

Group Task – 01 Assessment Report

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Only for Course Teacher** | | | | | | |
|  | | **Needs Improvement** | **Developing** | **Sufficient** | **Above Average** | **Total Mark** |
| **Allocate mark & Percentage** | | **25%** | **50%** | **75%** | **100%** | **10** |
| **Functionality** | **04** |  |  |  |  |  |
| **Code Quality** | **03** |  |  |  |  |  |
| **Collaboration & Presentation** | **03** |  |  |  |  |  |
| **Total obtained mark** | | | | | |  |
| **Comments** |  | | | | | |

**Semester:Summer 2025 Group No: 2**

**Name: Md. Najmus Sakib ID: 242-35-367**

**Name: Mostakin Ahmed ID: 242-35-831**

**Name: Md. Shadril Shafat ID: 242-35-849**

**Batch:** 43  **Section: J1 Course Code:** SE 133

**Course Name:** Software Development Capstone Project

**Course Teacher Name:** Md. Sakib Ali Mazumder **Designation:** Lecturer

**Submission Date:16/7/2025**

Stock And Inventory Management System



**ABSTRACT**

This project proposes the development of a console-based Stock and Inventory Management System (SIMS) using the C programming language. It is a one-stop solution for stock and inventory management, with a wide range of features designed to streamline management tasks and improve user experience. The system sets the best standard for stock and inventory management with proper efficiency and user interaction.

Admins using the software enjoy a wide range of features, including stock, sales, supplier, and user management. Admins will be able to access another important feature for managing a business or small organization, which is account management. Admins can manage accounts and their calculations easily with the help of our account section in our software. Admin can easily add and delete the products, and they can also update the stock in the inventory.

The software will also have a unique section for handling all the admin and user data. In this section, the admin will be able to handle other admins' user data and their account details. Admin will also be able to see all the user data and account details, like what the particular user has bought from the store and in how many quantities. He will also be able to manage all the users’ account data and analyse the demand for a particular item in the shop.

Clients will have a unique feature in your software, which is that the client will have the opportunity to buy his or her required product by seeing our product listing on our homepage. In this use case, the control flow will go under the strict supervision of the admin. The client can’t complete the process without the consent of the admin.

The application will use file handling in C to store and retrieve data, simulating a basic database system. Key features will include adding new items, updating item details, deleting items, searching for products, and displaying the current inventory status. The system will include menu-driven navigation and error-handling mechanisms to ensure ease of use and stability.

In conclusion, the Stock and Inventory Management System (SIMS) developed in C offers an efficient and user-friendly solution for managing stock, sales, suppliers, users, and accounts. With features like product updates, account tracking, and secure client transactions, it simplifies inventory operations while maintaining data accuracy through file handling.

Designed for small businesses and organizations, SIMS provides a solid foundation for inventory control with potential for future enhancements such as GUI support and analytics. This project showcases the practical application of C programming in solving real-world management challenges.

**Table of Contents**

**Chapter 1**

Introduction ……………………………….…………………………………………………………. 05

1.1 About the System………………………………………………………………...………………. 05

1.2 Purpose …………………….…………………………………………………………..………... 05

1.3 Scope …………………………………………………………………………………..………... 05

1.4 Definitions, Acronyms, and Abbreviations….….…………………………………………... 05

**Chapter 2**

Overall Description ……….………………………………….…………………………………..… 06

2.1 Product Functions or Features……………….…….…………………….…………………..… 06

2.2 Operating Environment ………………….………….………………….………….………..…. 07

**Chapter 3**

Requirement Analysis……….………………………………………………….…………………...… 08

3.1 Functional Requirements ……………….…….………………………….…………………..….08

3.2 Non Functional Requirements………………….….………….……………………….……...….10

3.3 Other Requirements…………………… ………...……………….…………….……………...….12

3.4 Stakeholder List………………………….………….………….…………………………………..12

3.5 Prioritizing…………………………………………………………………………..…….13

**Chapter 4**

Design……………....…………………….……………………………….. …………………….….…15

4.1 Use Case Diagram… ………………….……………………………..…………………….….....15

4.2 Use Case Description………………….…………………………….. …………………….….....18

4.3 Activity Diagram………………….……………………………………..…………………..……34

4.4 Sequence Diagram………………….………………………………………………………..…...42

**Chapter 5**

Implementation ……………....…………………….………………………………………….….……51

5.1 Methodology……………………………………………………………………………………..... 51

5.2 C Concepts we used for our Project ………………….….………………………………….....…..54

5.3 Gantt Chart…………………………………………………………………………………………55

**Chapter 6**

Conclusion …….……….………………….………………………………………..………………… 55

**Chapter 1: Introduction**

**1.1 About the system:**

We are very glad and pleased to announce the project stock and inventory management system. This project will help small to medium-sized businesses manage their stocks and inventory easily and efficiently. With the help of its features like product management, the admins will be able to add, delete, and manage products and stocks easily. With the help of a fast and interactive menu the work will be very much organized.

**1.2 Purpose:**

A Stock and Inventory Management System is designed to efficiently track and manage product information, including quantities, item codes, and user information. It helps maintain optimal stock levels by monitoring inflow and outflow, preventing problems like overstocking and shortages. The system organizes data for easy access with the help of SKU, allowing quick updates, searches, and deletions. It also records sales and purchases, automatically adjusting inventory levels. Additionally, it can check reports on current stock status and sales. The systems also include user management features to control access and permissions for buying any products directly from the system.

**1.3 Scope:**

Stock and inventory management system features sets will be:

* Sales management.
* Order management.
* Accounts.
* Date tracking.
* Product and stock management.
* Supplier management.
* User management.
* Client purchasing opportunity from the home page.
* Authorization and Authentication.

**1.4 Definitions, Acronyms, and Abbreviations:**

**OneMart:** The name of the software stock and inventory management system.

**Admin:** The person in charge of handling the software and is the main user of the software.

**Client:** A Potential customer buying from the admin or the software home page.  
**SKU:** SKU stands for “stock keeping unit,” and it is a unique alphanumeric code assigned to a product for internal inventory management.

**Chapter 2: Overall Description**

**2.1 Product Functions or Features:**

* **Authorization and Authentication:** One of the most important and key features of a software is its security system. Without any compromise, this software also includes authorization and authentication. The software will have a different interface for the admin and the user for robust security for the software. And all the necessary and important information will be protected in a different file to manage it properly.
* **Sales management:** The software has a dedicated section for managing the sales of the business or shop. In this section, there will be functionality like sales history, total income generated in a particular month, and lastly, total monthly income.
* **Order management:** The software has a functionality where the admin can approve or disapprove a particular order that has been placed by a client.
* **Date tracking:** The Date Tracking feature allows users to efficiently manage and review time-sensitive data within the system. It includes filtering sales reports by day and month to analyse performance over time. The feature also supports expiry tracking to monitor perishable stock and prevent losses due to expired items. Additionally, it enables sorting history so that the most recent transactions appear at the top, ensuring quick access to the latest activity for better decision-making and operational clarity.
* **Accounts:** The account section of the software has some unique features, including the Income Tracking feature provides real-time insights into the business’s financial performance by calculating daily, monthly, and total sales income. This helps admins monitor revenue trends, make informed decisions, and evaluate overall profitability efficiently.
* **Product and stock management:** The product and stock management section of the system will include features like add, delete, search, and display products. It will also show the full inventory of the store or business. Other features include stock operations like stock updating and checking.
* **Supplier management:** The supplier management feature of the software will help the admin to access features like the supplier list, the particular product search of a supplier, and create a supplier. It will also include features like a product list of an individual supplier and a supplier list that supplies products to the shop or business
* **User management:** The user management is a special feature of our software, in which functionality like the creation of new admins and users is included. Other features, like admin and user account deletion and change of ownership of the admin account, can be done. Admin and user password changes can be easily and efficiently performed in this functionality.
* **Client purchasing opportunity from the Online Store page:** A unique and special feature of the software is that the client can directly purchase a product from the home page of the software with the consent of the admin of the software.

**2.2 Operating Environment:**

As this project is solely built on the C programming language. It will be a console or terminal-based software. And for this reason, the operating environment of a program built in C refers to the underlying operating system (such as Windows, Linux, or macOS), a compatible C compiler (like GCC, Turbo C, Intel C++, or Clang), and the necessary hardware resources such as CPU, memory, and storage. The environment also includes runtime libraries that support standard input/output operations, and often a command-line interface or terminal for compiling and executing the code. For file-based applications like this, stock and inventory management systems, proper file access permissions, and the support of a proper file system are also crucial.

**Chapter 3: Requirement Analysis:**

**3.1 Functional Requirements:**

* **Authorization and Authentication**

**FR-01:** Sign Up

Users can create an account by providing a username, password, and required details. Data is securely stored for future login.

**FR-02:** Sign In

Users can log in with valid credentials. Access is granted based on user role (admin or client).

**FR-03:** Reset Password by User

Users can reset their password after verifying their identity through a security check.

* **Sales Managements**

**FR-04:** New Sale

Admins can record a new sale by selecting products and entering quantity and client details. Stock updates automatically.

**FR-05:** Sales Report

The system maintains a log of all past sales, viewable by date, product, or client.

**FR-06:** Search Sold Product

Admins can search sold products by name, date, or client to quickly retrieve sales records.

* **Stock/Product Management**

**FR-07:** Add New Product

Admins can add new products by entering details such as name, category, price, and quantity.

**FR-08:** Delete Product

Admins can remove products from the system when no longer available or discontinued.

**FR-09:** Return Product

Returned products can be recorded, and stock is updated accordingly.

**FR-10:** Update Stock

Admins can modify the quantity or details of existing products in the inventory.

**FR-11:** Stock List by Category

The system displays products grouped by category for easier navigation and analysis.

**FR-12:** Stock Check

Admins can view current stock levels of all products to monitor availability.

**FR-13:** Expired Product

The system automatically identifies and lists all products that have passed their expiration date and flags them for removal or disposal.

* **Accounts**

**FR-14:** Daily Income

The system calculates and displays the total income generated from sales each day.

**FR-15:** Monthly Income

The system summarizes and shows income earned from sales for each month.

**FR-16:** Total Income

The system provides a cumulative total of all income generated over time.

* **Supplier Management**

**FR-17:** Add Supplier

Admins can add supplier details, including name, contact info, and supplied products.

**FR-18:** Delete Supplier

Admins can remove supplier records when no longer associated with the system.

**FR-19:** Product Search

Users can search for products by name, category, or supplier to quickly access details.

**FR-20:** List of Products by Supplier

The system displays all products supplied by a selected supplier.

* **User Management**
* For Admin/Customer

**FR-21:** Add User

Admins can create new user accounts with assigned roles and credentials.

**FR-22:** Delete User

Admins can remove user accounts from the system as needed.

**FR-23:** Reset Password by Admin

Admins can reset passwords for users to assist with account recovery.

**FR-24:** List of All Admins or Customers

The system displays a complete list of all admin and customer users.

* **Online Store Management - Admin Panel**

**FR-25:** Order Pending List

The system shows all orders that are placed but not yet delivered.

**FR-26:** Order Delivered List

The system maintains a record of all completed and delivered orders.

