REPUBLIQUE DU CAMEROUN

PAIX -TRAVAIL-PATRIE

****

INSTITUT AFRICAIN
D'INFORMATIQUE

****

BP : 13 719 Yaoundé (Cameroun)

Tel: (237) 22 72 99 57 / (237) 22 72 99 58

Site Web: www.iaicameroun.com

REPUBLIQUE OF CAMEROON

PEACE-WORK-FATHERLAND

****

AFRICAN INSTITUTE OF
COMPUTERS SCIENCES

****

BP: 13 719 Yaoundé (Cameroun)

Tel: (237) 22 72 99 57 / (237) 22 72 99 58

Site Web: www.iaicameroun.com

JAVA PROJECT

GROUP 9: WINE STOCK MANAGEMENT APP

GROUP MEMBERS

- NOUBISSIE LAURIE
- NGOUATEU NGOUFACK DOLOIC

SUPERVISOR

Mr. FOMEKONG

ACADEMIC YEAR
2024 - 2025

TABLE OF CONTENTS

1 INTRODUCTION

Welcome to our app wine management system, the ultimate solution for managing your wine inventory with ease and precision. Whether you're a wine collector, a restaurant owner, or a wine shop manager, our app helps you efficiently track, organize, and optimize your wine stock. The system is built using Java with NetBeans IDE and JDK, and it employs a MySQL database for data storage. The system has three main actors: Admin , inventory manager and sale person. Each actor interacts with the system to perform specific tasks, such as managing stock, recording sales, and restocking items.

This report provides a detailed explanation of the UML diagrams (use case, activity, sequence, communication, state machine, class, object, package, component, and deployment) and how they relate to the system's functionality. The report also includes an overview of the database schema and Java code implementation.

2 UML DIAGRAMS

2.1 USE CASE DIAGRAM

In UML (Unified Modeling Language), a Use Case represents a specific interaction between users (actors) and a system to achieve a goal. It describes what the system does, not how it does it.

Actors:

- **Admin**: Manages Users, Manage Profile, View stock record, View sales record and view supplier record.
- **Inventory manager**: view sale record, Manage Profile, Manage Suppliers, and Manage sales.
- **Sale person**: Manage stock, Manage Profile, Manage Suppliers, and Manage sales.

Use Cases:

Admin:

- View, Add, Delete, Update User
- View, Add, Delete, Update Sales
- View, Add, Delete, Update Item
- View, Add, Delete, Update Supplier
- Manage Profile

Inventory Manager:

- View, Add, Update Sales
- View, Add, Update Item
- View, Add, Update Supplier
- Manage Profile

Salesperson:

- View, Add Sales
- View Item
- View Supplier
- Manage Profile

Relationships:

- ❖ Admin manages everything in the system.
- ❖ The Inventory Manager mostly handles the stocked item, supplier and sales records.
- ❖ The Salesperson is solely responsible for sale addition but can view items and supplier records

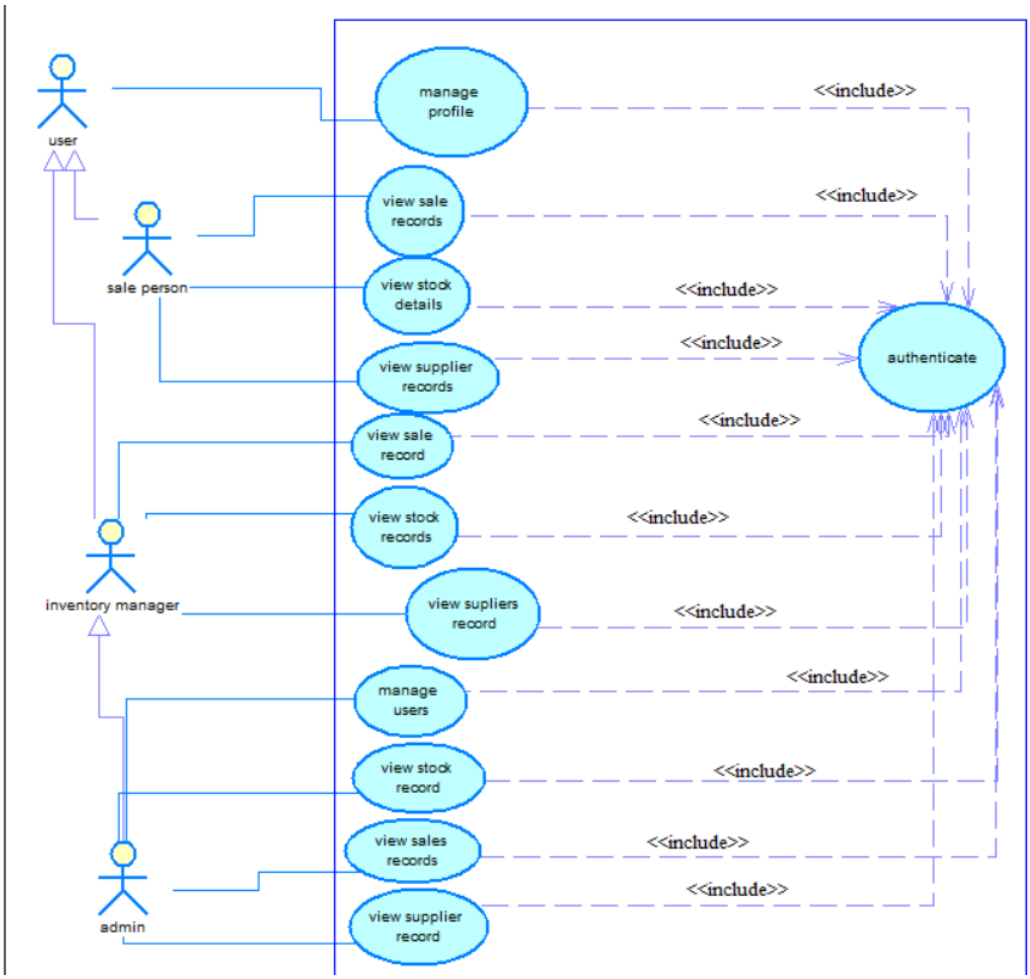


Figure 1.1: GENERAL USECASE DIAGRAM.

