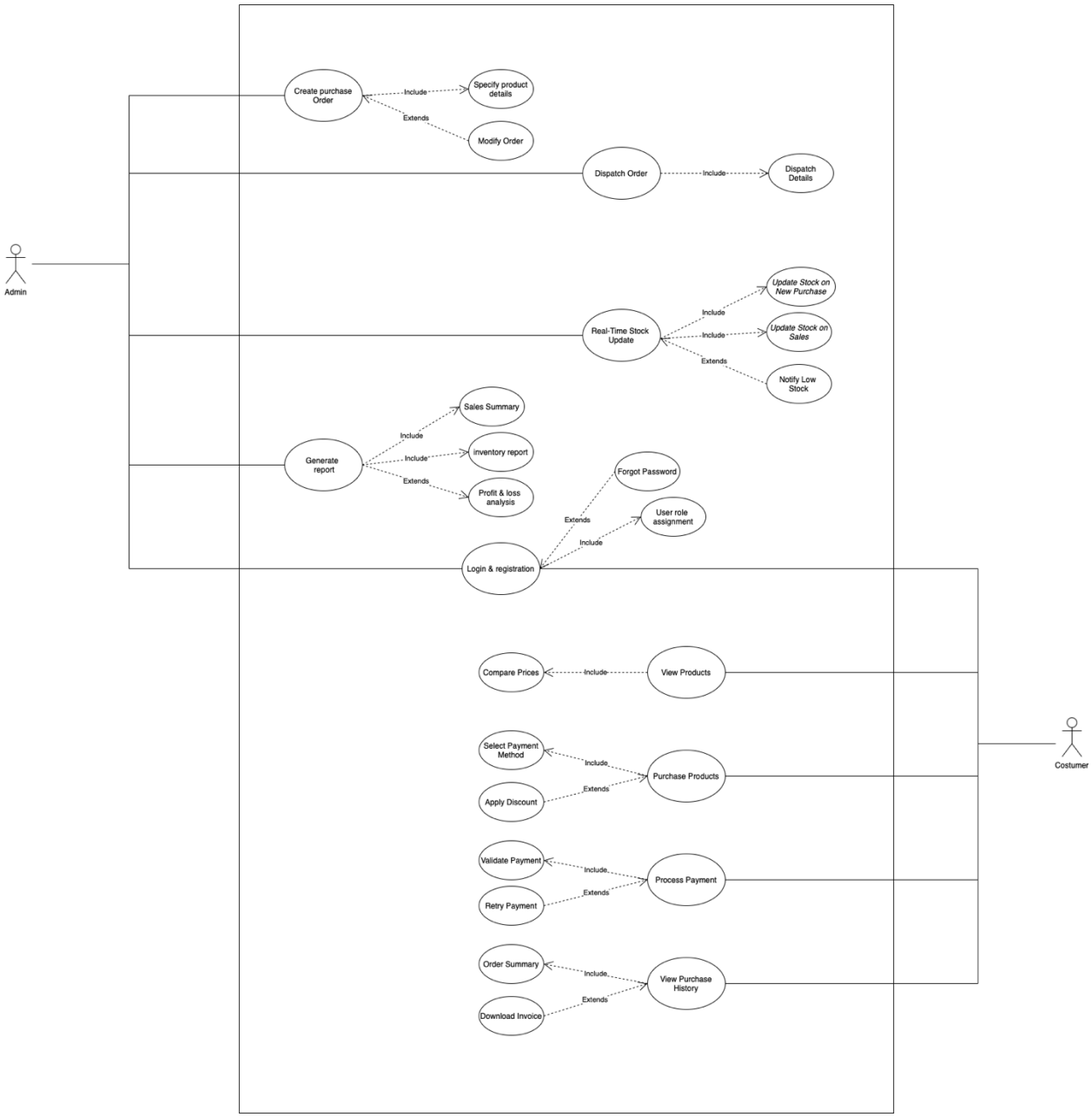

Figure 3: Use Case Diagram

The use case diagram represents the various ways in which users, more specifically Admin and Consumer, are going to interact with the system by bringing out the major functionalities of the system and who can use them.

Actors:

- Admin: On the left, it's a stick figure representing Admin, with a broad range of functionalities related to managing the system and inventory.
- Consumer: On the right, the stick figure represents Consumers, who interact with the system mostly in making his purchase decisions and managing the orders; this is a process in the purchasing lifecycle.

Use case (Ovals):

These are specific functionalities or goals that an actor can attend to by interacting with the system.

Admin Use Cases:

- Create purchase order: Admin has the ability to raise a new purchase order.
  - (Include) Specify the product details: It is a necessary step for the admin to raise a purchase order.
- Modify Order: An Admin can update his old Orders.
- Dispatch Orders: He can raise a request to dispatch the orders.
  - (Include) Generate Dispatch Labels: Dispatching an order is not complete without generating labels to put on it.
- Print Time Batch Update: This is nothing but printing offsets on time-based batches that most probably an administrator has something to do with in relation to inventory or production.
  - (Include) Update Stock on New Purchase: When new items are purchased, the price of stocks gets updated.
  - (Include) Update Stock on Sales: In the case of selling any item, accordingly, update the stock level.
  - (Extend) Notify Low Stock: Will extend this functionality getting triggered during a stock updation, which notifies the admin in case a stock goes below the certain limit.
- Generate report: The Admin can make reports.
  - (Include) Sales Summary: This is a brief of all the sales data.
  - (Extend) Inventory report: Optionally, the admin can get a report on the inventory details that are present at the moment.
  - (Extend) Profit & loss analysis: Optionally, the admin can extend his/herself to the report analyzing Profit & Loss.
- Login & registration: Admin Logout and registration back into the system, probably a subphase of an admin registration for creating a new account.
  - (Include) Forgot Password: There must be a provision for the Admin in the system to retrieve a lost password.
  - (Include) User role assignment: The Admin can configure user roles and permissions within the system.

Consumer Use Cases:

- View Products: Consumers should be able to get into product listing and view the products that are on offer.
    - (Include) Compare Prices: It is when the consumers get to compare prices for different products that they require.
- Select Payment Method: A consumer has to select How to Pay for his order.
- Apply Discount: It allows customers to apply discount codes or avail in-store offers.
    - (Extend) Purchase Products: This could be applied to the process of purchasing, which the discount may be related to.
- Update Payment: Customer can update his payment details.
- Retry Payment: If it fails, the customer can retry the payment.
    - (Extend) Process Payment: Retrying the payment is an act of the entire process of processing the payment.
- Order Summary: Consumers can view a summary of their current or past orders.
- View Purchase History: Customers can view their purchase history.
    - (Extend) Download Invoice: Customer is able to download invoices for orders. Include: Marks that all the base use cases demonstrate explicit behavior from another use case. The relationship means that the included use case is a part of the base use case. This relationship has been exemplified by using the 'include' and 'extend' relationships.

In simpler representations, this diagram improves on the visibility pertaining to what all things the system is going to allow the users to perform or take action on or against an Admin or a Consumer. It signals the primary activities of order management processes, updating inventory, and reporting for an Admin, and on the other hand, browsing products, completing purchases, and the history of orders for a Consumer. Further explanation of how they relate to each other is provided through the 'include' and 'extend' explanation for calling auxiliary processes.

High level use case descriptions

a. Use Case: Create Purchase Order

Actors: Admin (initiator)

Description: The admin creates a purchase order by specifying product details. The order can be modified before finalization.

b. Use Case: Dispatch Order

Actors: Admin (initiator)

Description: The admin processes and dispatches an order by providing dispatch details.

c. Use Case: Manage Products

Actors: Admin (initiator)

Description: The admin manages product inventory by adding, updating, or removing products.

d. Use Case: Real-Time Stock Update

Actors: Admin (initiator)

Description: The system updates stock levels automatically based on new purchases and sales. It notifies the admin when stock levels are low.

e. Use Case: Generate Report

Actors: Admin (initiator)

Description: The admin generates different reports such as sales summaries, inventory reports, and profit & loss analysis.

f. Use Case: Login & Registration

Actors: Admin (initiator)

Description: The admin registers or logs into the system, manages user roles, and can recover forgotten passwords.

g. Use Case: Compare Prices

Actors: Customer (initiator)

Description: The customer compares product prices by viewing available products.

h. Use Case: Purchase Products

Actors: Customer (initiator)

Description: The customer selects a payment method, applies discounts if available, and proceeds with purchasing products.

i. Use Case: Process Payment

Actors: Customer (initiator)

Description: The system validates and processes the payment. If needed, the customer can retry payment in case of failure.

j. Use Case: View Purchase History

Actors: Customer (initiator)

Description: The customer reviews past purchases, views order summaries, payment records, and downloads invoices.

Expanded use case descriptions

a. Login & Registration

Use Case: Login & Registration

Actors: Admin, Customer

Description: Users log in or register to access the system.

Typical course of events:

Actor Action	System Response
1. User provides login credentials or registers.	2. System validates user details.
3. If valid, user is granted access.	4. Display user dashboard.

