# HEXAWARE TECHNOLOGY

## CARCONNECT

## A CAR RENTAL PLATFORM

**Python - Batch - 4**

ANUSH KUMAR

## INTRODUCTION

Using Python and SQL Server, CarConnect is a cutting-edge database-driven vehicle rental management system that prioritizes modularity, maintainability, and practicality. By automating crucial tasks including client registration, vehicle management, reservation processing, and administrative controls, the system seeks to optimize a car rental company's operations.

Features including car availability tracking, customer and admin authentication, reservation creation and modifications, and the compilation of reports on vehicle usage, reservation history, and total income are all supported by the system. Through appropriate use of foreign key restrictions and exception handling, CarConnect guarantees dependable data permanence and integrity by utilizing MySQL Server as the backend database.

CarConnect was created with scalability in mind and may be modified for small-to-medium sized vehicle rental companies who want to automate their processes. It is also a great contender for future revisions that integrate with web frameworks or APIs due to its flexible architecture.

CarConnect is more than simply a project; it's a workable answer to the demands of contemporary rental businesses. All things considered, it offers a strong basis for learning enterprise-level application development with an emphasis on real-time problem-solving, data validation, database interface, and user experience.

**PURPOSE OF THE PROJECT**

The goal of the CarConnect project is to provide a complete management system for rental automobile rentals that streamlines and automates the essential functions of a car rental company. With the help of this system, administrators and clients will be able to effectively handle cars, reservations, and user data by replacing manual, old methods with a centralized digital solution. The purpose of CarConnect is to: Make it easy for clients to register, authenticate, and reserve available cars.

- Permit administrators to keep an eye on client reservations, adjust availability, and manage car inventory.
- By using appropriate database interaction, validation, and authentication, you may guarantee safe data processing.
- Using reporting tools and status updates, give insight into operations.
- Create analytical reports to aid in corporate decision-making, such as revenue summaries, vehicle usage, and reservation history.

**SCOPE OF THE PROJECT:**

All of the key functional areas needed to run a vehicle rental business effectively are covered by the CarConnect system. Its scope includes both customer-facing and administrative features, combining a relational database and organized backend for dependable data management.
The project's primary focus areas are as follows:
Administrative Management:

Admins have the ability to create, edit, and remove profiles. They may check reservation records, manage car data, and create reports to track company success.

Customer management: Clients are able to register, log in, see their personal information, and reserve cars. They have the ability to examine and manage booking history.

Vehicle Management: Admins have the ability to examine, edit, add, or delete vehicle information, such as availability and daily rental costs. This guarantees that the inventory will always be correct and up to date.

Reservation System: By choosing dates and figuring out the total cost, the system enables users to book available cars. Reservation statuses can be updated, canceled, or confirmed by administrators.

Reporting Module: For business insights and performance monitoring, administrators may create a variety of reports, such as those on vehicle usage, reservation history, and income.

Database Integration: To guarantee data permanence, relational integrity, and seamless CRUD operations, every action is linked to a strong MySQL database.

**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:**

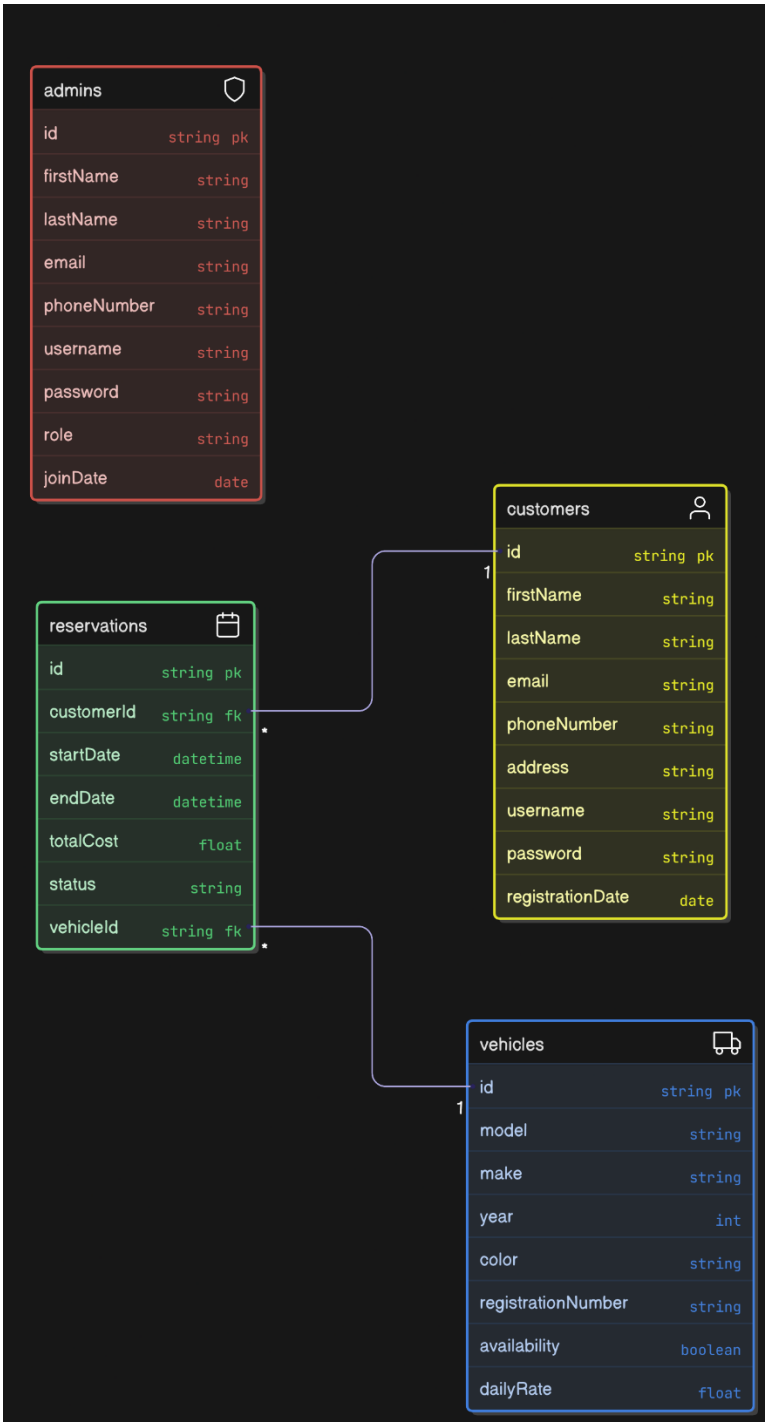
- **ReservationID (Primary Key):** Unique identifier for each reservation.
- **CustomerID (Foreign Key):** Foreign key referencing the Customer table.

- **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).