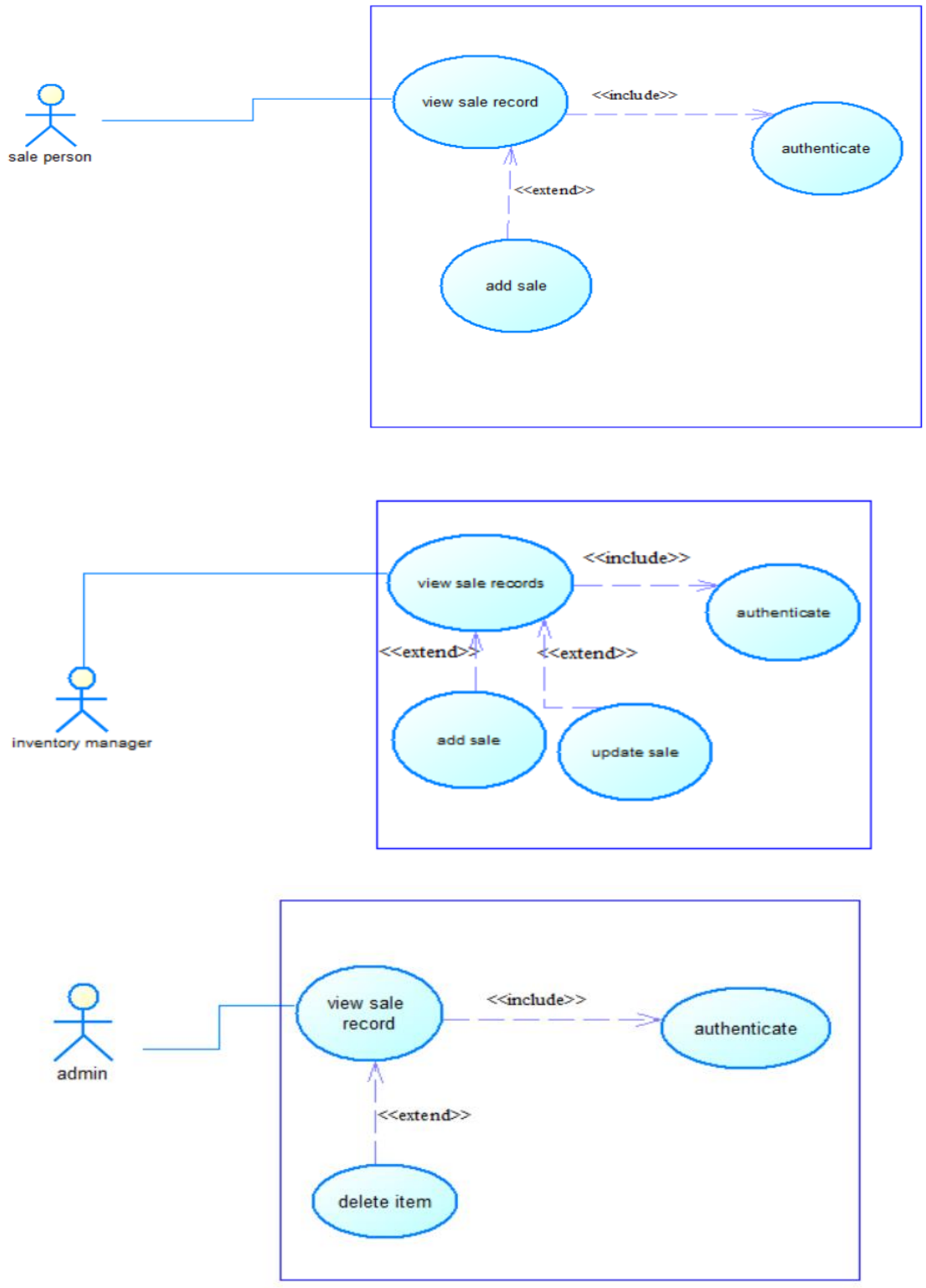


Figure1.2: SPECIFIC USECASE DIAGRAMS OF VIEW SALE RECORD

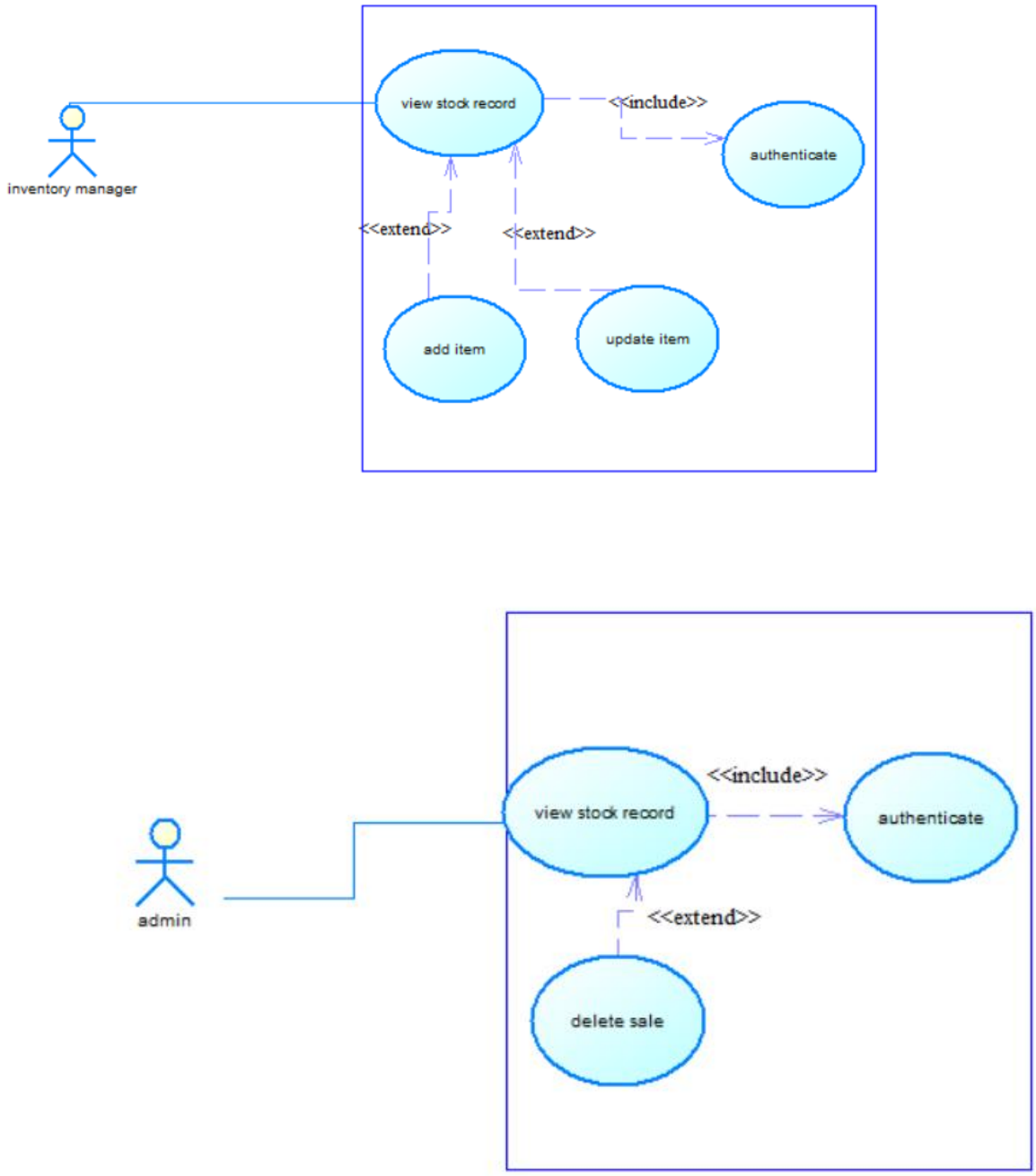
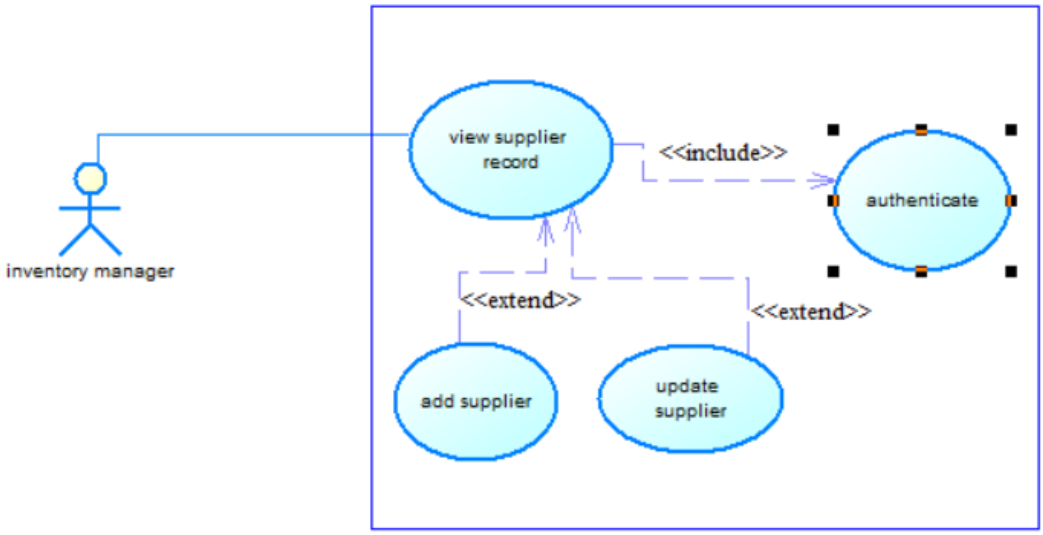


