CENTRAL BOARD OF SECONDARY EDUCATION

CREDENCE HIGH SCHOOL
DUBAI

A PROJECT DOCUMENTATION IS SUBMITTED FOR THE COMPUTER SCIENCE CLASS 12 SESSION 2024-25

Submitted By – Vinayak Priya Vinesh

Subject Teacher (CS) – Ms. Thamizhmani

Class – Grade 12

Roll no. 25

CERTIFICATE

This is to certify that Vinayak Priya Vinesh,
student of Class XII A, Credence High School,
has completed the Project File during the
academic year 2024-25 towards partial fulfilment of
credit for the Computer Science
project evaluation of CBSE and submitted
satisfactory report, as compiled in the following pages,
under my supervision.

INTERNAL SIGNATURE: EXTERNAL SIGNATURE:

_____ _____

ACKNOWLEDGEMENT

I am immensely grateful for the support received during the completion of my solo Computer Science project. My deepest appreciation goes to my respected Ms. Thamizhmani for their invaluable guidance and expertise, shaping the direction of my work.

I am also thankful to my friends and classmates for their support, motivation, and insightful discussions that provided new perspectives. The unwavering support and belief from my parents have been a constant source of motivation.

I would like to acknowledge the online resources, tutorials, and documentation that enriched my understanding and aided me in overcoming challenges. To everyone involved, your support and contributions have been truly invaluable.

Thank you for being part of my journey.

~Vinayak Priya Vinesh

INDEX

CAR RENTAL SYSTEM

VINAYAK PRIYA VINESH

FRONT-END DESCRIPTION

INTRODUCTION:

THE FRONT-END OF THIS LUXURY CAR SHOWROOM MANAGEMENT SYSTEM IS BUILT USING PYTHON'S TKINTER LIBRARY, WHICH PROVIDES A GRAPHICAL INTERFACE FOR USERS. THIS INTERFACE ALLOWS SEAMLESS INTERACTION WITH THE SHOWROOM DATABASE, ENABLING USERS TO VIEW AVAILABLE CARS, THEIR IMAGES, AND DETAILS. TKINTER'S WIDGETS AND LAYOUT FLEXIBILITY ENABLE THE CREATION OF A MODERN, VISUALLY APPEALING DISPLAY THAT RESEMBLES A REAL-WORLD LUXURY SHOWROOM. EACH CAR'S IMAGE AND SPECIFICATIONS ARE ORGANIZED AND PRESENTED CLEARLY, AND FEATURES LIKE SCROLLING AND INTERACTIVE BUTTONS ENHANCE USABILITY, ENSURING THAT USERS CAN EXPLORE THE COLLECTION EFFORTLESSLY.

ADVANTAGES:

- SIMPLE YET POWERFUL GUI CREATION WITH TKINTER.
- PYTHON'S READABILITY MAKES THE FRONT-END EASY TO MAINTAIN AND MODIFY.
- INTERACTIVE WIDGETS AND BUTTONS IMPROVE USER EXPERIENCE.
- SMOOTH INTEGRATION WITH BACK-END DATABASES FOR LIVE DATA DISPLAY. IDEAL FOR RAPID DEVELOPMENT AND PROTOTYPING OF USER INTERFACES.

BACK-END DESCRIPTION

INTRODUCTION:

THE BACK-END OF THIS LUXURY CAR SHOWROOM MANAGEMENT SYSTEM USES MYSQL, A SOLID DATABASE THAT STORES DETAILS ABOUT EACH CAR, INCLUDING IMAGES AND USER DATA. MYSQL'S STRUCTURED SETUP KEEPS DATA ORGANIZED AND RELIABLE, MAKING IT EASY TO RUN EFFICIENT QUERIES AND RETRIEVE INFORMATION QUICKLY. THIS SETUP MEANS THE SYSTEM IS RESPONSIVE TO FRONT-END REQUESTS, SO USERS SEE UP-TO-DATE INVENTORY AND CAR DETAILS AS THEY BROWSE. PYTHON CONNECTS SMOOTHLY WITH MYSQL, MAKING THE APPLICATION HIGHLY RESPONSIVE, DELIVERING REAL-TIME UPDATES, AND HANDLING DATA WITH MINIMAL LAG.

ADVANTAGES:

- DEPENDABLE AND EXPANDABLE STORAGE WITH MYSQL.
- QUICK DATA RETRIEVAL FOR A RESPONSIVE BROWSING EXPERIENCE.
- STRUCTURED TO MAINTAIN DATA ACCURACY AND RELIABILITY.
- REAL-TIME UPDATES THROUGH EASY PYTHON-MYSQL INTEGRATION. IDEAL FOR HANDLING LARGE CAR INVENTORIES EFFICIENTLY.

PROJECT DESCRIPTION

Aim:

The Car Rental Management System aims to revolutionize the way luxury car rentals are managed by providing a streamlined, user-friendly interface that simplifies the browsing and booking process, ensuring a seamless experience for customers.

Features:

- A visually engaging interface that allows users to effortlessly explore a diverse range of luxury cars.
- Robust MySQL database integration for secure and efficient storage of car specifications, images, and rental records.
- Dynamic functionality to display available cars along with high-quality images and detailed descriptions.
- A comprehensive dropdown menu providing quick access to features like Home, Add User, and Customer Profiles.
- User profile management that tracks rental history and preferences, enhancing customer experience.
- Efficient booking and return functionalities that ensure smooth operations and high levels of user satisfaction.

DATABASE FORMATION

CREATING DATABASE AND TABLES

```sql
CREATE DATABASE lux_cars;
USE lux_cars;

CREATE TABLE IF NOT EXISTS users (
    user_id INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL,
    user_mail VARCHAR(50) NOT NULL
);

CREATE TABLE IF NOT EXISTS cars (
    id INT AUTO_INCREMENT PRIMARY KEY,
    make VARCHAR(100) NOT NULL,
    model VARCHAR(100) NOT NULL,
    year INT NOT NULL,
    color VARCHAR(50),
    rented_by INT,
    image_path VARCHAR(255),
    price DECIMAL(15, 2),
    FOREIGN KEY (rented_by) REFERENCES users(user_id)
);


CREATE TABLE IF NOT EXISTS borrows (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    car_id INT NOT NULL,
    borrow_date DATE NOT NULL,
    return_date DATE DEFAULT NULL,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (car_id) REFERENCES cars(id)
);

CREATE TABLE available_cars (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    image_path VARCHAR(255),
    price DECIMAL(10, 2) NOT NULL
);

```

INSERTING RECORDS

```
42 •  INSERT INTO users (user_id, name, address, user_mail) VALUES
43    (1, 'Alice Johnson', '123 Elm Street', 'alice@gmail.com'),
44    (2, 'Bob Smith', '456 Maple Avenue', 'bob@gmail.com'),
45    (3, 'Charlie Brown', '789 Pine Lane', 'charlie@yahoo.com'),
46    (4, 'Diana Prince', '101 Oak Boulevard', 'diana@hotmail.com'),
47    (5, 'Ethan Hunt', '202 Cedar Road', 'ethan@gmail.com'),
48    (6, 'Fiona Apple', '303 Birch Street', 'fiona@yahoo.com');
49
50    -- Inserting Cars
51 •  INSERT INTO cars (make, model, year, color, rented_by, image_path, price) VALUES
52    ('Aston Martin', 'DB11', 2021, 'Silver', NULL, 'D:/CS Project/Car Rental/AstonMartinDB11.jpg', 950000.00),
53    ('Bentley', 'Continental GT', 2022, 'Black', NULL, 'D:/CS Project/Car Rental/BentleyContinentalGT.jpg', 900000.00),
54    ('Ferrari', '488 GTB', 2020, 'Red', NULL, 'D:/CS Project/Car Rental/Ferrari488GTB.jpg', 980000.00),
55    ('Lamborghini', 'Huracán', 2021, 'Yellow', NULL, 'D:/CS Project/Car Rental/LamborghiniHuracan.jpg', 1000000.00),
56    ('Porsche', '911 Turbo S', 2022, 'White', NULL, 'D:/CS Project/Car Rental/Porsche911TurboS.jpg', 850000.00),
57    ('Rolls-Royce', 'Phantom', 2023, 'Gray', NULL, 'D:/CS Project/Car Rental/RollsRoycePhantom.jpg', 990000.00),
58    ('Maserati', 'GranTurismo', 2021, 'Blue', NULL, 'D:/CS Project/Car Rental/MaseratiGranTurismo.jpg', 870000.00),
59    ('Mercedes-Benz', 'S-Class', 2022, 'Dark Green', NULL, 'D:/CS Project/Car Rental/MercedesBenzSClass.jpg', 860000.00),
60    ('McLaren', '720S', 2021, 'Orange', NULL, 'D:/CS Project/Car Rental/McLaren720S.jpg', 970000.00),
61    ('Bugatti', 'Chiron', 2023, 'Light Blue', NULL, 'D:/CS Project/Car Rental/BugattiChiron.jpg', 1000000.00),
62    ('Jaguar', 'F-Type', 2021, 'Black', NULL, 'D:/CS Project/Car Rental/JaguarFType.jpg', 860000.00),
63    ('Tesla', 'Model S', 2022, 'White', NULL, 'D:/CS Project/Car Rental/TeslaModelS.jpg', 850000.00),
64    ('BMW', 'M8', 2021, 'Dark Red', NULL, 'D:/CS Project/Car Rental/BMWM8.jpg', 890000.00),
65    ('Audi', 'R8', 2022, 'Silver', NULL, 'D:/CS Project/Car Rental/AudiR8.jpg', 880000.00),
```

TABLES FORMED:-

1. TABLE - USERS:

user_id	name	address	user_mail
1	Alice Johnson	123 Elm Street	alice@gmail.com
2	Bob Smith	456 Maple Avenue	bob@gmail.com
3	Charlie Brown	789 Pine Lane	charlie@yahoo.com
4	Diana Prince	101 Oak Boulevard	diana@hotmail.com
5	Ethan Hunt	202 Cedar Road	ethan@gmail.com
6	Fiona Apple	303 Birch Street	fiona@yahoo.com
7	Vinayak	Al Barsha, Dubai	vinayak@gmail.com
8	Adil	Al Quoz	adil@gmail.com
9	Adarsh	Samari	adarsh@gmail.com
10	Mani	Al Quoz	mani@gmail.com
NULL	NULL	NULL	NULL

2. TABLE - CARS:

id	make	model	year	color	rented_by	image_path
1	Aston Martin	DB11	2021	Silver	NULL	NULL
2	Bentley	Continental GT	2022	Black	NULL	NULL
3	Ferrari	488 GTB	2020	Red	NULL	NULL
4	Lamborghini	Huracán	2021	Yellow	NULL	NULL
5	Porsche	911 Turbo S	2022	White	NULL	NULL
6	Rolls-Royce	Phantom	2023	Gray	NULL	NULL
7	Maserati	GranTurismo	2021	Blue	8	NULL
8	Mercedes-Benz	S-Class	2022	Dark Green	NULL	NULL
9	McLaren	720S	2021	Orange	NULL	NULL
10	Bugatti	Chiron	2023	Light Blue	NULL	NULL
11	Jaguar	F-Type	2021	Black	NULL	NULL
12	Tesla	Model S	2022	White	NULL	NULL
13	BMW	M8	2021	Dark Red	NULL	NULL
14	Audi	R8	2022	Silver	NULL	NULL
15	Lexus	LC 500	2022	Purple	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

