Group Members:

Name	**TP Number**
TEO JUN JIE (Leader)	TP079880
NG TIAN XIN	TP081807
VICTOR CHIA BING CHEN	TP080934

Module Code: **AAPP013-4-2-OOP**

Module Title: **Object Oriented Programming**

Intake Code: **UCDF2405ICT(SE)**

Assignment Title: **APU Car Sales System**

Lecturer: **KAU GUAN KIAT**

Date Hand Out: **1st APRIL 2024**

Date Hand In: **30th MAY 2025**

TABLE OF CONTENT

1.0 Assumption

The APU Car Sales System (ACSS) is a Java-based application designed to streamline the operations of a car dealership. It caters to three primary user roles: Managing Staff, Salesmen, and Customers. Each role has a distinct interface and sets of functionalities tailored to their responsibilities and needs within the car sales workflow. The system emphasizes user authentication, data management for users and cars, and processes related to car booking, sales, and feedback. The application is built with a clear separation of concerns achieved through distinct packages for different modules.

Assumptions

1. File-Based Data Storage: All system data like user profiles, car inventory, sales records, logs, etc is stored in plain text files of CSV format in .txt files. The system relies on these files being present in the expected locations and formats.

2. User Authentication: All users including Managing Staff, Salesmen, and Customers must log in to access system functionalities. Authentication is based on credentials stored in their respective data files.

3. Manual ID Generation (Implicit): While not explicitly detailed for all entities, the system implies a mechanism for generating unique IDs for new entities like customers, managers, cars and sales records, often by incrementing the last known ID or a similar file-based sequential approach.

4. Input Validation: The system performs basic input validation for user entries like checking for empty fields and format correctness where applicableto prevent logical errors and ensure data consistency, as handled by the respective UI and management classes.

5. No Concurrent Access Handling: The system is designed with the assumption of single-user access or does not support simultaneous writes by multiple users.

6. Error Handling: The system includes error handling for common issues such as file I/O problems, invalid credentials, and data inconsistencies, typically by displaying error messages to the user via dialog boxes.

7. Reporting Scope: "Reporting functionality" for managing staff is interpreted as displaying data from the system's text files in a structured, readable format like tables for payments, feedback, sales records etc. Complex analytical reports or graphical representations are outside the scope.

8. Car Status Management: The car status (available, booked, paid, cancel) is a critical piece of information managed primarily by Salesmen and viewable by relevant parties.

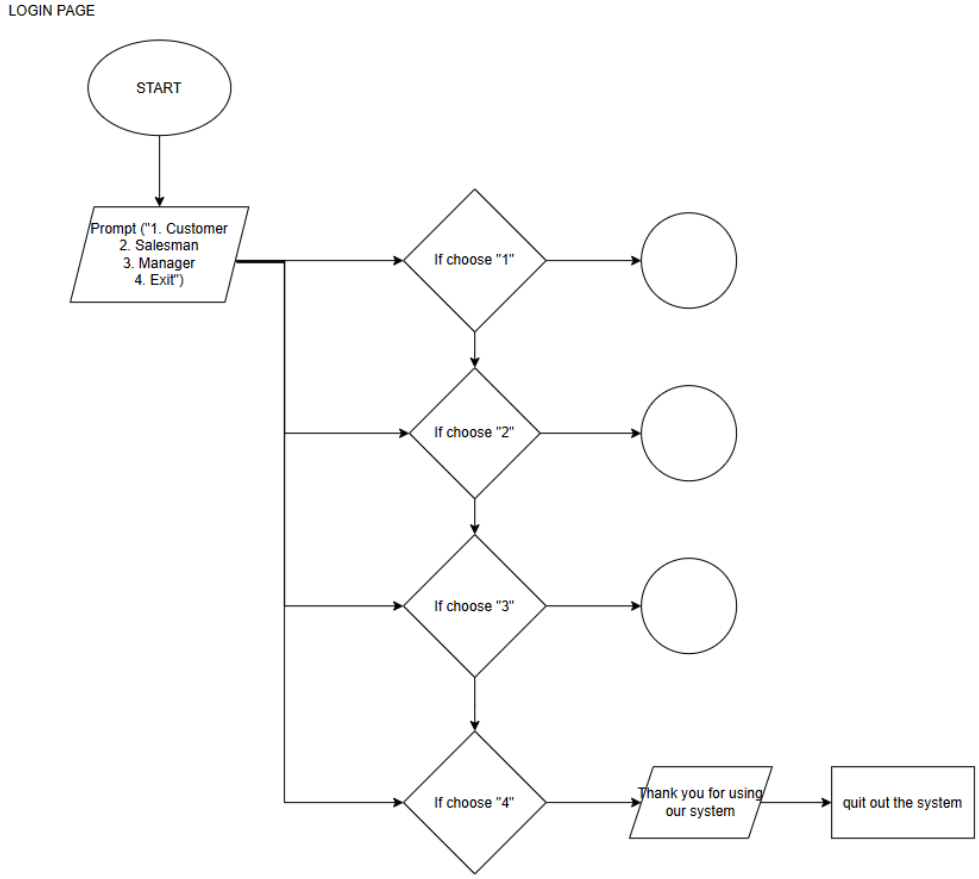
9. Sequential Operations: It's assumed that operations like booking a car, making a payment, and giving feedback occur in a logical sequence, and the system guides users through these processes.

10. Application Context (AppContext): A central AppContext class is used to manage and provide access to various management services (e.g., CarManagement , CustomerManagement) and current user session data throughout the application.

11. GUI Implementation: The user interface is implemented using Java Swing, providing graphical forms and components for interaction.

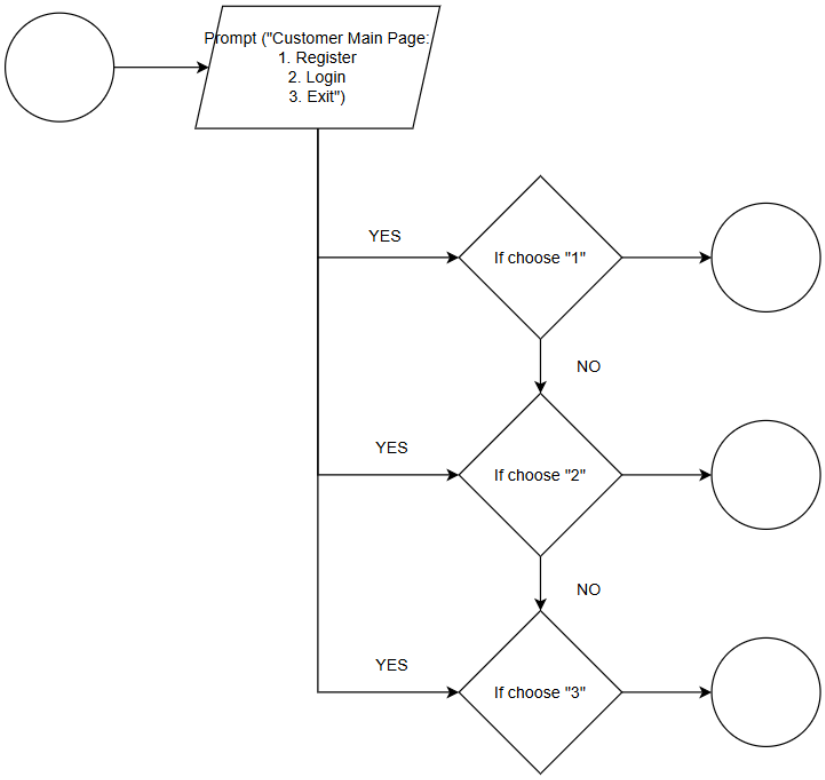
12. No External Database: The system explicitly avoids the use of any database tools like Access, Oracle, etc., adhering strictly to file-based data persistence.