Figure 1.3: SPECIFIC USECASE DIAGRAMS OF VIEW STOCK RECORD



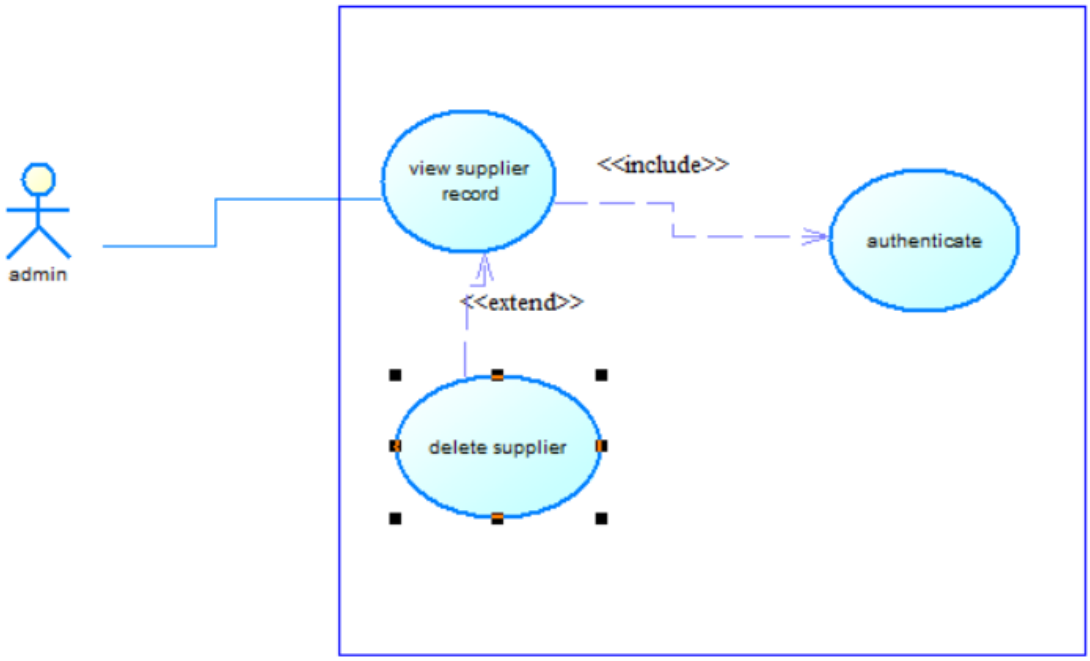


Figure1.4: SPECIFIC USECASE DIAGRAMS OF VIEW SUPPLIER RECORD

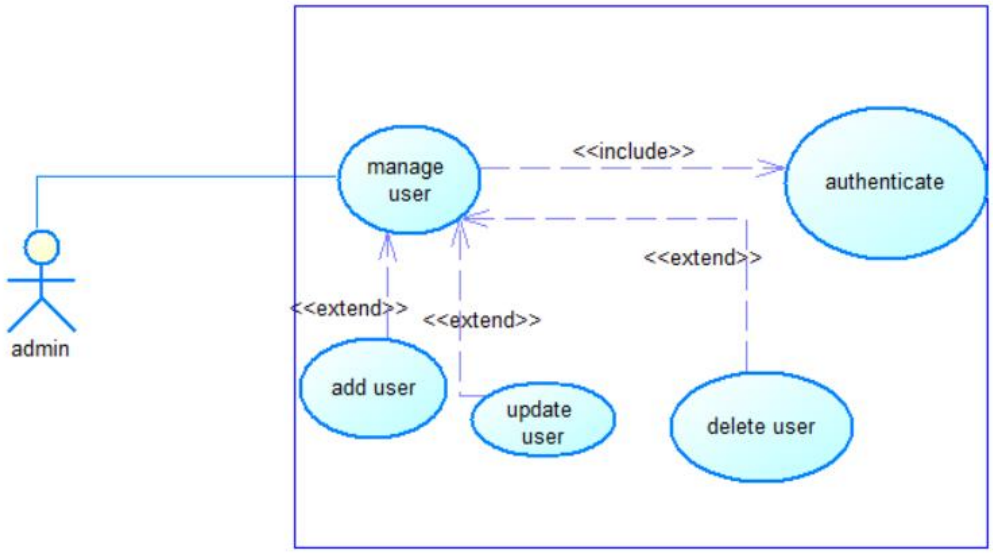


Figure1.5: SPECIFIC USECASE MANAGE USER

2.2 ACTIVITY DIAGRAM

An Activity Diagram in UML represents the flow of activities in a system, showing how a process progresses from start to finish. It is similar to a flowchart but focuses on workflows within a system.

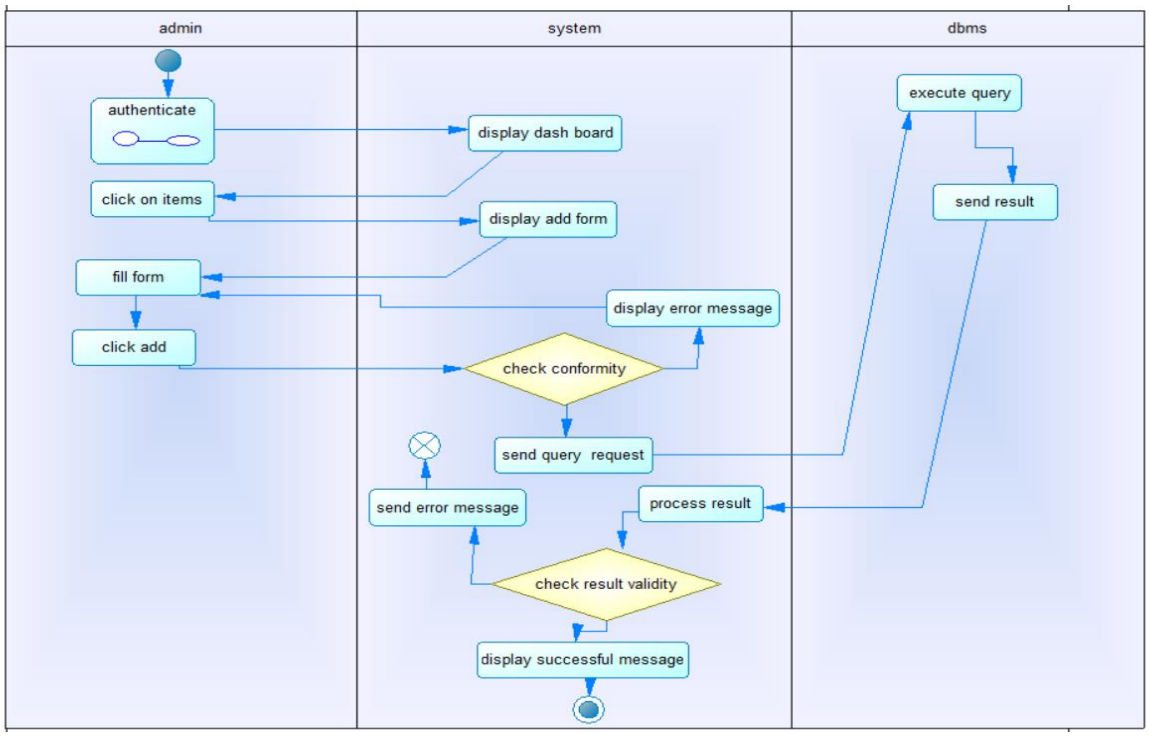


Figure 2.1 : ADD ITEM ACTIVITY DIAGRAM

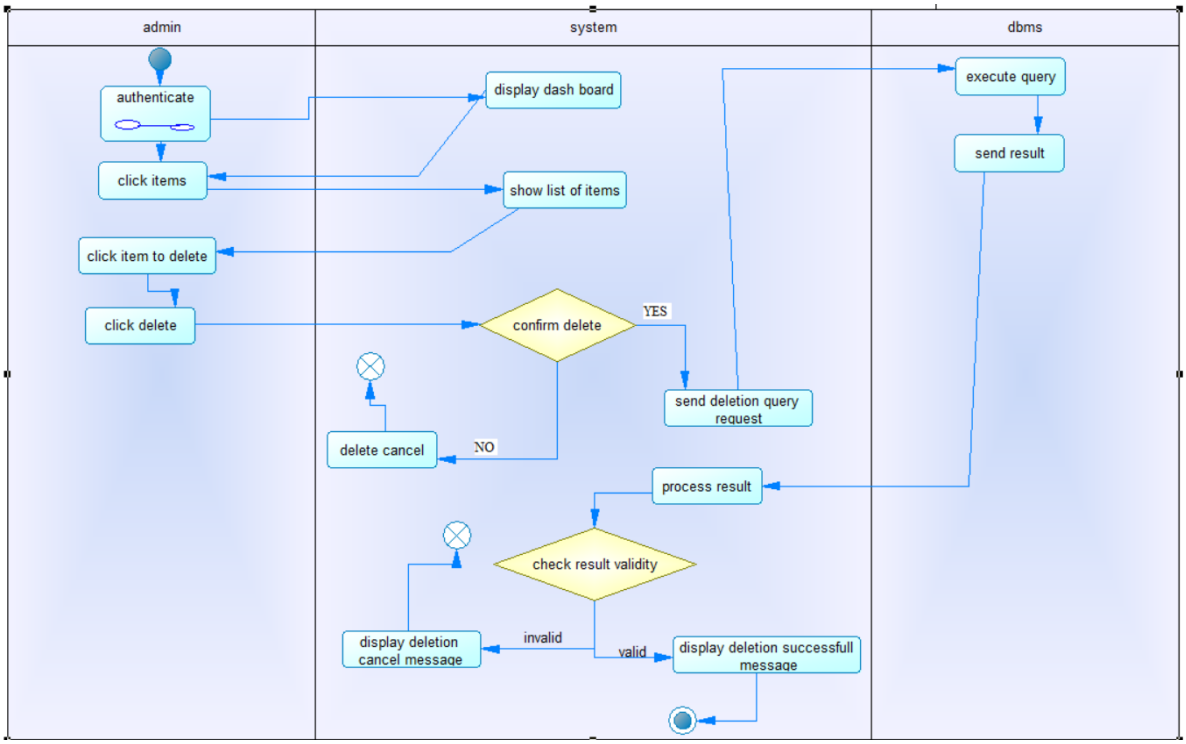


Figure 2.2: DELETE ITEM ACTIVITY DIAGRAM.

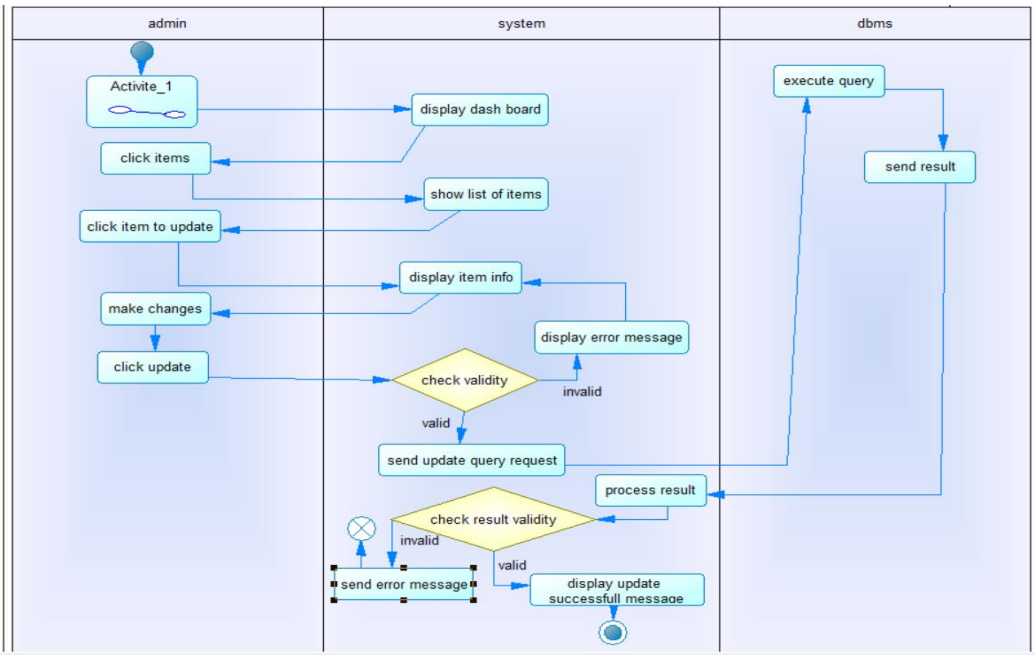


Figure 2.3: UPDATE ITEM ACTIVITY DIAGRAM.