TABLES FORMED:-

3. TABLE - BORROWS:

id	user_id	car_id	borrow_date	return_date
1	2	8	2024-11-03	2024-11-03
2	3	10	2024-11-03	2024-11-03
3	5	3	2024-11-03	2024-11-03
4	2	4	2024-11-03	2024-11-03
5	7	6	2024-11-04	2024-11-04
6	8	7	2024-11-04	2024-11-04
7	8	10	2024-11-04	2024-11-04
8	10	8	2024-11-04	2024-11-04
9	8	7	2024-11-05	NULL
NULL	NULL	NULL	NULL	NULL

4. TABLE - AVAILABLE_CARS:

id	name	image_path
1	Aston Martin DB11	D:/CS Project/Car Rental/AstonMartinDB11.jpg
2	Bentley Continental GT	D:/CS Project/Car Rental/BentleyContinentalGT.jpg
3	Ferrari 488 GTB	D:/CS Project/Car Rental/Ferrari488GTB.jpg
4	Lamborghini Huracán	D:/CS Project/Car Rental/LamborghiniHuracan.jpg
5	Porsche 911 Turbo S	D:/CS Project/Car Rental/Porsche911TurboS.jpg
6	Rolls-Royce Phantom	D:/CS Project/Car Rental/RollsRoycePhantom.jpg
7	Maserati GranTurismo	D:/CS Project/Car Rental/MaseratiGranTurismo.jpg
8	Mercedes-Benz S-Class	D:/CS Project/Car Rental/MercedesBenzSClass.jpg
9	McLaren 720S	D:/CS Project/Car Rental/McLaren720S.jpg
10	Bugatti Chiron	D:/CS Project/Car Rental/BugattiChiron.jpg
11	Jaguar F-Type	D:/CS Project/Car Rental/JaguarFType.jpg
12	Tesla Model S	D:/CS Project/Car Rental/TeslaModelS.jpg
13	BMW M8	D:/CS Project/Car Rental/BMWM8.jpg
14	Audi R8	D:/CS Project/Car Rental/AudiR8.jpg
15	Lexus LC 500	D:/CS Project/Car Rental/LexusLC500.jpg
NULL	NULL	NULL

PYTHON SOURCE CODE

```python
import tkinter as tk
import os
from tkinter import Frame, Label, PhotoImage, Tk
from tkinter import messagebox
from PIL import Image, ImageTk
import mysql.connector
from datetime import datetime, timedelta
from tkinter import ttk


def db_connect():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="1234",
        database="luxury_cars"
    )


def add_user():
    clear_frames()
    frame_user = tk.Frame(root, bg="#520576", bd=5, highlightbackground='white', highlightthickness=2)
    frame_user.place(relx=0.5, rely=0.2, relwidth=0.5, relheight=0.6, anchor='n')


    label_font = ("Caramel Candy", 14, "bold")
    entry_font = ("Modern Love", 12)


    def generate_user_id():
        try:
            conn = db_connect()
            cursor = conn.cursor()

            cursor.execute("SELECT MAX(user_id) FROM users")
            result = cursor.fetchone()


            if result[0] is None:
                return 1001
            else:
                return result[0] + 1

        except mysql.connector.Error as err:
            messagebox.showerror("Error", "Cannot generate new user id"
                                 "\n Sorry. Please try again")
            return None
        finally:
            conn.close()


    def submit_user():
        name = name_entry.get()
        address = address_entry.get()
        user_mail = contact_entry.get()

        if name and address and user_mail:
            try:
                conn = db_connect()
                cursor = conn.cursor()
```

```
                user_id = generate_user_id()


                if user_id:
                    cursor.execute("INSERT INTO users (user_id, name, address, user_mail) VALUES (%s, %s, %s, %s)",
                                   (user_id, name, address, user_mail))
                    conn.commit()
                    messagebox.showinfo("Success", f"User added successfully with User ID: {user_id}!")
                    frame_user.destroy()
                else:
                    messagebox.showerror("Error", "Failed to generate User ID.")

            except mysql.connector.Error as err:
                messagebox.showerror("Error", "Please enter details correctly")
            finally:
                conn.close()
        else:
            messagebox.showerror("Input Error", "Please provide all required details.")


    tk.Label(frame_user, text="Name:", bg="#520576", fg="white", font=label_font).grid(row=0, column=0, pady=10, sticky='e')
    name_entry = tk.Entry(frame_user, font=entry_font)
    name_entry.grid(row=0, column=1, padx=20, pady=10)

    tk.Label(frame_user, text="Address:", bg="#520576", fg="white", font=label_font).grid(row=1, column=0, pady=10, sticky='e')
    address_entry = tk.Entry(frame_user, font=entry_font)
    address_entry.grid(row=1, column=1, padx=20, pady=10)

    tk.Label(frame_user, text="Email ID:", bg="#520576", fg="white", font=label_font).grid(row=2, column=0, pady=10, sticky='e')
    contact_entry = tk.Entry(frame_user, font=entry_font)
    contact_entry.grid(row=2, column=1, padx=20, pady=10)

    submit_btn = tk.Button(frame_user, text="Submit", bg="#520576", fg="white", font=("Adrenaline Hit Italic", 14), command=submit_user)
    submit_btn.grid(row=3, column=0, columnspan=2, pady=20)

    frame_user.grid_columnconfigure(0, weight=1)
    frame_user.grid_columnconfigure(1, weight=1)

def add_car():
    clear_frames()

    # Frame for adding car details
    frame_car = tk.Frame(root, bg="#520576", bd=5, highlightbackground='white', highlightthickness=2)
    frame_car.place(relx=0.5, rely=0.2, relwidth=0.5, relheight=0.75, anchor='n')

    label_font = ("Caramel Candy", 14, "bold")
    entry_font = ("Modern Love", 12)

    # Function to handle car details submission
    def submit_car():
        make = make_entry.get()
        model = model_entry.get()
        year = year_entry.get()
        color = color_entry.get()
        price = price_entry.get()
        image_path = image_path_entry.get()

        # Check if all necessary details are filled
        if make and model and year and color and price and image_path:
            try:
                # Connect to the database
                conn = db_connect()
                cursor = conn.cursor()

                # Insert the car details into the cars table
                insert_query = """
                    INSERT INTO cars (make, model, year, color, price, image_path)
                    VALUES (%s, %s, %s, %s, %s, %s)
                """
                car_data = (make, model, int(year), color, float(price), image_path)
                cursor.execute(insert_query, car_data)

                # Commit the changes to the database
                conn.commit()

                messagebox.showinfo("Success", "Car added successfully!")
                frame_car.destroy()  # Close the add car frame after success

            except mysql.connector.Error as err:
                messagebox.showerror("Error", f"Please try again: {err}")
            finally:
                conn.close()  # Close the database connection
        else:
            messagebox.showerror("Input Error", "Please provide all required car details.")

    # UI for entering car details
    tk.Label(frame_car, text="Make:", bg="#520576", fg="white", font=label_font).grid(row=0, column=0, pady=10, sticky='e')
    make_entry = tk.Entry(frame_car, font=entry_font)
    make_entry.grid(row=0, column=1, padx=20, pady=10)

    tk.Label(frame_car, text="Model:", bg="#520576", fg="white", font=label_font).grid(row=1, column=0, pady=10, sticky='e')
    model_entry = tk.Entry(frame_car, font=entry_font)
    model_entry.grid(row=1, column=1, padx=20, pady=10)
```

```
    tk.Label(frame_car, text="Year:", bg="#520576", fg="white", font=label_font).grid(row=2, column=0, pady=10, sticky='e')
    year_entry = tk.Entry(frame_car, font=entry_font)
    year_entry.grid(row=2, column=1, padx=20, pady=10)

    tk.Label(frame_car, text="Color:", bg="#520576", fg="white", font=label_font).grid(row=3, column=0, pady=10, sticky='e')
    color_entry = tk.Entry(frame_car, font=entry_font)
    color_entry.grid(row=3, column=1, padx=20, pady=10)

    tk.Label(frame_car, text="Price:", bg="#520576", fg="white", font=label_font).grid(row=4, column=0, pady=10, sticky='e')
    price_entry = tk.Entry(frame_car, font=entry_font)
    price_entry.grid(row=4, column=1, padx=20, pady=10)

    tk.Label(frame_car, text="Image Path:", bg="#520576", fg="white", font=label_font).grid(row=5, column=0, pady=10, sticky='e')
    image_path_entry = tk.Entry(frame_car, font=entry_font)
    image_path_entry.grid(row=5, column=1, padx=20, pady=10)

    # Submit button
    submit_btn = tk.Button(frame_car, text="Submit", bg="#520576", fg="white", font=("Adrenaline Hit Italic", 14), command=submit_car)
    submit_btn.grid(row=6, column=0, columnspan=2, pady=20)

    # Configure column weights for better layout
    frame_car.grid_columnconfigure(0, weight=1)
    frame_car.grid_columnconfigure(1, weight=1)


def cust_profiles():
    clear_frames()

    # Create the frame to hold the treeview table
    frame_view_users = tk.Frame(root, bg="#520576", bd=5, highlightbackground='#A67B5B', highlightthickness=2)
    frame_view_users.place(relx=0.5, rely=0.2, relwidth=0.75, relheight=0.6, anchor='n')
    # Add columns for the treeview, including car rental details
    columns = ("User ID", "Name", "Address", "Email ID", "Car ID", "Borrow Date", "Return Date")
    tree = ttk.Treeview(frame_view_users, columns=columns, show="headings", height=15)

    # Define headings for each column
    tree.heading("User ID", text="User ID")
    tree.heading("Name", text="Name")
    tree.heading("Address", text="Address")
    tree.heading("Email ID", text="Email ID")
    tree.heading("Car ID", text="Car ID")
    tree.heading("Borrow Date", text="Borrow Date")
    tree.heading("Return Date", text="Return Date")

    # Define column widths
    tree.column("User ID", width=100, anchor='center')
    tree.column("Name", width=150, anchor='w')
    tree.column("Address", width=200, anchor='w')
    tree.column("Email ID", width=150, anchor='w')
    tree.column("Car ID", width=100, anchor='center')
    tree.column("Borrow Date", width=120, anchor='center')
    tree.column("Return Date", width=120, anchor='center')

    # Add the treeview to the frame
    tree.pack(fill=tk.BOTH, expand=True)

    # Add a scrollbar to the treeview
    scrollbar = ttk.Scrollbar(frame_view_users, orient=tk.VERTICAL, command=tree.yview)
    tree.configure(yscroll=scrollbar.set)
    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
    try:
        # Connect to the database
        conn = db_connect()
        cursor = conn.cursor()

        # SQL query to join 'users' and 'borrows' to fetch necessary details
        cursor.execute("""
            SELECT u.user_id, u.name, u.address, u.user_mail,
                   b.car_id, b.borrow_date, b.return_date
            FROM users u
            LEFT JOIN borrows b ON u.user_id = b.user_id
        """)
        records = cursor.fetchall()

        if records:
            for record in records:
                # Insert the record into the treeview
                tree.insert('', tk.END, values=record)
        else:
            # If no users or borrow records are found, display a message
            tk.Label(frame_view_users, text="No users found or no rentals.", font=("Adrenaline Hit Italic", 12), bg="#520576", fg="#A67B5B").pack(pady=10)

    except mysql.connector.Error as err:
        # Display an error message if something goes wrong
        messagebox.showerror("Error", "Please try again: " + str(err))
    finally:
        # Ensure the connection is closed
        conn.close()
```

```
def rent_car():
    clear_frames()  # Clear any existing frames before showing a new one
    frame_borrow = tk.Frame(root, bg="#520576", bd=5, highlightbackground='white', highlightthickness=2)
    frame_borrow.place(relx=0.5, rely=0.2, relwidth=0.375, relheight=0.6, anchor='n')

    label_font = ("Caramel Candy", 14, "bold")
    entry_font = ("Modern Love", 12)

    def update_car_id(event):
        selected_car = car_dropdown.get()
        car_id = car_id_map.get(selected_car, "")
        car_id_entry.delete(0, tk.END)
        car_id_entry.insert(0, car_id)

    def submit_borrow():
        user_id = user_id_entry.get()
        car_id = car_id_entry.get()

        if user_id and car_id:
            try:
                conn = db_connect()
                cursor = conn.cursor()

                # Check if the car is available
                cursor.execute("SELECT * FROM cars WHERE id = %s AND rented_by IS NULL", (car_id,))
                car = cursor.fetchone()

                if car:
                    borrow_date = datetime.now().date()
                    return_date = borrow_date + timedelta(days=14)
                    cursor.execute(
                        "INSERT INTO borrows (user_id, car_id, borrow_date, return_date) VALUES (%s, %s, %s, NULL)",
                        (user_id, car_id, borrow_date)
                    )

                    # Update the car's rented status
                    cursor.execute("UPDATE cars SET rented_by = %s WHERE id = %s", (user_id, car_id))
                    conn.commit()

                    messagebox.showinfo("Success", "Car rented successfully!")
                    frame_borrow.destroy()
                else:
                    messagebox.showerror("Error", "Car is not available for borrowing.")

            except mysql.connector.Error as err:
                messagebox.showerror("Error", "Please try again")
            finally:
                conn.close()
        else:
            messagebox.showerror("Input Error", "Please provide all required details.")

    try:
        conn = db_connect()
        cursor = conn.cursor()
        cursor.execute("SELECT id, make, model FROM cars WHERE rented_by IS NULL")
        cars = cursor.fetchall()

        car_id_map = {f"{make} {model} (ID: {car_id})": car_id for car_id, make, model in cars}

    except mysql.connector.Error as err:
        messagebox.showerror("Error", str(err))
        car_id_map = {}
    finally:
        conn.close()
    tk.Label(frame_borrow, text="User ID:", bg="#520576", fg="white", font=label_font).grid(row=0, column=0, pady=10, sticky='e')
    user_id_entry = tk.Entry(frame_borrow, font=entry_font)
    user_id_entry.grid(row=0, column=1, padx=20, pady=10)

    tk.Label(frame_borrow, text="Select Car:", bg="#520576", fg="white", font=label_font).grid(row=1, column=0, pady=10, sticky='e')
    car_dropdown = ttk.Combobox(frame_borrow, values=list(car_id_map.keys()), font=entry_font)
    car_dropdown.grid(row=1, column=1, padx=20, pady=10)
    car_dropdown.bind("<<ComboboxSelected>>", update_car_id)

    tk.Label(frame_borrow, text="Car ID:", bg="#520576", fg="white", font=label_font).grid(row=2, column=0, pady=10, sticky='e')
    car_id_entry = tk.Entry(frame_borrow, font=entry_font)
    car_id_entry.grid(row=2, column=1, padx=20, pady=10)

    submit_btn = tk.Button(frame_borrow, text="Submit", bg="#520576", fg="white", font=("Adrenaline Hit Italic", 14), command=submit_borrow)
    submit_btn.grid(row=3, column=0, columnspan=2, pady=20)

    frame_borrow.grid_columnconfigure(0, weight=1)
    frame_borrow.grid_columnconfigure(1, weight=1)

def return_car():
    clear_frames()
    frame_return = tk.Frame(root, bg="#520576", bd=5, highlightbackground='white', highlightthickness=2)
    frame_return.place(relx=0.5, rely=0.2, relwidth=0.375, relheight=0.6, anchor='n')

    label_font = ("Caramel Candy", 14, "bold")
    entry_font = ("Modern Love", 12)

    def update_car_id(event):
        selected_car = car_dropdown.get()
        car_id = car_id_map_return.get(selected_car, "")
        car_id_entry.delete(0, tk.END)
        car_id_entry.insert(0, car_id)
```

