Cardiff Metropolitan University | Prifysgol Metropolitan Caerdydd

Cardiff School of Technologies

Name: Ahsanul Haque Mamun

Student Id: 20317001

Course: BSc (Hons)Computer Science

Module Name: Principles of programming

Module Code: SEN4000

Module Leader: Dr Amrita Prasad

Tittle: Principles of Programming in Action:- A Car Warehouse System

TABLE OF CONTENTS

1. Project Overview and Development Objectives

As a junior software engineer at a reputable software development business, I was given a fun project to work on for a local car dealership. The dealership had a hard time keeping track of their cars, keeping their inventory up to date, and keeping good records of their sales operations. They didn't have a centralized digital tool to make running their business easier, which caused delays, mistakes, and the loss of important customer sales data.
The project's main goal was to create a strong and easy-to-use Car Inventory Management System that would bring the dealership's processes up to date. The project had to be carefully planned with both expert and non-technical users in mind. This meant making a Graphical User Interface (GUI) that was easy to use and making sure the data handling backend was strong and stable. (Python Software Foundation, 2023)
Python and the Tkinter tool for making graphical user interfaces were used to build the system. Tkinter makes it easy to make displays that change and can be interacted with. To keep track of car supplies and sales, simple CSV files were used to store the data. This made it portable and easy to make changes in the future without having to set up a complex database.

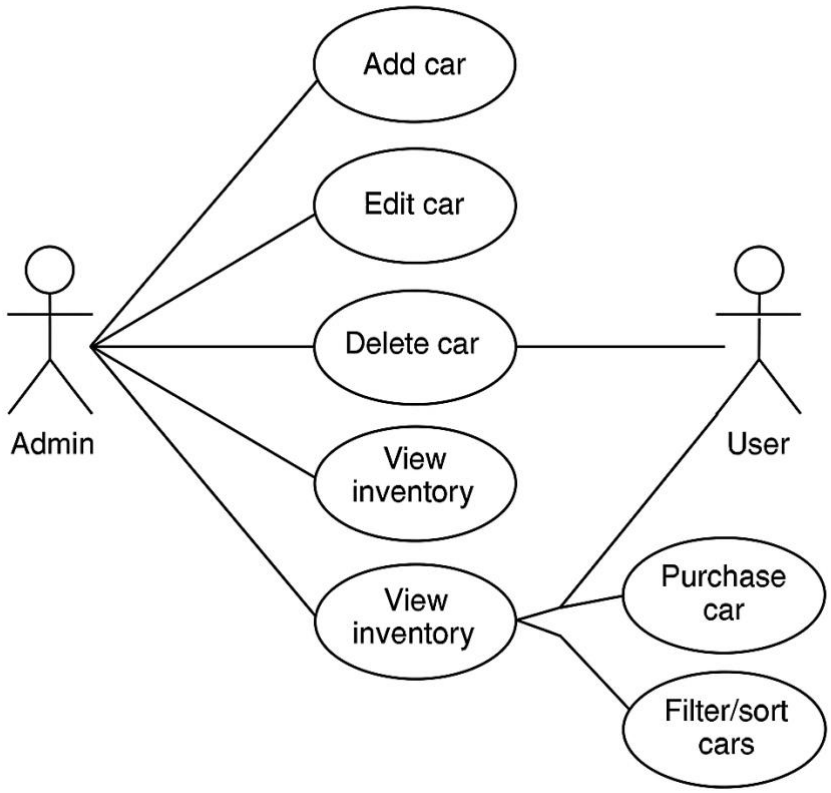


Figure 1: UML Use Case Diagram of System Actors and Action

The key development objectives were as follows:

- To implement a two-tab GUI system with distinct interfaces for admin and user operations.
- To allow full CRUD (Create, Read, Update, Delete) functionality for managing car listings efficiently.
- To enable users to search through the inventory based on key fields like brand, year, or cost, and to sort listings accordingly.
- To allow simulation of car purchases by users, updating the car status to "Sold" automatically.
- To log complete sales information, including the date of purchase, sales ID, brand, customer name, price sold, and salesperson, into a separate sales record file.

Overall, this project highlights the importance of modular programming, GUI responsiveness, proper data handling, and error management — all essential elements in modern software engineering.

2. User Interface Design and Architecture

The design of the Car Inventory Management System focused heavily on creating an intuitive, efficient, and visually appealing user interface (UI).