2.0 Flowchart

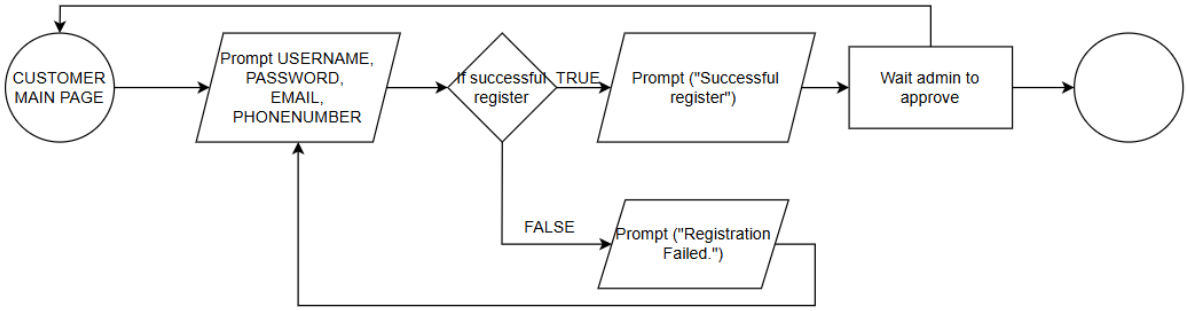


2.1 Customer

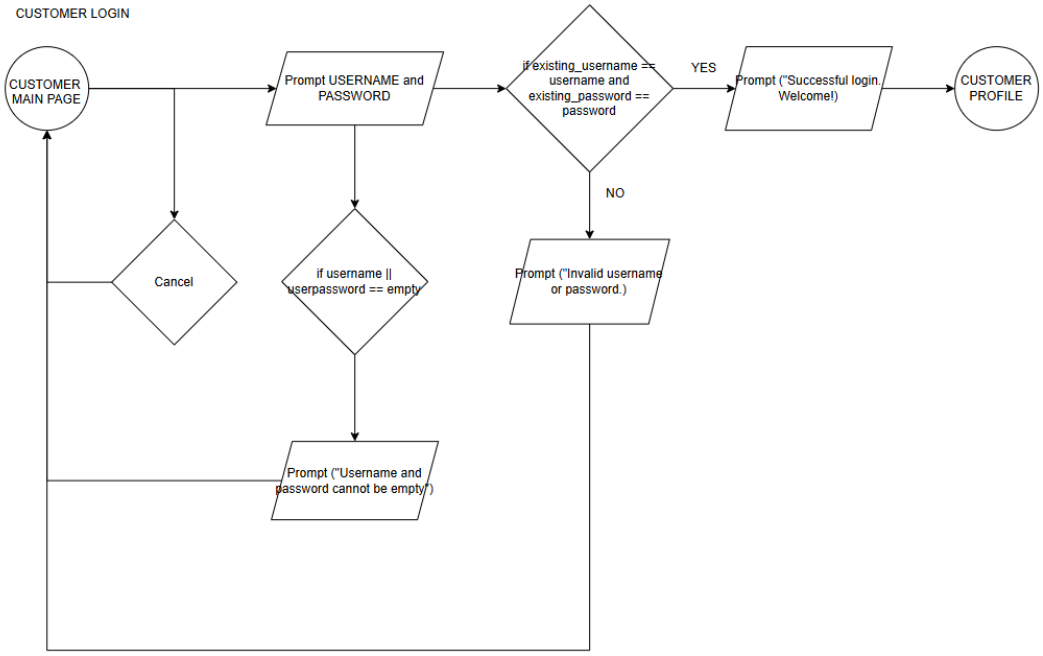
CUSTOMER MAIN PAGE



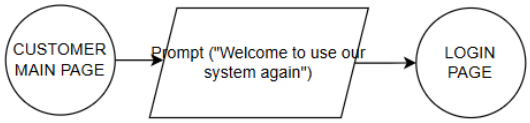
CUSTOMER REGISTER



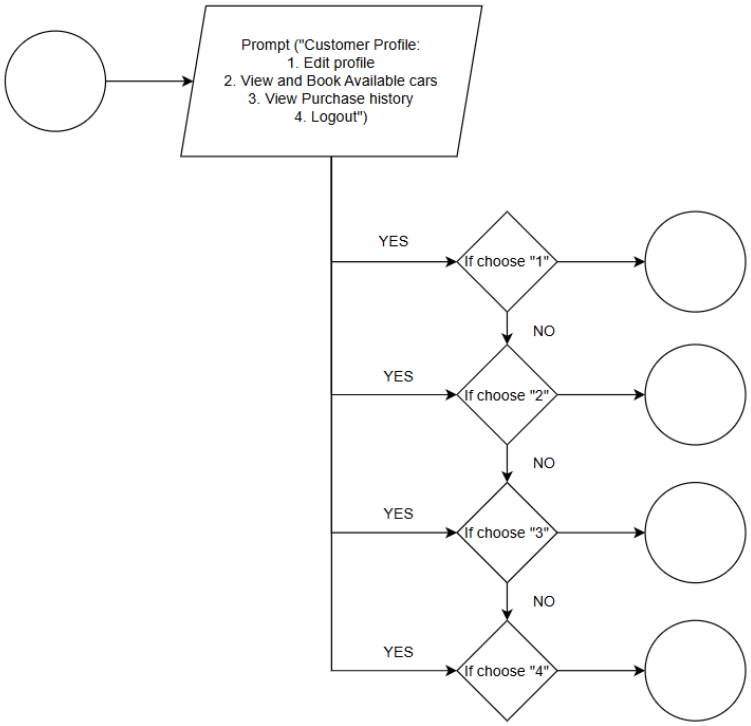
CUSTOMER LOGIN



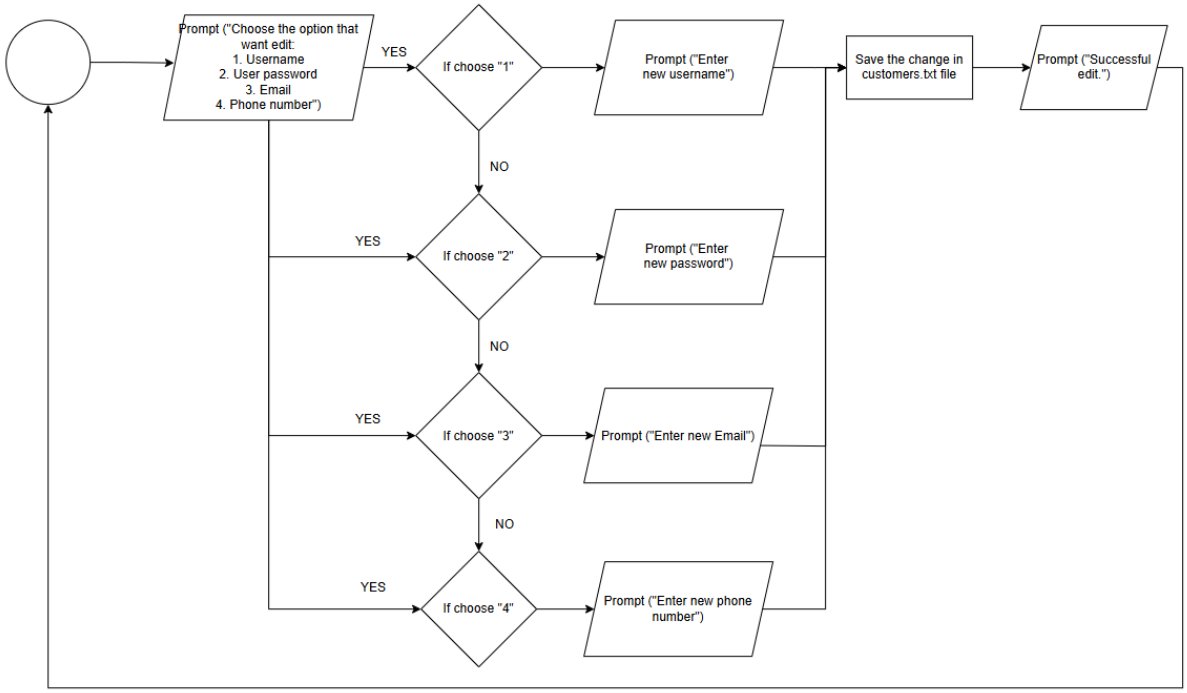
LOGOUT



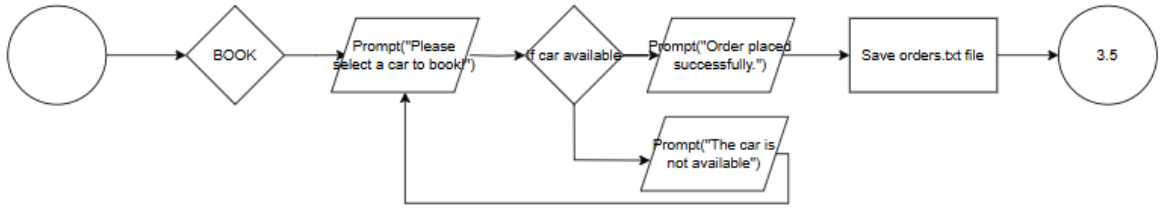
CUSTOMER PROFILE

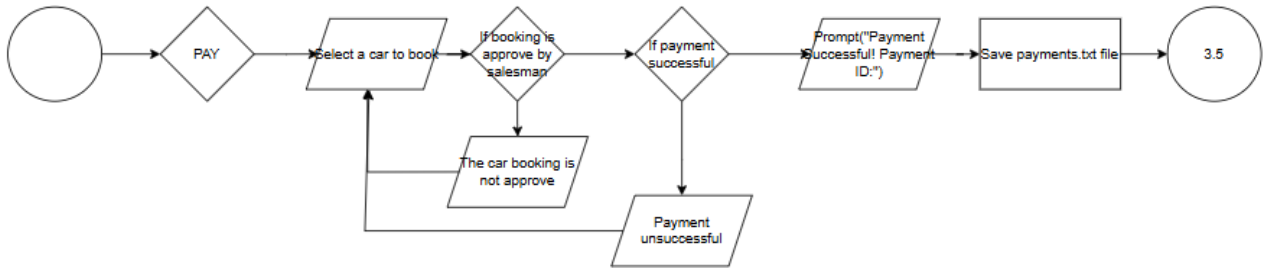


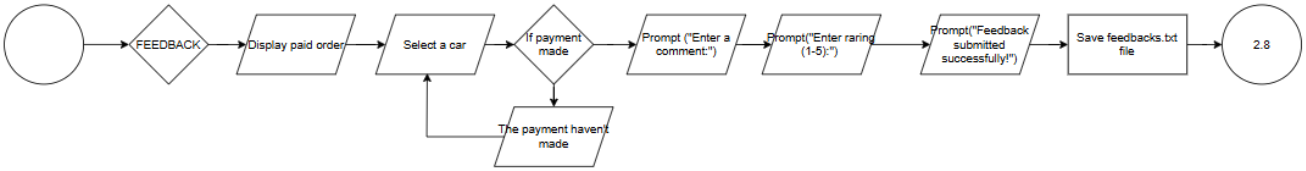
EDIT PROFILE

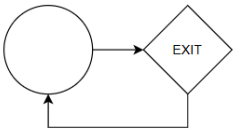


VIEW AND BOOK
AVAILABLE CARS

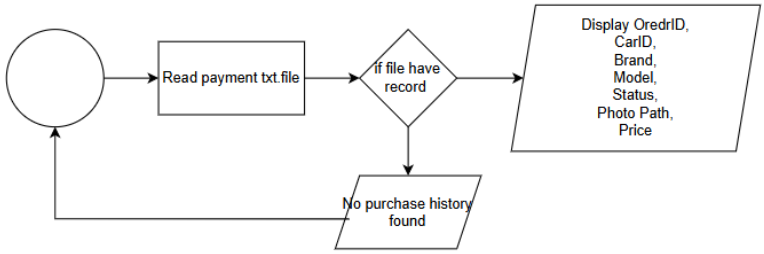




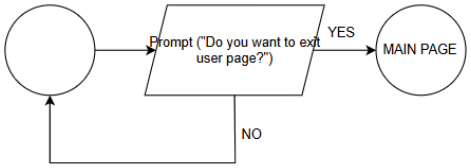




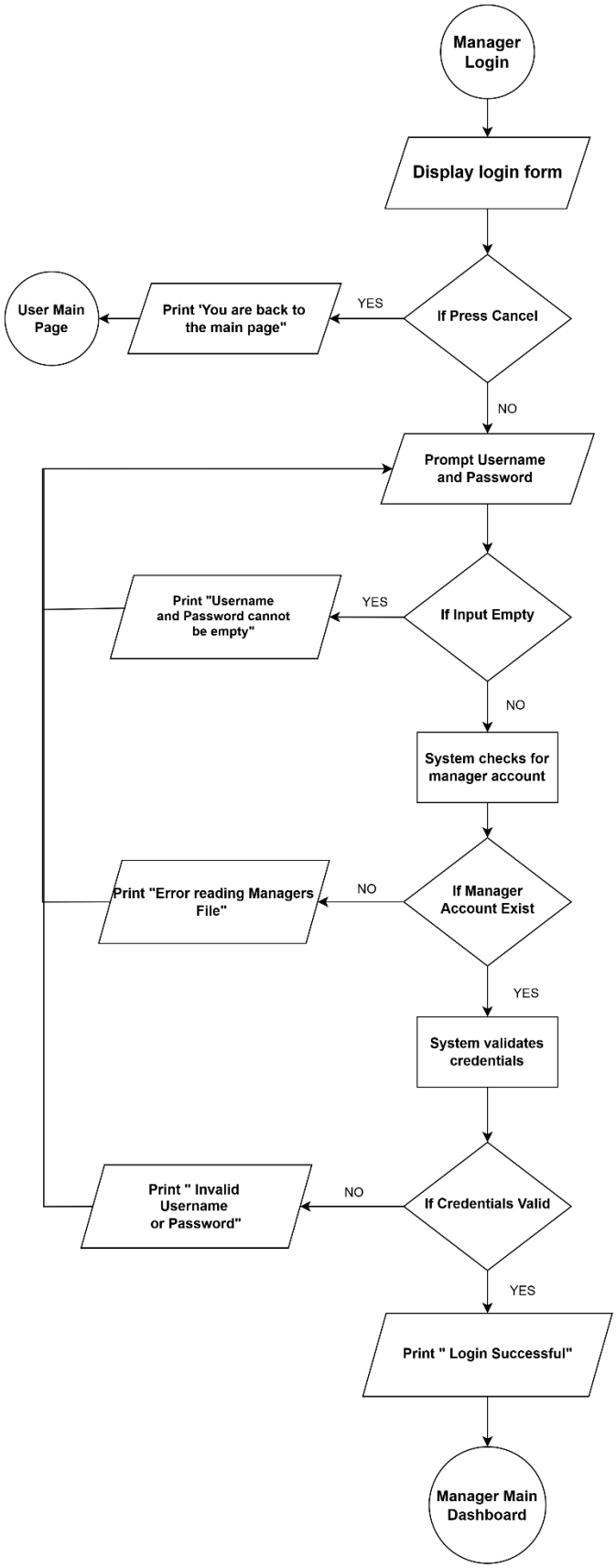
VIEW PURCHASE HISTORY

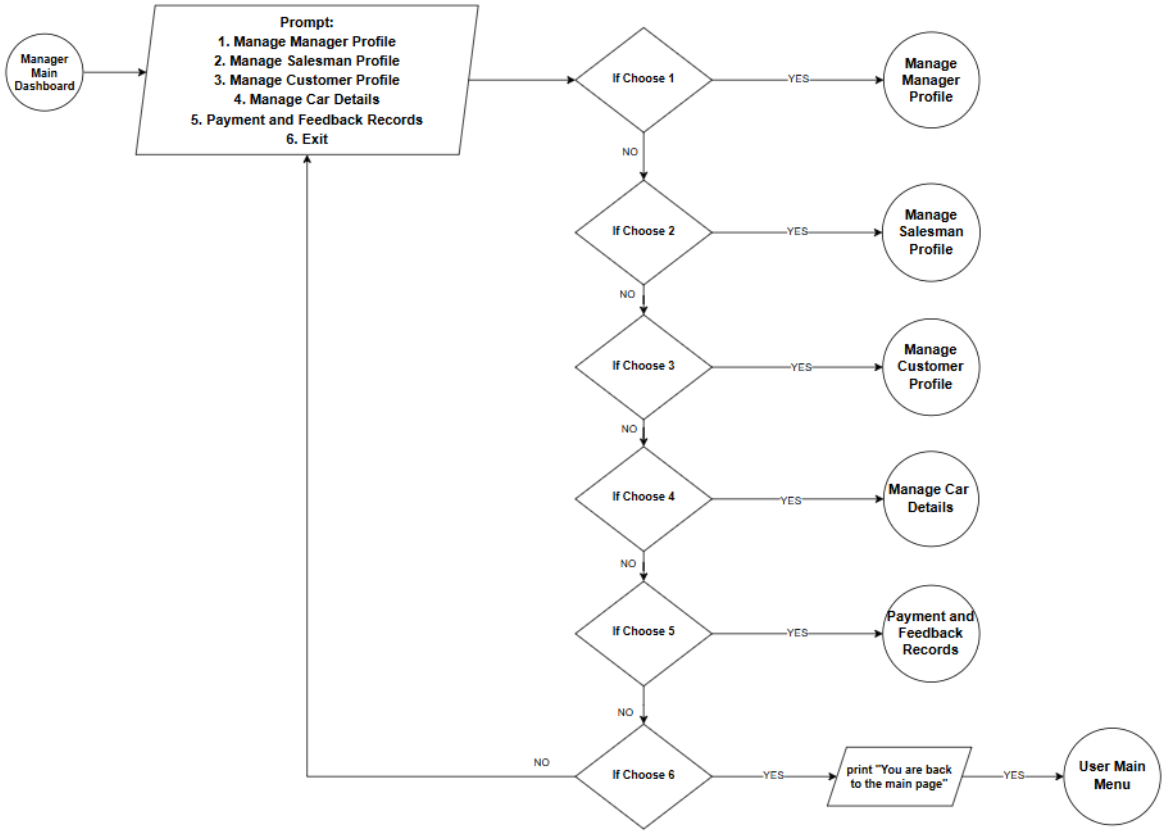


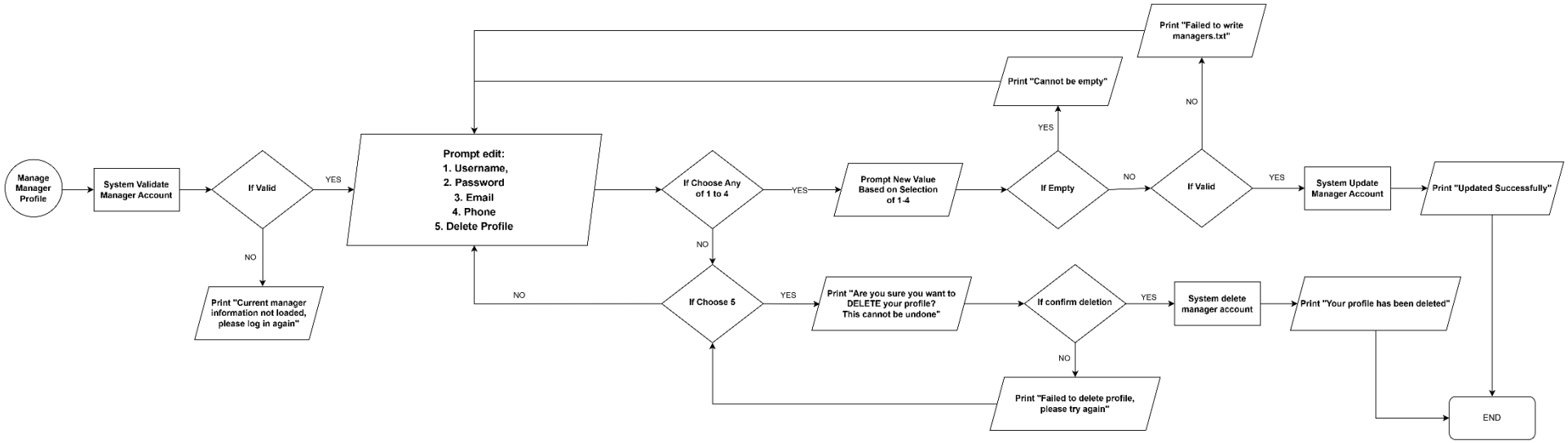
EXIT

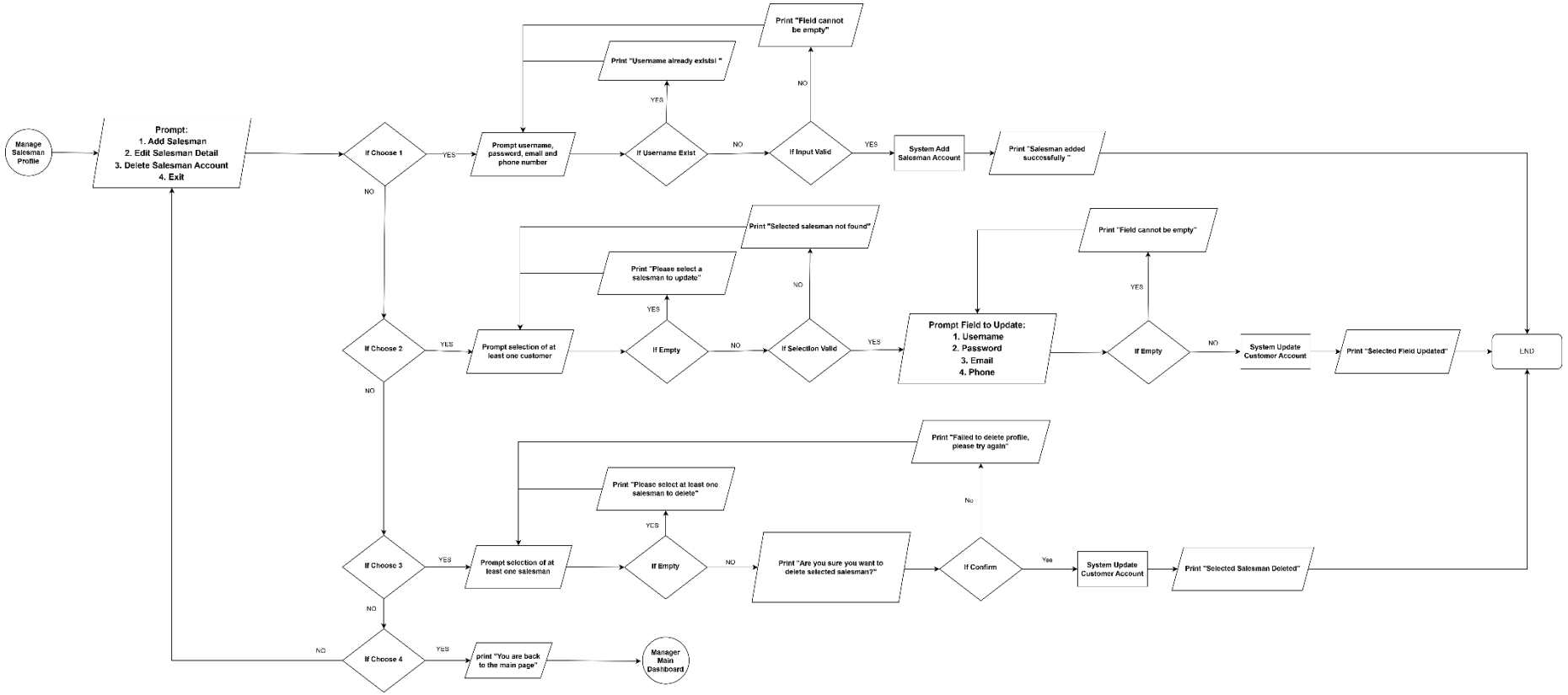


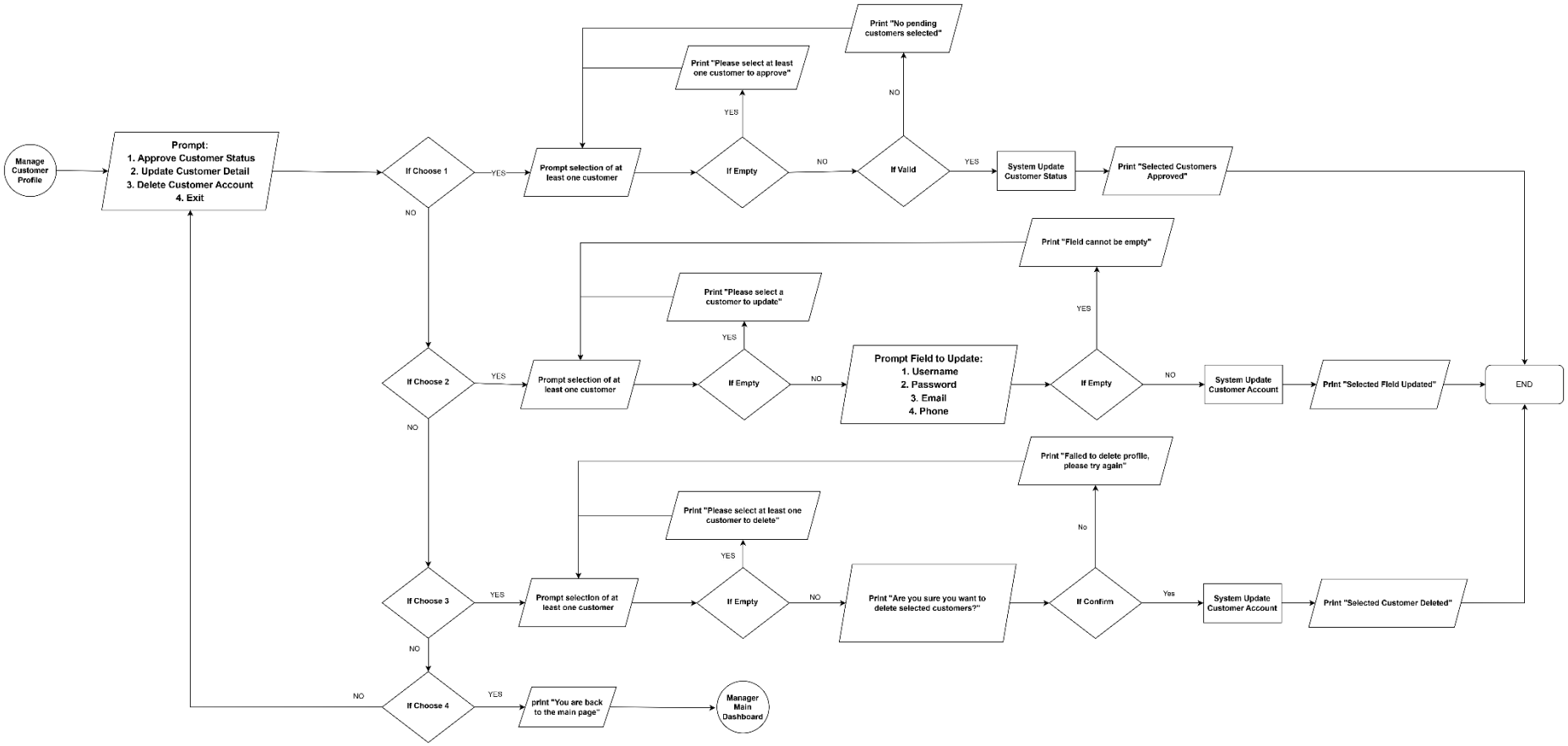
2.2 Manager

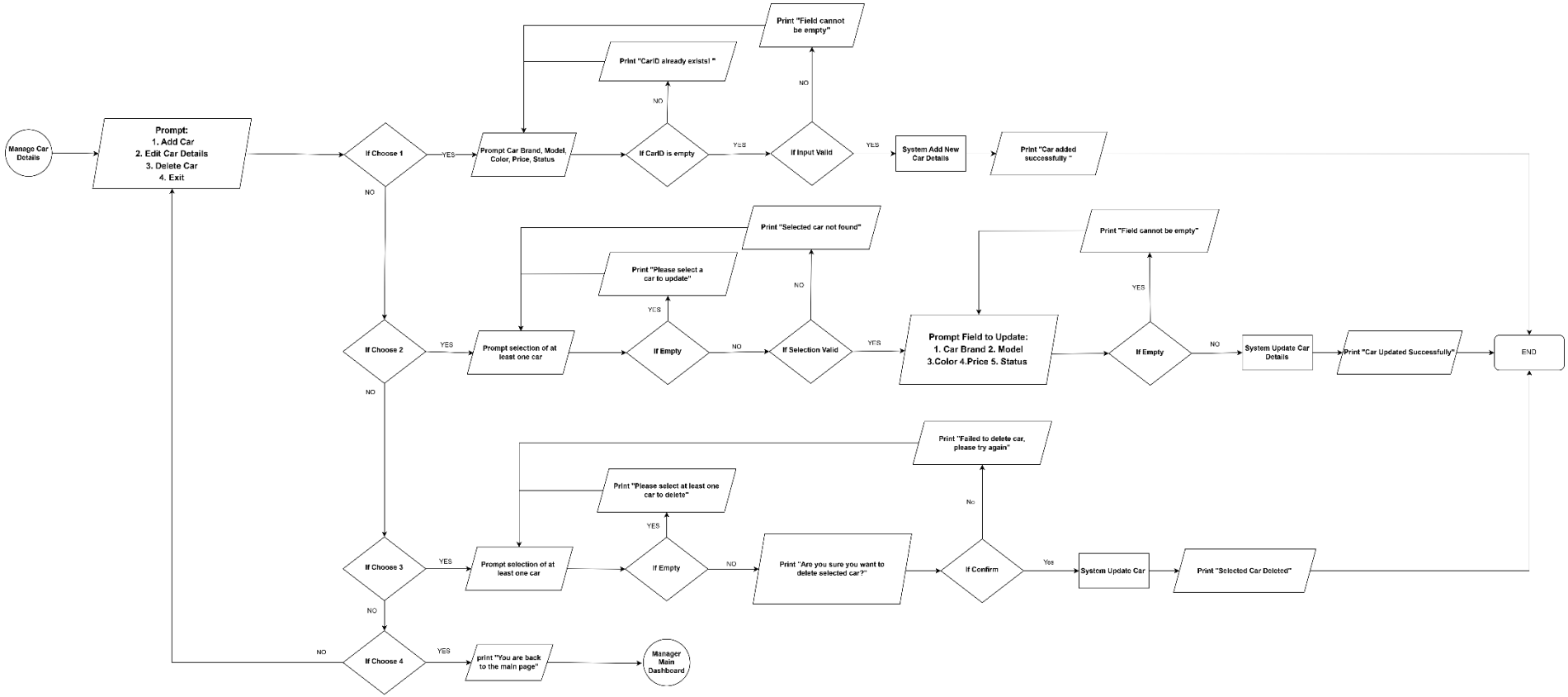

Manager Login
Display login form
If Press Cancel
YES
Print 'You are back to the main page"
User Main Page
NO
Prompt Username and Password
If Input Empty
YES
Print "Username and Password cannot be empty"
NO
System checks for manager account
If Manager Account Exist
NO
Print "Error reading Managers File"
YES
System validates credentials
If Credentials Valid
NO
Print " Invalid Username or Password"
YES
Print " Login Successful"
Manager Main Dashboard

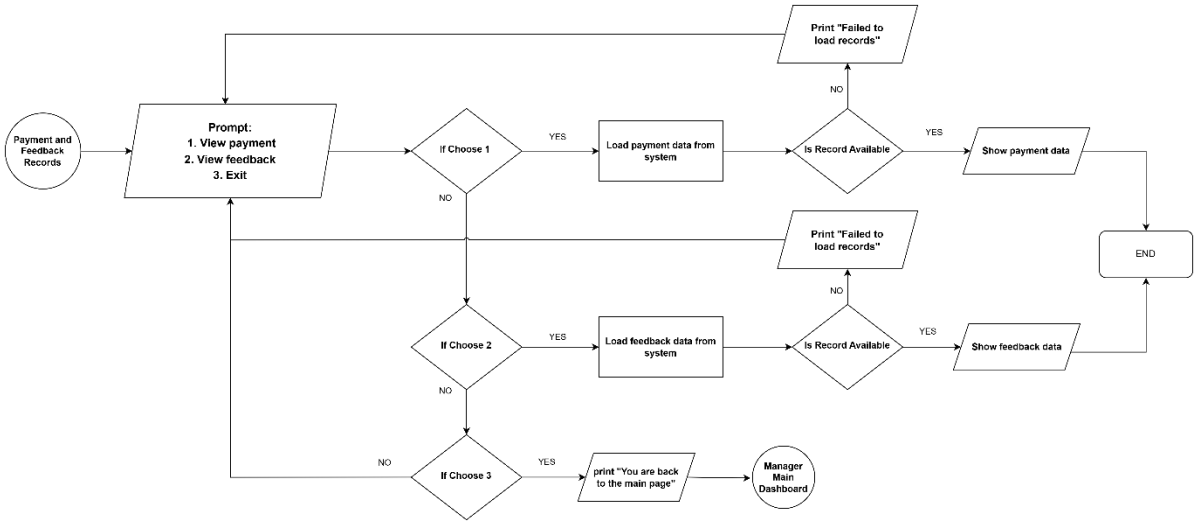

Manager Main Dashboard
Prompt:
1. Manage Manager Profile
2. Manage Salesman Profile
3. Manage Customer Profile
4. Manage Car Details
5. Payment and Feedback Records
6. Exit
If Choose 1
YES
Manage Manager Profile
NO
If Choose 2
YES
Manage Salesman Profile
NO
If Choose 3
YES
Manage Customer Profile
NO
If Choose 4
YES
Manage Car Details
NO
If Choose 5
YES
Payment and Feedback Records
NO
If Choose 6
NO
YES
print "You are back to the main page"
YES
User Main Menu


Manage Manager Profile
System Validate Manager Account
If Valid
YES
NO
Print "Current manager information not loaded, please log in again"
Prompt edit:
1. Username,
2. Password
3. Email
4. Phone
5. Delete Profile
If Choose Any of 1 to 4
YES
Prompt New Value Based on Selection of 1-4
If Empty
YES
Print "Cannot be empty"
NO
If Valid
NO
Print "Failed to write managers.txt"
YES
System Update Manager Account
Print "Updated Successfully"
NO
If Choose 5
YES
Print "Are you sure you want to DELETE your profile? This cannot be undone"
If confirm deletion
YES
System delete manager account
Print "Your profile has been deleted"
NO
Print "Failed to delete profile, please try again"
NO
END

