CASE STUDY- CAR CONNECT

HEXAWARE TRAINING

MEENAKSHI M - SAVEETHA ENGINEERING COLLEGE

Python - Batch - 4

INTRODUCTION

CarConnect is an innovative, database-driven car rental management system developed using Python and SQL Server, with a focus on modularity, maintainability, and real-world applicability. The system aims to streamline the operations of a car rental business by automating key functions such as customer registration, vehicle management, reservation handling, and administrative controls.

The system supports features like admin and customer authentication, vehicle availability tracking, reservation creation and updates, and report generation on vehicle utilization, reservation history, and overall revenue. By using MySQL Server as the backend database, CarConnect ensures reliable data persistence and integrity through proper use of foreign key constraints and exception handling.

Designed with scalability in mind, CarConnect can be adapted for small-to-medium car rental businesses looking to digitize their operations. Its extensible architecture also makes it a strong candidate for integration with web frameworks or APIs in future iterations.

Overall, CarConnect provides a solid foundation for learning enterprise-level application development with a focus on real-time problem-solving, data validation, database interaction, and user experience — making it not just a project, but a practical solution to modern rental business needs.

PURPOSE OF THE PROJECT

The purpose of the CarConnect project is to design and develop a comprehensive car rental management system that automates and simplifies the core operations of a car rental business. This system aims to replace traditional, manual processes with a centralized digital solution that allows administrators and customers to manage vehicles, reservations, and user data efficiently.

CarConnect is intended to:

- Enable customers to register, authenticate, and book available vehicles seamlessly.
- Allow administrators to manage vehicle inventories, update availability, and monitor customer reservations.
- Ensure secure data handling through proper authentication, validation, and database interaction.
- Provide visibility into operations via, status updates, and reporting features.
- Generate analytical reports such as reservation history, vehicle utilization, and revenue summaries to support business decision-making.

SCOPE OF THE PROJECT

The **CarConnect** system is designed to cover all major functional areas required to operate a car rental service efficiently. Its scope encompasses both administrative and customer-facing functionalities, integrating a structured backend with a relational database for reliable data management.

The key areas covered by the project include:

1. **Admin Management**:
   Admins can register, update, and delete their profiles. They can manage vehicle data, view reservation records, and generate reports to monitor business performance.

2. **Customer Management**:
   Customers can register, log in, view their profiles, and book vehicles. They can manage reservations and view booking history.

3. **Vehicle Management**:
   Admins can add, update, view, or remove vehicle details, including availability and daily rental rates. This ensures that the inventory remains up to date and accurate.

4. **Reservation System**:
   The system allows customers to make reservations for available vehicles by selecting dates and calculating total cost. Admins can confirm, cancel, or update the reservation status.

5. **Reporting Module**:
   Admins can generate various reports including reservation history, vehicle utilization, and revenue reports for business insights and performance tracking.

6. **Database Integration**:
   All operations are connected to a robust MySQL database ensuring data persistence, relational integrity, and smooth CRUD operations.

SQL TABLES

1. Customer Table:

- ***CustomerID (Primary Key):*** Unique identifier for each customer.
- ***FirstName:*** First name of the customer.
- ***LastName:*** Last name of the customer.
- ***Email:*** Email address of the customer for communication.
- ***PhoneNumber:*** Contact number of the customer.
- ***Address:*** Customer's residential address.
- ***Username:*** Unique username for customer login.
- ***Password:*** Securely hashed password for customer authentication.
- ***RegistrationDate:*** Date when the customer registered.

2. Vehicle Table:

- ***VehicleID (Primary Key):*** Unique identifier for each vehicle.
- ***Model:*** Model of the vehicle.
- ***Make:*** Manufacturer or brand of the vehicle.
- ***Year:*** Manufacturing year of the vehicle.
- ***Color:*** Color of the vehicle.
- ***RegistrationNumber:*** Unique registration number for each vehicle.
- ***Availability:*** Boolean indicating whether the vehicle is available for rent.
- ***DailyRate:*** Daily rental rate for the vehicle.

3. Reservation Table:

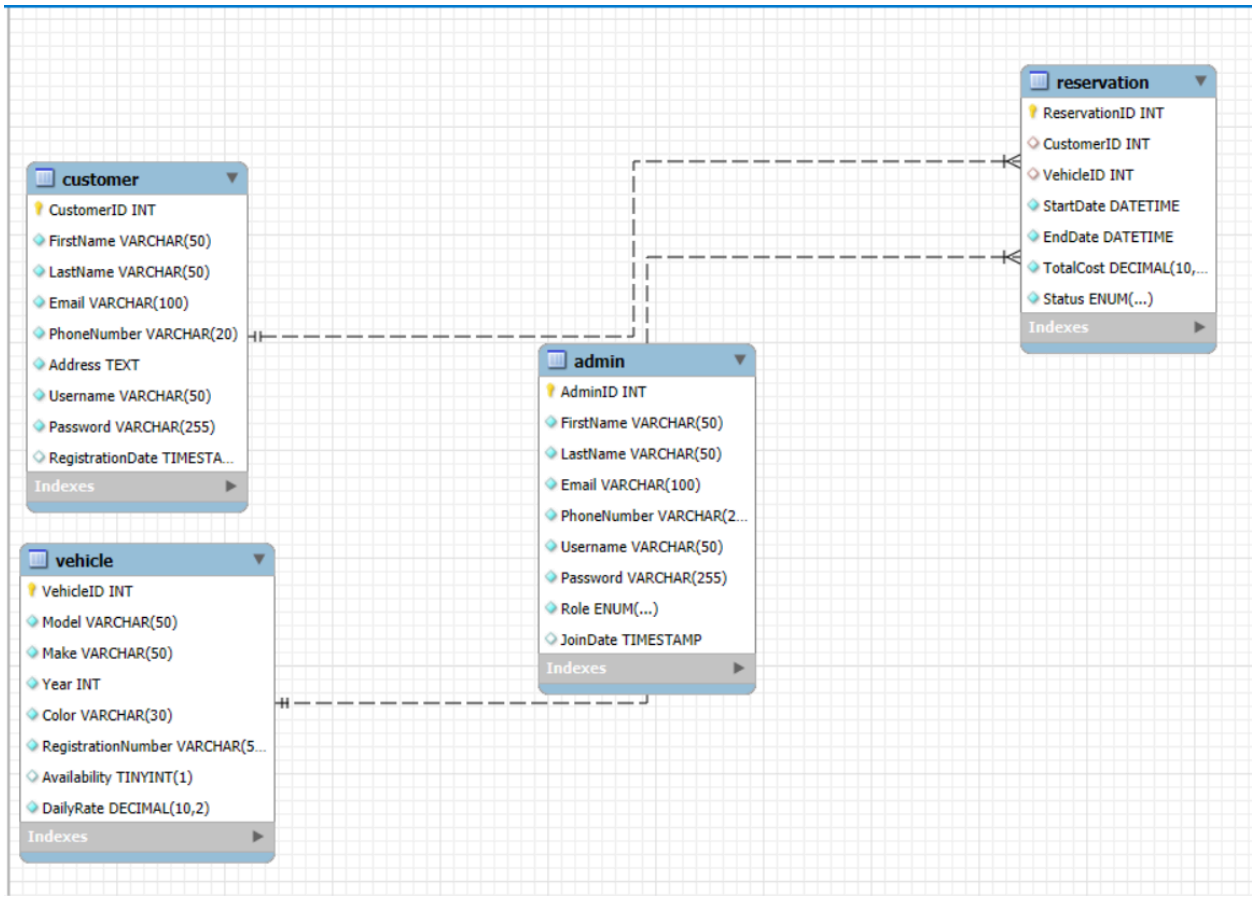
- ***ReservationID (Primary Key):*** Unique identifier for each reservation.
- ***CustomerID (Foreign Key):*** Foreign key referencing the Customer table.
- ***VehicleID (Foreign Key):*** Foreign key referencing the Vehicle table.
- ***StartDate:*** Date and time of the reservation start.
- ***EndDate:*** Date and time of the reservation end.
- ***TotalCost:*** Total cost of the reservation.
- ***Status:*** Current status of the reservation (e.g., pending, confirmed, completed).

4. Admin Table:

- ***AdminID (Primary Key):*** Unique identifier for each admin.
- ***FirstName:*** First name of the admin.
- ***LastName:*** Last name of the admin.
- ***Email:*** Email address of the admin for communication.
- ***PhoneNumber:*** Contact number of the admin.

- ***Username:*** Unique username for admin login.
- ***Password:*** Securely hashed password for admin authentication.
- ***Role:*** Role of the admin within the system (e.g., super admin, fleet manager).
- ***JoinDate:*** Date when the admin joined the system.

ER DIAGRAM



PYTHON PROGRAM

entity/ -

- Defines pure classes like Customer, Admin, Vehicle, and Reservation.
- Each class only holds attributes, no business logic.

dao/ -

- Contains service classes that interact with the database.
- Example: ReservationService contains CRUD operations for reservations.
- Follows DAO pattern: interface-like classes with methods like create, read, update, delete.

util/ -

- Manages the database connection (db_connection.py).
- Input validation utilities and helper functions.

exceptions/ -

- Houses all custom exceptions for input validation and business logic (e.g., InvalidInputException, ReservationException).

