



GROUP ASSIGNMENT

CT038-3-2-ODJ

Object Oriented Development with Java

INTAKE CODE: APU2F2409SE / APU2F2409CS(AI) / APD2F2409CS(AI)

LECTURER: TAN LI JUNE

DATE ASSIGNED: 15 OCTOBER 2024

DATE COMPLETED: 20 DECEMBER 2024

GROUP 29 MEMBERS:

DARYL SIM WEI SHERN (TP068964)

HO SHANE FOONG (TP068496)

JAEDEN LOONG DENG ZE (TP068347)

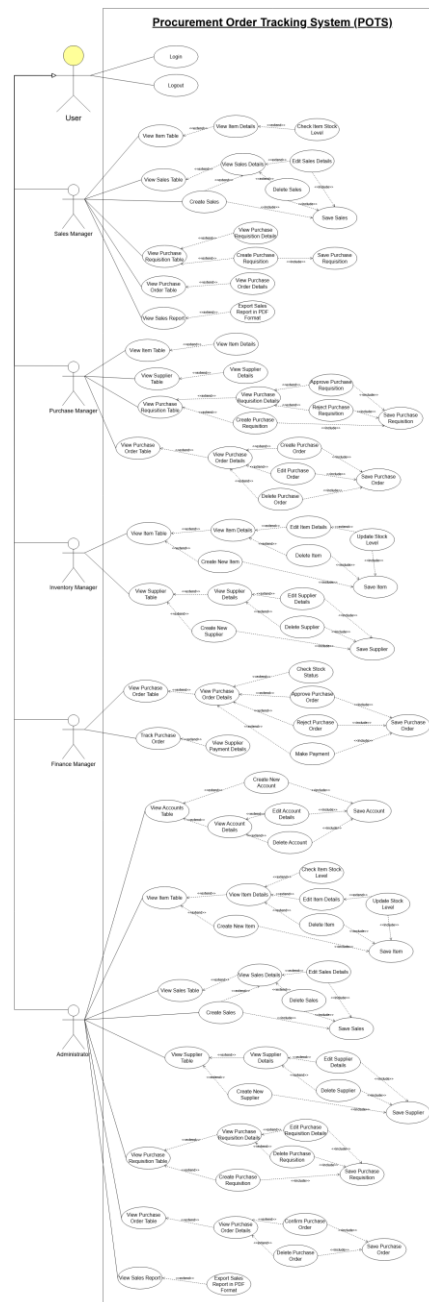
KEITH LO ZE HUI (TP067653)

LIM WEN YI (TP067930)

Table of Contents

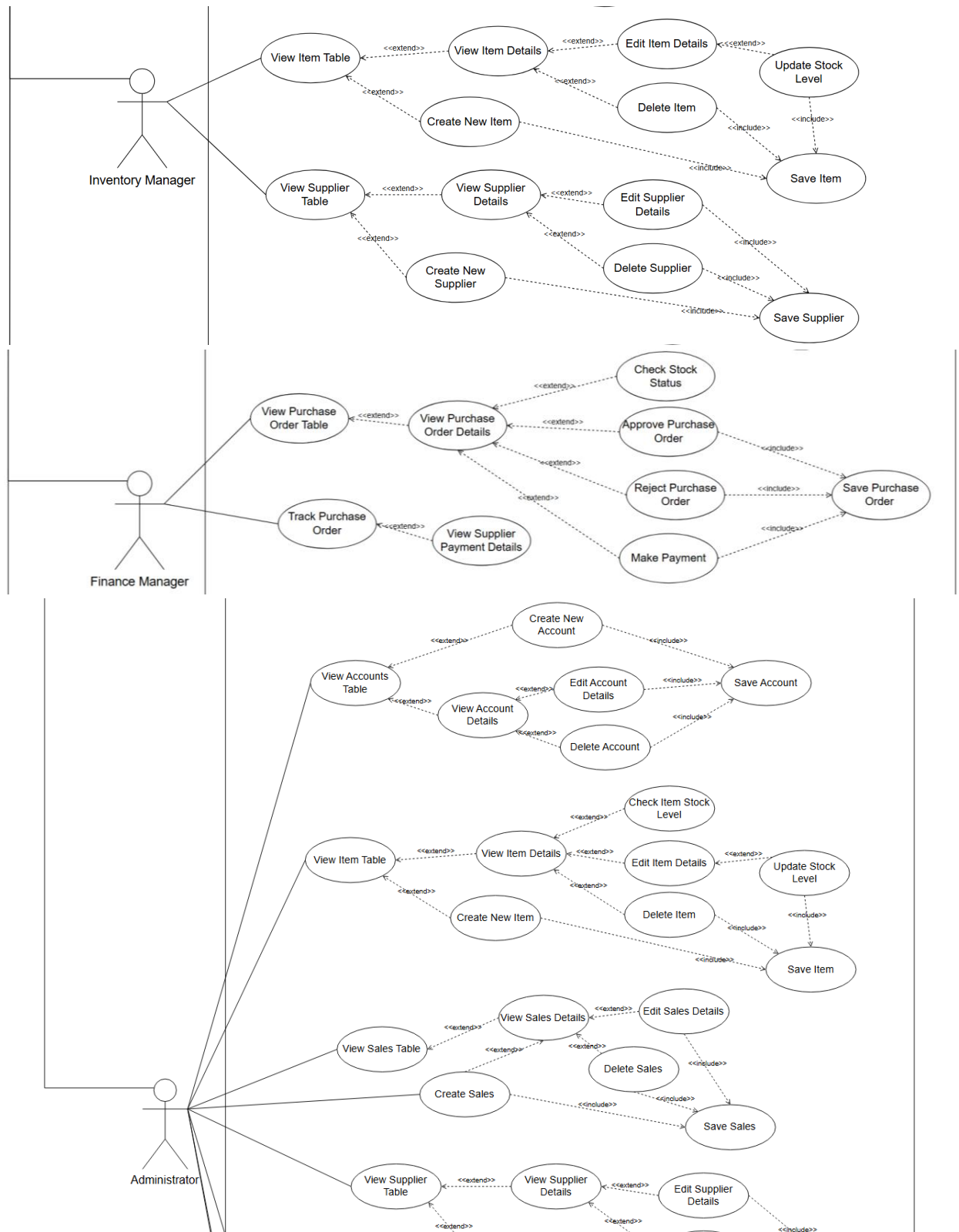
1.0 Design Solution.....	4
1.1 Use Case Diagram.....	4
1.1.1 Use Case Specification	8
1.2 Class Diagram	14
2.0 Output of the Program	18
2.1 Sales Manager	18
2.1.1 Edit Items Table.....	19
2.1.2 Edit Sales Table	20
2.1.3 Edit purchase Requisition Table.....	21
2.1.4 Edit Purchase Order Table.....	22
2.1.5 Create Sales	23
2.1.6 Sales Report.....	23
2.1.7 View Restock Items.....	23
2.2 Purchase Manager	24
2.2.1 Edit Items Table.....	24
2.2.2 Edit Suppliers Table	26
2.2.3 Edit Purchase Requisition Table.....	28
2.2.4 Edit Purchase Order Table.....	33
2.3 Inventory Manager	38
2.3.1 Edit Items table	39
2.3.2 Edit Suppliers Table	40
2.4 Finance Manager	41
2.4.1 Edit Purchase Order Table.....	41
2.4.2 Track Purchase Order	46
2.5 Administrator	48
2.5.1 View Accounts Table	48
2.5.2 View Account Details.....	50
2.5.3 Account Addition	50
2.5.4 Account Editing.....	53
2.5.5 Account Deletion.....	59

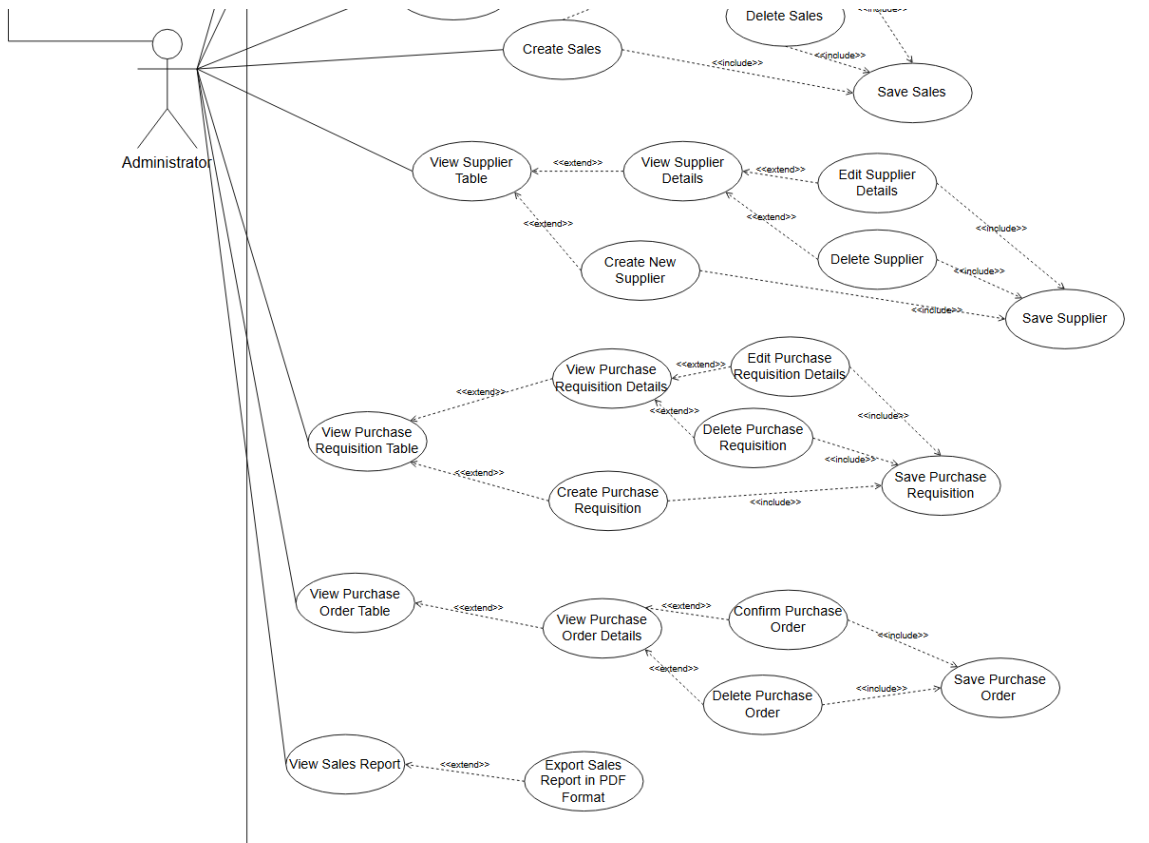
3.0 Object-Oriented Programming Concepts.....	63
3.1 Encapsulation	63
3.2 Modularity	64
3.3 Inheritance	66
3.4 Polymorphism	70
3.4.1 Overriding.....	70
3.4.2 Overloading	73
3.5 Abstraction	74
4.0 Additional Features	79
4.1 HashMaps and UUIDs	79
4.2 Sales Report PDF Export	79
5.0 Limitations	80
6.0 Conclusion	81
7.0 References	82
8.0 Workload Matrix.....	83



Procurement Order Tracking System (POTS)







1.1.1 Use Case Specification

1. Use Case: Login

Actors: User

Description: User accesses the system by entering their account details

Preconditions: User has a registered account and enters valid account details

Postconditions: User is logged into account and is redirected to main page

2. Use Case: Logout

Actors: User

Description: User exits the system

Preconditions: User has logged into account

Postconditions: User has logged out of account and has exited the main page

3. Use Case: View Item Table

Actors: Sales Manager, Purchase Manager, Inventory Manager, Administrator

Description: System displays all items in the inventory on a table

Preconditions: User is logged in and has suitable permissions

Postconditions: Table of items is displayed

Extensions: View Item Details

4. Use Case: View Item Details

Actors: Sales Manager, Purchase Manager, Inventory Manager, Administrator

Description: Details of selected item are displayed including item ID, item name, supplier, and price.

Preconditions: Selected item exists in inventory

Postconditions: Details of selected item are displayed

Extensions: Check Item Stock Level, Edit Item Details, Delete Item

5. Use Case: Check Item Stock Level

Actors: Sales Manager, Inventory Manager, Administrator

Description: Current stock level and minimum stock of selected item are displayed

Preconditions: Selected item exists in inventory

Postconditions: Stock level and minimum stock of selected item are displayed

Extensions: Update Stock Level

6. Use Case: Edit Item Details

Actors: Inventory Manager, Administrator

Description: User can edit details of a selected item including item ID, item name, supplier, and price.

Preconditions: Selected item exists in inventory

Postconditions: Edits to selected items are saved and updated.

7. Use Case: Delete Item

Actors: Inventory Manager, Administrator

Description: Selected item is deleted from system

Preconditions: Selected item exists

Postconditions: Selected item is deleted

8. Use Case: Create New Item

Actors: Inventory Manager, Administrator

Description: A new item can be created

Preconditions: User is logged in and has appropriate permissions

Postconditions: New item is created

Extensions: Save Item

9. Use Case: Save Item

Actors: Inventory Manager, Administrator

Description: New and updated items are saved into the system

Preconditions: Item details are complete and valid

Postconditions: New item is saved

10. Use Case: Update Stock Level

Actors: Inventory Manager

Description: User can edit the stock level of a selected item

Preconditions: Selected item exists in inventory

Postconditions: Edits to selected items' stock level are saved and updated.