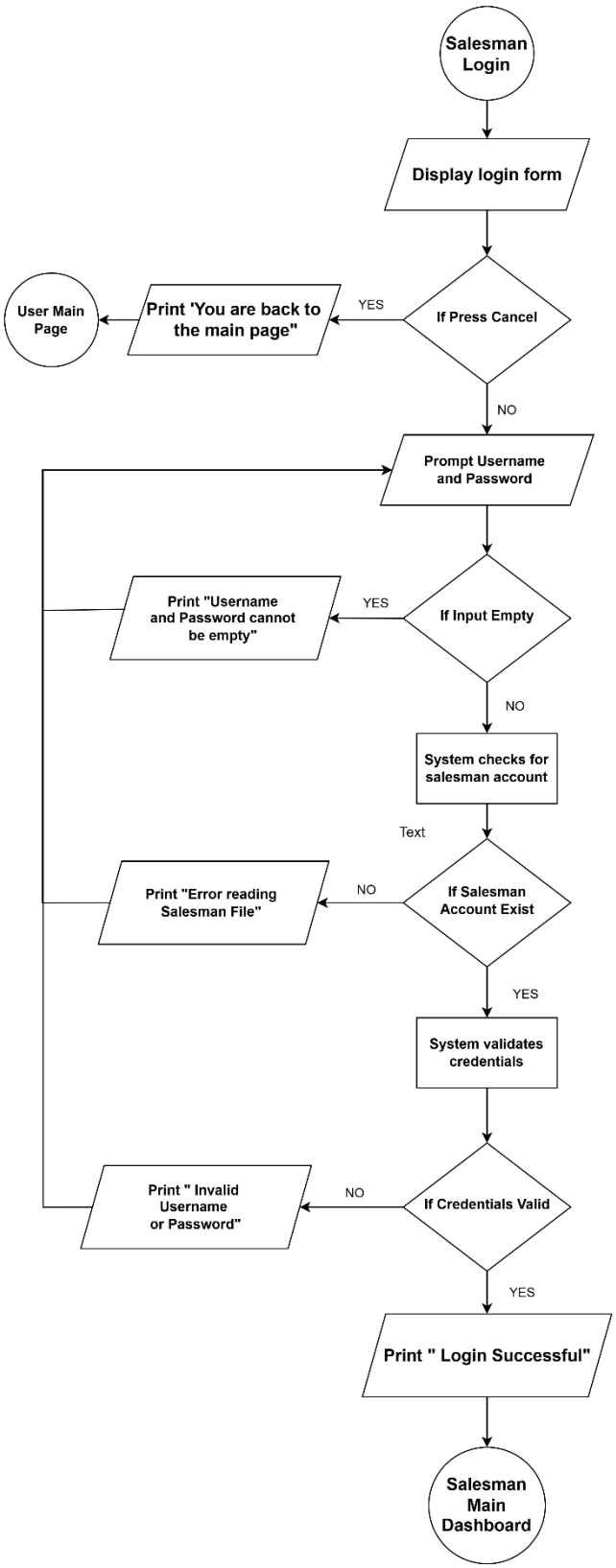

Manage Salesman Profile
Prompt:
1. Add Salesman
2. Edit Salesman Detail
3. Delete Salesman Account
4. Exit
If Choose 1
YES
NO
Prompt username, password, email and phone number
If Username Exist
YES
NO
Print "Username already exists! "
If Input Valid
YES
NO
Print "Field cannot be empty"
System Add Salesman Account
Print "Salesman added successfully "
If Choose 2
YES
NO
Prompt selection of at least one customer
If Empty
YES
NO
Print "Please select a salesman to update"
If Selection Valid
YES
NO
Print "Selected salesman not found"
Prompt Field to Update:
1. Username
2. Password
3. Email
4. Phone
If Empty
YES
NO
Print "Field cannot be empty"
System Update Customer Account
Print "Selected Field Updated"
If Choose 3
YES
NO
Prompt selection of at least one salesman
If Empty
YES
NO
Print "Please select at least one salesman to delete"
Print "Are you sure you want to delete selected salesman?"
If Confirm
Yes
No
Print "Failed to delete profile, please try again"
System Update Customer Account
Print "Selected Salesman Deleted"
If Choose 4
YES
NO
print "You are back to the main page"
Manager Main Dashboard
END

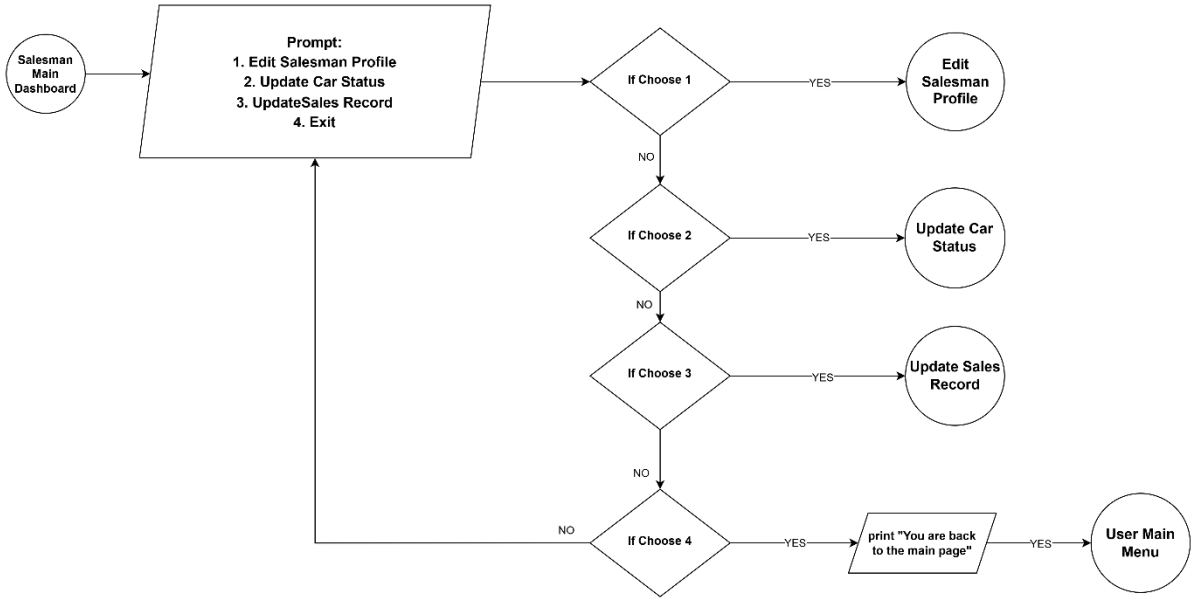

Manage Customer Profile
Prompt:
1. Approve Customer Status
2. Update Customer Detail
3. Delete Customer Account
4. Exit
If Choose 1
YES
Prompt selection of at least one customer
If Empty
YES
Print "Please select at least one customer to approve"
NO
If Valid
NO
Print "No pending customers selected"
YES
System Update Customer Status
Print "Selected Customers Approved"
NO
If Choose 2
YES
Prompt selection of at least one customer
If Empty
YES
Print "Please select a customer to update"
NO
Prompt Field to Update:
1. Username
2. Password
3. Email
4. Phone
If Empty
YES
Print "Field cannot be empty"
NO
System Update Customer Account
Print "Selected Field Updated"
END
NO
If Choose 3
YES
Prompt selection of at least one customer
If Empty
YES
Print "Please select at least one customer to delete"
NO
Print "Are you sure you want to delete selected customers?"
If Confirm
No
Print "Failed to delete profile, please try again"
Yes
System Update Customer Account
Print "Selected Customer Deleted"
NO
If Choose 4
YES
print "You are back to the main page"
Manager Main Dashboard
NO

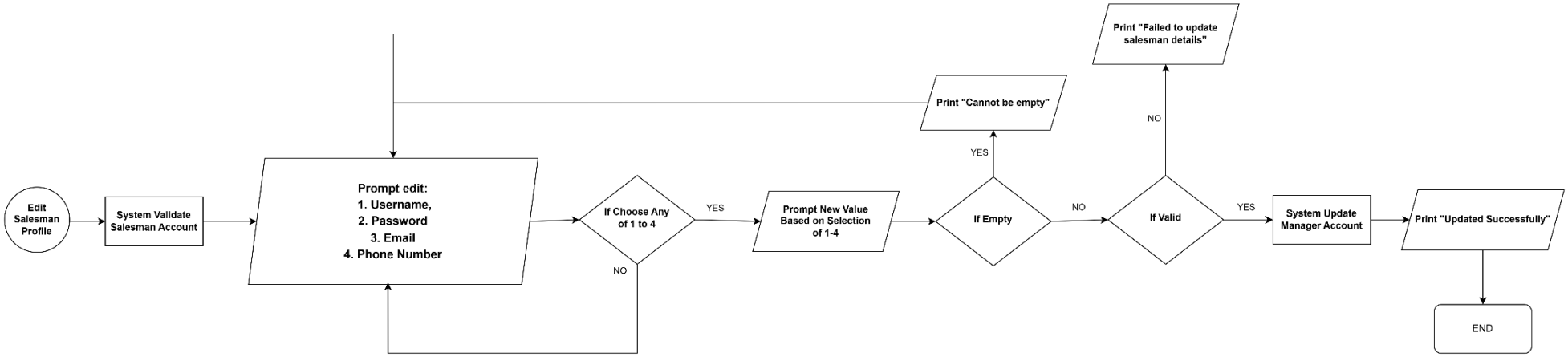

Manage Car Details
Prompt:
1. Add Car
2. Edit Car Details
3. Delete Car
4. Exit
If Choose 1
YES
NO
Prompt Car Brand, Model, Color, Price, Status
If CarID is empty
YES
NO
Print "CarID already exists! "
If Input Valid
YES
NO
Print "Field cannot be empty"
System Add New Car Details
Print "Car added successfully "
If Choose 2
YES
NO
Prompt selection of at least one car
If Empty
YES
NO
Print "Please select a car to update"
If Selection Valid
YES
NO
Print "Selected car not found"
Prompt Field to Update:
1. Car Brand 2. Model
3.Color 4.Price 5. Status
If Empty
YES
NO
Print "Field cannot be empty"
System Update Car Details
Print "Car Updated Successfully"
END
If Choose 3
YES
NO
Prompt selection of at least one car
If Empty
YES
NO
Print "Please select at least one car to delete"
Print "Are you sure you want to delete selected car?"
If Confirm
Yes
No
Print "Failed to delete car, please try again"
System Update Car
Print "Selected Car Deleted"
If Choose 4
YES
NO
print "You are back to the main page"
Manager Main Dashboard

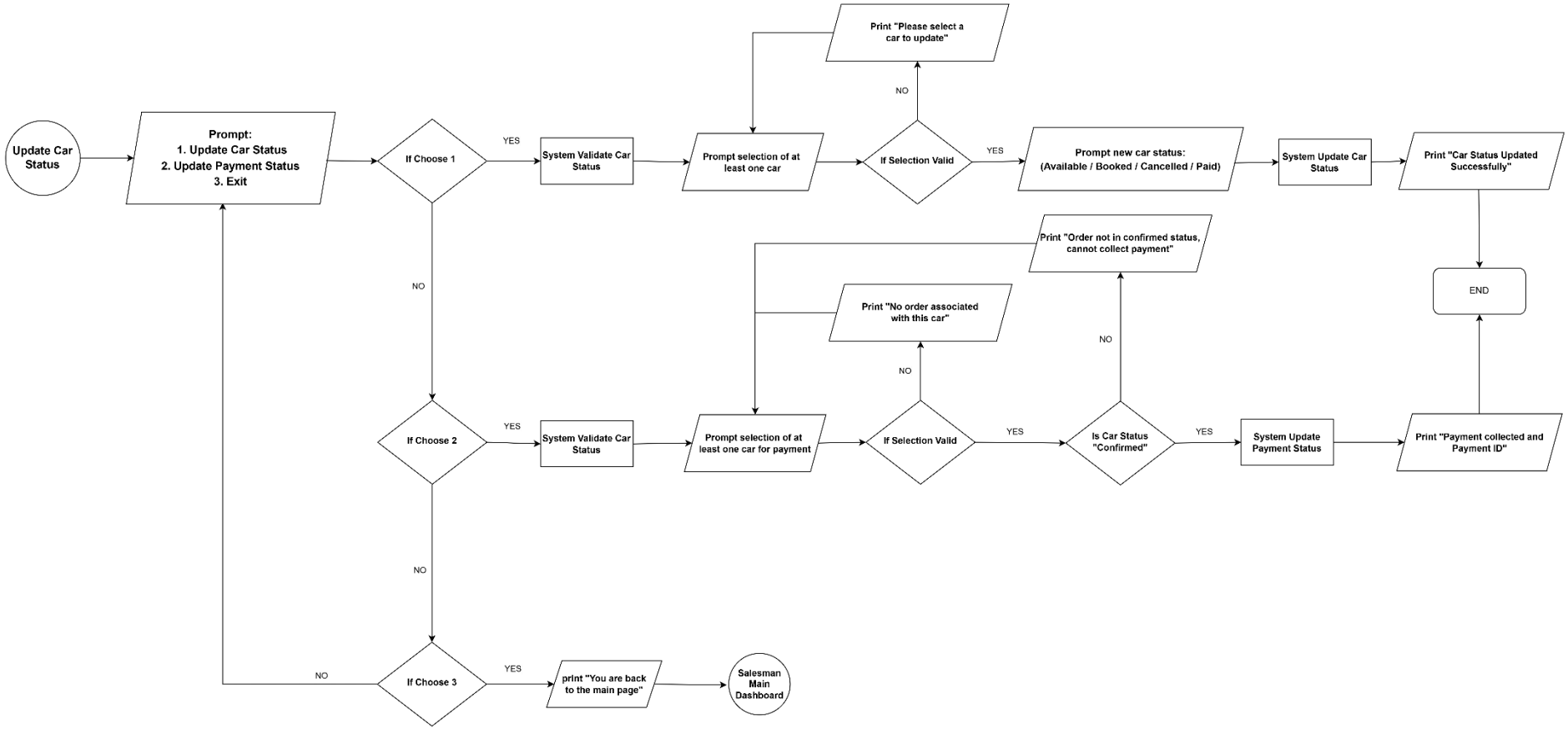

Payment and Feedback Records
Prompt:
1. View payment
2. View feedback
3. Exit
If Choose 1
YES
Load payment data from system
Is Record Available
YES
Show payment data
NO
Print "Failed to load records"
END
NO
Print "Failed to load records"
If Choose 2
YES
Load feedback data from system
Is Record Available
YES
Show feedback data
NO
NO
If Choose 3
YES
print "You are back to the main page"
Manager Main Dashboard

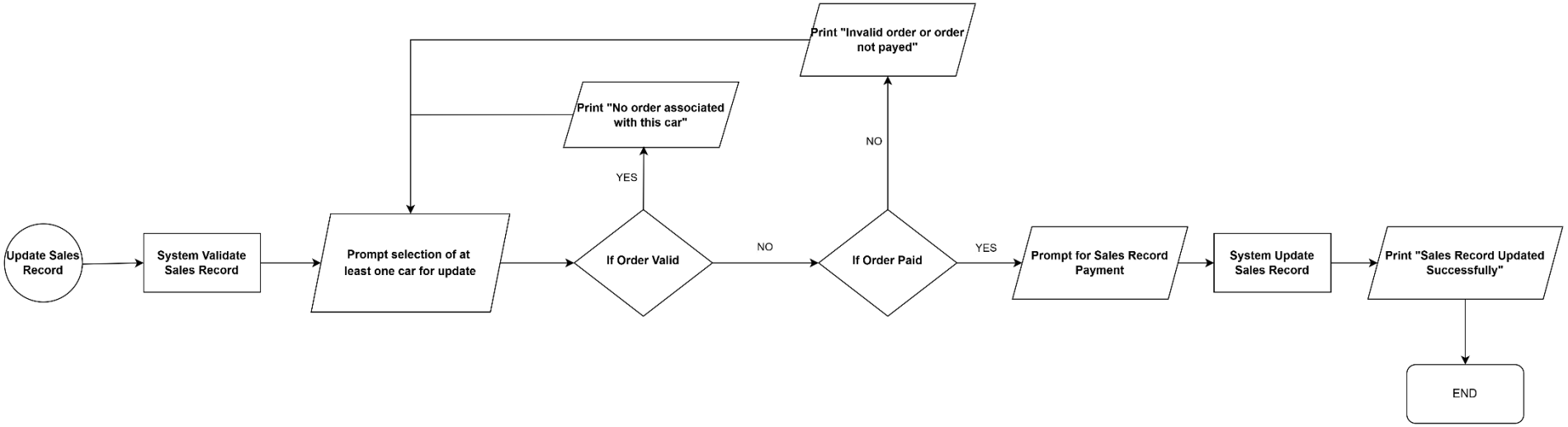
2.3 Salesman


Salesman Login
Display login form
If Press Cancel
YES
Print 'You are back to the main page"
User Main Page
NO
Prompt Username and Password
If Input Empty
YES
Print "Username and Password cannot be empty"
NO
System checks for salesman account
Text
If Salesman Account Exist
NO
Print "Error reading Salesman File"
YES
System validates credentials
If Credentials Valid
NO
Print " Invalid Username or Password"
YES
Print " Login Successful"
Salesman Main Dashboard

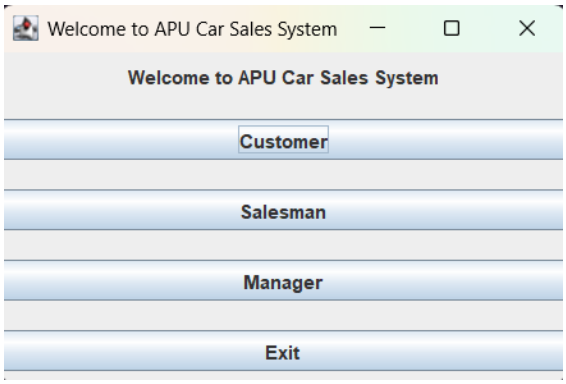
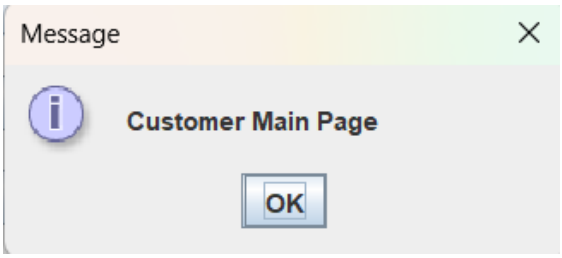

Salesman Main Dashboard
Prompt:
1. Edit Salesman Profile
2. Update Car Status
3. UpdateSales Record
4. Exit
If Choose 1
YES
Edit Salesman Profile
NO
If Choose 2
YES
Update Car Status
NO
If Choose 3
YES
Update Sales Record
NO
If Choose 4
NO
YES
print "You are back to the main page"
YES
User Main Menu


Edit Salesman Profile
System Validate Salesman Account
Prompt edit:
1. Username,
2. Password
3. Email
4. Phone Number
If Choose Any of 1 to 4
YES
NO
Prompt New Value Based on Selection of 1-4
If Empty
YES
Print "Cannot be empty"
NO
If Valid
NO
Print "Failed to update salesman details"
YES
System Update Manager Account
Print "Updated Successfully"
END


Update Car Status
Prompt:
1. Update Car Status
2. Update Payment Status
3. Exit
If Choose 1
YES
System Validate Car Status
Prompt selection of at least one car
If Selection Valid
NO
Print "Please select a car to update"
YES
Prompt new car status:
(Available / Booked / Cancelled / Paid)
System Update Car Status
Print "Car Status Updated Successfully"
END
NO
If Choose 2
YES
System Validate Car Status
Prompt selection of at least one car for payment
If Selection Valid
NO
Print "No order associated with this car"
YES
Is Car Status "Confirmed"
NO
Print "Order not in confirmed status, cannot collect payment"
YES
System Update Payment Status
Print "Payment collected and Payment ID"
NO
If Choose 3
YES
print "You are back to the main page"
Salesman Main Dashboard
NO


Update Sales Record
System Validate Sales Record
Prompt selection of at least one car for update
If Order Valid
YES
Print "No order associated with this car"
NO
If Order Paid
NO
Print "Invalid order or order not payed"
YES
Prompt for Sales Record Payment
System Update Sales Record
Print "Sales Record Updated Successfully"
END

3.0 Sample Output

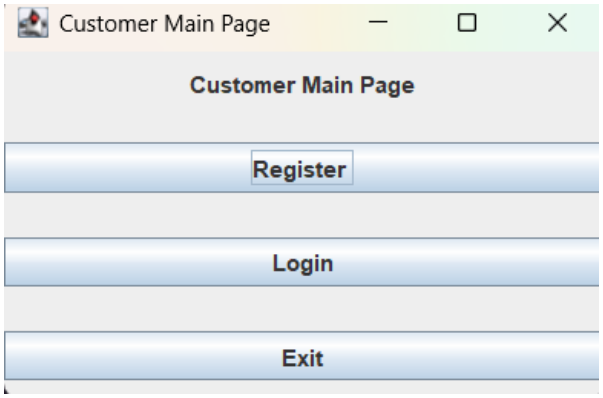
System Main Page	
	

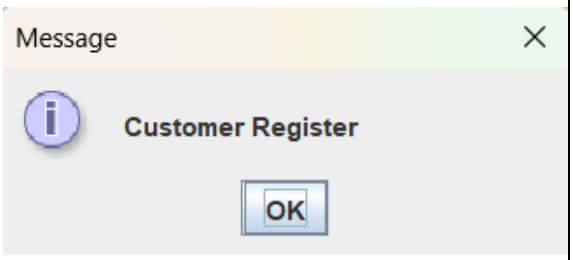
Upon running our code, you will be welcomed by our **System Main Page.** This page displays a header that reads **"Welcome to APU Car Sales System"** along with three icons on the top right for **minimize, full-screen, and close functions.** At the centre, we'll find four buttons labelled **Customer, Salesman, Manager, and an option to Exit**—each designed to navigate you to the corresponding main menus.

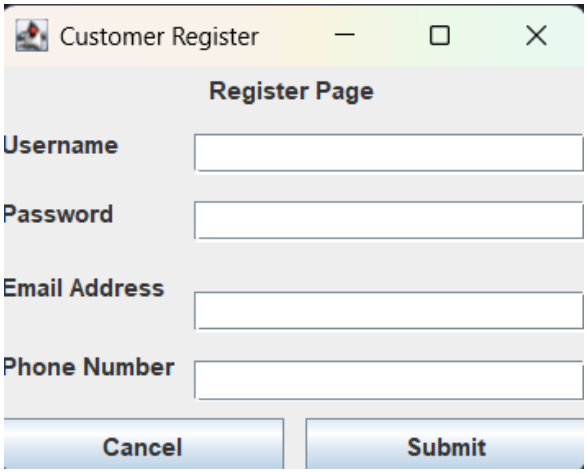
The second picture captures a pop-up message confirming the choice of the **Customer Main Page**, complete with an "**OK" button** to finalize the selection. This will bring us the Customer Main Page.