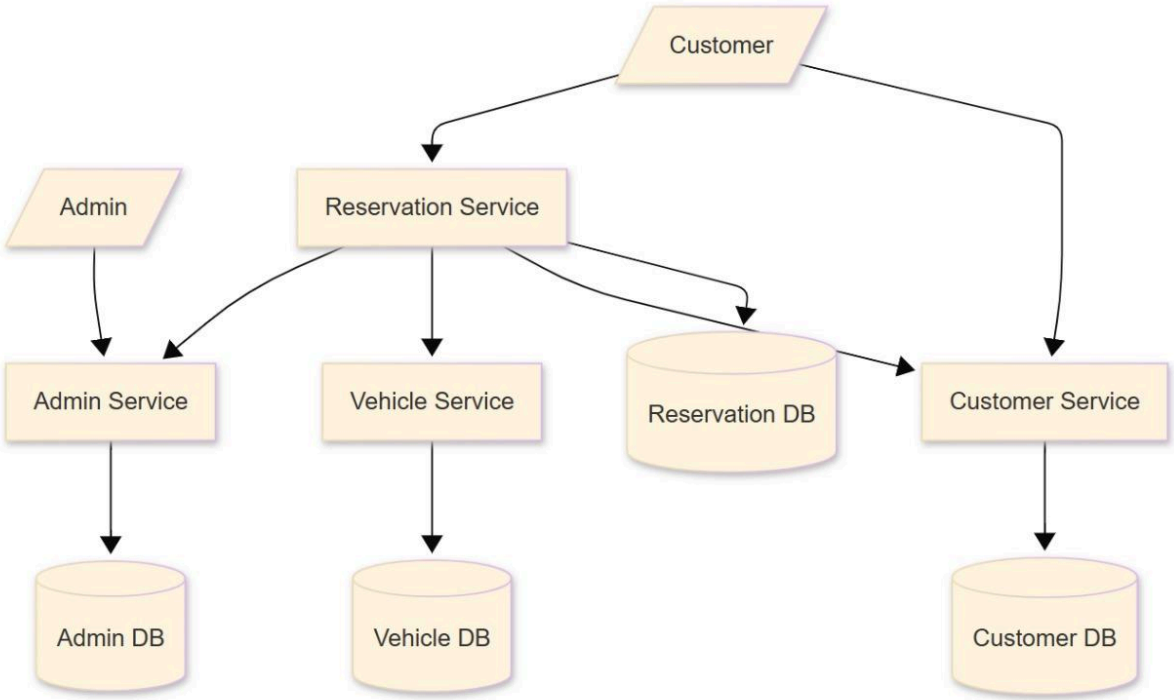
main/ -

- Provides the CLI interface with menu-driven flow for users/admins.
- Calls appropriate service classes based on user inputs.

test/ -

- Contains unit tests for each functional module.

DATA FLOW DIAGRAM



USED TECHNOLOGIES

The CarConnect project is developed using a modern and modular stack of technologies that ensures scalability, maintainability, and ease of use. Below are the key technologies and tools utilized:

Programming Language:

Python: Core language used for application logic, database interaction, and report generation.

Object-Oriented Programming (OOP) principles ensure modular and reusable code.

MySQL / SQL Server: Used to store all persistent data including customers, vehicles, reservations, and admins.

Supports relational integrity using foreign keys and constraints.

Testing :

unittest / pytest: Used to validate correctness of individual modules and business logic.

IDE:

PyCharm: Used for development and debugging.

SQL DATABASE:

1. Creating Database:

```
create database CarConnect;
use Carconnect;
```

2. Creating Tables:

Customer Table:

```
CREATE TABLE Customer (
    CustomerID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    PhoneNumber VARCHAR(20) NOT NULL,
    Address TEXT NOT NULL,
    Username VARCHAR(50) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL,
    RegistrationDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Vehicle Table:

```
CREATE TABLE Vehicle (
    VehicleID INT AUTO_INCREMENT PRIMARY KEY,
    Model VARCHAR(50) NOT NULL,
    Make VARCHAR(50) NOT NULL,
    Year INT NOT NULL,
    Color VARCHAR(30) NOT NULL,
    RegistrationNumber VARCHAR(50) UNIQUE NOT NULL,
    Availability BOOLEAN DEFAULT TRUE,
    DailyRate DECIMAL(10,2) NOT NULL
);
```

Reservation Table:

```
CREATE TABLE Reservation (
    ReservationID INT AUTO_INCREMENT PRIMARY KEY,
    CustomerID INT,
    VehicleID INT,
```

```
    StartDate DATETIME NOT NULL,
    EndDate DATETIME NOT NULL,
    TotalCost DECIMAL(10,2) NOT NULL,
    Status ENUM('pending', 'confirmed', 'completed', 'cancelled') NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE CASCADE,
    FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID) ON DELETE CASCADE
);
```

Admin Table:

```
CREATE TABLE Admin (
    AdminID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    PhoneNumber VARCHAR(20) NOT NULL,
    Username VARCHAR(50) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL,  -- Store hashed passwords
    Role ENUM('super admin', 'fleet manager') NOT NULL,
    JoinDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. Inserting Sample values

Customer Table:

```
INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address, Username,
Password)
VALUES
('Arjun', 'Rao', 'arjun.rao@example.com', '9876543210', '123 MG Road, Bangalore', 'arjunrao',
'pass123'),
('Priya', 'Sharma', 'priya.sharma@example.com', '9123456780', '456 Anna Salai, Chennai',
'priyasharma', 'pass123'),
('Vikram', 'Patel', 'vikram.patel@example.com', '9988776655', '789 FC Road, Pune', 'vikramp',
'pass123'),
('Sneha', 'Kumar', 'sneha.kumar@example.com', '9090909090', '11 Park Street, Kolkata', 'snehak',
'pass123'),
('Ravi', 'Verma', 'ravi.verma@example.com', '8012345678', '88 Marine Drive, Mumbai', 'raviv',
'pass123'),
('Divya', 'Singh', 'divya.singh@example.com', '9876501234', '19 Ashok Nagar, Delhi', 'divyasingh',
'pass123'),
('Karan', 'Mehta', 'karan.mehta@example.com', '9234567890', '40 JP Nagar, Bangalore', 'karanm',
```

'pass123'),
('Meena', 'Iyer', 'meena.iyer@example.com', '9345678901', '17 Purasawalkam, Chennai', 'meenai', 'pass123'),
('Ajay', 'Das', 'ajay.das@example.com', '9123456700', '9 EM Bypass, Kolkata', 'ajayd', 'pass123'),
('Lakshmi', 'Nair', 'lakshmi.nair@example.com', '9988007766', '55 Vyttila, Kochi', 'lakshmin', 'pass123');

Vehicle Table:

```
INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
VALUES
('Swift', 'Maruti', 2021, 'Red', 'KA01AB1234', TRUE, 1200.00),
('City', 'Honda', 2020, 'Black', 'TN02BC5678', TRUE, 1500.00),
('Innova', 'Toyota', 2019, 'Silver', 'MH03CD9101', TRUE, 2000.00),
('i20', 'Hyundai', 2022, 'White', 'DL04EF1122', TRUE, 1300.00),
('Creta', 'Hyundai', 2021, 'Grey', 'KL05GH3344', TRUE, 1800.00),
('Ertiga', 'Maruti', 2020, 'Blue', 'KA06IJ5566', TRUE, 1700.00),
('Fortuner', 'Toyota', 2023, 'Black', 'TN07KL7788', TRUE, 2500.00),
('Baleno', 'Maruti', 2021, 'Red', 'MH08MN9900', TRUE, 1400.00),
('Venue', 'Hyundai', 2022, 'White', 'DL09OP1112', TRUE, 1600.00),
('Altroz', 'Tata', 2020, 'Yellow', 'KL10QR1314', TRUE, 1100.00);
```

Reservation Table:

```
INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
VALUES
(1, 2, '2025-03-01 10:00:00', '2025-03-05 10:00:00', 6000.00, 'completed'),
(2, 4, '2025-03-03 09:00:00', '2025-03-04 09:00:00', 1300.00, 'completed'),
(3, 1, '2025-03-07 12:00:00', '2025-03-10 12:00:00', 3600.00, 'confirmed'),
(4, 6, '2025-03-11 08:00:00', '2025-03-14 08:00:00', 5100.00, 'cancelled'),
(5, 3, '2025-03-05 18:00:00', '2025-03-06 18:00:00', 2000.00, 'completed'),
(6, 7, '2025-03-08 10:00:00', '2025-03-09 10:00:00', 2500.00, 'confirmed'),
(7, 5, '2025-03-12 11:00:00', '2025-03-13 11:00:00', 1800.00, 'pending'),
(8, 9, '2025-03-14 13:00:00', '2025-03-16 13:00:00', 3200.00, 'pending'),
(9, 8, '2025-03-01 10:00:00', '2025-03-02 10:00:00', 1400.00, 'completed'),
(10, 10, '2025-03-02 09:00:00', '2025-03-04 09:00:00', 2200.00, 'confirmed');
```

Admin Table:

```
INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password, Role)
VALUES
('Ramesh', 'Iyer', 'ramesh.iyer@carconnect.com', '9999988888', 'rameshadmin', 'admin123', 'super
```

admin'),
('Geeta', 'Menon', 'geeta.menon@carconnect.com', '9888777666', 'geetamenon', 'admin123', 'fleet manager'),
('Suraj', 'Singh', 'suraj.singh@carconnect.com', '9777666555', 'surajsingh', 'admin123', 'fleet manager'),
('Kavita', 'Das', 'kavita.das@carconnect.com', '9666555444', 'kavitadas', 'admin123', 'super admin'),
('Anil', 'Jain', 'anil.jain@carconnect.com', '9555444333', 'aniljain', 'admin123', 'fleet manager'),
('Pooja', 'Rao', 'pooja.rao@carconnect.com', '9444333222', 'poojarao', 'admin123', 'super admin'),
('Naveen', 'Kumar', 'naveen.kumar@carconnect.com', '9333222111', 'naveenk', 'admin123', 'fleet manager'),
('Meera', 'Nair', 'meera.nair@carconnect.com', '9222111000', 'meeranair', 'admin123', 'fleet manager'),
('Rahul', 'Verma', 'rahul.verma@carconnect.com', '9111000099', 'rahulverma', 'admin123', 'super admin'),
('Divya', 'Joshi', 'divya.joshi@carconnect.com', '9000099999', 'divyajoshi', 'admin123', 'fleet manager');