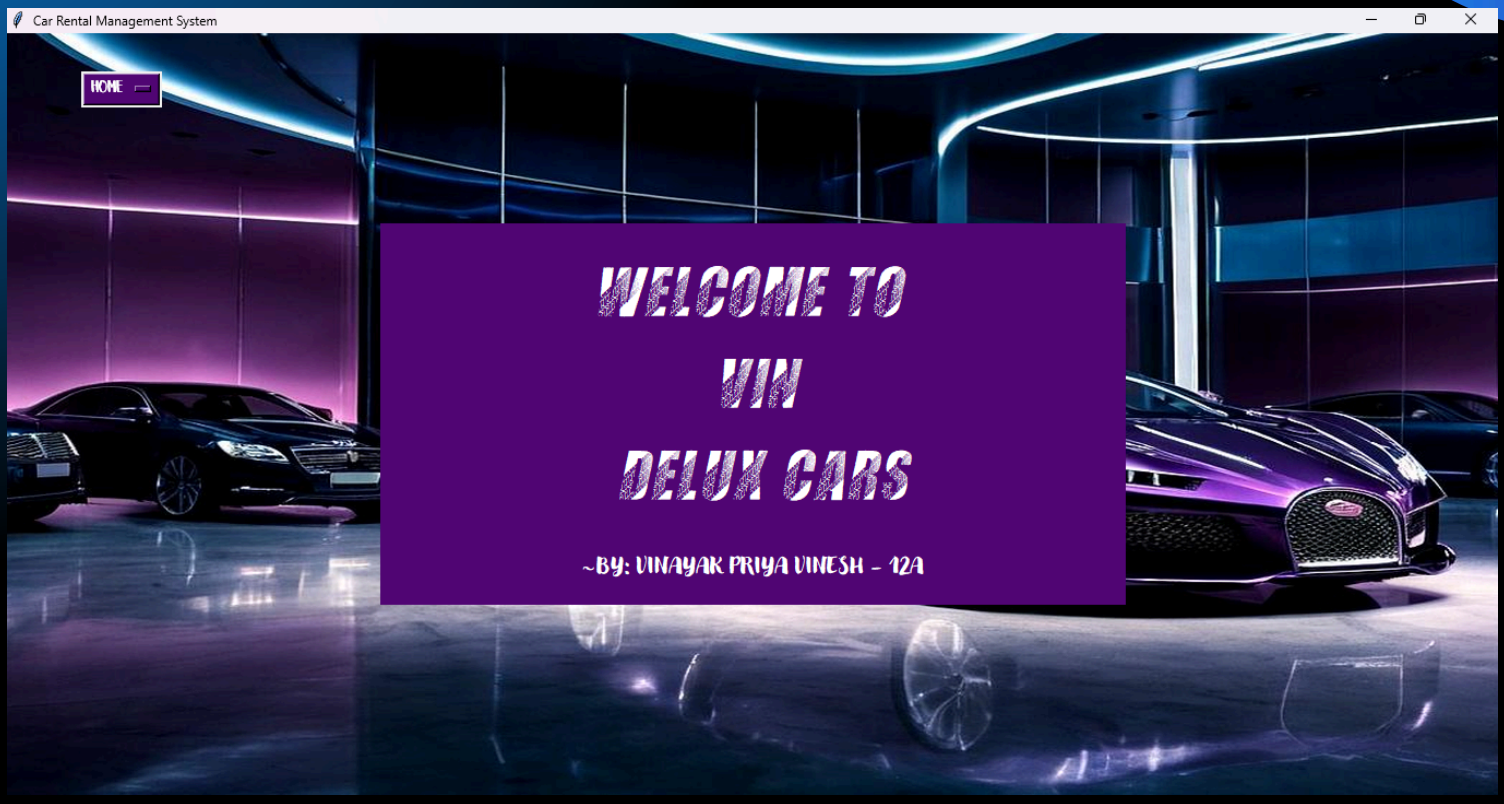
```
    def submit_return():
        user_id = user_id_entry.get()
        car_id = car_id_entry.get()

        if user_id and car_id:
            try:
                conn = db_connect()
                cursor = conn.cursor()

                # Check if the borrow record exists
                cursor.execute(
                    "SELECT borrow_date, return_date FROM borrows WHERE user_id = %s AND car_id = %s",
                    (user_id, car_id)
                )
                borrow_record = cursor.fetchone()

                if borrow_record:
                    borrow_date, return_date = borrow_record

                    # Update return date to current date
                    new_return_date = datetime.now().date()
                    cursor.execute(
                        "UPDATE borrows SET return_date = %s WHERE user_id = %s AND car_id = %s",
                        (new_return_date, user_id, car_id)
                    )

                    # Mark car as available by setting rented_by to NULL
                    cursor.execute("UPDATE cars SET rented_by = NULL WHERE id = %s", (car_id,))
                    conn.commit()

                    messagebox.showinfo("Success", "Car returned successfully!")
                    frame_return.destroy()
                else:
                    messagebox.showerror("Error", "No record of this car being rented by this user.")
                    print("No borrow record found for User ID:", user_id, "Car ID:", car_id)  # Debugging print

            except mysql.connector.Error as err:
                messagebox.showerror("Error", f"Database error: {err}")  # Show detailed database error
                print("Database error:", err)  # Print for debugging

            except Exception as e:
                messagebox.showerror("Error", f"An unexpected error occurred: {e}")
                print("Unexpected error:", e)  # Print any unexpected error for debugging

            finally:
                if conn.is_connected():
                    conn.close()

        else:
            messagebox.showerror("Input Error", "Please provide all required details.")

    try:
        conn = db_connect()
        cursor = conn.cursor()
        cursor.execute("SELECT id, make, model FROM cars WHERE rented_by IS NOT NULL")
        cars = cursor.fetchall()

        car_id_map_return = {f"{make} {model} (ID: {car_id})": car_id for car_id, make, model in cars}

        tk.Label(frame_return, text="Select Car:", bg="#520576", fg="white", font=label_font).grid(row=1, column=0, pady=10, sticky='e')
        car_dropdown = ttk.Combobox(frame_return, values=list(car_id_map_return.keys()), font=entry_font)
        car_dropdown.grid(row=1, column=1, padx=20, pady=10)
        car_dropdown.bind("<<ComboboxSelected>>", update_car_id)
    submit_btn = tk.Button(frame_return, text="Submit", bg="#520576", fg="white", font=("Adrenaline Hit Italic", 14), command=submit_return)
    submit_btn.grid(row=3, column=0, columnspan=2, pady=20)

    frame_return.grid_columnconfigure(0, weight=1)
    frame_return.grid_columnconfigure(1, weight=1)



def update_dropdown_cars(car_dropdown=None):
    try:
        conn = db_connect()
        cursor = conn.cursor()
        cursor.execute("SELECT id, make, model FROM cars WHERE rented_by IS NOT NULL")
        cars = cursor.fetchall()

        global car_id_map_return
        car_id_map_return = {f"{make} {model} (ID: {car_id})": car_id for car_id, make, model in cars}

        car_dropdown['values'] = list(car_id_map_return.keys())  # Update dropdown values
    except mysql.connector.Error as err:
        messagebox.showerror("Error", str(err))
    finally:
        conn.close()

def show_available_cars():
    clear_frames()

    # Create a main frame to hold the canvas and scrollbar
    main_frame = tk.Frame(root, bg="#520576", bd=5, highlightbackground='#A67B5B', highlightthickness=2)
    main_frame.place(relx=0.5, rely=0.2, relwidth=0.75, relheight=0.6, anchor='n')
```

```
    # Create a canvas inside the main frame
    canvas = tk.Canvas(main_frame, bg="#520576")
    canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

    # Add a scrollbar to the canvas
    scrollbar = tk.Scrollbar(main_frame, orient=tk.VERTICAL, command=canvas.yview)
    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
    canvas.configure(yscrollcommand=scrollbar.set)

    # Create a frame within the canvas to hold car images, names, and prices
    frame_available_cars = tk.Frame(canvas, bg="#520576")
    canvas.create_window((0, 0), window=frame_available_cars, anchor="nw")

    # Populate frame with car images, names, and prices
    try:
        conn = db_connect()
        cursor = conn.cursor()

        # Fetch the car details (make, model, year, color, price, image_path)
        cursor.execute("SELECT make, model, year, color, price, image_path FROM cars")
        cars = cursor.fetchall()

        # Ensure that no variables are None before attempting to format
        if cars:
            for i, (make, model, year, color, price, image_path) in enumerate(cars):
                # If price is None, set it to a default value
                if price is None:
                    price = 0.0

                # Load and resize the car image
                try:
                    car_image = Image.open(image_path)
                    car_image = car_image.resize((320, 180), Image.Resampling.LANCZOS)
                    car_photo = ImageTk.PhotoImage(car_image)

                    # Create a label for the car image
                    car_image_label = tk.Label(frame_available_cars, image=car_photo, bg="#520576")
                    car_image_label.image = car_photo  # Keep a reference to avoid garbage collection
                    car_image_label.grid(row=i, column=0, padx=10, pady=10)

                except Exception as e:
                    # Handle missing or corrupt image files
                    error_label = tk.Label(frame_available_cars, text="Image not found", font=("Arial", 12), fg="red", bg="#520576")
                    error_label.grid(row=i, column=0, padx=10, pady=10)

                # Format and create a label for the car info (Make, Model, Year, Color, and Price)
                car_info_label = tk.Label(
                    frame_available_cars,
                    text=f"{make} {model}\nYear: {year}\nColor: {color}\nPrice: ${price:,.2f}",
                    bg="#520576",
                    fg="white",
                    font=("Caramel", 18)
                )
                car_info_label.grid(row=i, column=1, padx=10, pady=10, sticky="w")

        else:
            tk.Label(frame_available_cars, text="No cars available.", font=("Adrenaline Hit Italic", 12), bg="#520576", fg="#A67B5B").pack(pady=10)


    except mysql.connector.Error as err:
        messagebox.showerror("Error", str(err))
    finally:
        if conn.is_connected():
            conn.close()

    # Configure scrolling region
    frame_available_cars.update_idletasks()  # Update the frame size
    canvas.config(scrollregion=canvas.bbox("all"))


def on_select(option):
    clear_frames()

    # Check which option was selected and call the corresponding function
    if option == "HOME":
        show_home()
    elif option == "ADD USER":
        add_user()
    elif option == "ADD CAR":
        add_car()
    elif option == "VIEW USER DETAILS":
        cust_profiles()
    elif option == "RENT CAR":
        rent_car()
    elif option == "RETURN CAR":
        return_car()
    elif option == "AVAILABLE CARS":
        show_available_cars()


def clear_frames():
    for widget in root.winfo_children():
        if isinstance(widget, tk.Frame):
            widget.destroy()
```