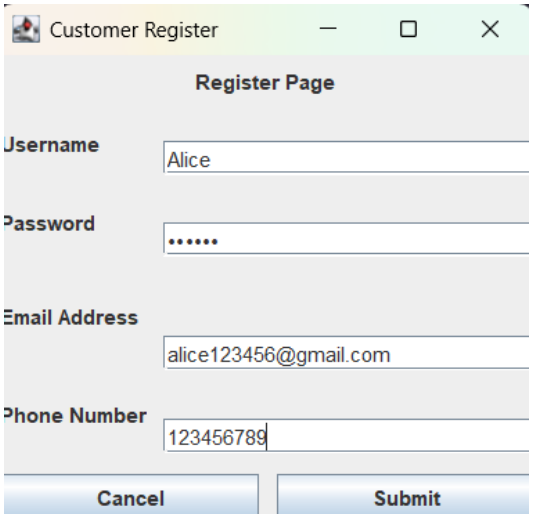
3.1 Customer

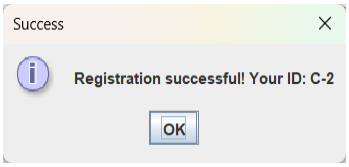
Register





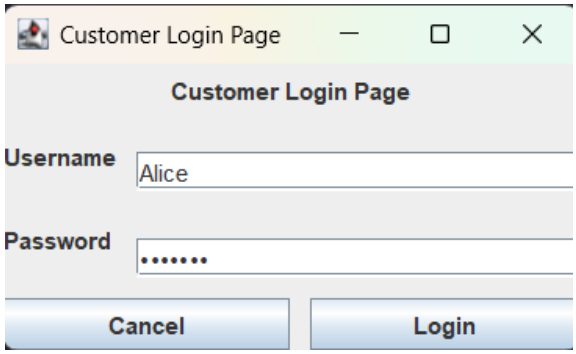


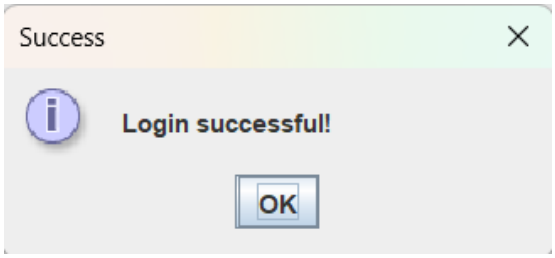




Here at the Customer Main Page, we have 3 buttons for **register, login and exit** respectively. Upon clicking the **"Register"** button, we'll be greeted with a pop-up message that confirms your choice if you press the OK button. We will be brought to the Register Page, where we enter our details such as **username, password, email address and phone number**. Note that the **Password** field is masked for security After entering our details and pressing the "Submit" button, the system confirms our successful registration with a pop-up message. This pop-up, titled "Success," displays an information icon alongside the message **"Registration Successful! Your ID: C-2,"** and includes an "OK" button to proceed.

Login

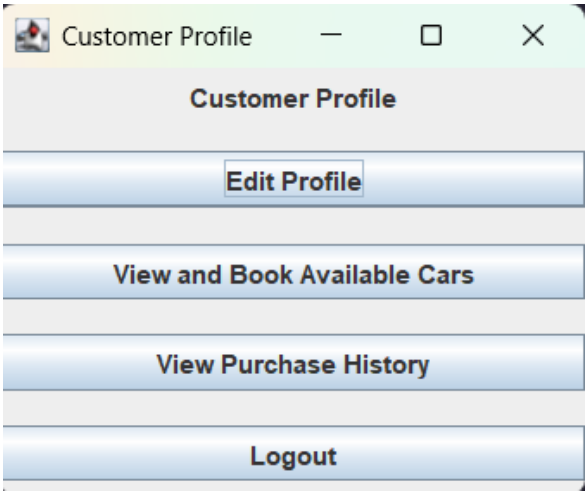




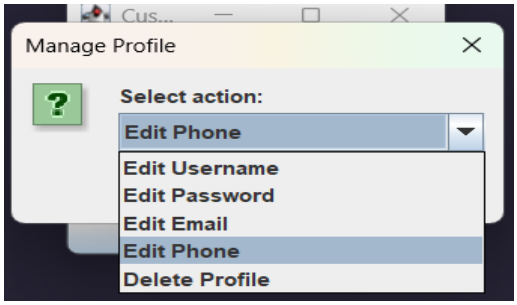
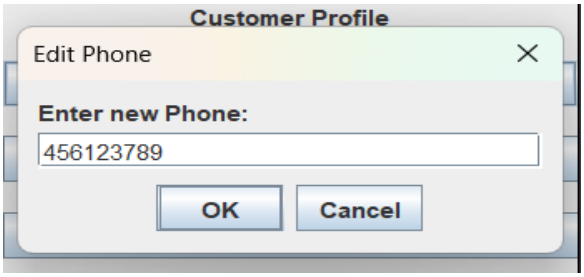
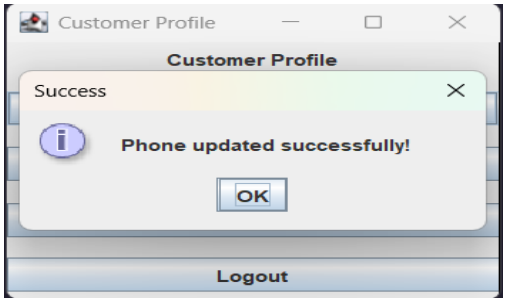
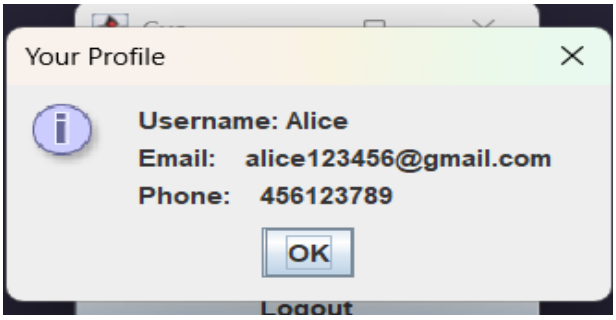
If we were to select **Login** on the Customer Main Page, you will be directed to the **Customer Login Page**. This page contains fields for **Username** and **Password**, allowing users to enter their credentials. Note that the **Password** field is masked for security. At the bottom, users can either **Cancel** the login process or proceed by clicking **Login** to access their account.

The second picture captures a **pop-up message** confirming the successful login of the customer. Below is an **"OK"** button to finalize the selection and will bring us the **Customer Profile Page.**

Customer Profile Page



Upon logging in, users are directed to the **Customer Profile** page, where they can manage their account and interactions within the system. The page features four key options: **Edit Profile**, **View and Book Available Cars**, **View Purchase History**, and **Logout**. Each button provides a clear pathway for users to update their details, browse available vehicles, review past transactions, or exit the system.

Edit Profile	
	
	

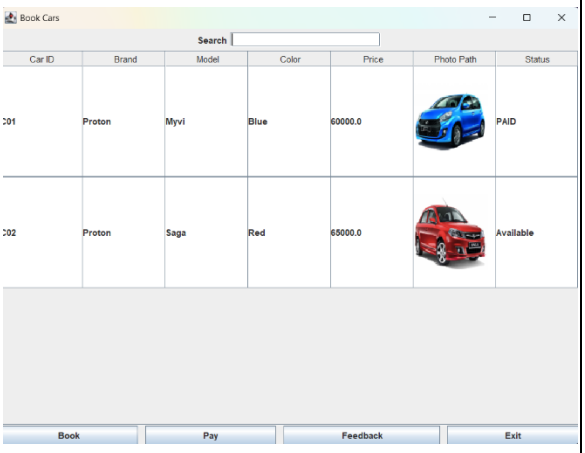
Upon selecting **Edit Profile** from the **Customer Profile** page, users are presented with a dropdown menu allowing them to update their details. The available actions include **Edit Username, Edit Password, Edit Email, Edit Phone, and Delete Profile**. The selected action appears in the dropdown, ensuring clear user interaction.

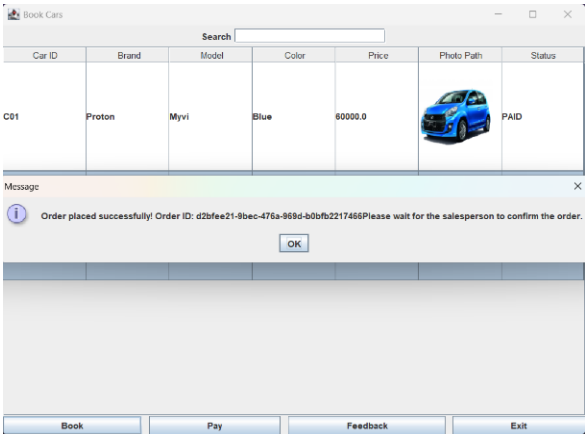
As an example, we will choose to **Edit Phone** from the dropdown menu. Users are presented with a pop-up window allowing them to update their phone number. The window prompts the user to enter a new phone number in a designated text box, with "456123789" typed in as an example. Below the input field, two buttons labelled **"OK"** and **"Cancel"** provide users with the option to confirm the change or exit without saving.

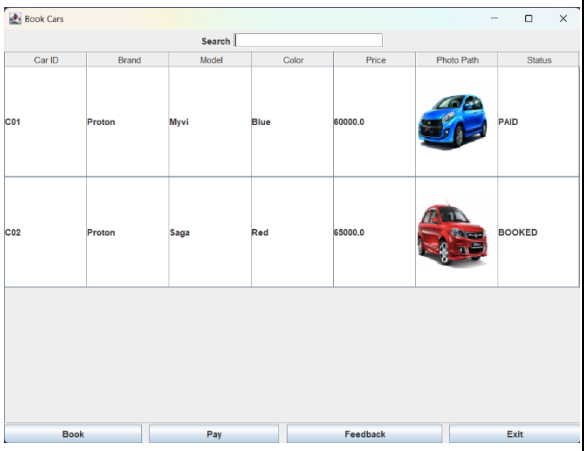
Upon successfully updating the phone number, a confirmation pop-up appears with the title **"Success."** This message displays the text **"Phone updated successfully!"** and the **"OK"** button to acknowledge the change.

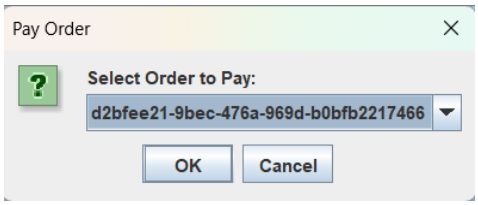
A confirmation pop-up for **Your Profile** allows users to view their account details, including **Username, Email, and Phone Number**. The displayed information confirms the registered credentials. In our case, the phone number has been successfully changed from the original 123456789 to 456123789. At the bottom, an **"OK"** button allows users to acknowledge and exit the profile view.

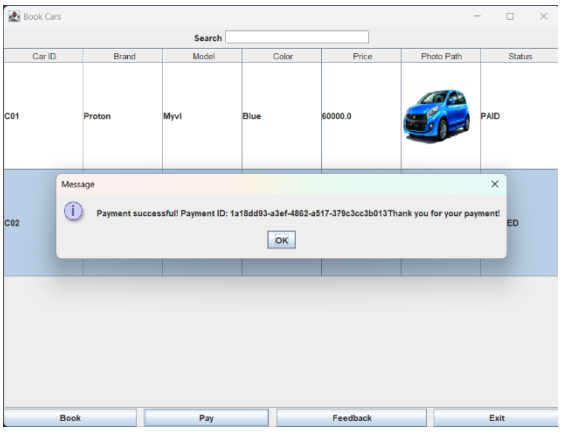
View and Book Available Cars

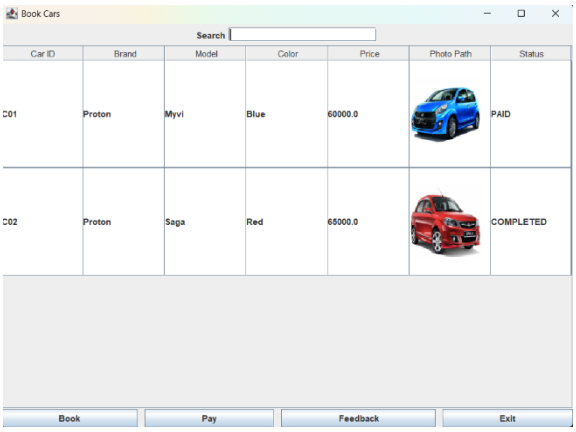


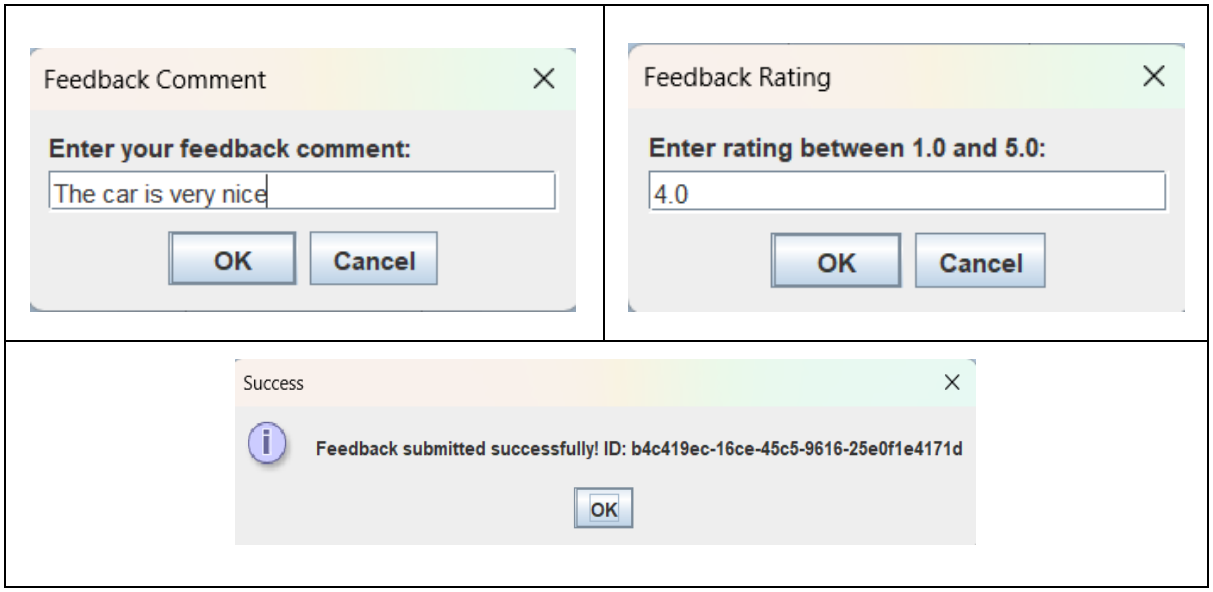












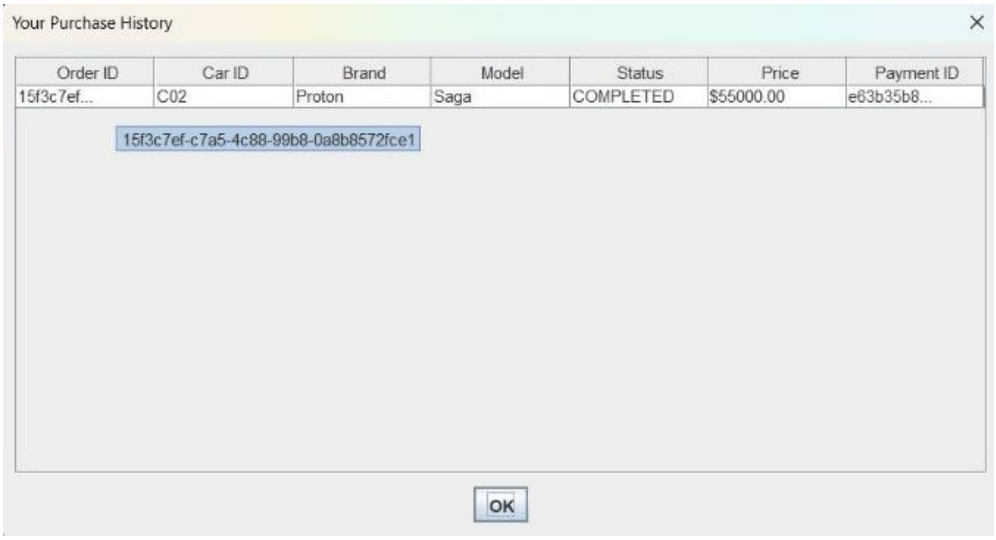
Upon accessing the **View and Book Available Cars Page**, users can view available cars in a structured table displaying details such as **Car ID, Brand, Model, Color, Price, Photo, and Status**. Each car entry provides essential information to help users make their selection. At the top, a **search bar** allows users to find specific vehicles quickly. At the bottom, four buttons—**Book, Pay, Feedback, and Exit** each enable users to proceed with their chosen action.

Upon selecting "**Book Button"** for a selected car, a confirmation pop-up appears with the message **"Order placed successfully!"** The dialog provides an **Order ID**, ensuring users can track their purchase. It also informs users to **wait for the salesman to confirm the order**, reinforcing a structured process. After salesman confirms the order, the **status will become Booked**.

Upon selecting "**Pay Button"**, a dialog box titled **"Pay Order"** appears, allowing users to make their payment. The interface includes a dropdown menu labelled **"Select Order to Pay:"**, where users can choose their specific order ID. The displayed order ID is **"d2bfee21-9bec-476a-969d-bb0fb2217466."** Below the dropdown, two buttons—**"OK"** and **"Cancel"**—let users confirm or cancel the payment process. Then, a confirmation pop-up appears with the message **"Order placed successfully!"** will appear. The status of the car will then change to **Completed**.

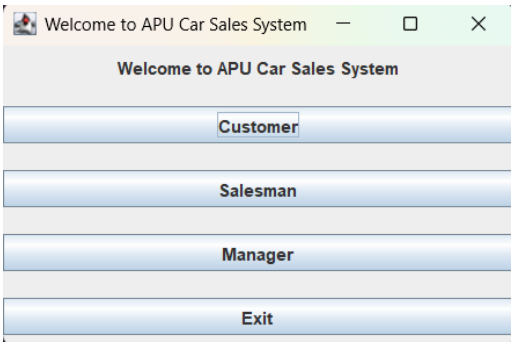
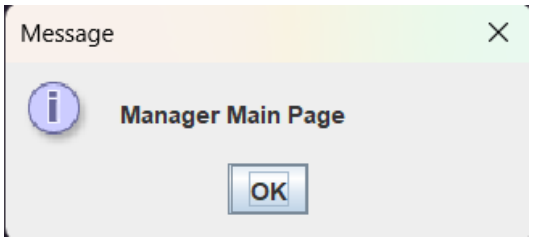
Upon selecting “**Feedback Button”,** a pop-up window appears, allowing users to provide their thoughts on their purchased car. The dialog prompts users with **"Enter your feedback comment:"** and includes a **text box** where they can type their response. Then, a pop-up window appears prompting users to enter a rating between **1.0 and 5.0** with an input field that is **pre-filled with 4.0**, still allowing users to **modify their rating** if desired. Upon submitting feedback with the “OK” button, a confirmation pop-up appears and displays "**Feedback submitted successfully!**” with the respective ID, ensuring users receive an acknowledgment for their input.

View Purchase History

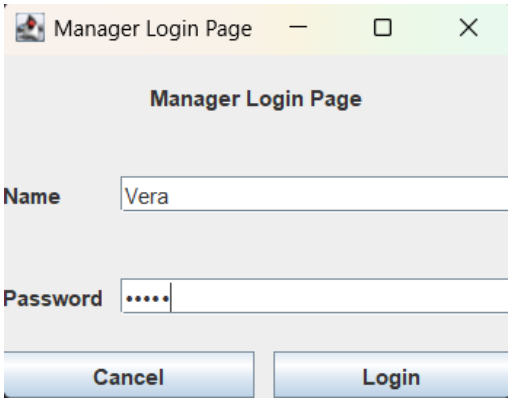
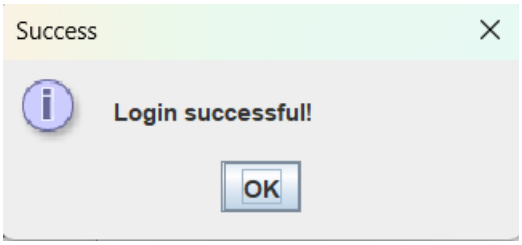


Upon accessing the **Purchase History** page, users can review details of their completed car orders in a structured table. The table displays essential information, including **Order ID, Car ID, Brand, Model, Status, Price, and Payment ID.** At the bottom, an **"OK"** button allows users to acknowledge and exit the history view, maintaining a smooth user experience.

3.2 Admin

System Main Page	
	

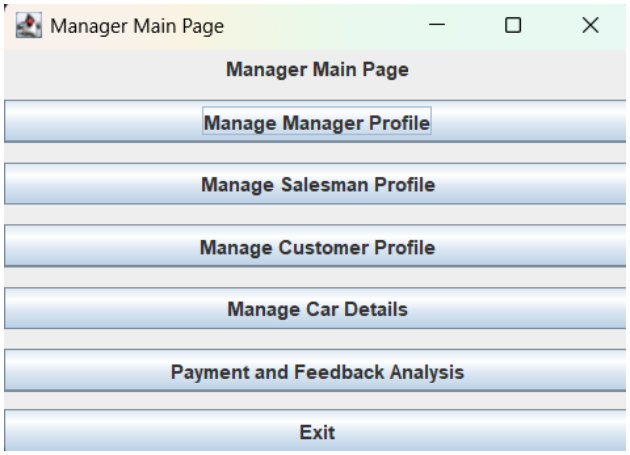
Back at by our system main page, we can select "**Manager Button"** on the system main page to be directed to the **Manager Login Page**. There will be a **pop-up message** confirming the choice of Manager, complete with an **"OK"** button to finalize the selection.

Manager Login	
	

Here is the **Manager Login Page**. This page contains fields for **Username** and **Password**, allowing users to enter their credentials. Note that the **Password** field is masked for security. At the bottom, users can either **Cancel** the login process or proceed by clicking **Login** to access their account.