4. Python Program

Entity:

admin.py:

```
      class Admin:

          def __init__(self, admin_id, first_name, last_name, email, phone,
username, password, role, join_date):

              self.admin_id = admin_id

              self.first_name = first_name

              self.last_name = last_name

              self.email = email

              self.phone = phone

              self.username = username

              self.password = password

              self.role = role

              self.join_date = join_date


          def authenticate(self, input_password):
```

```
            return self.password == input_password
```

customer.py:

```
    class Customer:

        def __init__(self, customer_id, first_name, last_name, email, phone,
address, username, password, registration_date):

            self.customer_id = customer_id

            self.first_name = first_name

            self.last_name = last_name

            self.email = email

            self.phone = phone

            self.address = address

            self.username = username

            self.password = password

            self.registration_date = registration_date



        def authenticate(self, input_password):

            return self.password == input_password
```

reservation.py:

```
    class Reservation:

        def __init__(self, reservation_id, customer_id, vehicle_id,
start_date, end_date, total_cost, status):

            self.reservation_id = reservation_id

            self.customer_id = customer_id

            self.vehicle_id = vehicle_id

            self.start_date = start_date

            self.end_date = end_date
```

```
            self.total_cost = total_cost

            self.status = status



        def calculate_total_cost(self, daily_rate, days):

            self.total_cost = daily_rate * days
```

Vehicle.py:

```
      class Vehicle:

         def __init__(self, vehicle_id, model, make, year, color,
registration_number, availability, daily_rate):

            self.vehicle_id = vehicle_id

            self.model = model

            self.make = make

            self.year = year

            self.color = color

            self.registration_number = registration_number

            self.availability = availability

            self.daily_rate = daily_rate
```

Doa:

Admin_serives.py:

```
from CarConnect.exceptions.admin_not_found_exception import
AdminNotFoundException

from CarConnect.exceptions.invalid_input_exception import
InvalidInputException

from CarConnect.exceptions.database_connection_exception import
DatabaseConnectionException



class AdminService:
```

```
    def __init__(self, db):
        self.db = db

    def get_admin_by_id(self, admin_id):
        if not admin_id.isdigit():
            raise InvalidInputException("Admin ID must be an integer.")
        try:
            query = "SELECT * FROM Admin WHERE AdminID = %s"
            row = self.db.fetch_query(query, (admin_id,))
            if not row:
                raise AdminNotFoundException(f"Admin with ID {admin_id} not
found.")
            print("The Admin is:",row)
        except DatabaseConnectionException as e:
            raise DatabaseConnectionException(f"Database error: {str(e)}")

    def get_admin_by_username(self, username):
        if not isinstance(username, str) or not username.strip():
            raise InvalidInputException("Username must be a non-empty string.")
        try:
            query = "SELECT * FROM Admin WHERE Username = %s"
            row = self.db.fetch_query(query, (username,))
            if not row:
                raise AdminNotFoundException(f"No admin found with username:
{username}")
            print("The user is:",row)
        except AdminNotFoundException:
            raise AdminNotFoundException(f"No admin found with username:
{username}")
```

```
        except DatabaseConnectionException as e:

            raise DatabaseConnectionException(f"Database error: {str(e)}")



    def register_admin(self, admin):

        if not all([admin.first_name.strip(), admin.last_name.strip(),
admin.email.strip(),

                    admin.phone.strip(), admin.username.strip(),
admin.password.strip(), admin.role.strip()]):

            raise InvalidInputException("Admin fields must not be empty.")



        if not (admin.phone.isdigit() and len(admin.phone) == 10):

            raise InvalidInputException("Phone number must be a 10-digit
number.")



        if admin.role not in ['super admin', 'fleet manager']:

            raise InvalidInputException("Role must be 'super admin' or 'fleet
manager'.")

        try:

            query = """

                INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber,
Username, Password, Role, JoinDate)

                VALUES (%s, %s, %s, %s, %s, %s, %s, NOW())

            """

            values = (

                admin.first_name, admin.last_name, admin.email,

                admin.phone, admin.username, admin.password, admin.role

            )

            self.db.execute_query(query, values)

        except DatabaseConnectionException as e:

            raise DatabaseConnectionException(f"Failed to register admin:
{str(e)}")
```

```
   def update_admin(self, admin_id, first_name,
last_name,email,phone,username,role):

       if not admin_id.isdigit():

           raise InvalidInputException("Admin ID must be an integer.")



       if not (phone.isdigit() and len(phone) == 10):

           raise InvalidInputException("Phone number must be a 10-digit
number.")



       if role not in ['super admin', 'fleet manager']:

           raise InvalidInputException("Role must be 'super admin' or 'fleet
manager'.")

       try:

           query = ("UPDATE Admin SET FirstName = %s, LastName = %s,"

                    "Email = %s, PhoneNumber = %s, username = %s, role = %s
WHERE AdminID = %s")

           result = self.db.execute_query(query, (first_name, last_name,email,
phone,username,role, admin_id))

           if result == 0:

               raise AdminNotFoundException(f"Admin with ID {admin_id} not
found.")

       except DatabaseConnectionException as e:

           raise DatabaseConnectionException(f"Failed to update admin:
{str(e)}")



   def delete_admin(self, admin_id):

       if not admin_id.isdigit():

           raise InvalidInputException("Admin ID must be an integer.")

       try:

           query = "DELETE FROM Admin WHERE AdminID = %s"
```

```
            result = self.db.execute_query(query, (admin_id,))

            if result == 0:

                raise AdminNotFoundException(f"Admin with ID {admin_id} not
found.")

        except DatabaseConnectionException as e:

            raise DatabaseConnectionException(f"Failed to delete admin:
{str(e)}")
```

Customer_service.py:

```
from CarConnect.exceptions.invalid_input_exception import
InvalidInputException

from CarConnect.exceptions.authentication_exception import
AuthenticationException

from CarConnect.exceptions.customer_not_found_exception import
CustomerNotFoundException



class CustomerService:

    def __init__(self, db):

        self.db = db



    def get_customer_by_id(self, customer_id):

        if not customer_id.isdigit():

            raise InvalidInputException("Customer ID must be an integer.")

        query = "SELECT * FROM Customer WHERE CustomerID = %s"

        result = self.db.fetch_query(query, (customer_id,))

        if not result:

            raise CustomerNotFoundException(f"Customer with ID '{customer_id}'
not found.")

        print("The Customer: ",result)
```

```
   def get_customer_by_username(self, username):

       if not isinstance(username, str) or not username.strip():

           raise InvalidInputException("Username must be a non-empty string.")

       query = "SELECT * FROM Customer WHERE Username = %s"

       result = self.db.fetch_query(query, (username,))

       if not result:

           raise CustomerNotFoundException(f"Customer with username
'{username}' not found.")

       print("The Customer by ID: ",result)



   def register_customer(self, customer):

       if not
all([customer.first_name.strip(),customer.last_name.strip(),customer.email.str
ip(),

                   customer.address.strip(),
customer.username.strip(),customer.password.strip()]):

           raise InvalidInputException("Fields must not be empty.")



       if not (customer.phone.isdigit() and len(customer.phone) == 10):

           raise InvalidInputException("Phone number must be a 10-digit
number.")



       query = """

           INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber,
Address, Username, Password, RegistrationDate)

           VALUES (%s, %s, %s, %s, %s, %s, %s, NOW())

       """

       self.db.execute_query(query, (

           customer.first_name, customer.last_name, customer.email,

           customer.phone, customer.address, customer.username,
customer.password
```

```
        ))



    def update_customer(self, customer_id,first_name,last_name ,email, phone,
address,username):

        if not customer_id.isdigit():

            raise InvalidInputException("Customer ID must be an integer.")

        query = """

            UPDATE Customer SET firstname = %s,lastname = %s,

            Email = %s, PhoneNumber = %s, Address = %s,username = %s WHERE
CustomerID = %s

        """

        result = self.db.execute_query(query, (first_name,last_name,email,
phone, address,username,customer_id))

        if result == 0:

            raise CustomerNotFoundException(f"Customer ID {customer_id} not
found")



    def delete_customer(self, customer_id):

        if not customer_id.isdigit():

            raise InvalidInputException("Customer ID must be an integer.")

        query = "DELETE FROM Customer WHERE CustomerID = %s"

        result = self.db.execute_query(query, (customer_id,))

        if result == 0:

            raise CustomerNotFoundException(f"Customer ID {customer_id} not
found")



    def authenticate_customer(self, username, password):

        if not isinstance(username, str) or not username.strip():

            raise InvalidInputException("Username must be a non-empty string.")

        if not isinstance(password, str) or not password.strip():
```

```
            raise InvalidInputException("Password must be a non-empty string.")


        query = "SELECT * FROM Customer WHERE Username = %s AND Password = %s"
        result = self.db.fetch_query(query, (username, password))
        if not result:
            raise AuthenticationException("Invalid username or password.")
        print("The User is:",result)
```