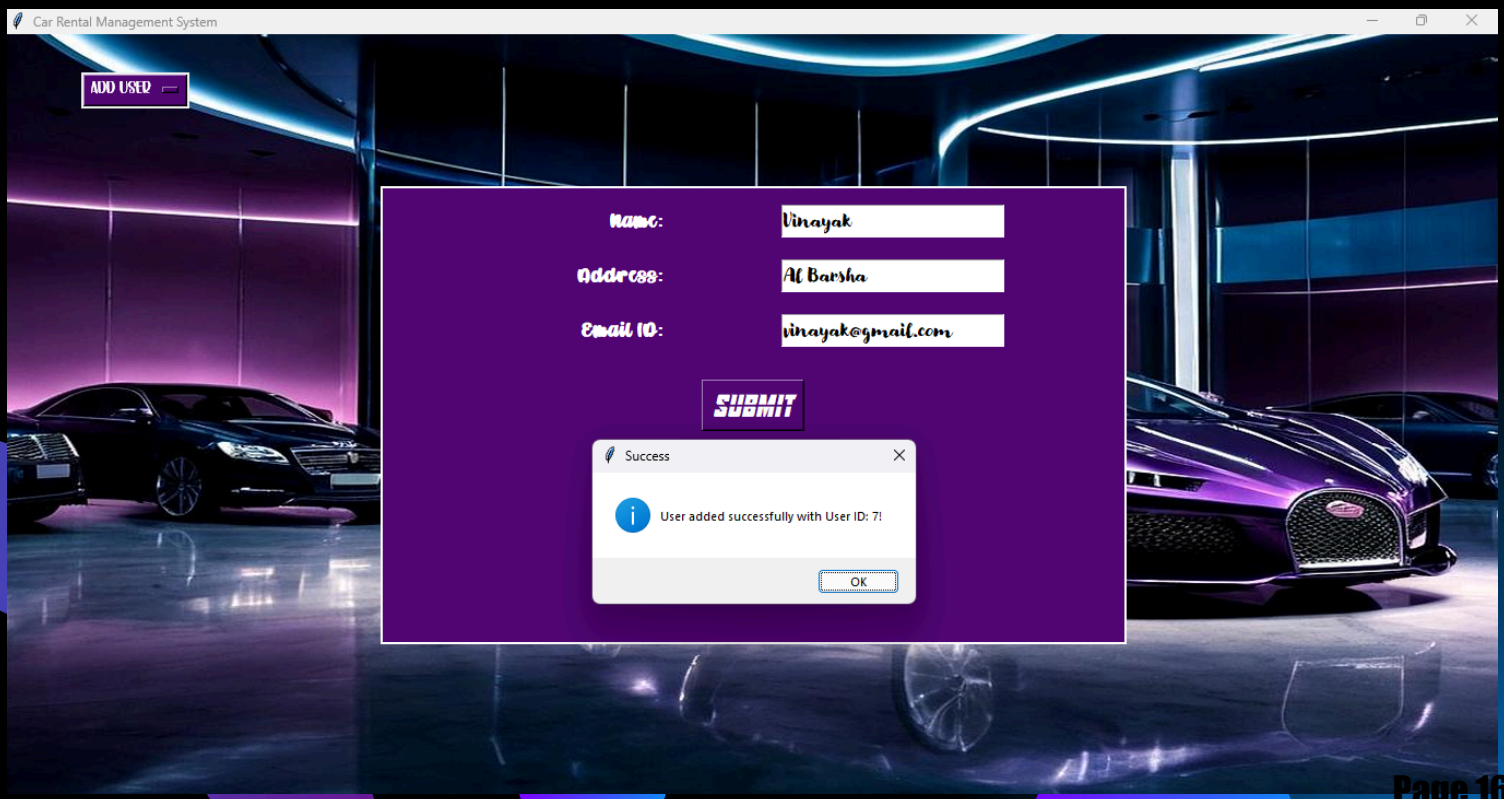
```python
def show_home():
    frame_home = tk.Frame(root, bg="#520576", bd=5)
    frame_home.place(relx=0.5, rely=0.25, relwidth=0.5, relheight=0.5, anchor='n')

    welcome_label = tk.Label(frame_home, text="Welcome to \n Vin \n Delux Cars", font=("TIRES ITALIC PERSONAL USE Bold Italic", 50), bg="#520576", fg='white')
    welcome_label.pack(pady=10)
    author_label = tk.Label(frame_home, text="~BY: VINAYAK PRIYA VINESH - 12A", font=("Modern Love", 14), bg="#520576", fg='white')
    author_label.pack(side=tk.BOTTOM, pady=20)
    frame_home.lift()


root = tk.Tk()
root.title("Car Rental Management System")
root.geometry("800x600")

try:
    bg_image = Image.open(r"D:\CS Project\Car Rental\luxury_cars2_enhanced.jpg")  # Use a suitable background image for the car rental system
    bg_image = bg_image.resize((1800, 800), Image.Resampling.LANCZOS)
    bg_photo = ImageTk.PhotoImage(bg_image)

    bg_label = tk.Label(root, image=bg_photo)
    bg_label.place(relwidth=1, relheight=1)

    overlay = tk.Label(root, bg='#ff5d5c')
    overlay.place(relwidth=1, relheight=1)
    overlay.lower(bg_label)
except Exception as e:
    print(f"Error loading image: {e}")

options = ["HOME", "ADD USER", "ADD CAR", "VIEW USER DETAILS", "RENT CAR", "RETURN CAR", "AVAILABLE CARS"]
option_option = tk.StringVar()
option_option.set(options[0])

dropdown = tk.OptionMenu(root, option_option, *options)
dropdown.config(bg='#520576', fg='white', font=('Caramel', 12))
dropdown.place(relx=0.05, rely=0.05, anchor='nw')

option_option.trace("w", lambda *args: on_select(option_option.get()))

show_home()  # Show the home screen initially

root.mainloop()
```

CODE OUTPUT:

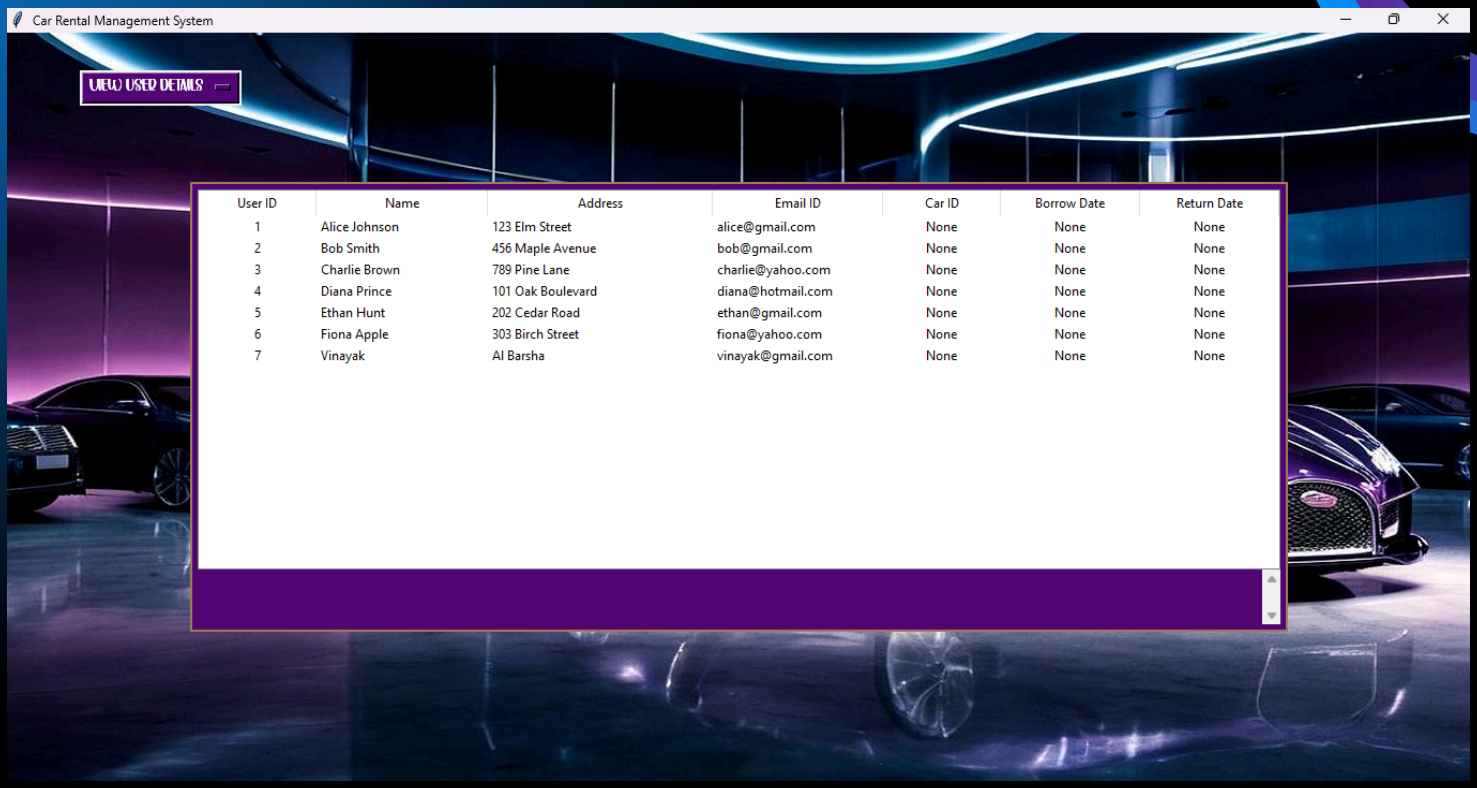
I. HOME PAGE



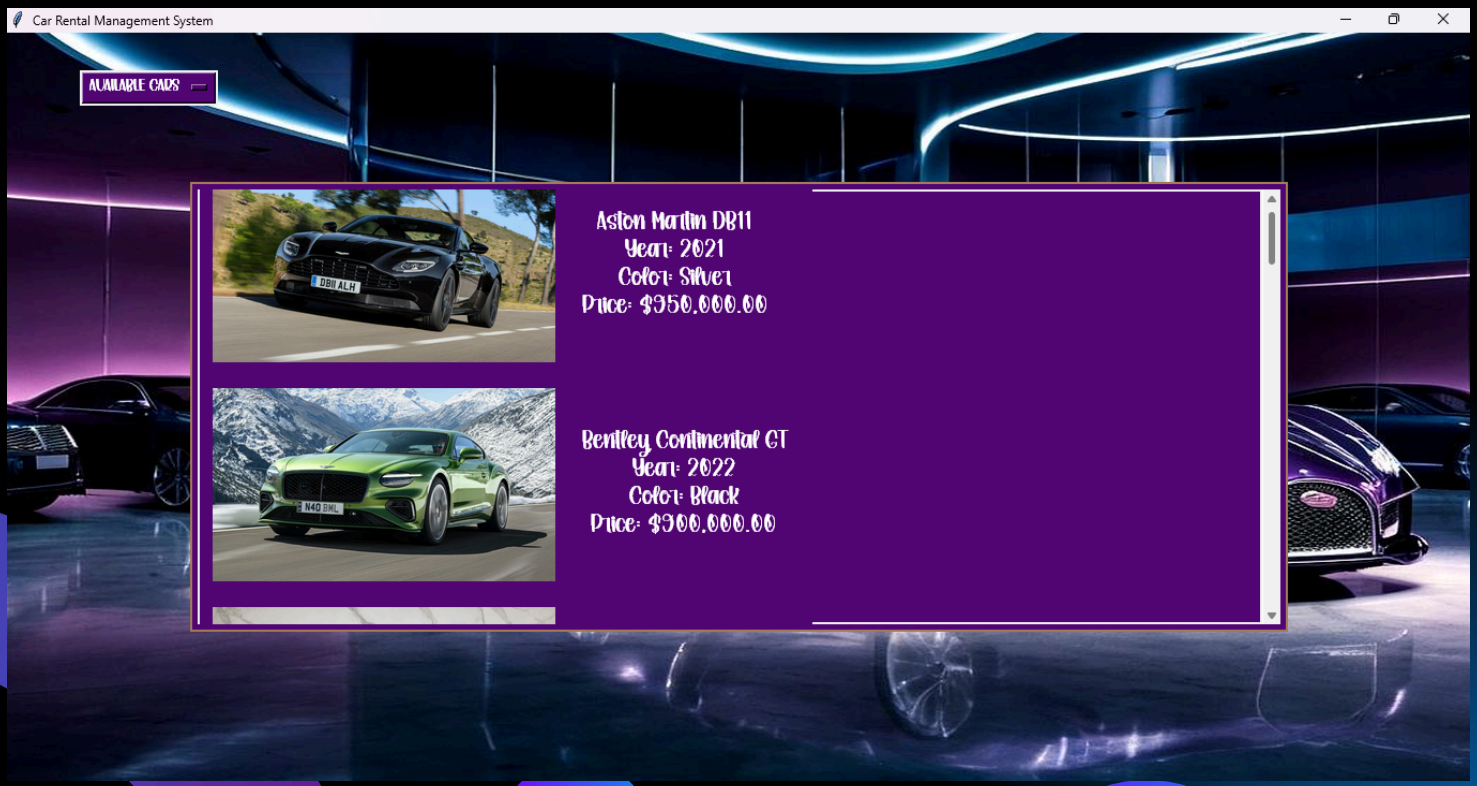
2. ADDING USER

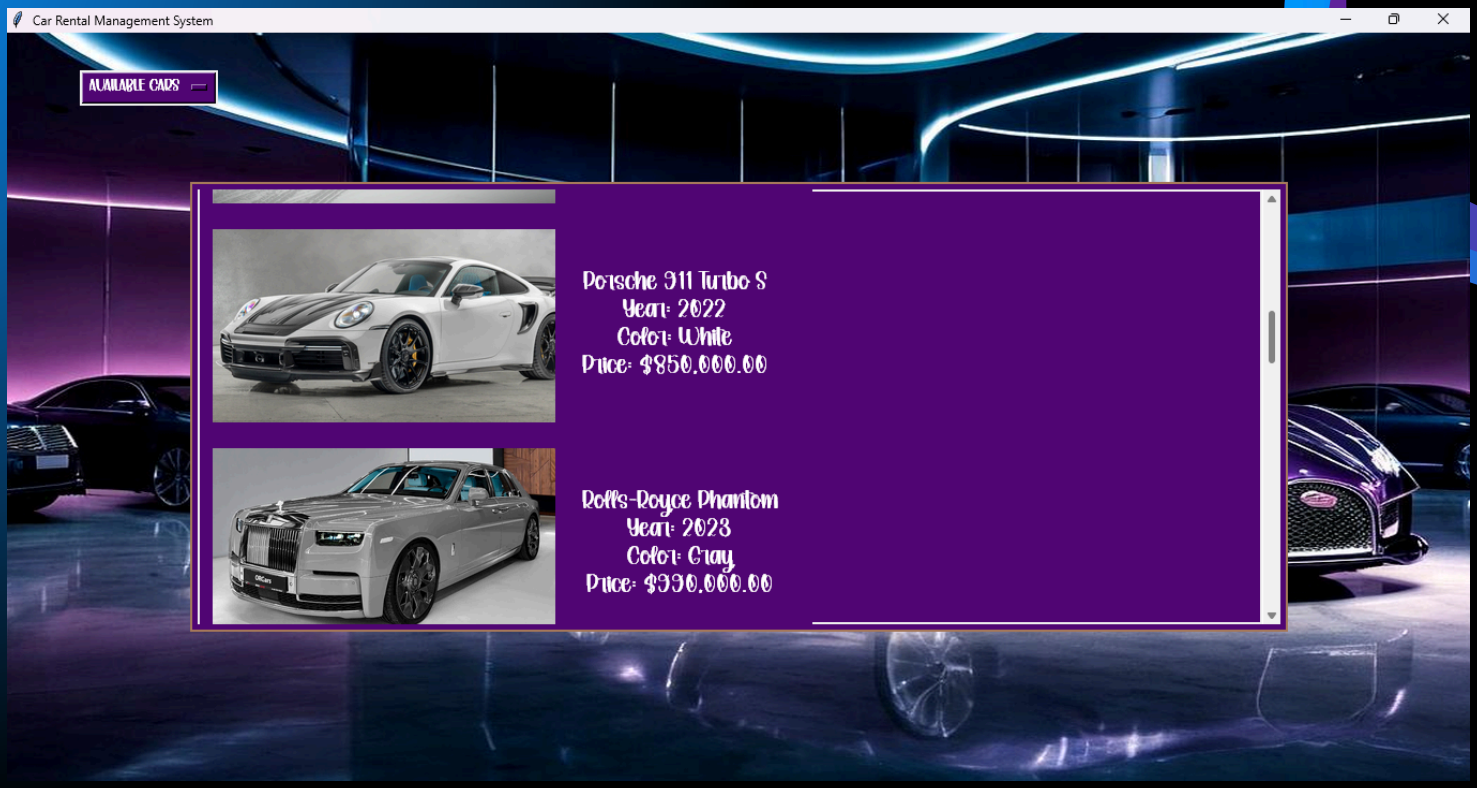


3. VIEWING USER DETAILS