11. Use Case: View Sales Table

Actors: Sales Manager, Administrator

Description: System displays all sales records on a table

Preconditions: User is logged in and has suitable permissions

Postconditions: Table of sales records is displayed

Extensions: View Sales Details

12. Use Case: View Sales Details

Actors: Sales Manager, Administrator

Description: Details of selected sales record are displayed including item name, item stock, quantity sold, person in charge.

Preconditions: Selected sales record exists

Postconditions: Details of sales record are displayed

Extensions: Edit Sales Details, Delete Sales

13. Use Case: Edit Sales Details

Actors: Sales Manager, Administrator

Description: User can edit details of a selected sales record including item name, item stock, quantity sold, person in charge.

Preconditions: Selected sales record exists

Postconditions: Edits to selected sales record are saved and updated.

14. Use Case: Save Sales

Actors: Sales Manager, Administrator

Description: New and updated sales records are saved into the system

Preconditions: Sales details are complete and valid

Postconditions: Sales record is saved

15. Use Case: Delete Sales

Actors: Sales Manager, Administrator

Description: Selected sales record is deleted from system

Preconditions: Selected sales record exists

Postconditions: Selected sales record is deleted

16. Use Case: View Sales Report

Actors: Sales Manager, Administrator

Description: A report consisting of a table displaying week, quantity sold, and total revenue. A report summary is also displayed.

Preconditions: User is logged in and has appropriate permissions. Sales data exists

Postconditions: Sales report is displayed

Extensions: Export Sales Report in PDF format

17. Use Case: Export Sales Report in PDF format

Actors: Sales Manager, Administrator

Description: Sales report is exported as a PDF file

Preconditions: Sales data exists

Postconditions: Sales report is exported as a PDF file

18. Use Case: View Purchase Requisition

Actors: Sales Manager, Purchase Manager, Administrator

Description: All purchase requisitions are displayed on a table

Preconditions: User is logged in and has appropriate permissions

Postconditions: Table containing purchase requisitions is displayed

Extensions: View Purchase Requisition Details, Create Purchase Requisition,

19. Use Case: View Purchase Requisition Details

Actors: Sales Manager, Purchase Manager, Administrator

Description: Details of selected purchase requisition are displayed including item name, stock, minimum stock, item ordered, and status.

Preconditions: Selected purchase requisition exists

Postconditions: Details of selected purchase requisition are displayed

Extensions: Approve Purchase Requisition, Reject Purchase Requisition

20. Use Case: Approve Purchase Requisition

Actors: Purchase Manager, Administrator

Description: User can approve purchase requisition for further processing

Preconditions: Selected purchase requisition exists

Postconditions: Purchase requisition is approved for further processing

21. Use Case: Reject Purchase Requisition

Actors: Purchase Manager, Administrator

Description: User can reject purchase requisition

Preconditions: Selected purchase requisition exists

Postconditions: Purchase requisition is rejected

22. Use Case: Create Purchase Requisition

Actors: Sales Manager, Purchase Manager, Administrator

Description: A new purchase requisition can be created

Preconditions: User is logged in and has appropriate permissions

Postconditions: New purchase requisition is created

Extensions: Save Purchase Requisition

23. Use Case: Save Sales

Actors: Sales Manager, Purchase Manager Administrator

Description: New and updated purchase requisitions are saved into the system

Preconditions: Purchase requisition details are complete and valid

Postconditions: Purchase requisition is saved

24. Use Case: Track Purchase Order

Actors: Finance Manager, Administrator

Description: All purchase orders are displayed on a table

Preconditions: User is logged in and has appropriate permissions

Postconditions: Table containing purchase orders is displayed

Extensions: View Supplier Payment Details

25. Use Case: View Supplier Payment Details

Actors: Finance Manager, Administrator

Description: Details of selected purchase order are displayed including item name, supplier, purchase requisition status, purchase order status, current stock and ordered stock

Preconditions: Selected purchase order exists

Postconditions: Purchase order details are displayed

Extensions: Make Payment, Approve Purchase Order, Reject Purchase Order, Check Stock Status

26. Use Case: Make Payment

Actors: Finance Manager, Administrator

Description: User can make payments for purchase order

Preconditions: Purchase order is approved

Postconditions: Payment for purchase order is processed

27. Use Case: Approve Purchase Order

Actors: Finance Manager, Administrator

Description: User can approve purchase orders for further processing

Preconditions: Purchase order exists

Postconditions: Purchase order is approved

28. Use Case: Reject Purchase Order

Actors: Finance Manager, Administrator

Description: User can reject purchase orders

Preconditions: Purchase order exists

Postconditions: Purchase order is rejected

29. Use Case: Check Stock Status

Actors: Finance Manager, Administrator

Description: User can check items stock quantity and minimum stock

Preconditions: Item exists in inventory

Postconditions: Items' stock quantity and minimum stock are displayed

30. Use Case: View Accounts Table

Actors: Administrator

Description: All user accounts are displayed on a table

Preconditions: User is logged in and has appropriate permissions

Postconditions: All user accounts are displayed on a table
Extensions: View Account Details, Create New Account

31. Use Case: View Account Details

Actors: Administrator

Description: Details of selected user account are displayed including username, password, and role

Preconditions: Selected user account exists

Postconditions: All user account details are displayed

Extensions: Edit Account Details, Delete Account

32. Use Case: Edit Account Details

Actors: Administrator

Description: User can edit details of a selected user account including username, password, and role

Preconditions: Selected user account exists

Postconditions: User account details are updated

Extensions: Save Account

33. Use Case: Delete Account

Actors: Administrator

Description: User can delete selected user account from system

Preconditions: Selected user account exists

Postconditions: Selected user account is deleted from system

34. Use Case: Create New Account

Actors: Administrator

Description: User can create a new user account

Preconditions: User is logged in and has appropriate user permissions

Postconditions: New user account is created

Extensions: Save Account

35. Use Case: Save Account

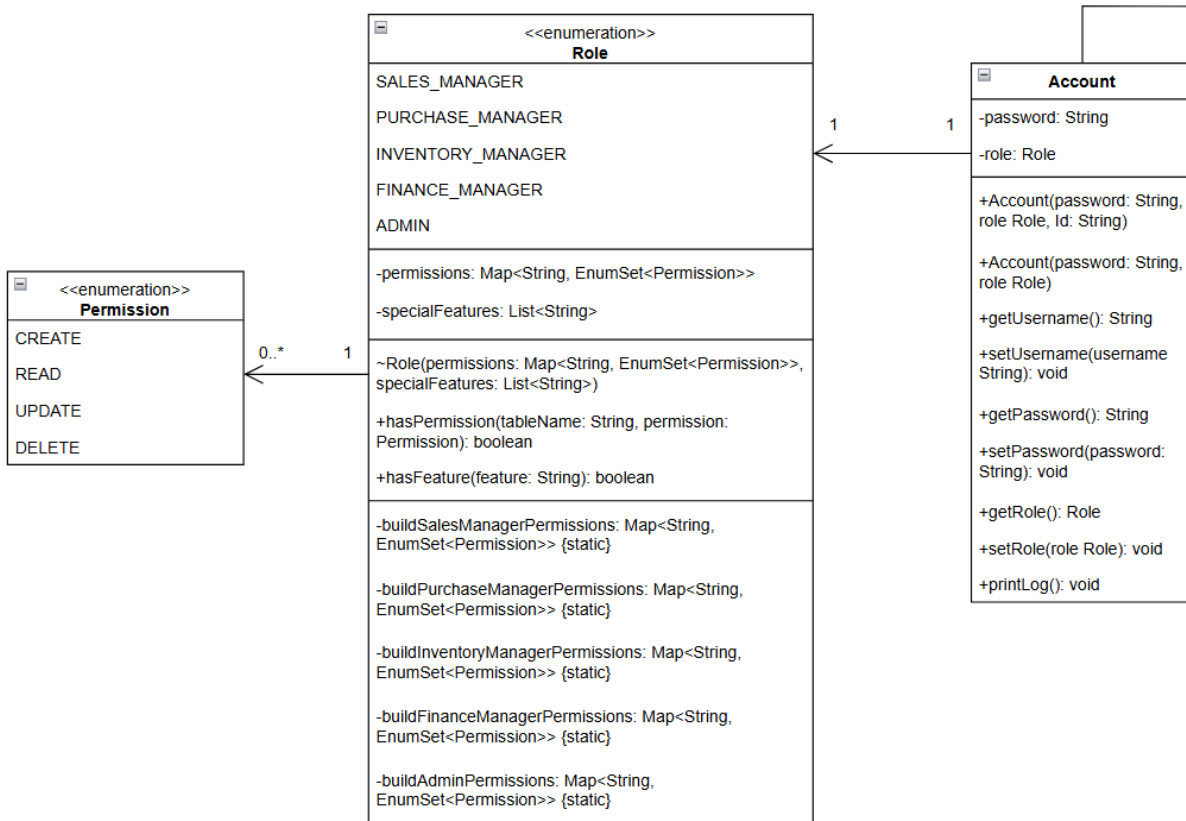
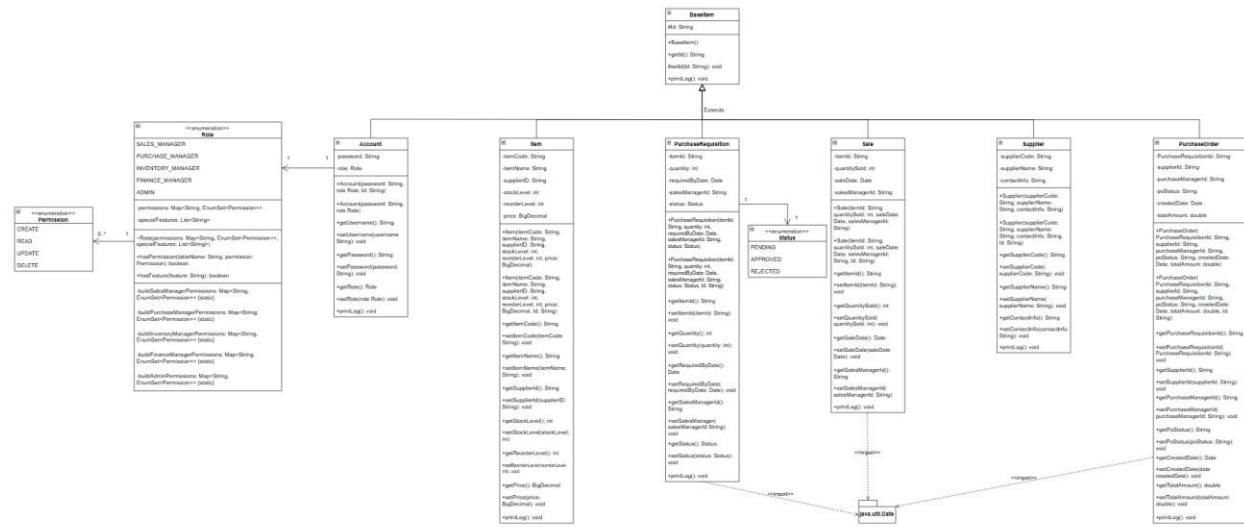
Actors: Administrator

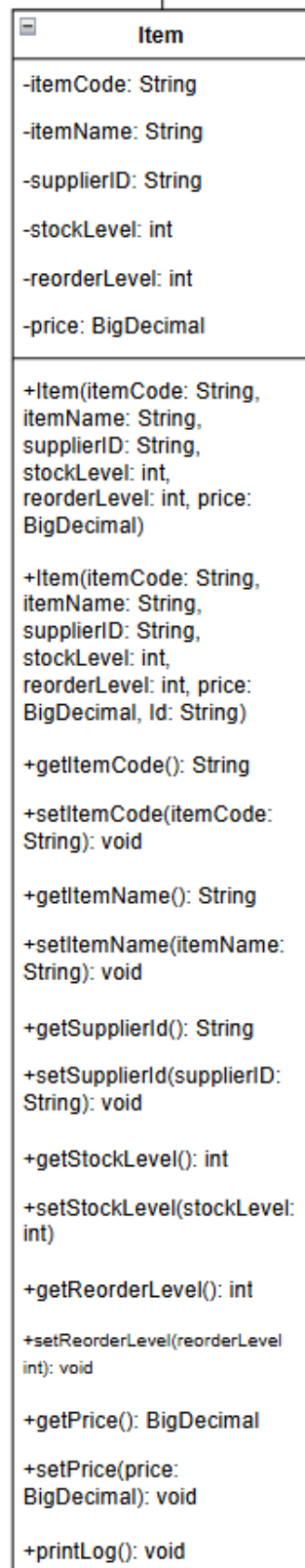
Description: Edited and newly created user accounts can be uploaded into the system