Reservation_services.py:

```
from CarConnect.exceptions.invalid_input_exception import
InvalidInputException
from CarConnect.exceptions.reservation_exception import ReservationException


class ReservationService:
    def __init__(self, db):
        self.db = db


    def get_reservation_by_id(self, reservation_id):
        if not reservation_id.isdigit():
            raise InvalidInputException("Reservation ID must be an integer.")


        query = "SELECT * FROM Reservation WHERE ReservationID = %s"
        result = self.db.fetch_query(query, (reservation_id,))


        if not result:
            raise ReservationException(f"No reservation found with ID:
{reservation_id}")
```

```
        print(result)


    def get_reservations_by_customer_id(self, customer_id):
        if not customer_id.isdigit():
            raise InvalidInputException("Customer ID must be an integer.")

        query = "SELECT * FROM Reservation WHERE CustomerID = %s"
        result = self.db.fetch_query(query, (customer_id,))

        if not result:
            raise ReservationException(f"No reservations found for customer ID:
{customer_id}")

        for row in result:
            reservation = Reservation(*row)
            print(f"Reservation ID: {reservation.reservation_id}")
            print(f"Vehicle ID    : {reservation.vehicle_id}")
            print(f"Start Date    : {reservation.start_date}")
            print(f"End Date      : {reservation.end_date}")
            print(f"Total Cost    : {reservation.total_cost}")
            print(f"Status        : {reservation.status}")

    def create_reservation(self, reservation):
        if not reservation.customer_id.isdigit() or not
reservation.vehicle_id.isdigit():
            raise InvalidInputException("Enter Integer value for Customer and
Vehicle")

        query = """
```

```
            INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate,
TotalCost, Status)

            VALUES (%s, %s, %s, %s, %s, %s)

        """

        self.db.execute_query(query, (

            reservation.customer_id, reservation.vehicle_id,
reservation.start_date,

            reservation.end_date, reservation.total_cost, reservation.status

        ))



    def update_reservation(self, reservation_id, status):

        if not reservation_id.isdigit():

            raise InvalidInputException("Reservation ID must be an integer.")



        query = "UPDATE Reservation SET Status = %s WHERE ReservationID = %s"

        rowcount = self.db.execute_query(query, (status, reservation_id))



        if rowcount == 0:

            raise ReservationException(f"No reservation found with ID:
{reservation_id}")



    def cancel_reservation(self, reservation_id):

        if not reservation_id.isdigit():

            raise InvalidInputException("Reservation ID must be an integer.")



        query = "DELETE FROM Reservation WHERE ReservationID = %s"

        rowcount = self.db.execute_query(query, (reservation_id,))



        if rowcount == 0:
```

```
            raise ReservationException(f"No reservation found with ID:
{reservation_id}")



    def generate_reservation_history_report(self):

        query = """

            SELECT ReservationID, CustomerID, VehicleID, StartDate, EndDate,
Status

            FROM Reservation

            ORDER BY StartDate DESC

        """

        results = self.db.fetch_query(query)

        print("\n--- Reservation History Report ---")

        for row in results:

            print(row)



    def generate_vehicle_utilization_report(self):

        query = """

            SELECT VehicleID, COUNT(*) AS TotalReservations

            FROM Reservation

            GROUP BY VehicleID

            ORDER BY TotalReservations DESC

        """

        results = self.db.fetch_query(query)

        print("\n--- Vehicle Utilization Report ---")

        for row in results:

            print(f"Vehicle ID: {row[0]}, Reservations: {row[1]}")



    def generate_revenue_report(self):

        query = """
```

```
            SELECT VehicleID, SUM(TotalCost) AS Revenue
            FROM Reservation
            WHERE Status = 'Completed'
            GROUP BY VehicleID
            ORDER BY Revenue DESC
        """
        results = self.db.fetch_query(query)
        print("\n--- Revenue Report ---")
        for row in results:
            print(f"Vehicle ID: {row[0]}, Revenue: ₹{row[1]:.2f}")
```

Vehicle_services.py:

```
from CarConnect.exceptions.vehicle_not_found_exception import
VehicleNotFoundException

from CarConnect.exceptions.invalid_input_exception import
InvalidInputException

from CarConnect.exceptions.database_connection_exception import
DatabaseConnectionException

import re



class VehicleService:

    def __init__(self, db):
        self.db = db



    def get_vehicle_by_id(self, vehicle_id):
        try:
            query = "SELECT * FROM Vehicle WHERE VehicleID = %s"
            row = self.db.fetch_query(query, (vehicle_id,))
```

```
            if not row:
                raise VehicleNotFoundException(f"No vehicle found with ID:
{vehicle_id}")
            print("The vehicle is: ",row)
        except DatabaseConnectionException as e:
            raise DatabaseConnectionException(f"Database error: {str(e)}")


    def get_available_vehicles(self):
        try:
            query = "SELECT * FROM Vehicle WHERE Availability = 1"
            rows = self.db.fetch_query(query)
            return rows
        except DatabaseConnectionException as e:
            raise DatabaseConnectionException(f"Database error: {str(e)}")


    def add_vehicle(self, vehicle):
        pattern = r'^[A-Za-z]{2}\s?[0-9]{2}\s?[A-Za-z]{2}\s?[0-9]{4}$'
        def is_valid_format(text):
            return bool(re.match(pattern, text))
        if not is_valid_format(vehicle.registration_number):
            raise InvalidInputException("Enter valid registration number")


        if not vehicle.year.isdigit() or len(vehicle.year) != 4:
            raise InvalidInputException("Year must be a valid integer.")
        try:
            query = """
```

```
                INSERT INTO Vehicle (Model, Make, Year, Color,
RegistrationNumber, Availability, DailyRate)

                VALUES (%s, %s, %s, %s, %s, %s, %s)

            """

            values = (

                vehicle.model, vehicle.make, vehicle.year, vehicle.color,

                vehicle.registration_number, vehicle.availability,
vehicle.daily_rate

            )

            self.db.execute_query(query, values)

        except Exception as e:

            raise DatabaseConnectionException(f"Failed to add vehicle:
{str(e)}")



    def update_vehicle(self, vehicle_id, daily_rate, availability):

        if not vehicle_id.isdigit():

            raise InvalidInputException("Vehicle ID must be an integer.")

        try:

            query = """

                UPDATE Vehicle SET DailyRate = %s, Availability = %s WHERE
VehicleID = %s

            """

            result = self.db.execute_query(query, (daily_rate, availability,
vehicle_id))

            if result == 0:

                raise VehicleNotFoundException(f"No vehicle found with ID:
{vehicle_id}")

        except DatabaseConnectionException as e:
```

```
            raise DatabaseConnectionException(f"Failed to update vehicle:
{str(e)}")



    def remove_vehicle(self, vehicle_id):

        if not vehicle_id.isdigit():

            raise InvalidInputException("Vehicle ID must be an integer.")

        try:

            query = "DELETE FROM Vehicle WHERE VehicleID = %s"

            result = self.db.execute_query(query, (vehicle_id,))

            if result == 0:

                raise VehicleNotFoundException(f"No vehicle found with ID:
{vehicle_id}")

        except DatabaseConnectionException as e:

            raise DatabaseConnectionException(f"Failed to delete vehicle:
{str(e)}")
```

Exceptions:

1. **Authentication Exception:**

```
class AuthenticationException(Exception):

    def __init__(self, message="Invalid username or password."):

        super().__init__(message)
```

2. **Database Connection Exception:**

```
class DatabaseConnectionException(Exception):

    def __init__(self, message="Unable to connect to the database."):

        super().__init__(message)
```

3. **Invalid Input Exception:**

```
class InvalidInputException(Exception):
    def __init__(self, message="Invalid input provided."):
        super().__init__(message)
```

4. **Reservation Exception:**

```
class ReservationException(Exception):
    def __init__(self, message="Error in processing the reservation."):
        super().__init__(message)
```

5. **Admin not found exception:**

```
class AdminNotFoundException(Exception):
    def __init__(self, message="Admin not found."):
        super().__init__(message)
```

6. **Vehicle not found exception:**

```
class VehicleNotFoundException(Exception):
    def __init__(self, message="Vehicle not found."):
        super().__init__(message)
```

7. **Customer not found exception**

```
class CustomerNotFoundException(Exception):
    def __init__(self, message="Customer not found"):
        super().__init__(message)
```

Testing:

1. **Admin testing**

```python
import unittest

from unittest.mock import MagicMock

from CarConnect.entity.admin import Admin

from CarConnect.dao.admin_service import AdminService

from CarConnect.exceptions.admin_not_found_exception import
AdminNotFoundException

from CarConnect.exceptions.invalid_input_exception import
InvalidInputException

from CarConnect.exceptions.database_connection_exception import
DatabaseConnectionException



class TestAdminService(unittest.TestCase):

    def setUp(self):

        self.mock_db = MagicMock()

        self.service = AdminService(self.mock_db)


    def test_get_admin_by_id_valid(self):

        self.mock_db.fetch_query.return_value = [("1", "John", "Doe")]

        self.service.get_admin_by_id("1")

        self.mock_db.fetch_query.assert_called_once()


    def test_get_admin_by_id_invalid_input(self):

        with self.assertRaises(InvalidInputException):

            self.service.get_admin_by_id("abc")


    def test_get_admin_by_id_not_found(self):

        self.mock_db.fetch_query.return_value = []

        with self.assertRaises(AdminNotFoundException):

            self.service.get_admin_by_id("999")
```

```
    def test_get_admin_by_id_db_error(self):

        self.mock_db.fetch_query.side_effect =
DatabaseConnectionException("DB error")

        with self.assertRaises(DatabaseConnectionException):

            self.service.get_admin_by_id("1")

    def test_get_admin_by_username_valid(self):

        self.mock_db.fetch_query.return_value = [("admin1", "Admin")]

        self.service.get_admin_by_username("admin1")

        self.mock_db.fetch_query.assert_called_once()


    def test_get_admin_by_username_invalid_input(self):

        with self.assertRaises(InvalidInputException):

            self.service.get_admin_by_username("")


    def test_get_admin_by_username_not_found(self):

        self.mock_db.fetch_query.return_value = []

        with self.assertRaises(AdminNotFoundException):

            self.service.get_admin_by_username("ghostadmin")


    def test_get_admin_by_username_db_error(self):

        self.mock_db.fetch_query.side_effect =
DatabaseConnectionException("DB error")

        with self.assertRaises(DatabaseConnectionException):

            self.service.get_admin_by_username("admin1")

    def test_register_admin_valid(self):

        admin = Admin(None, "John", "Doe", "john@example.com",
"1234567890", "admin1", "pass123", "super admin", None)

        self.service.register_admin(admin)
```

```
        self.mock_db.execute_query.assert_called_once()



    def test_register_admin_invalid_input_empty_fields(self):

        admin = Admin(None, "", "Doe", "john@example.com", "1234567890",
"admin1", "pass123", "super admin", None)

        with self.assertRaises(InvalidInputException):

            self.service.register_admin(admin)



    def test_register_admin_invalid_input_phone(self):

        admin = Admin(None, "John", "Doe", "john@example.com", "12345",
"admin1", "pass123", "super admin", None)

        with self.assertRaises(InvalidInputException):

            self.service.register_admin(admin)



    def test_register_admin_invalid_input_role(self):

        admin = Admin(None, "John", "Doe", "john@example.com",
"1234567890", "admin1", "pass123", "manager", None)

        with self.assertRaises(InvalidInputException):

            self.service.register_admin(admin)



    def test_register_admin_db_error(self):

        self.mock_db.execute_query.side_effect =
DatabaseConnectionException("Insert failed")

        admin = Admin(None, "John", "Doe", "john@example.com",
"1234567890", "admin1", "pass123", "super admin", None)

        with self.assertRaises(DatabaseConnectionException):

            self.service.register_admin(admin)



    def test_update_admin_valid(self):

        self.mock_db.execute_query.return_value = 1
```

```
        self.service.update_admin("1", "John", "Smith",
"johnsmith@example.com", "9876543210", "johnsmith", "super admin")

        self.mock_db.execute_query.assert_called_once()



    def test_update_admin_invalid_input_id(self):

        with self.assertRaises(InvalidInputException):

            self.service.update_admin("abc", "John", "Smith", "email",
"9876543210", "username", "super admin")



    def test_update_admin_invalid_input_phone(self):

        with self.assertRaises(InvalidInputException):

            self.service.update_admin("1", "John", "Smith", "email",
"phone", "username", "super admin")



    def test_update_admin_invalid_input_role(self):

        with self.assertRaises(InvalidInputException):

            self.service.update_admin("1", "John", "Smith", "email",
"9876543210", "username", "admin")



    def test_update_admin_not_found(self):

        self.mock_db.execute_query.return_value = 0

        with self.assertRaises(AdminNotFoundException):

            self.service.update_admin("1", "John", "Smith", "email",
"9876543210", "username", "super admin")



    def test_update_admin_db_error(self):

        self.mock_db.execute_query.side_effect =
DatabaseConnectionException("Update failed")

        with self.assertRaises(DatabaseConnectionException):

            self.service.update_admin("1", "John", "Smith", "email",
"9876543210", "username", "super admin")
```

```
    def test_delete_admin_valid(self):
        self.mock_db.execute_query.return_value = 1
        self.service.delete_admin("1")
        self.mock_db.execute_query.assert_called_once()

    def test_delete_admin_invalid_input(self):
        with self.assertRaises(InvalidInputException):
            self.service.delete_admin("abc")

    def test_delete_admin_not_found(self):
        self.mock_db.execute_query.return_value = 0
        with self.assertRaises(AdminNotFoundException):
            self.service.delete_admin("999")

    def test_delete_admin_db_error(self):
        self.mock_db.execute_query.side_effect =
DatabaseConnectionException("Delete failed")
        with self.assertRaises(DatabaseConnectionException):
            self.service.delete_admin("1")

if __name__ == "__main__":
    unittest.main()
```