The second picture captures a **pop-up message** confirming the successful login of the manager. Below is an **"OK"** button to finalize the selection and will bring us the **Manager Main Page.**

Manager Main Page

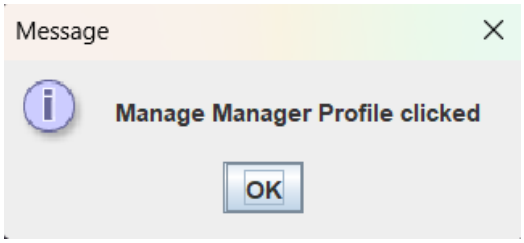


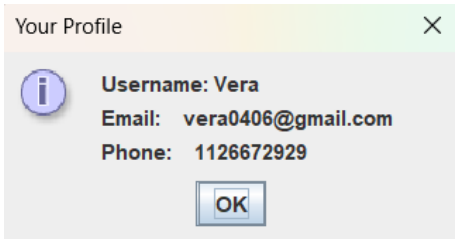
Upon accessing the **Manager Main Page**, users are presented with several management options in the form of buttons. The available functions include:

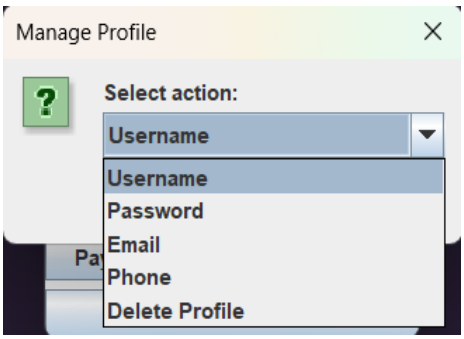
1. **Manage Manager Profile** – Allows managers to update their profile details.
2. **Manage Salesman Profile** – Enables management of salesman accounts and details.
3. **Manage Customer Profile** – Enables management of customer accounts and details.
4. **Manage Car Details** – Facilitates car inventory management.
5. **Payment and Feedback Analysis** – Offers insights into transactions and customer reviews.
6. **Exit** – Exit this page.

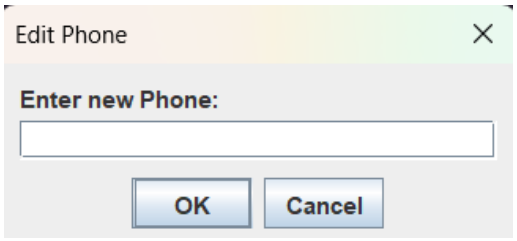
The structured layout ensures efficient navigation, allowing managers to oversee different aspects of the system seamlessly.

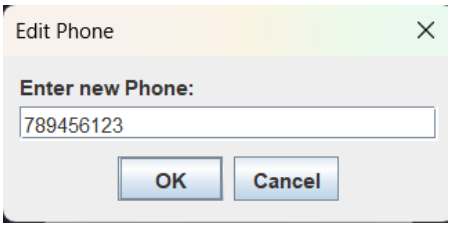
Manage Manager Profile

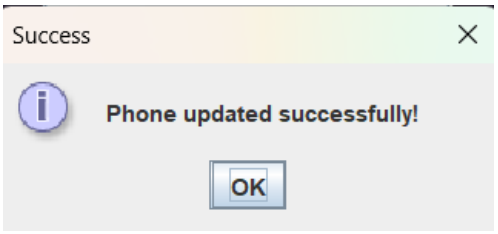


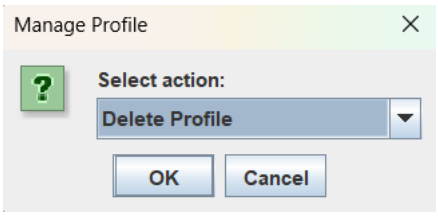


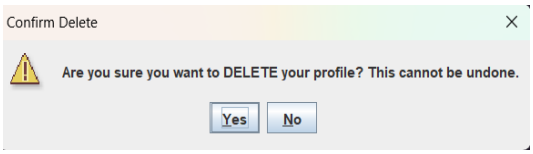


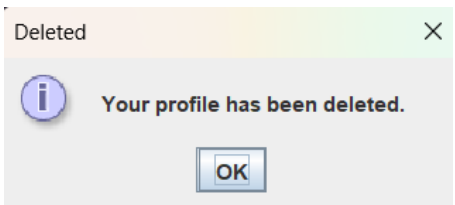












Upon selecting **Manage Manager Profile** from the **Manager Main** page, users are presented with a pop-up notification that displays their information such as **username, email and phone number.** Then, **a dropdown menu** will appear, allowing users to update their details. The available actions include **Edit Username, Edit Password, Edit Email, Edit Phone, and Delete Profile**. The selected action appears in the dropdown, ensuring clear user interaction.

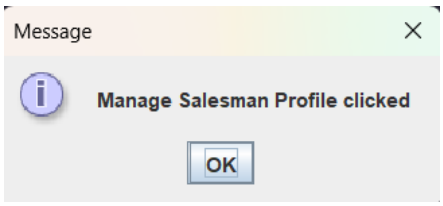
As an example, we will choose to **Edit Phone** from the dropdown menu. Users are presented with a pop-up window allowing them to update their phone number. The window prompts the user to enter a new phone number in a designated text box, with "789456123" typed in as an example. Below the input field, two buttons labelled **"OK"** and **"Cancel"** provide users with the option to confirm the change or exit without saving.

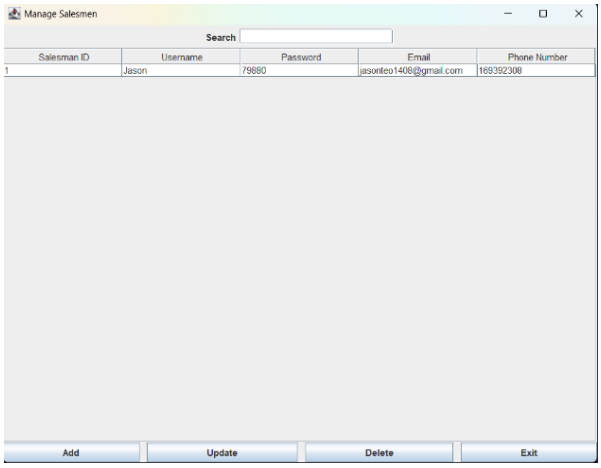
Upon successfully updating the phone number, a confirmation pop-up appears with the title **"Success."** This message displays the text **"Phone updated successfully!"** and the **"OK"** button to acknowledge the change.

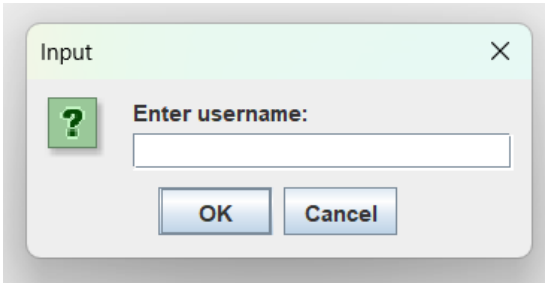
Upon selecting **Delete Profile** from the **dropdown menu**, the dialog prompts the user with two buttons labelled **"OK"** and **"Cancel",** allowing users to confirm or exit the process. Then, a confirmation pop-up titled **"Confirm Delete"** appears. It displays **a warning icon** alongside the message: **"Are you sure you want to DELETE your profile? This cannot be undone."** Below, two buttons labelled **"Yes"** and **"No"** provide users with the option to proceed with deletion or cancel the action.

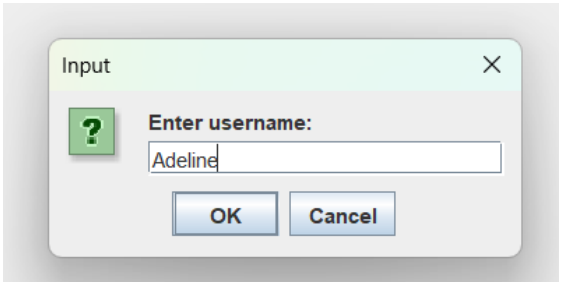
Upon successfully deleting the profile, a confirmation pop-up appears with the title **"Success."** The dialog displays a message: **"Profile deleted successfully!"** Below the message, an **"OK"** button allows users to confirm and proceed. This final confirmation ensures users are aware that their profile has been removed from the system

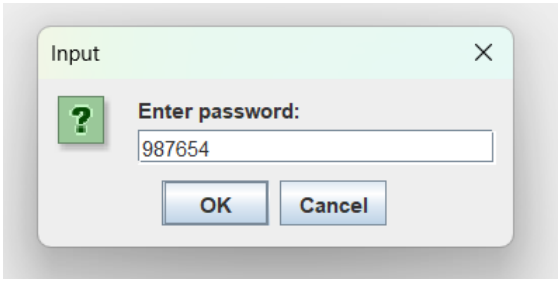
Manage Salesman Profile

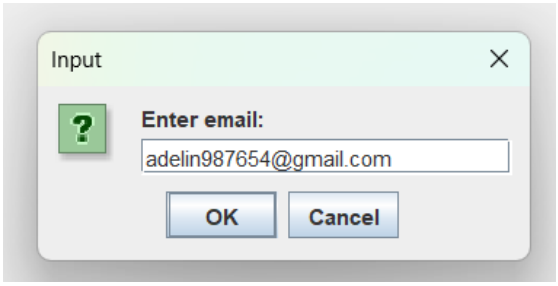


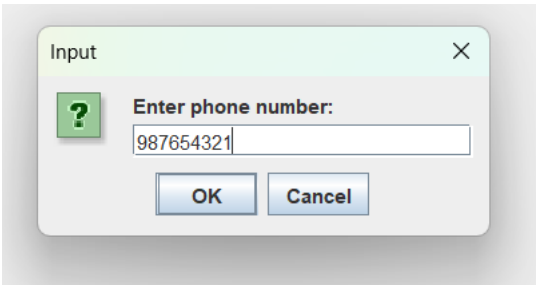


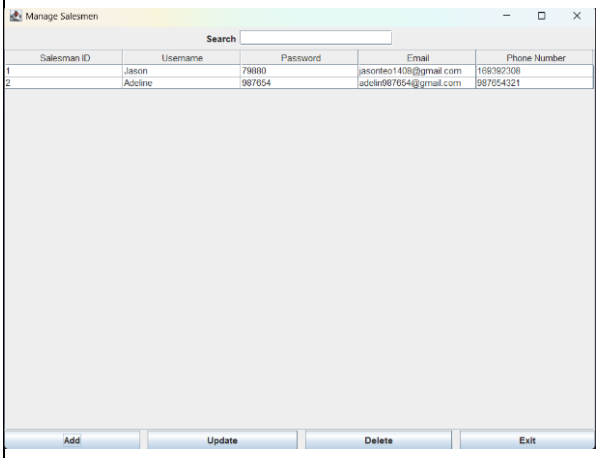


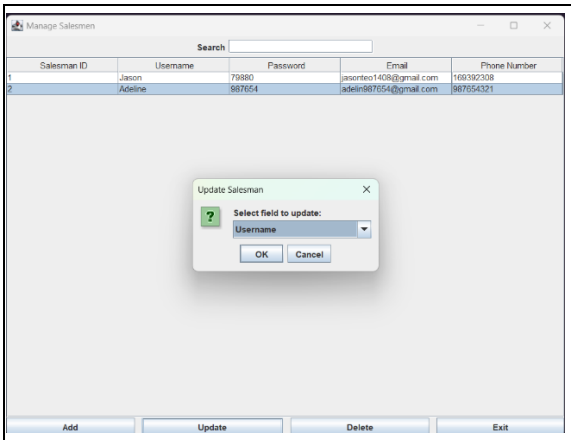


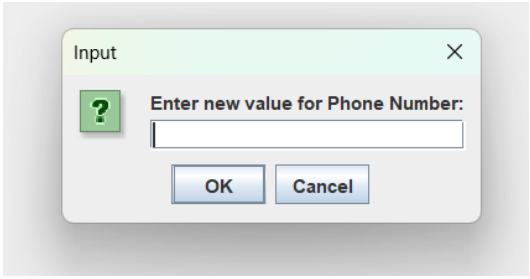


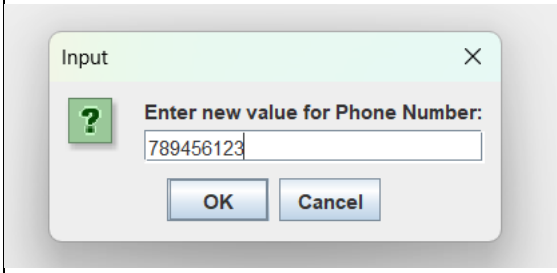


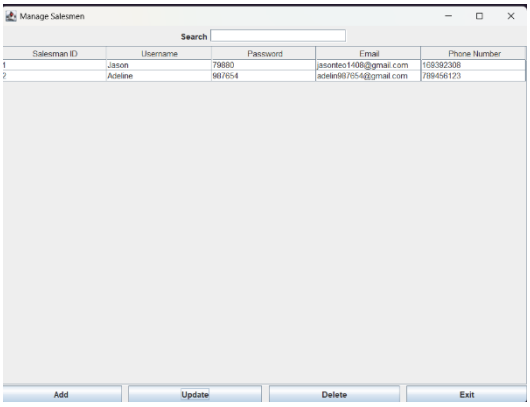


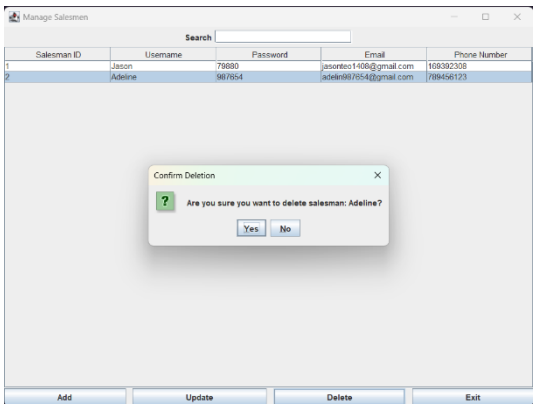


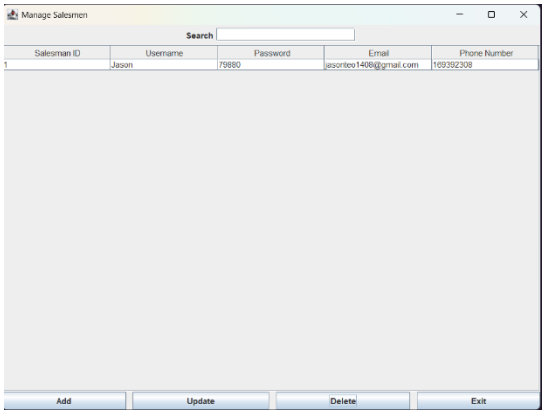

Manage Salesmen
Search
Salesman ID
Username
Password
Email
Phone Number
1
Jason
79880
jasonteo1408@gmail.com
169392308
2
Adeline
987654
adelin987654@gmail.com
987654321
Update Salesman
Select field to update:
Username
OK
Cancel
Add
Update
Delete
Exit


Input
Enter new value for Phone Number:
OK
Cancel


Input
Enter new value for Phone Number:
789456123
OK
Cancel


Manage Salesmen
Search
Salesman ID
Username
Password
Email
Phone Number
1
Jason
79880
jasonteo1408@gmail.com
169392308
2
Adeline
987654
adelin987654@gmail.com
789456123
Add
Update
Delete
Exit


Manage Salesmen
Search
Salesman ID
Username
Password
Email
Phone Number
1
Jason
79880
jasonteo1408@gmail.com
169392308
2
Adeline
987654
adelin987654@gmail.com
789456123
Confirm Deletion
Are you sure you want to delete salesman: Adeline?
Yes
No
Add
Update
Delete
Exit


Manage Salesmen
Search
Salesman ID
Username
Password
Email
Phone Number
1
Jason
79880
jasonteo1408@gmail.com
169392308
Add
Update
Delete
Exit

Upon accessing the **Manage Salesman Profile** page, users can oversee salesman details in a structured table format. The interface includes a **search bar** at the top for easy lookup. Below, the table displays essential information with columns such as **Salesman ID, Username, Password, Email, and Phone Number**. In this instance, there is already an entry for **Aaron** listed with relevant credentials. At the bottom, four buttons which are **Add, Update, Delete, and Exit**, allow users to modify, remove, or add new salesperson records efficiently.

Upon selecting the **“Add Button”**, a pop-up window appears prompting users to add a new salesman. The pop-up window has a label **"Enter username:"** and a text input field, which we will enter **"Adeline."** Below the input box, two buttons labelled **"OK"** and **"Cancel"** allow users to confirm or exit the username change process.

Then, the next pop-up window appears and prompts users to enter a **new password**. It has the label **"Enter password:"** and a text input field, which we will enter the following password **"987654."** Below, two buttons labelled **"OK"** and **"Cancel"** allow users to confirm or exit the password change process.

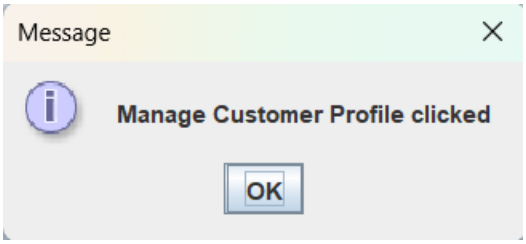
Then, a pop-up window appears prompting users to enter a **new email address**. It has the label "Enter email:" and a text input field, which we input the email **"adelin987654@gmail.com."** Below the input box, two buttons labelled **"OK"** and **"Cancel"** allow users to confirm or exit the email update process.

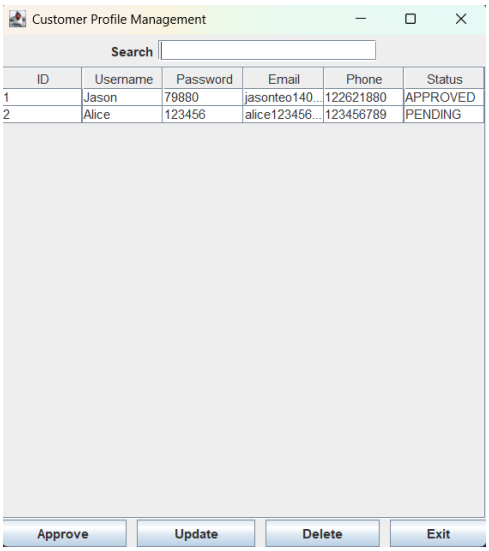
Lastly, a pop-up window that prompts users to enter a **new phone number.** After inputting the number **"987654321** ", we can press either the **"OK"** or **"Cancel"** button to confirm or exit the last process. If **“OK” is chosen**, all the credentials will be officially saved in the **Manage Salesman Profile** page as seen above.

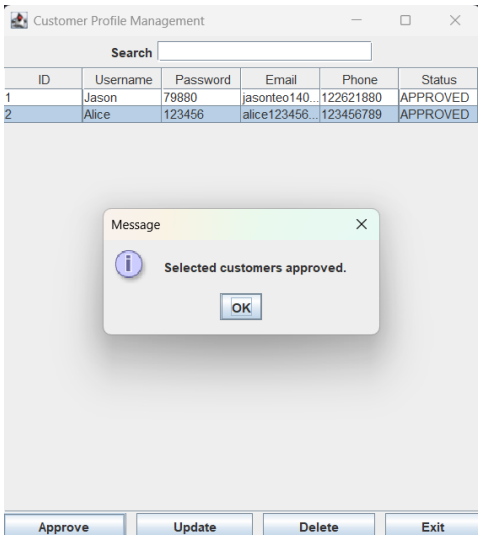
Upon selecting the one of the salesman entries**,** and then the **“Update Button”,** users are presented with a dropdown menu allowing them to update the Salesman details. As an example, we will choose to **Edit Phone** from the dropdown menu and enter a new phone number in a designated text box, with **"789456123"** typed in as an example. Below the input field, two buttons labelled **"OK"** and **"Cancel"** provide users with the option to confirm the change or exit without saving. As we can see above, the changes are saved in the **Manage Salesman Profile** page.

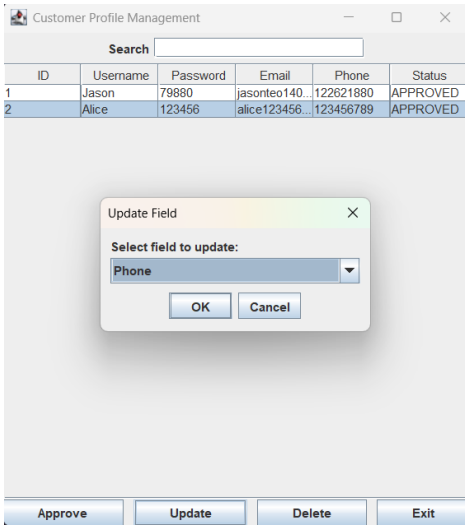
Upon selecting one of the salesman entries, and then selecting the “**Delete Button”**, a confirmation pop-up titled **"Confirm Delete"** appears. It displays a message: **"Are you sure you want to delete Salesman: Adeline?"** Below, two buttons labelled **"Yes"** and **"No"** provide users with the option to proceed with deletion or cancel the action. As seen above, if we proceed with “Yes”, we can see that the relevant information of Salesman Adeline has been **erased**.