Alternative flows:

- If the credentials are incorrect , prompt for retry.
- If new user, request additional registration details.

b. Process payment

Use Case: Process Payment

Actors: Customer

Description: Customers complete payment for purchases **Typical course of events:**

Actor Action	System Response
1. Customer selects payment methos.	2. System displays payment options.
3. Customer enters payment details.	4. System validates payment.
5. Customer confirms payment.	6. System processes transaction and updates order status

Alternative flows:

- If payment fails, system notifies customer and slows retry.
- If transaction is successful, system sends confirmation email.

Communication diagram

A communication diagram is a really simple UML diagram. It depicts very simply how parts of a system interrelate with each other to perform an action. Basically, it is a drawing of a project that needs to be executed; in this case, the execution involves various objects or team members, sending messages to complete the task. For our IMS, it is used to outline the process of the "Registration," where the formation of a new user is shown.

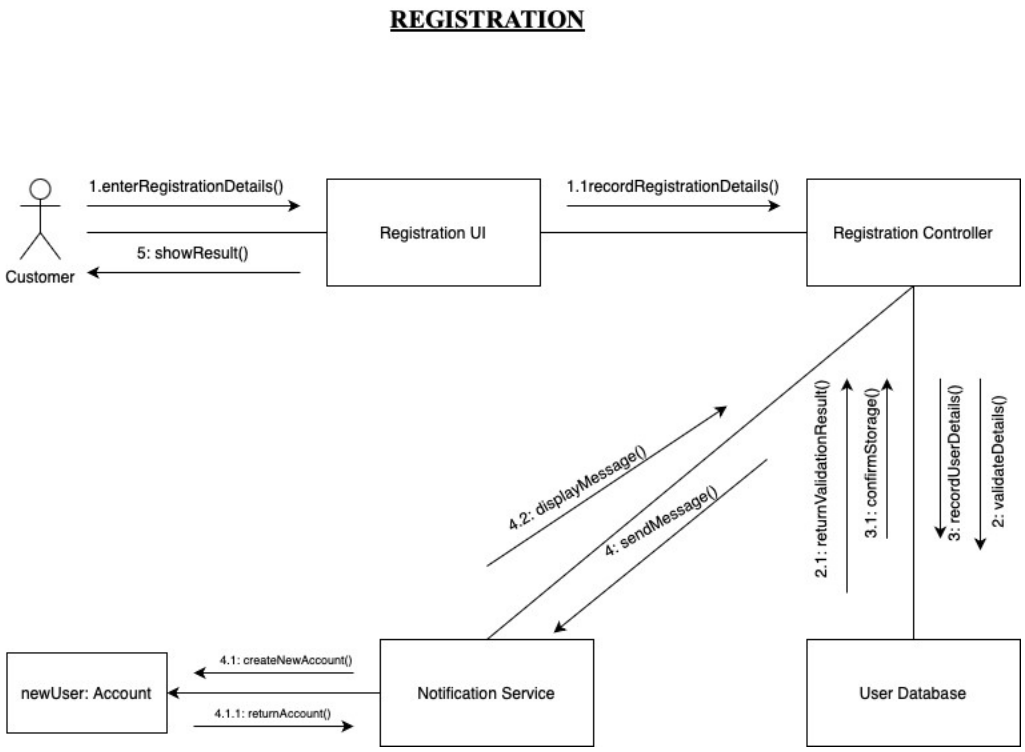

Figure 4: Communication Diagram

The communication diagram provides an insight into the various steps followed during customer registration. It shows the communication between several objects or parts of a system needed to open a new account. So now, let's see the breakdown of the interactions:

Objects/Components:

- Customer: The actor that started the registration procedure, in the form of a stick figure.
- Registration UI: Front-end provided to the user for registration details. Drawn in rectangular form.
- Registration Controller: A unit that will handle registering logic and coordinate intercommunications with other components. Represented by a rectangle.
- User Database: The database of the system where information about the user's account is stored. Represented by the rectangle.
- Notification Service: A service related to the delivery of a notification message, probably an email or SMS, to the new user when it is successfully registered. Represented by a rectangle.
- newUser: Account: This is the new account of the user, represented with a rectangle denoting the name of the object and the class.

Messages (Numbered Arrows):

Each of these arrows shows communication between the objects and numbers to identify the sequence of interactions.

1. customer → Registration UI: enterRegistrationDetails()
   This is the point where the user will make an initial request for registration by filling in information such as name and email and password through the respected Registration UI.

   1.1. Registration UI → Registration Controller: recordRegistrationDetails()

   Thus, Registration UI takes this data from the Customer and sends it to the Registration Controller for further processing.

2. Registration Controller → User Database: validateDetails()

The Registration Controller passes the registration details received on to the User Database to validate, ensuring that the email is still not in use at this point.

2.1. User Database → Registration Controller: returnValidationResult()

Therefore, from the above, the result of the validation has returned form User Database back to the Registration Controller with this message: the details are valid or not.

3. Registration Controller -> User Database: recordUserDetails()

Therefore, in the case of validating the Registration details as correct, he boils down to the Registration Controller requiring to brush off new user information within User Database.

3.1. User Database -> Registration Controller: confirmStorage()

The User Database acknowledges the proper storage of these new details of the user.

4. Registration Controller -> Notification Service: sendMessage()

   The Registration Controller will tell the Notification Service to serve up a message to the newly registered user confirming successful registration. For instance, this simple message could be a "welcome" email.

   4.1. Notification Service -> newUser: Account: createNewAccount()

   The Notification Service works on the newUser Account object—perhaps, at last, it creates or sets the initial properties.

   4.1.1. newUser: Account -> Notification Service: returnAccount()

   Then, the newly created Account perhaps sends back some information to the Notification Service that can be used to include in the confirmation message.

   4.2. Notification Service -> Registration UI: displayMessage()

   So, Notification Service informs the Registration UI, or it could be the Registration Controller, after messaging is sent, about the success or failure of the registration process, along with relevant messages to display to the Customer.

5. Registration UI -> Customer: showResult()

   The Registration UI finally reveals the results of registration to the Customer: success or failure.

Indeed, this gives a very simple sequential flow that the Registration UI acquires data from the user, the Registration Controller verifies and stores the user data into the Database, and the Notification Service will then provide notice about this operation to the same Database; finally, the UI informs the user about the results of the entire process.

Activity diagram

Activity diagram is a UML flow chart representing how a particular process is divided into steps, showing the sequence of events with their actors taking part in it. Just like a recipe for your favorite dish that enlists every action. So we are using it to show that in IMS, "Process Payment," payments are processed without any hassles to the customer.

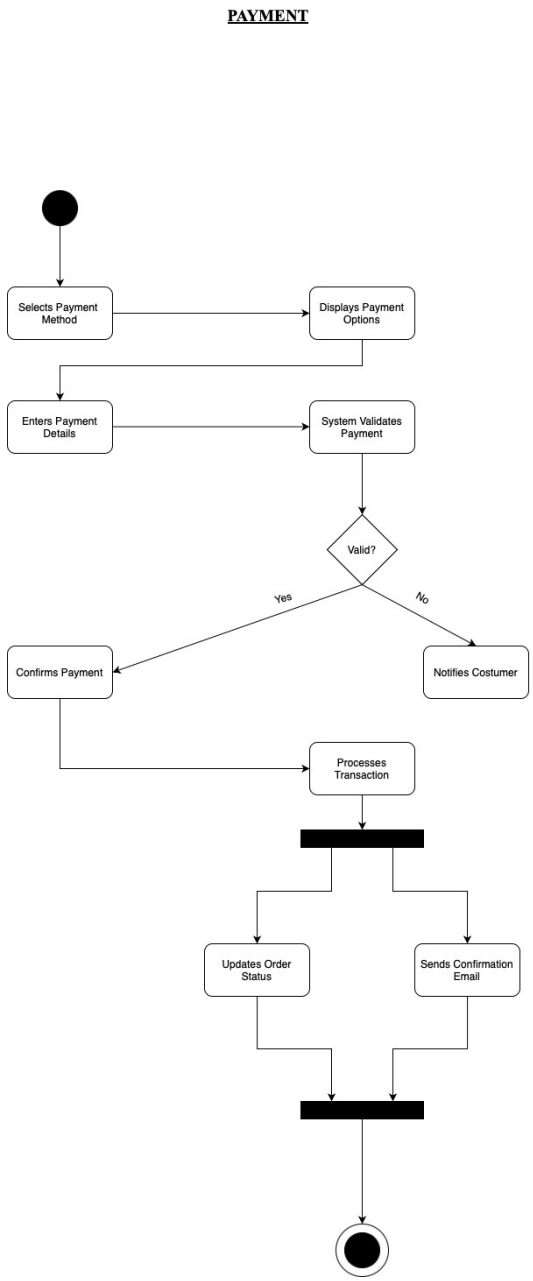

Figure 5: Activity Diagram

The following payment process flow activity diagram shows the payment action sequence that a user has to go through. It includes:

- Start (Black Circle): Begins the process.
- [Selects Payment Method]: The user selects the method of payment (e.g., credit card, or PayPal).
- [Displays Payment Options]: Based on the method selected, the system shall present the relevant input fields or options.
- [Enters Payment Details]: The user provides the payment information required for the operation (e.g., card number, expiry date).
- [System Validates Payment]: The system then verifies whether the entered payment details are correct and valid.
- Decision Diamond [Valid?]: Based on the validation result, the flow shall split.
    - Yes: If yes, the process or sub-process moved to this point is to confirm the payment.
    - No: The processor informs the user [Notifies Customer] about the invalidity of the payment details.
- [Confirms Payment]: Final confirmation on the payment made by the user.
- [Processes Transaction]: The actual trigger for processing the payment with the payment gateway is done by the system.
- Fork (Horizontal Bar): Upon acceptance of the transaction, the process will then be forked into parallel activities.
    - [Updates Order Status]: The system will update the status of payment in the order details.
    - [Sends Confirmation Email]: An email will be sent for the confirmation of the order by the system to the user.
- Join (Horizontal Bar): All parallel activities must be completed before continuing the process.
- End (Bullseye): The result of the payment process.