2. Customer Testing:

```
import unittest
from unittest.mock import MagicMock
from datetime import date
from CarConnect.dao.customer_service import CustomerService
```

```
from CarConnect.entity.customer import Customer

from CarConnect.exceptions.invalid_input_exception import
InvalidInputException

from CarConnect.exceptions.customer_not_found_exception import
CustomerNotFoundException

from CarConnect.exceptions.authentication_exception import
AuthenticationException



class TestCustomerService(unittest.TestCase):

    def setUp(self):

        self.mock_db = MagicMock()

        self.service = CustomerService(self.mock_db)



    def test_get_customer_by_id_valid(self):

        self.mock_db.fetch_query.return_value = [("1", "John", "Doe",
"john@example.com", "1234567890", "Address", "johndoe", "pass123",
date.today())]

        self.service.get_customer_by_id("1")

        self.mock_db.fetch_query.assert_called_once()



    def test_get_customer_by_id_invalid(self):

        with self.assertRaises(InvalidInputException):

            self.service.get_customer_by_id("abc")



    def test_get_customer_by_id_not_found(self):

        self.mock_db.fetch_query.return_value = []

        with self.assertRaises(CustomerNotFoundException):

            self.service.get_customer_by_id("999")



    def test_get_customer_by_username_valid(self):
```

```
        self.mock_db.fetch_query.return_value = [("1", "John", "Doe",
"john@example.com", "1234567890", "Address", "johndoe", "pass123",
date.today())]

        self.service.get_customer_by_username("johndoe")

        self.mock_db.fetch_query.assert_called_once()



    def test_get_customer_by_username_invalid(self):

        with self.assertRaises(InvalidInputException):

            self.service.get_customer_by_username("")



    def test_get_customer_by_username_not_found(self):

        self.mock_db.fetch_query.return_value = []

        with self.assertRaises(CustomerNotFoundException):

            self.service.get_customer_by_username("unknown_user")



    def test_register_customer_valid(self):

        customer = Customer(None, "Jane", "Doe", "jane@example.com",
"1234567890", "Somewhere", "janedoe", "securepass", None)

        self.service.register_customer(customer)

        self.mock_db.execute_query.assert_called_once()



    def test_register_customer_invalid_phone(self):

        customer = Customer(None, "Jane", "Doe", "jane@example.com",
"12345abc", "Somewhere", "janedoe", "securepass", None)

        with self.assertRaises(InvalidInputException):

            self.service.register_customer(customer)



    def test_register_customer_empty_fields(self):

        customer = Customer(None, "", "Doe", "jane@example.com",
"1234567890", "Somewhere", "janedoe", "securepass", None)
```

```
        with self.assertRaises(InvalidInputException):

            self.service.register_customer(customer)



    def test_update_customer_valid(self):

        self.mock_db.execute_query.return_value = 1

        self.service.update_customer("1", "Jane", "Doe",
"jane@example.com", "1234567890", "Somewhere", "janedoe")

        self.mock_db.execute_query.assert_called_once()



    def test_update_customer_invalid_id(self):

        with self.assertRaises(InvalidInputException):

            self.service.update_customer("abc", "Jane", "Doe",
"jane@example.com", "1234567890", "Somewhere", "janedoe")



    def test_update_customer_not_found(self):

        self.mock_db.execute_query.return_value = 0

        with self.assertRaises(CustomerNotFoundException):

            self.service.update_customer("99", "Jane", "Doe",
"jane@example.com", "1234567890", "Somewhere", "janedoe")



    def test_delete_customer_valid(self):

        self.mock_db.execute_query.return_value = 1

        self.service.delete_customer("1")

        self.mock_db.execute_query.assert_called_once()



    def test_delete_customer_invalid_id(self):

        with self.assertRaises(InvalidInputException):

            self.service.delete_customer("abc")
```

```
    def test_delete_customer_not_found(self):

        self.mock_db.execute_query.return_value = 0

        with self.assertRaises(CustomerNotFoundException):

            self.service.delete_customer("99")



    def test_authenticate_customer_valid(self):

        self.mock_db.fetch_query.return_value = [("1", "Jane", "Doe",
"jane@example.com", "1234567890", "Somewhere", "janedoe", "securepass",
date.today())]

        self.service.authenticate_customer("janedoe", "securepass")

        self.mock_db.fetch_query.assert_called_once()



    def test_authenticate_customer_invalid_input(self):

        with self.assertRaises(InvalidInputException):

            self.service.authenticate_customer("", "password")

        with self.assertRaises(InvalidInputException):

            self.service.authenticate_customer("username", "")



    def test_authenticate_customer_failure(self):

        self.mock_db.fetch_query.return_value = []

        with self.assertRaises(AuthenticationException):

            self.service.authenticate_customer("janedoe", "wrongpass")



if __name__ == '__main__':

    unittest.main()
```

3. Reservation Testing:

```
import unittest
```

```
from unittest.mock import MagicMock

from datetime import date

from CarConnect.entity.reservation import Reservation

from CarConnect.dao.reservation_service import ReservationService

from CarConnect.exceptions.invalid_input_exception import
InvalidInputException

from CarConnect.exceptions.reservation_exception import
ReservationException



class TestReservationService(unittest.TestCase):

    def setUp(self):

        self.mock_db = MagicMock()

        self.service = ReservationService(self.mock_db)



    def test_get_reservation_by_id_valid(self):

        self.mock_db.fetch_query.return_value = [

            ("1", "2", "3", date(2024, 5, 1), date(2024, 5, 5),
"5000.00", "Confirmed")]

        self.service.get_reservation_by_id("1")

        self.mock_db.fetch_query.assert_called_once()



    def test_get_reservation_by_id_invalid_input(self):

        with self.assertRaises(InvalidInputException):

            self.service.get_reservation_by_id("abc")



    def test_get_reservation_by_id_not_found(self):

        self.mock_db.fetch_query.return_value = []

        with self.assertRaises(ReservationException):

            self.service.get_reservation_by_id("99")
```

```
    def test_get_reservations_by_customer_id_valid(self):

        self.mock_db.fetch_query.return_value = [

            ("1", "2", "3", date(2024, 5, 1), date(2024, 5, 5),
"5000.00", "Confirmed"),

            ("2", "2", "4", date(2024, 6, 1), date(2024, 6, 3),
"3000.00", "Pending"),

        ]

        self.service.get_reservations_by_customer_id("2")

        self.mock_db.fetch_query.assert_called_once()


    def test_get_reservations_by_customer_id_invalid_input(self):

        with self.assertRaises(InvalidInputException):

            self.service.get_reservations_by_customer_id("abc")


    def test_get_reservations_by_customer_id_not_found(self):

        self.mock_db.fetch_query.return_value = []

        with self.assertRaises(ReservationException):

            self.service.get_reservations_by_customer_id("55")


    def test_create_reservation_valid(self):

        reservation = Reservation(None, "2", "3", date(2024, 5, 1),
date(2024, 5, 5), "5000.00", "Confirmed")

        self.service.create_reservation(reservation)

        self.mock_db.execute_query.assert_called_once()


    def test_create_reservation_invalid_customer_vehicle_id(self):

        reservation = Reservation(None, "abc", "3", date(2024, 5, 1),
date(2024, 5, 5), "5000.00", "Confirmed")

        with self.assertRaises(InvalidInputException):
```

```
            self.service.create_reservation(reservation)

    def test_update_reservation_valid(self):
        self.mock_db.execute_query.return_value = 1
        self.service.update_reservation("1", "Cancelled")
        self.mock_db.execute_query.assert_called_once()

    def test_update_reservation_invalid_input(self):
        with self.assertRaises(InvalidInputException):
            self.service.update_reservation("abc", "Cancelled")

    def test_update_reservation_not_found(self):
        self.mock_db.execute_query.return_value = 0
        with self.assertRaises(ReservationException):
            self.service.update_reservation("999", "Completed")

    def test_cancel_reservation_valid(self):
        self.mock_db.execute_query.return_value = 1
        self.service.cancel_reservation("1")
        self.mock_db.execute_query.assert_called_once()

    def test_cancel_reservation_invalid_input(self):
        with self.assertRaises(InvalidInputException):
            self.service.cancel_reservation("abc")

    def test_cancel_reservation_not_found(self):
        self.mock_db.execute_query.return_value = 0
        with self.assertRaises(ReservationException):
```

```
            self.service.cancel_reservation("999")


    def test_generate_reservation_history_report(self):
        self.mock_db.fetch_query.return_value = [
            (1, 2, 3, date(2024, 5, 1), date(2024, 5, 5), "Confirmed")
        ]
        self.service.generate_reservation_history_report()
        self.mock_db.fetch_query.assert_called_once()


    def test_generate_vehicle_utilization_report(self):
        self.mock_db.fetch_query.return_value = [
            (1, 5), (2, 3)
        ]
        self.service.generate_vehicle_utilization_report()
        self.mock_db.fetch_query.assert_called_once()


    def test_generate_revenue_report(self):
        self.mock_db.fetch_query.return_value = [
            (1, 10000.00), (2, 8000.00)
        ]
        self.service.generate_revenue_report()
        self.mock_db.fetch_query.assert_called_once()



if __name__ == "__main__":
    unittest.main()
```