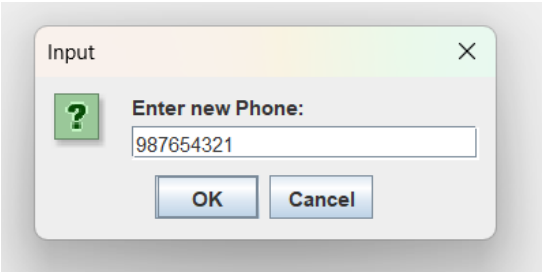
Manage Customer Profile

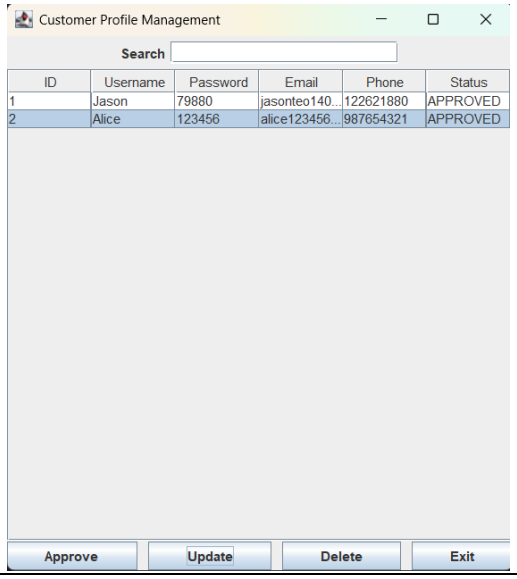


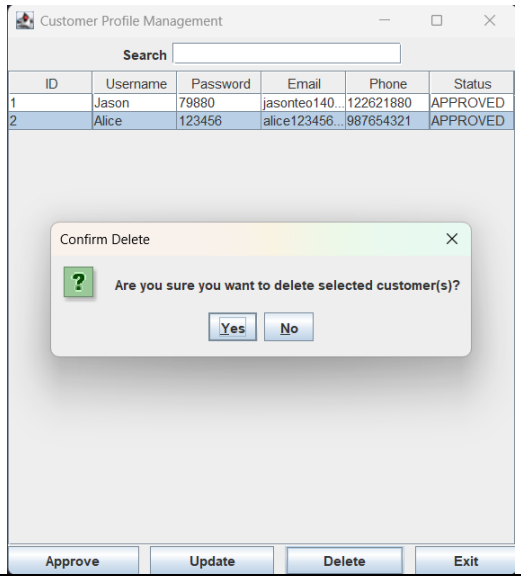


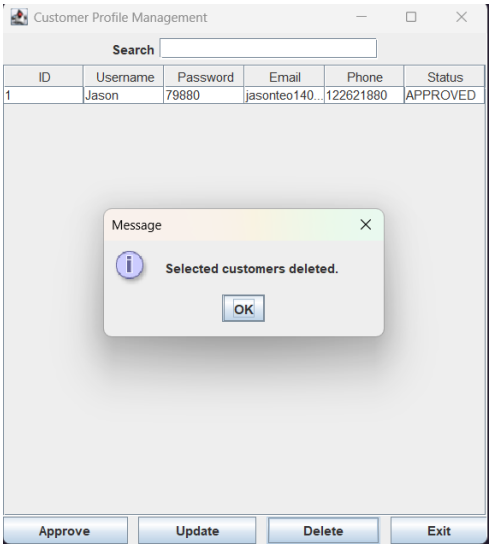












Upon accessing the **Customer Profile Management** page, users can oversee customer details in a structured table format. The interface includes a **search bar** at the top for easy lookup. Below, the table displays essential information with columns such as **Customer ID, Username, Password, Email, Phone Number and Status**. In this instance, there is already an entry for **Jason and Alice** listed with relevant credentials. At the bottom, four buttons which are **Approve, Update, Delete, and Exit**, allow users to modify, remove, or add new customer records efficiently.

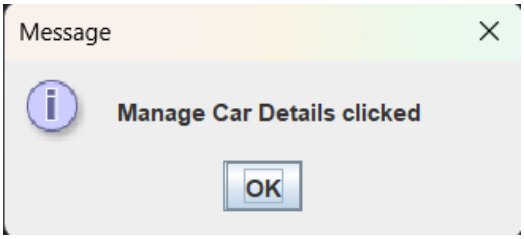
Upon selecting the customer Alice, and then the **“Approve Button”,** the status of the customer Alice will **change from pending to approve**d. A pop-up message appears with the text "Selected customers approved” and an "OK" button that allows users to acknowledge the approval of customer status.

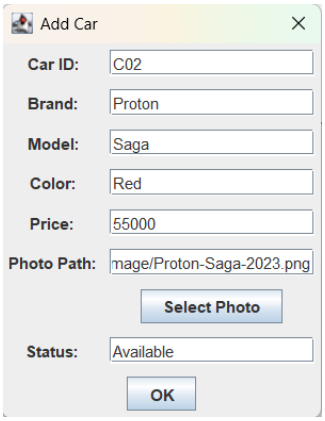
Upon selecting the **“Update Button”,** users are presented with a dropdown menu allowing them to update the Customer details. As an example, we will choose to **Edit Phone** from the dropdown menu and enter a new phone number in a designated text box, with **"987654321"** typed in as an example. Below the input field, two buttons labelled **"OK"** and **"Cancel"** provide users with the option to confirm the change or exit without saving. As we can see above, the changes are saved in the **Manage Customer Profile** page.

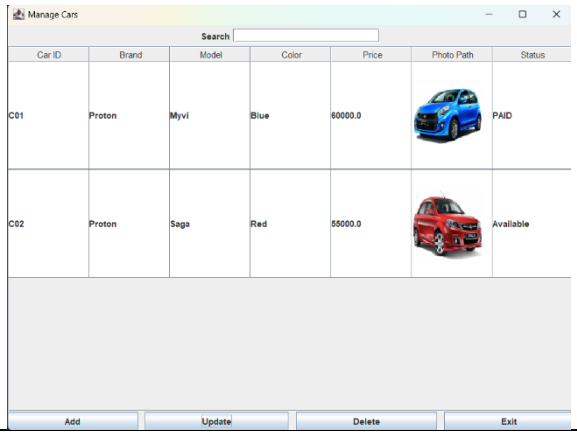
Upon selecting one of the customer entries, and then selecting the “**Delete Button”**, a confirmation pop-up titled **"Confirm Delete"** appears. It displays a message: **"Are you sure you want to delete selected customer(s)?"** Below, two buttons labelled **"Yes"** and **"No"**

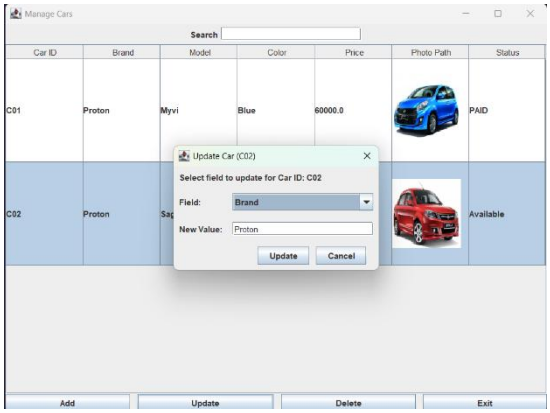
provide users with the option to proceed with deletion or cancel the action. As seen above, if we proceed with “Yes”, we can see that the relevant information of **customer Alice has been erased** along with a pop-up message that allows users to acknowledge the removal of customer.

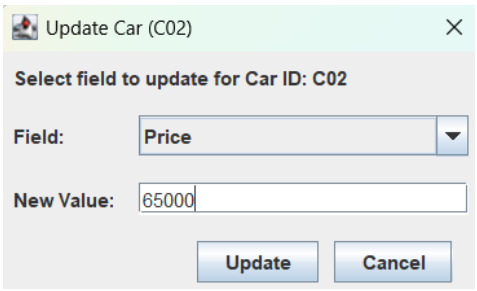
Manage Car Details

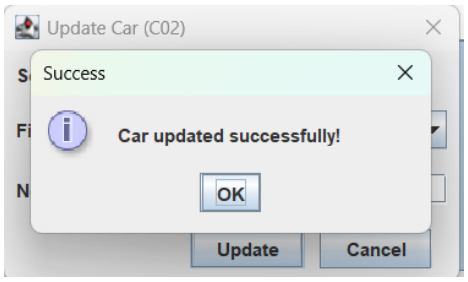


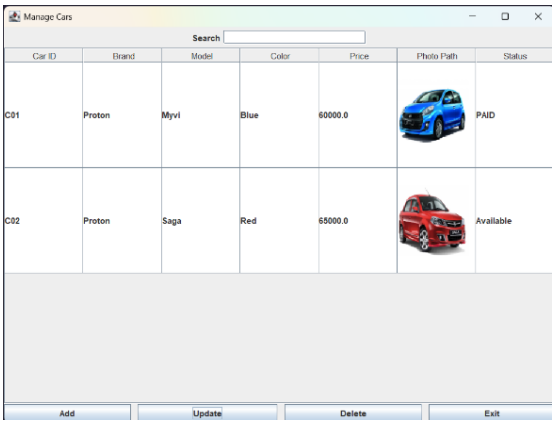


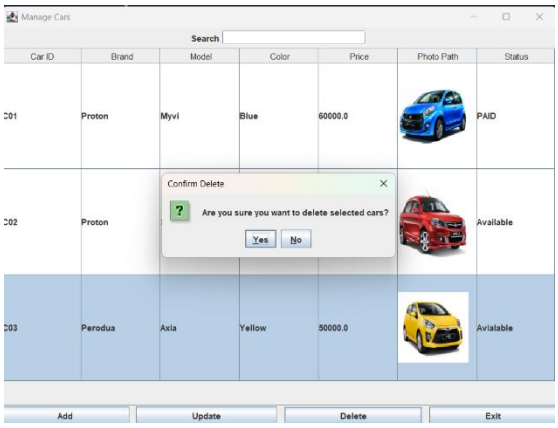


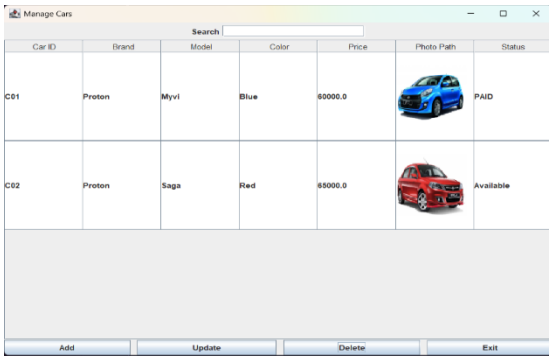












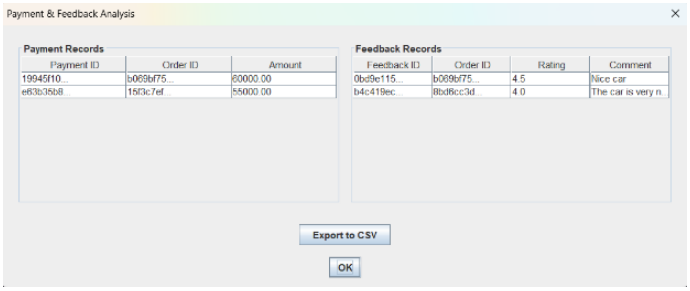
Upon accessing the **Manage Car** page, users can oversee car details in a structured table format. The interface includes a **search bar** at the top for easy lookup. Below, the table displays essential information with columns such as **Car ID, Brand, Model, Colour, Price, Photo Path, and Status.** At the bottom, four buttons which are **Add, Update, Delete, and Exit**, allow users to modify, remove, or add new car records efficiently.

Upon selecting "**Add Button"**, a dialog box appears prompting users to input details about a new vehicle. The form includes text fields for: Car ID, Model, Colour, Price, Photo Path and Status. Additionally, a button labelled **"Select Photo"** enables users to upload an image of the car, while the **"OK" button** confirms the entry. If **"OK" is chosen**, all the credentials will be officially saved in the **Manage Cars** page as seen above.

Upon selecting one of the car entries and **Update Car,** a pop-up window appears allowing users to modify specific details of the selected vehicle. The drop-down menu prompts users to choose a **field to update**, with the current selection set to **Brand**. The text input field contains **"Proton"** as the new value. The same can be done to **Price**, with the text field input having a **new value** of **65000**. Below, two buttons labelled **"Update"** and **"Cancel"** provide users with the option to confirm or exit the update process. As seen above, if we proceed with "Update", we can see that the relevant information of the car has been updated along with a pop-up message that allows users to acknowledge the update.

Upon selecting one of the car entries, and then selecting the "**Delete Button"**, a confirmation pop-up titled **"Confirm Delete"** appears. It displays a message: **"Are you sure you want to delete selected cars?"** Below, two buttons labelled **"Yes"** and **"No"** provide users with the option to proceed with deletion or cancel the action. As seen above, if we proceed with "Yes", we can see that the relevant information of **the Yellow Perodua has been erased** along with a pop-up message that allows users to acknowledge the removal of the car.

Payment and Feedback Analysis



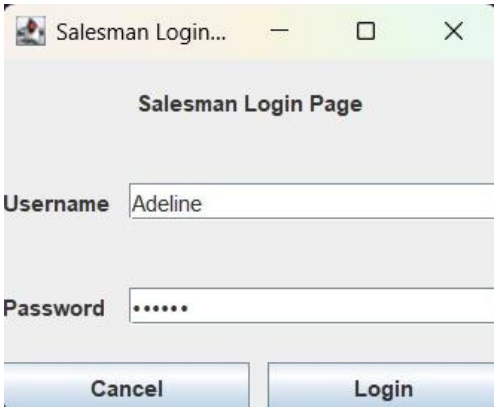
Upon accessing the **Payment & Feedback Analysis page,** the window presents structured financial and customer satisfaction data in two sections.

Payment Records include: **Payment ID, Order ID and Amount**

Feedback Records include: F**eedback ID, Order ID, Rating and Comment**

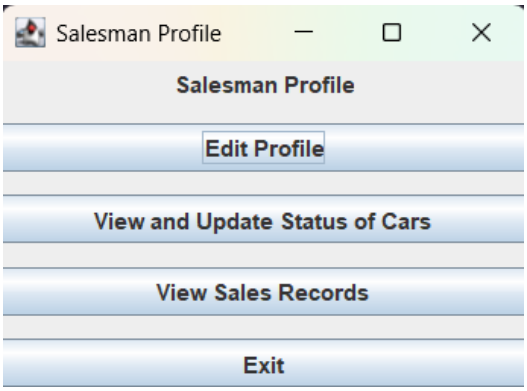
At the bottom, an **"Export to CSV"** button allows users to download the data, ensuring easy analysis and record-keeping. An **"OK"** button provides a simple way to exit the screen, maintaining a streamlined user experience.

3.3 Salesman

Salesman Login	
	

Back at by our system main page, we can select "**Salesman Button"** on the system main page to be directed to the **Salesman Login Page**. There will be a **pop-up message** confirming the choice of Manager, complete with an **"OK"** button to finalize the selection.

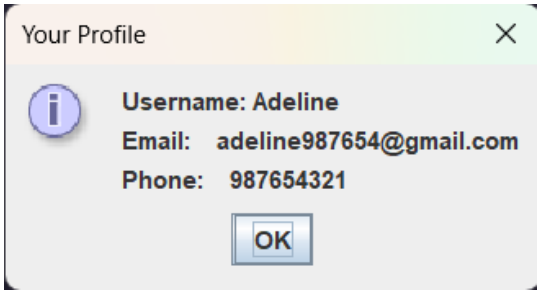
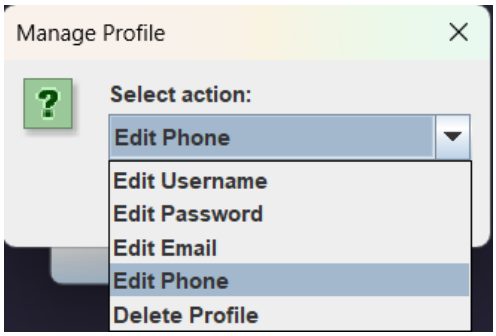
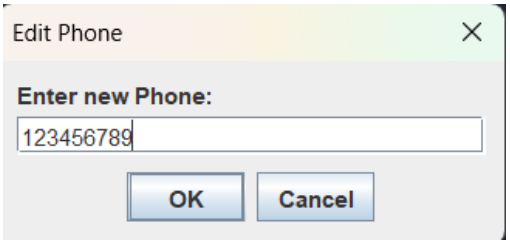
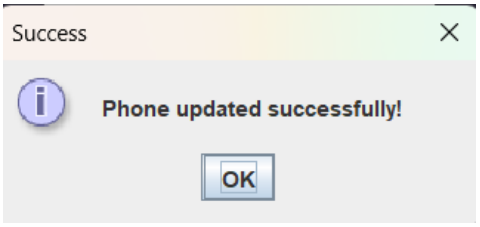
Here is the **Salesman Login Page**. This page contains fields for **Username** and **Password**, allowing users to enter their credentials. Note that the **Password** field is masked for security. At the bottom, users can either **Cancel** the login process or proceed by clicking **Login** to access their account. Clicking "Login" will bring us the **Salesman** Profile Page.

Salesman Profile


Upon accessing the **Salesman Profile** page, users are presented with four buttons corresponding to their key functionalities:

1. **Edit Profile** – Allows users to update their personal details.
2. **View and Update Status of Cars** – Provides an interface to manage vehicle availability and attributes.
3. **View Sales Records** – Displays transaction details related to car sales.
4. **Exit** – Closes the interface and returns to the main menu.

This structured layout ensures streamlined navigation for sales representatives managing their profiles and business activities.

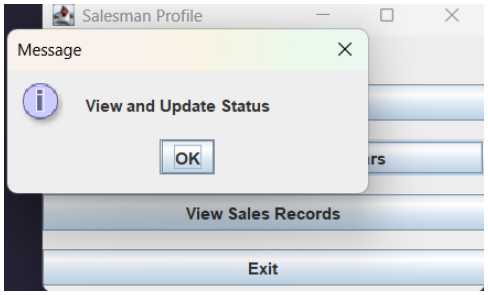
Edit Profile	
	
	

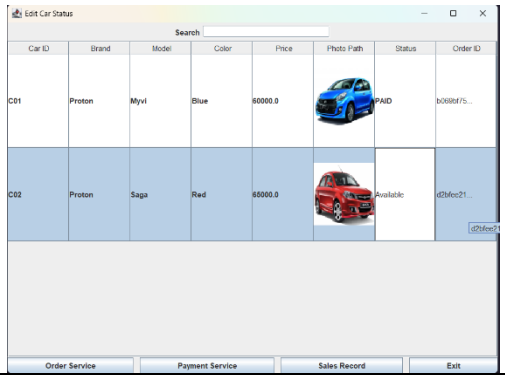
Upon selecting **Edit Profile** from the **Salesman Profile** page, users are presented with a pop-up notification that displays their information such as **username, email and phone number.** Then, dropdown menu will appear, allowing them to update their details. The available actions include **Edit Username, Edit Password, Edit Email, Edit Phone, and Delete Profile**. The selected action appears in the dropdown, ensuring clear user interaction.

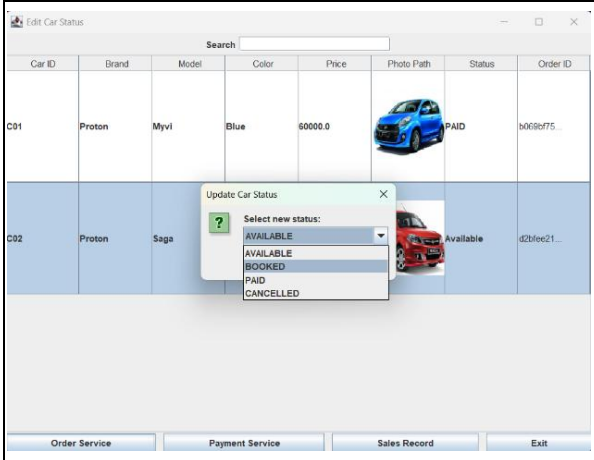
As an example, we will choose to **Edit Phone** from the dropdown menu. Users are presented with a pop-up window allowing them to update their phone number. The window prompts the user to enter a new phone number in a designated text box, with "123456789" typed in as an example. Below the input field, two buttons labelled **"OK"** and **"Cancel"** provide users with the option to confirm the change or exit without saving.

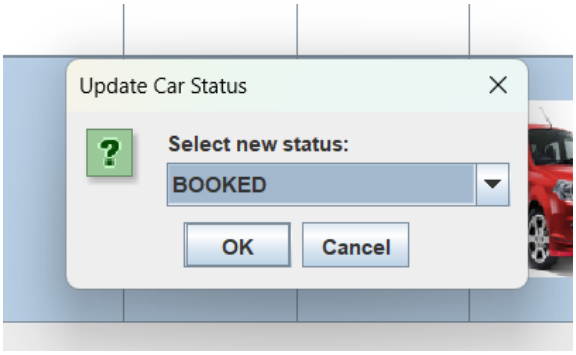
Upon successfully updating the phone number, a confirmation pop-up appears with the title **"Success."** This message displays the text **"Phone updated successfully!"** and the **"OK"** button to acknowledge the change.

