

CAR CONNECT – A CAR RENTAL PLATFORM

CASESTUDY

ABSTRACT:

Car Connect is a cutting-edge console application for renting cars that was created to simplify bookings, administration, and client communications. Customers can browse, book, and manage rentals with system's user-friendly interface and admins can add, remove, update vehicles and also can analyze reservations made by customer.

Important characteristics include:

Easy Booking: With real-time availability checks, customers can search, compare, and reserve cars online.

Admin management: Administrators can keep track of maintenance, add, edit, or remove vehicles, and also they can view the reservation made by customers.

Duplicate Booking Prevention: Real-time availability checks using start/end dates and time slots. Automated vehicle hiding upon booking to eliminate double-booking risks.

Dynamic Cancellation Policies: 20% penalty for cancellations within 24 hours of rental start. Zero-fee cancellations beyond the 24-hour window.

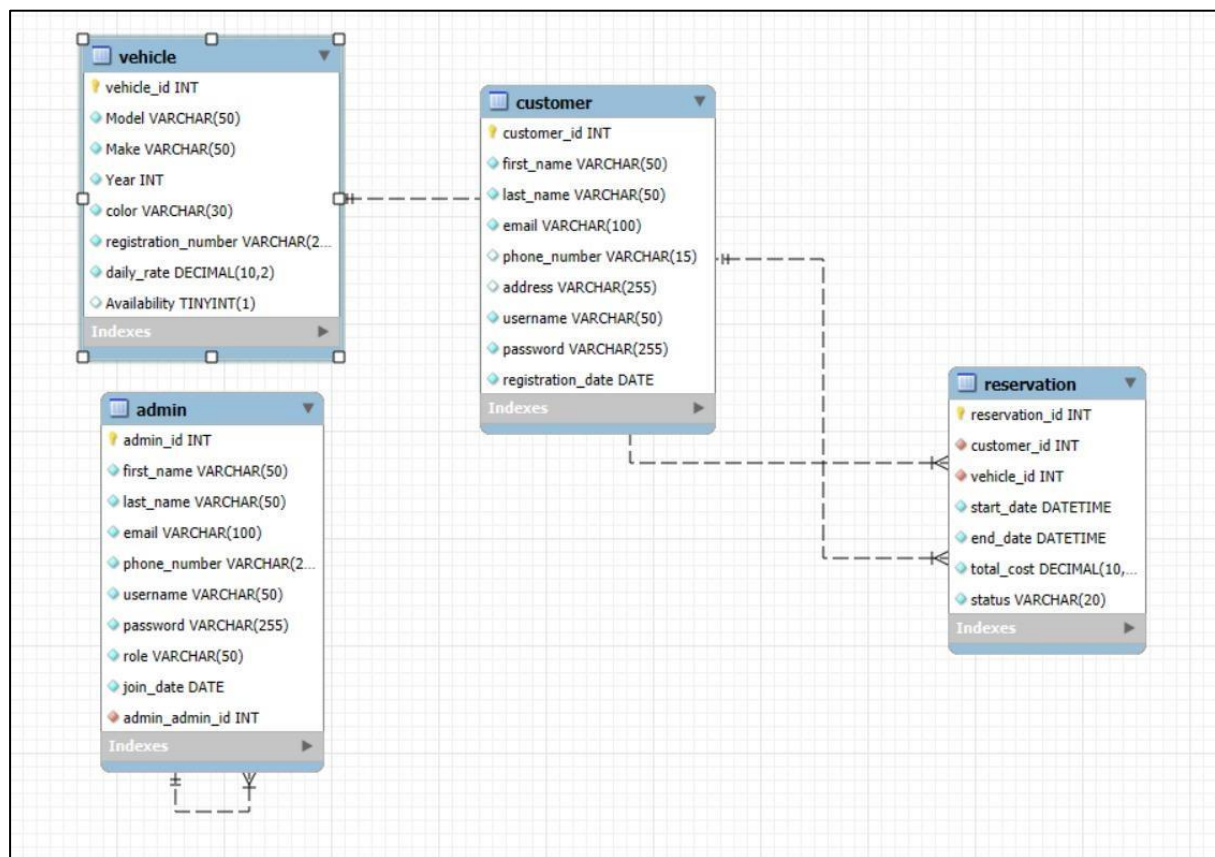
STRUCTURE:

Python_Project/

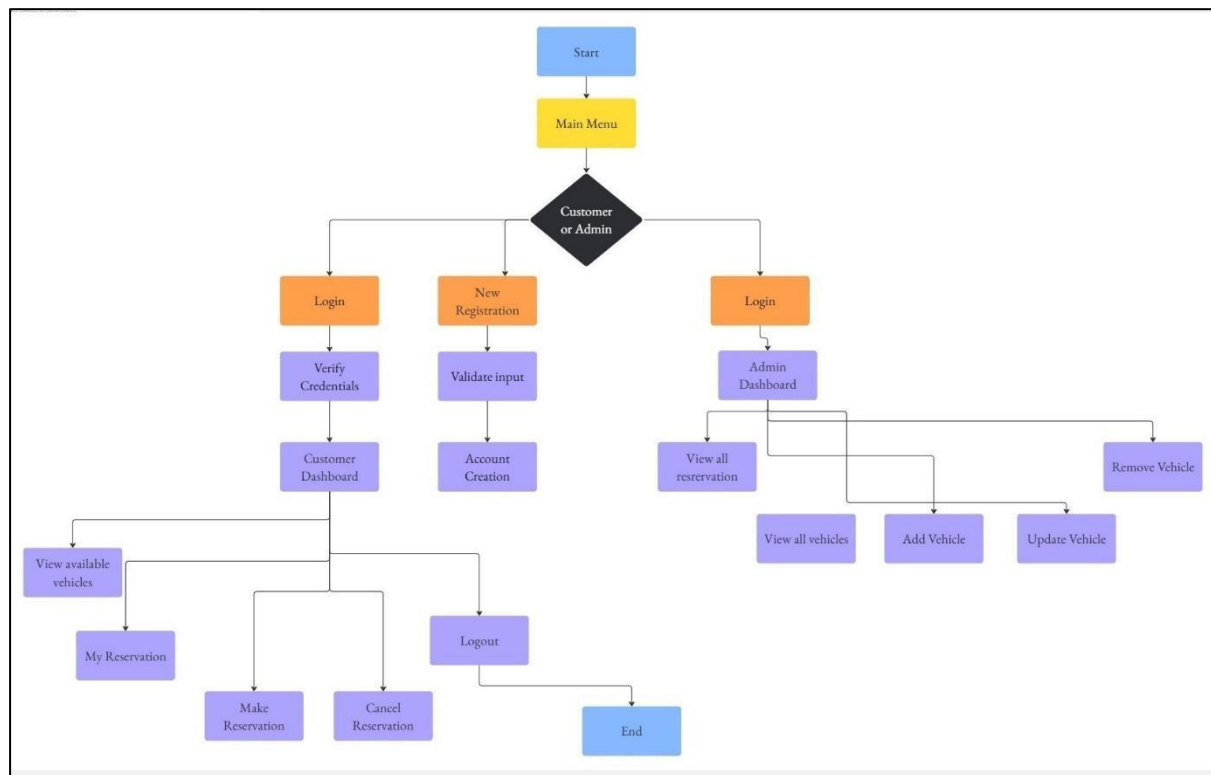
```
|— entity/
|   |— Customer.py
|   |— Vehicle.py
|   |— Admin.py
|   |— Reservation.py
|— service/
|   |— CustomerService.py
|   |— ICustomerService.py
|   |— VehicleService.py
```