4. **Vehicle Testing**

```
import unittest
```

```
from unittest.mock import MagicMock

from CarConnect.entity.vehicle import Vehicle

from CarConnect.dao.vehicle_service import VehicleService

from CarConnect.exceptions.vehicle_not_found_exception import
VehicleNotFoundException



class TestVehicleService(unittest.TestCase):

   def setUp(self):

       self.mock_db = MagicMock()

       self.mock_db.fetch_query.return_value = [

           (1, "Tesla", "Model S", 2023, "Black", "TS1234", 1, 3500.50,
1),

           (2, "Toyota", "Camry", 2022, "White", "TN9876", 1, 2500.00,
1)

       ]

       self.service = VehicleService(self.mock_db)



   def test_add_vehicle(self):

       vehicle = Vehicle(1, "Tesla", "Model S", 2023, "Black", "TS1234",
1, 3500.50)

       try:

           self.service.add_vehicle(vehicle)

           print("Vehicle added successfully.")

       except Exception as e:

           self.fail(f"Vehicle addition failed: {e}")



   def test_update_vehicle(self):

       try:

           self.service.update_vehicle(1, 4000.00, 0)

           print("Vehicle updated successfully.")
```

```
        except Exception as e:
            self.fail(f"Vehicle update failed: {e}")

    def test_get_available_vehicles(self):
        try:
            vehicles = self.service.get_available_vehicles()
            self.assertIsInstance(vehicles, list)
            print("Available vehicles fetched successfully.")
        except VehicleNotFoundException:
            print("No available vehicles found.")
        except Exception as e:
            self.fail(f"Fetching available vehicles failed: {e}")

if __name__ == "__main__":
    unittest.main()
```

Util:

Db_conn_util.py:

```
import mysql.connector

class DBConnUtil:
    def __init__(self, host="localhost", user="root", password="root",
database="CarConnect"):
        self.conn = mysql.connector.connect(host=host, user=user,
password=password, database=database)
        self.cursor = self.conn.cursor()

    def execute_query(self, query, values=None):
        try:
```

```
            self.cursor.execute(query, values) if values else
self.cursor.execute(query)

            self.conn.commit()

            print("Successful!!!")

        except mysql.connector.Error as e:

            print(f"Error executing query: {e}")



    def fetch_query(self, query, values=None):

        try:

            self.cursor.execute(query, values) if values else
self.cursor.execute(query)

            result = self.cursor.fetchall()

            if result:

                print("Data retrieved successfully!")

            else:

                print("No records found.")

            return result

        except mysql.connector.Error as e:

            print(f"Error fetching data: {e}")

            return []



    def close_connection(self):

        self.cursor.close()

        self.conn.close()
```

main:

```
from CarConnect.dao.admin_service import AdminService

from CarConnect.dao.customer_service import CustomerService
```

```
from CarConnect.dao.vehicle_service import VehicleService

from CarConnect.dao.reservation_service import ReservationService

from CarConnect.entity.admin import Admin

from CarConnect.entity.customer import Customer

from CarConnect.entity.vehicle import Vehicle

from CarConnect.entity.reservation import Reservation

from CarConnect.exceptions import DatabaseConnectionException

from CarConnect.util.db_conn_util import DBConnUtil

from CarConnect.exceptions.admin_not_found_exception import
AdminNotFoundException

from CarConnect.exceptions.invalid_input_exception import
InvalidInputException

from CarConnect.exceptions.authentication_exception import
AuthenticationException

from CarConnect.exceptions.vehicle_not_found_exception import
VehicleNotFoundException

from CarConnect.exceptions.reservation_exception import ReservationException

from CarConnect.exceptions.customer_not_found_exception import
CustomerNotFoundException



db = DBConnUtil()

admin_service = AdminService(db)

customer_service = CustomerService(db)

vehicle_service = VehicleService(db)

reservation_service = ReservationService(db)



def admin_menu():

   print("\n--- Admin Menu ---")

   print("1. Register Admin")

   print("2. Get Admin by ID")
```

```
    print("3. Get Admin by Username")
    print("4. Update Admin")
    print("5. Delete Admin")

    choice = input("Enter choice: ")

    if choice == '1':
        try:
            first = input("First name: ")
            last = input("Last name: ")
            email = input("Email: ")
            phone = input("Phone: ")
            username = input("Username: ")
            password = input("Password: ")
            role = input("Role('super admin', 'fleet manager'): ")

            admin = Admin(None, first, last, email, phone, username, password,
role, None)
            admin_service.register_admin(admin)
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except DatabaseConnectionException as e:
            print(f"Database Error: {e}")

    elif choice == '2':
        try:
            admin_id = input("Admin ID: ")
            admin = admin_service.get_admin_by_id(admin_id)
```

```
            print(admin)
        except AdminNotFoundException as e:
            print(e)
        except DatabaseConnectionException as e:
            print(e)
        except InvalidInputException as e:
            print(e)

    elif choice == '3':
        try:
            username = input("Enter Username: ")
            admin = admin_service.get_admin_by_username(username)
            print(admin)
        except AdminNotFoundException as e:
            print(e)
        except DatabaseConnectionException as e:
            print(e)
        except InvalidInputException as e:
            print(e)

    elif choice == '4':
        try:
            admin_id = input("Admin ID: ")
            first_name = input("Enter first name: ")
            last_name = input("Enter last name: ")
            email = input("New Email: ")
            phone = input("New Phone: ")
            username = input("New Username: ")
```

```
            role = input("Role('super admin', 'fleet manager'): ")
            admin_service.update_admin(admin_id,first_name,last_name ,email,
phone,username, role)
            print("Admin updated.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except DatabaseConnectionException as e:
            print(f"Registration Failed: {e}")

    elif choice == '5':
        try:
            admin_id = input("Admin ID: ")
            admin_service.delete_admin(admin_id)
        except AdminNotFoundException as e:
            print(f"Admin Not Found: {e}")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except DatabaseConnectionException as e:
            print(f"Registration Failed: {e}")

def customer_menu():
    print("\n--- Customer Menu ---")
    print("1. Register Customer")
    print("2. Get Customer by ID")
    print("3. Get Customer by Username")
    print("4. Update Customer")
    print("5. Delete Customer")
    print("6. Authenticate Customer")
```

```
    choice = input("Enter choice: ")


    if choice == '1':
        try:
            first = input("First name: ")
            last = input("Last name: ")
            email = input("Email: ")
            phone = input("Phone: ")
            address = input("Address: ")
            username = input("Username: ")
            password = input("Password: ")


            customer = Customer(None, first, last, email, phone, address,
username, password, None)
            customer_service.register_customer(customer)
            print("Customer registered.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")


    elif choice == '2':
        try:
            customer_id = input("Customer ID: ")
            customer = customer_service.get_customer_by_id(customer_id)
            print(customer)
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except CustomerNotFoundException as e:
```

```
            print(f"Customer Error: {e}")


    elif choice == '3':
        try:
            username = input("Enter Username: ")
            customer = customer_service.get_customer_by_username(username)
            print(customer)
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except CustomerNotFoundException as e:
            print(f"Customer Error: {e}")


    elif choice == '4':
        try:
            customer_id = input("Customer ID: ")
            first_name = input("First Name: ")
            last_name = input("Last Name: ")
            email = input("New Email: ")
            phone = input("New Phone: ")
            address = input("New Address: ")
            username = input("Username: ")
            customer_service.update_customer(customer_id,first_name,last_name,
email, phone, address,username)
            print("Customer updated.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except CustomerNotFoundException as e:
            print(f"Customer Error: {e}")
```

```
    elif choice == '5':
        try:
            customer_id = input("Customer ID: ")
            customer_service.delete_customer(customer_id)
            print("Customer deleted.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except CustomerNotFoundException as e:
            print(f"Customer Error: {e}")

    elif choice == '6':
        try:
            username = input("Username: ")
            password = input("Password: ")
            customer = customer_service.authenticate_customer(username,
password)
            print(customer)
            print("Authentication successful!")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except AuthenticationException as e:
            print(f"Error:{e}")

def vehicle_menu():
    print("\n--- Vehicle Menu ---")
    print("1. Add Vehicle")
```

```
    print("2. Get Vehicle by ID")
    print("3. List Available Vehicles")
    print("4. Update Vehicle")
    print("5. Remove Vehicle")

    choice = input("Enter choice: ")

    if choice == '1':
        try:
            model = input("Model: ")
            make = input("Make: ")
            year = input("Year: ")
            color = input("Color: ")
            reg_no = input("Registration Number: ")
            availability = input("Availability (1/0): ")
            daily_rate = input("Daily Rate: ")

            vehicle = Vehicle(None, model, make, year, color, reg_no,
availability, daily_rate)
            vehicle_service.add_vehicle(vehicle)
            print("Vehicle added.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except DatabaseConnectionException as e:
            print(f"Registration Failed: {e}")

    elif choice == '2':
        try:
```

```
            vehicle_id = input("Vehicle ID: ")
            if not vehicle_id.isdigit():
                raise InvalidInputException("Vehicle Id must be Integer")
            vehicle = vehicle_service.get_vehicle_by_id(vehicle_id)
            print(vehicle)
        except VehicleNotFoundException as e:
            print(f"Not Found: {e}")
        except DatabaseConnectionException as e:
            print(f"Database Error: {e}")

    elif choice == '3':
        try:
            vehicles = vehicle_service.get_available_vehicles()
            for v in vehicles:
                print(v)
        except DatabaseConnectionException as e:
            print(f"Database Error: {e}")

    elif choice == '4':
        try:
            vehicle_id = input("Vehicle ID: ")
            rate = input("New Daily Rate: ")
            availability = input("Availability (1/0): ")
            vehicle_service.update_vehicle(vehicle_id, rate, availability)
            print("Vehicle updated.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except VehicleNotFoundException as e:
```

```
            print(f"Not Found: {e}")
        except DatabaseConnectionException as e:
            print(f"Database Error: {e}")

    elif choice == '5':
        try:
            vehicle_id = input("Vehicle ID: ")
            vehicle_service.remove_vehicle(vehicle_id)
            print("Vehicle removed.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except VehicleNotFoundException as e:
            print(f"Not Found: {e}")
        except DatabaseConnectionException as e:
            print(f"Database Error: {e}")

def reservation_menu():
    print("\n--- Reservation Menu ---")
    print("1. Create Reservation")
    print("2. Get Reservation by ID")
    print("3. Get Reservations by Customer ID")
    print("4. Update Reservation Status")
    print("5. Cancel Reservation")
    print("6. Reservation History Report")
    print("7. Generate Vehicle Report")
    print("8. Generate Revenue Report")

    choice = input("Enter choice: ")
```

```
   if choice == '1':

       try:

           customer_id = input("Customer ID: ")

           vehicle_id = input("Vehicle ID: ")

           start_date = input("Start Date (YYYY-MM-DD): ")

           end_date = input("End Date (YYYY-MM-DD): ")

           total_cost = input("Total Cost: ")

           status = input("Status('pending', 'confirmed', 'completed',
'cancelled'): ")



           reservation = Reservation(None, customer_id, vehicle_id,
start_date, end_date, total_cost, status)

           reservation_service.create_reservation(reservation)

           print("Reservation created.")

       except InvalidInputException as e:

           print(f"Input Error: {e}")



   elif choice == '2':

       try:

           reservation_id = input("Reservation ID: ")

           reservation =
reservation_service.get_reservation_by_id(reservation_id)

           print(reservation)

       except InvalidInputException as e:

           print(f"Input Error: {e}")

       except ReservationException as e:

           print(f"Error: {e}")
```

```
    elif choice == '3':
        try:
            customer_id = input("Customer ID: ")
            reservations = reservation_service.get_reservations_by_customer_id(customer_id)
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except ReservationException as e:
            print(f"Error: {e}")

    elif choice == '4':
        try:
            reservation_id = input("Reservation ID: ")
            status = input("New Status('pending', 'confirmed', 'completed', 'cancelled'): ")
            reservation_service.update_reservation(reservation_id, status)
            print("Reservation updated.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except ReservationException as e:
            print(f"Error: {e}")

    elif choice == '5':
        try:
            reservation_id = input("Reservation ID: ")
            reservation_service.cancel_reservation(reservation_id)
            print("Reservation canceled.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
```

```
        except ReservationException as e:
            print(f"Error: {e}")

    elif choice == '6':
        reservation_service.generate_reservation_history_report()

    elif choice == '7':
        reservation_service.generate_vehicle_utilization_report()

    elif choice == '8':
        reservation_service.generate_revenue_report()

def main():
    while True:
        print("\n===== CarConnect Main Menu =====")
        print("1. Admin Services")
        print("2. Customer Services")
        print("3. Vehicle Services")
        print("4. Reservation Services")
        print("0. Exit")

        option = input("Select option: ")

        if option == '1':
            admin_menu()
        elif option == '2':
            customer_menu()
        elif option == '3':
```

```
            vehicle_menu()
        elif option == '4':
            reservation_menu()
        elif option == '0':
            print("Exiting CarConnect...")
            break
        else:
            print("Invalid option. Try again.")

if __name__ == "__main__":
    main()
```

