Module: GUI Design and Database Connection

Module Code: COMP4604

Course Code: TU819/4

Course Group: Group A/B

Student Names: Jason Leonard & Brian Twene

Student IDs: C19500766, C19344543

Submission Date: 04/12/2022

Topic: Vehicle Dealership/Rental System

Introduction

This design document is based on a software solution for a vehicle dealership which allows a user to both purchase vehicles, rent vehicles and return vehicles when they're finished using them. Additionally, the program also allows an admin to alter a users and vehicles table which is located

within a Microsoft Access database, and allows the admin to add, edit, and remove information from their respective tables. Our application also features stock and inventory control, and allows management to print a list of users and a list of our stock and inventory. Lastly, an admin is also able to use this program to check overdue accounts for people who have not returned their rented vehicle.

High Level Description of the System

In order to access the vehicle dealership program, the user must first enter the application. Once the software application is running and they have gained access to the software itself, they will initially be prompted with a screen looking for login credentials to gain further access to the program. As a new user won't have any credentials to login with, they must click on the register button which will be on the bottom left of the left panel on the program window.

Once clicked, the user will be brought onto a new page which will feature five different sets of credentials that must be entered, the required credentials that need to be input range from a customer id, username, password, first name and last name. Once the user has input all of their credentials they will then click on the register button on the bottom center of the right panel, this press of a button will then initiate an SQL statement which will gather all of the information input by the user and insert it into the users table.

Afterwards they will be redirected back to the login screen where their credentials can be entered into three parts requiring information, consisting of a username, password, and a drop down combo box which identifies what role they are, either customer or admin. If the user has entered credentials that are in line with an admin's credentials then the user will be brought to the admin's main menu, if the user has entered credentials that are in line with a customer's credentials then the user will be brought to the customers main menu. If the user has been marked as a customer in our database yet is trying to login as an admin, then they will be prompted with a dialog message that states "Login Failed, Please Try Again!", same goes for an admin trying to login as a customer. All of the information input by the user is checked, and if the information is incorrect then the program will state so and ask the user to try entering credentials again. Additionally, when the user clicks login after their credentials have been entered, it will then initialise an SQL statement which will select all of the information for the specific username entered and check if it's correct within the database.

After the login process, the user will be brought to a main menu which features a large vehicle which is identified in this case as a button, once this vehicle or button is clicked they will be brought to a page which will allow them to both purchase and rent vehicles. This page will also feature a plethora of information about vehicles, from the name of the vehicle, picture of the vehicle, price of the vehicle (both daily rate price for renting and total purchase price), etc.

As for the admin, after their login credentials have been verified through our system they will be directed over to a main menu which consists of two large buttons which when either is clicked will bring you to a manage users or a manage vehicles page.