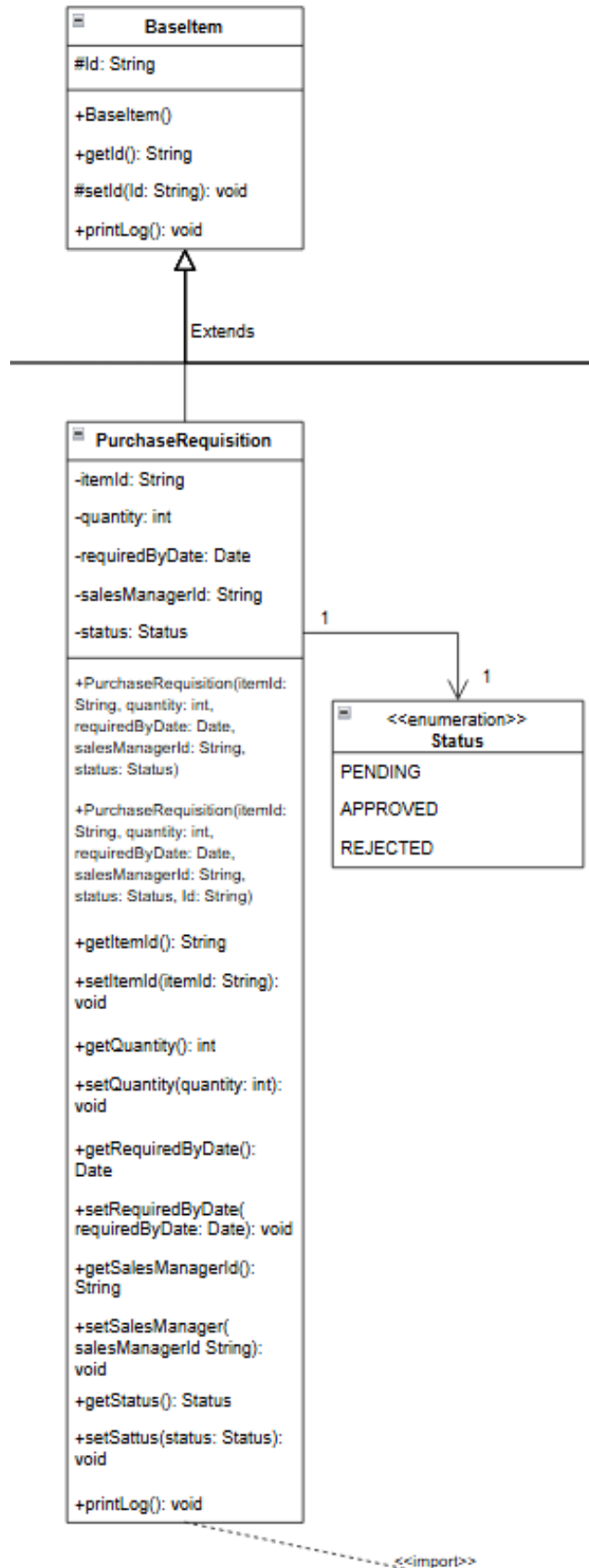
Preconditions: User account details are complete and valid

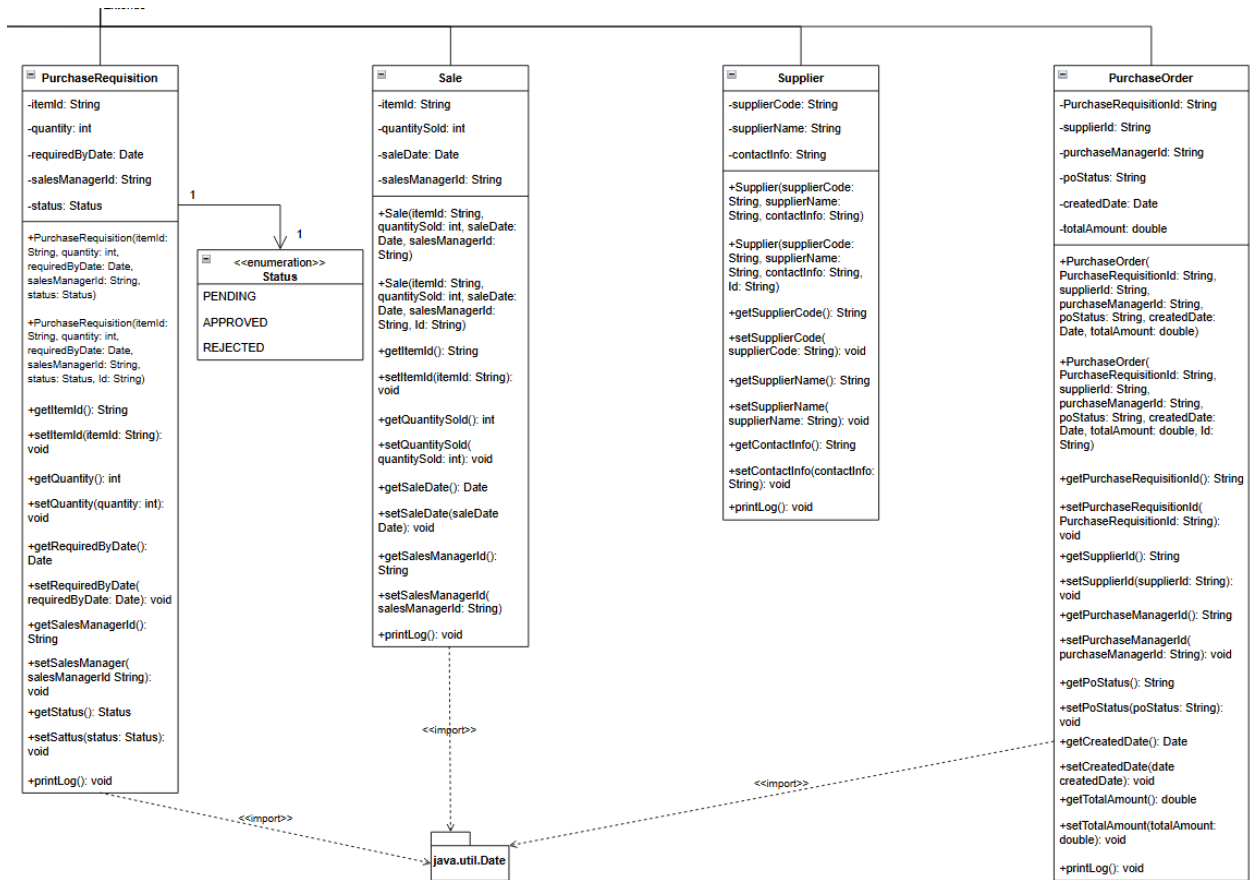
Postconditions: User account is saved into the system

1.2 Class Diagram



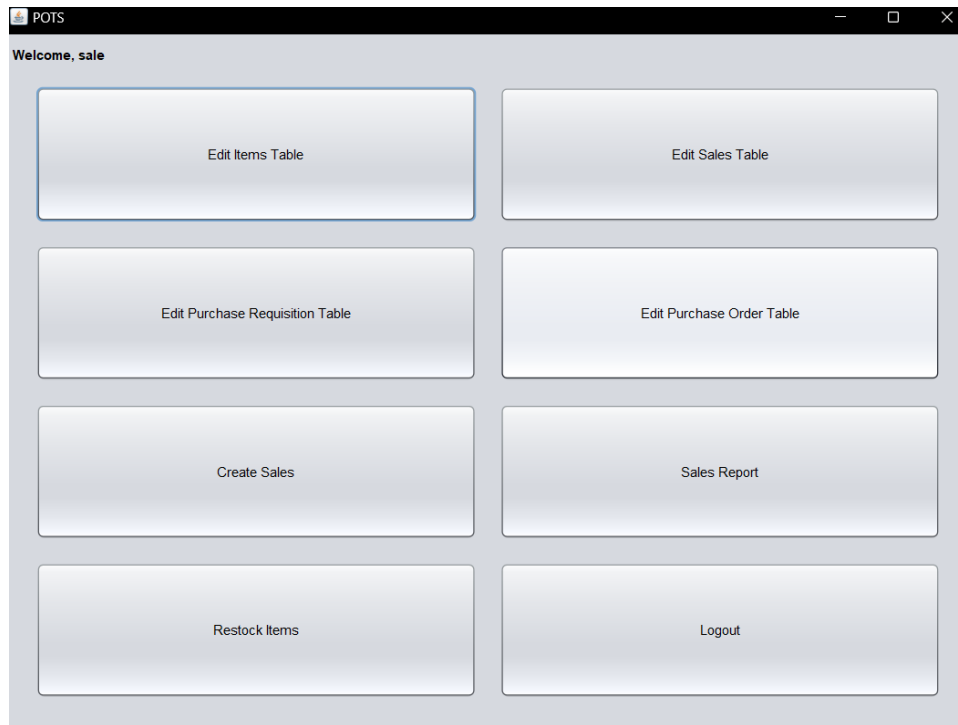




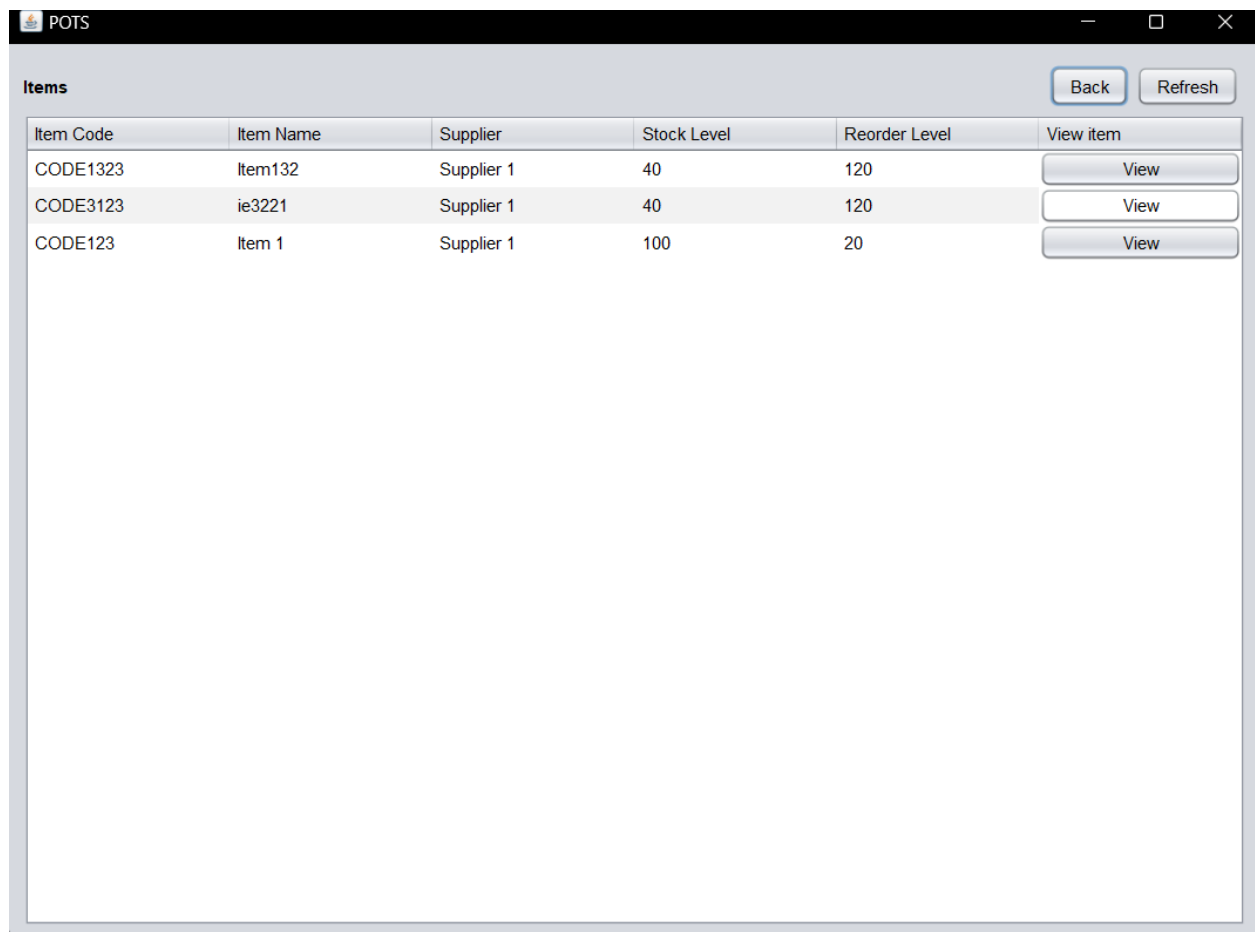


2.0 Output of the Program

2.1 Sales Manager



2.1.1 Edit Items Table



Item Code	Item Name	Supplier	Stock Level	Reorder Level	View item
CODE1323	Item132	Supplier 1	40	120	View
CODE3123	ie3221	Supplier 1	40	120	View
CODE123	Item 1	Supplier 1	100	20	View

In this we can see all the items that are available, and the amount left in stock

2.1.2 Edit Sales Table

Sales

Back

Refresh

Add New

Item	Quantity Sold	Sale Date	Sales Manager	View
Item 1	10	19-12-2024 10:48:17	admin	<div>View</div>

2.1.3 Edit purchase Requisition Table

PurchaseRequisition

Back

Refresh

Add New

Item	Quantity	Required By Date	Sales Manager	Status	View
Item132	50	19-12-2024 10:48:17	admin	PENDING	<div>View</div>
Item 1	50	19-12-2024 10:48:17	admin	PENDING	<div>View</div>

Above the edit purchase requisition table allows the sales manager to view and edit any and all purchase requisitions made.

Purchase Requisition		Back
Item Name	Item 132	
Stock	40	
Min Stock	120	
Item Ordered	50	
Status	PENDING	

And this is where to view with details.

Purchase Requisition

Back

Confirm

Item Name

Item 132

Stock

40

Min Stock

120

Item Ordered

Here we can also add new purchase requisitions.