* **Online Store Dashboard - For Customer Panel**

**FR-27:** All Product List

The system displays a complete list of all available products, including names, categories, prices, and stock status, allowing users to browse or search through the inventory.

**FR-28:** Checkout

Clients can review their selected products and proceed to finalize the order.

**FR-29:** Payment

The system processes payments and updates the order status upon a successful transaction.

**FR-30:** Customer Profile

The system allows users to view and manage their personal information, such as name, contact details, and order history. Users can update their profiles as needed.

**3.2 Non-Functional Requirements:**

* **Performance:**

1. The system should respond to user actions (like searching or updating inventory) within 1 second.
2. It should handle up to 1,000 product entries without significant performance degradation.
3. Its stability should be up to the mark so that it can handle multiple maintenance and runtime changes without significant disruptions at a time.

* **Availability:**

1. The system should be available and operational whenever the business is open, ensuring uninterrupted access for stock updates, sales, and reporting.
2. Users should be able to access stored inventory data at any time without delays or errors, even after system restarts or minor failures, ensuring continuous workflow and decision-making.

* **Reliability:**

1. The system must run consistently without crashing or data corruption.
2. All operations (add, update, delete) should maintain data integrity, even after unexpected interruptions.
3. Data integrity is one of the crucial non-functional requirements for this type of software, as the whole system is about data handling.

* **Usability:**

1. The interface should be simple, text-based, and easy to navigate for non-technical admins and users.
2. Clear prompts and error messages should guide users through the system.

* **Maintainability:**

1. The code should be modular and well-commented to support future updates and debugging.
2. Configuration (e.g., file paths and file limits) should be easy to change.

* **Scalability:**

1. The system should allow easy addition of more products, users, or features without major changes to the codebase.

* **Security:**

1. Basic input validation should prevent illegal or harmful inputs.
2. Mainly, access to inventory updates can be restricted to Admins only; users will not be able to access this part of the software. For this, password-protected admin access is very important.

* **Portability:**

1. The system should run on any OS that supports a standard C compiler (like GCC or Turbo C).
2. Data files should be stored in a format compatible across platforms (e.g., .txt).

**3.3 Other Requirements:**

* **Development Tools:**

1. The system must be developed using a standard C compiler like GCC, Turbo C, or Code::Blocks.
2. A basic text editor or IDE (such as Code::Blocks or VS Code) will be used for writing and debugging the code.

* **Documentation:**

1. The project must include user documentation explaining how to run the program, input data, and navigate features.
2. Code should be well-commented for ease of understanding and future modification.

* **Legal and Compliance:**

1. If used in a real business, the system should comply with local data handling and storage regulations, even in basic form.

**3.4 Stakeholder List:**

**Admin/Inventory Manager:**

Admins are responsible for managing the entire system, including product inventory, sales records, user accounts, and supplier details. They have full control over system settings and oversee business operations to ensure smooth workflow.

**Customer:**

Clients interact with the system primarily to browse products, place orders, and make payments. Their activities are monitored and controlled by admins to maintain order and security within the system.

**3.5 Prioritizing:**

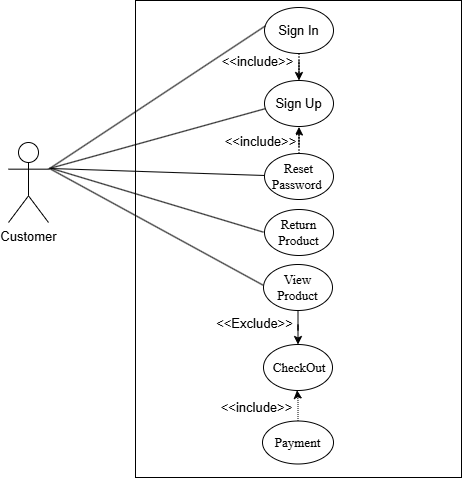
Using Numerical assignment prioritization technique to prioritize our project functionality.

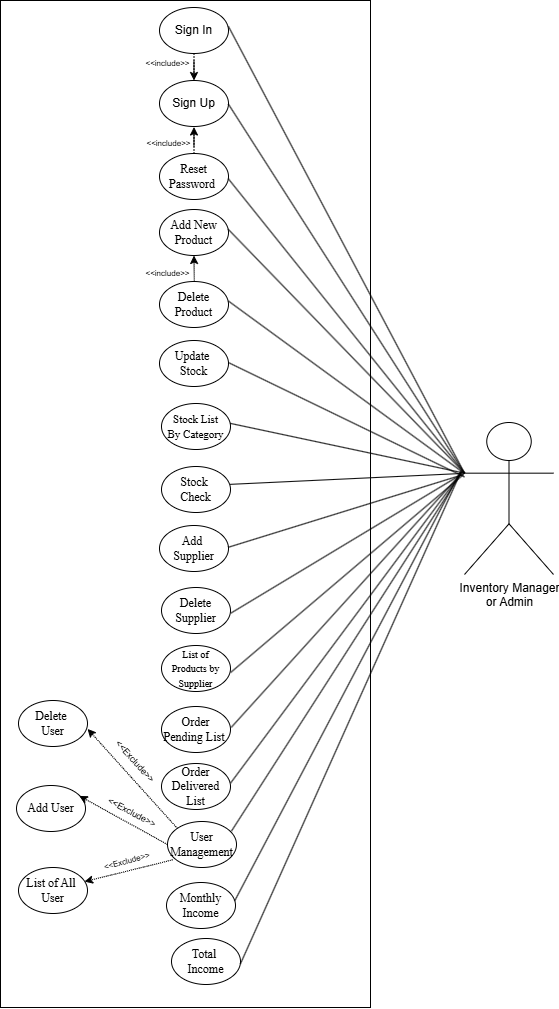
Priority groups: **1 – Critical 2 – Moderate 3 – Optional**

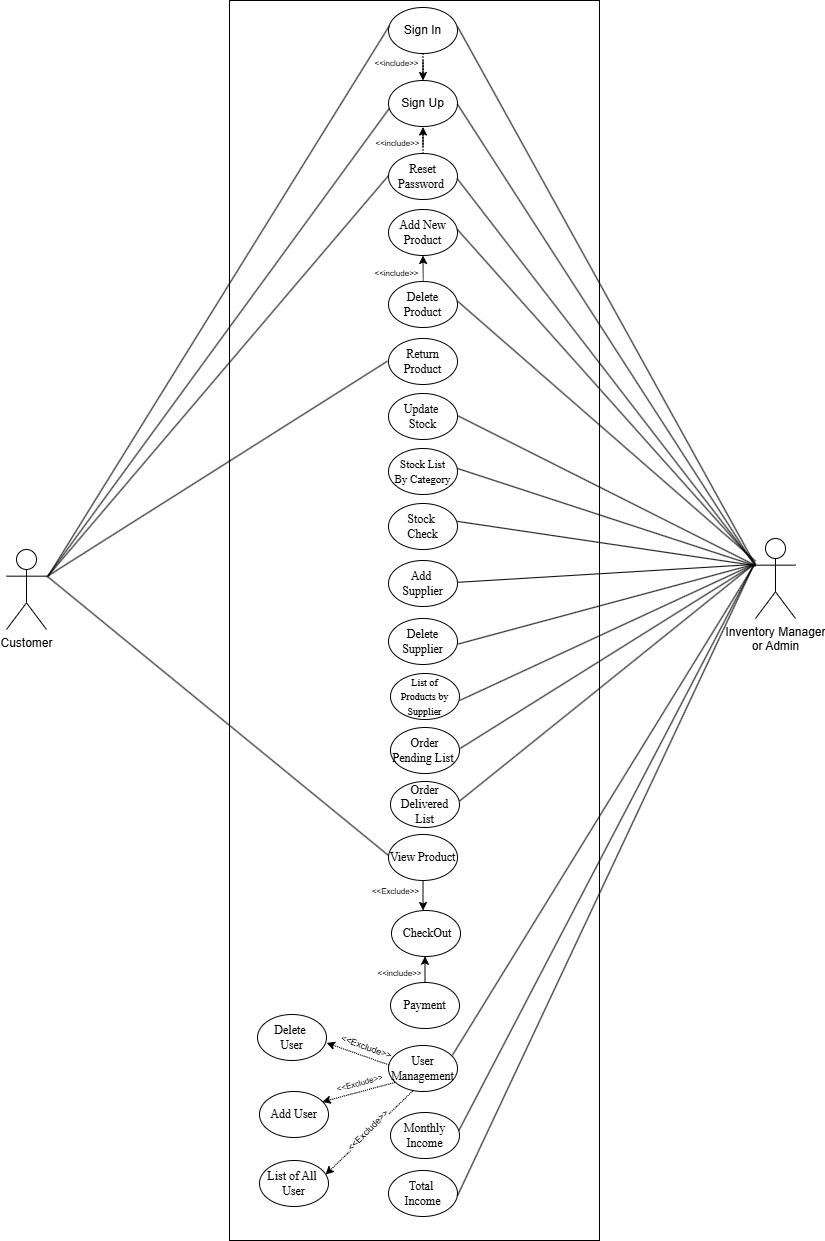
|  |  |
| --- | --- |
| **App Features** | **Priority Group** |
| Sign Up | **1** |
| Sign In | **1** |
| Reset Password by User | 2 |
| New Sale | **1** |
| Sales Report | **1** |
| Search Sold Product | 2 |
| Add New Product | **1** |
| Delete Product | 2 |
| Return Product | 2 |
| Update Stock | **1** |
| Stock List by Category | 2 |
| Stock Check | **1** |
| Expired Product | 2 |
| Daily Income | **1** |
| Monthly Income | **1** |
| Total Income | **1** |
| Add Supplier | 2 |
| Delete Supplier | 3 |
| Product Search | 2 |
| List of Products by Supplier | 2 |
| Add User | **1** |
| Delete User | **1** |
| Reset Password by Admin | 2 |
| List of All Admins or Customers | 2 |
| Order Pending List | **1** |
| Order Delivered List | 2 |
| All Product List | 3 |
| CheckOut | **1** |
| Payment | **1** |
| Customer Profile | 2 |

**Chapter 4: Design**

4.1 Use Case Diagram

* **For Customer**
* **For Admin/Manager**

****

* **For Customer And Manager**

**4.2 Use Case Description**

**1. Sign Up**

|  |  |
| --- | --- |
| **Use case** | **Sign Up** |
| Goal | Allow new users (Admins or Customers) to create an account in the system for future access. |
| Preconditions | User is not already registered, and the system is online. |
| Success End Condition | A new user account is successfully created and confirmation is provided. |
| Failed End  Condition | Account creation is blocked due to validation errors. |
| Primary Actor | New user |
| Secondary Actor | Authentication system |
| Trigger | User clicks on the "Sign Up" option |
| Success Scenario | * User accesses the Sign-Up page from the login interface. * User fills in all required fields, including name, email, password, and selects a role (Admin/Customer). * System checks for input validity and ensures no duplicate email exists.   If valid, the system creates the user record.  Confirmation message or redirection to login is provided. |
| **Alternative Flows** | 2a: Required fields are left empty → System prompts user to complete all fields.  3a: Email is already associated with an existing account → System displays an error message. |