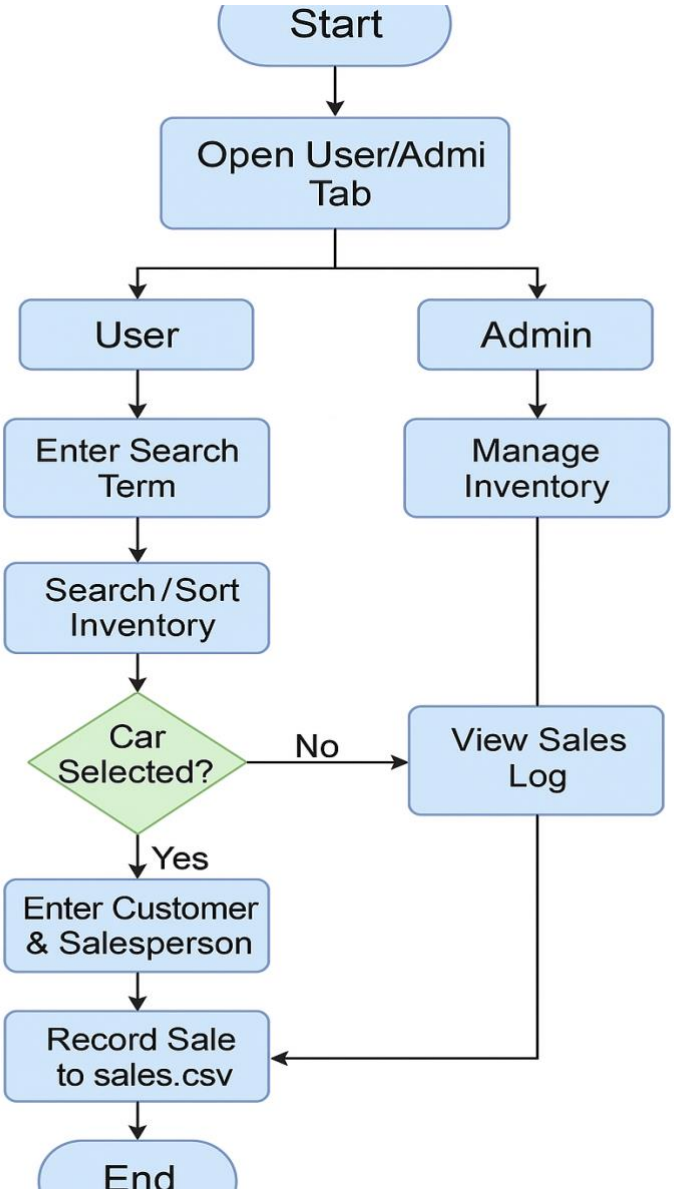


Figure 2: Flowchart Representing Main System Logic

2.1 Design Principles and Planning

The initial planning stage involved researching best practices in user interface design for small business management systems. Key design principles that guided the development included:

Simplicity and Clarity: The design keeps things simple by focusing on the most important parts. Labels, buttons, and text areas have clear names that make it easy for users to do what they need to do.

Consistency: Consistent fonts, button sizes, and colours across the User and Admin tabs make the system predictable, reducing learning time.

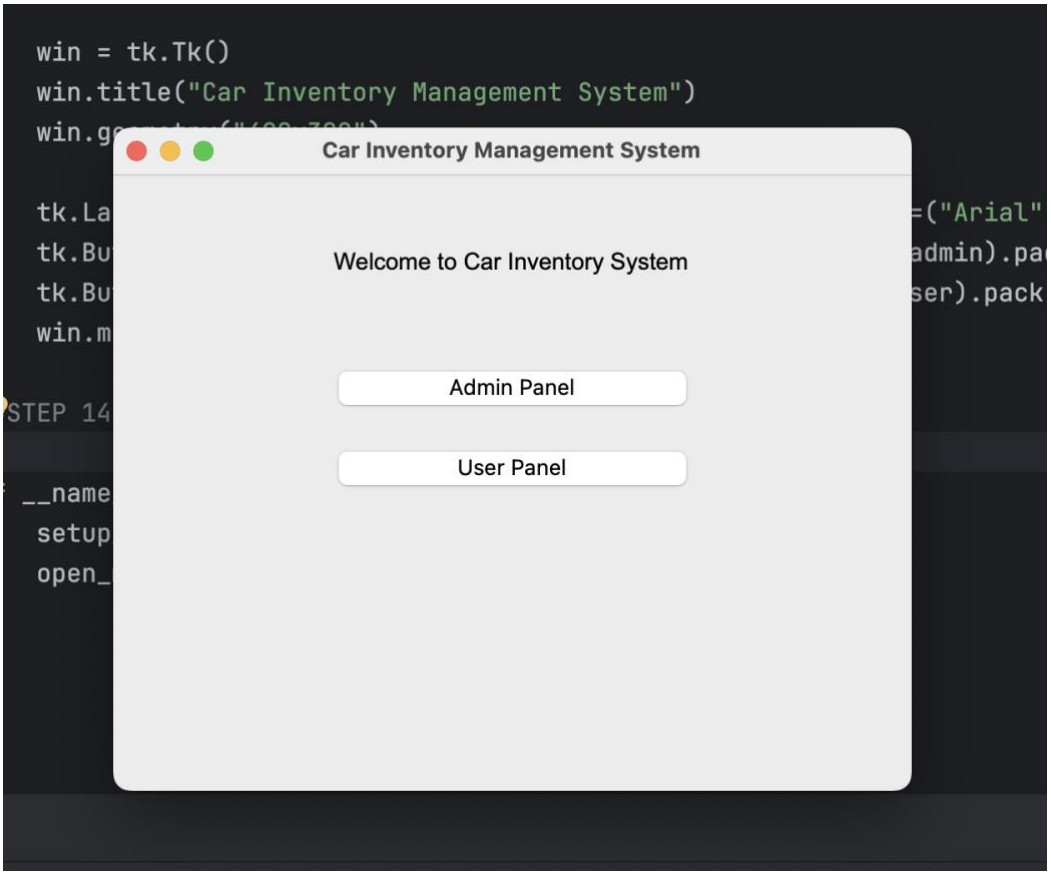
Logical Grouping: Related functions are grouped together. For example, all the car management actions (Add, Update, Delete) are placed close together in the admin panel.