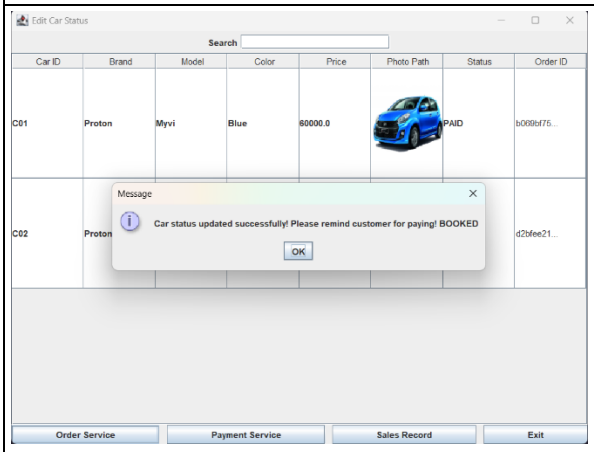
View and Update Status of Cars

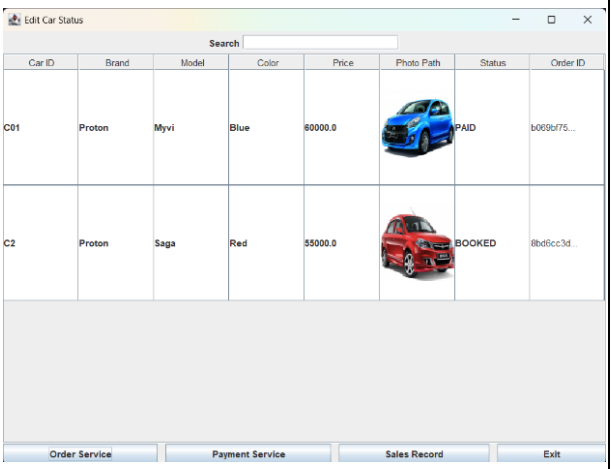


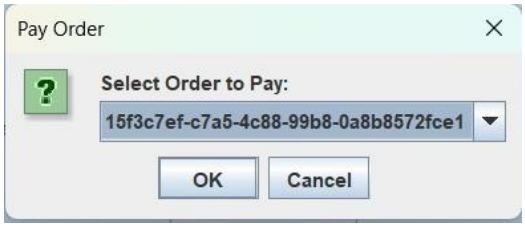


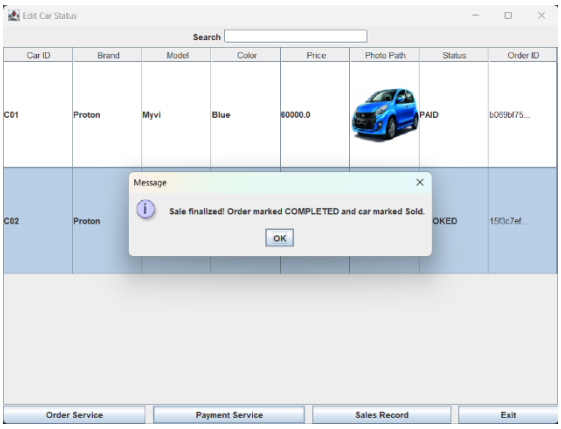


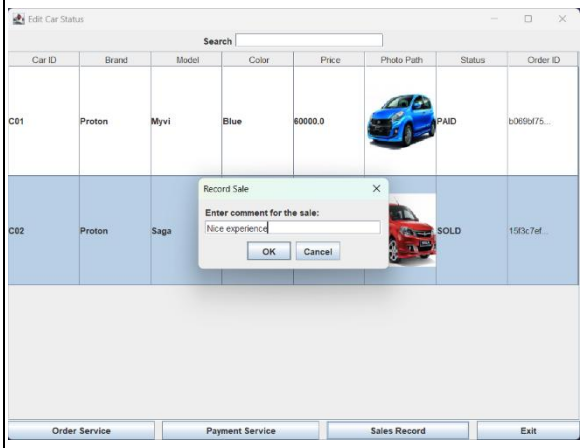


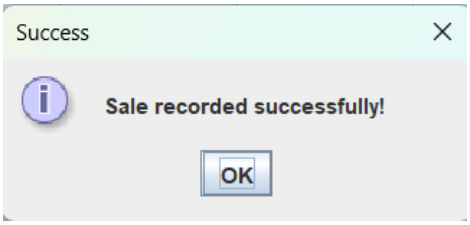












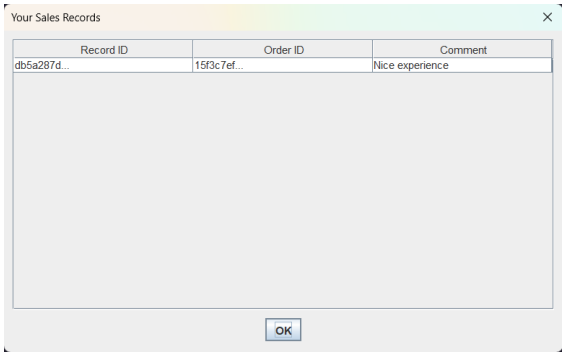
Upon accessing the **View and Update Status of Cars** page, users can oversee car details in a structured table format. The interface includes a **search bar** at the top for easy lookup. Below, the table displays essential information with columns such as **Car ID, Brand, Model, Colour, Price, Photo Path, and Status.** At the bottom, four buttons which are **Order Service, Payment Service, Sales Record, and Exit**, allow users to modify, remove, or add new car records efficiently.

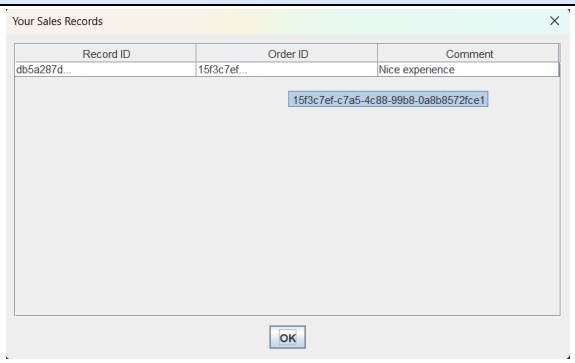
Upon selecting one of the car entries and the **“Order Service”** button, a dropdown menu labelled **“Select new status:”** will prompt one of the following options: **AVAILABLE, BOOKED, PAID and CANCELLED.** Below, two buttons labelled **"OK"** and **"Cancel"** provide users with the option to confirm or exit the update process. As seen above, if we proceed with the choice **“BOOKED”** and choose **“OK”**, we can see that the relevant information of the car has been updated along with a pop-up message **"Car status updated successfully! Please remind customer for paying! BOOKED"**

Upon selecting one of the car entries and **“Payment Service”** button**,** a pop-up window appears allowing users to select which vehicle order is to be recorded for payment. The drop-down menu prompts users to choose a **OrderID to pay**. Below, two buttons labelled **"OK"** and **"Cancel"** provide users with the option to confirm or exit the update process. As seen above, if we proceed and choose **“OK”**, we can see that the relevant information of the car has been updated along with a pop-up message **" Sale finalized. Order marked COMPLETED and car marked Sold."**

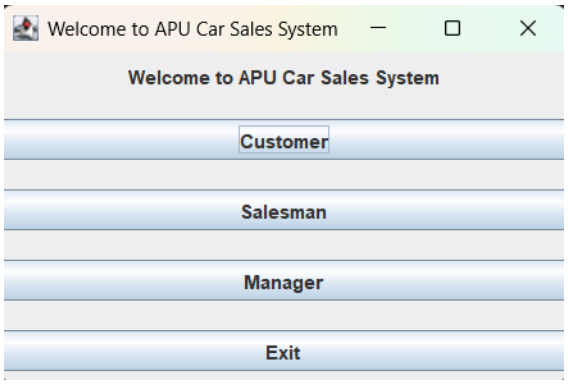
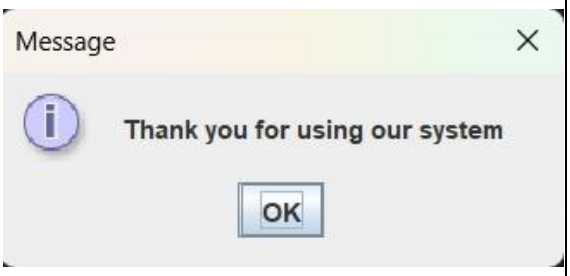
Upon selecting one of the car entries, and then selecting the **“Sales Record”** button, a pop-up window titled **"Record Sale”** appears, prompting users to enter a comment for the sale. Here we have already filled the text field with "Nice experience". Below, two buttons labelled **"OK"** and **"Cancel"** provide users with the option to confirm or exit the update process. As seen above, if we proceed with our comment and choose **“OK”**, we can see that the relevant information of the car has been updated along with a pop-up message "**Sale recorded successfully" and an “OK” button** that allows users to acknowledge the sales record.

View Sales Records





Upon accessing the **View Sales Record** page, users can oversee sales record in a structured table format. The table displays essential information with columns such as **RecordId, OrderID and Comment.** At the bottom, there is a **OK button** that allows users to view and acknowledge the sales record table.

System Exit Page	
	

Last but not least, upon selecting the **'Exit" button** from the system main page, a pop-up message will appear and displaying **a confirmation notification**. The text reads **"Thank you for using our system"**. Below, an **"OK" button** allows users to acknowledge and close the message box. This simple confirmation ensures users aware that their interaction with the system has been completed successfully.

4.0 Sample Code (OO concepts and Java features)

```java
package UserPackage;

import ...

Edit | Explain | Test | Document | Fix
public abstract class User {  TeoJunJie *
    protected String id;
    protected String username;
    protected String password;
    protected String email;
    protected String phoneNumber;  4 usages
    protected UserStatus status;
    protected UserRole role;  7 usages
    protected LocalDateTime createdAt;  1 usage

    Edit | Explain | Test | Document | Fix
    public User(String username, String password, String email, String phoneNumber, UserRole role) {  TeoJunJie
        this.username = username;
        this.password = password;
        this.email = email;
        this.phoneNumber = phoneNumber;
        this.role = role;
        this.status = UserStatus.PENDING;
        this.createdAt = LocalDateTime.now();
    }
```

```java
public String getId() { return id; }  TeoJunJie
public void setId(String id) { this.id = id;}  3 overrides  TeoJunJie
public String getPhoneNumber() { return phoneNumber; }  1 override  TeoJunJie
public void setPhoneNumber(String phoneNumber) { this.phoneNumber = phoneNumber; }
public String getUsername() { return username; }  TeoJunJie
public String getPassword() { return password; }  TeoJunJie
public String getEmail() { return email; }  TeoJunJie
public void setEmail(String email) { this.email = email != null ? email : ""; }
public UserStatus getStatus() { return status; }  TeoJunJie
public void setUsername(String username) { this.username = username; }  TeoJunJie
public void setPassword(String password) { this.password = password; }  TeoJunJie
public void setStatus(UserStatus status) { this.status = status; }  TeoJunJie

public abstract String toCSV();  3 implementations  TeoJunJie
Edit | Explain | Test | Document | Fix
public static User fromCSV(String csv) {  TeoJunJie *
    String[] parts = csv.split( regex: ",",  limit: -1);
    if (parts.length < 7) {
        throw new IllegalArgumentException("Insufficient CSV fields");
    }
    UserRole role = UserRole.valueOf(parts[6]);
    return switch (role) {
        case CUSTOMER -> Customer.fromCSV(csv);
        case MANAGER -> Manager.fromCSV(csv);
        case SALESMAN -> Salesman.fromCSV(csv);
    };
}
```

```java
package CustomerPackage;

import ...

Edit | Explain | Test | Document | Fix
public class Customer extends User {  TeoJunJie *
    private String customerId;  4 usages
    private final List<String> purchaseHistory = new ArrayList<>();  no usages
    Edit | Explain | Test | Document | Fix
    public Customer(String username, String password, String email, String phoneNumber) {  TeoJunJie
        super(username, password, email, phoneNumber, UserRole.CUSTOMER);
        this.customerId = null;
    }
    Edit | Explain | Test | Document | Fix
    public Customer(String id, String username, String password, String email, String phoneNumber) {  TeoJunJie
        this(username, password, email, phoneNumber);
        this.id = id;
        this.customerId = "C-" + id;
    }

    Edit | Explain | Test | Document | Fix
    // Getters and setters
    public void setId(String id) {  TeoJunJie
        this.id = id;
        this.customerId = "C-" + id;
    }
    public String getCustomerId() { return customerId; }
    public void setPhoneNumber(String phoneNumber) { this.phoneNumber = phoneNumber != null ? phoneNumber : ""; }
    public void setRole(UserRole role) {  1 usage  TeoJunJie
        this.role = role;
    }
```

```java
Edit | Explain | Test | Document | Fix
@Override  TeoJunJie
public String toCSV() {
    return String.join( delimiter: ",",
            getId(),
            getUsername(),
            getPassword(),
            getEmail(),
            getPhoneNumber(),
            getStatus().name()
    );
}

Edit | Explain | Test | Document | Fix
public static Customer fromCSV(String csv) {  TeoJunJie *
    String[] parts = csv.split( regex: ",",  limit: -1);
    if (parts.length < 6) {
        throw new IllegalArgumentException("Invalid customer CSV: " + csv);
    }
    String id        = parts[0].trim();
    String username  = parts[1].trim();
    String password  = parts[2].trim();
    String email     = parts[3].trim();
    String phone     = parts[4].trim();
    String statusStr = parts[5].trim();
    Customer c = new Customer(id, username, password, email, phone);
    if (!statusStr.isEmpty()) {
        c.setStatus(UserStatus.valueOf(statusStr));
    }
    c.setRole(UserRole.CUSTOMER);
    return c;
}
```

Object Oriented Concepts	Embodiment in code
Abstraction	User is declared as an abstract class and defines the public abstract String toCSV(). All concrete users are 'what' - but do not expose their respective implementation details, which must be completed by subclasses. The static factory User.fromCSV(...) is also an abstraction of 'how to construct objects based on roles'.
Inheritance	Public class Customer extends User: Customer automatically inherits fields (such as username, status) and methods (such as setStatus()) from User, and expands with customerId. At the same time, it overrides toCSV() to output itself in a subclass-specific format.
Encapsulation	Declare all key attributes (id, username, password, email, phoneNumber, status, etc.) as private or protected, and use getXxx()/setXxx() methods to read and modify them, avoiding direct manipulation of fields from the outside, which ensures the integrity and security of the object's state.
Modularity	Package Structure Division: Place User, Customer, Manager, and Salesman in different packages such as user, customer, manager, and salesman respectively, and clarify the dependency relationships through imports to form clear functional module boundaries. Single Responsibility: User is responsible for general user attributes and behaviors,

	Customer handles client-specific logic, and CSV serialization is placed in their respective classes. Each module has low coupling and high cohesion, making it easy to maintain and expand.

```java
// Make a payment
public Payment makePayment(String orderId, double amount) {  1 usage  TeoJunJie *
    Order order = findOrder(orderId);
    if (order == null || !"CONFIRMED".equalsIgnoreCase(order.getStatus())) {
        throw new IllegalStateException("Order not ready for payment");
    }

    String paymentId = UUID.randomUUID().toString();
    Payment payment = new Payment(paymentId, orderId, amount, new Date());
    appendToFile(PAYMENTS_FILE, payment.toCSV());
    order.setStatus("PAID");
    saveOrders();
    order.setStatus("COMPLETED");
    saveOrders();
    Car car = carManagement.getCar(order.getCarId());
    if (car != null) {
        car.setStatus("SOLD");
        carManagement.updateCar(car.getCarId(), car);
    }
    log( action: "PAYMENT_AND_COMPLETE",  detail: orderId + ", payment=" + paymentId);
    return payment;
}
```

- Key Object-Oriented concepts: Encapsulation

The entire payment and completion process (validation, file writing, status update, logging) is converged in a single method, and the caller simply makesPayment(...)without having to know the internal details.

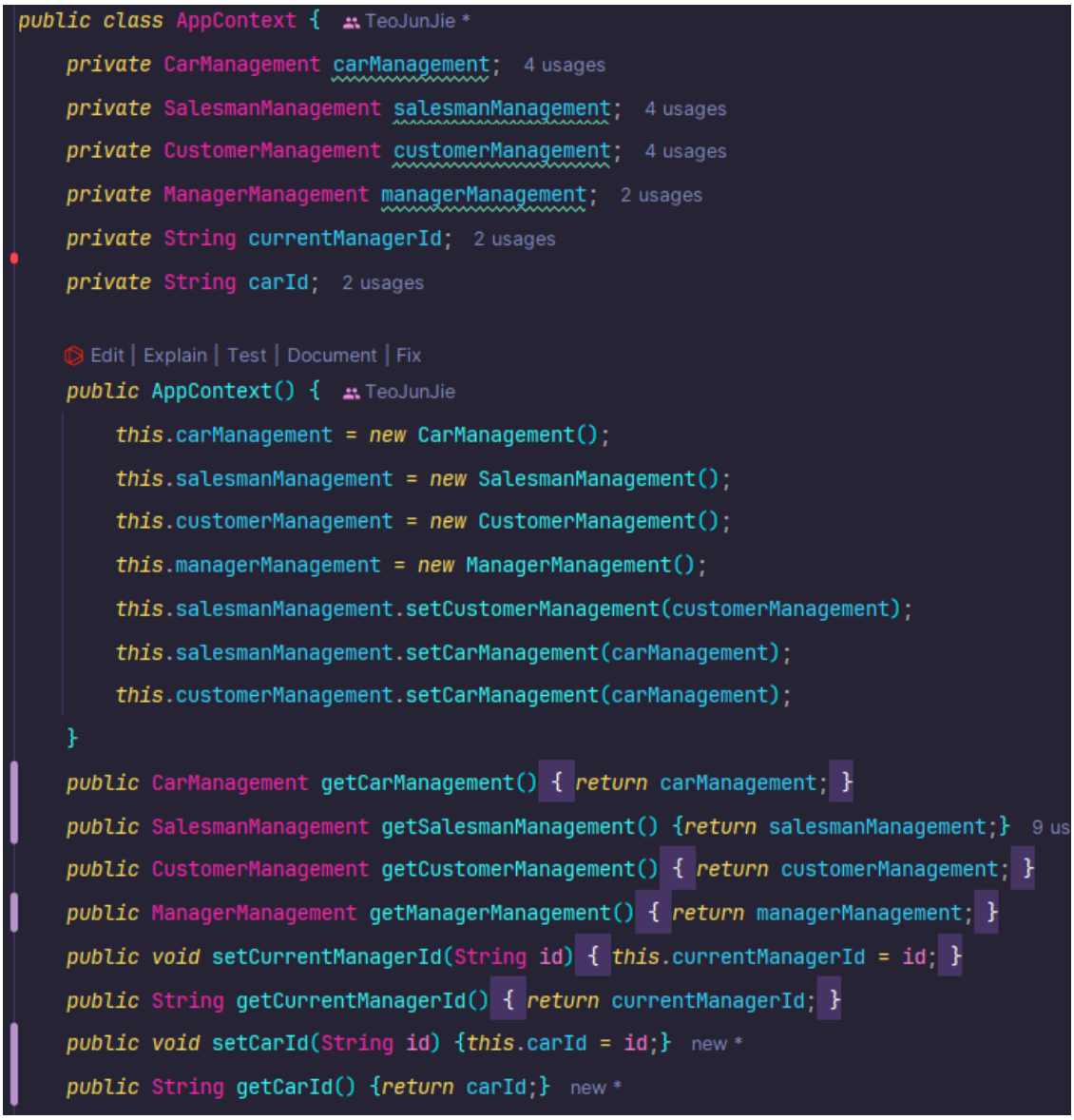
- Run Process Overview

1) Find and verify order status.
2) Generate Payment object and write it to the persistence file.
3) Mark the order as "PAID" and "COMPLETED" successively and save it back to orders.txt.
4) Update the corresponding vehicle status to "SOLD".
5) Logging and finally returning to the Payment instance.

The whole process is invoked once, and all persistence and object state changes are done automatically.

5.0 Additional Features

5.1 Centralized Dependency & Session Context



```
public class AppContext {  TeoJunJie *
    private CarManagement carManagement;  4 usages
    private SalesmanManagement salesmanManagement;  4 usages
    private CustomerManagement customerManagement;  4 usages
    private ManagerManagement managerManagement;  2 usages
    private String currentManagerId;  2 usages
    private String carId;  2 usages

    Edit | Explain | Test | Document | Fix
    public AppContext() {  TeoJunJie
        this.carManagement = new CarManagement();
        this.salesmanManagement = new SalesmanManagement();
        this.customerManagement = new CustomerManagement();
        this.managerManagement = new ManagerManagement();
        this.salesmanManagement.setCustomerManagement(customerManagement);
        this.salesmanManagement.setCarManagement(carManagement);
        this.customerManagement.setCarManagement(carManagement);
    }
    public CarManagement getCarManagement() { return carManagement; }
    public SalesmanManagement getSalesmanManagement() {return salesmanManagement;}  9 us
    public CustomerManagement getCustomerManagement() { return customerManagement; }
    public ManagerManagement getManagerManagement() { return managerManagement; }
    public void setCurrentManagerId(String id) { this.currentManagerId = id; }
    public String getCurrentManagerId() { return currentManagerId; }
    public void setCarId(String id) {this.carId = id;}  new *
    public String getCarId() {return carId;}  new *
```

Key Benefits

1. Modularity

- All service objects are created and wired in one place.
- Individual screens simply call context.getXxxManagement() rather than new call each service themselves.

2. Encapsulation

- AppContext hides the details of how services are instantiated and interlinked.
- Future changes only require updates here.

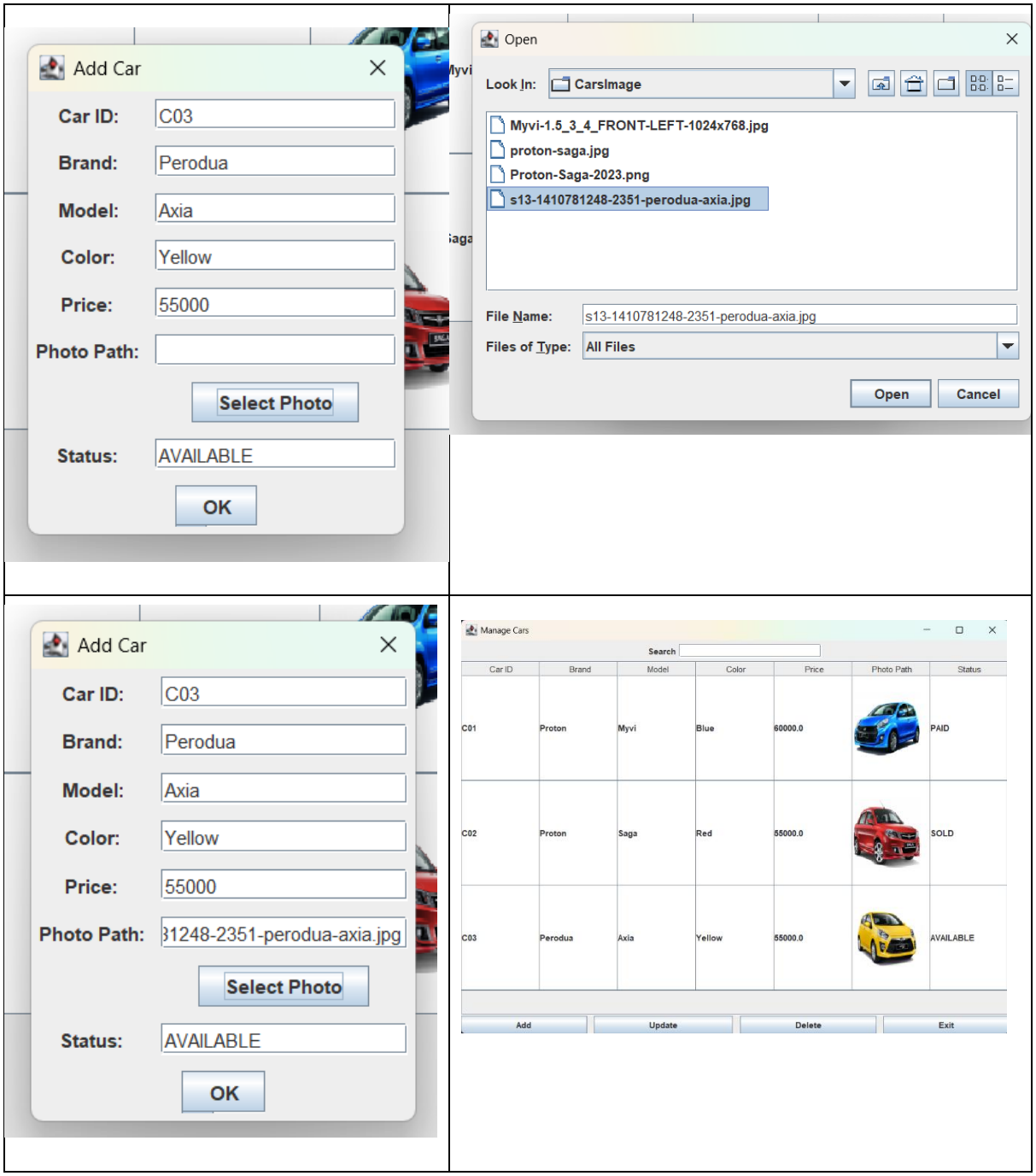
3. Session Management

- The field currentManagerId carries the logged-in-manager's identity across screens.
- Additional state like carId for editing workflows can be stored and retrieved consistently.
- No need to modify every UI class, just expose a new getter on AppContext.