- | | — IVehicleService.py
- | | — AdminService.py
- | | — IAdminService.py
- | | — ReservationService.py
- | | — IReservationService.py
- | — Exceptions.py
- | — mainfunc.py
- | — test/
 - | — test_carconnect.py

ENTITY RELATIONSHIP DIAGRAM:



FLOW CHART:



Business Logic Implementation:

1. Authentication & Validation

- Password hashing (bcrypt) for both customers and admins
- Strict validation of:
 - Customer emails (regex pattern)
 - Password strength (minimum length + special characters)
 - Phone numbers (length-10)
 - Vehicle year (1900–current year range)
 - Registration numbers (Eg., KL-1234)

2. Booking Management

Duplicate Prevention:

Vehicles are automatically hidden when:

- Start/end dates overlap existing bookings
- Time slots conflict (hourly/day-based rentals)

Cancellation

Rules:

- 20% fee if cancelled <24 hours before rental start
- Zero penalty for ≥ 24 -hour cancellations

Testing: Some cases are verified via pytest (test_carconnect.py).

PROJECT:

application.

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.

```
CREATE TABLE customer (  
    customer_id INT AUTO_INCREMENT PRIMARY KEY,  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL,  
    email VARCHAR(100) NOT NULL,  
    phone_number VARCHAR(15),  
    address VARCHAR(255),  
    username VARCHAR(50) NOT NULL,  
    password VARCHAR(255) NOT NULL,  
    registration_date DATE NOT NULL  
);
```

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.

```

CREATE TABLE Vehicle (
    vehicle_id INT AUTO_INCREMENT PRIMARY KEY,
    Model VARCHAR(50) NOT NULL,
    Make VARCHAR(50) NOT NULL,
    Year INT NOT NULL,
    color VARCHAR(30) NOT NULL,
    registration_number VARCHAR(20) UNIQUE NOT NULL,
    daily_rate DECIMAL(10, 2) NOT NULL,
    Availability BOOLEAN DEFAULT TRUE
);

```

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).

```

CREATE TABLE reservation (
    reservation_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT NOT NULL,
    vehicle_id INT NOT NULL,
    start_date DATETIME NOT NULL,
    end_date DATETIME NOT NULL,
    total_cost DECIMAL(10, 2) NOT NULL,
    status VARCHAR(20) NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (vehicle_id) REFERENCES vehicle(vehicle_id)
);

```

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.

```
CREATE TABLE admin (  
    admin_id INT PRIMARY KEY AUTO_INCREMENT,  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL,  
    email VARCHAR(100) UNIQUE NOT NULL,  
    phone_number VARCHAR(20) NOT NULL,  
    username VARCHAR(50) UNIQUE NOT NULL,  
    password VARCHAR(255) NOT NULL,  
    role VARCHAR(50) NOT NULL,  
    join_date DATE NOT NULL  
);
```

Classes:

- **Customer:**
 - Properties: **CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate**
 - Methods: **Authenticate(password)**

Customer.py

```
from Exceptions import AuthenticationException  
from datetime import date  
  
class Customer():  
    def  
    __init__(self, customer_id: int, first_name: str, last_name: str, email: str, phone_number: str, address
```

```

:str,username:str,password:str,registration_date:date):
    self.customer_id = customer_id
    self.first_name = first_name
    self.last_name = last_name
    self.email = email
    self.phone_number = phone_number
    self.address = address
    self.username = username
    self.password = password
    self.registration_date = registration_date
def authenticate_password(self,password):
    if self.password != password:
        raise AuthenticationException()
    return True

```

- **Vehicle:**
 - Properties: **VehicleID, Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate**

Vehicle.py

```

class Vehicle():
    def
__init__(self,vehicle_id:int,Model:str,Make:str,Year:str,color:str,registration_number:int,daily
_rate:float,Availability:bool):
    self.vehicle_id = vehicle_id
    self.Model = Model
    self.Make = Make
    self.Year = Year
    self.color = color
    self.registration_number = registration_number
    self.daily_rate = daily_rate
    self.Availability = Availability