Minimal User Input: Wherever possible, combo boxes were used instead of text inputs (for example, when selecting search fields) to reduce typing errors and speed up the workflow.

Error Prevention and Feedback: All user input fields are validated, and the system uses pop-up message boxes (Tkinter's messagebox) to alert users to success, errors, or missing information. This prevents system crashes and improves reliability.

Tab-Based Navigation: Tkinter's Notebook widget allowed clear separation between the User and Admin interfaces, providing a clean and organized structure.

This planning ensured that even users without deep technical knowledge could navigate and operate the system easily. (Tkinter Docs, 2023)



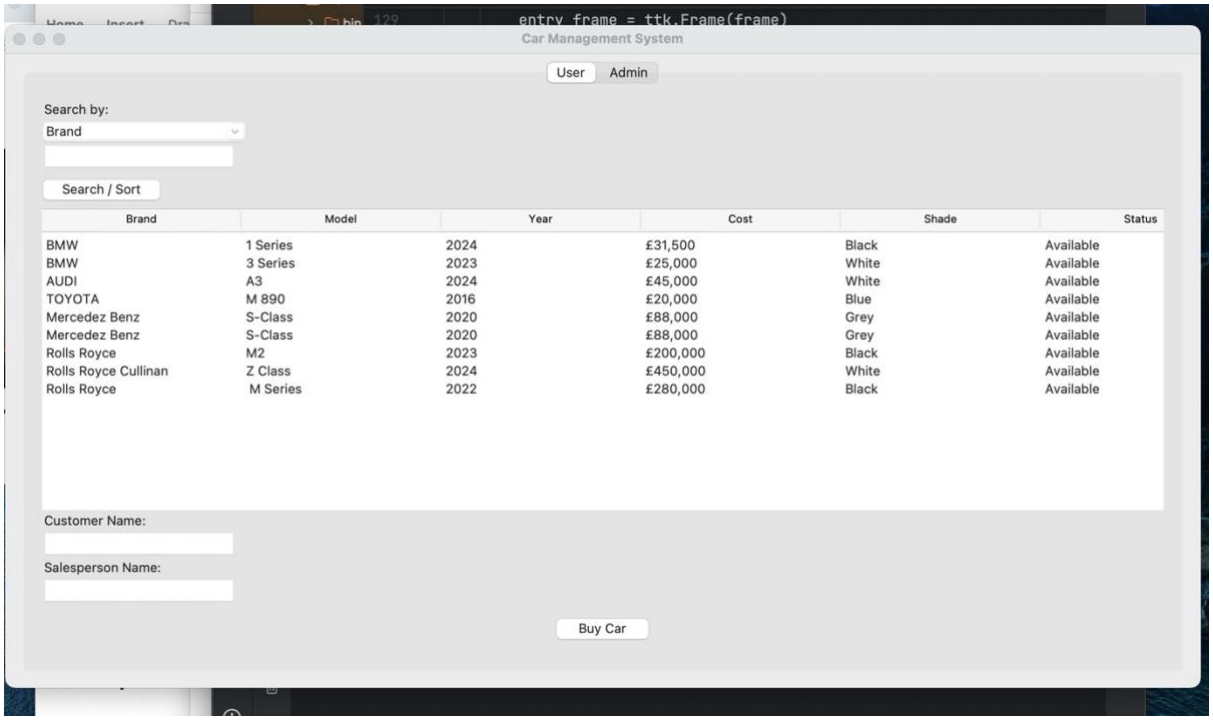
2.2 Functional UI Elements and Layout

The application interface is split into two major sections:

User Panel

The User tab is designed for customers or front-desk employees. The User Panel provides the following functionality:

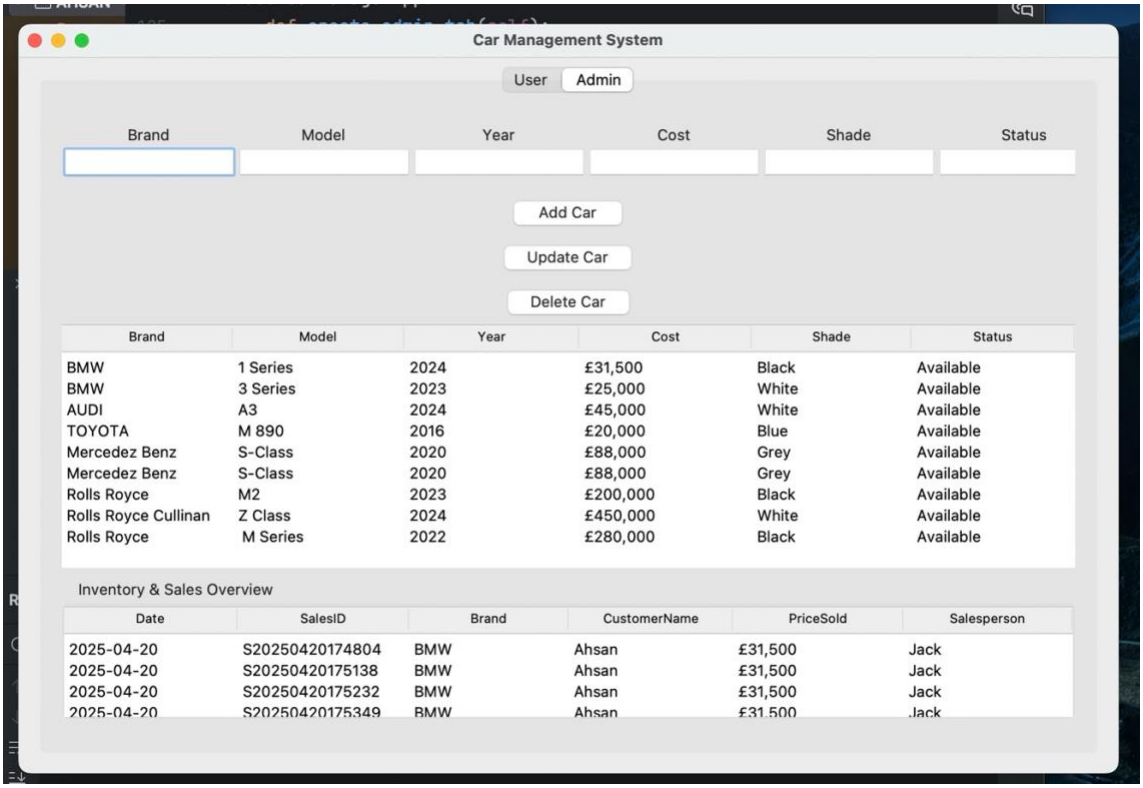
- **Search and Sort Area**: At the top of the User tab, a combo box allows users to select a search criterion (e.g., Brand, Year, Cost). Below that, a text entry box allows them to input the search term. A Search/Sort button triggers the search or sort function.
- **Inventory Table (Treeview)**: The middle section displays the car listings in a Tkinter Treeview widget. Each car's Brand, Model, Year, Cost, Shade, and Status are displayed in a structured table format with scrollable support. (Tkinter Docs, 2023)
- **Purchase Section**: Below the table, users can enter the customer's name and salesperson name into two separate entry fields. A "Buy Car" button is provided to complete the purchase, automatically updating the car status and recording the sale.



Admin Panel

The admin tab provides all the management tools needed by the dealership staff:

- **Car Details Entry Form**: Across the top of the Admin tab, entry fields are available for Brand, Model, Year, Cost, Shade, and Status. Admins use these fields to add new cars or edit existing entries.
- **Admin Action Buttons**: Below the form, three primary buttons (Add Car, Update Car, Delete Car) allow the administrator to perform CRUD operations on the inventory.
- **Admin Inventory Table**: A Treeview table like the User Panel displays the full car inventory, but here admins can select a car to edit or delete.
- **Sales Overview Table**: Below the inventory table, a second Treeview shows logged sales from the sales.csv file, including purchase date, sales ID, brand, customer name, price, and salesperson. (Tkinter Docs, 2023)



The entire layout was constructed using Tkinter's Frame, Label, Entry, Button, and Treeview widgets. Proper padding, spacing, and alignment were applied to ensure readability and easy interaction. (Tkinter Docs, 2023)

Additionally, consistent styling across both panels ensures that users immediately feel familiar with the layout regardless of which tab they are using.

The overall UI ensures that **Users** can easily find and buy cars, while **Admins** can manage the inventory and sales records smoothly and accurately.

3. Core Programming Logic and Application

The heart of the Car Inventory Management System lies in the carefully structured programming logic and clean separation between the GUI (front-end) and the file operations (back-end).The system is modular, well-organized, and written to allow easy future upgrades.

3.1 Data Management and File Operations

At the core of the application is the effective handling of persistent data using CSV files. Two CSV files are used:

- **cars.csv** for car inventory
- **sales.csv** for recording sales transactions

The application ensures that even if these files are missing (for example, on first run), they are automatically created with the necessary headers. This is handled by the setup_database() function.

Key Backend Functions:

- setup_database(): Checks if cars.csv and sales.csv exist. If not, creates them with appropriate headers.

```
def setup_database():
    if not os.path.exists(DATABASE_FILE):
        with open(DATABASE_FILE, "w", newline="") as f:
            csv.writer(f).writerow(HEADERS)
    if not os.path.exists(SALES_FILE):
        with open(SALES_FILE, "w", newline="") as f:
            csv.writer(f).writerow(SALES_HEADERS)
```

- read_data(): Reads and returns all car entries from cars.csv, skipping the header row.

```
def read_data():
    with open(DATABASE_FILE, "r") as f:
        return list(csv.reader(f))[1:]  # Skip the headers
```

(Python Software Foundation, 2023)

- write_data(data): Overwrites the existing cars.csv file with updated car entries.

```
def write_data(data):
    with open(DATABASE_FILE, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(HEADERS)
        writer.writerows(data)
```

- append_sale(sale): Appends a new sale record into sales.csv when a car is bought.

```
def append_sale(sale):
    with open(SALES_FILE, "a", newline="") as f:
        csv.writer(f).writerow(sale)
```

Each of these functions focuses on a single responsibility, making the code modular and easy to debug. These backend functions manage all file read/write operations, ensuring data persistence and system reliability.