In simple terms, the diagram sets out selecting the payment method, providing the details and making an entry for validation and processing with updating the order and notifying the customer at the end of the process. It also describes a scenario where the payment details are found invalid.

Class Diagram

A class diagram is a sort of static structure diagram that describes the classes present in an articulated system, in addition, to establish the relationships between these classes. So effectively, it provides a blueprint regarding how data and functionality is organized in a system. It includes classes themselves, attributes of classes like fields, operations, and relationships between objects. This system will have:

- User is a basic class with two specializations: Admin and Customer.
- The Admin sends user lists, accesses the reports of the system.
- The Customer fills in numerous purchase orders, and their histories will be recorded correspondingly.

Purchase History tracks a customer's purchases, and Reports if desired. Purchase Order contains those products selected by any customer to buy. Payment will have the details of the process of payment of that purchase order.

Further, after the payment is completed, an order dispatch is generated, which is used to ship the items.

An invoice is generated after dispatching any order for formal billing.

Product represents the entities put on sale, while Stock keeps the inventory required for the products. Aggregates Suppliers with Stocks maintaining their products. It exhibits the relating of the following categories:

- Inheritance - Admin and Customer inherit from User.
- Association - Founded Link
- Aggregation - Created
- Dependency - Generates

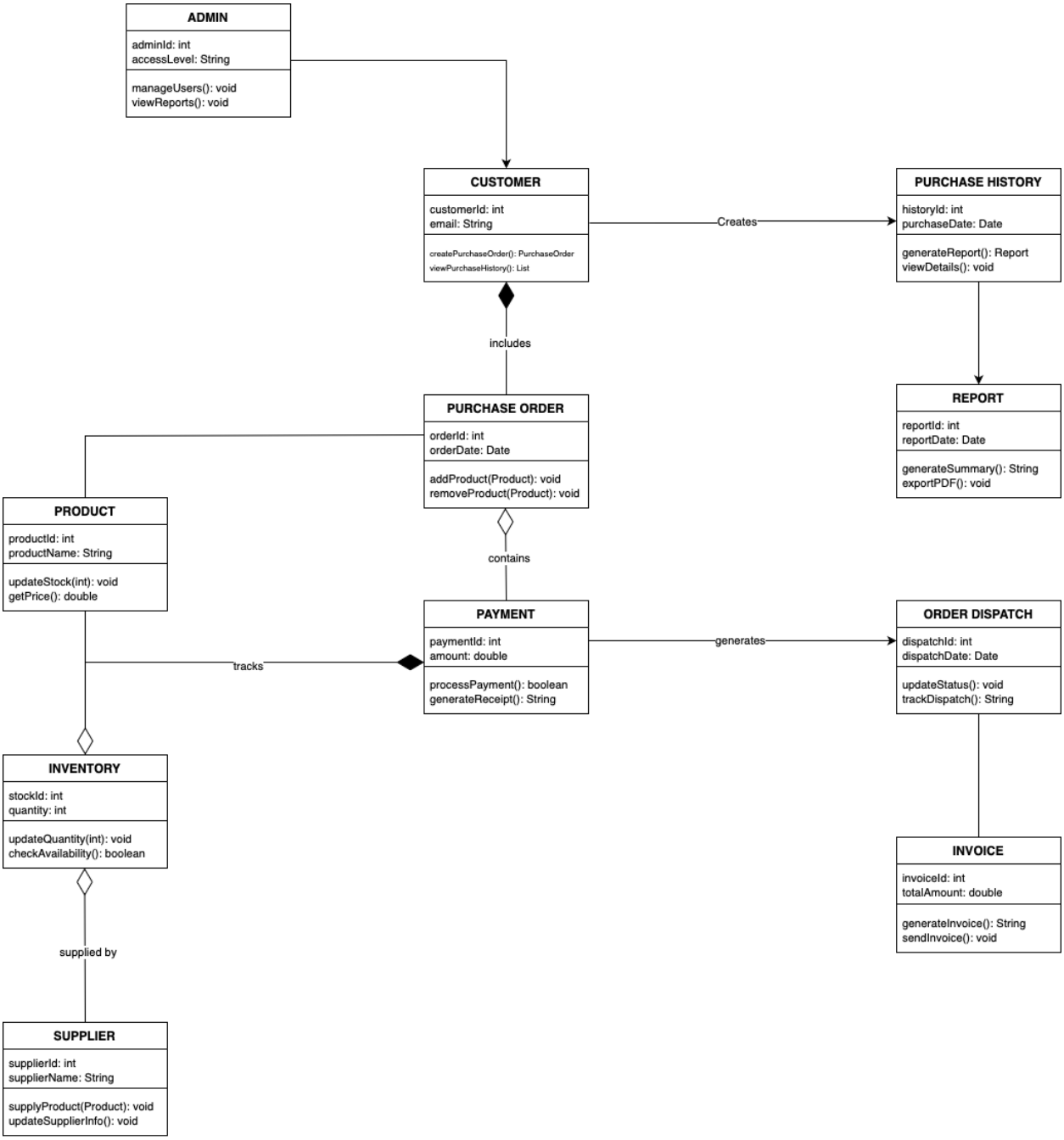

Figure 6: Class Diagram

Description of further development section

a. Architectural Choice

I opted for a three-layered architecture of Presentation, Business Logic, and Data Access to the Inventory Management System. It is not arbitrary; it has been well thought of on a realistic basis with regard to maintainability and the effective collaboration of teams. Having a clear separation of concerns will enable me to isolate changes in the UI from business logic or update changes in the database, thus potentially reducing accidental breakages.

In addition, since it is in Iterative Waterfall, every stage, but particularly in the design and development, is getting refined through incrementality, which makes it combine well with the layered approach. Assuming one point of feedback is that payment logic has to be changed for some reason, I could rectify that mishap at the independent layer without tweaking the whole system again. Such a setup would help me in setting an architectural mindset for things like using PostgreSQL—it is rock-solid, full of advanced querying capabilities, and integrates nicely with Django's ORM to help me enforce consistency at a data access layer.

b. Design Patterns

The patterns I specifically selected were thought to be very useful at the time of the development:

- Factory Pattern:

  This was used in managing the creation of different user roles, specifically for the Admin, Customer. This removes all the scattered if-else logic or manual object instantiation; it thereby centralizes and simplifies by using factory. I found it very handy while simulating logins in the early stage of development. The switching of user types was seamless.

- Observer Pattern:

  These stock/inventory updates would, in turn, result in changes to the whole system—UI, report module. I had no intention of making modules tightly coupled and stock changes need to broadcast automatically without the UI asking every time. This will help me especially when I plug in real-time dashboard features later.

- Strategy Pattern:

  The different payment methods need to have dynamic selection logic. As to avoid hard coding behaviors like these in the application, I was getting to the point where it seemed doable with a Strategy to be able to swap out algorithms at runtime without breaking the payment flow. It proved useful during prototyping, which allows me to test different scenarios by just changing a method call.

I have not selected them just because they are very popular but because they solve real issues in my planning phase of making my codebase more future-proof.

c. Development Plan

My approach was based on the Iterative Waterfall model methodology, an essentially structured model that provided for planning, design, development, and testing phases, with re-iteration through them based on feedback. Everything went on seamlessly; it was quite fine that there were several refinements during UI and functionality implementations.

Below are the tools and technologies that I used:

- Backend: Java (using the Eclipse IDE)

  Java works perfectly fine in creating an object-oriented strong base to execute core system logic, right from user authentication up to updating inventories and processing payments. Java popped right into my mind because of its strong support for modular development, which hung well with the layered architecture I offered. The choice that made Eclipse IDE was because of its integration features by debug support, that eased the way handling multiclass Java projects.

- Frontend: HTML (embedded Java, and simple CSS)

  The front end was implemented in HTML and styled with very basic CSS. It has been further enhanced with Java (Servlets or JSP), which facilitated dynamic rendering of pages. Form submission, viewing of inventories, and confirmation of orders ensued, while the backend logic and user interactions were easily integrated.

- Database: PostgreSQL

  I chose it because of being reliable and its great querying capacity to distance itself from the number of complicated queries at the time of filtering stock of products and creating reports for transactions. It had good support for integration with Java through JDBC drivers.

- Version Control System: Git

Git was used as a version control tool to maintain all revisions while developing. At every phase end, a commit was taken to be stable that can be reverted within no time or moved ahead with features as per need.

The major development steps included:

1. Design wireframes for UI and database schema.

2. Developed individual modules such as login, stock management module, and ordering module, etc., using Java classes and HTML pages.

3. All the modules successfully integrate with each other, and data flow checked between frontend forms and backend logic.

4.Test, debug, and iterate functionalities based on the feedback received.