Output:

1. **Register Admin**

```
1. Admin Services
2. Customer Services
3. Vehicle Services
4. Reservation Services
0. Exit
Select option: 1

--- Admin Menu ---
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
Enter choice: 1
First name: Meenakshi
Last name: M
Email: meenakshiapril25@gmail.com
Phone: 9791092131
Username: meens
Password: meens
Role('super admin', 'fleet manager'): super admin
Successful!!!
Admin registered successfully.
```

11	Meenakshi	M	meenakshiapril25@gmail.com	9791092131	meens	meens	super admin	2025-04-09 22:22:33

2. Get Admin By ID:

```
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
Enter choice: 2
Admin ID: 11
Data retrieved successfully!
The Admin is: [(11, 'Meenakshi ', 'M', 'meenakshiapril25@gmail.com', '9791092131', 'meens', 'meens', 'super admin', datetime.datetime(2025, 4, 9, 22, 22, 33))]
```

3. Get Admin By Username:

```
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
Enter choice: 3
Enter Username: meens
Data retrieved successfully!
The user is: [(11, 'Meenakshi ', 'M', 'meenakshiapril25@gmail.com', '9791092131', 'meens', 'meens', 'super admin', datetime.datetime(2025, 4, 9, 22, 22, 33))]
```

4. Update Admin:

```
--- Admin Menu ---
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
Enter choice: 4
Admin ID: 11
Enter first name: Meenakshi
Enter last name: M
New Email: meenakshi@gmail.com
New Phone: 9791092131
New Username: meens
Role('super admin', 'fleet manager'): super admin
Successful!!!
Admin updated.
```

11	Meenakshi	M	meenakshi@gmail.com	9791092131	meens	meens	super admin	2025-04-09 22:22:33

5. **Register Customer**

```
===== CarConnect Main Menu =====
1. Admin Services
2. Customer Services
3. Vehicle Services
4. Reservation Services
0. Exit
Select option: 2

--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 1
First name: Harry
Last name: Potter
Email: harry@gmail.com
Phone: 1266326376
Address: Hogwards
Username: harr
Password: harr
Successful!!!
Customer registered.
```

11	Harry	Potter	harry@gmail.com	1266326376	Hogwards	harr	harr	2025-04-10 11:26:22

6. **Get Customer By ID:**

```
--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 2
Customer ID: 11
Data retrieved successfully!
The Customer:  [(11, 'Harry ', 'Potter ', 'harry@gmail.com', '1266326376', 'Hogwards', 'harr', 'harr', datetime.datetime(2025, 4, 10, 11, 26, 22))]
```

7. Get Customer By Username:

```
--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 3
Enter Username: harr
Data retrieved successfully!
The Customer by ID:  [(11, 'Harry ', 'Potter ', 'harry@gmail.com', '1266326376', 'Hogwards', 'harr', 'harr', datetime.datetime(2025, 4, 10, 11, 26, 22))]
```

8. Update Customer:

```
--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 4
Customer ID: 11
First Name: Harry
Last Name: Potter
New Email: harr.pot@gmail.com
New Phone: 5621561256
New Address: hogward
Username: harr
Successful!!!
Customer updated.
```

11	Harry	Potter	harr.pot@gmail.com	5621561256	hogward	harr	harr	2025-04-10 11:26:22

9. Authenticate Customer

```
--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 6
Username: harr
Password: harr
Data retrieved successfully!
The User is: [(11, 'Harry', 'Potter', 'harr.pot@gmail.com', '5621561256', 'hogward', 'harr', 'harr', datetime.datetime(2025, 4, 10, 11, 26, 22))]
None
Authentication successful!
```

10. Add Vehicle:

```
--- Vehicle Menu ---
1. Add Vehicle
2. Get Vehicle by ID
3. List Available Vehicles
4. Update Vehicle
5. Remove Vehicle
Enter choice: 1
Model: Porsh
Make: Porsh
Year: 2023
Color: Pink
Registration Number: wqg134154q
Availability (1/0): 1
Daily Rate: 2000
Successful!!!
Vehicle added.
```

11	Porsh	Porsh	2023	Pink	wqg134154q	1	2000.00

11. Get Vehicle My ID

```
--- Vehicle Menu ---
1. Add Vehicle
2. Get Vehicle by ID
3. List Available Vehicles
4. Update Vehicle
5. Remove Vehicle
Enter choice: 2
Vehicle ID: 11
Data retrieved successfully!
The vehicle is:  [(11, 'Porsh ', 'Porsh', 2023, 'Pink', 'wqg134154q', 1, Decimal('2000.00'))]
```