2.3 SEQUENCE DIAGRAM

A Sequence Diagram in UML represents the step-by-step interaction between objects in a system over time. It focuses on the order of messages exchanged between actors and system components.

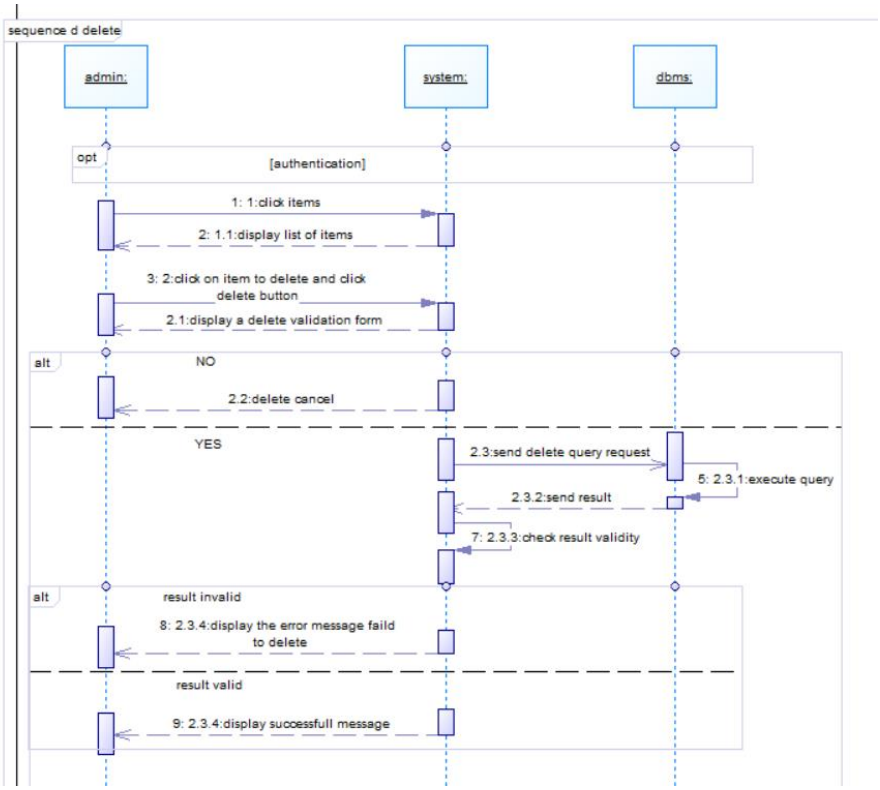


Figure 3.1: DELETE ITEM SEQUENCE DIAGRAM.

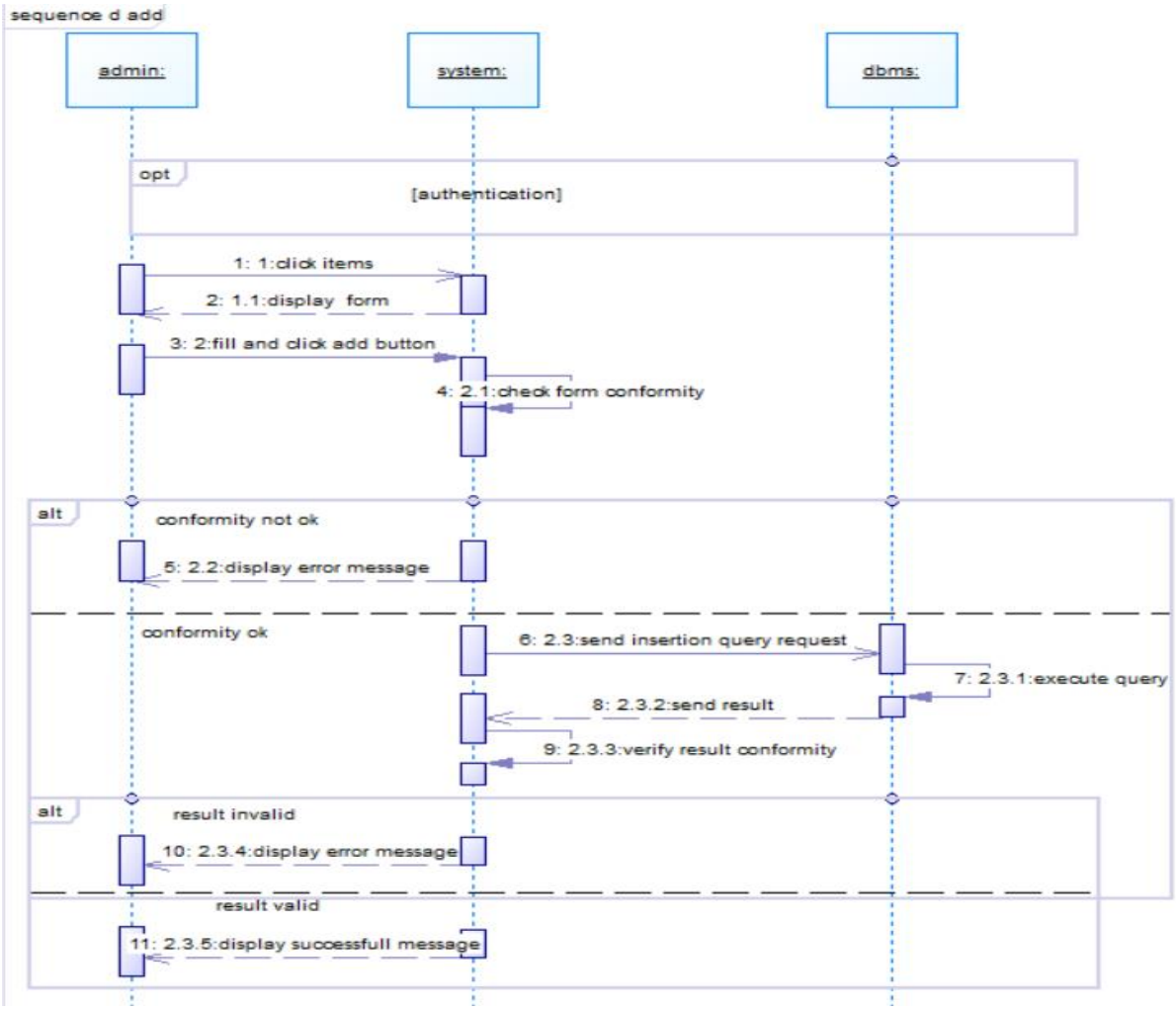


Figure 3.2: ADD ITEM SEQUENCE DIAGRAM.