2.1.4 Edit Purchase Order Table

PurchaseOrder										<div> <div>Back</div> <div>Refresh</div> </div>
Item	Quantity	PR Status	Sales Man...	Required By	Supplier	PO Manag...	PO Status	PO Created	Total Amou...	View
Item 1	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	PENDING	Thu Dec 1...	500.0	View
Item132	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	DELIVERED	Thu Dec 1...	500.0	View

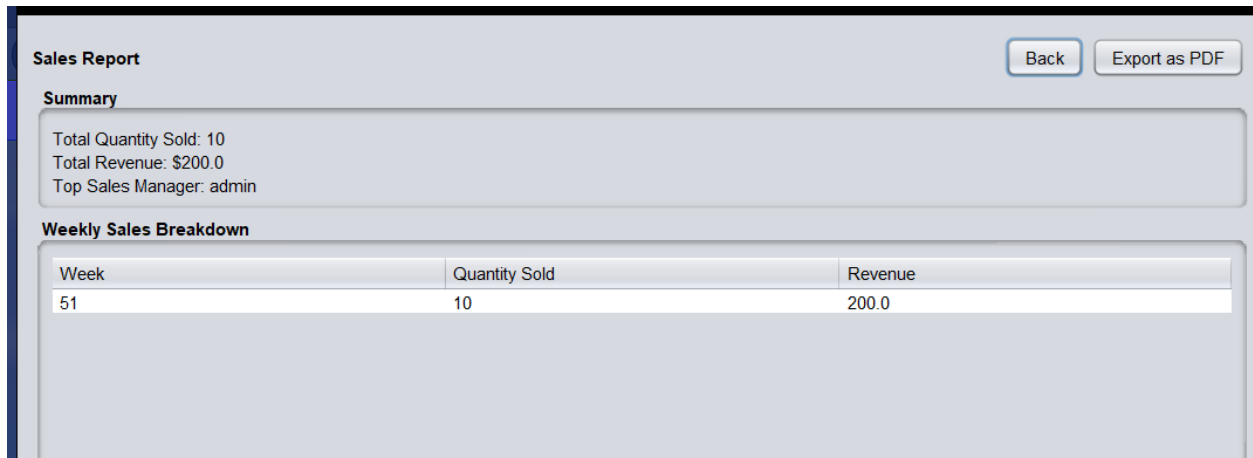
In the edited purchase order table, the sales manager would be able to view all purchase orders and view them in detail.

2.1.5 Create Sales



In the create sales, we the sales manager are able to select the product we want to create the sale, then it would be shown in the " Stock: ". Then the sales manager could type in the sales quantity, if it exceeds the stock quantity, an error message would then be shown

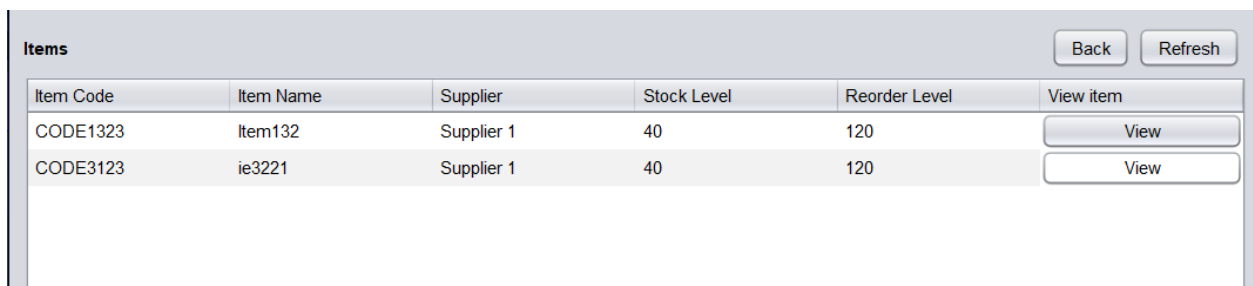
2.1.6 Sales Report



Week	Quantity Sold	Revenue
51	10	200.0

In the sales report page, the sales manager would be able to generate a pdf for the sales made.

2.1.7 View Restock Items



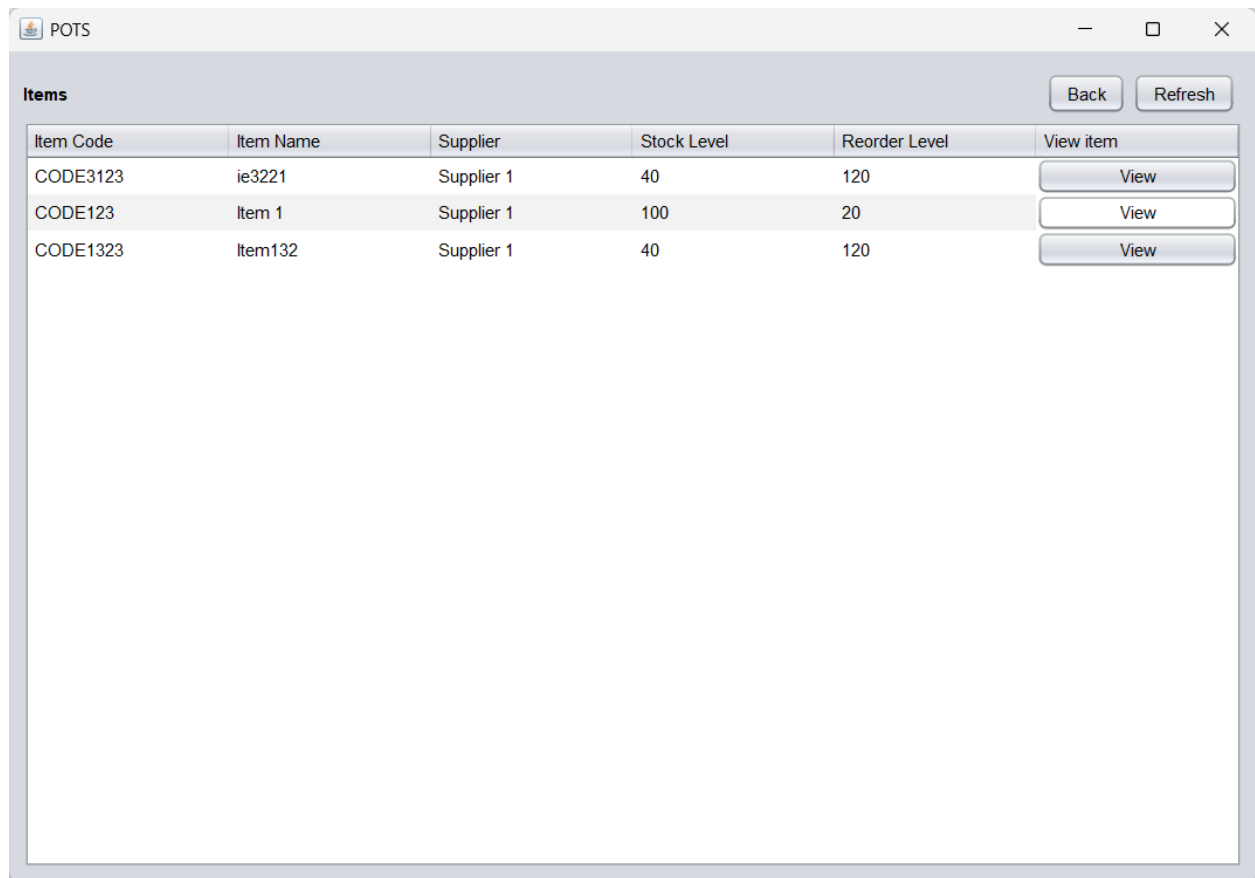
Item Code	Item Name	Supplier	Stock Level	Reorder Level	View item
CODE1323	Item132	Supplier 1	40	120	<button>View</button>
CODE3123	ie3221	Supplier 1	40	120	<button>View</button>

In this page, the sales manager would be able to see the restock requested made and the amount of restock stock ordered.

2.2 Purchase Manager



2.2.1 Edit Items Table



The screenshot shows a window titled "POTS" with standard window controls (minimize, maximize, close). Below the title bar, there is a header area with the word "Items" on the left and two buttons, "Back" and "Refresh", on the right. The main content area contains a table with six columns: "Item Code", "Item Name", "Supplier", "Stock Level", "Reorder Level", and "View item". There are three data rows in the table. Each row has a "View" button in the "View item" column. The table is styled with alternating light and dark gray rows.

Item Code	Item Name	Supplier	Stock Level	Reorder Level	View item
CODE3123	ie3221	Supplier 1	40	120	<button>View</button>
CODE123	Item 1	Supplier 1	100	20	<button>View</button>
CODE1323	Item132	Supplier 1	40	120	<button>View</button>

The purchase manager can view the list of items in the inventory. They can also view the details of each item by clicking on the View button.

POTS

— □ ×

Add New Item

Back

Item ID	CODE123
Item Name	Item 1
Supplier	Supplier 1
Stock	100
Minimum Stock	20
Price	20.00

These are the details of the item.

2.2.2 Edit Suppliers Table

Suppliers			
Code	Name	Contact	View
SUP123	Supplier 1	contact@supplier1.com	View
suppp2321	Supplier 3123121	contact@supplier1.com	View

The purchase manager can view the list of suppliers. They can also view the details of each supplier by clicking on the View button.

The screenshot shows a web browser window with the title 'POTS'. Inside the window, there is a form titled 'Supplier Form'. The form has three input fields, each with a label to its left: 'Supplier Code' containing 'SUP123', 'Supplier Name' containing 'Supplier 1', and 'Contact Info' containing 'contact@supplier1.com'. A 'Back' button is located in the top right corner of the form area.

Field Label	Value
Supplier Code:	SUP123
Supplier Name:	Supplier 1
Contact Info:	contact@supplier1.com

These are the details of the supplier.

2.2.3 Edit Purchase Requisition Table

POTS						—	□	×
PurchaseRequisition						Back	Refresh	Add New
Item	Quantity	Required By Date	Sales Manager	Status	View			
Item 1	50	19-12-2024 11:21:39	admin	PENDING	View			
Item132	50	19-12-2024 11:21:39	admin	PENDING	View			
Item 1	22	19-12-2024 11:28:53	sale	PENDING	View			

The purchase manager can view the pending purchase requisitions created by themselves, sales manager, or other purchase managers. They can also add new purchase requisitions by clicking on the Add New button.

POTS

Purchase Requisition

Back Confirm

Item Name	le3221
Stock	40
Min Stock	120
Item Ordered	

This is the Add New Purchase Requisition form.

POTS


Purchase Requisition

Back

Item Name	Item 1
Stock	100
Min Stock	20
Item Ordered	22
Status	PENDING

Approve Reject

They can view the details of each purchase requisition and decide to approve or reject the purchase requisitions. If they reject the purchase requisition, the purchase requisition will be deleted and removed from the purchase requisition table.

 POTS

—

□

×

Purchase Requisition

Back

Item Name	Item 1
Stock	100
Min Stock	20
Item Ordered	22
Status	APPROVED

POTS					
PurchaseRequisition					
<div>Back Refresh Add New</div>					
Item	Quantity	Required By Date	Sales Manager	Status	View
Item 1	50	19-12-2024 11:21:39	admin	PENDING	View
Item132	50	19-12-2024 11:21:39	admin	PENDING	View
Item 1	22	19-12-2024 11:28:53	sale	APPROVED	View

If the purchase manager approves the purchase requisition, the status will be changed to “APPROVED”.

2.2.4 Edit Purchase Order Table

POTS										
PurchaseOrder										
<div>Back Refresh Add New</div>										
Item	Quantity	PR Status	Sales Man...	Required By	Supplier	PO Manag...	PO Status	PO Created	Total Amou...	View
Item 1	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	PENDING	Thu Dec 1...	500.0	View
Item132	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	DELIVERED	Thu Dec 1...	500.0	View

The purchase manager can view the list of purchase orders in this table. They can also generate new purchase orders by clicking on the Add New button.

POTS

Purchase Order Panel

Back

Confirm

Name Order Amount	Item 1 22 Stocks
Item Name	Item 1
Supplier	Supplier 1
PR Status	APPROVED
Current Stock	100
Order Stock	22
Total	150

This is the Add New Purchase Order form, the purchase manager can enter the total cost of the purchase order.

POTS										
PurchaseOrder										
<div>BackRefreshAdd New</div>										
Item	Quantity	PR Status	Sales Man...	Required By	Supplier	PO Manag...	PO Status	PO Created	Total Amou...	View
Item 1	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	PENDING	Thu Dec 1...	500.0	View
Item 1	22	APPROV...	sale	Thu Dec 1...	Supplier 1	pm	PENDING	Thu Dec 1...	150.0	View
Item132	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	DELIVERED	Thu Dec 1...	500.0	View

POTS

Purchase Order Panel

Back

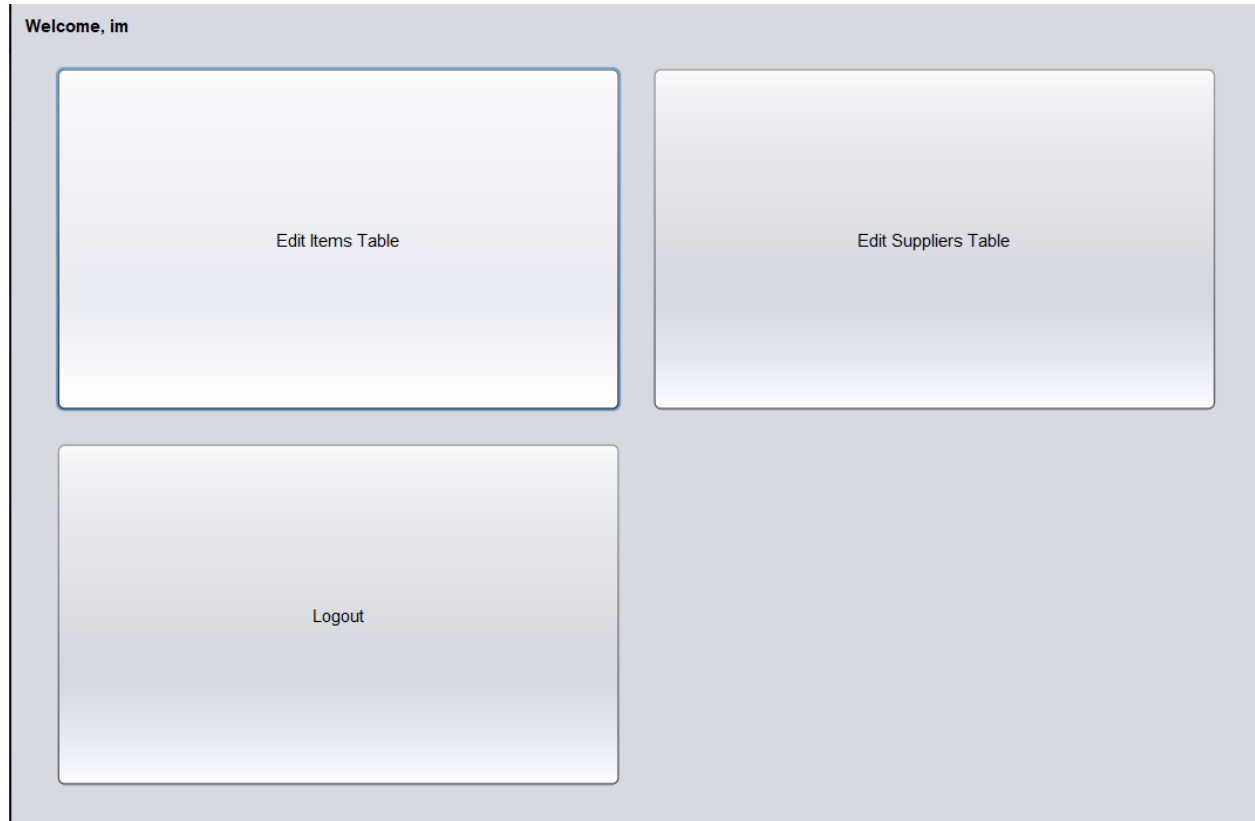
Delete

Edit

Item Name	Item 1
Supplier	Supplier 1
PR Status	PENDING
PO Status	PENDING
Current Stock	100
Order Stock	50
Total	500.0

They can also edit and delete the purchase order.