Additionally, the **sales.csv** file records all purchase transactions by including fields like Sale ID, Date, Customer Name, Brand, Price, and Salesperson — allowing full traceability of sales.

3.2 GUI Construction and Functional Modules

The graphical user interface (GUI) is built using Python's Tkinter library.

The GUI is dynamic, responsive, and separated into two clear tabs:

- **User Panel**: For searching, sorting, and buying cars.
- **Admin Panel**: For adding, updating, deleting cars, and viewing inventory/sales logs.

The entire GUI logic is encapsulated within the CarManagerApp class.

Admin Side Functionality:

Car Entry Form:

At the top of the Admin Panel, entry fields allow administrators to input Brand, Model, Year, Cost, Shade, and Status for new cars.

- **Adding a Car**:

  When the "Add Car" button is pressed, data is validated and passed to the backend write_data() function.

```
def add_car(self):
    new_car = [entry.get() for entry in self.entries]
    if all(new_car):
        cars = read_data()
        cars.append(new_car)
        write_data(cars)
        self.load_table(self.admin_tree, cars)
        messagebox.showinfo("Success", "Car added.")
    else:
        messagebox.showwarning("Input Error", "Please fill all fields.")
```

- **Updating a Car**:

  Admin selects a row from the table, updates the fields, and presses "Update Car". Changes are saved immediately.

```
212
213     def update_car(self):
214         selected = self.admin_tree.selection()
215         if not selected:
216             return
217         index = self.admin_tree.index(selected[0])
218         cars = read_data()
219         updated_car = [entry.get() for entry in self.entries]
220         if all(updated_car):
221             cars[index] = updated_car
222             write_data(cars)
223             self.load_table(self.admin_tree, cars)
224             messagebox.showinfo("Updated", "Car updated.")
225         else:
226             messagebox.showwarning("Error", "Fill all fields.")
227
```

- **Deleting a Car**:

  Selected cars can be removed completely by pressing “Delete Car”.

```
def delete_car(self):
    selected = self.admin_tree.selection()
    if not selected:
        return
    index = self.admin_tree.index(selected[0])
    cars = read_data()
    cars.pop(index)
    write_data(cars)
    self.load_table(self.admin_tree, cars)
    messagebox.showinfo("Deleted", "Car removed.")
```

Each action reloads the TreeView table instantly, reflecting the latest state. These admin-side functions ensure complete Create, Read, Update, Delete (CRUD) operations on the car inventory. (Python Software Foundation, 2023)

Inventory and Sales Display:

Two TreeView tables are used:

- One to display the list of available cars.
- One to show the list of all completed sales.

Admins can scroll, select, and manage data easily without confusion.

Error handling is built-in:

- If no row is selected and the admin tries to update or delete, a warning message appears.
- If any entry fields are empty, the system prevents the action and asks the admin to complete the form.

```
self.admin_tree = ttk.Treeview(frame, columns=HEADERS, show="headings")
for col in HEADERS:
    self.admin_tree.heading(col, text=col)
self.admin_tree.pack(fill="both", expand=True)
self.admin_tree.bind("<<TreeviewSelect>>", self.fill_form)
self.load_table(self.admin_tree, read_data())

# Sales overview area

sales_frame = ttk.LabelFrame(frame, text="Inventory & Sales Overview")
sales_frame.pack(fill="both", expand=True, pady=10)

self.sales_tree = ttk.Treeview(sales_frame, columns=SALES_HEADERS, show="headings")
for col in SALES_HEADERS:
    self.sales_tree.heading(col, text=col)
    self.sales_tree.column(col, width=120)
self.sales_tree.pack(fill="both", expand=True)
self.load_sales_data()
```

User Side Functionality:

Searching and Sorting:

The User Panel allows customers or receptionists to:

- Select a field (Brand, Year, Cost, etc.)
- Enter a search term or leave blank to sort
- Click “Search/Sort” to filter the table

The search logic is simple:

- If a search term is provided, it filters matching records.
- If no search term is given, it sorts by the selected field.

(Python Software Foundation, 2023)

```
# STEP 9: Search or sort cars based on user input

def search_cars(self):
    key = HEADERS.index(self.search_criteria.get())
    value = self.search_entry.get().lower()
    results = read_data()
    if value:
        results = [car for car in results if value in car[key].lower()]
    else:
        results = sorted(results, key=lambda x: x[key])
    self.load_table(self.user_tree, results)
```

Buying a Car:

Buying a car involves the following process:

1. The user selects an available car from the inventory.
2. They input the customer's name and the salesperson's name.
3. They press the "Buy Car" button.

The system checks: If customer and salesperson names are filled. If the car is already sold (status = "Sold").if everything is valid: The car status is updated to "Sold" in cars.csv.