5.2 Thumbnail Renderer

```
public class CarTableCellRenderer extends JLabel implements TableCellRenderer {  3 usages  TeoJunJie *
    Edit | Explain | Test | Document | Fix
    @Override  TeoJunJie *
    public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {
        setText("");
        setIcon(null);
        if (column == 5) {
            String photoPath = value.toString();
            URL imgUrl = getClass().getClassLoader().getResource(photoPath);
            if (imgUrl != null) {
                ImageIcon icon = new ImageIcon(imgUrl);
                Image scaled = icon.getImage()
                        .getScaledInstance( width: 100,  height: 100, Image.SCALE_SMOOTH);
                setIcon(new ImageIcon(scaled));
            } else {
                ImageIcon icon = new ImageIcon(photoPath);
                if (icon.getIconWidth() == -1) {
                    System.out.println("Image not found: " + photoPath);
                    setText("Image not found");
                } else {
                    Image scaled = icon.getImage()
                            .getScaledInstance( width: 100,  height: 100, Image.SCALE_SMOOTH);
                    setIcon(new ImageIcon(scaled));
                }
            }
        } else {setText(value.toString());}
        if (isSelected) {
            setBackground(table.getSelectionBackground());
            setForeground(table.getSelectionForeground());
        } else {
            setBackground(table.getBackground());
            setForeground(table.getForeground());}
        setOpaque(true);
        return this;
    }
```

```
    Edit | Explain | Test | Document | Fix
    private void selectPhoto() {  1 usage  TeoJunJie
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setCurrentDirectory(new File(System.getProperty("user.dir")));
        int result = fileChooser.showOpenDialog( parent: this);
        if (result == JFileChooser.APPROVE_OPTION) {
            File f = fileChooser.getSelectedFile();
            photoField.setText(f.getAbsolutePath());
        }
    }
```



1. Photo-Picker Dialog (selectPhoto())

Function

- Presents a standard file-chooser dialog rooted at the user's working directory
- Allows users to visually navigate folders and select an image file.
- Automatically fills the associated text field (photofield) with the chosen file's absolute path.

Benefit

- User-friendly input
  i) No more manual typing of long file paths
  ii) Reduces input errors like typos, wrong extensions and so on.
- Immediate validation
  i) If the chosen file later fails to render, the system can catch that and alert the user.
- Seamless integration
  i) Works hand-in-hand with our table renderer to show real thumbnails.

2. Thumbnail Renderer (CarTableCellRenderer)

Function

- Implements TableCellRenderer to display a 100x100 thumbnail in the table's "Photo" column.
- First tries to load the image as a classpath resource; if not found, falls back to the filesystem path.
- Gracefully shows a "Image not found" label if both loading attempts fail.
- Applies standard selection highlighting so thumbnails feel native inside a JTable.

Benefit

1. Visual clarity

- Users see actual car images rather cryptic file paths

2. Robust error handling

- Missing or mis-typed paths don't crash the UI; instead, the cell displays an informative message.

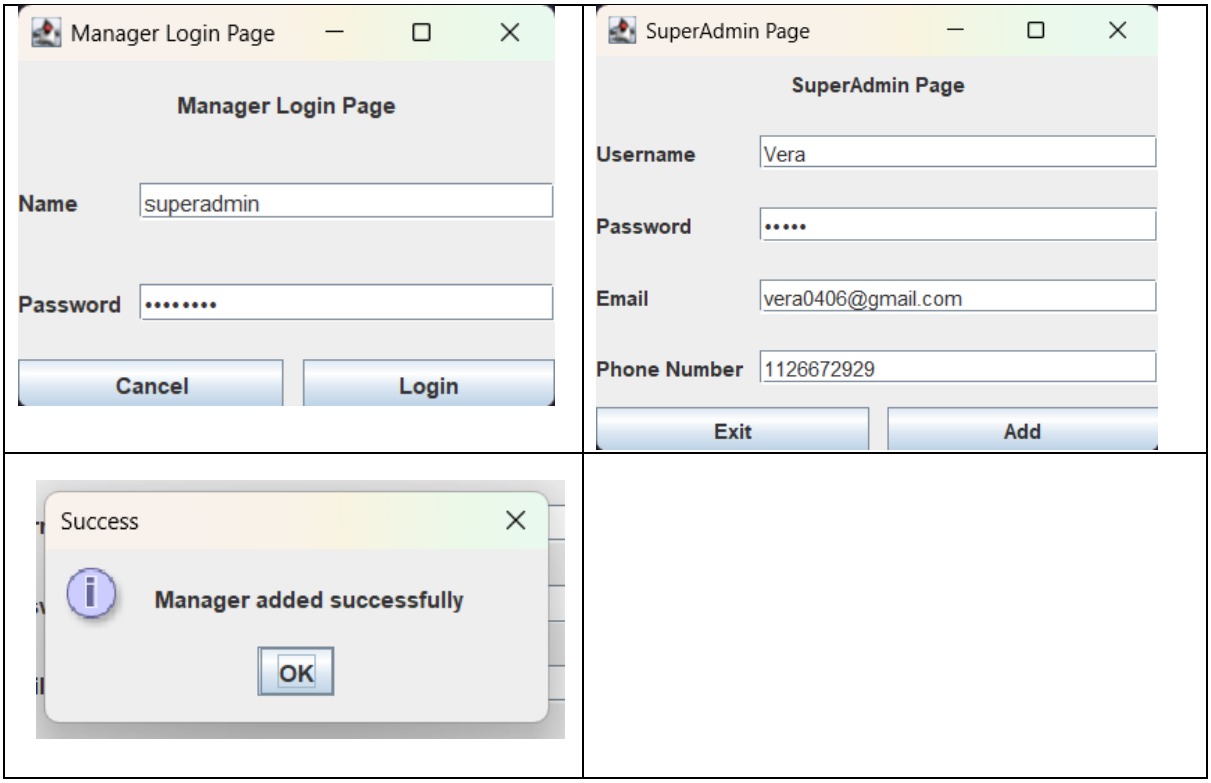
3. Reusability

- Any table with a photo-column can reuse this renderer, promoting modularity

5.3 Super Admin “Backdoor” Feature

```
loginButton.addActionListener( ActionEvent e -> {
    String username = NameTextField.getText().trim();
    String password = new String(PasswordField.getPassword()).trim();
    if (username.isEmpty() || password.isEmpty()) {
        JOptionPane.showMessageDialog( parentComponent: this,
                message: "Username and password cannot be empty",  title: "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    String id = authenticateAndGetId(username, password);
    if (id != null) {
        context.setCurrentManagerId(id);
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Login successful!",  title: "Success", JOptionPane.INFORMATION_MESSAGE);
        new ManagersMain(context);
        this.dispose();
    } else if ("superadmin".equals(username) && "admin123".equals(password)) {
        context.setCurrentManagerId(username);
        new SuperAdminPage(context);
        this.dispose();
    } else {
        JOptionPane.showMessageDialog( parentComponent: this,
                message: "Invalid username or password",  title: "Login Failed", JOptionPane.ERROR_MESSAGE);
    }
});
```

```
addButton.addActionListener( ActionEvent e -> {
    String username = UsernameTextField.getText().trim();
    String password = new String(PasswordField.getPassword()).trim();
    String email = EmailTextField.getText().trim();
    String phoneNumber = PhoneNumberTextField.getText().trim();
    if (username.isEmpty() || password.isEmpty() || email.isEmpty() || phoneNumber.isEmpty()) {
        JOptionPane.showMessageDialog( parentComponent: this,  message: "All fields are required",  title: "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    Manager newManager = new Manager(username, password, email, phoneNumber);
    ManagerManagement mm = context.getManagerManagement();
    boolean added = mm.addManager(newManager);
    if (!added) {
        JOptionPane.showMessageDialog( parentComponent: this,
                message: "Username already exists",
                title: "Error", JOptionPane.ERROR_MESSAGE);
        return;}
    String newId = newManager.getId();
    context.setCurrentManagerId(newId);
    JOptionPane.showMessageDialog( parentComponent: this,
            message: "Manager added successfully! " + username,
            title: "Success", JOptionPane.INFORMATION_MESSAGE);
    UsernameTextField.setText("");
    PasswordField.setText("");
    EmailTextField.setText("");
    PhoneNumberTextField.setText("");
    new ManagerLogin(context);
    dispose();
});
}
```



Upon recognizing "superadmin"/ "admin123", we store managerId = "superadmin" in our AppContext. This allows downstream screens to know the session is elevated. Instead of opening ManagersMain, we open SuperAdminPage, which provides an "Add Manager" form:

- Fields: Username, Password, Email, Phone Number
- On submit: Invokes ManagerManagement.addManager(…) to append a new line in managers.txt, then redirects back to the login screen.

Benefits

- Emergency Access: In case all other manager accounts are misconfigured or lost, the super-admin credentials remain a fail-safe to regain control.
- Controlled Privilege: Only this single, well-documented credential pair can reach the high-privilege "Add Manager" UI-regular users cannot.
- Minimal Overhead: No separate user-type table or Database schema is required; it lives simply in the login listener's fallback branch.

5.4 ID Shortener Renderer

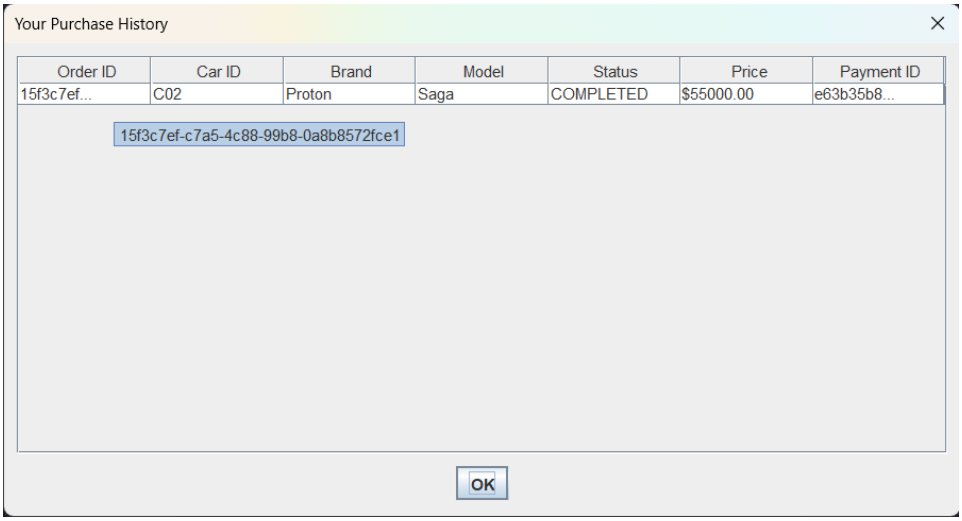
```java
public class IdShortenerRenderer extends DefaultTableCellRenderer {  13 usages  TeoJunJie

    private static final int PREFIX_LEN = 8;  2 usages

    Edit | Explain | Test | Document | Fix
    @Override  TeoJunJie
    public Component getTableCellRendererComponent(JTable table,
                                                   Object value,
                                                   boolean isSelected,
                                                   boolean hasFocus,
                                                   int row,
                                                   int column) {
        super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);

        String fullId = value == null ? "" : value.toString();
        String text;
        if (fullId.length() > PREFIX_LEN) {
            text = fullId.substring(0, PREFIX_LEN) + "...";
        } else {
            text = fullId;
        }
        setText(text);
        setToolTipText(fullId);
        return this;
    }
}
```

```java
ViewPurchaseHistories.addActionListener( ActionEvent e -> {
            String[] cols = {
                    "Order ID", "Car ID", "Brand", "Model",
                    "Status", "Price", "Payment ID"
            };
            DefaultTableModel model = new DefaultTableModel(cols,  rowCount: 0) {  TeoJunJie
                @Override public boolean isCellEditable(int r, int c) { return false; }  TeoJunJie
            };
            for (Payment p : userPayments) {
                Order o = customerManagement.findOrder(p.getOrderId());
                String carId = o.getCarId();
                Car car = carManagement.getCar(carId);
                String brand     = car  != null ? car.getBrand()     : "N/A";
                String modelName = car  != null ? car.getModel()     : "N/A";
                double price     = car  != null ? car.getPrice()     : 0.0;
                model.addRow(new Object[]{
                        o.getOrderId(),
                        carId,
                        brand,
                        modelName,
                        o.getStatus(),
                        String.format("$%.2f", price),
                        p.getPaymentId()
                });
            }
            JTable table = new JTable(model);
            table.setAutoCreateRowSorter(true);
            table.getColumnModel().getColumn( columnIndex: 0).setCellRenderer(new IdShortenerRenderer());
            table.getColumnModel().getColumn( columnIndex: 6).setCellRenderer(new IdShortenerRenderer());
            JScrollPane scroll = new JScrollPane(table);
            scroll.setPreferredSize(new Dimension( width: 700,  height: 300));
```



Benefits

- Readability: Users see a concise 8-character prefix instead of a 36-character UUID, preventing columns from becoming excessively wide.
- Discoverability: Hovering over any truncated ID reveals the full identifier in a tooltip, preserving traceability.
- Modularity: Encapsulated in its own class, this renderer can be reused anywhere long IDs need display without altering the table's data model.

5.5 TableExporter

```java
public class TableExporter {  3 usages  TeoJunJie
    Edit | Explain | Test | Document | Fix
    public static void exportToCSV(DefaultTableModel model, File outFile) throws IOException {
        try (PrintWriter pw = new PrintWriter(outFile)) {
            for (int c = 0; c < model.getColumnCount(); c++) {
                pw.print(model.getColumnName(c));
                if (c < model.getColumnCount() - 1) pw.print(",");
            }
            pw.println();
            for (int r = 0; r < model.getRowCount(); r++) {
                for (int c = 0; c < model.getColumnCount(); c++) {
                    String cell = String.valueOf(model.getValueAt(r, c))
                            .replace( target: "\"",  replacement: "\"\"");
                    pw.print("\"" + cell + "\"");
                    if (c < model.getColumnCount() - 1) pw.print(",");
                }
                pw.println();
            }
        }
    }
}
```

TableExporter Utility

Function

Converts any DefaultTableModel into a properly quoted CSV file.

Key Points

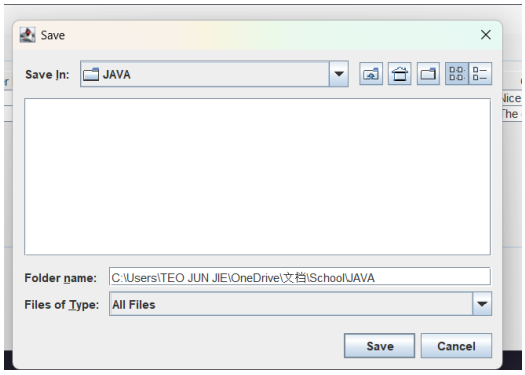
- Outputs coloumn headers, separated by commas.
- Wraps each cell value in double-quotes and escapes internal quotes by doubling them.
- Writes one row per table row, ensuring compatibility with Excel and other spreadsheet tools.

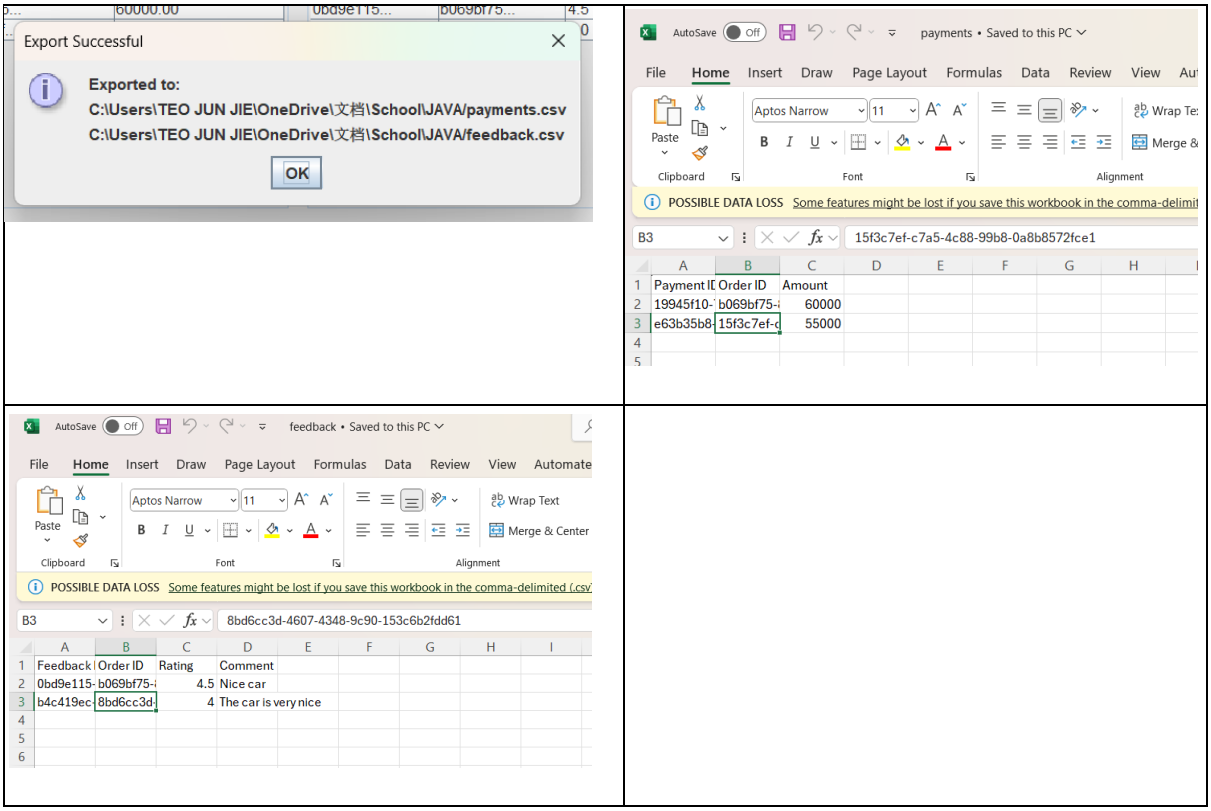
```java
viewPurchaseHistories.addActionListener( ActionEvent e -> {
        String[] cols = {
                "Order ID", "Car ID", "Brand", "Model",
                "Status", "Price", "Payment ID"
        };
        DefaultTableModel model = new DefaultTableModel(cols,  rowCount: 0) {  TeoJunJie
            @Override public boolean isCellEditable(int r, int c) { return false; }  TeoJunJie
        };
        for (Payment p : userPayments) {
            Order o = customerManagement.findOrder(p.getOrderId());
            String carId = o.getCarId();
            Car car = carManagement.getCar(carId);
            String brand     = car  != null ? car.getBrand()     : "N/A";
            String modelName = car  != null ? car.getModel()     : "N/A";
            double price     = car  != null ? car.getPrice()     : 0.0;
            model.addRow(new Object[]{
                    o.getOrderId(),
                    carId,
                    brand,
                    modelName,
                    o.getStatus(),
                    String.format("$%.2f", price),
                    p.getPaymentId()
            });
        }
        JTable table = new JTable(model);
        table.setAutoCreateRowSorter(true);
        table.getColumnModel().getColumn( columnIndex: 0).setCellRenderer(new IdShortenerRenderer());
        table.getColumnModel().getColumn( columnIndex: 6).setCellRenderer(new IdShortenerRenderer());
        JScrollPane scroll = new JScrollPane(table);
        scroll.setPreferredSize(new Dimension( width: 700,  height: 300));
```

“Export to CSV” Button Integration

Function

Opens a JFileChooser restricted to directories and calls TableExporter.exportToCSV(…) twice, once for the payments model and once for the feedback model, producing payments.csv and feedback.csv in the chosen folder. After that displays a success or error message.





Benefits

- Data Portability: Managers can open the CSV files in Excel or other tools for further analysis or record-keeping.
- Reusability: TableExporter is generic and can be applied to any table model across the application.
- User Experience: Clear prompts and success/failure feedback guide users through the export process without manual file manipulation.

6.0 References

Lavand, M., & Pawar, R. A Study On Abstraction in Object Oriented Programming with Java and Its Implementation in Developing Programs. *IJSAT-International Journal on Science and Technology*, *16*(2). https://doi.org/10.71097/IJSAT.v16.i2.3438

Nagineni, R. B. (2021). A Research on Object Oriented Programming and Its Concepts. *Andhra Pradesh, India: International Journal of Advanced Trends in Computer Science and Engineering*. https://doi.org/10.30534/ijatcse/2021/401022021

Végh, L., & Czakóová, K. (2023). Possibilities of using games in teaching and learning the basic concepts of object-oriented programming. In *INTED2023 Proceedings* (pp. 5329-5334). IATED. https://doi.org/10.21125/inted.2023.1383