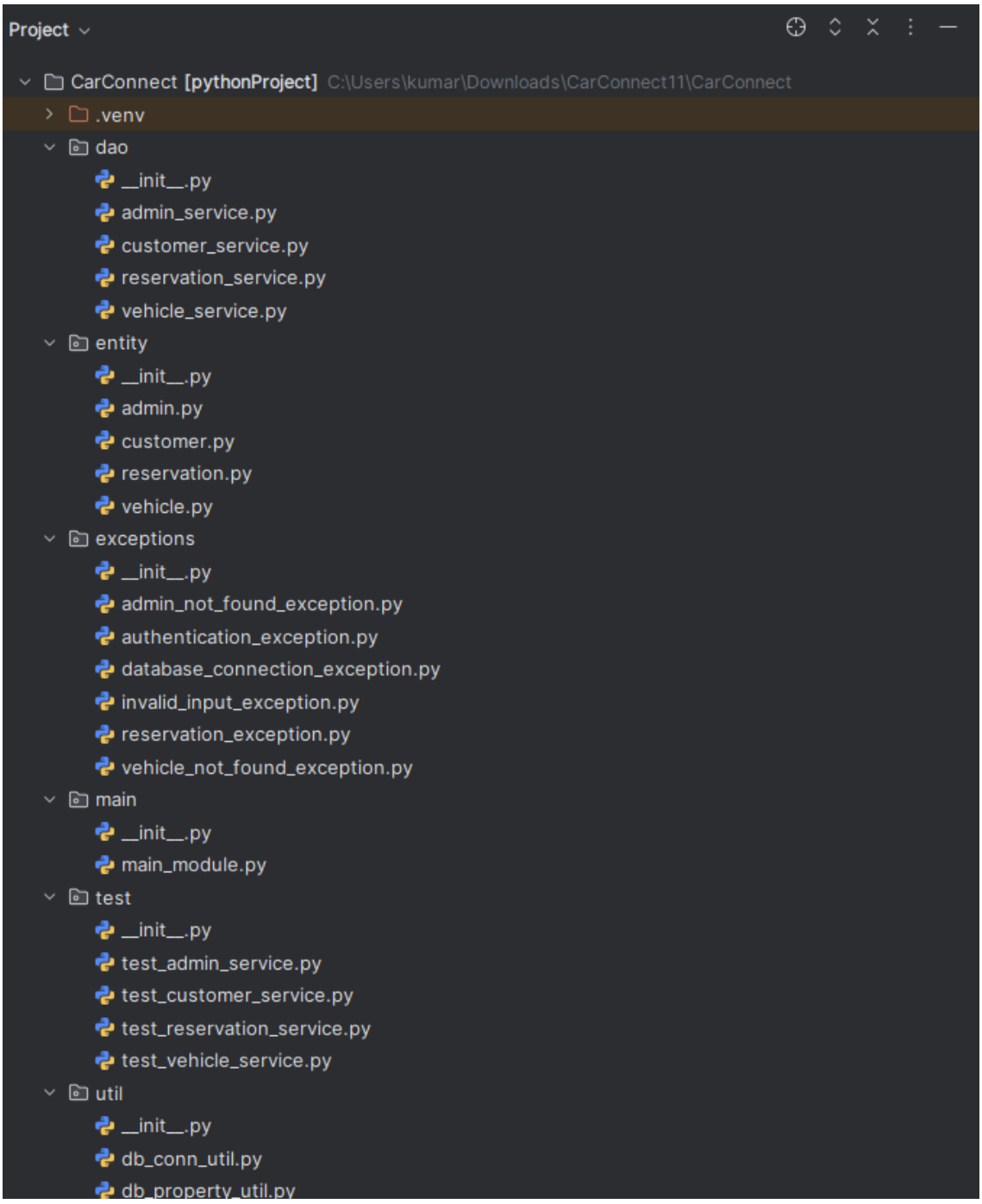
**4. Admin Table:**

- **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 DIAGARM


admins
id string pk
firstName string
lastName string
email string
phoneNumber string
username string
password string
role string
joinDate date
customers
id string pk
firstName string
lastName string
email string
phoneNumber string
address string
username string
password string
registrationDate date
reservations
id string pk
customerId string fk
startDate datetime
endDate datetime
totalCost float
status string
vehicleId string fk
vehicles
id string pk
model string
make string
year int
color string
registrationNumber string
availability boolean
dailyRate float
1
*
1
*

**PYTHON DRECTORY:**



**SQL QUERIES:**

**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') 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');

**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
```

**DAO:**

**Admin_service.py:**

```
from entity.admin import Admin
from exceptions.admin_not_found_exception import AdminNotFoundException
from exceptions.invalid_input_exception import InvalidInputException
from exceptions.database_connection_exception import DatabaseConnectionException
from tabulate import tabulate


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.")

            headers = ["AdminID", "Name", "Email", "Username", "Password", "Role", "JoinDate"]
            print(tabulate([row[0]], headers=headers, tablefmt="fancy_grid"))

        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}")
            headers = ["AdminID", "Name", "Email", "Username", "Password", "Role", "JoinDate"]
            print(tabulate([row[0]], headers=headers, tablefmt="fancy_grid"))
```

```
        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)}")

    def authenticate_admin(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 FirstName, Role FROM Admin WHERE Username = %s AND
Password = %s"
        results = self.db.fetch_query(query, (username, password))

        if results and len(results) > 0:
            first_name, role = results[0]  # get the first row
            print("Successfully logged in!")
            print(f"\nWelcome, {first_name} ({role})")
            return role
        else:
            raise AuthenticationException("Invalid admin credentials.")
```

**customer_service.py:**

```
from entity.customer import Customer
from exceptions.invalid_input_exception import InvalidInputException
from exceptions.authentication_exception import AuthenticationException
from exceptions.customer_not_found_exception import CustomerNotFoundException
from tabulate import tabulate

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.")


        customer_data = result[0]
        display_data = [customer_data[0], customer_data[1], customer_data[3], customer_data[4],
                        customer_data[8]]
        headers = ["CustomerID", "Name", "Email", "Phone", "RegistrationDate"]

        print(tabulate([display_data], headers=headers, tablefmt="fancy_grid"))

    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.")

        customer_data = result[0]
```

```
        display_data = [customer_data[0], customer_data[1], customer_data[3], customer_data[4],
                        customer_data[8]]
        headers = ["CustomerID", "Name", "Email", "Phone", "RegistrationDate"]

        print(tabulate([display_data], headers=headers, tablefmt="fancy_grid"))

    def register_customer(self, customer):
        if not all([customer.first_name.strip(),customer.last_name.strip(),customer.email.strip(),
                    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 CustomerID, FirstName 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.")

        customer_id, first_name = result[0]

        print("Successfully logged in!")
        print(f"Welcome, {first_name} (Customer ID: {customer_id})")
```

**reservation_service.py:**