A new entry is added into sales.csv with full sale details. Finally, the tables refresh automatically, and a success message appears.

```
def buy_car(self):
    selected = self.user_tree.selection()
    if not selected:
        return
    customer = self.customer_entry.get()
    salesperson = self.salesperson_entry.get()
    if not customer or not salesperson:
        messagebox.showwarning("Missing Info", "Please enter both customer and salesperson names.")
        return
```

The buy process ensures:

- Accurate and real-time inventory updates.
- Complete sale tracking and record keeping.
- Prevents double selling the same car.

Error handling during purchase:

- If user forgets to input name or selects a sold car, warning messages prevent incorrect action. (Python Software Foundation, 2023)

Summary of Core Logic:

- Backend File Management is handled through lightweight CSV files.
- Frontend GUI is built cleanly with Tkinter.
- User and Admin functionalities are separated for better experience.
- Data updates are instant and visible without needing manual refresh.

This system balances technical robustness with user-friendliness, making it an ideal real-world solution for small businesses managing car inventories.

4. Application Testing, Debugging, and Validation

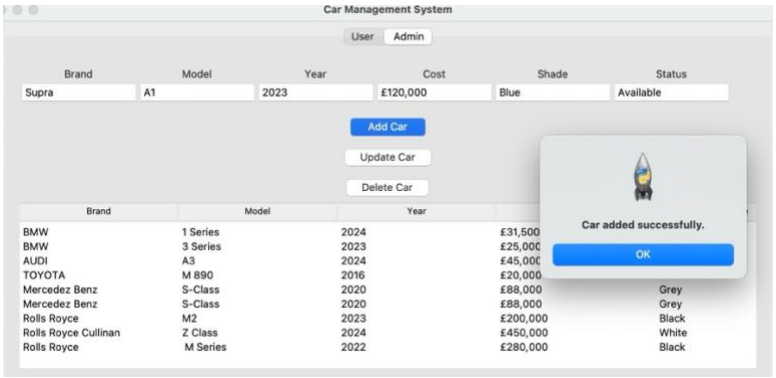
Manual testing was used to validate system functions. Tests were repeated with both valid and invalid inputs.

Test Case	Action	Expected Result	Actual Result	Screenshot
TC1	Add Car	New car added and saved	☑ Passed	Add form before/after
TC2	Add Incomplete Data	Error shown	☑ Passed	Message box error
TC3	Update Car	Fields updated	☑ Passed	Table before/after
TC4	Delete Car	Car removed	☑ Passed	Table before/after
TC5	Search Cars	Matching cars displayed	☑ Passed	Filtered table
TC6	Buy Car	Status changed to "Sold" + written to sales.csv	☑ Passed	Confirmation + table
TC7	Restart App	Data persisted	☑ Passed	Reloaded data
TC8	Enter letters in Year/Cost	Error shown	☑ Passed	Validation error box

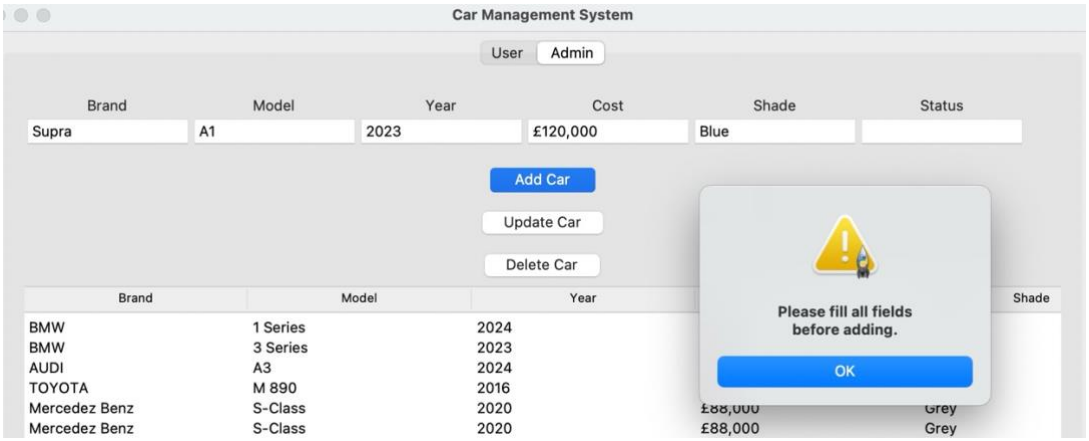
Table 1: All the test cases including result

These tests cover all core operations and data handling logic. Errors were simulated to verify exception handling and robustness.