**2. Sign In**

|  |  |
| --- | --- |
| **Use case** | Sign in |
| Goal | Allow existing users to securely log into the system using valid credentials. |
| Preconditions | The user has already signed up and remembers their credentials. |
| Success End Condition | The user is granted access and redirected to their respective dashboard. |
| Failed End  Condition | User is denied access due to invalid credentials or system error. |
| Primary Actor | Registered User. |
| Secondary Actor | Authentication System. |
| Trigger | User clicks on the "Sign In" button after entering credentials. |
| Success Scenario | 1. The user navigates to the sign-in page. 2. Enter email and password. 3. The system validates credentials. 4. If valid, the system starts a user session and redirects to the dashboard. |
| Alternative Flows | 3a: Email or password is incorrect → System shows an error and asks the user to retry.  3b: Account locked due to multiple failed attempts → Inform user and suggest password reset. | |

**3. Reset Password by User**

|  |  |
| --- | --- |
| **Use case** | Reset Password by User |
| Goal | Allow users to recover account access by resetting their password. |
| Preconditions | The user has previously registered an email. |
| Success End Condition | A new password is successfully set and confirmed by the user. |
| Failed End Condition | Reset fails due to unregistered email or an expired reset link. |
| Primary Actor | User |
| Secondary Actor |  |
| Trigger | User clicks on the "Forgot Password" link on the login |
| Success Scenario | 1. User clicks the "Forgot Password" option. 2. The user enters their registered email address. 3. The system verifies the email and sends a reset link. 4. User clicks the link and is redirected to a password reset page. 5. User inputs and confirms the new password. 6. System updates password and notifies user of success. |
| Alternative Flows | 2a: Email not recognized by system → Error message.  4a: Reset link expired → System prompts for a new reset request. |

**4. New Sale**

|  |  |
| --- | --- |
| **Use case** | New Sale |
| Goal | Enable recording of a new product sale and automatically update stock. |
| Preconditions | The product is available in stock, and the user is authorized. |
| Success End Condition | The sale is successfully recorded, stock updated, and a receipt is issued. |
| Failed End  Condition | Sale fails due to a system error or stock unavailability. |
| Primary Actor | Admin |
| Secondary Actor | Inventory System |
| Trigger | Customer requests to purchase one or more products. |
| Success Scenario | 1. Staff opens the "New Sale" screen. 2. Selects customer (if needed) and products. 3. Enter the quantity for each item. 4. System checks stock availability. 5. Displays total price including tax/discounts. 6. Staff confirms the sale. 7. The system deducts the sold quantity from the inventory and records the sale. 8. Receipt is generated and shown/printed. |
| Alternative Flows | 4a: Stock not available → System blocks transaction. |

**5. Sales History**

|  |  |
| --- | --- |
| **Use case** | Sales History |
| Goal | Allow users to view historical sales transactions for review and reporting. |
| Preconditions | The system contains previous sales |
| Success End Condition | All relevant sales data is displayed correctly. |
| Failed End Condition | Data fails to load, or no records. |
| Primary Actor | Admin |
| Secondary Actor | Database System |
| Trigger | The user accesses the "Sales History". |
| Success Scenario | 1. User opens Sales History. 2. The system retrieves past transactions. 3. Displays in tabular format with filters for date, product, or customer. |
| Alternative Flows | 2a: No sales data available → Show "No results" message. | |

1. **Search Sold Products**

|  |  |
| --- | --- |
| **Use case** | Search Sold Products |
| Goal | Allow users to find specific product sale information from historical records. |
| Preconditions | The product must have been sold at least once. |
| Success End Condition | Sales records are displayed. |
| Failed End Condition | No matching product found. |
| Primary Actor | Sales Staff / Admin |
| Trigger | User enters product name or ID in the search field. |
| Success Scenario | 1. User inputs product criteria. 2. System scans sales database. 3. Returns matching entries with date, quantity, and amount. |
| Alternative Flows | 2a: Invalid input or product never sold. |

**7. Add New Product**

|  |  |
| --- | --- |
| **Use case** | Add New Product |
| Goal | Allow users to add new products to the inventory database. |
| Preconditions | User is authenticated as Admin or Inventory Manager. |
| Success End Condition | The product is successfully stored in the inventory system. |
| Failed End Condition | Product addition fails due to validation or system. |
| Primary Actor | Inventory Manager /Admin. |
| Secondary Actor | Database System. |
| Trigger | User selects "Add Product". |
| Success Scenario | 1. User navigates to Add Product. 2. Enter product name, category, quantity, price, and supplier. 3. The product is added to the database. |
| Alternative Flows | 2a: Incomplete data → Prompt for missing fields. |

**8. Delete Product**

|  |  |
| --- | --- |
| **Use case** | **Delete Product** |
| Goal | Remove a product from the inventory. |
| Preconditions | The product should exist. |
| Success End Condition | The product is removed successfully. |
| Failed End Condition | Deletion is blocked. |
| Primary Actor | Admin |
| Secondary Actor | Inventory System |
| Trigger | Admin selects a product for deletion. |
| Success Scenario | 1. Admin selects the product. 2. Confirms deletion. 3. The system removes the product from inventory. |
| Alternative Flows | 1a: Product is linked to active orders → Deletion denied. |

**9. Return Product**

|  |  |
| --- | --- |
| **Use case** | **Return Product** |
| Goal | Process a returned product and update stock. |
| Preconditions | The product was previously sold. |
| Success End Condition | Return is recorded and inventory adjusted. |
| Failed End Condition | Return not processed. |
| Primary Actor | User8888 |
| Secondary Actor | Inventory System |
| Trigger | Customer initiates return request. |
| Success Scenario | 1. Staff records return. 2. The system verifies the sale. 3. Update stock and mark as returned. |
| Alternative Flows | 2a: Sale not found → Return rejected. |

**10. Update Stock**

|  |  |
| --- | --- |
| **Use case** | Update Stock |
| Goal | Allow updating of product quantities in inventory |
| Preconditions | The product must already exist. |
| Success End Condition | Stock is updated successfully. |
| Failed End Condition | Updating stock failed. |
| Primary Actor | Inventory manager |
| Secondary Actor | Database System |
| Trigger | The user initiates stock |
| Success Scenario | 1. Select the product to update. 2. Enter new quantity or adjustment. 3. The system validates and updates. |
| Alternative Flows | 2a: Negative stock entered → Prompt correction. |

**11. Stock List by Category**

|  |  |
| --- | --- |
| **Use case** | Stock List by Category |
| Goal | Display product inventory filtered by category. |
| Preconditions | Products must be categorized. |
| Success End Condition | List of filtered products displayed. |
| Failed End Condition | No products match the filter. |
| Primary Actor | Admin |
| Secondary Actor | Inventory System |
| Trigger | The user selects the category filter. |
| Success Scenario | 1. The user selects a category. 2. The system filters and displays products. |
| Alternative Flows | 2a: No products found → Display message. |

**12. Check Stock**

|  |  |
| --- | --- |
| **Use case** | Check Stock |
| Goal | Check the current quantity of a product in inventory. |
| Preconditions | The product should exist. |
| Success End Condition | Quantity is shown. |
| Failed End Condition | Product not found. |
| Primary Actor | Admin |
| Secondary Actor | Inventory System |
| Trigger | Product ID entered. |
| Success Scenario | 1. The user enters the product ID. 2. The system fetches and shows the quantity. |
| Alternative Flows | 2a: ID invalid → Prompt for correction. | |

**13. Daily Income**

|  |  |
| --- | --- |
| **Use case** | Daily Income |
| Goal | To know the total income per day |
| Preconditions | There must be sales. |
| Success End Condition | Accurate income displayed. |
| Failed End Condition | Data unavailable. |
| Primary Actor | Admin |
| Secondary Actor | Database System |
| Trigger | Daily report requested. |
| Success Scenario | 1. The system aggregates today's sales. 2. Displays total earnings. |
| Alternative Flows | 1a: No sales today → Show zero income. |

**14. Monthly Income**

|  |  |
| --- | --- |
| **Use case** | Monthly Income |
| Goal | To know the monthly income. |
| Preconditions | Sales data should be available for the month. |
| Success End Condition | Monthly income shown. |
| Failed End Condition | Data not available. |
| Primary Actor | Admin |
| Secondary Actor | Database System |
| Trigger | Monthly report requested. |
| Success Scenario | 1. The system filters and totals monthly sales. 2. Displays the amount. |
| Alternative Flows | 2a: No data → Empty report. | |

**15. Total Income**

|  |  |
| --- | --- |
| **Use case** | Total Income |
| Goal | To know the total income. |
| Preconditions | Every sales data till date should be available. |
| Success End Condition | Total income shown. |
| Failed End Condition | Data not available. |
| Primary Actor | Admin |
| Secondary Actor | Database System |
| Trigger | Report requested. |
| Success Scenario | 1. System calculates cumulative sales. 2. Shows the total amount. |
| Alternative Flows | 2a: No data/data corrupted → Error! | |

**16. Add Supplier**

|  |  |
| --- | --- |
| **Use case** | Add Supplier |
| Goal | To add new suppliers. |
| Preconditions | The user has admin rights. |
| Success End Condition | Supplier saved in the database. |
| Failed End Condition | Wrong entry or validation error. |
| Primary Actor | Admin |
| Secondary Actor | Database system |
| Trigger | Admin chooses to add a supplier. |
| Success Scenario | 1. Admin fills the supplier form. 2. The system validates and saves info. |
| Alternative Flows | 2a: Required field missing → Prompt to complete. |

**17. Delete Supplier**

|  |  |
| --- | --- |
| **Use case** | Delete Supplier |
| Goal | Remove the supplier record from the system. |
| Preconditions | The supplier must exist. |
| Success End Condition | Removed supplier successfully. |
| Failed End Condition | Failed to remove the supplier & products are still linked. |
| Primary Actor | Admin |
| Secondary Actor | Database System |
| Trigger | The admin chooses to remove the supplier. |
| Success Scenario | 1. Admin selects the supplier. 2. Confirms deletion. 3. The system removes the record |
| Alternative Flows | 2a: Product still linked → Block deletion. |

**18. List of Products by Supplier**

|  |  |
| --- | --- |
| **Use case** | List of Products by Supplier |
| Goal | Display all products provided by a specific supplier. |
| Preconditions | The supplier and products are linked. |
| Success End Condition | List available. |
| Failed End Condition | No linked products. |
| Primary Actor | Admin |
| Secondary Actor | Database System |
| Trigger | Supplier selected. |
| Success Scenario | 1. The user selects a supplier. 2. The system fetches and shows products. |
| Alternative Flows | 2a: No items found → Inform user. |

**19. Add User**

|  |  |
| --- | --- |
| **Use case** | Add User |
| Goal | Create a new user profile. |
| Preconditions | Admin must be logged in. |
| Success End Condition | User added successfully. |
| Failed End Condition | Invalid data, failed to add user. |
| Primary Actor | Admin |
| Secondary Actor | Database System |
| Trigger | Add user-initiated. |
| Success Scenario | 1. Admin enters user info. 2. The system validates and adds a user. |
| Alternative Flows | 2a: Email already used → Show error.  2b: Invalid entry → Show error. |