The development schedule kept me focused but still allowed enough leeway to improve the system as requirements evolved.

d. Testing Plan

The way I viewed testing was not as a last-minute checkbox; I had thought of it as something that was supposed to be carried out progressively from the very beginning during development.

- Unit Testing: I checked single elements like payment-processing logic, form input validations, etc., which helped me find type and logic errors at an early stage.

- Integration Testing: Ordering and inventory modules linked together through the main focus was data flow between them. Examples were that of checking products' removals to maintain synchronization with inventory and purchase history records.

- System Testing: Everything was cross-verified against the original requirements. Having a set of real-user scenarios made it easier for me to act as a customer and complete check out procedures or generate his/her invoices.

- UAT (User Acceptance Testing): Peer feedbacks from the fellow who acted as users were very useful in refining form layouts, button placements, and readability of reports.

The idea was that the end product would not only be bug-free but also refined in usability and responsiveness to perform optimally in realistic usage scenarios.

e. Maintenance Plan

Maintenance isn't just about fixing problems once a system is already in place, but also about designing a system from day one that makes it easy to maintain. Corrective maintenance In this situation, all issues reported by the users were logged in a small document to be worked on transparently.

- Adaptive Maintenance: I have made my codebase flexible and the UI components have been made very flexible. For example, if tomorrow I have to change my database from PostgreSQL to MySQL or change UI libraries, I need to make minimal change as my design is layered.
- Perfective Maintenance: User feedback is logged and reviewed for feature improvements. For example, such a future feature might be low stock alerts—the system is built to allow the addition of this without reworking core logic.

Maintenance updates will be bunched under scheduled cycles, and not just spontaneous changes, so that the user stays disruption-free and testing remains smooth.

Prototypes

a. Login page

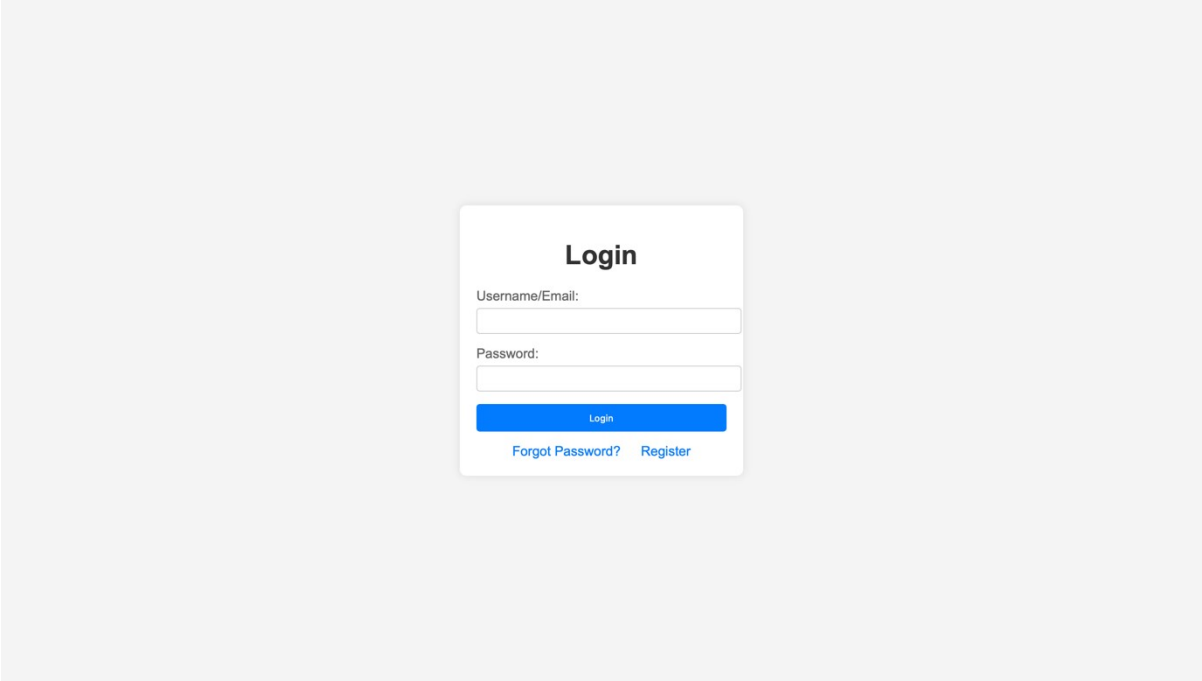

Figure 7: Login Page

b. Registration page

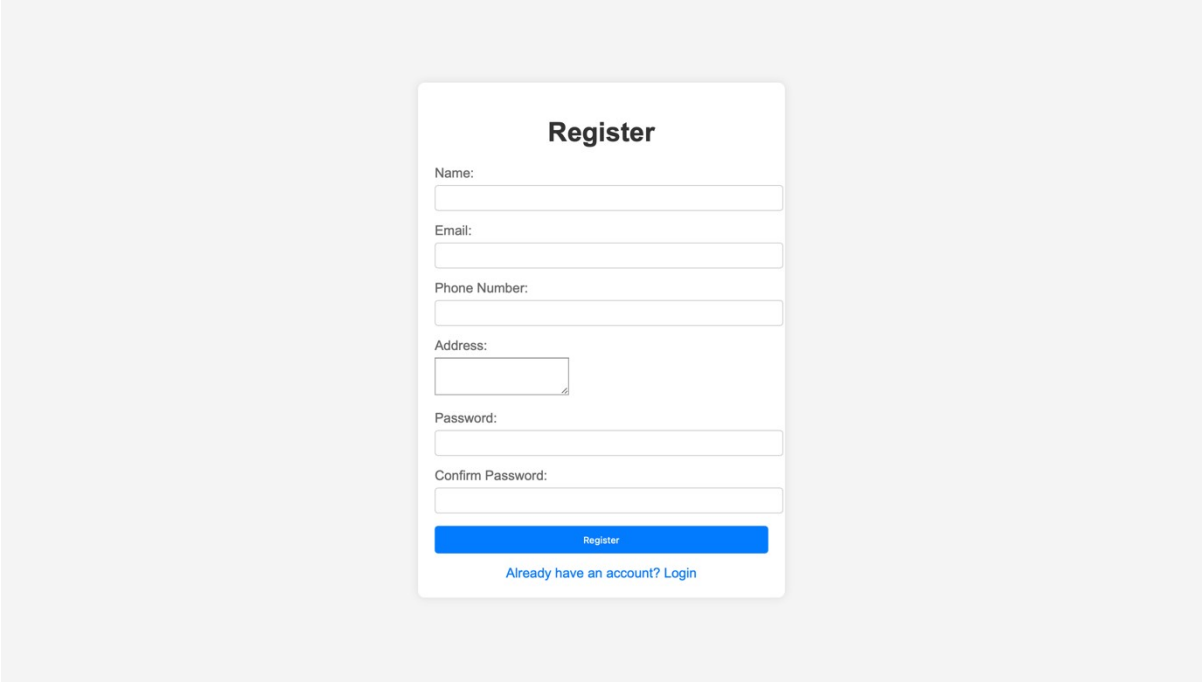

Figure 8: Registration Page

c. Home page

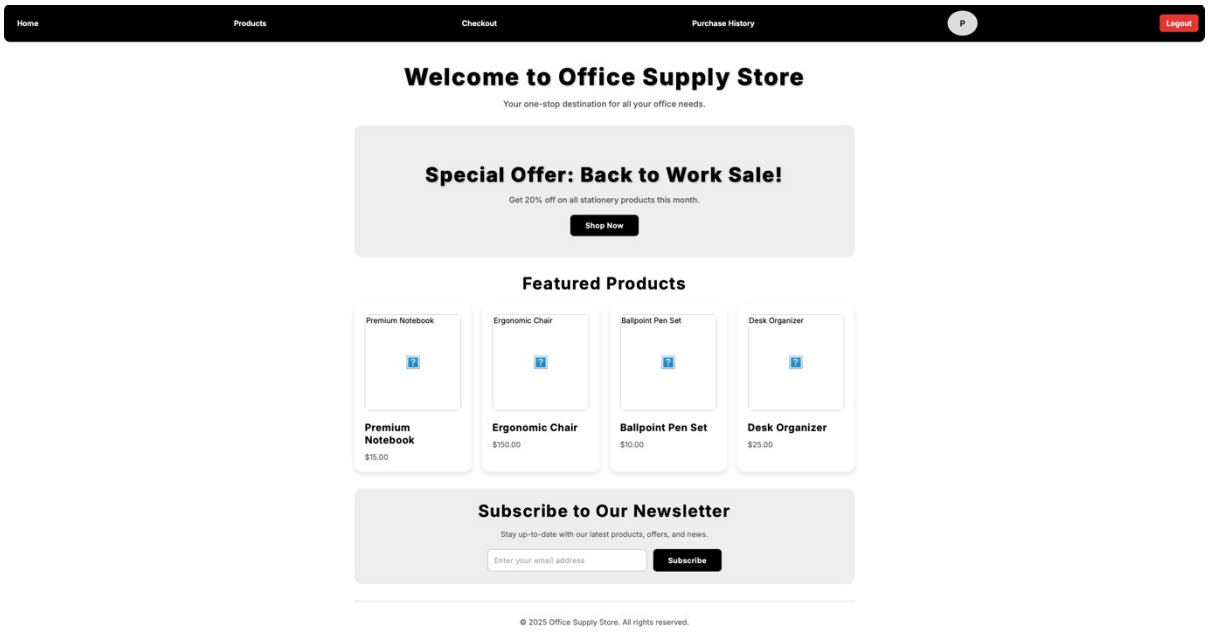

Figure 9: Home Page

d. Our products page (User)