2.3 Inventory Manager



In the inventory manager page, the inventory manager has access to the edit items table and the edit supplier's table.

2.3.1 Edit Items table

Items

BackRefreshAdd New

Item Code	Item Name	Supplier	Stock Level	Reorder Level	View item
CODE1323	Item132	Supplier 1	40	120	View
CODE3123	ie3221	Supplier 1	40	120	View
CODE123	Item 1	Supplier 1	100	20	View

Add New Item

BackConfirm

Item ID

CODE1323

Item Name

Item132

Supplier

Supplier 1

Stock

40

Minimum Stock

120

Price

10.00

In this section, the inventory manager can view all existing items and also add new items.

2.3.2 Edit Suppliers Table

Suppliers

BackRefreshAdd New

Code	Name	Contact	View
SUP123	Supplier 1	contact@supplier1.com	View
suppp2321	Supplier 3123121	contact@supplier1.com	View

Supplier Form

BackAdd Supplier

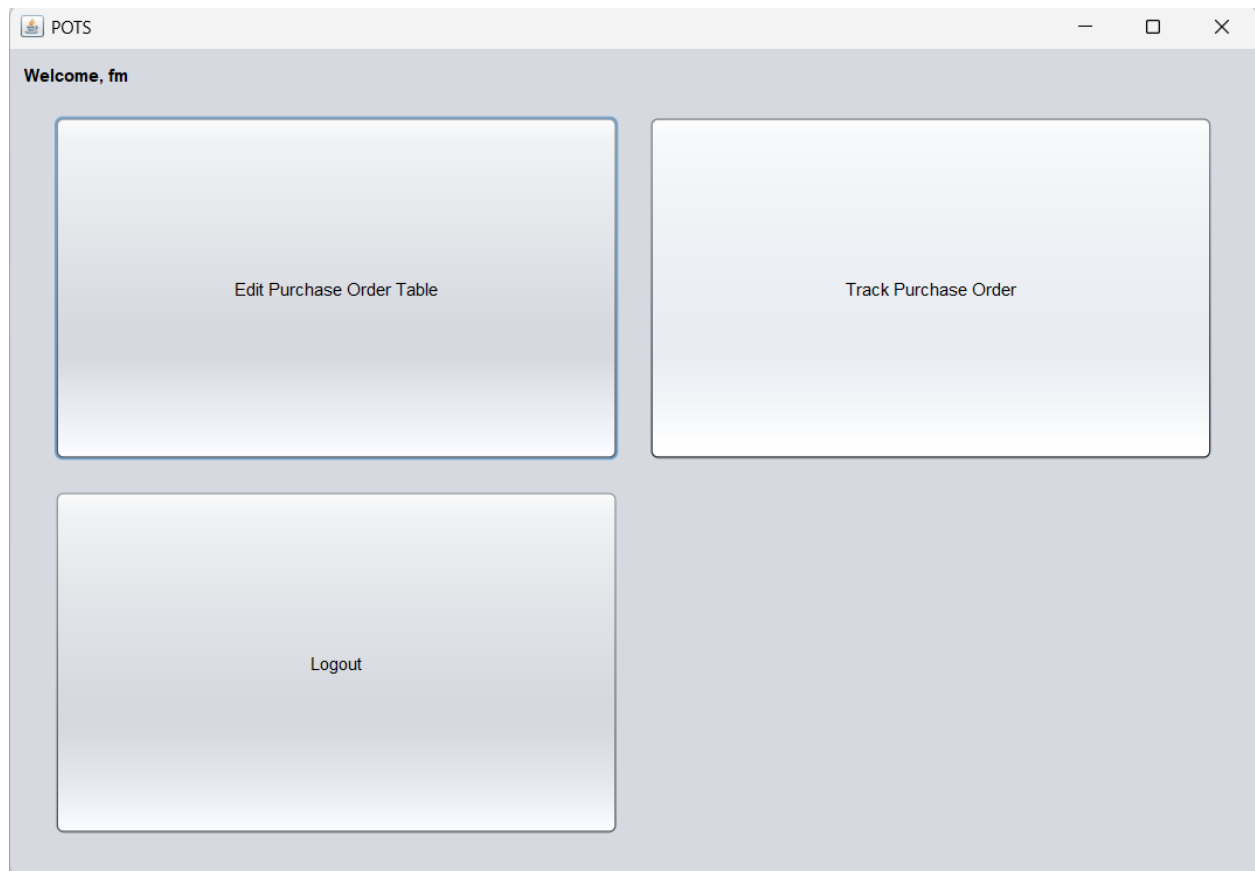
Supplier Code:

Supplier Name:

Contact Info:

In the edit suppliers table, the inventory manager would be able to view all currently existing suppliers. Moreover, the inventory manager could also add new suppliers if new product is ordered from a new supplier.

2.4 Finance Manager




This is the main menu of the finance manager.

2.4.1 Edit Purchase Order Table

POTS										
PurchaseOrder										Back Refresh
Item	Quantity	PR Status	Sales Man...	Required By	Supplier	PO Manag...	PO Status	PO Created	Total Amou...	View
Item 1	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	PENDING	Thu Dec 1...	500.0	View
Item 1	22	APPROV...	sale	Thu Dec 1...	Supplier 1	pm	PENDING	Thu Dec 1...	150.0	View
Item132	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	DELIVERED	Thu Dec 1...	500.0	View

The finance manager can view the purchase order table and view the details of each purchase order by clicking on the View button. For example, they can check the current available stock and order stock.

 POTS

Purchase Order Panel

Back

Item Name	Item 1
Supplier	Supplier 1
PR Status	APPROVED
PO Status	PENDING
Current Stock	100
Order Stock	22

Approve

Reject

They can choose to approve or reject the purchase order. If they reject the purchase order, the purchase order will be deleted and removed from the purchase order table.

POTS

Purchase Order Panel

Back

Pay

Item Name	Item 1
Supplier	Supplier 1
PR Status	APPROVED
PO Status	APPROVED
Current Stock	100
Order Stock	22

If they approve the purchase order, then the Purchase Order (PO) status will be changed to “APPROVED”. Then, the Pay button will appear for the finance manager to make payment.

 POTS

Purchase Order Panel

Back

Item Name	Item 1
Supplier	Supplier 1
PR Status	APPROVED
PO Status	PAID
Current Stock	100
Order Stock	22

POTS										
PurchaseOrder										Back Refresh
Item	Quantity	PR Status	Sales Man...	Required By	Supplier	PO Manag...	PO Status	PO Created	Total Amou...	View
Item 1	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	PENDING	Thu Dec 1...	500.0	View
Item 1	22	APPROV...	sale	Thu Dec 1...	Supplier 1	pm	PAID	Thu Dec 1...	150.0	View
Item132	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	DELIVERED	Thu Dec 1...	500.0	View

After the finance manager made the payment, the Purchase Order status will be changed to “PAID”.

2.4.2 Track Purchase Order

POTS										
PurchaseOrder										Back Refresh
Item	Quantity	PR Status	Sales Man...	Required By	Supplier	PO Manag...	PO Status	PO Created	Total Amou...	View
Item 1	22	APPROV...	sale	Thu Dec 1...	Supplier 1	pm	PAID	Thu Dec 1...	150.0	View
Item132	50	PENDING	admin	Thu Dec 1...	Supplier 1	admin	DELIVERED	Thu Dec 1...	500.0	View

This Track Purchase Order is to allow the finance manager to view the purchase orders with the Purchase Order (PO) status of “DELIVERED” or “PAID”. They can view the details of the purchase order like in Edit Purchase Order Table by clicking the View button.

2.5 Administrator



This is the Administrator main page. The Administrator role has the highest authority and has access to features of all the other user roles (See 2.1 to 2.4) along with its own exclusive features.

2.5.1 View Accounts Table

Accounts			Back	Refresh	Add New
User Name	Role	View			
sale	SALES_MANAGER	View			
im	INVENTORY_MANAGER	View			
fm	FINANCE_MANAGER	View			
admin	ADMIN	View			
pm	PURCHASE_MANAGER	View			

The administrator is able to view the details of all user accounts including the username and role which are displayed on a table.

2.5.2 View Account Details

The screenshot displays a web form titled "Add User" in the top left corner. On the right side of the form, there are three buttons: "Back", "Delete", and "Edit". The form contains three labeled input fields: "Username:" with the value "sale", "Password" with masked characters "****", and "Role:" with a dropdown menu currently showing "Sales Manager".

By pressing the “View” button of an account, the administrator is able to view the details of a specific account. The password is encrypted for security.

2.5.3 Account Addition

By pressing the “Add New” button in the Accounts Table, the administrator is able to add a new account. The administrator must enter the new account’s username, password, and role before creating the account. The new account will be saved once the administrator has pressed the “Add User” button

POTS

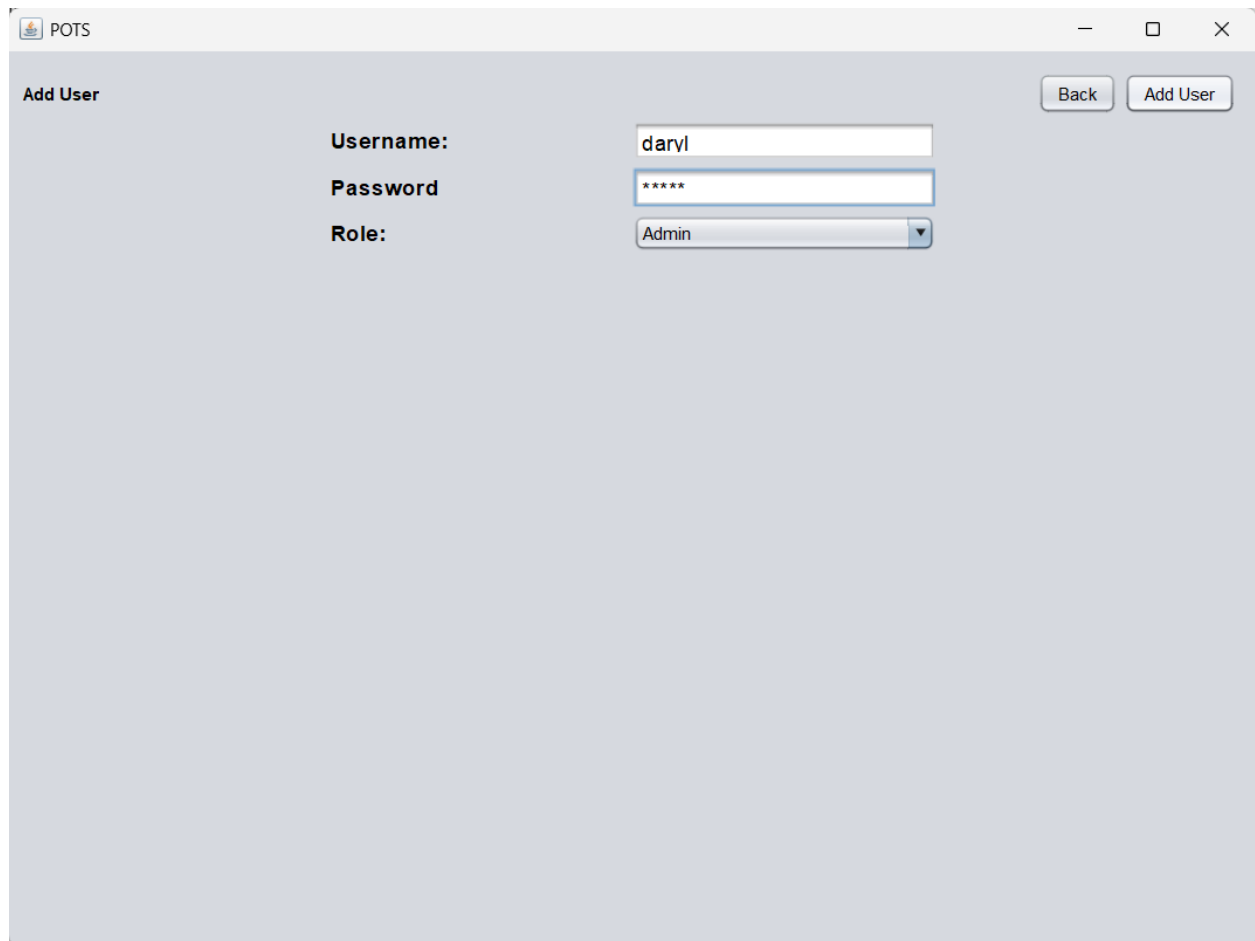
Accounts

Back

Refresh

Add New

User Name	Role	View
sale	SALES_MANAGER	<div>View</div>
im	INVENTORY_MANAGER	<div>View</div>
fm	FINANCE_MANAGER	<div>View</div>
admin	ADMIN	<div>View</div>
pm	PURCHASE_MANAGER	<div>View</div>



The image shows a web application window titled "POTS" with standard window controls (minimize, maximize, close). The main content area is titled "Add User". On the right side of the header, there are two buttons: "Back" and "Add User". The form contains three labeled input fields: "Username:" with the value "daryl", "Password:" with masked characters "*****", and "Role:" with a dropdown menu currently showing "Admin".