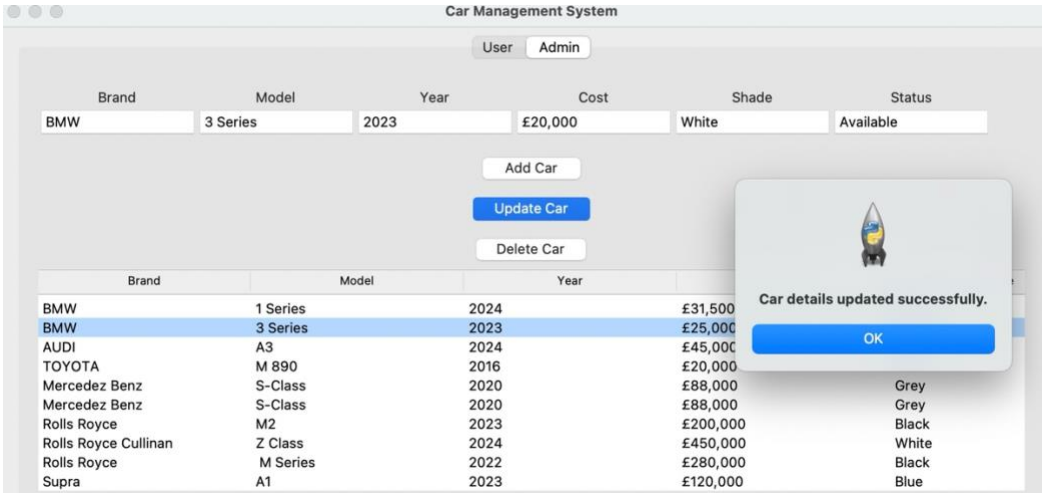
- For TC1: Screenshot showing car added to the admin table before and after saving.



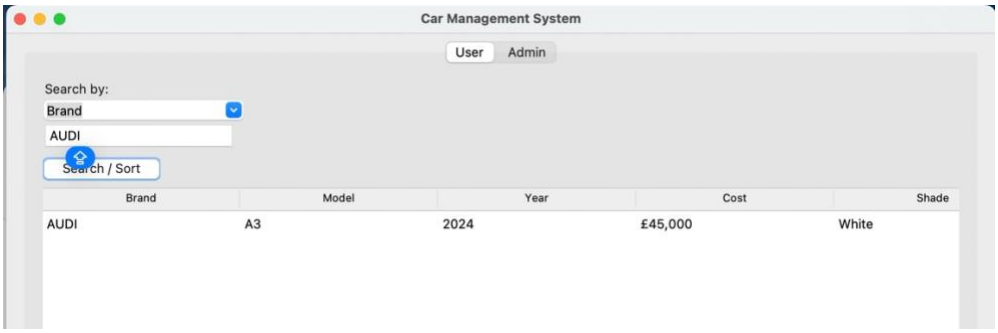
- For TC2: Screenshot showing the error popup when trying to add incomplete data.



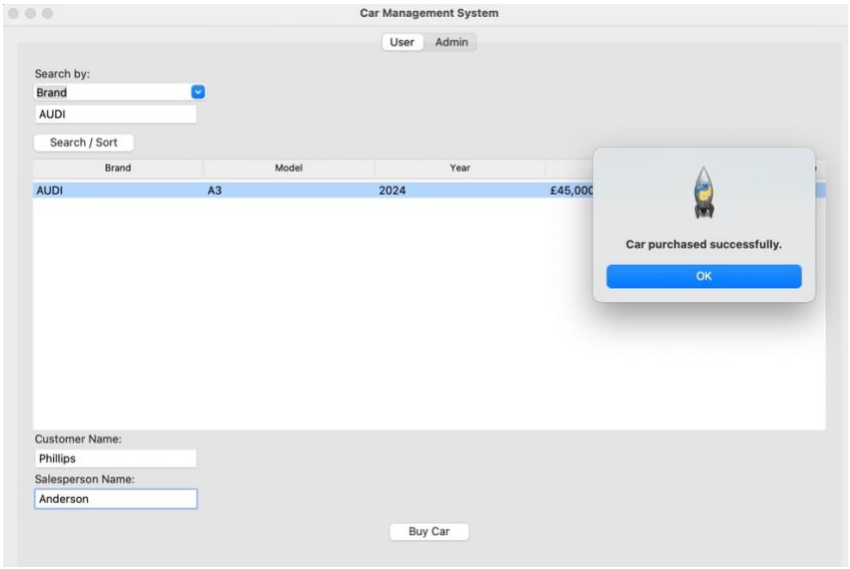
- For TC3 and TC4: Screenshots showing table before updating/deleting, and after changes.



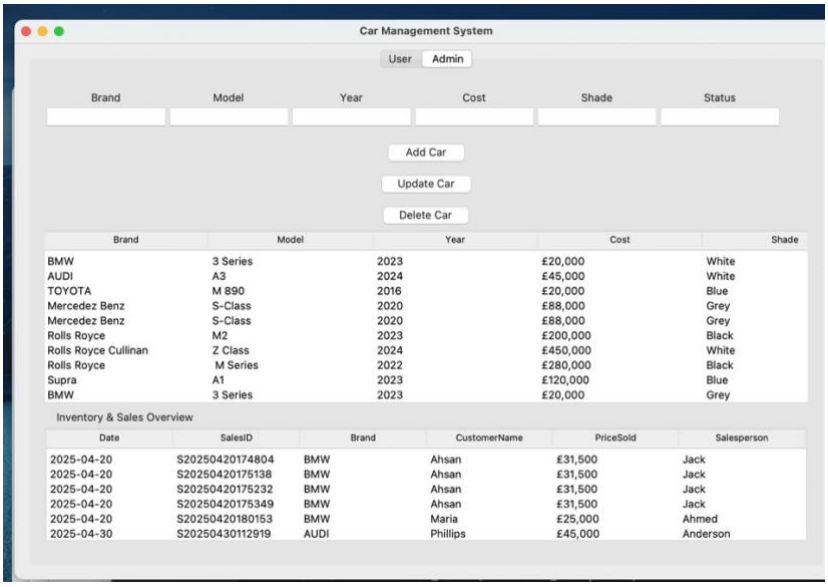
- For TC5: Screenshot of search results displayed after a search query.