```
from entity.reservation import Reservation
from exceptions.invalid_input_exception import InvalidInputException
from exceptions.reservation_exception import ReservationException
from datetime import datetime
from tabulate import tabulate


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}")
        headers = ["ReservationID", "CustomerID", "VehicleID", "StartDate", "EndDate",
"TotalCost", "Status"]
        print(tabulate(result, headers=headers, tablefmt="fancy_grid"))

    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}")

        headers = ["ReservationID", "CustomerID", "VehicleID", "StartDate", "EndDate",
"TotalCost", "Status"]
        print(tabulate(result, headers=headers, tablefmt="fancy_grid"))

    def create_reservation(self, reservation):
        if not reservation.customer_id.isdigit() or not reservation.vehicle_id.isdigit():
            raise InvalidInputException("Customer ID and Vehicle ID must be integers.")


        query = "SELECT Availability, DailyRate FROM Vehicle WHERE VehicleID = %s"
        result = self.db.fetch_one(query, (reservation.vehicle_id,))
        if not result:
            raise ReservationException("Vehicle does not exist.")
        if result[0] != 1:
            raise ReservationException("Vehicle is not available for reservation.")

        daily_rate = result[1]
```

```
        start_date = datetime.strptime(reservation.start_date, "%Y-%m-%d")
        end_date = datetime.strptime(reservation.end_date, "%Y-%m-%d")
        number_of_days = (end_date - start_date).days
        if number_of_days <= 0:
            raise InvalidInputException("End date must be after start date.")

        total_cost = number_of_days * daily_rate


        insert_query = """
            INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost,
Status)
            VALUES (%s, %s, %s, %s, %s, %s)
        """
        self.db.execute_query(insert_query, (
            reservation.customer_id, reservation.vehicle_id,
            reservation.start_date, reservation.end_date,
            total_cost, reservation.status
        ))

        self.db.conn.commit()


        reservation_id = self.db.cursor.lastrowid

        print(f"Reservation created successfully with ID: {reservation_id}")
        print(f"Total cost calculated: ₹{total_cost:.2f}")

    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 ---")

        if not results:
            raise ReservationException("No reservations found.")

        headers = ["Reservation ID", "Customer ID", "Vehicle ID", "Start Date", "End Date",
"Status"]
        print(tabulate(results, headers=headers, tablefmt="fancy_grid"))

    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 ---")

        if not results:
            raise ReservationException("No reservation data available.")

        headers = ["Vehicle ID", "Total Reservations"]
        print(tabulate(results, headers=headers, tablefmt="fancy_grid"))
```

```
    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 ---")

        if not results:
            raise ReservationException("No completed reservations found.")


        formatted_results = [(row[0], f"₹{row[1]:.2f}") for row in results]
        headers = ["Vehicle ID", "Revenue"]

        print(tabulate(formatted_results, headers=headers, tablefmt="fancy_grid"))

    def get_pending_reservation(self):
        query = "SELECT * from Reservation WHERE Status = 'Pending'"
        rows = self.db.fetch_query(query)
        if not rows:
            raise ReservationException("No Pending reservations")

        headers = ["ReservationID", "CustomerID", "VehicleID", "StartDate", "EndDate",
"TotalCost", "Status"]
        print(tabulate(rows, headers=headers, tablefmt="fancy_grid"))

    def get_confirmed_reservation(self):
        query = "SELECT * from Reservation WHERE Status = 'Confirmed'"
        rows = self.db.fetch_query(query)
        if not rows:
            raise ReservationException("No Confirmed reservations")

        headers = ["ReservationID", "CustomerID", "VehicleID", "StartDate", "EndDate",
"TotalCost", "Status"]
        print(tabulate(rows, headers=headers, tablefmt="fancy_grid"))
```

**vehicle_service.py:**

```
from entity.vehicle import Vehicle
from exceptions.vehicle_not_found_exception import VehicleNotFoundException
from exceptions.invalid_input_exception import InvalidInputException
from exceptions.database_connection_exception import DatabaseConnectionException
import re
from tabulate import tabulate

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}")

            headers = ["VehicleID", "Model", "Make", "Year", "Color", "RegistrationNumber",
"Availability", "DailyRate"]
            print(tabulate([row[0]], headers=headers, tablefmt="fancy_grid"))

        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)
            if not rows:
                raise VehicleNotFoundException("No available vehicles found.")
            headers = ["VehicleID", "Model", "Make", "Year", "Color", "RegistrationNumber",
"Availability", "DailyRate"]
            print(tabulate(rows, headers=headers, tablefmt="fancy_grid"))
            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]{1}\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:**

**Admin_not_found_exception.py:**

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

```
Admin ID: 400
No records found.
Admin Error: Admin with ID 400 not found.
```

**authentication_exceoption.py:**

```
class AuthenticationException(Exception):
    def __init__(self, message="Invalid username or password."):
        super().__init__(message)
```

```
Enter choice: 2
Username: kamala
Password: 741
No records found.
Login Error: Invalid username or password.
```

**database_connection_exception.py:**

```
class DatabaseConnectionException(Exception):
    def __init__(self, message="Unable to connect to the database."):
        super().__init__(message)
```

**invalid_input_exception.py:**

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

```
Customer ID: anush
Input Error: Customer ID must be an integer.
```

**reservation_exception.py:**

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

```
Reservation Error: No reservation found with ID: 500
```

**vehicle_not_found_exception.py:**

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

```
No records found.
Vehicle Error: No vehicle found with ID: 100
```

**UTIL:**

**db_conn_util.py:**

```
import mysql.connector

class DBConnUtil:
    def __init__(self, host="localhost", user="root", password="root@123",
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("DB Successful! !!")
            return self.cursor.rowcount
        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:
                pass
            else:
                print("No records found.")
            return result
        except mysql.connector.Error as e:
            print(f"Error fetching data: {e}")
            return []

    def fetch_one(self, query, params=None):
        try:
            with self.conn.cursor() as cursor:
                cursor.execute(query, params)
                return cursor.fetchone()
        except Exception as e:
            raise DatabaseConnectionException(f"Database fetch_one failed: {str(e)}")

    def close_connection(self):
        self.cursor.close()
        self.conn.close()
```