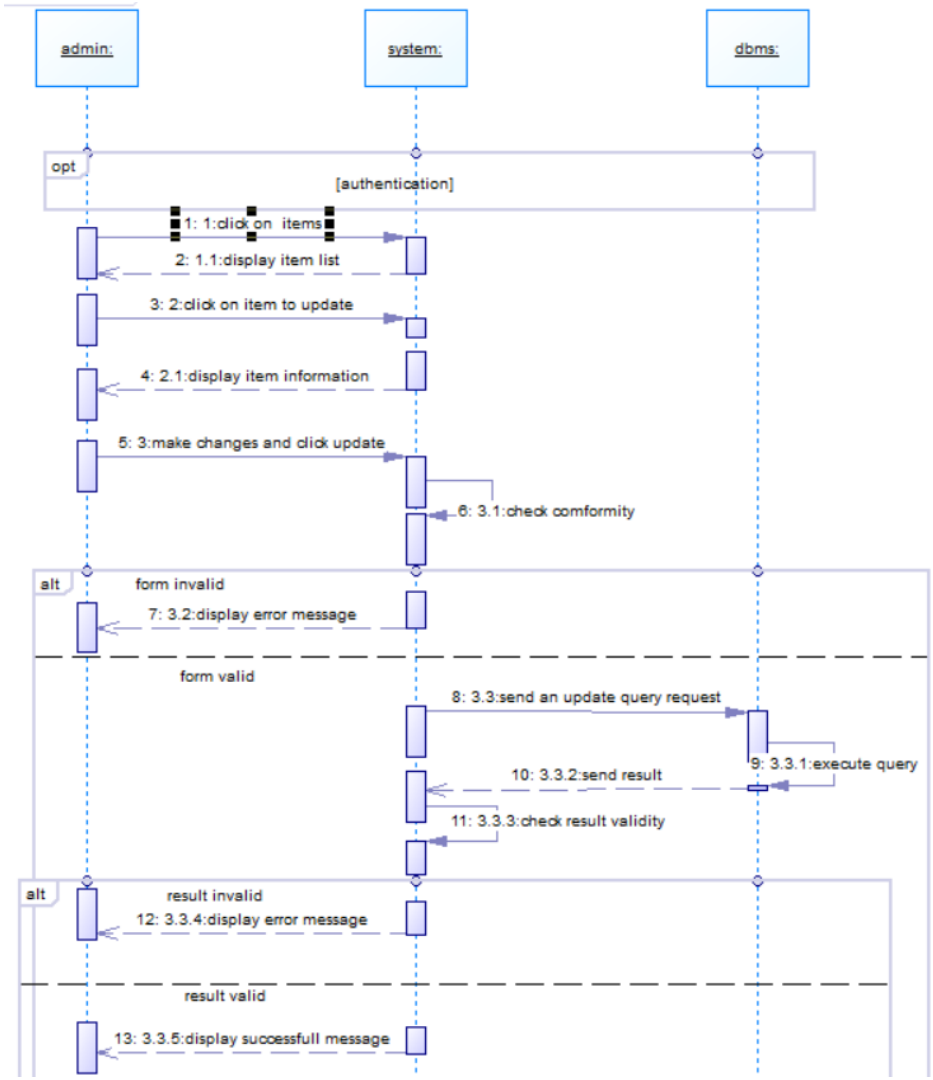


Figure3.3: UPDATE ITEM SEQUENCE DIAGRAM.

2.4 COMMUNICATION DIAGRAM

In UML, a Communication Diagram focuses on how objects interact and exchange messages within a system. It is similar to a Sequence Diagram but emphasizes the relationships between objects rather than the order of messages

Interactions:

- Admin interacts with all Management components.
- Inventory Manager interacts with the Stock Management, Sales Management and Supplier Management components.

❖ Salesperson interacts with the Sales Management components.

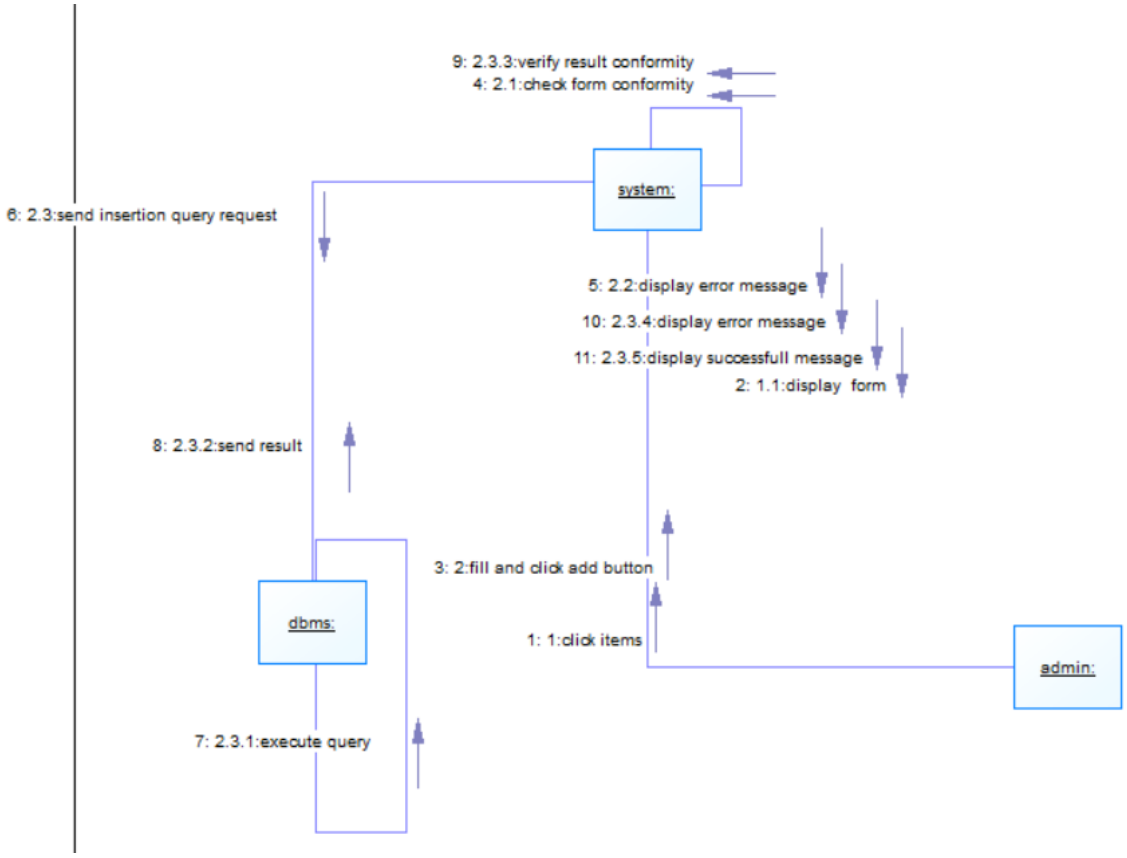


Figure 4.1: ADD ITEM COMMUNICATION DIAGRAM.

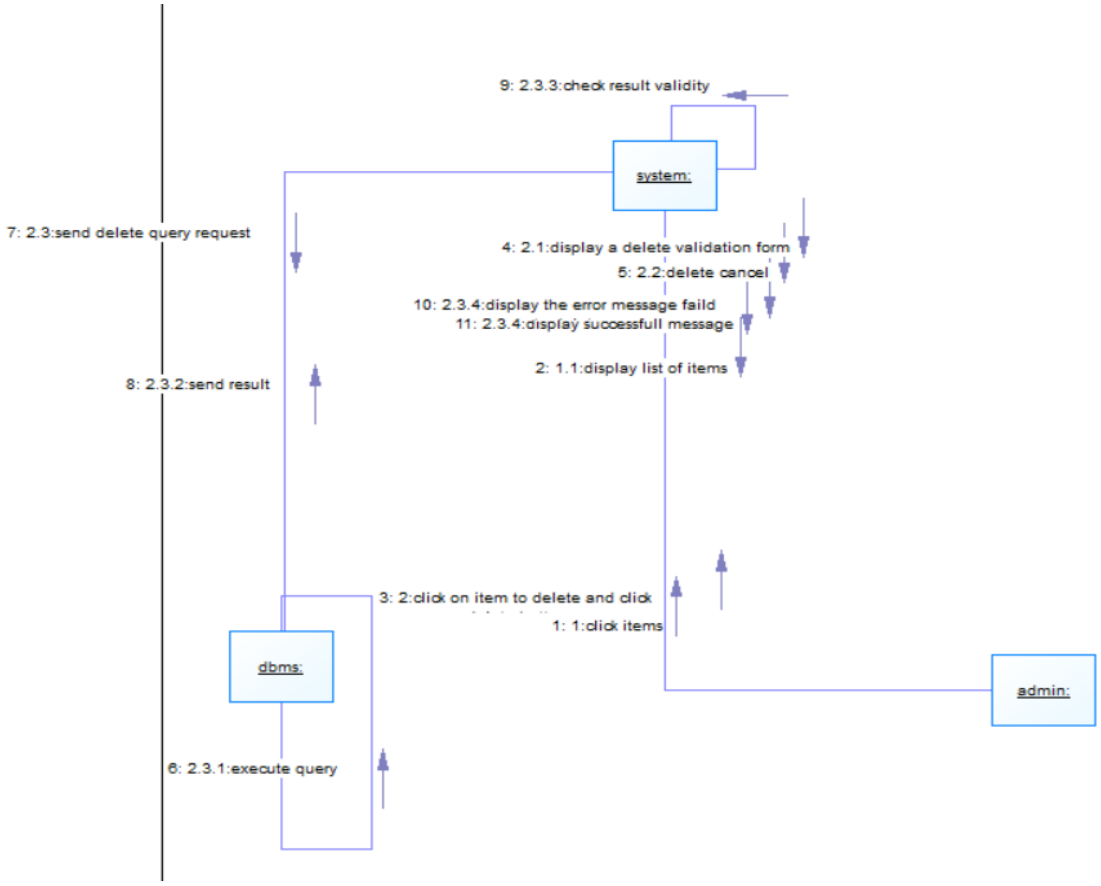


Figure 4.2: DELETE ITEM COMMUNICATION DIAGRAM.