Field	Value
Username:	daryl
Password	*****
Role:	Admin

After the new account has been created, it will be displayed on the Accounts table and saved in the accounts.csv file.

POTS

Accounts

Back Refresh Add New

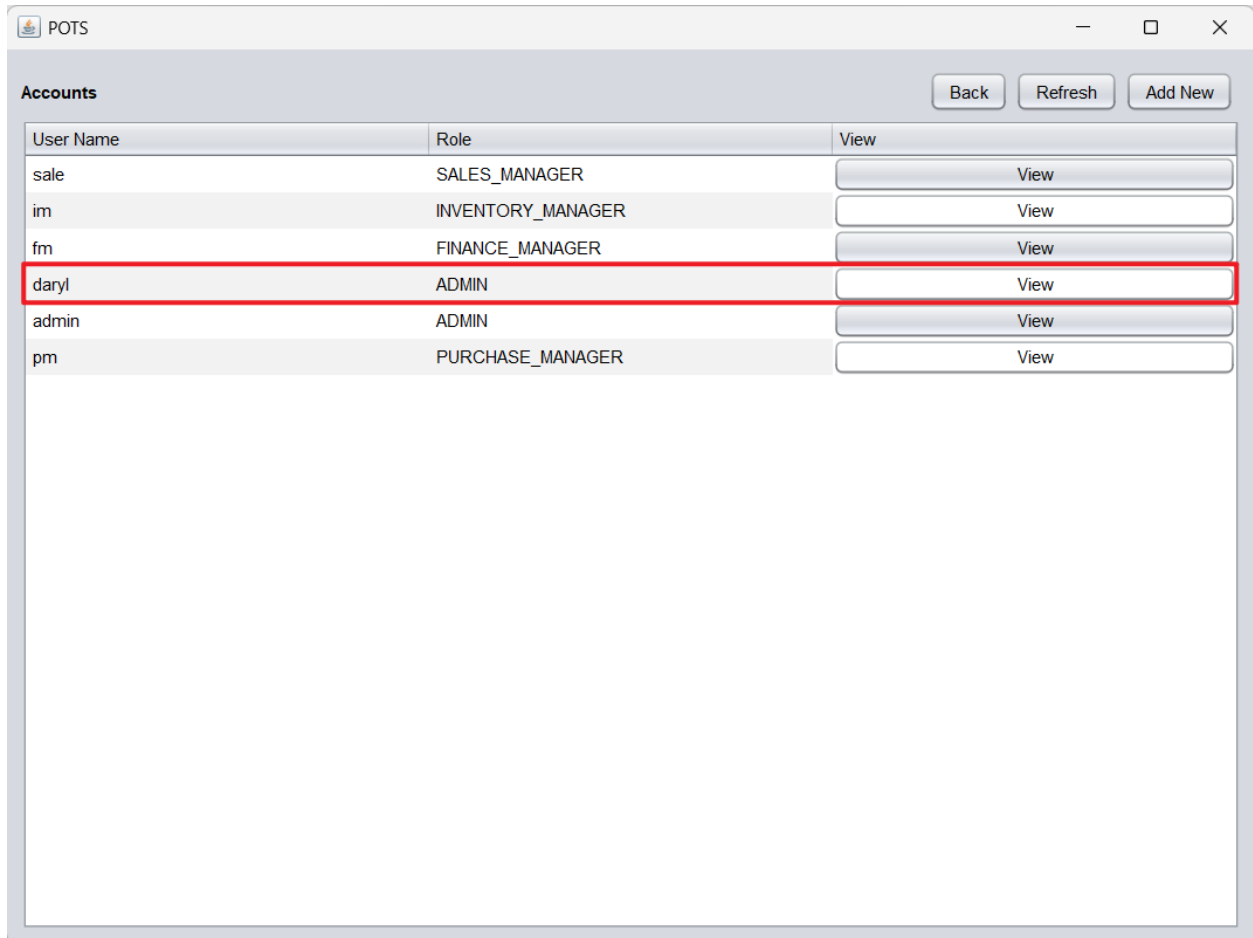
User Name	Role	View
sale	SALES_MANAGER	View
im	INVENTORY_MANAGER	View
fm	FINANCE_MANAGER	View
daryl	ADMIN	View
admin	ADMIN	View
pm	PURCHASE_MANAGER	View

```
data > accounts.csv > data
1 sale,SALES_MANAGER,sale
2 im,INVENTORY_MANAGER,im
3 fm,FINANCE_MANAGER,fm
4 daryl,ADMIN,daryl
5 admin,ADMIN,admin
6 pm,PURCHASE_MANAGER,pm
7
```

2.5.4 Account Editing

By pressing the “Edit” button when viewing account details, the administrator is able to edit the details of an account including username, password, and role. The administrator can save their

changes by pressing the “Confirm” button. Pressing the “Cancel” button will restore the previous account details.



Accounts			Back	Refresh	Add New
User Name	Role	View			
sale	SALES_MANAGER	View			
im	INVENTORY_MANAGER	View			
fm	FINANCE_MANAGER	View			
daryl	ADMIN	View			
admin	ADMIN	View			
pm	PURCHASE_MANAGER	View			

POTS

—

□

×

Add User

Back

Delete

Edit

Username:

daryl

Password

Role:

Admin

Confirm

Cancel

POTS

Add User

Back

Delete

Edit

Username:

daryl

Password

Role:

Finance Manager

Confirm

Cancel

POTS

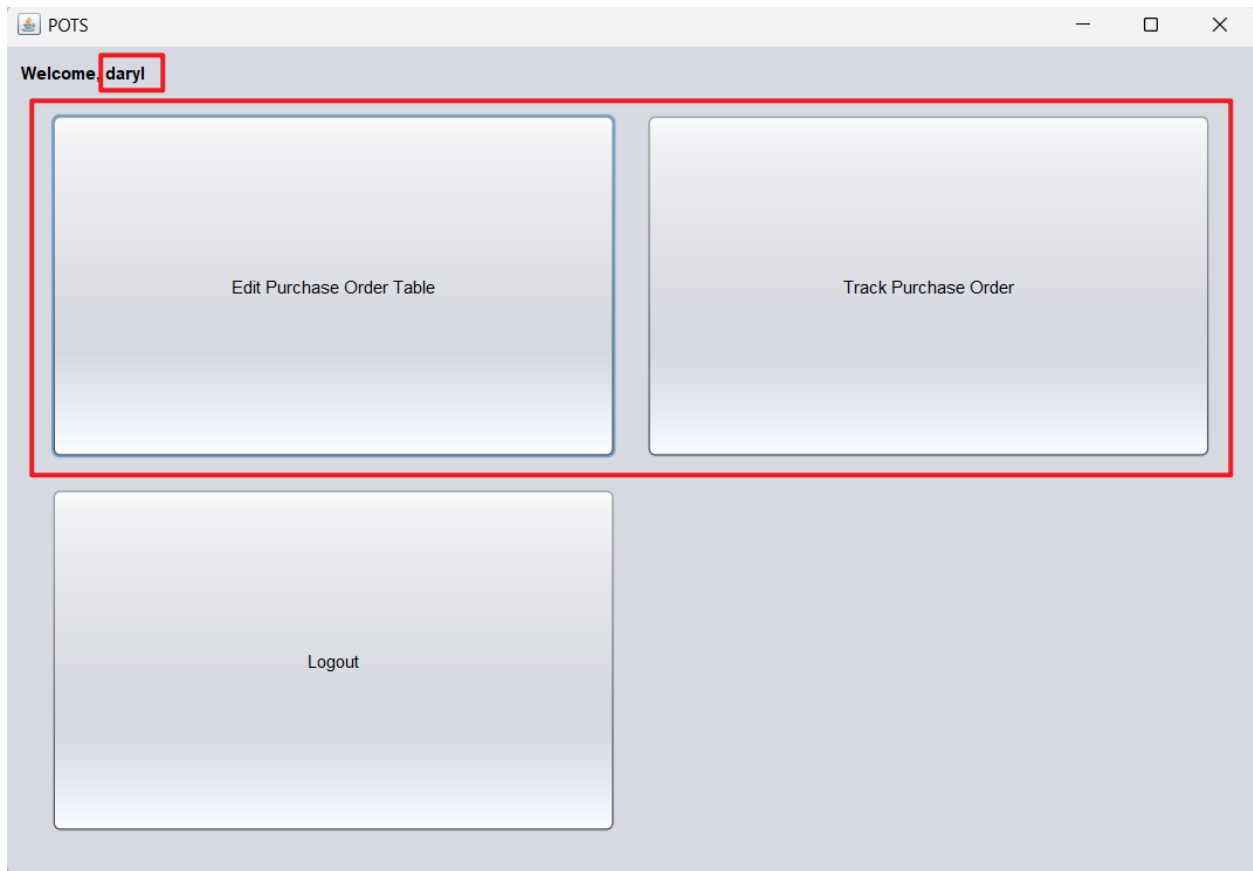
Accounts

Back Refresh Add New

User Name	Role	View
sale	SALES_MANAGER	View
im	INVENTORY_MANAGER	View
fm	FINANCE_MANAGER	View
daryl	FINANCE_MANAGER	View
admin	ADMIN	View
pm	PURCHASE_MANAGER	View


After the account details have been confirmed, they will be updated on the Accounts table and accounts.csv file.

```
data > accounts.csv > data
1 sale,SALES_MANAGER,sale
2 im,INVENTORY_MANAGER,im
3 fm,FINANCE_MANAGER,fm
4 daryl,FINANCE_MANAGER,daryl
5 admin,ADMIN,admin
6 pm,PURCHASE_MANAGER,pm
7
```



The account's details will also be portrayed on the pages to reflect the changes.

2.5.5 Account Deletion

 POTS

Accounts

Back

Refresh

Add New

User Name	Role	View
sale	SALES_MANAGER	<div>View</div>
im	INVENTORY_MANAGER	<div>View</div>
fm	FINANCE_MANAGER	<div>View</div>
daryl	FINANCE_MANAGER	<div>View</div>
admin	ADMIN	<div>View</div>
pm	PURCHASE_MANAGER	<div>View</div>

The screenshot shows a web application window titled "POTS". Inside the window, there is a section titled "Add User". To the right of this title are three buttons: "Back", "Delete", and "Edit". The "Delete" button is highlighted with a red rectangular border. Below the title, there are three form fields: "Username:" with the value "daryl", "Password:" with the value "*****", and "Role:" with a dropdown menu showing "Finance Manager".

When viewing the account details, the administrator is able to delete the account by pressing the "Delete" button. By doing this, the account is no longer exists in the Accounts table and accounts.csv file. Attempts to log into the deleted account will not be successful.