**MAIN**

**main.py:**

```
from dao.admin_service import AdminService
from dao.customer_service import CustomerService
from dao.vehicle_service import VehicleService
from dao.reservation_service import ReservationService
from entity.admin import Admin
```

```
from entity.customer import Customer
from entity.vehicle import Vehicle
from entity.reservation import Reservation
from exceptions import DatabaseConnectionException
from util.db_conn_util import DBConnUtil
from exceptions.admin_not_found_exception import AdminNotFoundException
from exceptions.invalid_input_exception import InvalidInputException
from exceptions.authentication_exception import AuthenticationException
from exceptions.vehicle_not_found_exception import VehicleNotFoundException
from exceptions.reservation_exception import ReservationException
from 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 login_menu():
    while True:
        print("\n===== CarConnect Login Menu =====")
        print("1. Customer Sign Up")
        print("2. Customer Login")
        print("3. Admin Login")
        print("0. Exit")

        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 successfully!")
        except InvalidInputException as e:
            print(f"Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '2':
        try:
            username = input("Username: ")
            password = input("Password: ")
            customer = customer_service.authenticate_customer(username, password)
            customer_logged_in_menu(customer)
        except (InvalidInputException, AuthenticationException) as e:
            print(f"Login Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '3':
        try:
            username = input("Username: ")
            password = input("Password: ")
            admin = admin_service.authenticate_admin(username, password)
            if admin == 'super admin':
                super_admin_menu(admin)
            elif admin == 'fleet manager':
                fleet_admin_menu(admin)
        except (InvalidInputException, AuthenticationException) as e:
            print(f"Admin Login Error: {e}")

    elif choice == '0':
        print("Exiting CarConnect...")
        break

    else:
        print("Invalid option. Try again.")

def customer_logged_in_menu(customer):
    while True:
        print("\n--- Customer Dashboard ---")
        print("1. Update Profile")
```

```
        print("2. Check Customer Details")
        print("3. Create Reservation")
        print("4. Get Reservation by ID")
        print("5. Delete Account")
        print("6. Cancel Reservation")  # <-- NEW OPTION
        print("0. Logout")

        choice = input("Enter choice: ")

        if choice == '1':
            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 == '2':
            try:
                print(customer_service.get_customer_by_id(input("Customer ID: ")))
            except CustomerNotFoundException as e:
                print(f"Customer Error: {e}")
            except InvalidInputException as e:
                print(f"Input Error: {e}")

        elif choice == '3':
            try:
                vehicle = vehicle_service.get_available_vehicles()
                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): ")

            reservation = Reservation(None, customer_id, vehicle_id, start_date, end_date,
total_cost=0,
                                      status="pending")
            reservation_service.create_reservation(reservation)
            print("Reservation created.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")

    elif choice == '4':
        try:
            reservation_id = input("Reservation ID: ")
            reservation = reservation_service.get_reservation_by_id(reservation_id)
            print(reservation)
        except ReservationException as e:
            print(f"Error: {e}")
        except Exception as e:
            print(f"Unexpected 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:
            reservation_id = input("Reservation ID to cancel: ")
            reservation_service.cancel_reservation(reservation_id)
            print("Reservation cancelled.")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except ReservationException as e:
            print(f"Reservation Error: {e}")
```

```
        elif choice == '0':
            print("Logging out...")
            break
        else:
            print("Invalid choice.")

def super_admin_menu(admin):
    while True:
        print("\n--- Super Admin Dashboard ---")
        print("1. Register Admin")
        print("2. Get Admin by ID")
        print("3. Get Admin by Username")
        print("4. Update Admin")
        print("5. Delete Admin")
        print("6. Get Customer by ID")
        print("7. Get Customer by Username")
        print("8. Delete Customer")
        print("9. Add Vehicle")
        print("10. Get Vehicle by ID")
        print("11. Get Available Vehicles")
        print("12. Update Vehicle")
        print("13. Delete Vehicle")
        print("14. Get Reservation by ID")
        print("15. Get Reservation by Customer ID")
        print("16. Update Reservation")
        print("17. Cancel Reservation")
        print("18. Generate Reservation History Report")
        print("19. Generate Vehicle Utilization Report")
        print("20. Generate Revenue Report")
        print("0. Logout")

        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: ")
            print(admin_service.get_admin_by_id(admin_id))
        except AdminNotFoundException as e:
            print(f"Admin Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '3':
        try:
            uname = input("Username: ")
            print(admin_service.get_admin_by_username(uname))
        except AdminNotFoundException as e:
            print(f"Admin Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {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 to delete: ")
            admin_service.delete_admin(admin_id)
        except AdminNotFoundException as e:
            print(f"Admin Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '6':
        try:
            cid = input("Customer ID: ")
            print(customer_service.get_customer_by_id(cid))
        except CustomerNotFoundException as e:
            print(f"Customer Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '7':
        try:
            uname = input("Username: ")
            print(customer_service.get_customer_by_username(uname))
        except CustomerNotFoundException as e:
            print(f"Customer Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '8':
        try:
            cid = input("Customer ID to delete: ")
            customer_service.delete_customer(cid)
            print("Customer deleted.")
        except CustomerNotFoundException as e:
            print(f"Customer Error: {e}")
```

```
        except Exception as e:
            print(f"Unexpected error: {e}")

    if choice == '9':
        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 == '10':
        try:
            vid = input("Vehicle ID: ")
            print(vehicle_service.get_vehicle_by_id(vid))
        except VehicleNotFoundException as e:
            print(f"Vehicle Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '11':
        try:
            print(vehicle_service.get_available_vehicles())
        except VehicleNotFoundException as e:
            print(f"Error: {e}")

    elif choice == '12':
        try:
            vid = input("Vehicle ID to update: ")
            rate = input("New daily rate: ")
            availability = input("Availability (1 or 0): ")
```

```
            vehicle_service.update_vehicle(vid, rate, availability)
        except VehicleNotFoundException as e:
            print(f"Vehicle Error: {e}")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '13':
        try:
            vid = input("Vehicle ID to delete: ")
            vehicle_service.remove_vehicle(vid)
        except VehicleNotFoundException as e:
            print(f"Vehicle Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '14':
        try:
            rid = input("Reservation ID: ")
            print(reservation_service.get_reservation_by_id(rid))
        except ReservationException as e:
            print(f"Reservation Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '15':
        try:
            cid = input("Customer ID: ")
            print(reservation_service.get_reservations_by_customer_id(cid))
        except ReservationException as e:
            print(f"Reservation Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '16':
        try:
            rid = input("Reservation ID: ")
            status = input("New status (pending/confirmed/completed): ")
            reservation_service.update_reservation(rid, status)
```

```
        except ReservationException as e:
            print(f"Reservation Error: {e}")
        except InvalidInputException as e:
            print(f"Input Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '17':
        try:
            rid = input("Reservation ID to cancel: ")
            reservation_service.cancel_reservation(rid)
            print("Reservation cancelled.")
        except ReservationException as e:
            print(f"Reservation Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '18':
        try:
            reservation_service.generate_reservation_history_report()
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '19':
        try:
            reservation_service.generate_vehicle_utilization_report()
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '20':
        try:
            reservation_service.generate_revenue_report()
        except Exception as e:
            print(f"Unexpected error: {e}")

    elif choice == '0':
        break
    else:
        print("Invalid choice.")
```

```
def fleet_admin_menu(admin):
    while True:
        print("\n--- Fleet Admin Dashboard ---")
        print("1. Get Admin by ID")
        print("2. Get Admin by Username")
        print("3. Get Customer by ID")
        print("4. Get Customer by Username")
        print("5. Delete Customer")
        print("6. List Available Vehicles")
        print("7. Update Vehicle")
        print("8. Show Confirmed Reservations")
        print("9. Show Pending Reservations")
        print("10. Update Reservation Status")
        print("11. Get Reservation by Customer ID")
        print("0. Logout")

        choice = input("Enter choice: ")

        if choice == '1':
            try:
                print(admin_service.get_admin_by_id(input("Admin ID: ")))
            except AdminNotFoundException as e:
                print(f"Admin Error: {e}")

        elif choice == '2':
            try:
                print(admin_service.get_admin_by_username(input("Username: ")))
            except AdminNotFoundException as e:
                print(f"Admin Error: {e}")

        elif choice == '3':
            try:
                print(customer_service.get_customer_by_id(input("Customer ID: ")))
            except CustomerNotFoundException as e:
                print(f"Customer Error: {e}")
            except InvalidInputException as e:
                print(f"Input Error: {e}")

        elif choice == '4':
            try:
```

```
            print(customer_service.get_customer_by_username(input("Username: ")))
        except CustomerNotFoundException as e:
            print(f"Customer Error: {e}")
        except InvalidInputException as e:
            print(f"Input Error: {e}")

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

    elif choice == '6':
        try:
            available_vehicles = vehicle_service.get_available_vehicles()
        except VehicleNotFoundException as e:
            print(f"Vehicle Error: {e}")

    elif choice == '7':
        try:
            vehicle_id = input("Vehicle ID: ")
            daily_rate = input("New Daily Rate: ")
            availability = input("Availability (1/0): ")
            vehicle_service.update_vehicle(vehicle_id, daily_rate, availability)
            print("Vehicle updated successfully.")
        except VehicleNotFoundException as e:
            print(f"Vehicle Error: {e}")
        except InvalidInputException as e:
            print(f"Input Error: {e}")

    elif choice == '8':
        try:
            confirmed = reservation_service.get_confirmed_reservation()
        except ReservationException as e:
            print(f"ReservationError:{e}")

    elif choice == '9':
```

```
            try:
                pending = reservation_service.get_pending_reservation()
            except ReservationException as e:
                print(f"Reservation Error: {e}")

        elif choice == '10':
            try:
                reservation_id = input("Reservation ID: ")
                status = input("New Status (pending/confirmed/completed): ")
                reservation_service.update_reservation(reservation_id, status)
                print("Reservation status updated.")
            except ReservationException as e:
                print(f"Reservation Error: {e}")
            except InvalidInputException as e:
                print(f"Input Error: {e}")

        elif choice == '11':
            try:
                customer_id = input("Customer ID: ")
                reservations = reservation_service.get_reservations_by_customer_id(customer_id)
            except ReservationException as e:
                print(f"Reservation Error: {e}")
            except InvalidInputException as e:
                print(f"Input Error: {e}")

        elif choice == '0':
            break
        else:
            print("Invalid choice.")

if __name__ == '__main__':
    login_menu()
```