**20. Delete User**

|  |  |
| --- | --- |
| **Use case** | Delete User |
| Goal | Remove a user. |
| Preconditions | The user must exist. |
| Success End Condition | User removed successfully. |
| Failed End Condition | The user is not available or user linked to active data. |
| Primary Actor | Admin |
| Secondary Actor | Database system |
| Trigger | Delete user-initiated |
| Success Scenario | 1. Admin selects the user. 2. Confirms deletion. 3. The system validates and removes the user. |
| Alternative Flows | 1a: User not found → Warning shown.  2a: Cannot delete by admin → Warning shown. |

**21. Reset Password by Admin**

|  |  |
| --- | --- |
| **Use case** | Reset Password by Admin |
| Goal | Admin resets another user's password. |
| Preconditions | Admin must be authorized, and a valid target user should exist. |
| Success End Condition | Password reset successful. |
| Primary Actor | Admin |
| Secondary Actor | Database System |
| Trigger | Admin initiates reset. |
| Success Scenario | 1. Admin selects user. 2. Inputs new password. 3. System updates credentials. |
| Alternative Flows | 2a: User not found → Show error. |

**22. List of All Admins & Customers**

|  |  |
| --- | --- |
| **Use case** | List of All Admins & Customers |
| Goal | View all registered users filtered by role. |
| Preconditions | Users must exist. |
| Success End Condition | Filtered list shown. |
| Failed End Condition | Unable to display the list. |
| Primary Actor | Admin |
| Secondary Actor | Database System |
| Trigger | Rolls are selected to filter. |
| Success Scenario | 1. Admin selects role. 2. The system filters and displays a list. |
| Alternative Flows | 2a: No matches found → Display error! |

**23. Pending Order List**

|  |  |
| --- | --- |
| **Use case** | Pending Order List |
| Goal | View pending orders. |
| Preconditions | The order must be placed. |
| Success End Condition | The list is shown successfully. |
| Failed End Condition | No pending orders. |
| Primary Actor | Admin |
| Secondary Actor | Database system |
| Trigger | Tap to see pending orders. |
| Success Scenario | 1. The system filters orders by "pending". 2. Displays details for each order. |
| Alternative Flows | 1a: No results → Display empty message. | |

**24. Delivered Order List**

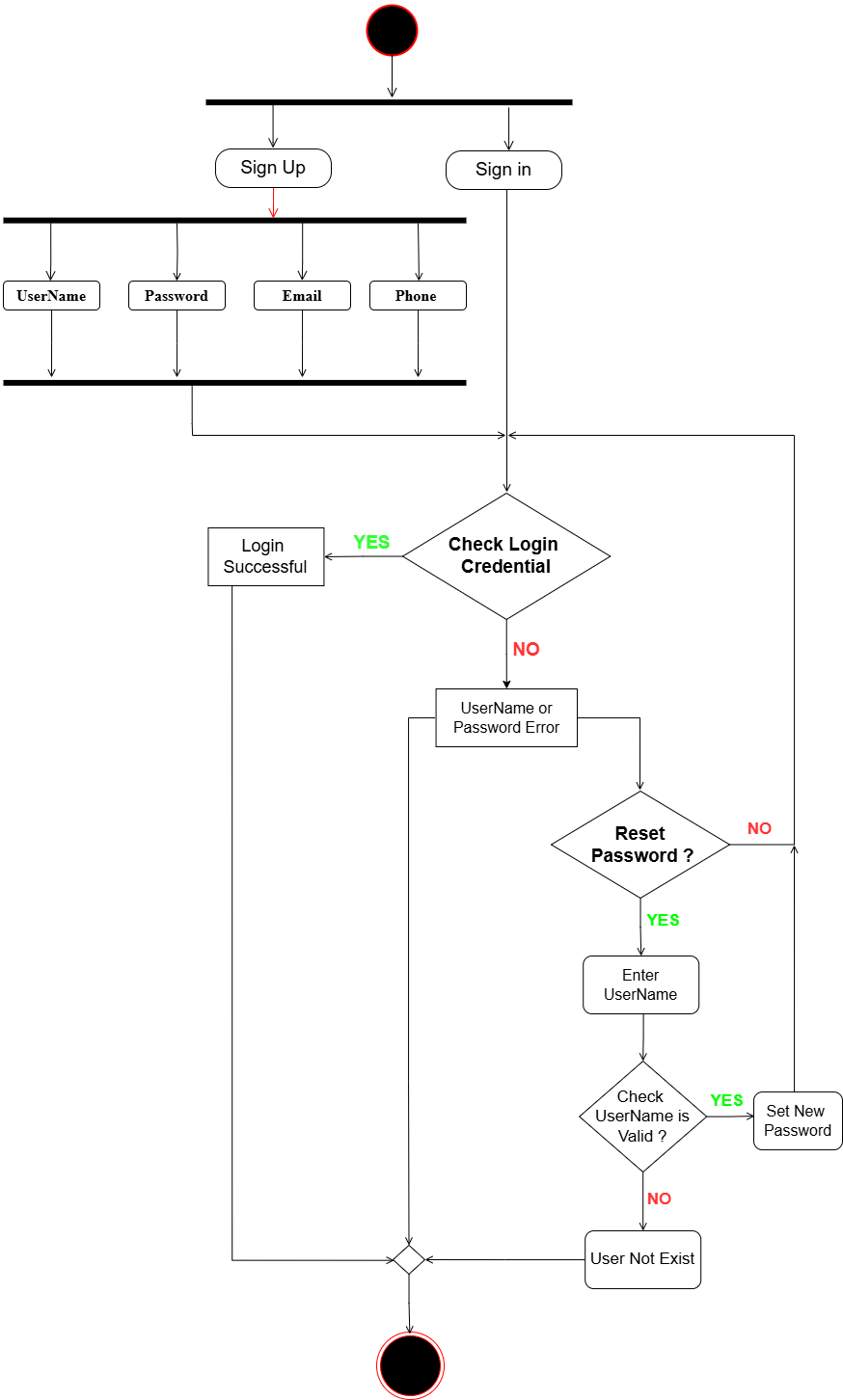
|  |  |
| --- | --- |
| **Use case** | Delivered Order List |
| Goal | View delivered orders. |
| Preconditions | The order must be placed. |
| Success End Condition | List shown successfully |
| Failed End Condition | No orders. |
| Primary Actor | Admin |
| Secondary Actor | Database System |
| Trigger | Tap to see delivered orders. |
| Success Scenario | 1. System filters by "delivered" status. 2. Displays order details |
| Alternative Flows | 1a: No results → Display empty message. |

**25. Checkout**

|  |  |
| --- | --- |
| **Use case** | Checkout |
| Goal | Finalize cart items. |
| Preconditions | The user is logged in, and the cart is not empty. |
| Success End Condition | Order placed. |
| Failed End Condition | Failed to check out. |
| Primary Actor | User |
| Secondary Actor | Inventory System |
| Trigger | User clicks checkout. |
| Success Scenario | 1. User reviews items. 2. Confirms shipping. 3. Proceeds to payment. |
| Alternative Flows | 1a: Item out of stock.  2a: Cannot ship to this address. |

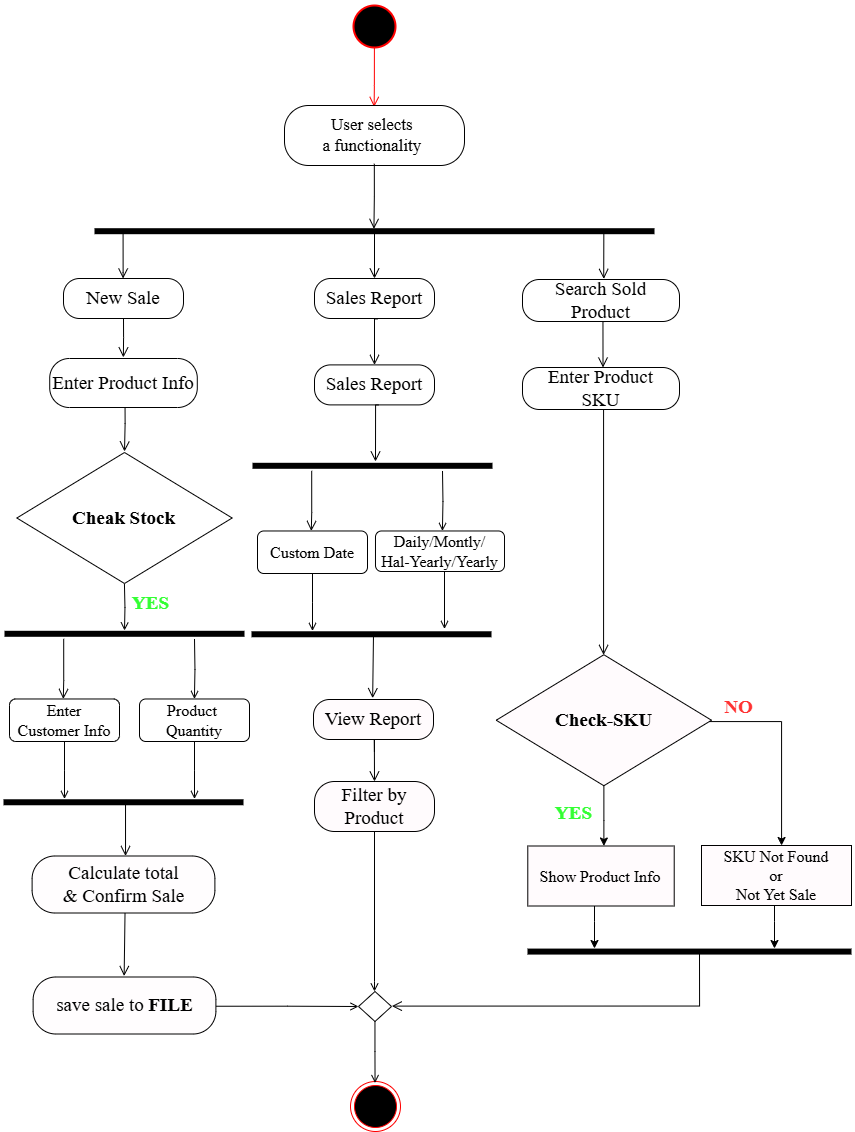
**26. Payment**

|  |  |
| --- | --- |
| **Use case** | Payment |
| Goal | To complete payment for an order. |
| Preconditions | Valid checkout credentials. |
| Success End Condition | Payment Successful. |
| Failed End Condition | Payment Unsuccessful. |
| Primary Actor | User |
| Secondary Actor | Inventory System |
| Trigger | Select payment method. |
| Success Scenario | 1. The user selects a payment method. 2. Enter the required info. 3. Payment processed |
| Alternative Flows | 2a: Invalid info → Display payment unsuccessful.  3a: Insufficient Balance → Display payment unsuccessful. |