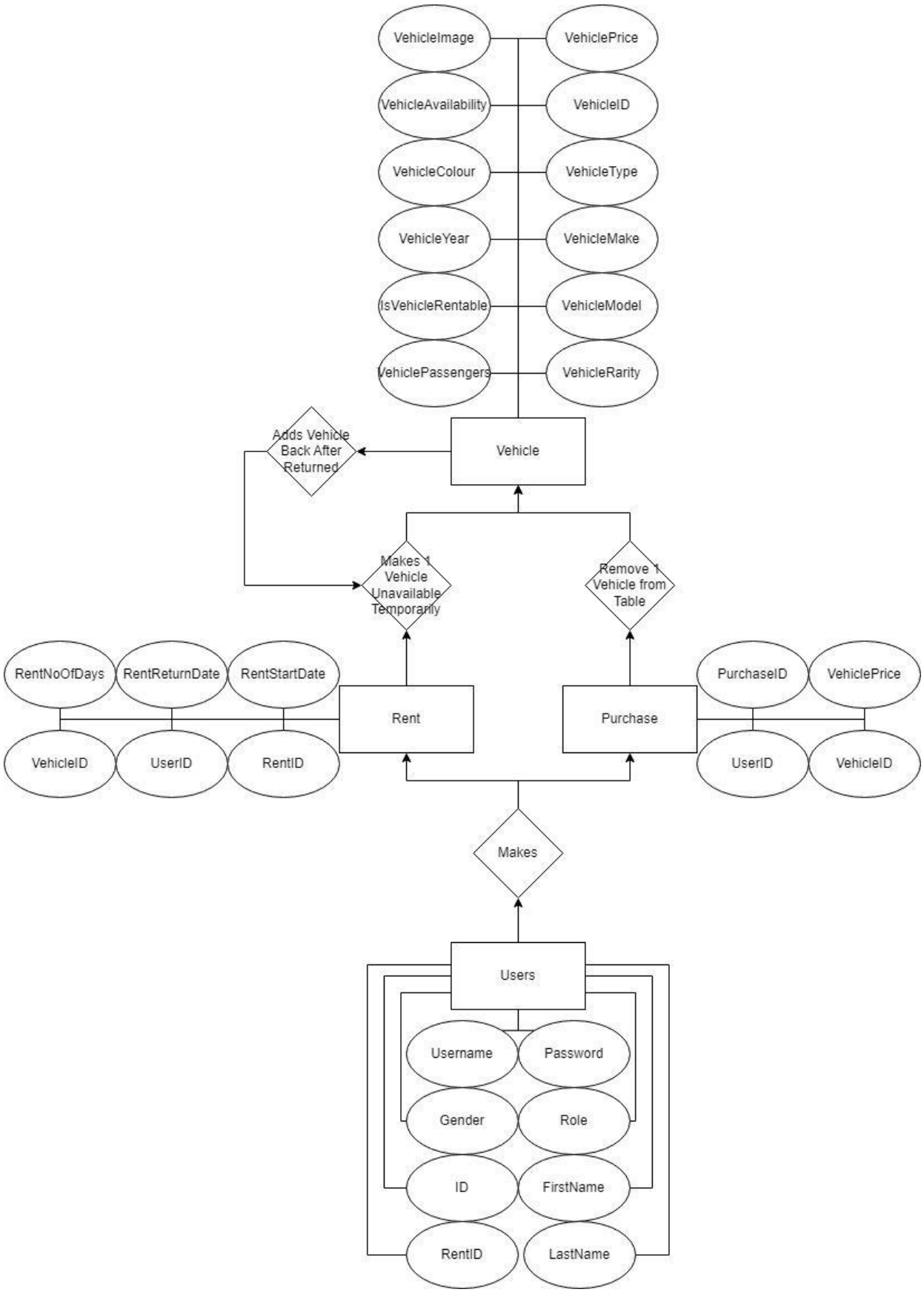
The management of vehicles and users is possible on the pages the admin has access to and a plethora of information is available on these pages which includes tables to show the current customers and vehicles within the database and user input section which allow the option to add, edit or remove information from specific tables within the database. When adding information, the program will send an INSERT INTO SQL statement to the database, when editing information the program will send an UPDATE SQL statement to the database and lastly when the admin would like to remove information, the program will then send a DELETE FROM SQL statement to the database and promptly delete whatever information that was specified to be removed.

Diagrams & Flow Charts

Enhanced Entity–Relationship Model Diagram

As can be seen below this is our EERD or in better terms, our enhanced entity-relationship model diagram for our program. It accurately shows each table that we have created within our database with each object that is connected to it. Our Vehicle table required a plethora of information so that our user could better identify what vehicle they were renting or purchasing and thus we had to create twelve objects inside of it. Our Purchase table displays four objects with three of those objects having a direct relationship with two different tables (UserID in User and both VehiclePrice and VehicleID in Vehicle). The Users table much like the Vehicle table also required a lot of information to be processed and thus we had to create eight objects within the realms of it, with only one of these objects having a relationship with another table (RentID in Rent). Lastly, we have the Rent table which has six objects attached to it, with two of these objects (VehicleID and UserID) being linked closely via a relationship with both the Vehicle and Users tables.

To better describe the diagram below, essentially a user makes either a decision to rent or purchase a vehicle, once done the effect it has will either temporarily make the vehicle unavailable in the database via a yes/no data type or permanently delete the vehicle from the database. If the user decides to rent the vehicle, after a certain period of time and when the customer decides to return the vehicle, it will be made available again within the database.