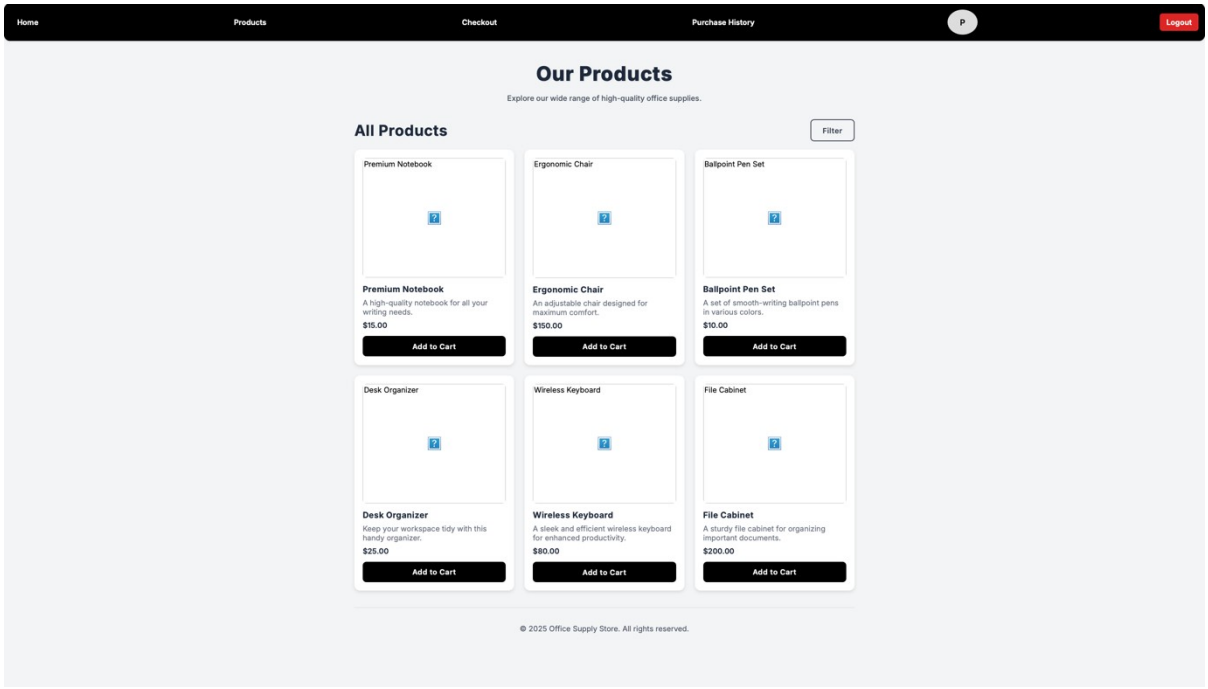

Figure 10: Our Product Page (User)