****

* 1. **Activity Diagram**

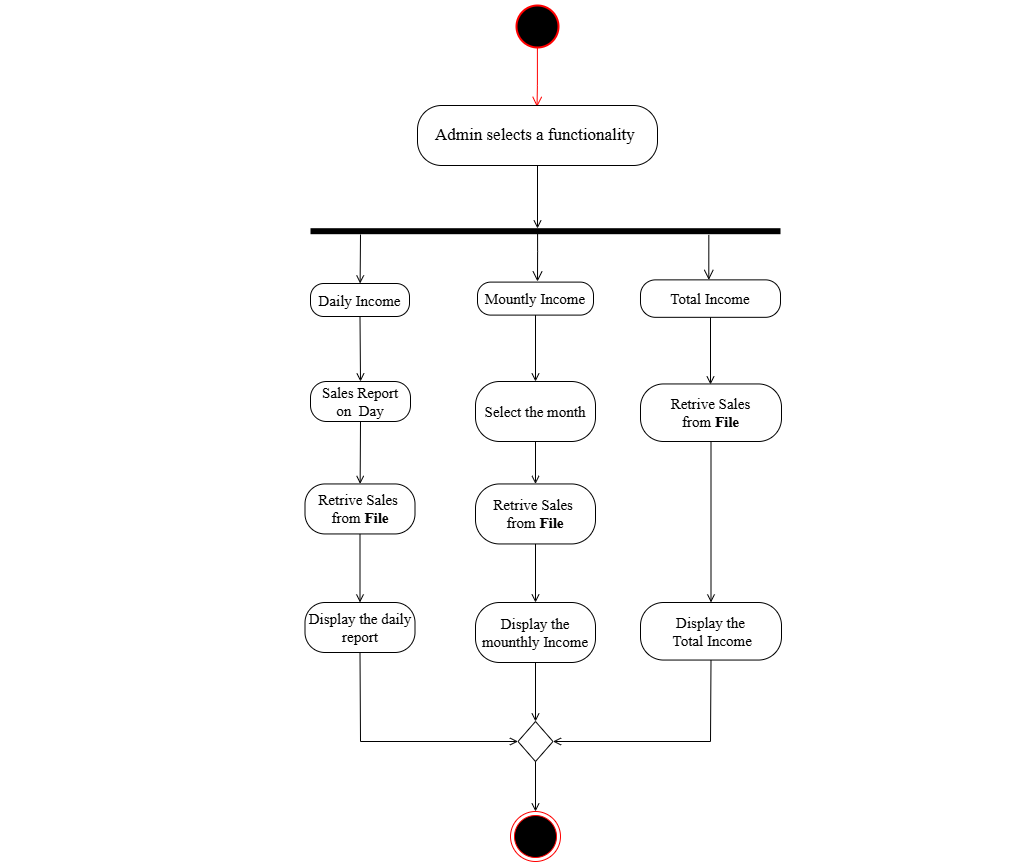
1. **Authentication & Authorization**
2. **Sales Management**