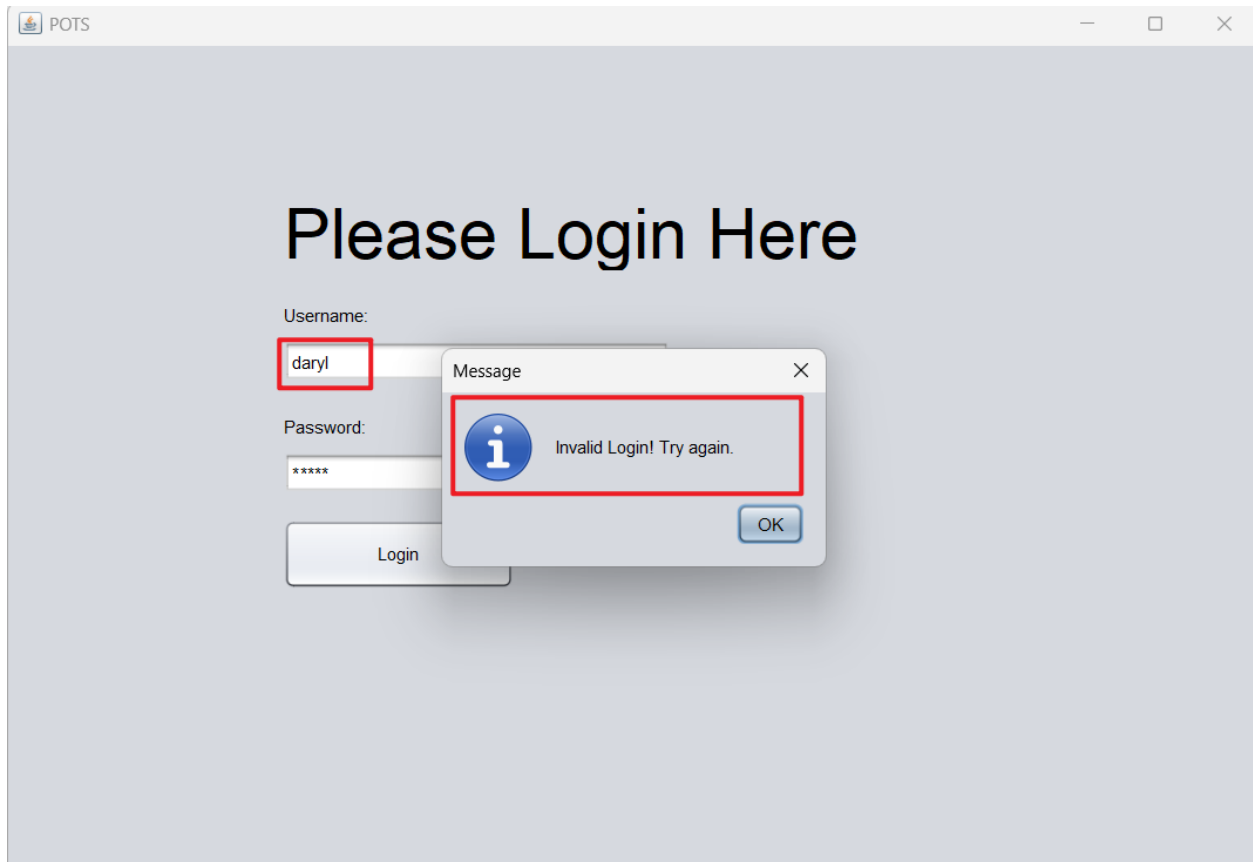
POTS

Accounts

Back Refresh Add New

User Name	Role	View
sale	SALES_MANAGER	View
im	INVENTORY_MANAGER	View
fm	FINANCE_MANAGER	View
admin	ADMIN	View
pm	PURCHASE_MANAGER	View

```
data > accounts.csv > data
1 sale,SALES_MANAGER,sale
2 im,INVENTORY_MANAGER,im
3 fm,FINANCE_MANAGER,fm
4 admin,ADMIN,admin
5 pm,PURCHASE_MANAGER,pm
6 |
```



3.0 Object-Oriented Programming Concepts

3.1 Encapsulation

Encapsulation is the concept of grouping variables into a single class. This is done in order to hide sensitive data from the users, as they are unable to view the class itself. It also allows data to be restricted in terms of how it can be accessed or modified. In Java, this is done by declaring variables as private whilst providing getters and setters in order to update and retrieve the private variables.

In this example, the variables “username”, “password”, and “role” are declared as private. This means that they cannot be accessed without using getter and setter methods.

```
public class Account extends BaseItem {  
    private String username;  
    private String password;  
    private Role role;
```

As such, public getter and setter classes are implemented to allow for the ability to retrieve and update these variables. With this, variables can be written and read by employing their respective getter and setter methods.

```
// Getters and Setters  
public String getUsername() {  
    return username;  
}  
  
public void setUsername(String username) {  
    this.username = username;  
}  
  
public String getPassword() {  
    return password;  
}  
  
public void setPassword(String password) {  
    this.password = password;  
}  
  
public Role getRole() {  
    return role;  
}  
  
public void setRole(Role role) {  
    this.role = role;  
}
```

3.2 Modularity

Modularity is a concept that involves dividing a system into several smaller and simpler units called modules which represent a specific method or variable. These modules work in conjunction with each other in order to achieve the function of the system. This allows systems to be more efficiently extended as new modules can be added without interfering with the other modules. Debugging is also easier as any errors can be isolated into a specific module. These modules can be ported into other systems, reducing the need to program a new function from scratch.

In Java, the most common way of achieving modularity is by defining a class. A class is the most fundamental form of modularity as it contains data and methods relating to a specific function. In this example, several classes are made which serve a specific function. These classes contain methods related to purchase orders, purchase requisitions, sales, and items respectively. These methods are contained in each class so as to not interfere with one another and any issues can be easily resolved by deducing faulty module.

```
1 package data;
2
3 import java.util.Date;
4
5 You: 5 hours ago | 3 authors (meow-d and others)
6 public class PurchaseOrder extends BaseItem {
7     private String PurchaseRequisitionId;
8     private String supplierId;
9     private String purchaseManagerId;
10    private Status poStatus;
11    private Date createdDate;
12    private double totalAmount;
13
14    // Constructor
15    public PurchaseOrder(String PurchaseRequisitionId, String supplierId, String purchaseManagerId, Status poStatus,
16        Date createdDate, double totalAmount) {
17        this.PurchaseRequisitionId = PurchaseRequisitionId;
18        this.supplierId = supplierId;
19        this.purchaseManagerId = purchaseManagerId;
20        this.poStatus = poStatus;
21        this.createdDate = createdDate;
22        this.totalAmount = totalAmount;
23    }
24
25    public PurchaseOrder(String PurchaseRequisitionId, String supplierId, String purchaseManagerId, Status poStatus,
26        Date createdDate, double totalAmount, String Id) {
27        this.PurchaseRequisitionId = PurchaseRequisitionId;
28        this.supplierId = supplierId;
29        this.purchaseManagerId = purchaseManagerId;
30        this.poStatus = poStatus;
31        this.createdDate = createdDate;
32        this.totalAmount = totalAmount;
33        this.Id = Id;
34    }
35 }
```



```

1 package data;
2
3 import java.util.Date;
4
5 You, 5 hours ago | 2 authors (meow-d and one other)
6 public class PurchaseRequisition extends BaseItem {
7     private String itemId;
8     private int quantity;
9     private Date requiredByDate;
10    private String salesManagerId;
11    private Status status;
12
13    // Constructor
14    public PurchaseRequisition(String itemId, int quantity, Date requiredByDate, String salesManagerId, Status status) {
15        this.itemId = itemId;
16        this.quantity = quantity;
17        this.requiredByDate = requiredByDate;
18        this.salesManagerId = salesManagerId;
19        this.status = status;
20    }
21
22    public PurchaseRequisition(String itemId, int quantity, Date requiredByDate, String salesManagerId, Status status,
23        String Id) {
24        this.itemId = itemId;
25        this.quantity = quantity;
26        this.requiredByDate = requiredByDate;
27        this.salesManagerId = salesManagerId;
28        this.status = status;
29        this.Id = Id;
30    }

```

```

1 package data;
2
3 import java.util.Date;
4
5 You, 5 hours ago | 2 authors (meow-d and one other)
6 public class Sale extends BaseItem {
7     private String itemId;
8     private int quantitySold;
9     private Date saleDate;
10    private String salesManagerId;
11
12    // Constructor
13    public Sale(String itemId, int quantitySold, Date saleDate, String salesManagerId) {
14        this.itemId = itemId;
15        this.quantitySold = quantitySold;
16        this.saleDate = saleDate;
17        this.salesManagerId = salesManagerId;
18    }
19
20    public Sale(String itemId, int quantitySold, Date saleDate, String salesManagerId, String Id) {
21        this.itemId = itemId;
22        this.quantitySold = quantitySold;
23        this.saleDate = saleDate;
24        this.salesManagerId = salesManagerId;
25        this.Id = Id;
26    }

```

```

5 public class Item extends BaseItem {
6     private String itemCode;
7     private String itemName;
8     private String supplierId;
9     private int stockLevel;
10    private int reorderLevel;
11    private BigDecimal price;
12
13    // Constructor
14    public Item(String itemCode, String itemName, String supplierId, int stockLevel, int reorderLevel,
15               BigDecimal price) {
16        this.itemCode = itemCode;
17        this.itemName = itemName;
18        this.supplierId = supplierId;
19        this.stockLevel = stockLevel;
20        this.reorderLevel = reorderLevel;
21        this.price = price;
22    }
23
24    public Item(String itemCode, String itemName, String supplierId, int stockLevel, int reorderLevel, BigDecimal price,
25               String Id) {
26        this.itemCode = itemCode;
27        this.itemName = itemName;
28        this.supplierId = supplierId;
29        this.stockLevel = stockLevel;
30        this.reorderLevel = reorderLevel;
31        this.price = price;
32        this.Id = Id;
33    }

```

3.3 Inheritance

Inheritance is a concept that involves two elements, a superclass and a subclass. A superclass is the parent class in which the subclass inherits attributes and methods from while a subclass is the child class that inherits from the parent class. This allows for more efficient coding as instead of defining a new class every time, developers are able to reuse the attributes of the superclass. Subclasses can also be made to feature additional methods without adding to the superclass, reducing complexity when coding.

In Java, inheritance is defined with the keyword “extends” which is used to indicate a subclass of a parent class.

In this example, a superclass “BaseItem” is defined. All subclasses, like Sale, Account, and more, inherit from the BaseItem class, which contains an auto generated ID field. This ensures all items have an ID without duplicate code.

Example 1

```
1 package data;
2
3 import java.util.UUID;
4
5 You, 4 hours ago | 2 authors (meow-d and one other)
6 public abstract class BaseItem {
7     protected String Id;
8
9     public BaseItem() {
10         this.Id = UUID.randomUUID().toString();
11     }
12
13     public String getId() {
14         return Id;
15     }
16
17     protected void setId(String id) {
18         this.Id = id;
19     }
20
21     public abstract void printLog();
22 }
```

```
1 package data;
2
3 You, 4 hours ago | 2 authors (meow-d and one other)
4 public class Supplier extends BaseItem {
5     private String supplierCode;
6     private String supplierName;
7     private String contactInfo;
8
9     // Constructor
10    public Supplier(String supplierCode, String supplierName, String contactInfo) {
11        this.supplierCode = supplierCode;
12        this.supplierName = supplierName;
13        this.contactInfo = contactInfo;
14    }
15
16    public Supplier(String supplierCode, String supplierName, String contactInfo, String Id) {
17        this.supplierCode = supplierCode;
18        this.supplierName = supplierName;
19        this.contactInfo = contactInfo;
20        this.Id = Id;
21    }
22
23    // Getters and Setters
24    public String getSupplierCode() {
25        return supplierCode;
26    }
27
28    public void setSupplierCode(String supplierCode) {
29        this.supplierCode = supplierCode;
30    }
31
32    public String getSupplierName() {
33        return supplierName;
34    }
35 }
```

```

1 package data;
2
3 import java.util.Date;
4
5 You, 4 hours ago | 2 authors (meow-d and one other)
6 public class Sale extends BaseItem {
7     private String itemId;
8     private int quantitySold;
9     private Date saleDate;
10    private String salesManagerId;
11
12    // Constructor
13    public Sale(String itemId, int quantitySold, Date saleDate, String salesManagerId) {
14        this.itemId = itemId;
15        this.quantitySold = quantitySold;
16        this.saleDate = saleDate;
17        this.salesManagerId = salesManagerId;
18    }
19
20    public Sale(String itemId, int quantitySold, Date saleDate, String salesManagerId, String Id) {
21        this.itemId = itemId;
22        this.quantitySold = quantitySold;
23        this.saleDate = saleDate;
24        this.salesManagerId = salesManagerId;
25        this.Id = Id;
26    }
27
28    // Getters and Setters
29    public String getItemId() {
30        return itemId;
31    }
32
33    public void setItemId(String itemId) {
34        this.itemId = itemId;
35    }

```

Example 2

```

public abstract class BasePanel<T extends BaseItem> extends JPanel {
    protected JPanel panel, titleButtonPanel;
    protected JButton backButton;
    protected CustomJPanel contentPanel;
    protected Map<String, T> items;
    protected Backend backend;
    protected MainMenu parent;

    public BasePanel(String title, MainMenu parent, Map<String, T> items, Backend backend) {
        this.items = items;
        this.backend = backend;
        this.parent = parent;

        // Title
        setLayout(new BorderLayout());
        JPanel titlePanel = new TitlePanel(title);
        add(titlePanel, BorderLayout.NORTH);
    }

```

```

public class AccountForm extends BasePanel<Account> {
    protected JTextField fieldUsername;
    protected FieldPassword fieldPassword;
    protected JLabel greenLabel, roleLabel;
    protected JComboBox<String> roleComboBox;
    private Role role;
    protected Account account;
    private AccountList accountList;
    private String rowData;
    private JButton editConfirm, editCancel, deleteButton, editButton, confirmButton;
    private JPanel editcfm;

    public AccountForm(Backend backend, MainMenu parent) {
        super(title:"Add User", parent, backend.db.accountsMap, backend);
        this.accountList = new AccountList();
        role = backend.getCurrentAccount().getRole();

        fieldUsername = new JTextField(fieldName:"Username:");
        contentPanel.add(fieldUsername);

        fieldPassword = new FieldPassword(fieldName:"Password");
        contentPanel.add(fieldPassword);

        // Creating Panel for Roles
        JPanel rolePanel = new JPanel(new GridLayout(rows:1, cols:2));

```

3.4 Polymorphism

Polymorphism is a concept enabled by Inheritance, in which a subclass may have different methods or parameters than its parent class. This allows the inherited methods to perform different tasks and enables more dynamic system behaviour as the methods from the parent class can perform differently based on different conditions and interactions. It also makes programming more efficient as developers no longer need to define a new function for every condition.

In Java, this is achieved by redefining the parent class's methods in the child class. There are two mechanics related to Polymorphism: Overriding and Overloading.

3.4.1 Overriding

Overriding refers to when the subclass provides its own implementation of a method that already exists within the parent class. The subclass's method must have name, data type, and parameters as the parent class although the output can be different. This is done to give subclasses different behaviour from the parent class. In Java, this is defined using the keyword “@Override”.