VehicleImage
VehiclePrice
VehicleAvailability
VehicleID
VehicleColour
VehicleType
VehicleYear
VehicleMake
IsVehicleRentable
VehicleModel
VehiclePassengers
VehicleRarity
Vehicle
Adds Vehicle Back After Returned
Makes 1 Vehicle Unavailable Temporarily
Remove 1 Vehicle from Table
RentNoOfDays
RentReturnDate
RentStartDate
Rent
VehicleID
UserID
RentID
Purchase
PurchaseID
VehiclePrice
UserID
VehicleID
Makes
Users
Username
Password
Gender
Role
ID
FirstName
RentID
LastName

Relational Model Diagram

Below is our RD (relational model diagram) for our software application. It shows both the Schema and State which clearly identifies each object within the database and the type of data that it stores.

Schema

Vehicle
VehicleID
VehicleMake
VehicleModel
VehicleYear
VehicleColour
VehicleAvailability
VehiclePassengers
VehicleType
VehicleRarity
VehicleImage
VehiclePrice
IsVehicleRentable

State

Vehicle
1, 2, 3, 4, 5, etc.
Toyota, BMW, Audi, etc.
Camry, M5, A6, etc.
2002, 2016, 2022, etc.
Silver, Black, Red, etc.
YES OR NO.
1, 3, 4, 6.
Sport, Sedan, Compact, etc.
Luxury, Premium, Standard, Budget.
Link.
149999, 49999, 9999, 19999, etc.
YES OR NO.

Users
ID
Username
Password
FirstName
LastName
Gender
Role
RentID

Users
1, 2, 3, 4, 5, etc.
JasonLeonard, JohnSmith, BrianTwene, etc.
123456, 654321, etc.
Jason, Brian, John, etc.
Leonard, Twene, Smith, etc.
Male, Female, etc.
Admin, Customer.
1, 2, 3, 4, 5, etc.

Rent
RentID
RentStartDate
RentReturnDate
RentNoOfDays
VehicleID
UserID

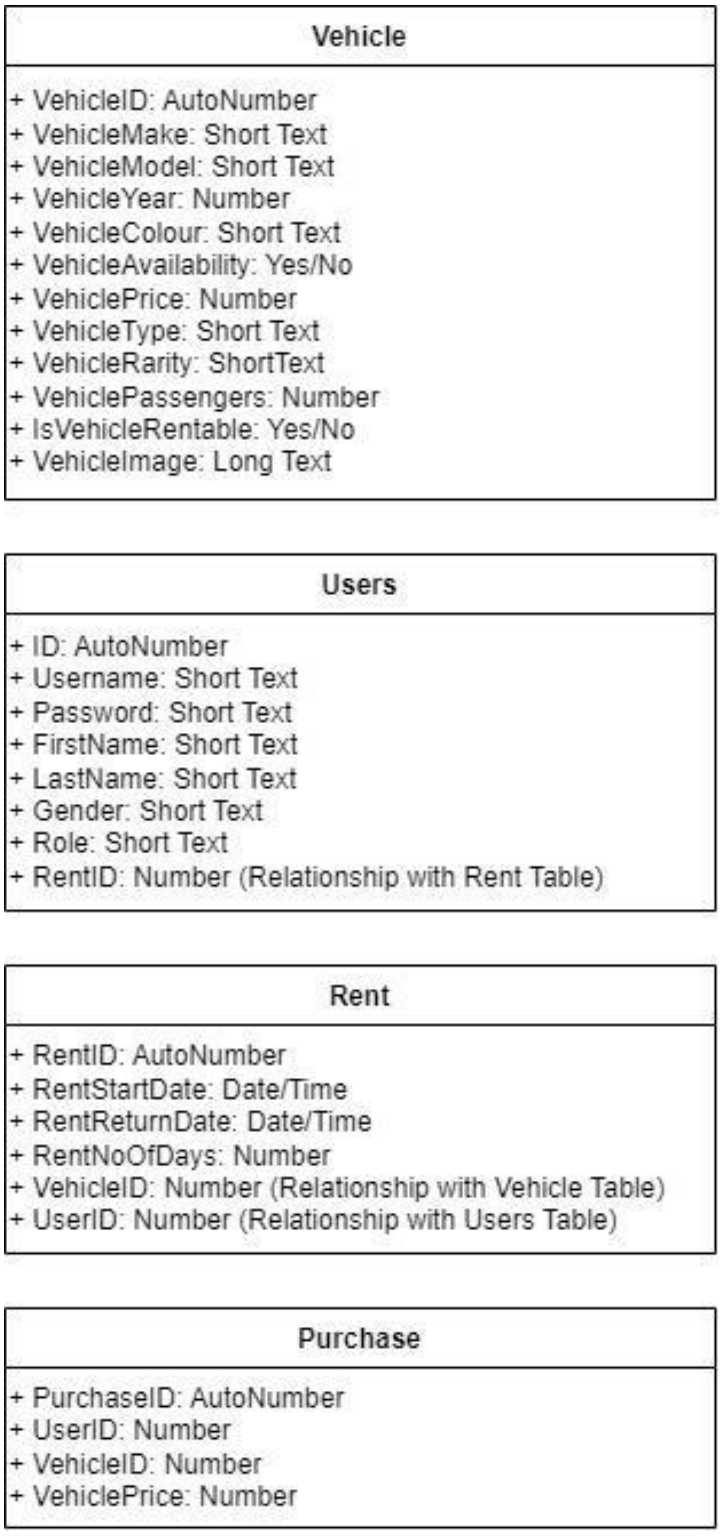
Rent
1, 2, 3, 4, 5, etc.
30/11/2022, 07/12/2022, 14/12/2022, etc.
07/12/2022, 14/12/2022, 21/12/2022, etc.
7, 14, 21, etc.
1, 2, 3, 4, 5, etc.
1, 2, 3, 4, 5, etc.