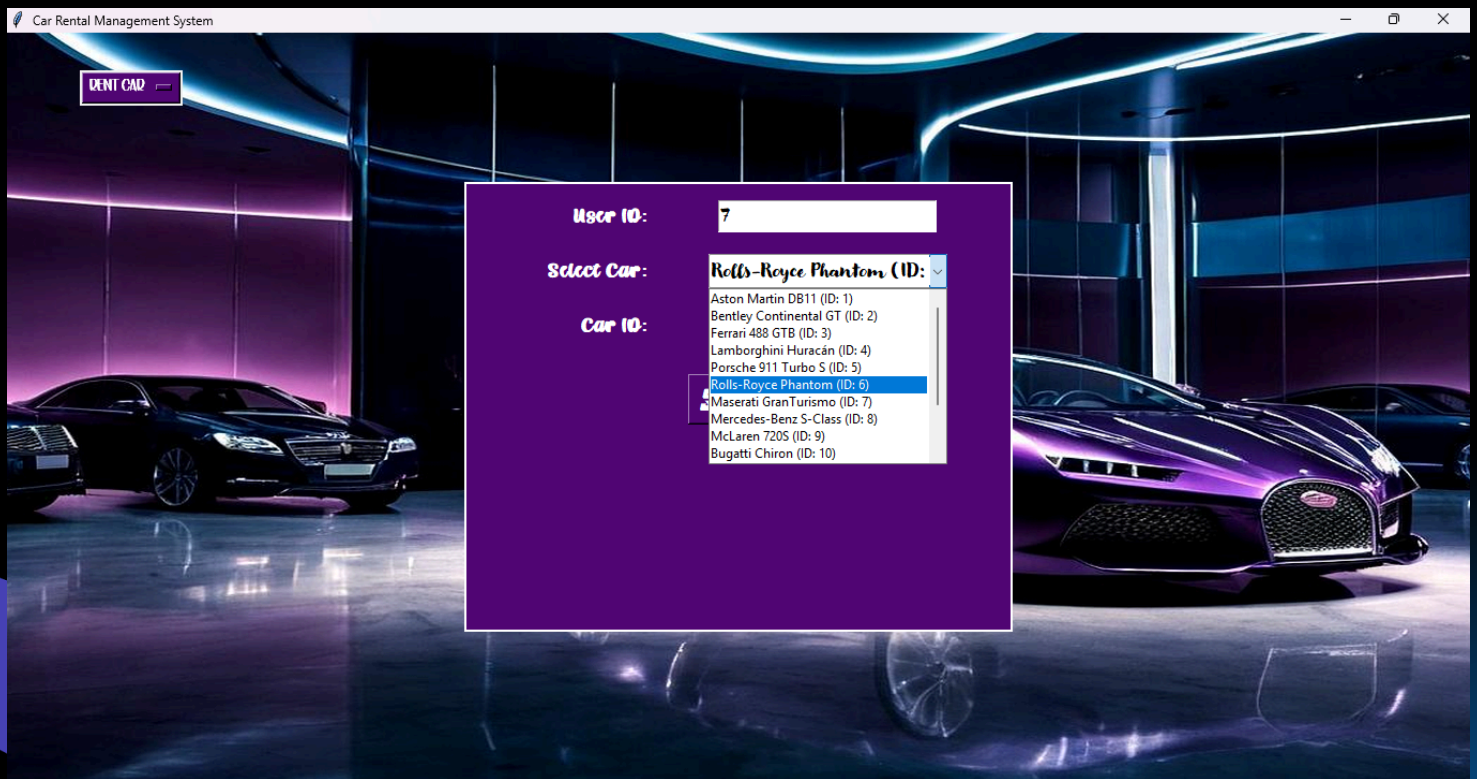


4. CHECKING AVAILABLE CARS

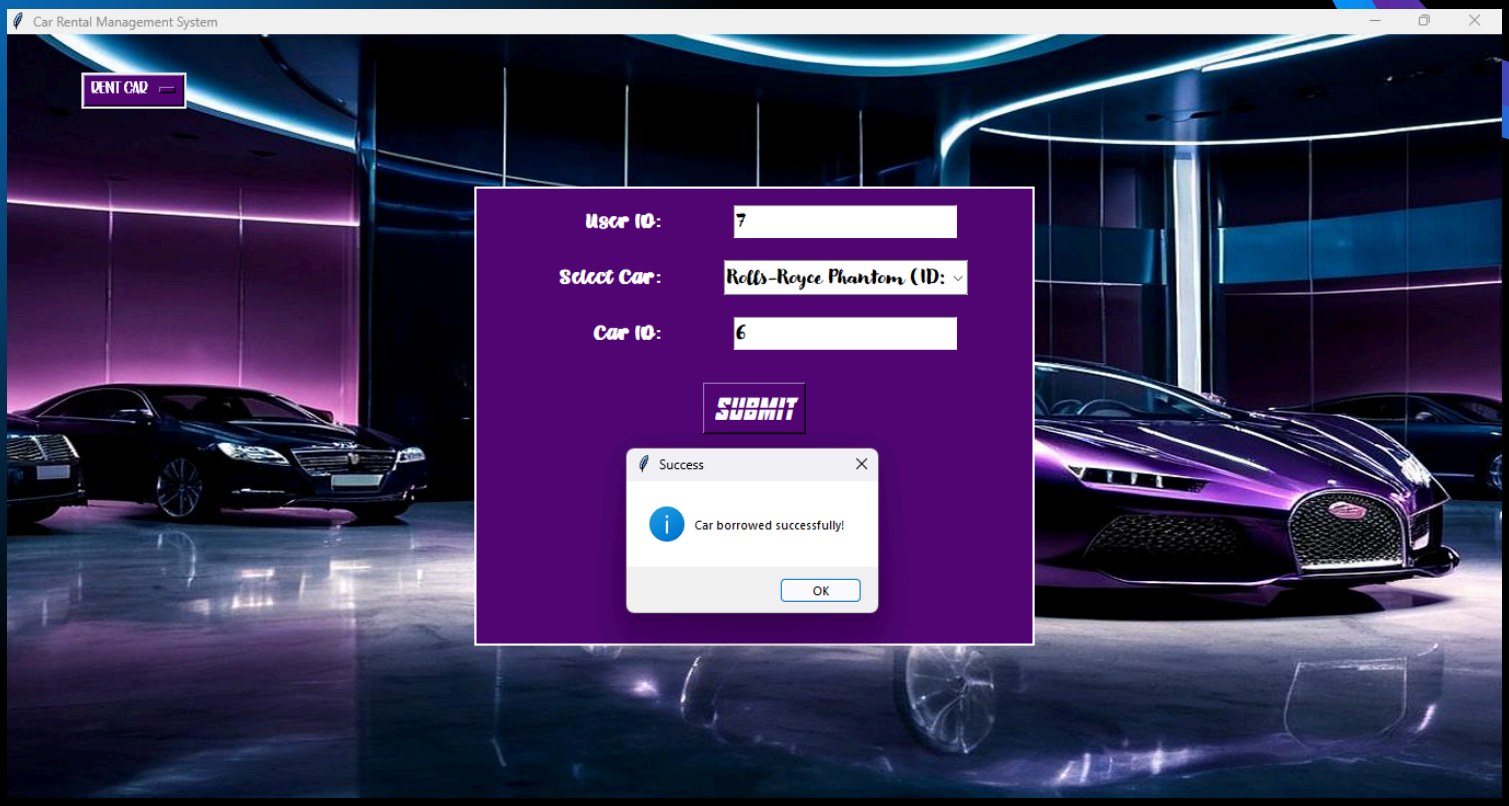


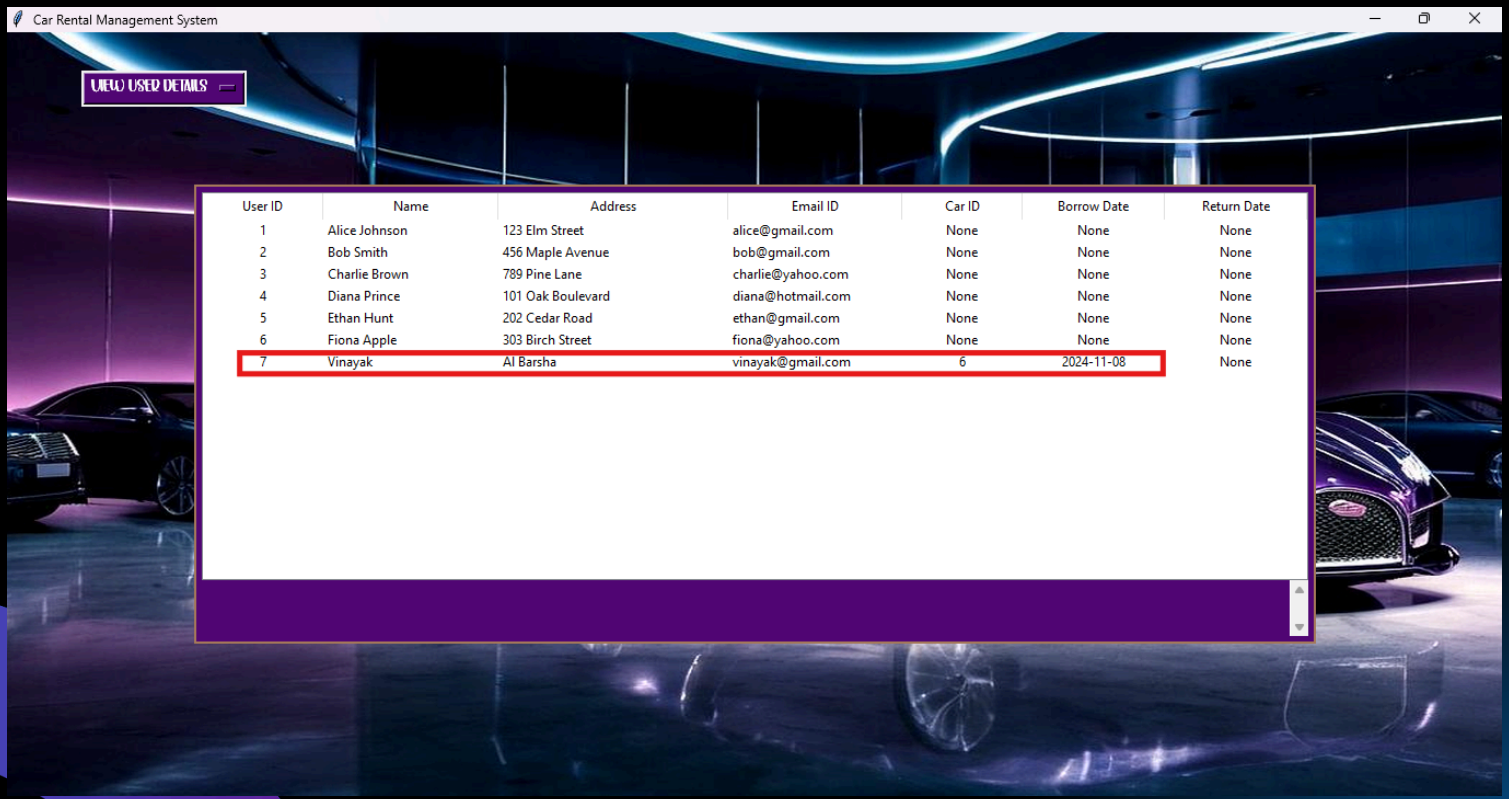


5. RENTING A CAR