12. List Of available vehicles

```
--- Vehicle Menu ---
1. Add Vehicle
2. Get Vehicle by ID
3. List Available Vehicles
4. Update Vehicle
5. Remove Vehicle
Enter choice: 3
Data retrieved successfully!
(1, 'Swift', 'Maruti', 2021, 'Red', 'KA01AB1234', 1, Decimal('1200.00'))
(2, 'City', 'Honda', 2020, 'Black', 'TN02BC5678', 1, Decimal('1500.00'))
(3, 'Innova', 'Toyota', 2019, 'Silver', 'MH03CD9101', 1, Decimal('2000.00'))
(4, 'i20', 'Hyundai', 2022, 'White', 'DL04EF1122', 1, Decimal('1300.00'))
(5, 'Creta', 'Hyundai', 2021, 'Grey', 'KL05GH3344', 1, Decimal('1800.00'))
(6, 'Ertiga', 'Maruti', 2020, 'Blue', 'KA06IJ5566', 1, Decimal('1700.00'))
(7, 'Fortuner', 'Toyota', 2023, 'Black', 'TN07KL7788', 1, Decimal('2500.00'))
(8, 'Baleno', 'Maruti', 2021, 'Red', 'MH08MN9900', 1, Decimal('1400.00'))
(9, 'Venue', 'Hyundai', 2022, 'White', 'DL09OP1112', 1, Decimal('1600.00'))
(10, 'Altroz', 'Tata', 2020, 'Yellow', 'KL10QR1314', 1, Decimal('1100.00'))
(11, 'Porsh ', 'Porsh', 2023, 'Pink', 'wqg134154q', 1, Decimal('2000.00'))
```

13. Update Vehicle

```
--- Vehicle Menu ---
1. Add Vehicle
2. Get Vehicle by ID
3. List Available Vehicles
4. Update Vehicle
5. Remove Vehicle
Enter choice: 4
Vehicle ID: 11
New Daily Rate: 2000
Availability (1/0): 0
Successful!!!
Vehicle updated.
```

11	Porsh	Porsh	2023	Pink	wqg134154q	0	2000.00

14. Create Reservation

```
--- Reservation Menu ---
1. Create Reservation
2. Get Reservation by ID
3. Get Reservations by Customer ID
4. Update Reservation Status
5. Cancel Reservation
6. Reservation History Report
7. Generate Vehicle Report
8. Generate Revenue Report
Enter choice: 1
Customer ID: 11
Vehicle ID: 11
Start Date (YYYY-MM-DD): 2025-04-10
End Date (YYYY-MM-DD): 2025-04-11
Total Cost: 4000
Status: pending
Successful!!!
Reservation created.
```

11	11	11	2025-04-10 00:00:00	2025-04-11 00:00:00	4000.00	pending

15. Get Reservation By ID

```
--- Reservation Menu ---
1. Create Reservation
2. Get Reservation by ID
3. Get Reservations by Customer ID
4. Update Reservation Status
5. Cancel Reservation
6. Reservation History Report
7. Generate Vehicle Report
8. Generate Revenue Report
Enter choice: 2
Reservation ID: 11
Data retrieved successfully!
[(11, 11, 11, datetime.datetime(2025, 4, 10, 0, 0), datetime.datetime(2025, 4, 11, 0, 0), Decimal('4000.00'), 'pending')]
```

16. Get reservation By customer ID

```
--- Reservation Menu ---
1. Create Reservation
2. Get Reservation by ID
3. Get Reservations by Customer ID
4. Update Reservation Status
5. Cancel Reservation
6. Reservation History Report
7. Generate Vehicle Report
8. Generate Revenue Report
Enter choice: 3
Customer ID: 11
Data retrieved successfully!
Reservation ID: 11
Vehicle ID   : 11
Start Date   : 2025-04-10 00:00:00
End Date     : 2025-04-11 00:00:00
Total Cost   : 4000.00
Status       : pending
```

17. Update reservation Status

```
Enter choice: 4
Reservation ID: 11
New Status: completed
Successful!!!
Reservation updated.
```

11	11	11	2025-04-10 00:00:00	2025-04-11 00:00:00	4000.00	completed

18. Reservation History Report

```
--- Reservation History Report ---
(11, 11, 11, datetime.datetime(2025, 4, 10, 0, 0), datetime.datetime(2025, 4, 11, 0, 0), 'completed')
(8, 8, 9, datetime.datetime(2025, 3, 14, 13, 0), datetime.datetime(2025, 3, 16, 13, 0), 'pending')
(7, 7, 5, datetime.datetime(2025, 3, 12, 11, 0), datetime.datetime(2025, 3, 13, 11, 0), 'pending')
(4, 4, 6, datetime.datetime(2025, 3, 11, 8, 0), datetime.datetime(2025, 3, 14, 8, 0), 'cancelled')
(6, 6, 7, datetime.datetime(2025, 3, 8, 10, 0), datetime.datetime(2025, 3, 9, 10, 0), 'confirmed')
(3, 3, 1, datetime.datetime(2025, 3, 7, 12, 0), datetime.datetime(2025, 3, 10, 12, 0), 'confirmed')
(5, 5, 3, datetime.datetime(2025, 3, 5, 18, 0), datetime.datetime(2025, 3, 6, 18, 0), 'completed')
(2, 2, 4, datetime.datetime(2025, 3, 3, 9, 0), datetime.datetime(2025, 3, 4, 9, 0), 'completed')
(10, 10, 10, datetime.datetime(2025, 3, 2, 9, 0), datetime.datetime(2025, 3, 4, 9, 0), 'confirmed')
(1, 1, 2, datetime.datetime(2025, 3, 1, 10, 0), datetime.datetime(2025, 3, 5, 10, 0), 'completed')
(9, 9, 8, datetime.datetime(2025, 3, 1, 10, 0), datetime.datetime(2025, 3, 2, 10, 0), 'completed')
```

19. Generate Vehicle Report

```
--- Vehicle Utilization Report ---
Vehicle ID: 1, Reservations: 1
Vehicle ID: 2, Reservations: 1
Vehicle ID: 3, Reservations: 1
Vehicle ID: 4, Reservations: 1
Vehicle ID: 5, Reservations: 1
Vehicle ID: 6, Reservations: 1
Vehicle ID: 7, Reservations: 1
Vehicle ID: 8, Reservations: 1
Vehicle ID: 9, Reservations: 1
Vehicle ID: 10, Reservations: 1
Vehicle ID: 11, Reservations: 1
```

20. Generate Revenue Report

```
--- Reservation Menu ---
1. Create Reservation
2. Get Reservation by ID
3. Get Reservations by Customer ID
4. Update Reservation Status
5. Cancel Reservation
6. Reservation History Report
7. Generate Vehicle Report
8. Generate Revenue Report
Enter choice: 8
Data retrieved successfully!

--- Revenue Report ---
Vehicle ID: 2, Revenue: ₹6000.00
Vehicle ID: 11, Revenue: ₹4000.00
Vehicle ID: 3, Revenue: ₹2000.00
Vehicle ID: 8, Revenue: ₹1400.00
Vehicle ID: 4, Revenue: ₹1300.00
```

BUSINESS LOGICS

1. **Customer Module**
    - **Registration:** Validate unique username and email. Save customer details with hashed password.
    - **View Profile:** Retrieve and display customer information by customer ID.
    - **Update Profile:** Allow customer to update personal details. Validate input before updating.
    - **Delete Account:** Confirm deletion and remove customer record. Cancel all future reservations tied to the customer.
2. **Vehicle Module**
    - **Add Vehicle:** Insert new vehicle with unique registration number and availability status.
    - **Update Vehicle:** Modify details like rate, availability, color, etc.
    - **Delete Vehicle:** Remove vehicle from the system; cascade deletes reservations if needed.
    - **View Available:** VehicleFetch only vehicles marked as available.
3. **Admin Module**
    - **Login Authenticate admin credentials:** Differentiate roles (super admin, fleet manager).
    - **Manage Customers:** View, search, delete, or update customer records.
    - **Manage Vehicles:** Full CRUD operations.
    - **Generate Reports:** Access revenue, vehicle, and reservation reports.
4. **Reservation Module**.
    - **Get Reservation by ID:** Fetch reservation details using reservation ID.
    - **Get Reservations by Customer ID:** Fetch all past/current reservations for a customer. Raise error if none found.
    - **Update Reservation Status:** Allow admin to update to confirmed, completed, or cancelled.
    - **Cancel Reservation:** Delete or update reservation status. Ensure cancellation rules apply (e.g., cannot cancel completed).
5. **Reports Module**
    - **Reservation History Report:** Retrieve all reservations (filterable by date, customer, status).
    - **Vehicle Report:** Show vehicles and number of times rented, total revenue generated.
    - **Revenue Report:** Calculate total revenue earned from all completed reservations within a period.

FUTURE SCOPE

The CarConnect Car Rental System has a strong foundation and offers immense potential for future enhancements. As the automotive rental industry evolves with customer-centric digital transformation, CarConnect can be scaled and enriched with advanced features to improve user experience and operational efficiency.

In the future, the system can integrate real-time vehicle tracking using GPS, allowing both customers and admins to monitor vehicle locations and optimize logistics. Adding dynamic pricing algorithms based on demand, location, and time can improve revenue generation. Integration with mobile apps (iOS & Android) will offer on-the-go convenience and broader accessibility. Moreover, incorporating online payment gateways and e-wallet support will enable seamless transactions. The use of AI-based recommendation systems can suggest vehicles based on customer preferences, driving history, or trip type

Future versions could also include loyalty programs, subscription-based rentals, and support for electric vehicles (EVs) with charging station integration. Additionally, expanding to a multi-branch system would allow CarConnect to cater to a wider customer base and grow as a full-fledged national or even global rental platform.

CONCLUSION

The CarConnect Car Rental System is a comprehensive and user-friendly solution designed to streamline the vehicle rental process for both customers and administrators. By integrating a structured database, object-oriented programming, and essential business logic, the system ensures efficient reservation handling, secure authentication, and smooth management of vehicles and users. It provides a digital transformation for traditional rental operations, improving accessibility, transparency, and reliability.

Through features like reservation management, vehicle availability tracking, revenue and history reports, and admin functionalities, CarConnect enhances operational efficiency and customer satisfaction. It demonstrates how technology can be effectively used to solve real-world problems in the transportation and logistics sector.

With a scalable architecture and modular design, the system lays a solid foundation for future enhancements such as mobile app integration, online payments, AI recommendations, and multi-location support. In conclusion, CarConnect is a powerful tool for modernizing car rental businesses, offering both convenience and control, while setting the stage for continuous innovation and expansion.