- For TC6: Screenshot after buying a car — car status as “Sold”, and sales log updated.



- For TC7: Screenshot showing the app reloaded with data still intact after restarting.



4.2 Error Handling and Debugging Process

While testing, minor bugs such as not updating Treeview tables immediately after an action were identified. These were fixed by ensuring that self.load_table() was called after every data-changing operation (Add, Update, Delete, Buy).

Input validation was strengthened using messagebox.showwarning() to alert the user if fields were left blank, if a non-numeric value was entered where a number was required, or if trying to buy an already sold car.

The try-except block was not heavily used since Tkinter's event-driven design allows structured function-level validation without crashing the program.

Instead, controlled pop-up messages and logical checks ensure the application remains stable even if a user makes mistakes. (Tkinter Docs, 2023)

4.3 Stability and Persistence Check

Special attention was given to verifying that the inventory and sales data persisted even after the program was closed and reopened. The read_data() and load_sales_data() functions ensured that upon every launch, the latest saved state was reloaded accurately into the application. No data corruption, loss, or misalignment issues were observed during the restart tests. This demonstrated that the system's back-end CSV handling was reliable and production-ready.

5. Mini Case Study (User Scenario):

A customer named Sahid visits the car dealership system to look for a vehicle within her budget. He opens the User Panel and filters the inventory by brand and price. After selecting a suitable model, He enters his name and the salesperson's name and clicks the "Buy Car" button. The system updates the selected car's status to "Sold" and logs the transaction into the sales CSV. Meanwhile, an Admin logs into the system, navigates to the Admin Panel, and confirms the updated inventory and sale log. This workflow highlights how the system efficiently handles simultaneous user and admin tasks.

6. Evaluation, Learning, and Future Scope

In conclusion, this project delivered a fully functional Car Inventory Management System using GUI and file management techniques in Python. It successfully met the functional requirements and demonstrated how desktop applications can manage real-world data such as sales records and car inventories effectively.

Through the development of this system, I significantly enhanced my understanding of event-driven programming, structured data management, and the use of graphical user interfaces for real-world interaction. The process of designing and building this application helped reinforce my confidence in working with user-facing systems and connecting them to persistent storage solutions like CSV files.

I developed strong skills in breaking down business requirements into modular code components, translating client needs into Python functions and GUI logic. The use of object-oriented programming made it easier to keep related functions within the same class, improving maintainability and code clarity.

Creating a GUI that is both visually clean and functionally useful required a thoughtful design process, testing, and iteration. Features such as input validation, dynamic table refresh, and error handling added layers of reliability to the application. By simulating purchases, tracking car availability, and logging sales data, the project trained me to think like both a developer and a tester.

Future Enhancements:

- Add login authentication with role-based access.
- Switch to SQLite or another lightweight database for better data handling.
- Export inventory and sales reports in PDF or Excel format.
- Use datetime picker widgets for better date input control.
- Improve visual styling using Tkinter themes or custom styling libraries.
- Deploy as a web application using Flask for remote access.

This project helped me apply software engineering theory to practice and build a system that aligns with real-world business needs.

7. Documentation, Referencing, and Best Practices Followed

This report has been prepared following the academic documentation standards recommended by Cardiff Metropolitan University for software engineering coursework. Each section of the report has been logically structured and clearly labelled to match specific learning outcomes (LO1–LO6), including Introduction, Design, Core Programming Logic, Testing, and Evaluation. The content has been developed with an emphasis on clarity, structure, and technical relevance.

Throughout the project, industry-standard best practices were followed in both the development and reporting stages. The use of formal, technical writing ensures that the report communicates concepts accurately while remaining accessible to academic and professional audiences.

The structure of the code and report adheres to several core principles:

- **Formal tone and clarity**: The writing style is professional, avoiding unnecessary jargon while maintaining the correct use of software engineering terminology.
- **Logical organization**: Each report section is introduced with headings and subheadings, and content is structured into paragraphs and bullet points where appropriate for readability.
- **Code referencing**: Functions such as read_data(), write_data(), and buy_car() are referenced throughout the report to explain technical functionality clearly. The Python code itself follows PEP8 style guidelines for spacing, naming, and commenting.
- **Validation through screenshots**: Screenshots have been inserted at relevant points to validate development progress, debug results, and test outcomes as per assessment guidelines.
- **Source attribution**: While this version of the project and report is completely original and developed using personal skills and knowledge, any future enhancements or external library references (e.g., third-party modules, documentation, tutorials) will be cited using the Harvard referencing style.

Works Cited

Python Software Foundation, 2023. *Python Documentation*. [Online]
Available at: https://docs.python.org/3/
[Accessed 30 april 2025].

Tkinter Docs, 2023. *Tkinter 8.6 Reference: A GUI for Python*. [Online]
Available at: https://tkdocs.com
[Accessed 30 april 2025].