e. Checkout page

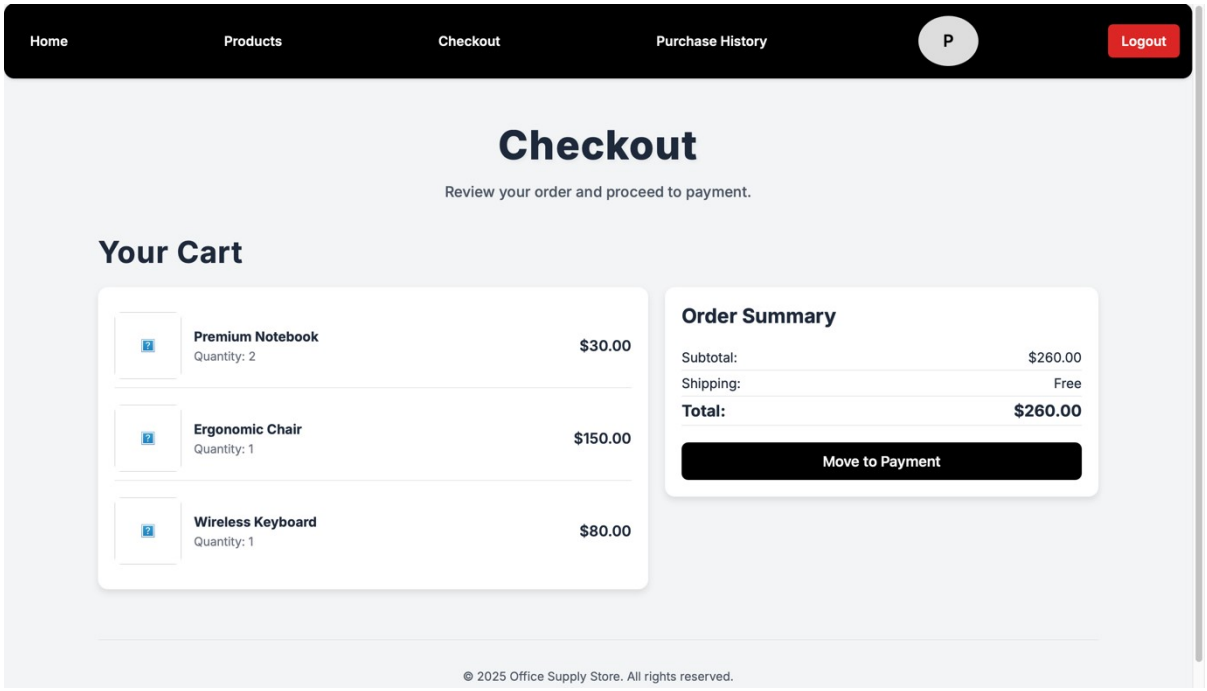

Figure 11: Checkout Page

f. Payment page

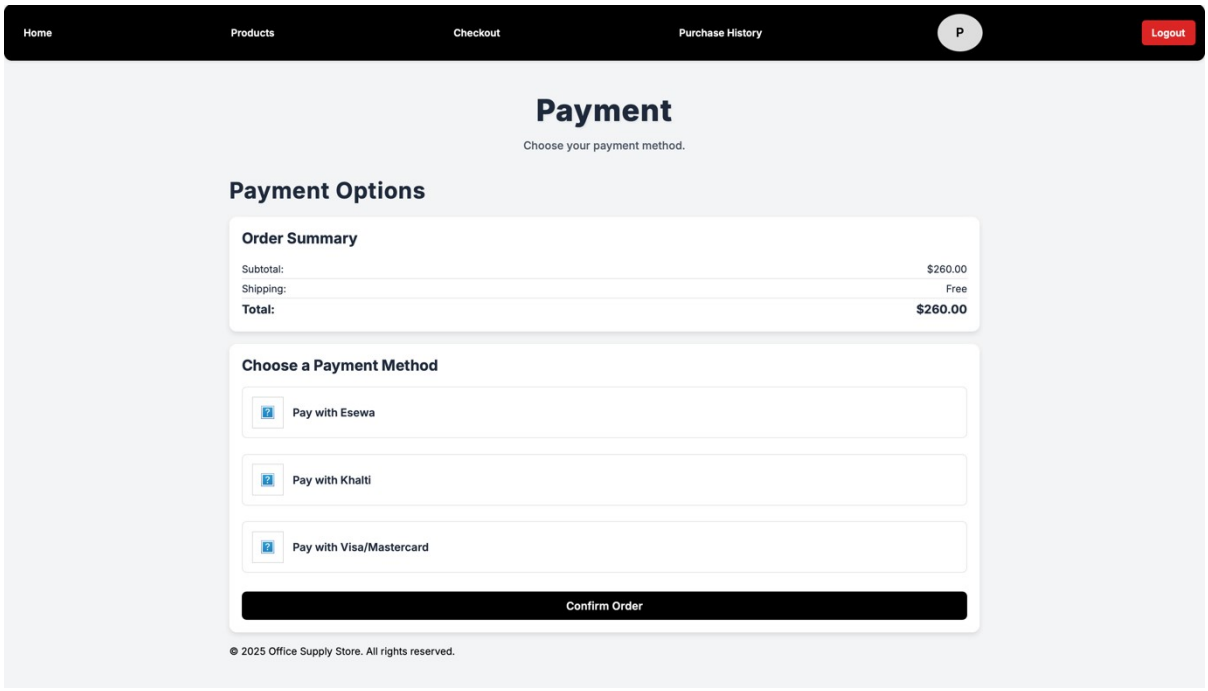

Figure 12: Payment Page

g. Purchase History Page

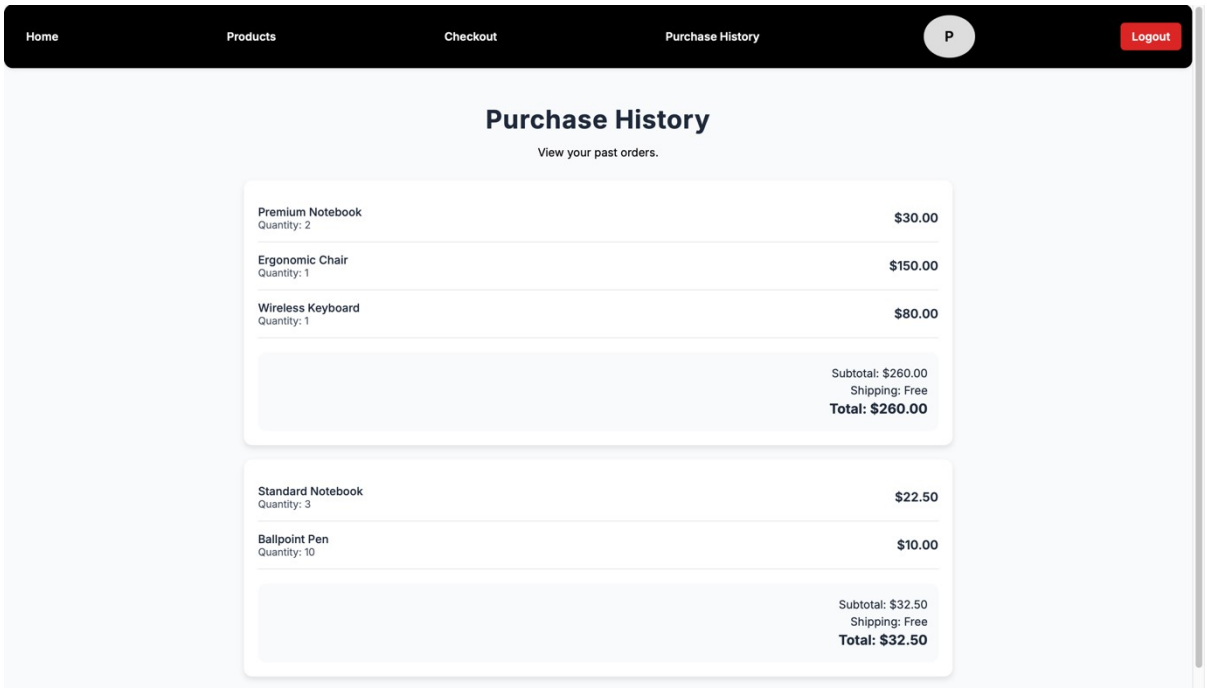

Figure 13: Purchase History Page

h. User Profile Page

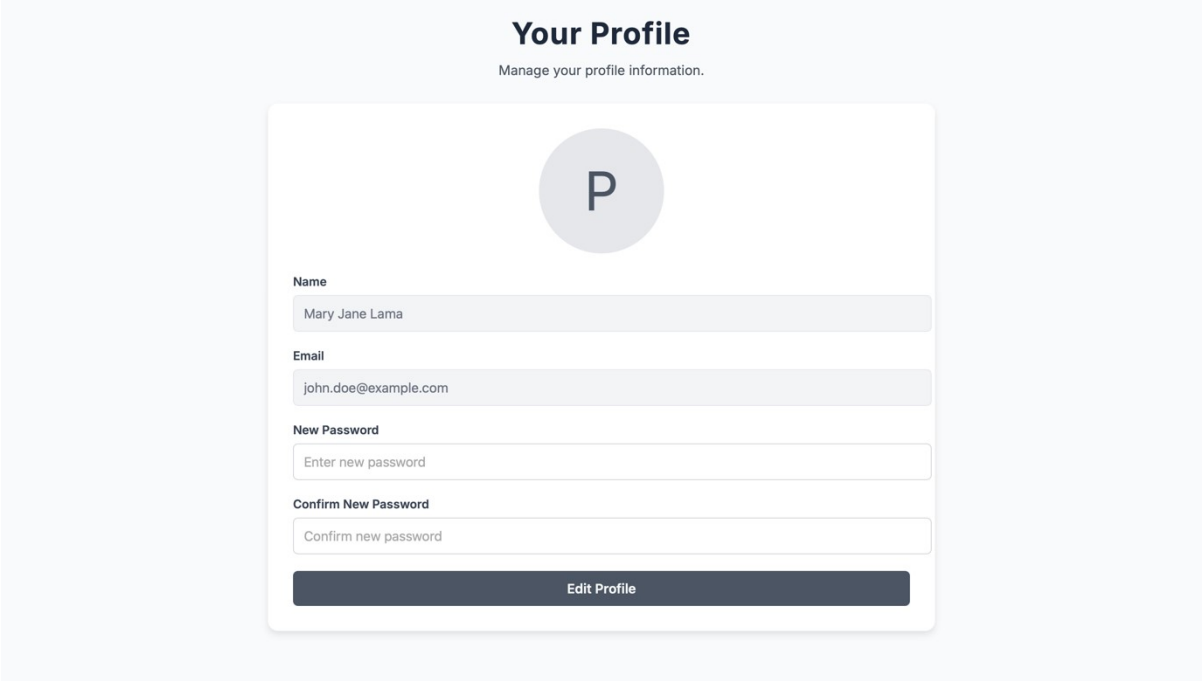

Figure 14: User Profile Page

i. Admin Dashboard page

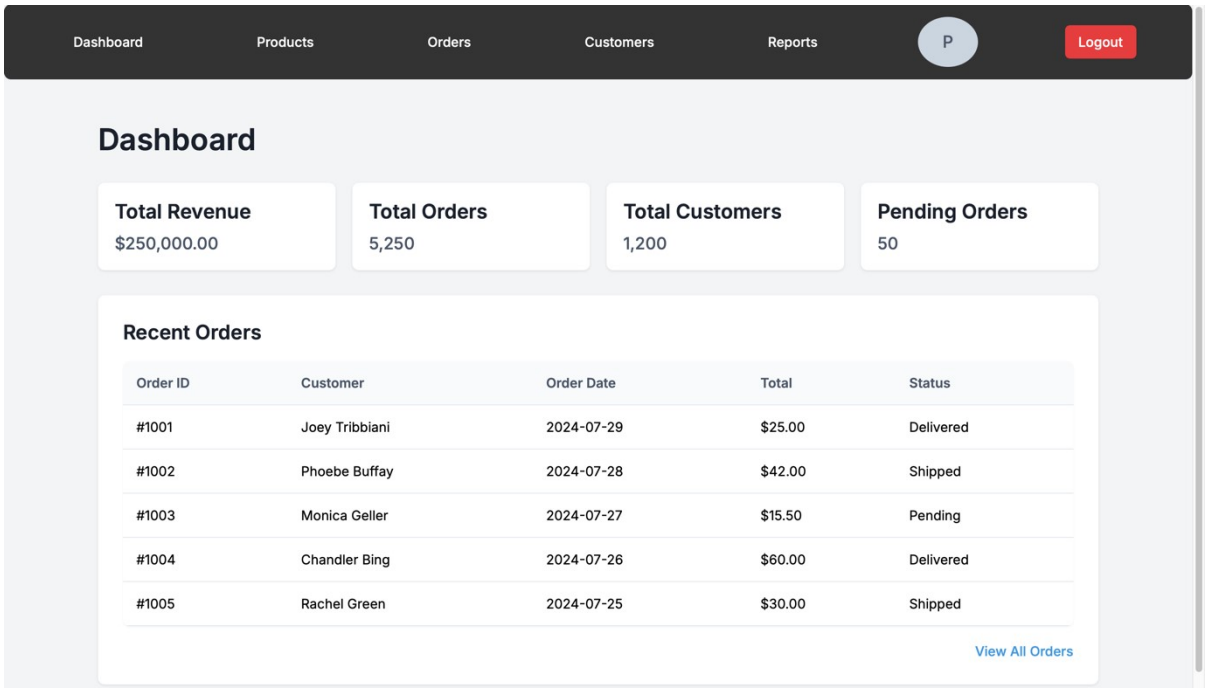

Figure 15: Admin Dashboard Page

j. Products page (Admin)

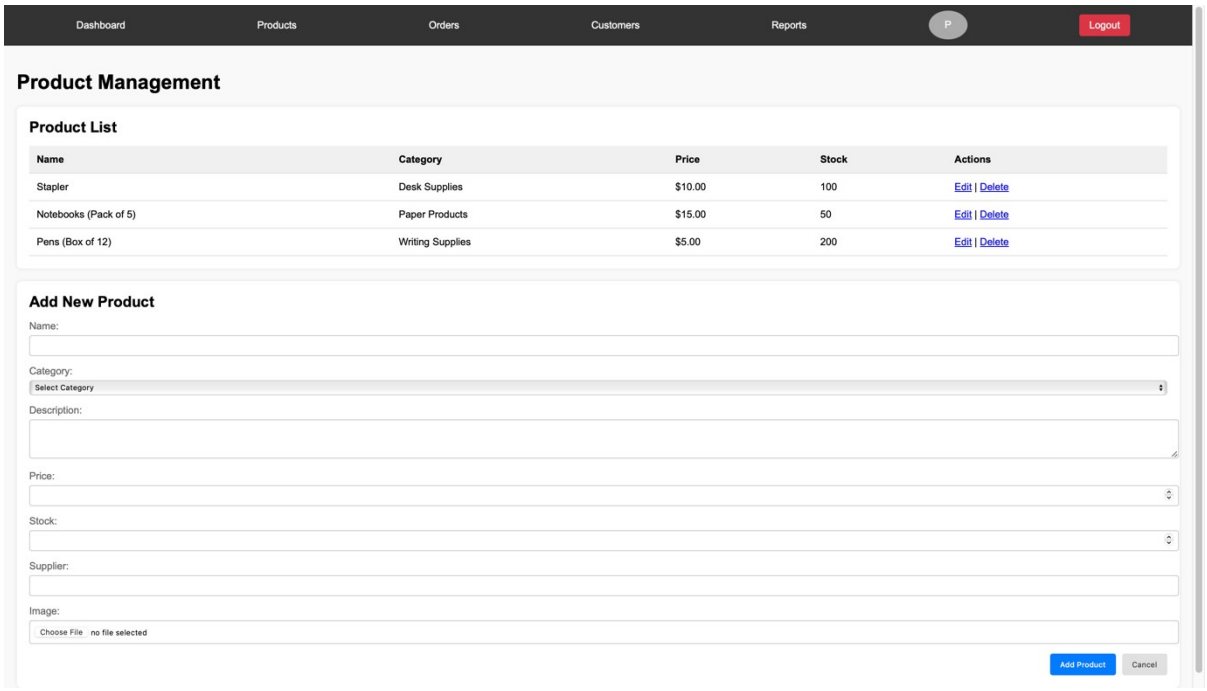

Figure 16: Products page (Admin)