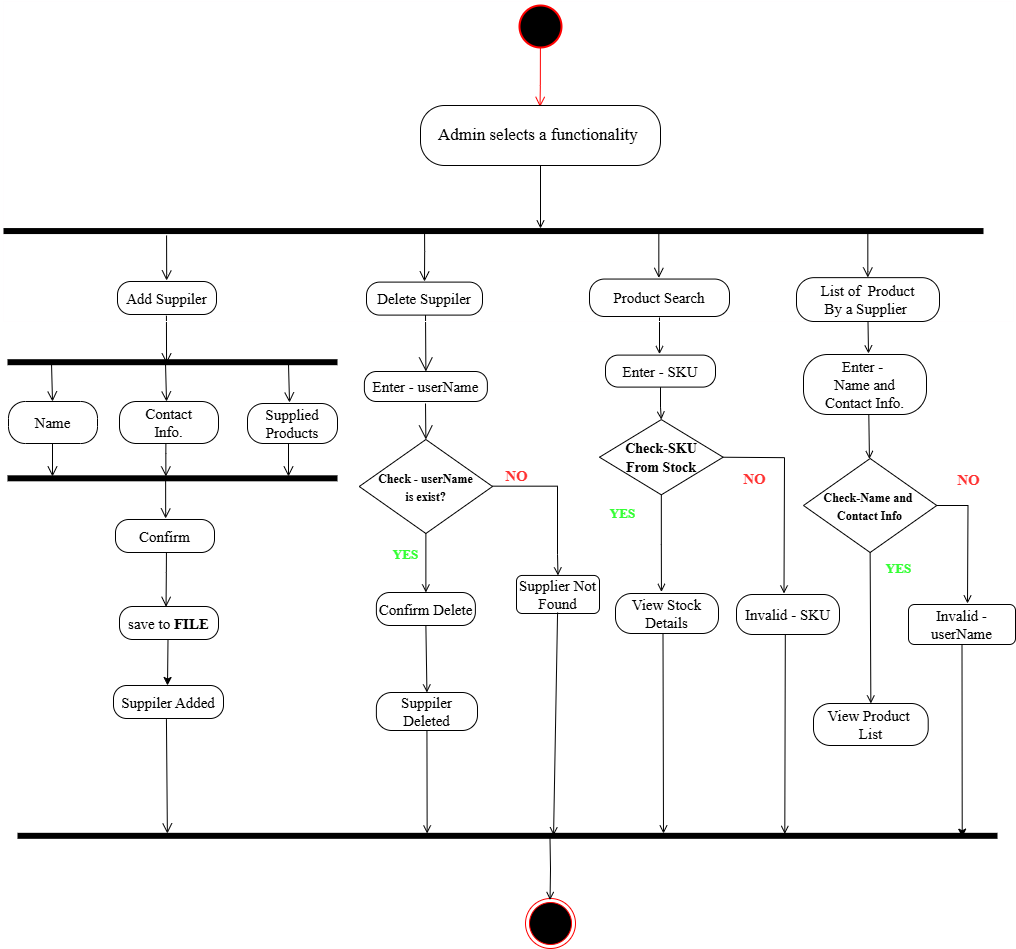
1. **Stock and Product Management**

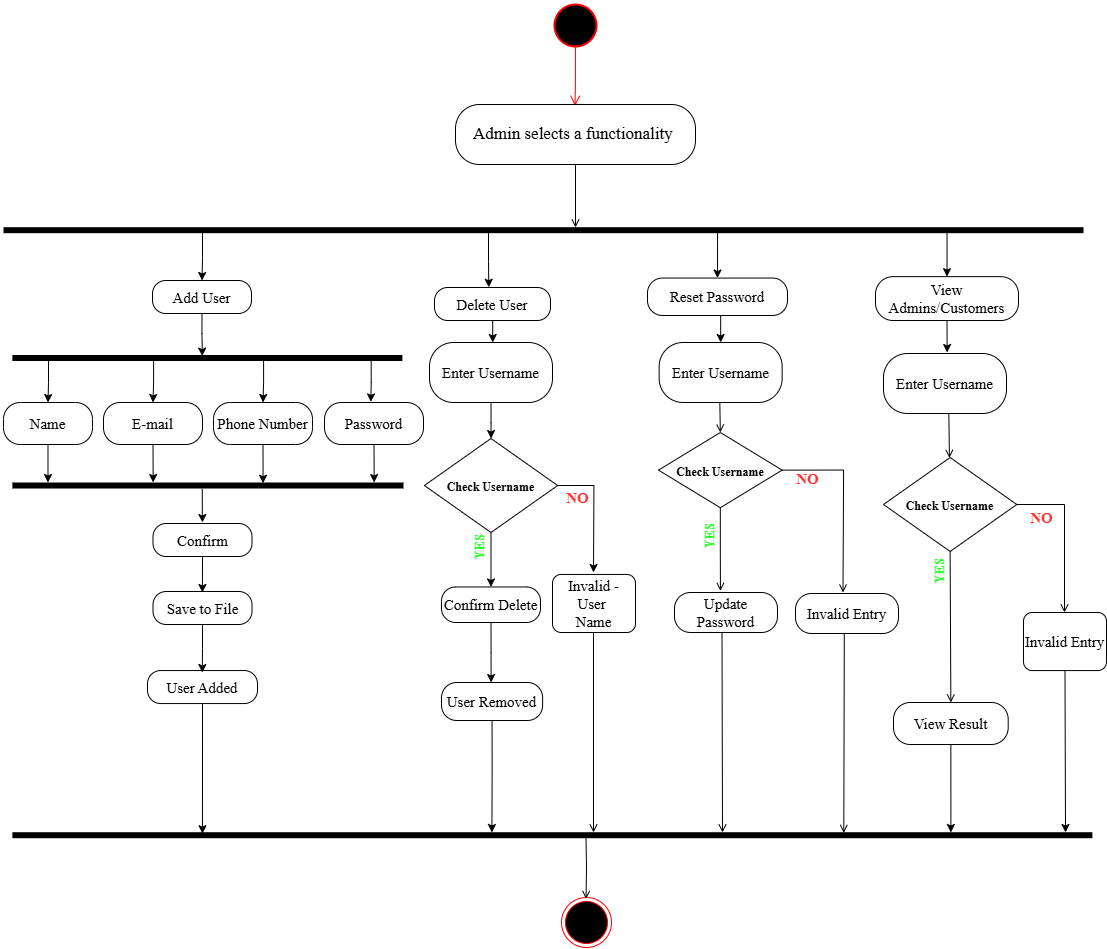


1. **Accounts**

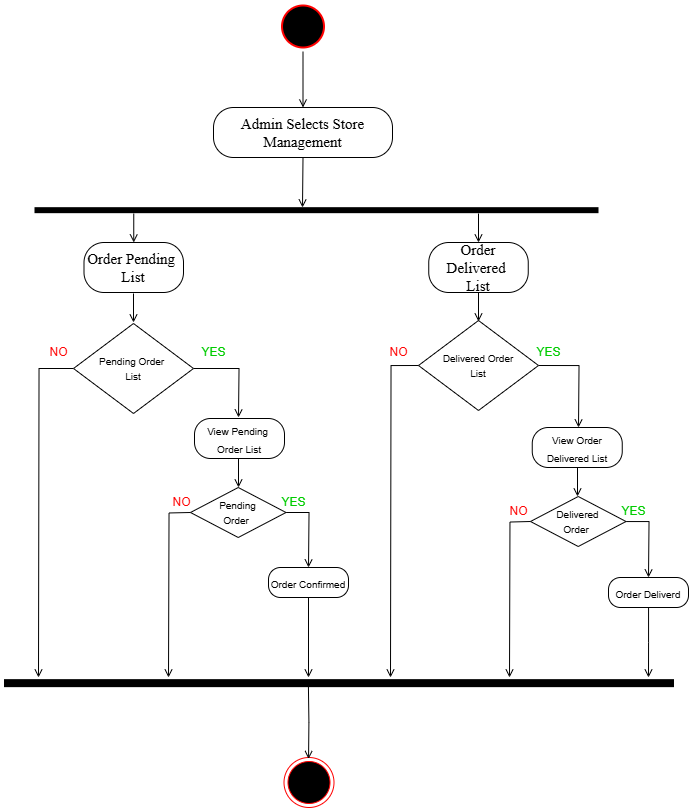


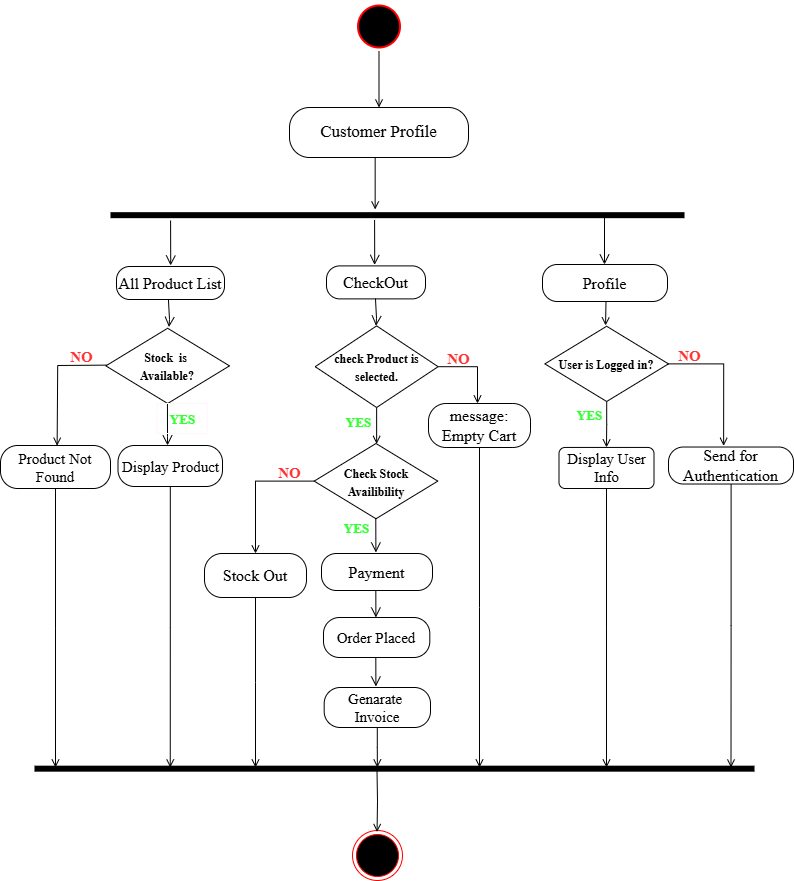
**5. Supplier Management**



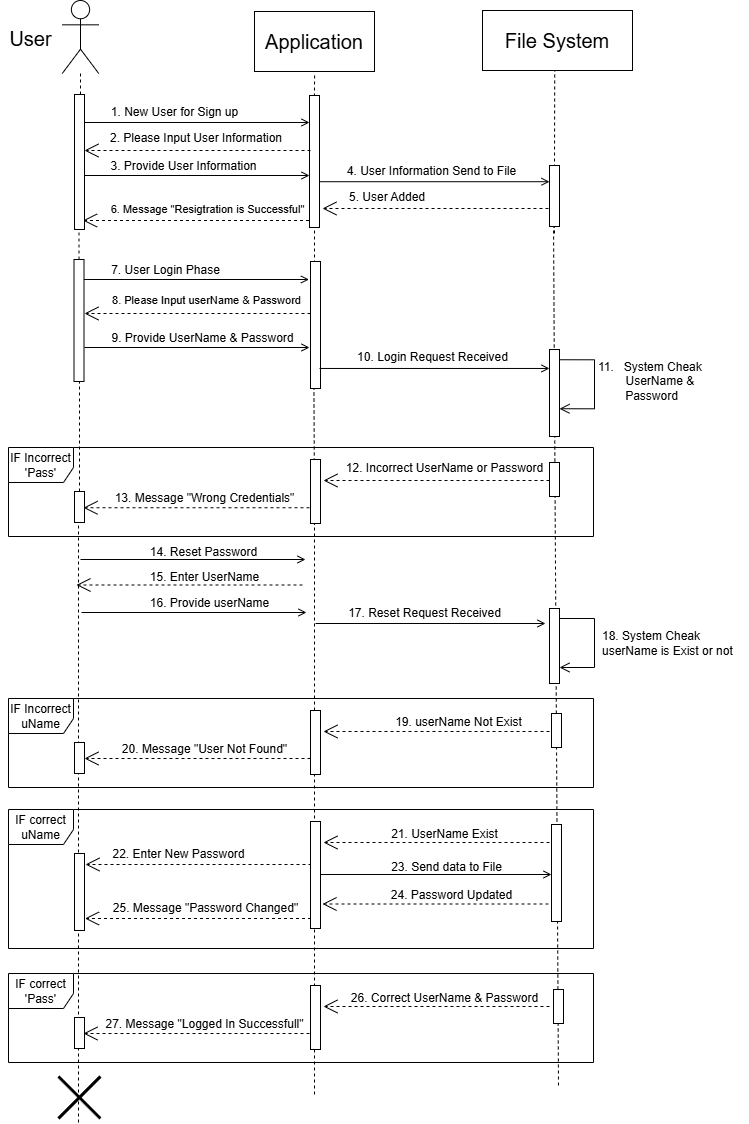
6. User Management

**7. Online Store Management - Admin Panel**

****



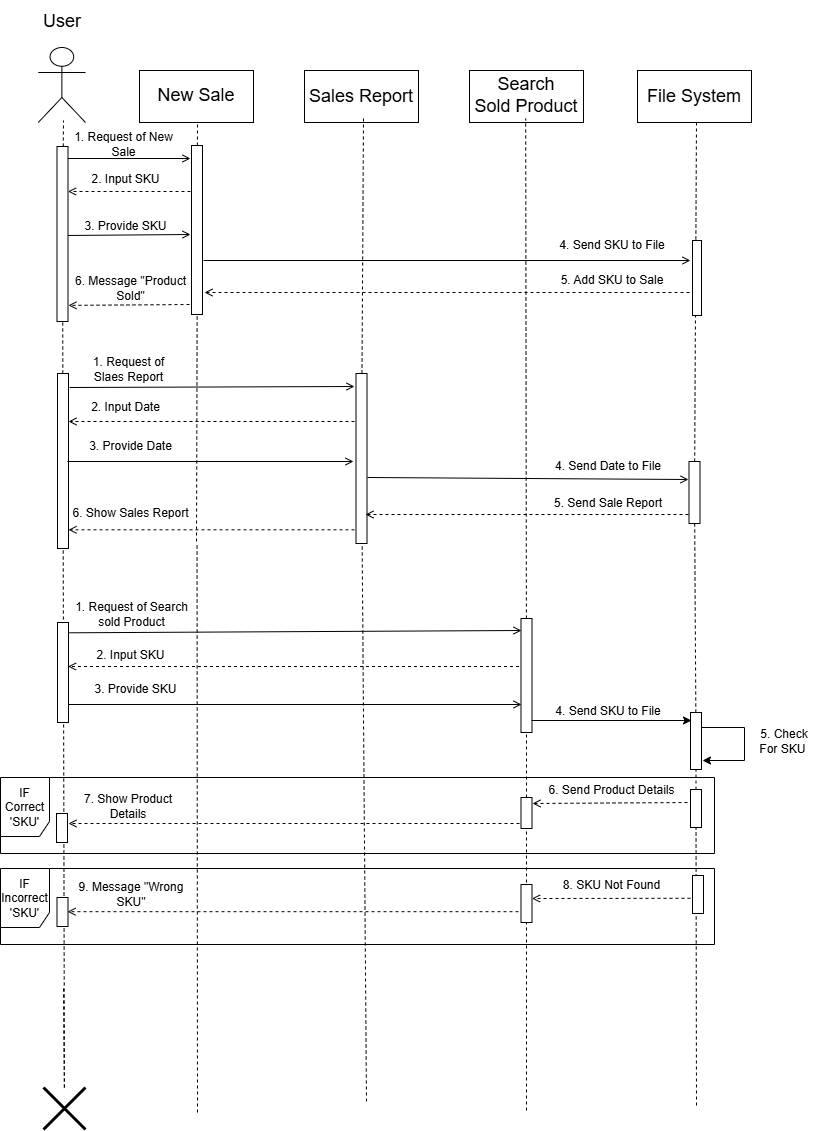
**8. Online Store Dashboard – Customer Panel**