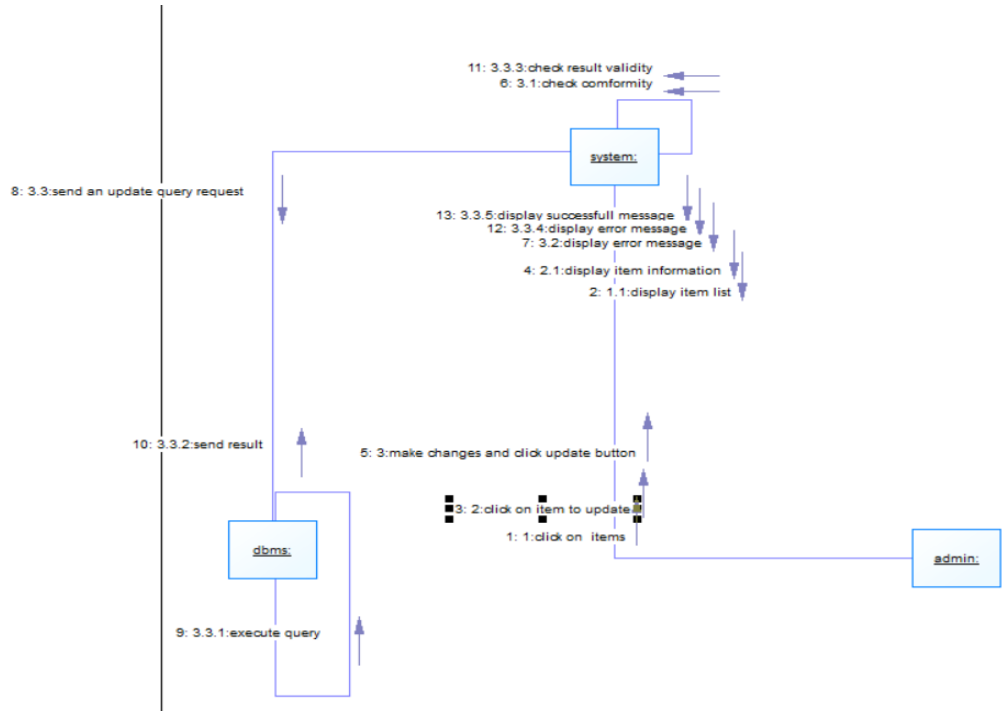


Figure 4.3: UPDATE ITEM COMMUNICATION DIAGRAM.

2.5 STATE MACHINE DIAGRAM

A State Machine Diagram in UML represents the different states an object can be in and how it transitions between those states based on events. It is useful for modeling behavioral

changes in a system.

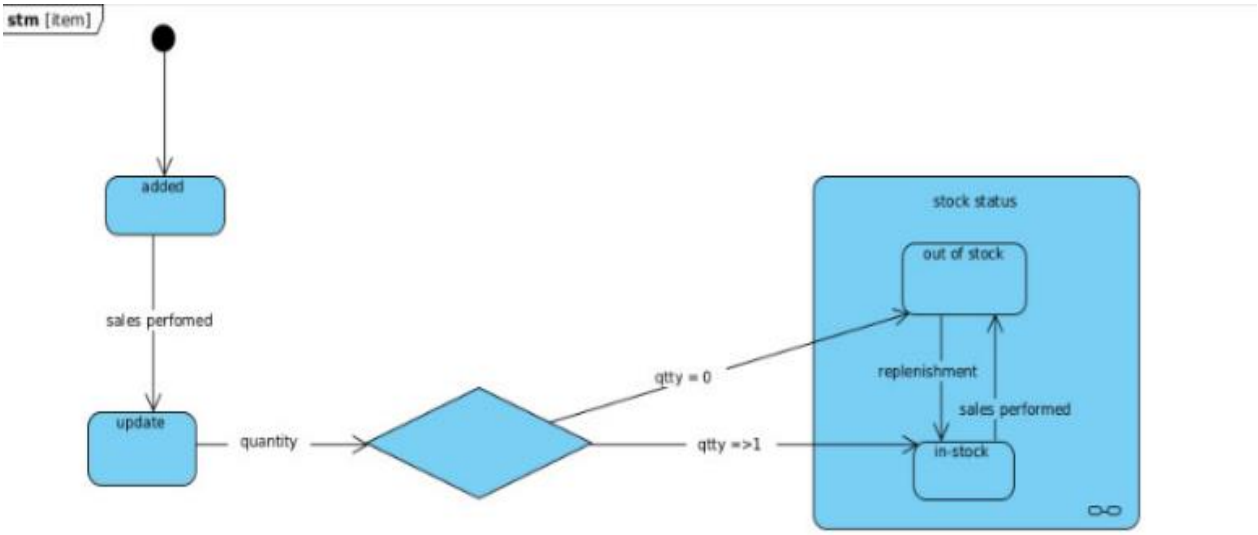


Figure 5: STATE MACHINE DIAGRAM FOR THE OBJECT ITEM.

2.6 CLASS DIAGRAM

A Class Diagram in UML represents the static structure of a system by showing its classes, attributes, methods, and relationships between them. It is essential for object-oriented design and helps in modeling system architecture.

Classes:

- User (Abstract)
- Admin (extends Manager)
- Manager(extends User)
- Sale person(extends User)
- Supplier
- items
- Sales

Relationships:

- Admin and manager interact with Stock and Sales.
- Supplier interacts with items .
- Sale person interact with sales

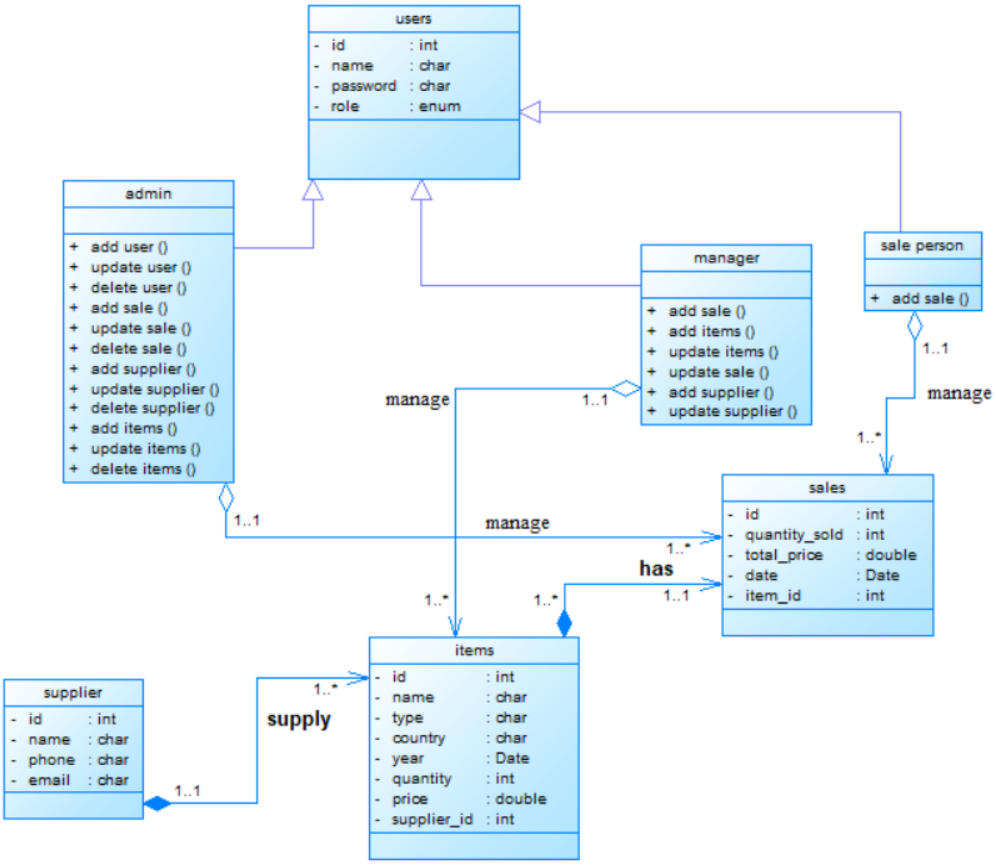


Figure6: CLASS DIAGRAM.

2.7 OBJECT DIAGRAM

An Object Diagram in UML represents a snapshot of a system at a specific point in time, showing instances of classes and their relationships. It is similar to a Class Diagram, but instead of defining the structure, it shows real-world examples of objects with actual values.