**TESTING:**

**test_admin_service.py:**

```
import unittest
from unittest.mock import MagicMock
from datetime import date
from entity.reservation import Reservation
from dao.reservation_service import ReservationService
from exceptions.invalid_input_exception import InvalidInputException
from 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", "2024-05-01", "2024-05-05", "0.0", "Confirmed"
        )

        self.mock_db.fetch_one.return_value = (1, 1000.0)
        self.service.create_reservation(reservation)
        self.mock_db.fetch_one.assert_called_once_with(
            "SELECT Availability, DailyRate FROM Vehicle WHERE VehicleID = %s", ("3",)
        )
        self.mock_db.execute_query.assert_called_once_with(
            """
            INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost,
Status)
            VALUES (%s, %s, %s, %s, %s, %s)
        """,
            ("2", "3", "2024-05-01", "2024-05-05", 4000.0, "Confirmed")
        )

    def test_create_reservation_invalid_customer_vehicle_id(self):
        reservation = Reservation(
            None, "abc", "3", "2024-05-01", "2024-05-05", "0.0", "Confirmed"
        )
        with self.assertRaises(InvalidInputException) as context:
            self.service.create_reservation(reservation)
        self.assertEqual(str(context.exception), "Customer ID and Vehicle ID must be integers.")
```

```
def test_create_reservation_vehicle_not_found(self):
    reservation = Reservation(
        None, "2", "99", "2024-05-01", "2024-05-05", "0.0", "Confirmed"
    )
    self.mock_db.fetch_one.return_value = None
    with self.assertRaises(ReservationException) as context:
        self.service.create_reservation(reservation)
    self.assertEqual(str(context.exception), "Vehicle does not exist.")

def test_create_reservation_vehicle_not_available(self):
    reservation = Reservation(
        None, "2", "3", "2024-05-01", "2024-05-05", "0.0", "Confirmed"
    )
    self.mock_db.fetch_one.return_value = (0, 1000.0)
    with self.assertRaises(ReservationException) as context:
        self.service.create_reservation(reservation)
    self.assertEqual(str(context.exception), "Vehicle is not available for reservation.")

def test_create_reservation_invalid_date_range(self):
    reservation = Reservation(
        None, "2", "3", "2024-05-05", "2024-05-01", "0.0", "Confirmed"
    )
    self.mock_db.fetch_one.return_value = (1, 1000.0)
    with self.assertRaises(InvalidInputException) as context:
        self.service.create_reservation(reservation)
    self.assertEqual(str(context.exception), "End date must be after start date.")


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()

    def test_get_pending_reservation_success(self):
        self.mock_db.fetch_query.return_value = [
            (1, 101, 201, "2025-04-20", "2025-04-22", 7000.0, "Pending")
        ]
        self.service.get_pending_reservation()
        self.mock_db.fetch_query.assert_called_once_with(
            "SELECT * from Reservation WHERE Status = 'Pending'"
        )

    def test_get_pending_reservation_empty(self):
        self.mock_db.fetch_query.return_value = []
        with self.assertRaises(ReservationException) as context:
            self.service.get_pending_reservation()
        self.assertEqual(str(context.exception), "No Pending reservations")


    def test_get_confirmed_reservation_success(self):
        self.mock_db.fetch_query.return_value = [
            (2, 102, 202, "2025-04-18", "2025-04-20", 6000.0, "Confirmed")
        ]
        self.service.get_confirmed_reservation()
        self.mock_db.fetch_query.assert_called_once_with(
            "SELECT * from Reservation WHERE Status = 'Confirmed'"
        )

    def test_get_confirmed_reservation_empty(self):
        self.mock_db.fetch_query.return_value = []
        with self.assertRaises(ReservationException) as context:
            self.service.get_confirmed_reservation()
        self.assertEqual(str(context.exception), "No Confirmed reservations")

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

**test_customer_service.py:**