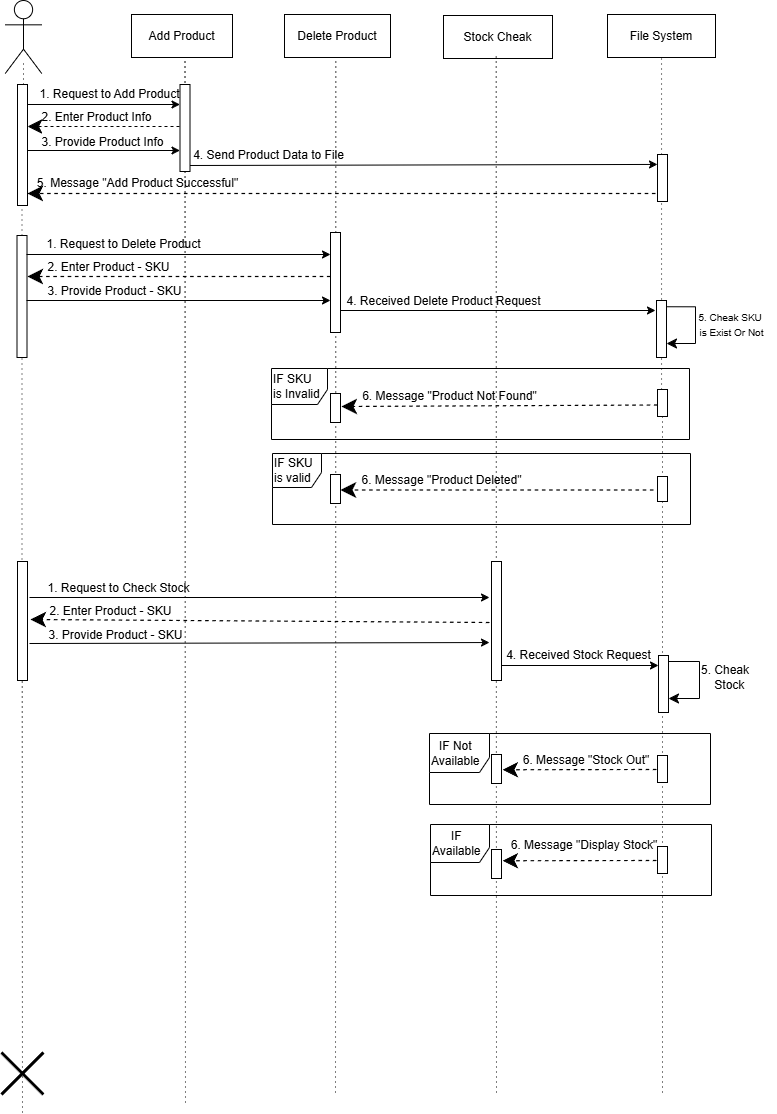


**4.4 Sequence Diagram:**

1. Authentication & Authorization

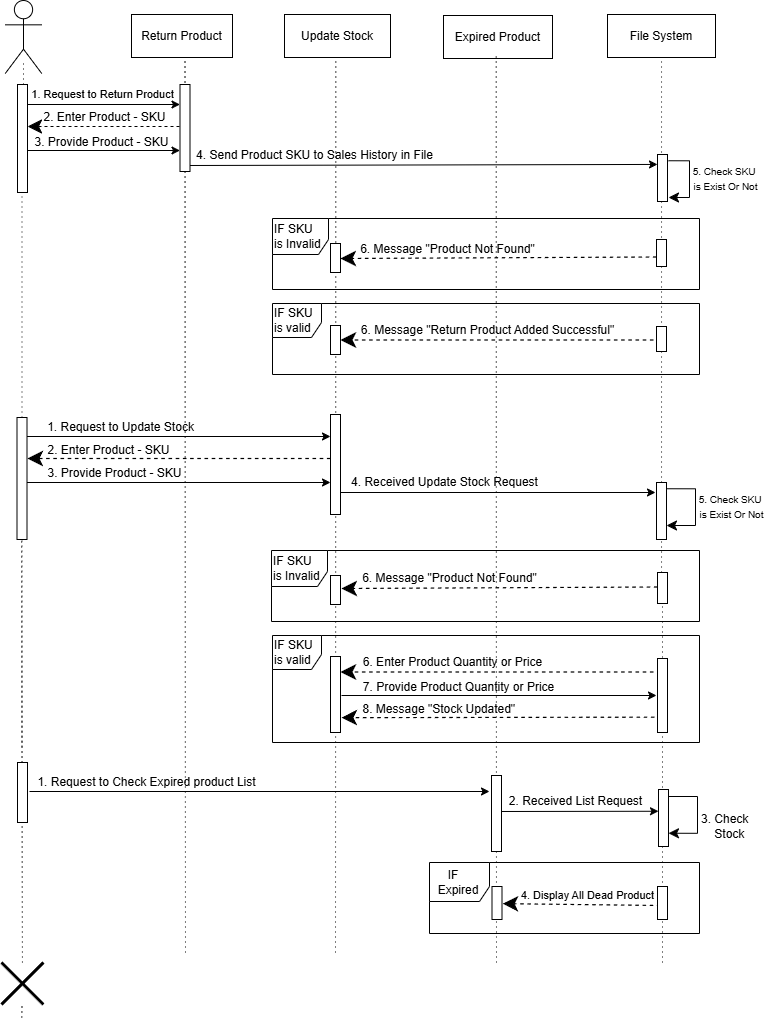
**2.** Sales Management



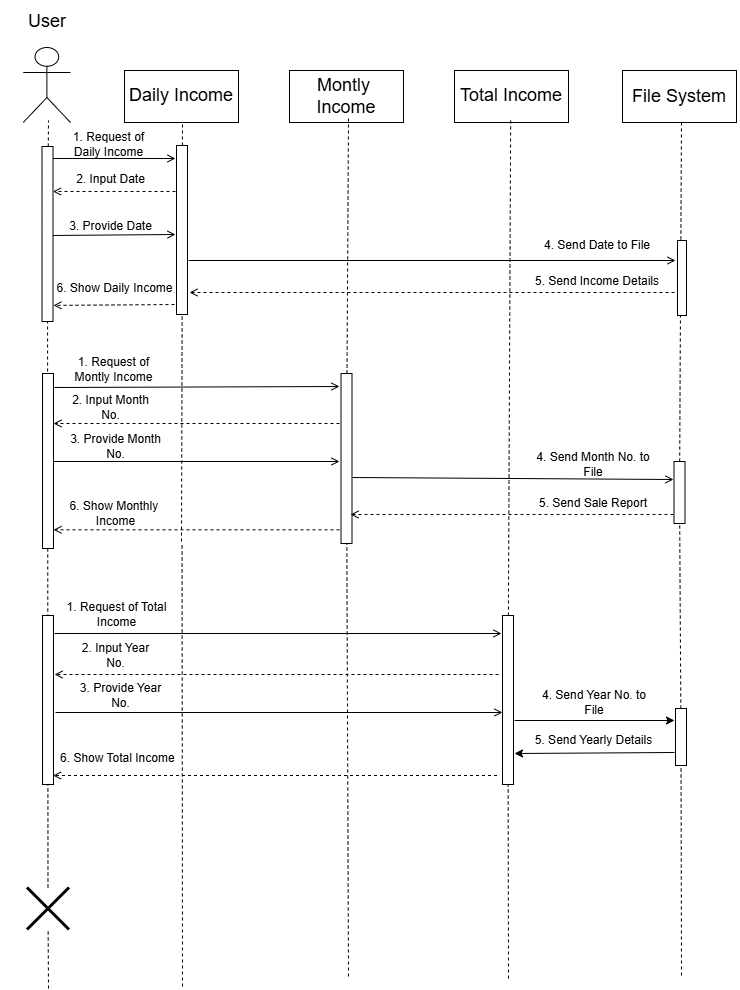


**3.** Stock & Product Management - Part 1

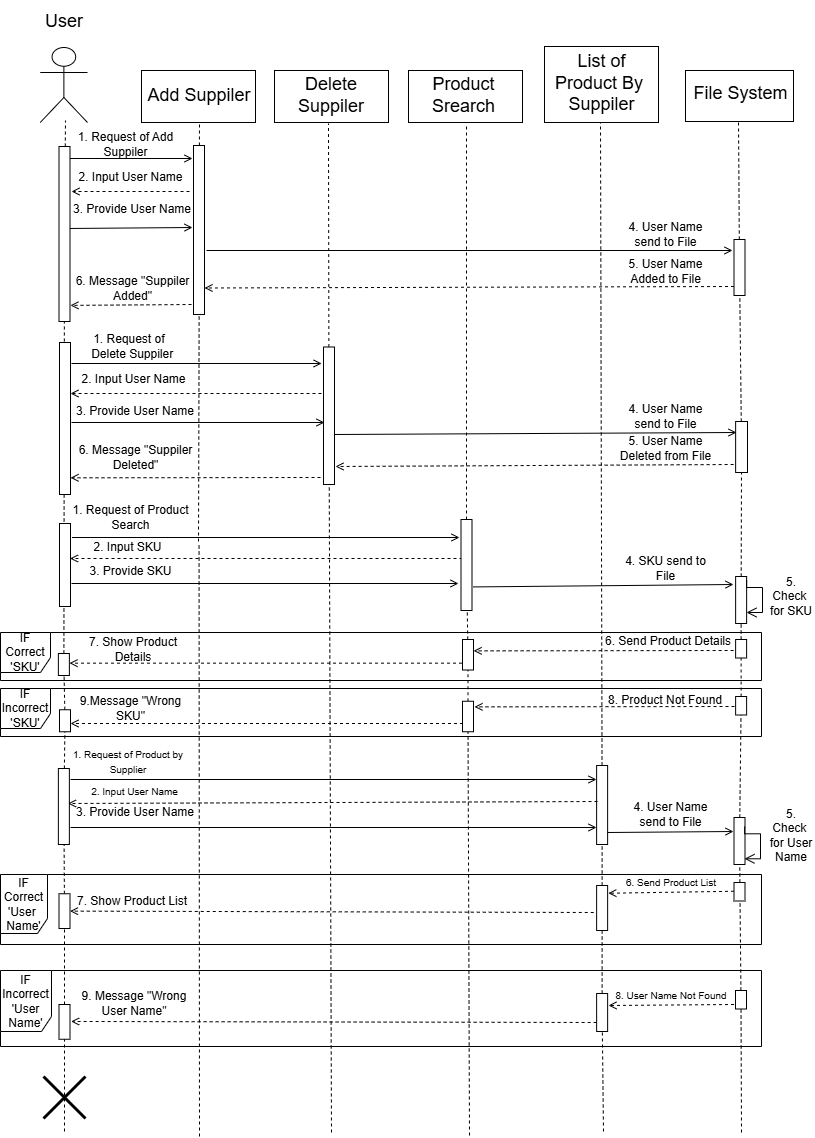
Part 2



**4.** Accounts



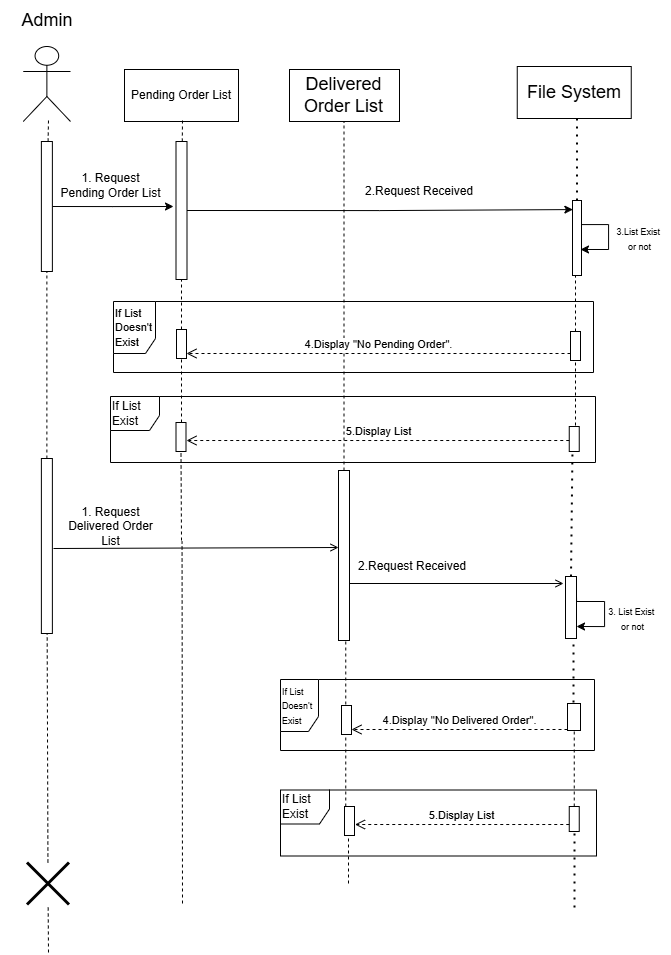
**5.** Supplier Management



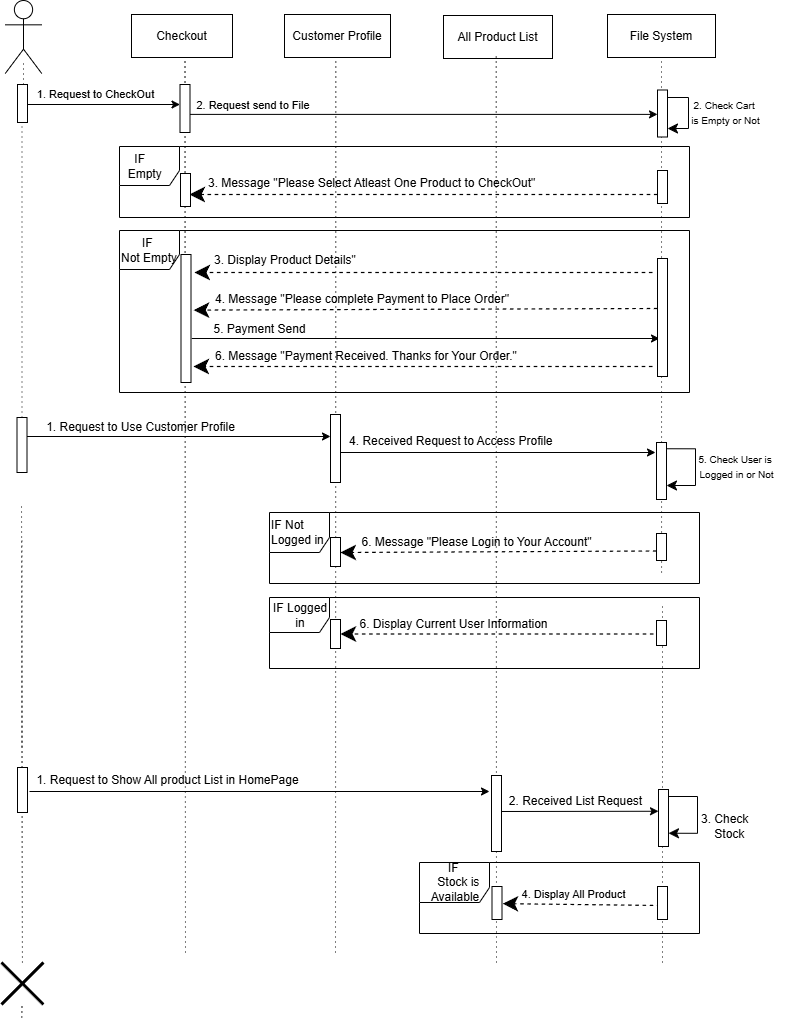


6. User Management

7. Online Store Management – Admin Panel



**8.** Online Store Dashboard – For Customer



**Chapter 5: Implementation**

**5.1 Methodology:**

**Approach:**

The stock and inventory management project is a software application entirely built in the C language. So, it is a console-based software, and it has a modular and step-by-step development approach. The main goal is to build a terminal-based stock and inventory management system that is secure, easy to use, and well-structured. The system will follow the basic principles of software engineering, including:

* Problem identification:
* Requirement analysis:
* Design:
* Implementation:
* Testing and bug fixing:
* Documentation and presentation:

This approach allows us to develop a working system that is easy to test, maintain, and extend in the future. And it will allow us to have an overall idea about this type of inventory management project.

**Tools:**

As the project is a console-based software, some basic yet important tools are used, which are very crucial for a terminal-based software. They are:

* C language:
  + Reason: Fast execution, Modularity, Close to the hardware, Dynamic memory allocation, Pointers, and Low cost.
  + Supports file handling, structures, string handling, and encryption.
* Code::Blocks / VS Code:
  + For writing and compiling C code.
* GCC (GNU Compiler Collection):
  + For compiling in Windows, Linux, and macOS environments.
* Terminal / Command Prompt:
  + For running and testing the application.

**Techniques:**

* File Handling:
  + All data, like stocks, inventory, accounts, and supplier details stored in text files.
* Structured Programming:
  + Use of functions and structures for modular code.
* Role-Based Access:
  + Different dashboards for Admin and client.
* Unique ID Generation:
  + For tracking each stock and inventory.
* Input Validation:
  + To avoid incorrect data and improve security

**Model:**

As the whole software is built on the C language, that’s why the software will be a console-based software. And for this, there is no scope for increment, and this lids to no scope for feedback and iteration of the software. And for this reason, we have to choose a model where the project is at first properly planned and documented. Now, by analysing the pros and cons of the project, we thought that the waterfall model would be a perfect model for this project. As the project is small and not that complex, we easily access different documents about our project and make a well-planned document, and start the implementation part. The plan for the project for proper execution is the most crucial part of the project.

Waterfall Model:

The Waterfall Model is a linear and sequential approach to software development where each phase, such as requirements analysis, system design, implementation, testing, deployment, and maintenance, is completed fully before the next begins. It flows in one direction, like a waterfall, with little to no room for revisiting or modifying earlier stages once they are completed. This model is best suited for projects with well-defined, stable requirements and a low likelihood of changes. While it offers a structured and disciplined process, its main drawback is the lack of flexibility and minimal customer feedback during development, which can lead to challenges if requirements evolve later.

**Phases:**

Here is a phase-wise description of an Inventory and Stock Management System developed using the C language, based on the Waterfall Model:

**1. Requirement Analysis**

In this phase, all requirements for the inventory and stock management system are gathered by consulting with stakeholders such as admins or the inventory manager. Key features identified include product entry, stock updates, product search, stock level alerts, expired item tracking, and sales records. Both functional and non-functional requirements are documented clearly to guide the entire development process.

**2. System Design**

Based on the requirements, a detailed system design is created. This includes defining the overall architecture of the program in C, such as using structures (struct) for products, file handling for data storage, and menu-driven interfaces. The design also outlines how different modules (like product management, stock management, and reporting) will interact. Data flow diagrams and pseudocode are also prepared.

**3. Implementation (Coding)**

In this phase, the system is developed using the C programming language. Each module is coded as per the design specifications. Arrays, structures, file operations (fopen, fwrite, fread, etc.), and functions are used to implement features like adding products, updating stock, searching inventory, and generating reports on sales and accounts. The code is written in a modular and reusable format for clarity and maintainability. And proper commenting is done to maintain proper readability of the code.