In this example, a subclass “Account” extends from superclass “BaseItem”. Both classes contain a “printlog” method however the output from “Account” is different due to overriding. This allows for its output to be independent of the superclass while still inheriting from it.

Example 1

```
1 package data;
2
3 import java.util.UUID;
4
5 You, 4 hours ago 12 authors (meow-d and one other)
6 public abstract class BaseItem {
7     protected String Id;
8
9     public BaseItem() {
10         this.Id = UUID.randomUUID().toString();
11     }
12
13     public String getId() {
14         return Id;
15     }
16
17     protected void setId(String id) {
18         this.Id = id;
19     }
20     public abstract void printLog();
21 }
22
```

```
3 public class Account extends BaseItem {
27     public void setUsername(String username) {
28     }
29
30
31     public String getPassword() {
32         return password;
33     }
34
35     public void setPassword(String password) {
36         this.password = password;
37     }
38
39     public Role getRole() {
40         return role;
41     }
42
43     public void setRole(Role role) {
44         this.role = role;
45     }
46
47     @Override
48     public void printLog(){
49         System.out.println(x:"Account loaded.");
50     }
51 }
52
```

```

5 public class Item extends BaseItem {
71
72     public void setReorderLevel(int reorderLevel) {
73         this.reorderLevel = reorderLevel;
74     }
75
76     public BigDecimal getPrice() {
77         return price;
78     }
79
80     public void setPrice(BigDecimal price) {
81         this.price = price;
82     }
83
84     @Override
85     public void printLog(){
86         System.out.println(x:"Item loaded.");
87     }
88 }
89

```

Example 2

```

abstract class ComboList<T extends BaseItem> {
    protected Map<String, T> items;
    protected String[] UUID;
    protected T[] values;
    protected Backend backend;

    public void setBackend(Backend backend) {
        this.backend = backend;
    }

    public void setItem(Map<String, T> items) {
        this.items = items;
    }

    public void setValue() {
        initGetValue();
    }
}

```



```

class PRListPurOrder extends ComboList<PurchaseRequisition> {

    @Override
    public void setItem(Map<String, PurchaseRequisition> items) {
        this.items = items;
    }

    @Override
    public String getName(String UUID) {
        String itemName = backend.db.getItem(this.items.get(UUID).getItemId()).getItemName();
        String name = itemName + " || " + this.items.get(UUID).getQuantity() + " Stocks";
        return name;
    }

    @Override
    public void setValue() {
        this.values = new PurchaseRequisition[this.items.size()];
        initGetValue();
    }

    @Override

```

3.4.2 Overloading

Overloading refers to when multiple methods in a class that share the same name have different data type and/or number of parameters. This allows the class to handle different scenarios and inputs with the output varying based on them. Programming is more efficient this way as a new class does not need to be defined to account for multiple conditions.

In Java, this is achieved by defining two or more methods of the same name but with different parameters and/or data types. In this example, two “Item” methods are defined. Each of the item classes have two constructors. One is intended for normal usage, the other is for loading CSV data into classes, which all already have their ID set.

```

3  import java.math.BigDecimal;
4
5  You, 4 hours ago | 2 authors (meow-d and one other)
6  public class Item extends BaseItem {
7      private String itemCode;
8      private String itemName;
9      private String supplierId;
10     private int stockLevel;
11     private int reorderLevel;
12     private BigDecimal price;
13
14     // Constructor
15     public Item(String itemCode, String itemName, String supplierId, int stockLevel, int reorderLevel,
16         BigDecimal price) {
17         this.itemCode = itemCode;
18         this.itemName = itemName;
19         this.supplierId = supplierId;
20         this.stockLevel = stockLevel;
21         this.reorderLevel = reorderLevel;
22         this.price = price;
23     }
24
25     public Item(String itemCode, String itemName, String supplierId, int stockLevel, int reorderLevel, BigDecimal price,
26         String Id) {
27         this.itemCode = itemCode;
28         this.itemName = itemName;
29         this.supplierId = supplierId;
30         this.stockLevel = stockLevel;
31         this.reorderLevel = reorderLevel;
32         this.price = price;
33         this.Id = Id;
34     }
35
36     // Getters and Setters
37     public String getItemCode() {

```

3.5 Abstraction

Abstraction is the concept of hiding sensitive information and only allowing essential information to be visible to the user. This allows for simpler programming as there is more focus on essential details and less on implementation. Abstraction also has elements of modularity as abstracted code can be reused and extended, which is more efficient than recoding from scratch.

In Java, abstraction is achieved by using abstract classes or interfaces, these can only be accessed via inheritance. Abstract classes feature a body in their methods, however they will only operate when inherited, while interfaces do not feature a body in their methods as to be implemented via inheritance.

Example 1: Abstract Class and Abstract Method

```

1 package data;
2
3 import java.util.UUID;
4
5 You, 4 hours ago 12 authors (meow-d and one other)
6 public abstract class BaseItem {
7     protected String Id;
8
9     public BaseItem() {
10         this.Id = UUID.randomUUID().toString();
11     }
12
13     public String getId() {
14         return Id;
15     }
16
17     protected void setId(String id) {
18         this.Id = id;
19     }
20     public abstract void printLog();
21 }
22

```

```

3 public class Account extends BaseItem {
27     public void setUsername(String username) {
28     }
29
30
31     public String getPassword() {
32         return password;
33     }
34
35     public void setPassword(String password) {
36         this.password = password;
37     }
38
39     public Role getRole() {
40         return role;
41     }
42
43     public void setRole(Role role) {
44         this.role = role;
45     }
46
47     @Override
48     public void printLog(){
49         System.out.println(x:"Account loaded.");
50     }
51 }
52

```

```

5 public class Item extends BaseItem {
71
72     public void setReorderLevel(int reorderLevel) {
73         this.reorderLevel = reorderLevel;
74     }
75
76     public BigDecimal getPrice() {
77         return price;
78     }
79
80     public void setPrice(BigDecimal price) {
81         this.price = price;
82     }
83
84     @Override
85     public void printLog(){
86         System.out.println(x:"Item loaded.");
87     }
88 }
89

```

Example 2

```

1 package user_interface.table;
2
3 meow-d, 4 weeks ago | 1 author (meow-d)
4 public interface TableRefreshable {
5     public void refresh();
6 }

```

```

public abstract class TablePanel<T extends BaseItem> extends JPanel implements TableRefreshable {
    public void refresh() {
        ArrayList<T> array = new ArrayList<>(items.values());
        tableModel.setItems(array);
    }

    abstract public void createAddPanel();

    abstract public void createEditPanel(int modelRow);
}

```

```

public class PurchaseOrdersTable extends TablePanel<PurchaseOrder> {
    @Override
    public void refresh() {
        ArrayList<PurchaseOrder> array = new ArrayList<>(items.values());
        tableModel.setItems(array);
        ((PurchaseOrdersTableModel) tableModel).setMaps(backend.db.purchaseRequisitionsMap, backend.db.suppliersMap,
            backend.db.itemsMap);
    }
}

```

```

public class TrackPurchaseOrderTable extends PurchaseOrdersTable{
    @Override
    public void refresh() {
        ArrayList<PurchaseOrder> array = new ArrayList<>();
        for (PurchaseOrder item : items.values()) {
            if (item.getPoStatus() != Status.PENDING) {
                array.add(item);
            }
        }

        tableModel.setItems(array);
        ((PurchaseOrdersTableModel) tableModel).setMaps(backend.db.purchaseRequisitionsMap, backend.db.suppliersMap,
            backend.db.itemsMap);
    }
}

```

Example 3

```

public abstract class TablePanel<T extends BaseItem> extends JPanel implements TableRefreshable {
    public void refresh() {
        ArrayList<T> array = new ArrayList<>(items.values());
        tableModel.setItems(array);
    }

    abstract public void createAddPanel();

    abstract public void createEditPanel(int modelRow);
}

```

```

public class PurchaseOrdersTable extends TablePanel<PurchaseOrder> {

    @Override
    public void createAddPanel() {
        // TODO add item panel
        role = backend.getCurrentAccount().getRole();
        POPanel = parent.getPanel(panelName:"PurOrdPane", panelClass:PurchaseOrderPanel.class);
        POPanel.setBack(PanelName:"purchaseOrdersTable");
        if(role.hasPermission(tableName:"PurchaseOrder", Permission.CREATE)){
            POPanel.createPO();
        }

        parent.showPanel(panelName:"PurOrdPane");
    }

    @Override
    public void createEditPanel(int modelRow) {
        role = backend.getCurrentAccount().getRole();
        System.out.println(x:"helo from purchase order table"); //TODO: remove
        POPanel = parent.getPanel(panelName:"PurOrdPane", panelClass:PurchaseOrderPanel.class);

        POPanel.setBack(PanelName:"purchaseOrdersTable");
        POPanel.setRowNum(tableModel.getValueAt(modelRow, columnIndex:11).toString());

        POPanel.viewOnly();

        if(role.hasPermission(tableName:"PurchaseOrder", Permission.CREATE)){
            POPanel.viewOnlyUpdate();
        } else if(role == Role.FINANCE_MANAGER){
            System.out.println(x:"im a FMFMMMM");
            POPanel.FMView(bool:true);
        }
        POPanel.setData();
        parent.showPanel(panelName:"PurOrdPane");
    }
}

```

4.0 Additional Features

4.1 HashMaps and UUIDs

```
public class Database {
    public HashMap<String, Account> accountsMap = new HashMap<>();
    public HashMap<String, Item> itemsMap = new HashMap<>();
    public HashMap<String, Supplier> suppliersMap = new HashMap<>();
    public HashMap<String, Sale> salesMap = new HashMap<>();
    public HashMap<String, PurchaseRequisition> purchaseRequisitionsMap = new HashMap<>();
    public HashMap<String, PurchaseOrder> purchaseOrdersMap = new HashMap<>();
}
```

```
public abstract class BaseItem {
    protected String Id;

    public BaseItem() {
        this.Id = UUID.randomUUID().toString();
    }
}
```

Our system stores data in key-value pairs, with each key being randomly generated unique for each data, thanks to HashMaps and Universally Unique Identifiers (UUIDs) (GeeksforGeeks, 2017). This provides extremely efficient access and the convenience of manipulating the data through the unique key (Kumar. A, 2024). In fact, using the big o notation, the time complexity is usually $O(1)$, which is significantly faster than manually cycling through data without HashMaps. In addition, duplicated data is prevented with the presence of the unique key.

4.2 Sales Report PDF Export

Monthly Sales Report

Generated on: 2024-12-19
Month: December 2024

Summary

Total Quantity Sold: 10
Total Revenue: \$200.0
Top Sales Manager: admin

Weekly Sales Breakdown

Week	Quantity Sold	Revenue
51	10	200.0

In the sales report page, there is a button to export the sales report as a PDF document for convenience; allowing it to be viewed outside of the program, shared around, and stored on the computer. This is powered by the OpenPDF library.

5.0 Limitations

The system has plenty of room for improvement that we would like to improve in the future.

1. Lack of specialized user interface (UI)

Most of the system's functionality uses the same table and form design, providing a similar interface for all, and making it easier for us to develop, but it also means the functionality lacks more specialized UI.

2. Navigation constraints

The navigation could be improved. Right now, our system displays one page at a time, with back buttons to navigate back. The single page design only works well on small windows, doesn't scale well, and doesn't utilize the entire screen. This can be solved by adding a sidebar. The back buttons are also hardcoded for each page, when ideally, we should have a navigation system.

3. No support for keyboard shortcuts

Speaking of UI/UX, the system doesn't support keyboard shortcuts and navigation. Keyboard shortcuts not only speed up workflows for power users, they're also essential for people with certain disabilities.

4. Lack of customization in report generation

The report function is limited. It currently does not allow generating reports of previous months, and customization of info displayed.

5. Database constraints

The system should ideally use a proper database, instead of HashMap and text files. SQLite would be ideal for our purpose, as it's designed for smaller databases, lightweight, and performant. Using a database would allow us to stop worrying about converting data to string, loading data, getting data from HashMap; while providing much more flexibility than our current HashMap system.

6.0 Conclusion

In conclusion, the system we have created using Java successfully streamlines the purchasing process of NEXUS SDN BHD (NSB) while integrating the concepts of object-oriented programming such as encapsulation, modularity, inheritance, polymorphism, and abstraction. This results in a system that is capable of achieving its intended function effectively and efficiently, while also having additional features such as exporting reports and assistance from HashMap and UUIDs to help with data management. Our system is limited in terms of accessibility options, our system can serve as a foundation for future improvements and shows proof of a working concept.

7.0 References

GeeksforGeeks. (2017, April 28). *HashMap in Java*. GeeksforGeeks.

<https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/>

Kumar. A. (2024). *A Guide to HashMap in Java With Examples / Built In*. Built In.

<https://builtin.com/articles/hashmap-in->

[java#:~:text=HashMap%20is%20a%20data%20structure%20that%20uses%20the%20Map%20interface,data%20based%20on%20unique%20keys.](https://builtin.com/articles/hashmap-in-java#:~:text=HashMap%20is%20a%20data%20structure%20that%20uses%20the%20Map%20interface,data%20based%20on%20unique%20keys.)

8.0 Workload Matrix

Name	Daryl Sim Wei Shern	Jaeden Loong Deng Ze	Keith Lo Ze Hui	Lim Wen Yi	Ho Shane Foong
TP number	TP068964	TP068347	TP067653	TP067930	TP068496
Documentation	✓	✓	✓	✓	✓
Programming	✓	✓	✓	✓	✓