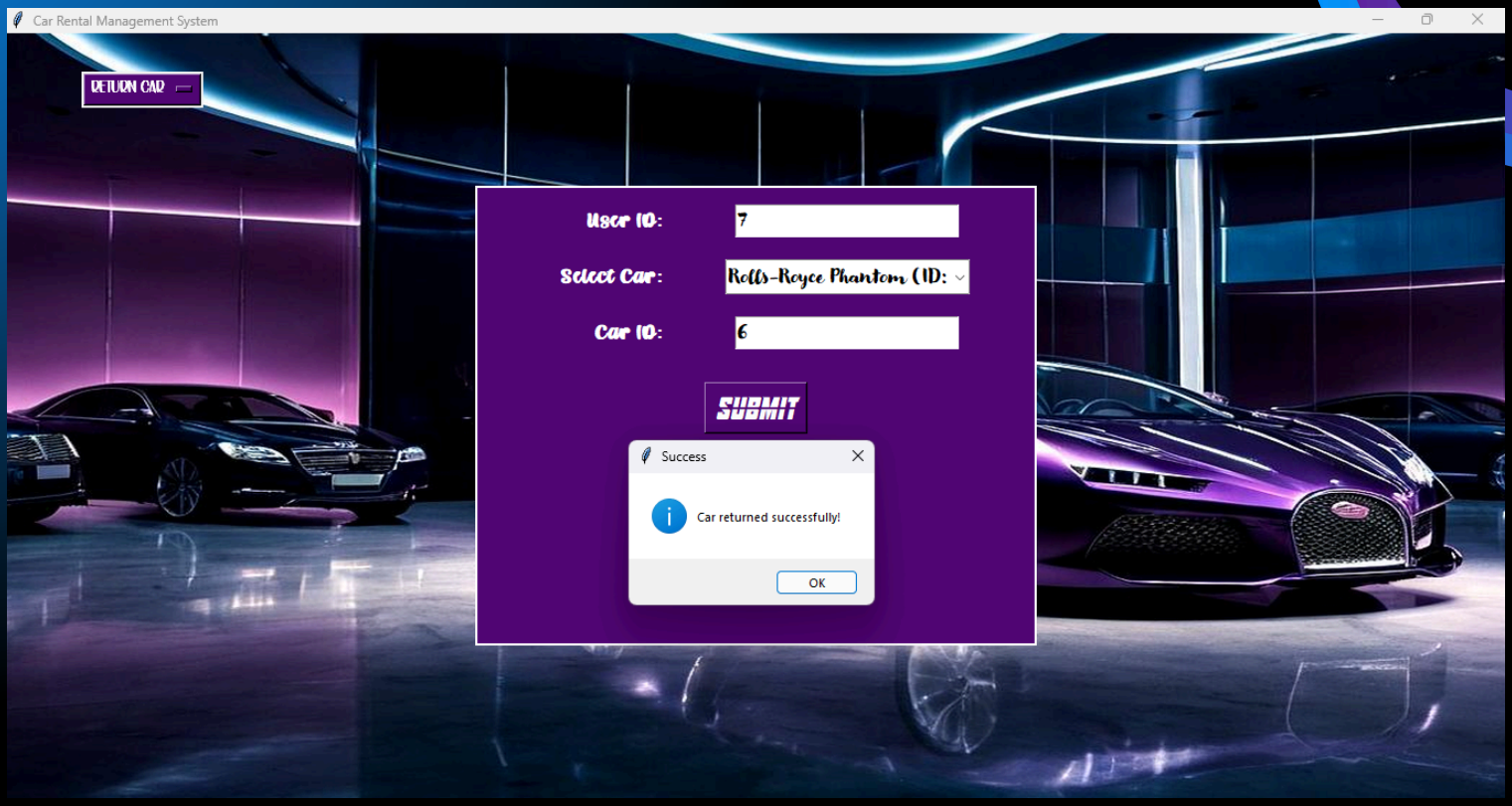


-->SHOWING RESULT:

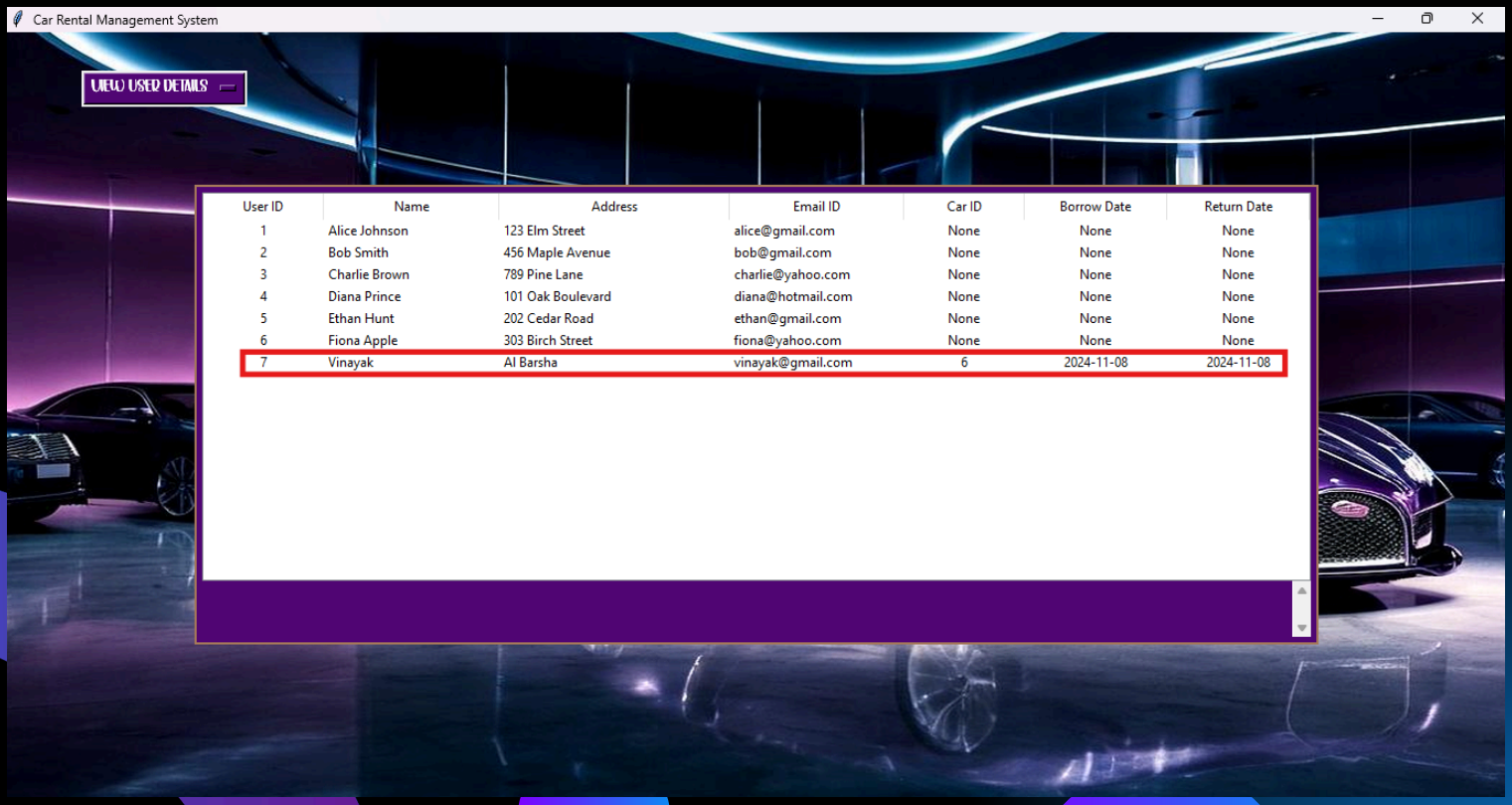




6. RETURNING THE CAR:

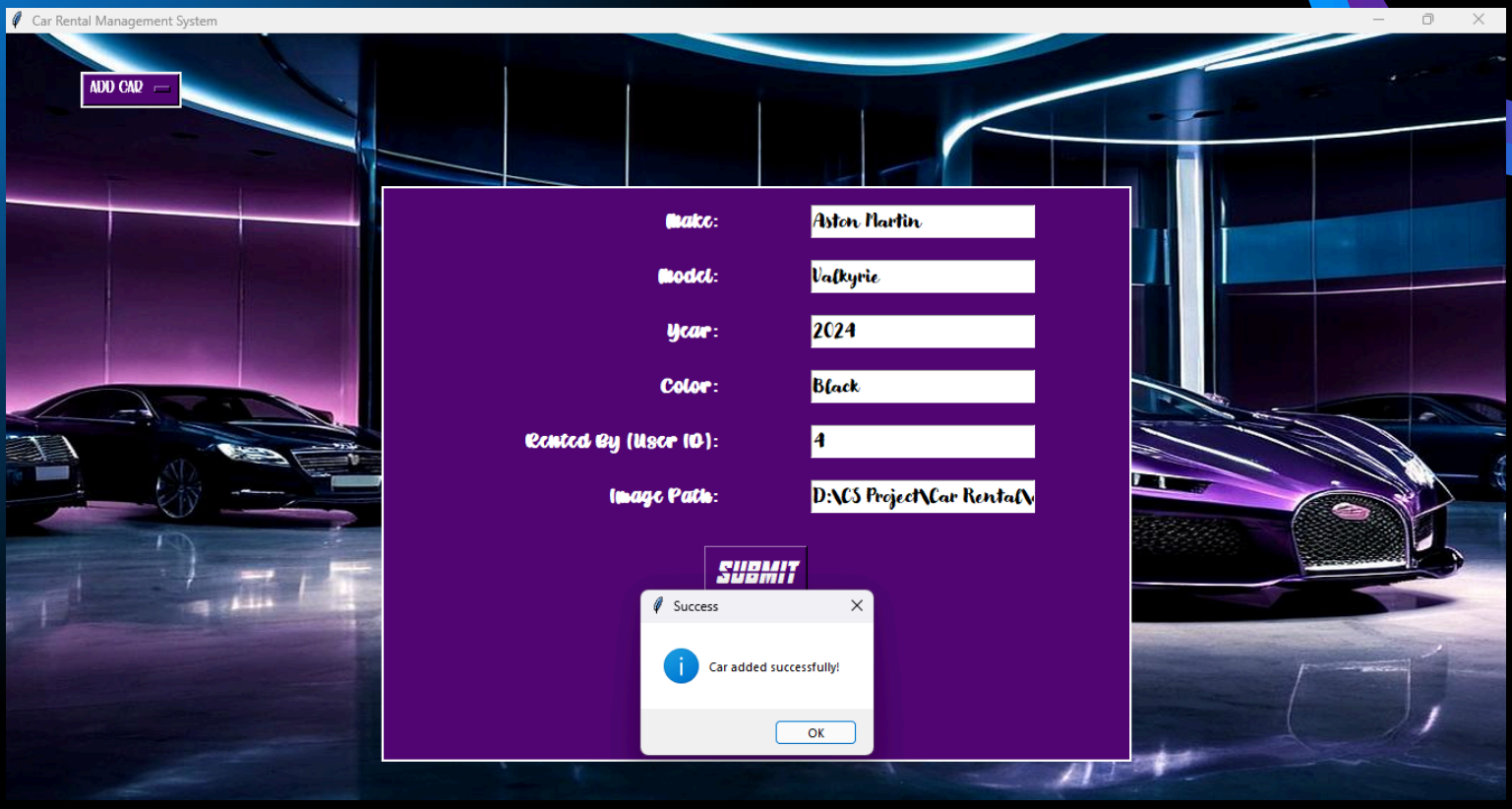


-->SHOWING RESULT IN VIEW USERS:

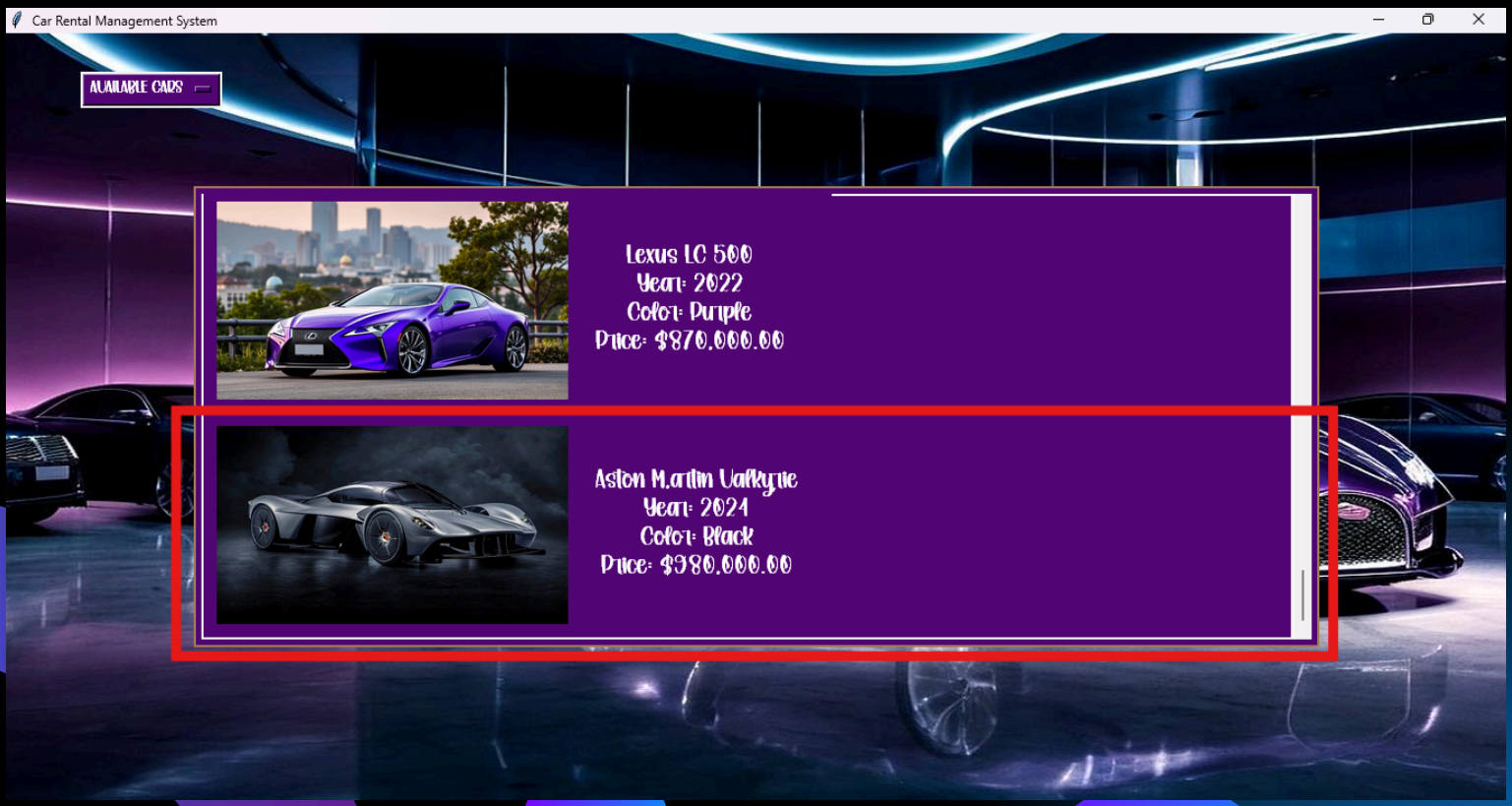


User ID	Name	Address	Email ID	Car ID	Borrow Date	Return Date
1	Alice Johnson	123 Elm Street	alice@gmail.com	None	None	None
2	Bob Smith	456 Maple Avenue	bob@gmail.com	None	None	None
3	Charlie Brown	789 Pine Lane	charlie@yahoo.com	None	None	None
4	Diana Prince	101 Oak Boulevard	diana@hotmail.com	None	None	None
5	Ethan Hunt	202 Cedar Road	ethan@gmail.com	None	None	None
6	Fiona Apple	303 Birch Street	fiona@yahoo.com	None	None	None
7	Vinayak	Al Barsha	vinayak@gmail.com	6	2024-11-08	2024-11-08

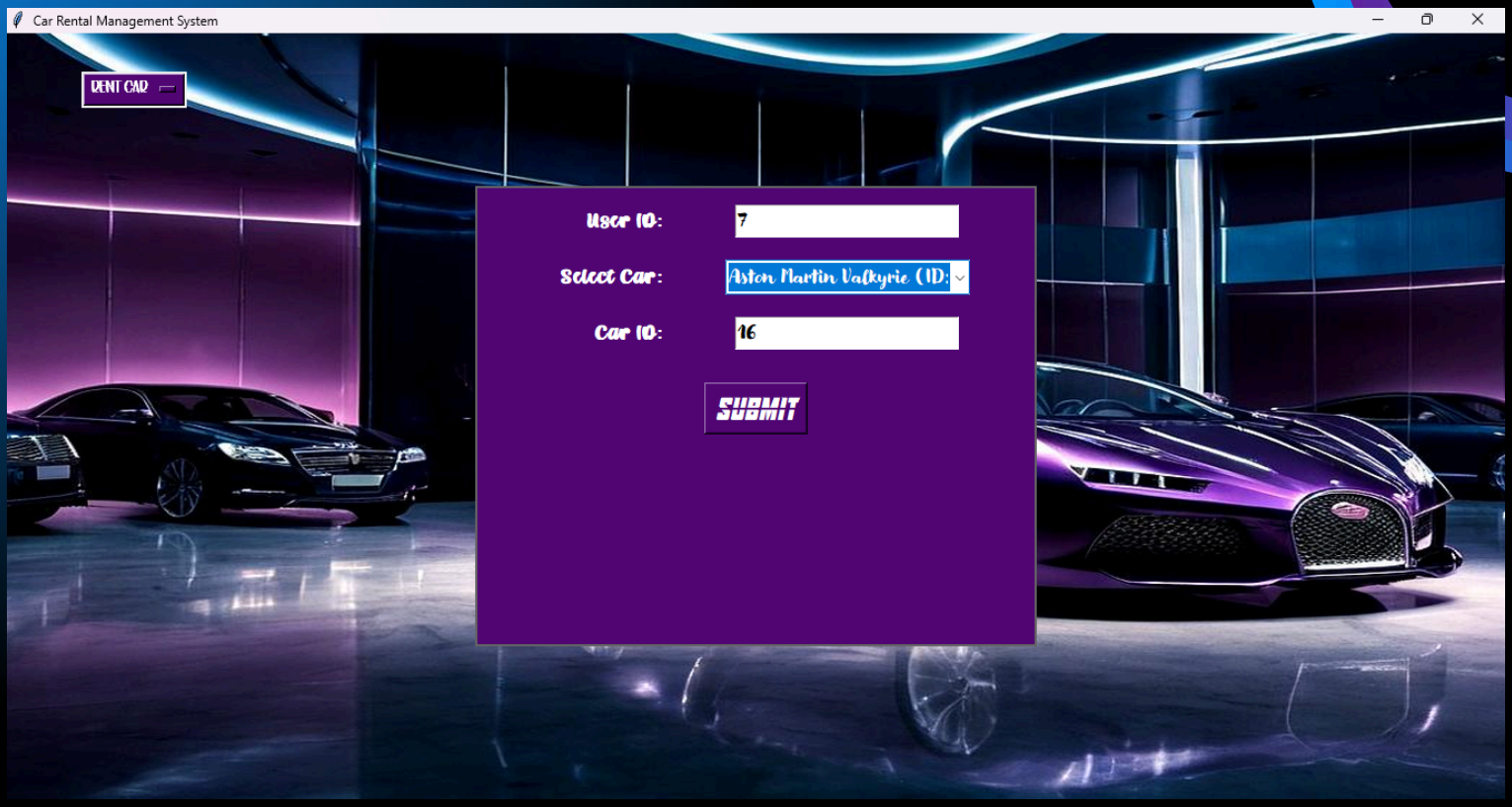
7. ADDING A CAR:



--> RESULT IN AVAILABLE CARS:



--> RESULTS IN RENT CAR:



FUTURE SCOPE:

- **Online Reservation System:** Allow customers to reserve cars online before visiting the showroom. This feature could streamline the renting process and give users a convenient booking option.
- **Automated Payment Processing:** Integrate a payment gateway to handle payments directly through the system. This feature would make transactions easier and reduce manual processing time.
- **Enhanced Security Features:** Add security measures like user authentication, encryption for sensitive data, and audit trails to make the system more secure for both the showroom and the customers.
- **Augmented Reality (AR) Car Customizer:** Customers could use AR to visualize and customize car colors, interiors, or add-ons right on their phones. They could "place" the car in their driveway or garage to see how it looks in real life, making the selection process highly interactive and personal.
- **AI-Powered Car Matchmaker:** Integrate an AI recommendation engine that suggests cars based on customer preferences, lifestyle, and budget. Think of it as a "dating app" for finding the perfect luxury car match!
- **Virtual Staff Service:** Add a virtual staff that provides real-time assistance for any queries or car-related information. This could include guided virtual tours of each car, explaining unique features, and answering questions.

- **Exclusive Club Memberships for Car Enthusiasts:** Introduce exclusive membership tiers offering benefits like early access to new models, personalized offers, or private events. Each level could come with its own perks, giving members the VIP experience they crave.
- **Eco-Footprint Dashboard for Each Car:** Display the eco-impact of each car in terms of emissions, fuel efficiency, and other metrics, helping environmentally-conscious customers make informed choices. For electric models, it could showcase estimated charging locations and energy savings.
- **Personalized Luxury Enhancements:** Allow users to view custom luxury upgrades such as personalized interiors, limited-edition rims, or bespoke paint finishes that they can "try on" for each car. This could offer a more exclusive feel and align with the luxury experience.
- **"Test Drive" Simulation Experience:** Create a virtual "test drive" experience where customers can simulate driving their favorite car models in various environments, from city streets to countryside roads, right from the showroom's website or app.
- **Luxury Car NFTs for Collectors:** Introduce limited-edition NFTs (non-fungible tokens) representing exclusive models or car features for collectors. Each NFT could include virtual benefits, like first access to new models or participation in exclusive events.

CONCLUSION:

The Luxury Car Showroom Management System is built to simplify and enhance the experience of managing and browsing premium cars. With a user-friendly interface developed in Python, it provides customers a smooth, immersive experience as they explore detailed car listings, images, and specifications, almost like stepping into a real showroom. Meanwhile, the MySQL back-end keeps everything organized, tracking each car, customer, and transaction seamlessly. The system is flexible, allowing for exciting future features like virtual test drives, AR customization, and VIP membership options. Altogether, this project brings the luxury car experience to users with both style and convenience.

BIBLIOGRAPHY

- https://www.mysqltutorial.org
- https/docs.python.org/3/library/sqlite3.html
- https:/dev.mysql.com/doc/
- https:/realpython.com/python-mysql/
- https:/pymysql.readthedocs.iohttps:/datatofish.com
- https:/towardsdatascience.com
- https:/www.programiz.com/python-programming
- https:/docs.sqlalchemy.org


thank
you