k. Orders Page

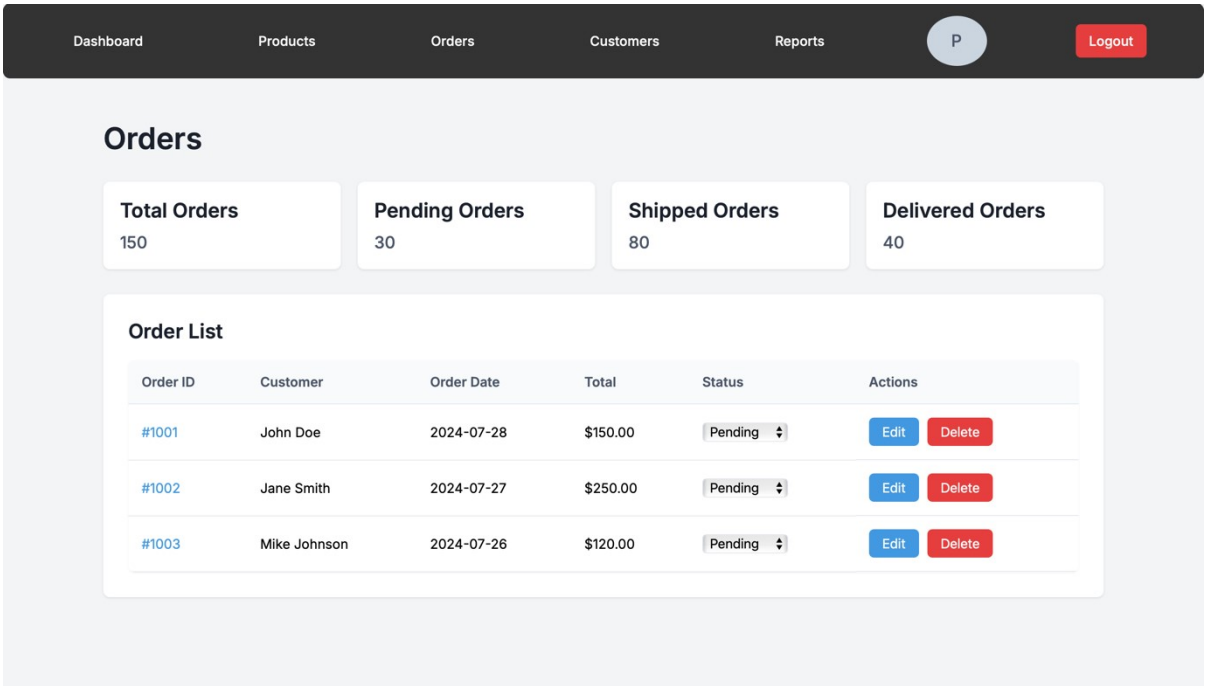

Figure 17: Orders Page

I. Customers page

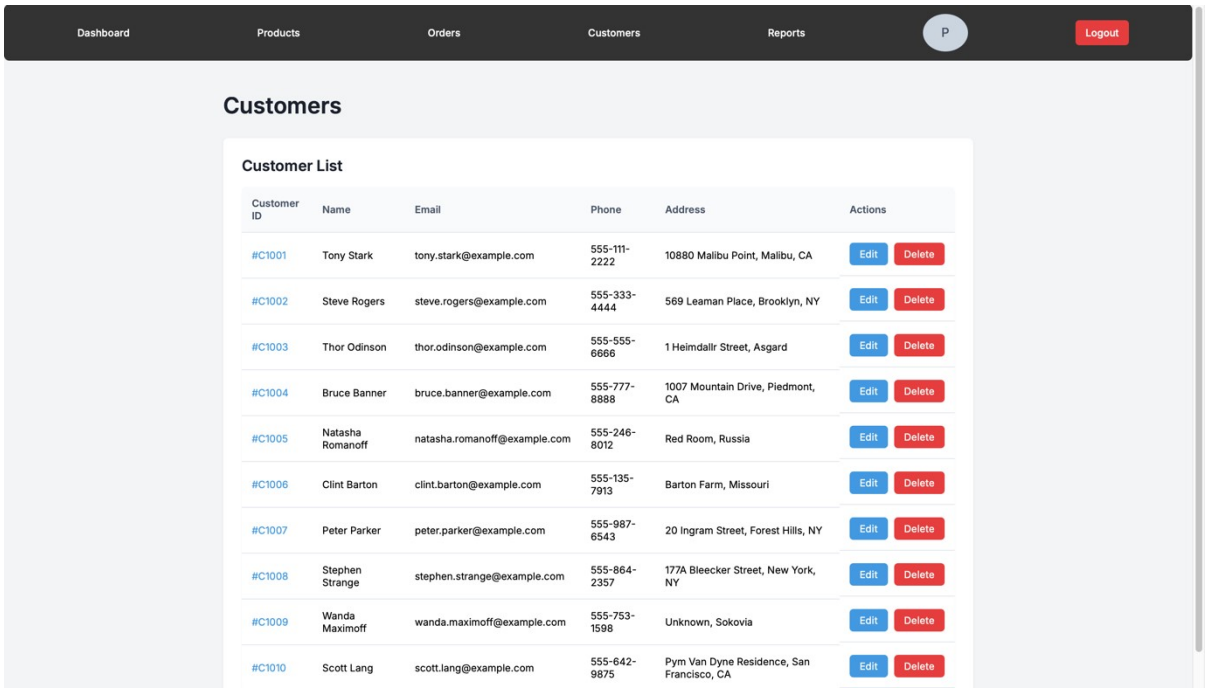

Figure 18: Customers Page

m. Reports Page

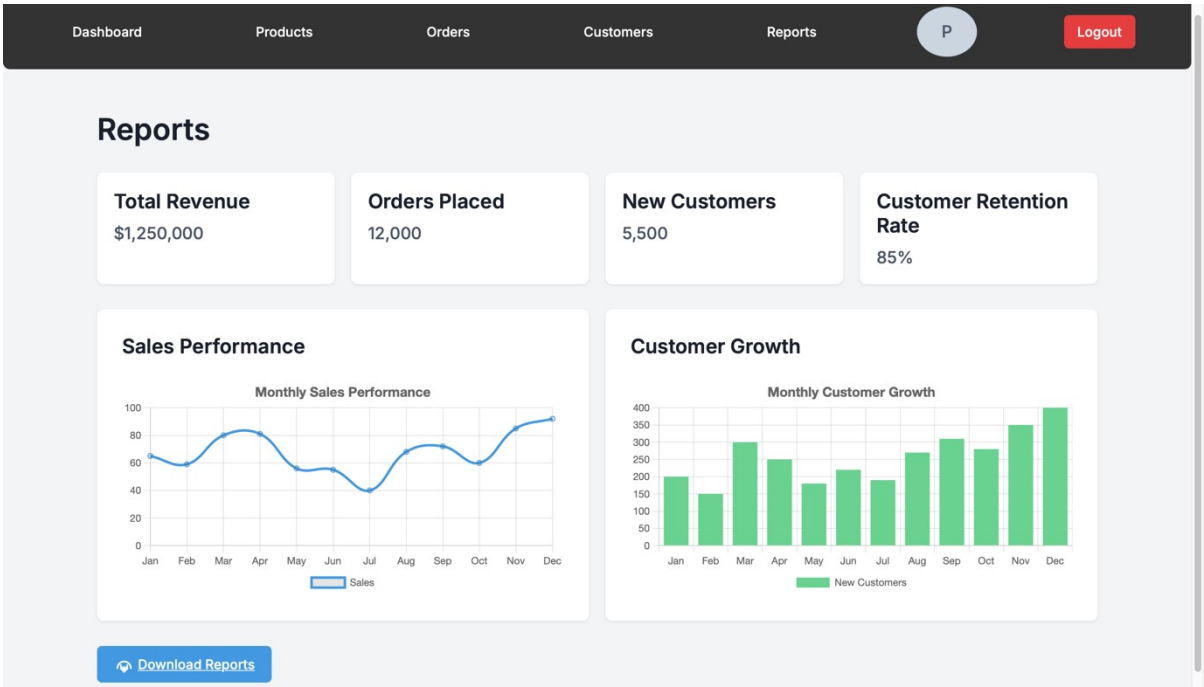

Figure 19: Reports Page

n. Admin Profile page

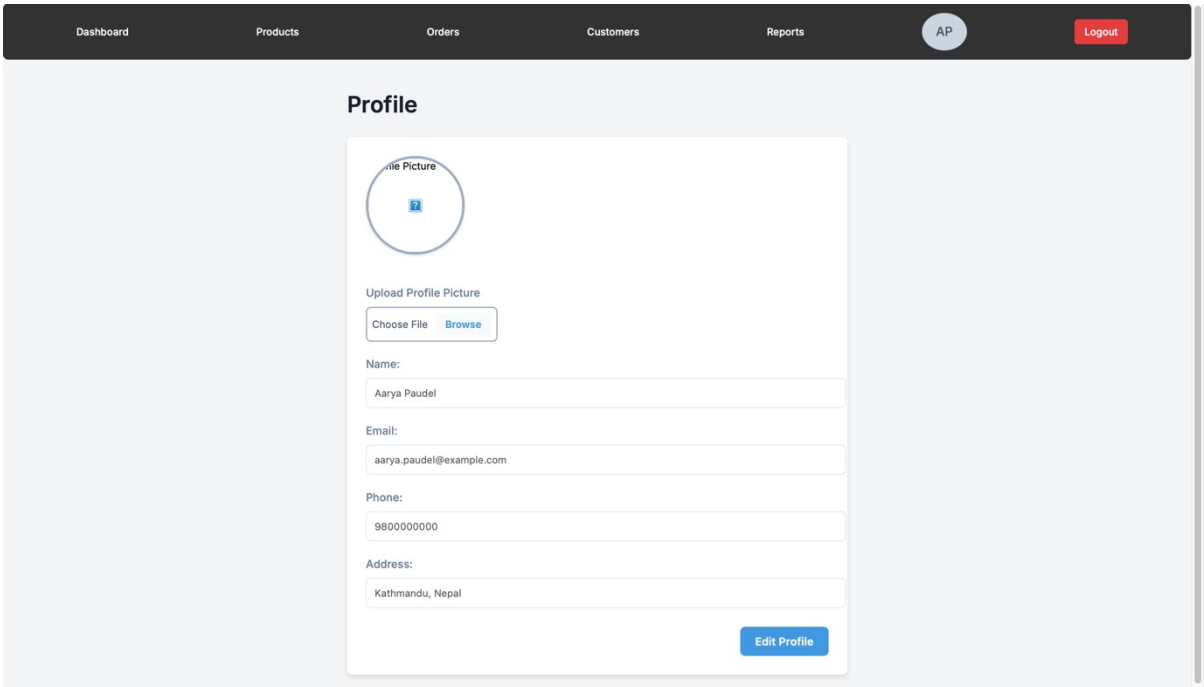

Figure 20: Admin Profile Page

o. Real-Time Stock Update Page

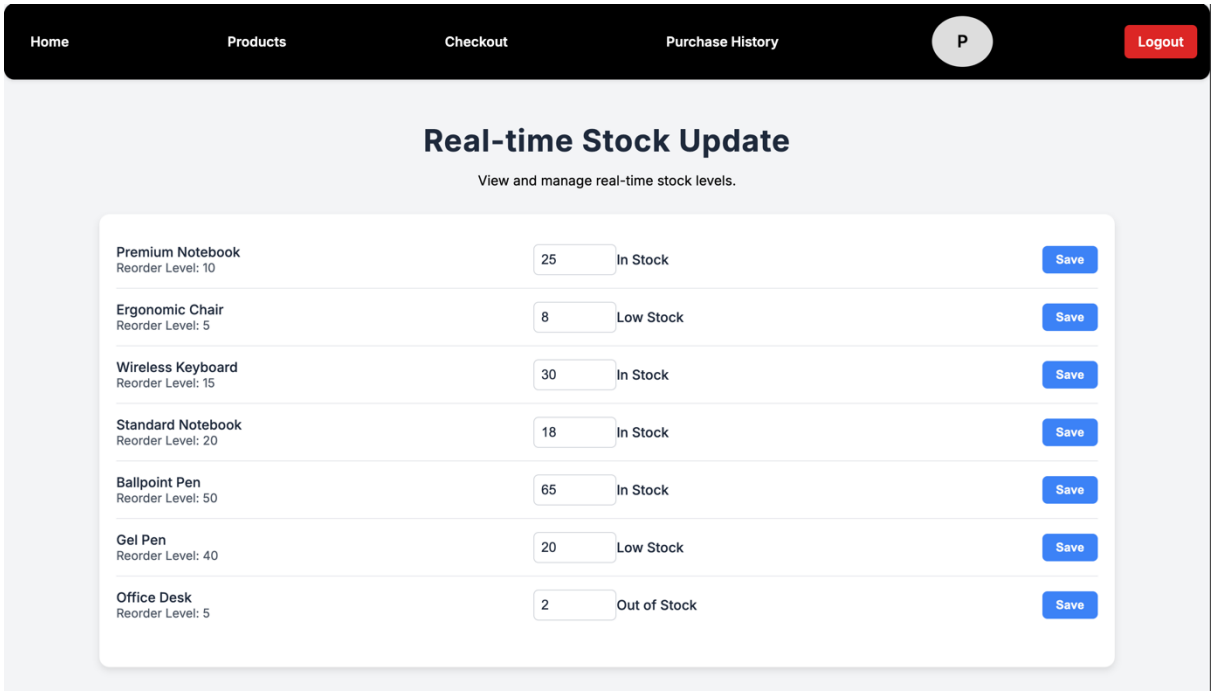

Figure 21: Real-Time Stock Update

p. Purchase Order Page

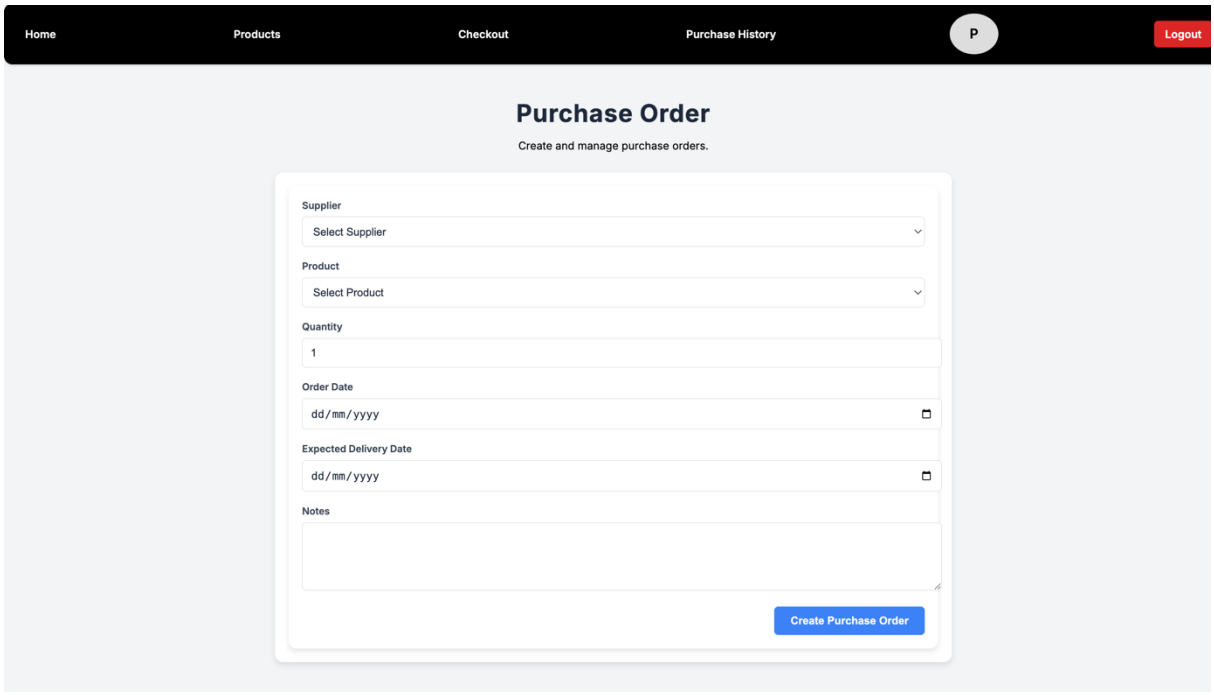

Figure 22: Purchase Order Page

Conclusion

The Milestone 3 report deals with the inventory management system, furnishing all the important design elements that were required. It contains the Class Diagram in detail, including the central components and their relationships within the system structure. This will be crucial in the next development phases. It also elaborates on the plans for proceeding with development works, like architectural options, design patterns, and an iterative phased development strategy following the Waterfall model—thorough testing consideration for the continuous success of the developed system and maintenance planning.

The next step is to develop the UI prototypes, which will help visualize all the functions of the application and the interactions required from users. These prototypes will play a crucial role in refining the design of the system and in making sure that the final implementation is in compliance with the requirements.