```

- **Reservation:**
 - Properties: **ReservationID, CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status**
 - Methods: **CalculateTotalCost()**

Reservation.py

```
from decimal import Decimal
class Reservation:
    def __init__(self, reservation_id: int, customer_id: int, vehicle_id: int, start_date, end_date,
total_cost: float, status: str):
        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 generate_invoice(self) -> str:
        subtotal = self.total_cost / Decimal("1.18")
        tax = self.total_cost * Decimal("0.18")
        return f"""
Invoice for Reservation #{self.reservation_id}
Dates: {self.start_date} to {self.end_date}
Subtotal: {subtotal:.2f}
Tax (18%): {tax:.2f}
Total: {self.total_cost:.2f}
"""
```

- **Admin:**
 - Properties: **AdminID, FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate**
 - Methods: **Authenticate(password)**

Admin.py

```
import bcrypt

class Admin:
    def __init__(self, admin_id: int, first_name: str, last_name: str, email: str, phone_number:
str,
        username: str, password: str, role: str, join_date):
        self.admin_id = admin_id
        self.first_name = first_name
```



```
self.last_name = last_name
self.email = email
self.phone_number = phone_number
self.username = username
self.password = password
self.role = role
self.join_date = join_date
```

```
def authenticate(self, input_password):
    return bcrypt.checkpw(input_password.encode(), self.password.encode())
```

Interfaces:

- **ICustomerService:**
 - GetCustomerById(customerId)
 - GetCustomerByUsername(username)
 - RegisterCustomer(customerData)
 - UpdateCustomer(customerData)
 - DeleteCustomer(customerId)

ICustomerService.py

```
from abc import ABC, abstractmethod
```

```
class ICustomerService(ABC):
```

```
    @abstractmethod
```

```
    def getCustomerbyId(self, customer_id):
        pass
```

```
    @abstractmethod
```

```
    def getCustomerByUsername(self, username):
        pass
```

```
    @abstractmethod
```

```
    def registerCustomer(self, customer_data):
        pass
```

```
    @abstractmethod
```

```
    def updateCustomer(self, customer_data):
        pass
```

```
@abstractmethod
def deletecustomer(self, customer_id):
    pass
```

```
@abstractmethod
def authenticate_customer(self, username, password):
    pass
```

- **CustomerService (implements ICustomerService):**
 - **Methods: GetCustomerById, GetCustomerByUsername, RegisterCustomer, UpdateCustomer, DeleteCustomer**

CustomerService.py

```
import mysql.connector
from mysql.connector import Error
import re
import bcrypt
from service.ICustomerService import ICustomerService
from entity.Customer import Customer
from datetime import date
from Exceptions import (AuthenticationException, DuplicateCustomerException,
                        CustomerNotFoundException,
                        InvalidPhoneNumberException, InvalidEmailException,
                        WeakPasswordException)
```

```
class CustomerService(ICustomerService):
```

```
    def __init__(self):
```

```
        self.con = self._get_db_connection()
```

```
    def _get_db_connection(self):
```

```
        try:
            return mysql.connector.connect(
                host="localhost",
                username="root",
                password="bablu345.",
                database="carconnect",
                port=3306
            )
```

```

except Error as e:

    print(f'Database not connected: {e}')
    raise

def _hash_password(self, password):

    salt = bcrypt.gensalt()
    return bcrypt.hashpw(password.encode('utf-8'), salt).decode('utf-8')

def _check_password(self, provided_password, stored_hash):

    return bcrypt.checkpw(provided_password.encode('utf-8'), stored_hash.encode('utf-8'))

def migrate_passwords_to_hashes(self):
    cursor = None
    try:
        cursor = self.con.cursor(dictionary=True)

        cursor.execute("select customer_id, password from customer")
        customers = cursor.fetchall()

        for customer in customers:
            plain_password = customer["password"]

            if plain_password.startswith("$2b$"):
                continue

            hashed_password = self._hash_password(plain_password)

            cursor.execute(
                "update customer set password = %s where customer_id = %s",
                (hashed_password, customer["customer_id"])
            )

            self.con.commit()
            print(f'Updated {len(customers)} passwords to hashed format.')

    except Error as e:
        self.con.rollback()
        print(f'Migration failed: {e}')
        raise

    finally:
        if cursor:
            cursor.close()