**4. Testing**

Once the system is developed, it undergoes thorough testing to ensure accuracy and reliability. Each module is unit-tested individually to check for logic errors or incorrect outputs. Then, integration testing is performed to verify the interaction between modules. Boundary cases, such as invalid input or empty stock, are also tested to ensure robustness and proper reliability.

**5. Deployment**

After successful testing, the system is deployed for use in the target environment, such as a retail shop or warehouse. Executables are created, and necessary documentation and user manuals are provided. Training may also be given to users. Since the system is built in C, it runs on any platform with a compatible C compiler or runtime environment. As it is a program written in C, it is mainly interacted with on a terminal or console.

The development of the Inventory and Stock Management System using the Waterfall Model ensures a structured and disciplined approach where each phase is completed in a logical sequence. By following the stages from clear requirement gathering to final deployment, the system is built with a strong foundation and minimal ambiguity. Although the Waterfall Model offers limited flexibility for changes once development begins, its phase-by-phase clarity makes it ideal for well-defined systems like this one. Ultimately, the model helps deliver a functional, efficient, and reliable inventory system developed in C, tailored to meet user needs with precision and consistency.

**5.2 C Concepts Used in the Project:**

**1 Variables and Data Types:**

Variables and data types are mainly used to store item details like product ID, name, quantity, and price for a short time when we retrieve the data from the file, where all the data is stored.

**2 Control Structures:**

if, else, switch, for, while, and do-while loops are used to control the program flow, such as switch case, which are the most crucial part of taking user input to navigate through the program, navigating the menu, or processing user inputs.

**3 Functions:**

The system is broken into modular functions. Which helps us to easily assess different functionality. We have also used a particular function for the user interface of the program for easy UI management. Also, we have functions like addProduct(), searchProduct(), updateStock(), generateReport(), and many more to improve reusability and code organization. It also helps us to manage different features of the program.

**4 Arrays:**

Array is one of the most crucial parts used. It is mainly used to temporarily store and manipulate a list of products if a particular data needs to be updated, as we cannot update data in a C file using a SKU or code number, we need to update the data in an array to do the task of updating.

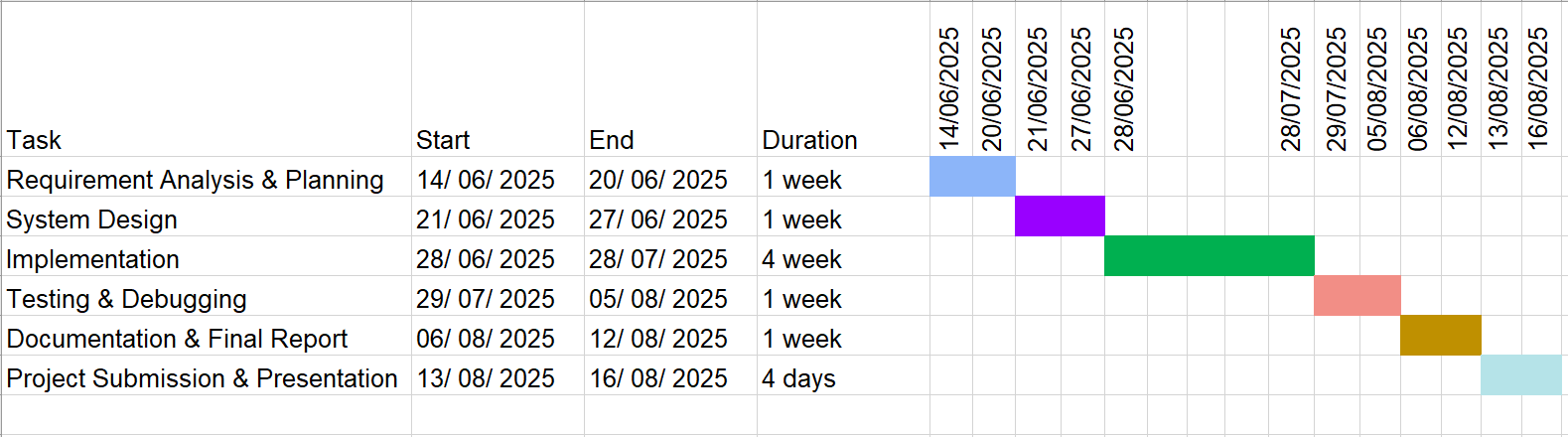
**5 Structures (struct):**

Structure is used to group related product attributes (ID, name, quantity, price, expiry date) into a single data type, making inventory management easier. It also makes it easier to manage different data types in a single type.

**6 File Handling:**

Functions like fopen(), fclose(), fprintf(), fscanf(), fread(), and fwrite() are used for persistent storage of product records, enabling saving and retrieving inventory data between sessions. It is used mainly to store all the data of the inventory and the user details. It can be identified as the most important function of the program, as the program's main task will be to store data and retrieve it when it is needed. And for this, the file handling of the C language will be a crucial part of the software.

**5.3 Gantt Chart:**

****

**Chapter 6: Conclusion**

The development of the Stock and Inventory Management System (SIMS) marks a significant stride toward efficient, organized, and accessible inventory control for small to medium-sized businesses. Designed using the C programming language and structured under the Waterfall Model, this project delivers a terminal-based solution that is not only functionally robust but also modular, secure, and user-friendly.

Throughout the project, careful attention was given to every phase starting from clear requirement analysis and detailed system design to implementation, testing, and deployment. With features such as sales management, supplier and user management, account reporting, and product categorization, the software meets the critical needs of business inventory operations. File handling techniques in C ensured reliable data persistence, while structured programming and role-based access management strengthened usability and security.

Despite being a console application, SIMS demonstrates the power and flexibility of C in solving real-world management challenges. It successfully combines essential business functionalities into a lightweight, low-resource tool. Future enhancements could include graphical user interface (GUI) support, cloud integration, and real-time analytics to further improve the system’s usability and reach.

In conclusion, this project stands as a practical application of core C concepts and software engineering principles. It showcases how even foundational technologies, when structured properly, can provide impactful solutions to modern-day problems in the domain of business management.