Purchase
PurchaseID
UserID
VehicleID
VehiclePrice

Purchase
1, 2, 3, 4, 5, etc.
1, 2, 3, 4, 5, etc.
1, 2, 3, 4, 5, etc.
149999, 49999, 9999, 19999, etc.

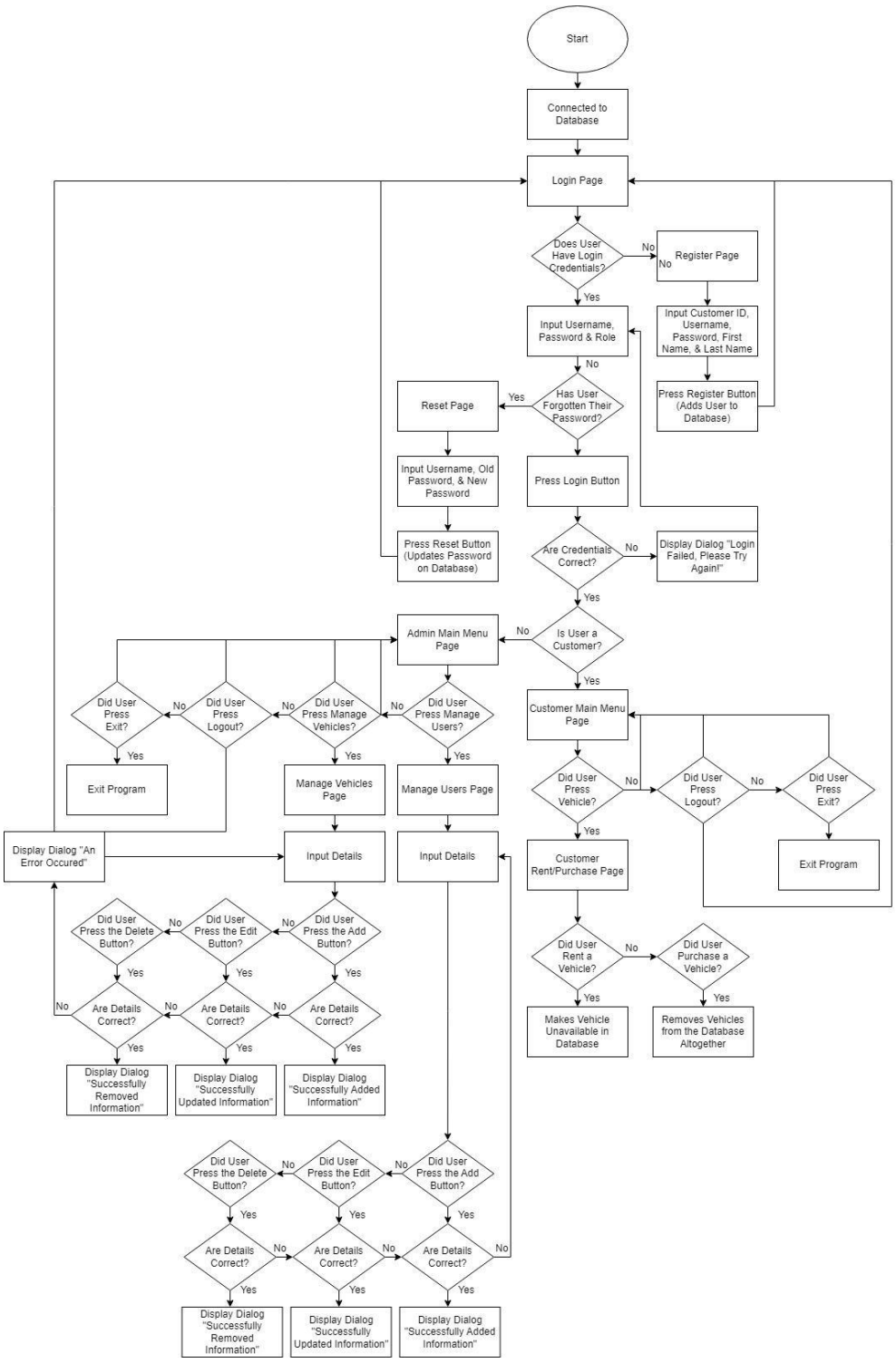
UML Class Diagram

The UML Class diagram is a graphical notation used to construct and visualise object oriented systems. It features all four tables within the database, each of their field names and each of their data types.



Flowchart Diagram

Our flowchart diagram shows all of the working behind our software solution, it features what happens when the end user presses certain buttons, shows what happens with user input checks, and even displays what occurs when an admin decides to add, update, or remove information from the database.



Persistent Storage in Databases

A persistent database holds records or objects that remain intact even when hardware and software are changed. Data that is persistent is reliable and recoverable. Records and tables are the primary formats for durable data storage in traditional relational database management systems (RDBMS). They are unable to store items and their connections, nevertheless. Encapsulation, inheritance, persistence, and polymorphism are essential characteristics of objects that do not convert well into records and tables. In order to store objects and maintain object persistence, specialised databases like object-oriented database management systems (OODBMS) and object relational database management systems (ORDBMS) are required.

Importance of Persistent Storage in Databases:

- Data that is persistent is static and does not evolve over time (not dynamic).
- Core information is stored in persistent data. Financial data for a business, for instance, needs to be durable.
- Data that is persistent is stable because external processes or objects cannot remove it until the user does.
- Time-independent data is persistent data. The data produced by an application or object is still available outside of object borders, other processes, or transactions long after the parent application or object has been removed.
- Data that is persistent keeps its original format. When the process is finished, any data that was stored in volatile memory is destroyed. However, non-volatile storage, such as persistent databases, retain their persistent data long after an application has been closed.
- Data that is persistent is non-volatile and resistant to power interruptions.
- Even when a machine is restarted or shut down, persistent data can still be recovered.
- Changes to persistent data are irreversible and accessible at all times.

Overview on the Usability Heuristics Adherence

Visibility of System Status

Users should always be kept up to date on developments by the design, which should provide pertinent feedback in a timely manner. We met this usability heuristic by adding a title to the top of each page stating what page the user is on such as Login for the login page, Register for the register page, Reset for the reset password page, etc. This allows the user to understand what page they are on within the program itself which in itself allows them to navigate through the program with ease. Additionally, we added buttons with titles attached to them which will redirect the user to certain pages, the titles may be Main Menu to go to the main menu page, etc.