```

```

def _validate_email(self, email):

    if not re.match(r'[\w\.-]+@[\w\.-]+\.\w+$', email):
        raise InvalidEmailException()

def _validate_password(self, password):

    if len(password) < 8:
        raise WeakPasswordException("Password must contain atleast 8 characters")

    if not re.search(r'[!@#$%^*&().,?":|{}<>]', password):
        raise WeakPasswordException("Password must contain a special character")

def _validate_phonenumber(self, phone_number):

    if not re.match(r'^\d{10}$', phone_number):
        raise InvalidPhoneNumberException()

def registerCustomer(self, customer_data):
    cursor = None

    try:
        required_field = ['first_name', 'last_name', 'email', 'phone_number', 'address',
'username', 'password']
        for field in required_field:
            if field not in customer_data:
                raise KeyError(f"Missing required field: {field}")
        self._validate_email(customer_data["email"])
        self._validate_password(customer_data["password"])
        self._validate_phonenumber(customer_data["phone_number"])
        cursor = self.con.cursor()
        cursor.execute(
            """
            select customer_id from customer where email = %s or username = %s
            """, (customer_data["email"], customer_data["username"]))
        if cursor.fetchone():
            raise DuplicateCustomerException()

        hashed_password = self._hash_password(customer_data["password"])
        registration_date = customer_data.get("registration_date", date.today())

        cursor.execute(
            """
            insert into
customer(first_name,last_name,email,phone_number,address,username,password,registration

```

```

_date)
        values (%s,%s,%s,%s,%s,%s,%s,%s,%s)
        """, (customer_data["first_name"], customer_data["last_name"],
customer_data["email"],
        customer_data["phone_number"], customer_data["address"],
customer_data["username"],
        hashed_password,
        registration_date

    ))
    self.con.commit()
    customer_id = cursor.lastrowid
    print(f'Customer {customer_id} registered successfully!')
    return True
except KeyError as e:
    print(f'Missing required field: {e}')
    return False
except Error as e:
    self.con.rollback()
    print("Database error")
    raise
finally:
    if cursor:
        cursor.close()

def authenticate_customer(self, username, password):
    cursor = None

    try:
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("Select*from customer where username = %s ", (username,))
        customer_data = cursor.fetchone()
        if not customer_data:
            raise AuthenticationException("Invalid Username or Password")
        if not self._check_password(password, customer_data["password"]):
            raise AuthenticationException("Invalid Password")
        print("Authentication successful")
        return True
    except Error as e:
        print(f'Database Error: {e}')
        raise
    finally:
        if cursor:
            cursor.close()

def updateCustomer(self, customer_data: dict):