```
import unittest
from unittest.mock import MagicMock
```

```
from datetime import date
from dao.customer_service import CustomerService
from entity.customer import Customer
from exceptions.invalid_input_exception import InvalidInputException
from exceptions.customer_not_found_exception import CustomerNotFoundException
from 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")]
        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()
```

**test_reservation_service.py:**

```
import unittest
from unittest.mock import MagicMock
from datetime import date
from entity.reservation import Reservation
from dao.reservation_service import ReservationService
from exceptions.invalid_input_exception import InvalidInputException
from 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", "2024-05-01", "2024-05-05", "0.0", "Confirmed"
        )
        # Vehicle is available and daily rate is 1000.0
        self.mock_db.fetch_one.return_value = (1, 1000.0)
        self.service.create_reservation(reservation)
        self.mock_db.fetch_one.assert_called_once_with(
            "SELECT Availability, DailyRate FROM Vehicle WHERE VehicleID = %s", ("3",)
        )
        self.mock_db.execute_query.assert_called_once_with(
            """
            INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost,
Status)
            VALUES (%s, %s, %s, %s, %s, %s)
        """,
            ("2", "3", "2024-05-01", "2024-05-05", 4000.0, "Confirmed")
        )

    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()
```

**test_vehicle_service.py:**

```
import unittest
from unittest.mock import MagicMock
from dao.vehicle_service import VehicleService
from entity.vehicle import Vehicle
from exceptions.invalid_input_exception import InvalidInputException
from exceptions.vehicle_not_found_exception import VehicleNotFoundException
from exceptions.database_connection_exception import DatabaseConnectionException

class TestVehicleService(unittest.TestCase):
    def setUp(self):
        self.mock_db = MagicMock()
        self.service = VehicleService(self.mock_db)


    def test_get_vehicle_by_id_valid(self):
        self.mock_db.fetch_query.return_value = [(1, "ModelX", "Tesla", "2023", "Red",
"TS12B1234", 1, 3500)]
        self.service.get_vehicle_by_id("1")
        self.mock_db.fetch_query.assert_called_once()

    def test_get_vehicle_by_id_not_found(self):
        self.mock_db.fetch_query.return_value = []
        with self.assertRaises(VehicleNotFoundException):
            self.service.get_vehicle_by_id("999")

    def test_get_vehicle_by_id_db_error(self):
        self.mock_db.fetch_query.side_effect = DatabaseConnectionException("DB down")
        with self.assertRaises(DatabaseConnectionException):
            self.service.get_vehicle_by_id("1")


    def test_get_available_vehicles_success(self):
        self.mock_db.fetch_query.return_value = [
            (1, "ModelX", "Tesla", "2023", "Red", "TS12AB1234", 1, 3500)
        ]
```

```
    vehicles = self.service.get_available_vehicles()
    self.assertIsInstance(vehicles, list)

def test_get_available_vehicles_db_error(self):
    self.mock_db.fetch_query.side_effect = DatabaseConnectionException("DB error")
    with self.assertRaises(DatabaseConnectionException):
        self.service.get_available_vehicles()


def test_add_vehicle_valid(self):
    vehicle = Vehicle(None, "Model3", "Tesla", "2023", "Blue", "TN12C3456", "1", 4500)
    self.service.add_vehicle(vehicle)
    self.mock_db.execute_query.assert_called_once()

def test_add_vehicle_invalid_registration(self):
    vehicle = Vehicle(None, "Model3", "Tesla", "2023", "Blue", "INVALID", "1", 4500)
    with self.assertRaises(InvalidInputException):
        self.service.add_vehicle(vehicle)

def test_add_vehicle_invalid_year(self):
    vehicle = Vehicle(None, "Model3", "Tesla", "23", "Blue", "TN12C3456", "1", 4500)
    with self.assertRaises(InvalidInputException):
        self.service.add_vehicle(vehicle)

def test_add_vehicle_db_exception(self):
    vehicle = Vehicle(None, "Model3", "Tesla", "2023", "Blue", "TN12B3456", "1", 4500)
    self.mock_db.execute_query.side_effect = Exception("Insert failed")
    with self.assertRaises(DatabaseConnectionException):
        self.service.add_vehicle(vehicle)


def test_update_vehicle_valid(self):
    self.mock_db.execute_query.return_value = 1
    self.service.update_vehicle("1", "3000", "1")
    self.mock_db.execute_query.assert_called_once()

def test_update_vehicle_invalid_id(self):
    with self.assertRaises(InvalidInputException):
        self.service.update_vehicle("abc", "3000", "1")
```

```
    def test_update_vehicle_not_found(self):
        self.mock_db.execute_query.return_value = 0
        with self.assertRaises(VehicleNotFoundException):
            self.service.update_vehicle("999", "3000", "1")

    def test_update_vehicle_db_exception(self):
        self.mock_db.execute_query.side_effect = DatabaseConnectionException("Update failed")
        with self.assertRaises(DatabaseConnectionException):
            self.service.update_vehicle("1", "3000", "1")


    def test_remove_vehicle_valid(self):
        self.mock_db.execute_query.return_value = 1
        self.service.remove_vehicle("1")
        self.mock_db.execute_query.assert_called_once()

    def test_remove_vehicle_invalid_id(self):
        with self.assertRaises(InvalidInputException):
            self.service.remove_vehicle("xyz")

    def test_remove_vehicle_not_found(self):
        self.mock_db.execute_query.return_value = 0
        with self.assertRaises(VehicleNotFoundException):
            self.service.remove_vehicle("999")

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

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

**LOGIN MENU:**

```
===== CarConnect Login Menu =====
1. Customer Sign Up
2. Customer Login
3. Admin Login
0. Exit
Enter choice:
```

**CUSTOMER SIGN UP:**

```
===== CarConnect Login Menu =====
1. Customer Sign Up
2. Customer Login
3. Admin Login
0. Exit
Enter choice: 1
First name: anush
Last name: kumar
Email: anushkumar@gmail.com
Phone: 8527413690
Address: Chennai
Username: anush01
Password: anush123
DB Successful! !!
Customer registered successfully!
```

**CUSTOMER LOGIN:**

```
2. Customer Login
3. Admin Login
0. Exit
Enter choice: 2
Username: anush01
Password: anush123
Successfully logged in!
Welcome, anush (Customer ID: 15)
```

**ADMIN LOGIN(super admin):**

```
===== CarConnect Login Menu =====
1. Customer Sign Up
2. Customer Login
3. Admin Login
0. Exit
Enter choice: 3
Username: anush
Password: 123
Successfully logged in!

Welcome, anush (super admin)
```

**ADMIN LOGIN (Fleet admin):**

```
===== CarConnect Login Menu =====
1. Customer Sign Up
2. Customer Login
3. Admin Login
0. Exit
Enter choice: 3
Username: hinac
Password: admin456
Successfully logged in!

Welcome, Hina (fleet manager)
```