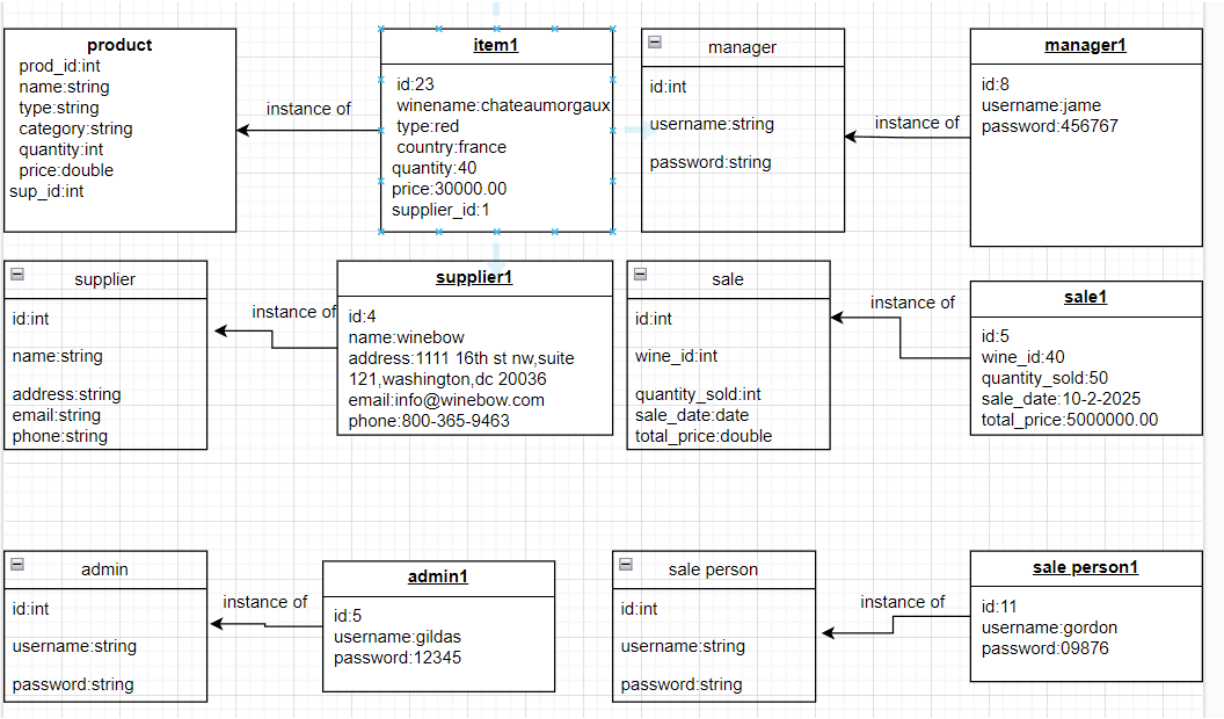


Figure7: OBJECT DIAGRAM.

2.8 PACKAGE DIAGRAM

A Package Diagram in UML is used to organize and group related elements in a system, such as classes, use cases, or components. It helps manage complex projects by structuring them into logical modules.

Packages:

- ❖ User Management (Admin, inventory manager, sale person)
- ❖ Stock Management (Item)
- ❖ Sales Management (Sales)
- ❖ Supplier Management (Supplier)
- ❖ Authentication Management(authentication)

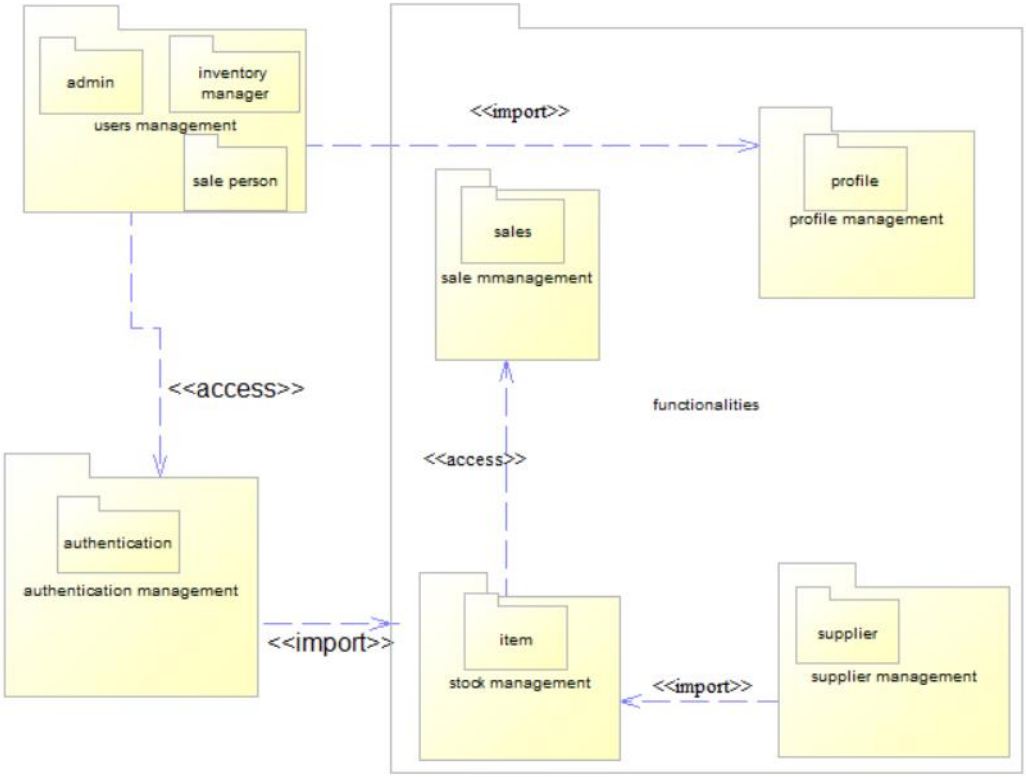


Figure 8: PACKAGE DIAGRAM.

2.9 COMPONENT DIAGRAM

A Component Diagram in UML represents the physical structure of a system by showing how software components interact. It helps in designing modular, scalable, and maintainable systems.

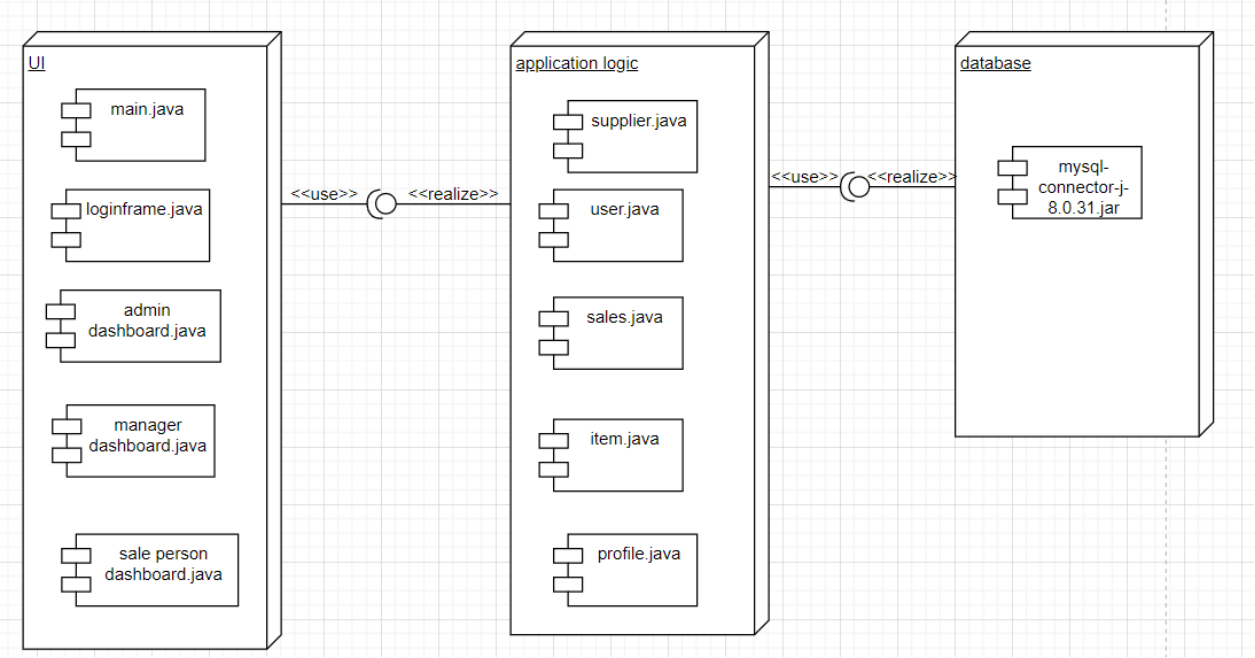


Figure 9: COMPONENT DIAGRAM

.

2.10 DEPLOYMENT DIAGRAM

A Deployment Diagram in UML represents the physical architecture of a system, showing how software components are deployed on hardware nodes. It helps in understanding the infrastructure and execution environment of an application

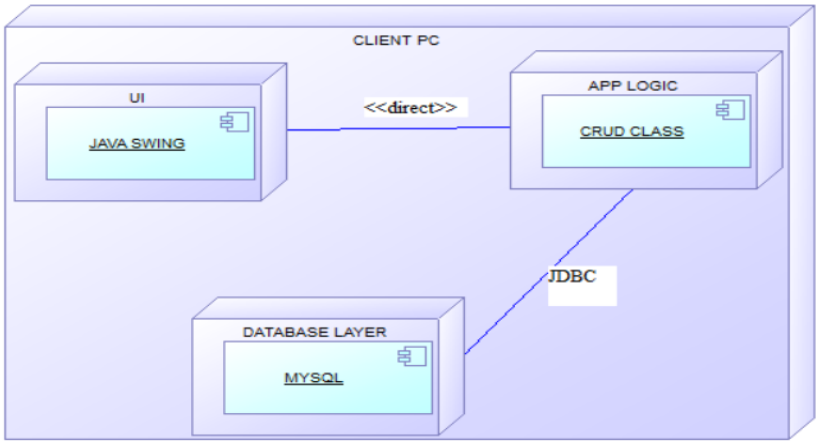


Figure 10: DEPLOYMENT DIAGRAM

3 DATABASE SCHEMA

The database consists of the following tables:

a. **Users Table**: Stores information about Admins , manager and sale person.

```sql
DROP TABLE IF EXISTS `users`;
CREATE TABLE `users` (
  `UserId` int NOT NULL AUTO_INCREMENT,
  `Username` varchar(50) NOT NULL,
  `Password` varchar(255) NOT NULL,
  `Role` enum('admin','manager','saleperson') DEFAULT 'manager',
  PRIMARY KEY (`UserId`),
  UNIQUE KEY `Username` (`Username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO `users` (`UserId`, `Username`, `Password`, `Role`) VALUES
(1, 'Fomekong', '237',  'admin'),
(2, 'Soh Laurie', '123',  'admin'),
(3, 'Doloick',  '234',  'manager'),
(4, 'Malcom', '456',  'saleperson');
```

b. **Suppliers Table**: Stores information about Suppliers.

```sql
DROP TABLE IF EXISTS `suppliers`;
CREATE TABLE `suppliers` (
  `SupId` int NOT NULL AUTO_INCREMENT,
  `Name` varchar(100) NOT NULL,
  `Email` varchar(100) DEFAULT NULL,
  `Phone` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`SupId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO `suppliers` (`SupId`, `Name`, `Email`, `Phone`) VALUES
(1, 'French wines Ltd', 'jeandupont@fftx.com',  '+331 23 45 67 89'),
(2, 'Australian Wine imports',  'markthompson@awi.com', '+61 2 7839 5432'),
(3, 'Italian wine masters ',  'luigirossi@iwm.it',  '+39 055 123 4567'),
(4, 'Napa Valley Distributors', 'emilycarter@nvd.com',  '+1 707 987 6543'),
(5, 'Spanish Vino Co',  'carlosfernandez@svc.com',  '+34 91 345 6789');
```

c. **Item Table**: Tracks the quantity of items in stock.

```sql
DROP TABLE IF EXISTS `sales`;
CREATE TABLE `sales` (
  `SalesId` int NOT NULL AUTO_INCREMENT,
  `Wine_id` int DEFAULT NULL,
  `Quantity_sold` int NOT NULL,
  `Sale_date` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  `Total_price` int DEFAULT NULL,
  `StaffId` int DEFAULT NULL,
  PRIMARY KEY (`SalesId`),
  KEY `Wine_id` (`Wine_id`),
  KEY `StaffId` (`StaffId`),
  CONSTRAINT `sales_ibfk_1` FOREIGN KEY (`Wine_id`) REFERENCES `item` (`Id`) ON DELETE CASCADE,
  CONSTRAINT `sales_ibfk_2` FOREIGN KEY (`StaffId`) REFERENCES `users` (`UserId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO `sales` (`SalesId`, `Wine_id`, `Quantity_sold`, `Sale_date`, `Total_price`, `StaffId`) VALUES
(1, 1,  5,  '2024-02-01 13:30:00',  6000, 2),
(2, 3,  2,  '2024-01-15 09:00:00',  22000,  3),
(3, 5,  3,  '2025-02-13 22:19:22',  55000,  4),
(4, 6,  1,  '2025-02-13 22:20:25',  100000, 4);
```

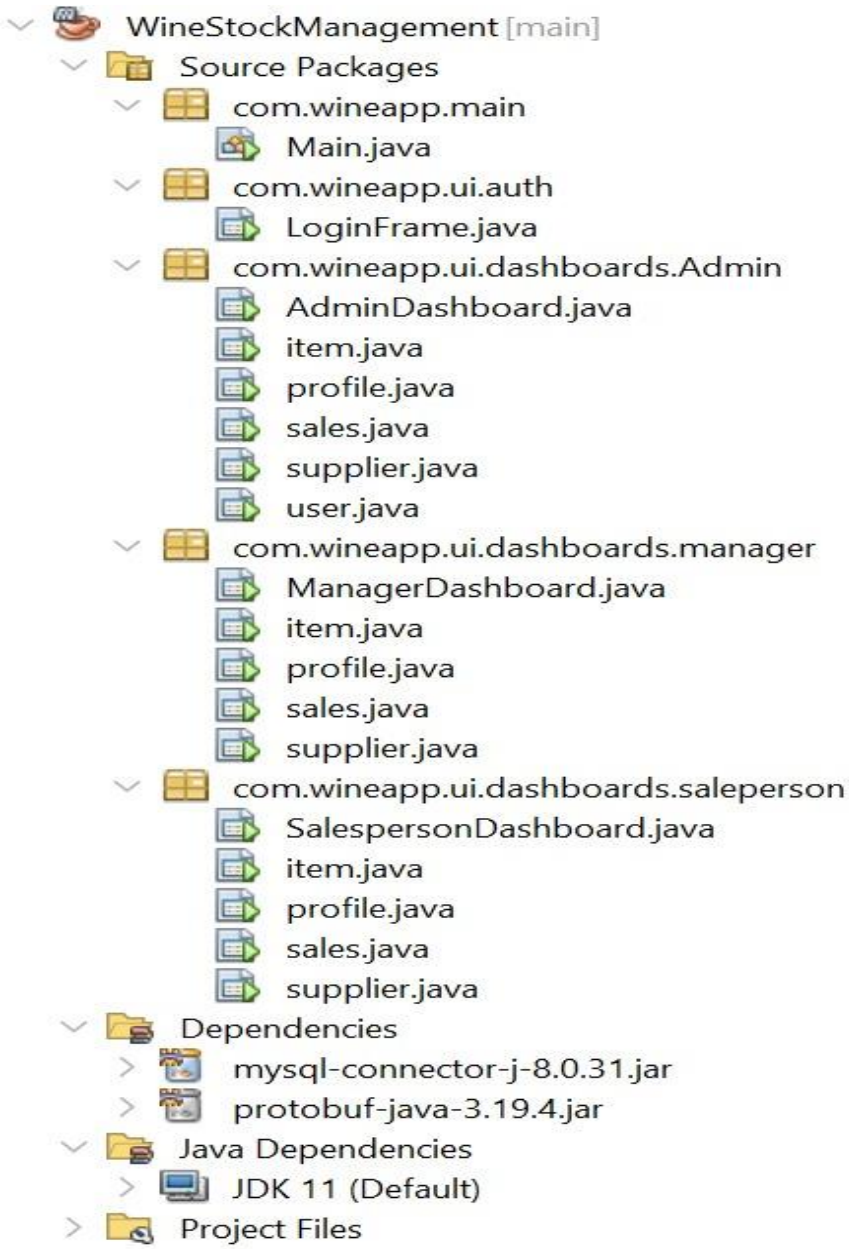
d. **Sales Table**: Records sales transactions.

```sql
DROP TABLE IF EXISTS `item`;
CREATE TABLE `item` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Name` varchar(100) NOT NULL,
  `Type` varchar(50) DEFAULT NULL,
  `Country` varchar(50) DEFAULT NULL,
  `Year` int DEFAULT NULL,
  `Quantity` int NOT NULL,
  `Price` int DEFAULT NULL,
  `supID` int DEFAULT NULL,
  PRIMARY KEY (`Id`),
  KEY `fk_supID` (`supID`),
  CONSTRAINT `fk_supID` FOREIGN KEY (`supID`) REFERENCES `suppliers` (`SupId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO `item` (`Id`, `Name`, `Type`, `Country`, `Year`, `Quantity`, `Price`, `supID`) VALUES
(1, 'Chateau Margaux',  'Red',  'France', 2017, 50, 12000,  1),
(2, 'Penfolds Grange',  'Red',  'Australia',  2020, 30, 7500, 2),
(3, 'Tignanello', 'Red',  'Italy',  2019, 12, 15000,  3),
(4, 'Beringer Chardonnay',  'White',  'USA',  2019, 16, 30000,  4),
(5, 'CloudyBay Sauvignon Blanc',  'White',  'New Zealand',  2021, 45, 5500, 4),
(6, 'Vega Silica Unico',  'Red',  'Spain',  2010, 15, 60000,  5);
```

4 JAVA CODE IMPLEMENTATION

The system is implemented in Java, with classes for each entity (e.g., User, Admin, manager, Supplier, item, Sales). The Database class handles database connections, and DAO (Data Access Object) classes manage CRUD operations.The folder path is as shown below:



The user.java, item.java, supplier.java, sales.java and profile.java are management panels which uses basic CRUD functionalities. The respective UI interface for the dashboards are in AdminDashboard.java, ManagerDashboard.java and SalespersonDashboard.java, the UI for authentication is in loginFrame.java. More details can be clearly seen in the source code inside our WineStockManagement GitHub Repository

5 CONCLUSION

The Wine Stock Management System is a crucial tool for efficiently managing wine inventory, ensuring accurate stock tracking, and optimizing supply chain operations. By implementing this system, businesses can reduce wastage, improve record-keeping, and enhance decision-making regarding stock levels and sales trends.

Through features such as real-time inventory updates, automated alerts, and detailed reports, the system minimizes manual errors and streamlines workflow, leading to increased productivity. Additionally, the ability to track stock movements and analyze sales patterns provides valuable insights for future business strategies.

In conclusion, the Wine Stock Management System enhances operational efficiency, improves inventory accuracy, and supports better financial management. Its implementation will contribute to the seamless management of wine stocks, ultimately benefiting business growth and customer satisfaction.