```

```
cursor = None
```

```
try:
```

```
    customer_id = customer_data.get("customer_id")
```

```
    if not customer_id:
```

```
        raise ValueError("Customerid required")
```

```
    existing_customer = self.getCustomerbyId(customer_id)
```

```
    if not existing_customer:
```

```
        raise CustomerNotFoundException("User not found")
```

```
    if "email" in customer_data:
```

```
        self._validate_email(customer_data["email"])
```

```
    if "password" in customer_data:
```

```
        self._validate_password(customer_data["password"])
```

```
    customer_data["password"] = self._hash_password(customer_data["password"])
```

```
    if "phone_number" in customer_data:
```

```
        self._validate_phonenumber(customer_data["phone_number"])
```

```
    cursor = self.con.cursor(dictionary=True)
```

```
    cursor.execute(  
        """
```

```
        """
```

```
    update customer set
```

```
        first_name = %s,
```

```
        last_name = %s,
```

```
        email = %s,
```

```
        phone_number = %s,
```

```
        address = %s,
```

```
        username = %s,
```

```
        password = %s,
```

```
        registration_date = %s
```

```
    where customer_id = %s
```

```
        """,
```

```
        ,
```

```
        (customer_data.get("first_name", existing_customer.first_name),
```

```
        customer_data.get("last_name", existing_customer.last_name),
```

```
        customer_data.get("email", existing_customer.email),
```

```
        customer_data.get("phone_number", existing_customer.phone_number),
```

```
        customer_data.get("address", existing_customer.address),
```

```
        customer_data.get("username", existing_customer.username),
```

```
        customer_data.get("password", existing_customer.password),
```

```
        customer_data.get("registration_date", existing_customer.registration_date),
```

```
        customer_id
```

```
    )
```

```
)
```

```
    self.con.commit()
```

```
    print("Customer updated Successfully")
```

```
    return True
```

```

except Error as e:
    print(f"Database error: {e}")
    raise
finally:
    if cursor:
        cursor.close()

def deletecustomer(self, customer_id):
    cursor = None
    try:
        cursor = self.con.cursor()
        cursor.execute("select customer_id from customer where customer_id = %s",
(customer_id,))
        if not cursor.fetchone():
            raise CustomerNotFoundException()

        cursor.execute("delete from customer where customer_id = %s", (customer_id,))
        self.con.commit()
        print(f"Customer Id: {customer_id} deleted successfully")
        return True
    except Error as e:
        self.con.rollback()
        print(f"Database error: {e}")
        raise
    finally:
        if cursor:
            cursor.close()

def getCustomerbyId(self, customer_id):
    cursor = None
    try:
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("select*from customer where customer_id = %s", (customer_id,))
        customer_data = cursor.fetchone()
        if customer_data:
            return Customer(
                customer_id=customer_data["customer_id"],
                first_name=customer_data["first_name"],
                last_name=customer_data["last_name"],
                email=customer_data["email"],
                phone_number=customer_data["phone_number"],
                address=customer_data["address"],
                username=customer_data["username"],
                password=customer_data["password"],
                registration_date=customer_data["registration_date"]
            )
    except Error as e:
        self.con.rollback()
        print(f"Database error: {e}")
        raise
    finally:
        if cursor:
            cursor.close()

```

```

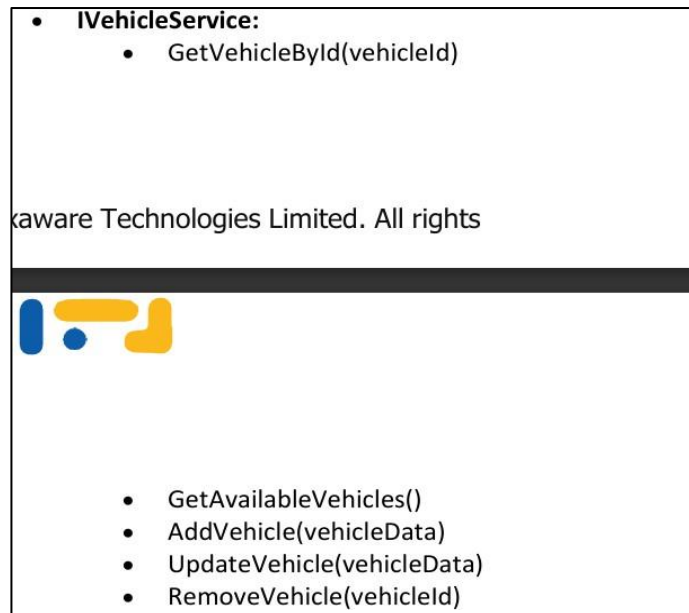
        )
        return None
    except Error as e:
        print(f"Database error: {e}")
        raise
    finally:
        if cursor:
            cursor.close()

def getCustomerByUsername(self, username):
    cursor = None
    try:
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("select*from customer where username = %s", (username,))
        customer_data = cursor.fetchone()
        if customer_data:
            return Customer(
                customer_id=customer_data["customer_id"],
                first_name=customer_data["first_name"],
                last_name=customer_data["last_name"],
                email=customer_data["email"],
                phone_number=customer_data["phone_number"],
                address=customer_data["address"],
                username=customer_data["username"],
                password=customer_data["password"],
                registration_date=customer_data["registration_date"]
            )
        return None

    except Error as e:
        print(f"Database error: {e}")
        raise
    finally:
        if cursor:
            cursor.close()

def close(self):
    if hasattr(self, 'con') and self.con:
        self.con.close()