**CUSTOMER DASHBOARD:**

```
Welcome, anush (Customer ID: 15)

--- Customer Dashboard ---
1. Update Profile
2. Check Customer Details
3. Create Reservation
4. Get Reservation by ID
5. Delete Account
6. Cancel Reservation
0. Logout
Enter choice:
```

1. **UPDATE PROFILE :**

```
6. Cancel Reservation
0. Logout
Enter choice: 1
Customer ID: 15
First Name: anush
Last Name: kumar
New Email: anush@yahoo.com
New Phone: 4561237890
New Address: London
Username: anush001
DB Successful! !!
Customer updated.
```

2. **CHECK CUSTOMER DETAIL:**

```
6. Cancel Reservation
0. Logout
Enter choice: 2
Customer ID: 15
```

| CustomerID | Name | Email | Phone | RegistrationDate |
|---|---|---|---|---|
| 15 | anush | anush@yahoo.com | 4561237890 | 2025-04-20 07:21:37 |

## 3. CREATE RESERVATION:

```
Enter choice: 3
```

| VehicleID | Model | Make | Year | Color | RegistrationNumber | Availability | DailyRate |
|---|---|---|---|---|---|---|---|
| 1 | Polo | Volkswagen | 2022 | Blue | MH01XY1234 | 1 | 12 |
| 2 | Verna | Hyundai | 2021 | Silver | DL02YZ5678 | 1 | 1800 |
| 3 | Thar | Mahindra | 2023 | Black | KA03AB9101 | 1 | 2200 |
| 4 | Nexon | Tata | 2022 | White | TN04CD1122 | 1 | 1600 |
| 5 | Seltos | Kia | 2021 | Red | KL05EF3344 | 1 | 1900 |
| 6 | Brezza | Maruti | 2020 | Orange | AP06GH5566 | 1 | 1700 |
| 7 | Compass | Jeep | 2023 | Grey | GJ07IJ7788 | 1 | 2800 |
| 8 | Harrier | Tata | 2022 | Brown | MP08KL9900 | 1 | 2100 |
| 9 | Sonet | Kia | 2021 | Yellow | RJ09MN1112 | 1 | 1750 |
| 11 | TVS | lp | 2020 | black | 3469 | 1 | 14 |

```
Customer ID: 15
Vehicle ID: 11
Start Date (YYYY-MM-DD): 2025-04-10
End Date (YYYY-MM-DD): 2025-04-15
DB Successful! !!
Reservation created successfully with ID: 41
Total cost calculated: ₹70.00
Reservation created.
```

## 4. GET RESRVATION BY ID:

```
Enter choice: 4
Reservation ID: 41
```

| ReservationID | CustomerID | VehicleID | StartDate | EndDate | TotalCost | Status |
|---|---|---|---|---|---|---|
| 41 | 15 | 11 | 2025-04-10 00:00:00 | 2025-04-15 00:00:00 | 70 | pending |

5. **CANCEL RESERVATION:**

```
--- Customer Dashboard ---
1. Update Profile
2. Check Customer Details
3. Create Reservation
4. Get Reservation by ID
5. Delete Account
6. Cancel Reservation
0. Logout
Enter choice: 6
Reservation ID to cancel: 41
DB Successful! !!
Reservation cancelled.
```

6. **DELETE ACCOUNT :**

```
--- Customer Dashboard ---
1. Update Profile
2. Check Customer Details
3. Create Reservation
4. Get Reservation by ID
5. Delete Account
6. Cancel Reservation
0. Logout
Enter choice: 5
Customer ID: 15
DB Successful! !!
Customer deleted.
```

**SUPER ADMIN DASHBOARD :**

```
Successfully logged in!

Welcome, anush (super admin)

--- Super Admin Dashboard ---
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
6. Get Customer by ID
7. Get Customer by Username
8. Delete Customer
9. Add Vehicle
10. Get Vehicle by ID
11. Get Available Vehicles
12. Update Vehicle
13. Delete Vehicle
14. Get Reservation by ID
15. Get Reservation by Customer ID
16. Update Reservation
17. Cancel Reservation
18. Generate Reservation History Report
19. Generate Vehicle Utilization Report
20. Generate Revenue Report
0. Logout
Enter choice:
```

1. **GET ADMIN BY ID:**

```
Enter choice: 2
Admin ID: 14
```

| | | AdminID | Name | Email | Username | Password | Role | JoinDate |
|---|---|---|---|---|---|---|---|---|
| 14 | kamal | hasan | kamal@gmail.com | 1452369870 | kamal | kamal | super admin | 2025-04-20 08:07:44 |

2. **GET ADMIN BY USERNAME :**

```
Enter choice: 3
Username: vk
```

| | | AdminID | Name | Email | Username | Password | Role | JoinDate |
|---|---|---|---|---|---|---|---|---|
| 12 | vk | vk | vk | 1236547890 | vk | vk | super admin | 2025-04-15 09:20:05 |

3. **UPDATE ADMIN:**

```
Enter choice: 4
Admin ID: 12
Enter first name: varun
Enter last name: kumar
New Email: varun@gmail.com
New Phone: 7412589630
New Username: varun
Role('super admin', 'fleet manager'): super admin
DB Successful! !!
Admin updated.
```

4. **DELETE ADMIN:**

```
Enter choice: 5
Admin ID to delete: 14
DB Successful! !!
```

5. **ADD VEHICLE :**

```
Enter choice: 9
Model: honda
Make: SUV
Year: 2024
Color: white
Registration Number: TN13A7854
Availability (1/0): 1
Daily Rate: 500
DB Successful! !!
Vehicle added.
```

6. **GET VEHICLE BY ID :**

```
Enter choice: 10
Vehicle ID: 16
```

| VehicleID | Model | Make | Year | Color | RegistrationNumber | Availability | DailyRate |
|---|---|---|---|---|---|---|---|
| 16 | honda | SUV | 2024 | white | TN13A7854 | 1 | 500 |

7. **UPDATE VEHICLE :**

```
0. Logout
Enter choice: 12
Vehicle ID to update: 16
New daily rate: 1000
Availability (1 or 0): 1
DB Successful! !!
```

8. **DELETE VEHICLE:**

```
19. Generate Vehicle Utilization Report
20. Generate Revenue Report
0. Logout
Enter choice: 13
Vehicle ID to delete: 16
DB Successful! !!
```

9. **GET RSERVATION BU CUSTOMER ID:**

```
Enter choice: 15
Customer ID: 6
```

| ReservationID | CustomerID | VehicleID | StartDate | EndDate | TotalCost | Status |
|---|---|---|---|---|---|---|
| 7 | 6 | 4 | 2025-04-13 11:00:00 | 2025-04-14 11:00:00 | 1600 | completed |
| 17 | 6 | 4 | 2025-04-13 11:00:00 | 2025-04-14 11:00:00 | 1600 | pending |