Match Between System & the Real World

The interface should be user-friendly. Instead of using internal jargon, use words, statements, and ideas that the user is already familiar with. The entirety of our program is user-friendly from the interface to the design. We didn't use any sort of internal jargon and instead used simple words and statements. An example of this can be found when a user needs to login, register or even reset their password where in each text field it shows what needs to be added, such as it will say Username in the username text field and when the user clicks their mouse on it, the text field empties, allowing the user to type their username. All words we use such as Main Menu, Login, Register, Rent, Purchase, Logout, Exit, etc are all understandable words that a user doesn't have to go looking for the definition of.

User Control & Freedom

Users frequently act incorrectly. They require a prominent "emergency exit" so they can leave the undesirable action without going through a lengthy procedure. On each of our pages we have an X button at the top right either in red or white, which when pressed exits the program in its entirety, this X button/symbol is well known anywhere in programs and we decided to implement it in ours as it made the most sense.

Consistency & Standards

Users shouldn't have to guess whether certain expressions, circumstances, or actions are equivalent. Our whole program is consistent with each other from the design to the user interface. It is easy for a user to start our program up and understand where to go and what to do next thanks to the helpful design that we chose to use. In our manage users and manage vehicles section, we have three buttons which allow for either add, edit, or remove, these buttons cannot be mixed up with each other and means our user won't be confused with what each button does.

Error Prevention

While clear error warnings are vital, the greatest solutions take care to avoid problems altogether. Error-prone situations should either be eliminated, checked for, and given a confirmation choice before users commit to an action. With each user input we have in our program, it comes with an error prevention and check system that will contact the database to see if the information is correct. If the information is incorrect then the program will display a dialog stating so, this could be something that we have done which states whether the login credentials are wrong, to which it would display "Login Failed, Please Try Again!" it allows the user to understand that they've input the wrong information and allows them to input new information and try again.

Recognition Rather than Recall

Reduce the amount of memory required from the user by making components, actions, and options visible. Information shouldn't need to be remembered as the user moves from one area

of the interface to another. Field labels and menu items, for example, should be clearly visible or simple to find when using the design. We implemented a main menu into both our customer and admin side of the program. All the customer has to do is click one simple button when on their main menu which then brings them to the rental/purchase page, the same can be said for the admin's main menu all they have to do is click one out of two buttons, one for managing users and the other for managing vehicles. Additionally, the login, register and reset page all have the buttons in the same position, the button to send the information to the database is on the bottom right, whilst the button to go to other pages is on the bottom left, this is consistent throughout and easily recognisable.

Flexibility & Efficiency of Use

In order to accommodate both new and experienced users, shortcuts may speed up interaction for the expert user while remaining hidden from novice users. Permit users to customise routine tasks. We have added a plethora of buttons into our program which allows for users to quickly change between pages, it's efficient and flexible for the user and it allows them to easily navigate around the program.

Aesthetic & Minimalistic Design

Information that is unnecessary or infrequently used shouldn't be present in interfaces. Each additional piece of information that is added to an interface competes with the pertinent pieces and reduces their relative visibility. Our program has been designed with a minimalistic and aesthetic look. We've added images to each of our cars, and added nice colours to the background of each page that complement the program nicely, not only that but no matter what page you're on, the theme is consistent with our iconic purple, black, white and grey colours. The content we ask the user to input and the details we display are all easily visible in a way that we're very proud of, the user understands where they are at all times and the design isn't distracting in any way that would make the user confused about where they are and where they can go to next.

Help Users Recognise, Diagnose, & Recover from Errors

Error messages ought to be written in plain English (i.e., without using error codes), accurately describe the issue, and helpfully offer a remedy. Each of our error messages are simple to read and understand, if a user has failed to login with the correct credentials then the program will prompt the user with a dialog stating "Login Failed, Please Try Again!". The same can be said for when the user wants to register or reset their password, where if they want to do so they have to input the correct information and if not then it will state so with a dialog much like the login failed example above. This allows a user to understand where they have gone wrong and what's best is it enables them to diagnose the issues themself recovering what they had done wrong in the process.

Help & Documentation

Lastly, Help & Documentation is to put it best to ensure that users are able to fulfil their jobs and if they can't do so then they may look for some help along the way. We have created a user manual for this which states what each button does, what each page is, how to get to certain pages, and how to rent or purchase vehicles. It is with this user manual that it ensures that no matter what issues the user has, they can always return to the user manual to see what can be done to resolve said issue.