```

IVehicleService.py

```
from abc import ABC, abstractmethod
```

```
class IVehicleService(ABC):
```

```
    @abstractmethod
```

```
    def GetVehicleById(self, vehicle_id):
```

```
        pass
```

```
    @abstractmethod
```

```
    def GetAvailableVehicles(self, Availability):
```

```
        pass
```

```
    @abstractmethod
```

```
    def AddVehicle(self, vehicle_data):
```

```
        pass
```

```
    @abstractmethod
```

```
    def UpdateVehicle(self, vehicle_data):
```

```
        pass
```

```
@abstractmethod
```

```
def RemoveVehicle(self, vehicle_id):
```

```
    pass
```

```
@abstractmethod
```

```
def GetAllVehicles(self):
```

```
    pass
```

- **VehicleService (implements IVehicleService):**
 - Methods: **GetVehicleById, GetAvailableVehicles, AddVehicle, UpdateVehicle, RemoveVehicle**

VehicleService.py

```
from service.IVehicleService import IVehicleService
import mysql.connector
import re
from mysql.connector import Error
from entity.Vehicle import Vehicle
from datetime import date
from Exceptions import (VehicleNotFoundException, DuplicateVehicleException,
                        InvalidDailyRateException,
                        InvalidRegNumberException, InvalidYearException,
                        OverlappingBookingException)
```

```
class VehicleService(IVehicleService):
```

```
    def __init__(self):
        self.con = self._get_db_connection()

    def _get_db_connection(self):
        try:
            return mysql.connector.connect(
                host="localhost",
                user="root",
                database="carconnect",
                password="bablu345.",
                port=3306
            )
        except Error as e:
            print(f"Database not connected: {e}")

    def _validate_year(self, year):
        curr_year = date.today().year
        if not (1900 <= int(year) <= curr_year):
            raise InvalidYearException()

    def _validate_registration_number(self, registration_number):
        if not re.match(r'^[A-Z]{2,3}-\d{3,4}$', registration_number):
            raise InvalidRegNumberException()

    def _validate_daily_rate(self, daily_rate):
        if daily_rate <= 0.0:
            raise InvalidDailyRateException()

    def AddVehicle(self, vehicle_data):
        cursor = None
```

```

try:
    required_fields = ['Model', 'Make', 'Year', 'color', 'registration_number', 'daily_rate',
'Availability']
    for field in required_fields:
        if field not in vehicle_data:
            raise KeyError(f"Missing required field: {field}")

    self._validate_year(vehicle_data["Year"])
    self._validate_registration_number(vehicle_data["registration_number"])
    self._validate_daily_rate(vehicle_data["daily_rate"])

    cursor = self.con.cursor()
    cursor.execute(
        """
        insert into vehicle(Model,Make,Year,color,registration_number,daily_rate,Availability)
        values(%s,%s,%s,%s,%s,%s,%s)
        """
        ,
        (vehicle_data["Model"], vehicle_data["Make"], vehicle_data["Year"],
vehicle_data["color"],
        vehicle_data["registration_number"], vehicle_data["daily_rate"],
vehicle_data["Availability"])
    )
    self.con.commit()
    vehicle_id = cursor.lastrowid
    print(f"Vehicle {vehicle_id} registered successfully")
    return True

except Error as e:
    self.con.rollback()
    print("Database Error")
    raise
finally:
    if cursor:
        cursor.close()

def UpdateVehicle(self, vehicle_data: dict):
    cursor = None
    try:
        vehicle_id = vehicle_data.get("vehicle_id")
        if not vehicle_id:
            print("Vehicle id required")
        existing_vehicle: Vehicle = self.GetVehicleById(vehicle_id)
        if not existing_vehicle:
            raise VehicleNotFoundException("Vehicle not found")

        if "Year" in vehicle_data:
            self._validate_year(vehicle_data["Year"])
        if "registration_number" in vehicle_data:
            self._validate_registration_number(vehicle_data["registration_number"])
        if "daily_rate" in vehicle_data:
            self._validate_daily_rate(vehicle_data["daily_rate"])

        cursor = self.con.cursor(dictionary=True)
        cursor.execute(
            """