10. **UPDATE RESRVATION:**

```
0. Logout
Enter choice: 16
Reservation ID: 40
New status (pending/confirmed/completed): confirmed
DB Successful! !!
```

11. **CANCEL RSERVATION :**

```
Enter choice: 17
Reservation ID to cancel: 40
DB Successful! !!
Reservation cancelled.
```

## 12. GENERATE RSERVATION HISTORY REPORT :

```
Enter choice: 18

--- Reservation History Report ---
```

| Reservation ID | Customer ID | Vehicle ID | Start Date | End Date | Status |
|---|---|---|---|---|---|
| 7 | 6 | 4 | 2025-04-13 11:00:00 | 2025-04-14 11:00:00 | completed |
| 17 | 6 | 4 | 2025-04-13 11:00:00 | 2025-04-14 11:00:00 | pending |
| 4 | 5 | 7 | 2025-04-12 08:00:00 | 2025-04-15 08:00:00 | pending |
| 14 | 5 | 7 | 2025-04-12 08:00:00 | 2025-04-15 08:00:00 | pending |
| 6 | 7 | 9 | 2025-04-09 10:00:00 | 2025-04-10 10:00:00 | confirmed |
| 16 | 7 | 9 | 2025-04-09 10:00:00 | 2025-04-10 10:00:00 | confirmed |
| 3 | 1 | 2 | 2025-04-08 12:00:00 | 2025-04-11 12:00:00 | confirmed |
| 13 | 1 | 2 | 2025-04-08 12:00:00 | 2025-04-11 12:00:00 | confirmed |
| 5 | 3 | 1 | 2025-04-06 18:00:00 | 2025-04-07 18:00:00 | completed |
| 15 | 3 | 1 | 2025-04-06 18:00:00 | 2025-04-07 18:00:00 | completed |
| 2 | 4 | 5 | 2025-04-05 09:00:00 | 2025-04-06 09:00:00 | completed |
| 12 | 4 | 5 | 2025-04-05 09:00:00 | 2025-04-06 09:00:00 | completed |
| 9 | 8 | 6 | 2025-04-02 10:00:00 | 2025-04-03 10:00:00 | completed |
| 19 | 8 | 6 | 2025-04-02 10:00:00 | 2025-04-03 10:00:00 | completed |

**13. GENERATE VEHICLE UTILIZATION REPORT :**

```
Enter choice: 19

--- Vehicle Utilization Report ---
```

| Vehicle ID | Total Reservations |
|---|---|
| 1 | 6 |
| 2 | 3 |
| 9 | 3 |
| 11 | 3 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |
| 7 | 2 |
| 12 | 1 |
| 15 | 1 |

**14. GENERATE REVENUE REPORT:**

```
Enter choice: 20

--- Revenue Report ---
```

| Vehicle ID | Revenue |
|---|---|
| 3 | ₹8800.00 |
| 5 | ₹3800.00 |
| 6 | ₹3400.00 |
| 1 | ₹3000.00 |
| 4 | ₹1600.00 |

**FLEET ADMIN DASHBOARD:**

```
--- Fleet Admin Dashboard ---
1. Get Admin by ID
2. Get Admin by Username
3. Get Customer by ID
4. Get Customer by Username
5. Delete Customer
6. List Available Vehicles
7. Update Vehicle
8. Show Confirmed Reservations
9. Show Pending Reservations
10. Update Reservation Status
11. Get Reservation by Customer ID
0. Logout
```

## 1. LIST AVAILABLE VEHICLE:

```
11. Get Reservation by Customer ID
0. Logout
Enter choice: 6
```

| VehicleID | Model | Make | Year | Color | RegistrationNumber | Availability | DailyRate |
|---|---|---|---|---|---|---|---|
| 1 | Polo | Volkswagen | 2022 | Blue | MH01XY1234 | 1 | 12 |
| 2 | Verna | Hyundai | 2021 | Silver | DL02YZ5678 | 1 | 1800 |
| 3 | Thar | Mahindra | 2023 | Black | KA03AB9101 | 1 | 2200 |
| 4 | Nexon | Tata | 2022 | White | TN04CD1122 | 1 | 1600 |
| 5 | Seltos | Kia | 2021 | Red | KL05EF3344 | 1 | 1900 |
| 6 | Brezza | Maruti | 2020 | Orange | AP06GH5566 | 1 | 1700 |
| 7 | Compass | Jeep | 2023 | Grey | GJ07IJ7788 | 1 | 2800 |
| 9 | Sonet | Kia | 2021 | Yellow | RJ09MN1112 | 1 | 1750 |
| 11 | TVS | lp | 2020 | black | 3469 | 1 | 14 |

## 2. SHOW CONFIRMED RSERVATION:

```
Enter choice: 8
```

| ReservationID | CustomerID | VehicleID | StartDate | EndDate | TotalCost | Status |
|---|---|---|---|---|---|---|
| 3 | 1 | 2 | 2025-04-08 12:00:00 | 2025-04-11 12:00:00 | 5400 | confirmed |
| 6 | 7 | 9 | 2025-04-09 10:00:00 | 2025-04-10 10:00:00 | 1750 | confirmed |
| 13 | 1 | 2 | 2025-04-08 12:00:00 | 2025-04-11 12:00:00 | 5400 | confirmed |
| 16 | 7 | 9 | 2025-04-09 10:00:00 | 2025-04-10 10:00:00 | 1750 | confirmed |
| 23 | 2 | 1 | 2020-03-02 00:00:00 | 2020-03-04 00:00:00 | 1550 | confirmed |
| 24 | 3 | 1 | 2020-03-02 00:00:00 | 2020-03-04 00:00:00 | 1550 | confirmed |

## 3. SHOW PENDING RSERVATION:

```
Enter choice: 9
```

| ReservationID | CustomerID | VehicleID | StartDate | EndDate | TotalCost | Status |
|---|---|---|---|---|---|---|
| 4 | 5 | 7 | 2025-04-12 08:00:00 | 2025-04-15 08:00:00 | 8400 | pending |
| 14 | 5 | 7 | 2025-04-12 08:00:00 | 2025-04-15 08:00:00 | 8400 | pending |
| 17 | 6 | 4 | 2025-04-13 11:00:00 | 2025-04-14 11:00:00 | 1600 | pending |
| 22 | 1 | 1 | 2020-12-12 00:00:00 | 2020-12-14 00:00:00 | 12 | pending |
| 28 | 14 | 15 | 2020-03-20 00:00:00 | 2020-03-21 00:00:00 | 12 | pending |
| 29 | 14 | 2 | 2020-02-03 00:00:00 | 2020-03-02 00:00:00 | 150 | pending |
| 30 | 1 | 1 | 2020-02-03 00:00:00 | 2020-02-03 00:00:00 | 150 | pending |