```

```

        update vehicle set
        Model = %s,
        Make = %s,
        Year = %s,
        color = %s,
        registration_number = %s,
        daily_rate = %s,
        Availability = %s
        where vehicle_id = %s
        """
        (vehicle_data.get("Model", existing_vehicle.Model),
        vehicle_data.get("Make", existing_vehicle.Make),
        vehicle_data.get("Year", existing_vehicle.Year),
        vehicle_data.get("color", existing_vehicle.color),
        vehicle_data.get("registration_number", existing_vehicle.registration_number),
        vehicle_data.get("daily_rate", existing_vehicle.daily_rate),
        vehicle_data.get("Availability", existing_vehicle.Availability),
        vehicle_id
        )
    )

    self.con.commit()
    print("Vehicle Updated Successfully")
except Error as e:
    print("Database error")
    raise
finally:
    if cursor:
        cursor.close()

def RemoveVehicle(self, vehicle_id):
    cursor = None
    try:
        cursor = self.con.cursor()
        cursor.execute("select vehicle_id from vehicle where vehicle_id = %s", (vehicle_id,))
        if not cursor.fetchone():
            raise VehicleNotFoundException()
        cursor.execute("delete from vehicle where vehicle_id = %s", (vehicle_id,))
        self.con.commit()
        print(f"Vehicle {vehicle_id} removed successfully")
    except Error as e:
        print("Database error")
        raise
    finally:
        if cursor:
            cursor.close()

def GetVehicleById(self, vehicle_id):
    cursor = None
    try:
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("select*from vehicle where vehicle_id = %s", (vehicle_id,))
        vehicle_data = cursor.fetchone()
        if vehicle_data:
            return Vehicle(

```

```

        vehicle_id=vehicle_data["vehicle_id"],
        Model=vehicle_data["Model"],
        Make=vehicle_data["Make"],
        Year=vehicle_data["Year"],
        color=vehicle_data["color"],
        registration_number=vehicle_data["registration_number"],
        daily_rate=vehicle_data["daily_rate"],
        Availability=vehicle_data["Availability"]
    )
    return None
except Error as e:
    print("Database error")
    raise
finally:
    if cursor:
        cursor.close()

def GetAllVehicles(self):
    cursor = None
    try:
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("SELECT * FROM vehicle")
        vehicles = [
            Vehicle(
                vehicle_id=v["vehicle_id"],
                Make=v["Make"],
                Model=v["Model"],
                Year = v["Year"],
                color = v["color"],
                registration_number = v["registration_number"],
                daily_rate = v["daily_rate"],
                Availability=bool(v["Availability"])
            )
            for v in cursor.fetchall()
        ]
        return vehicles
    except Exception as e:
        print(f"Database error: {e}")
        raise
    finally:
        if cursor:
            cursor.close()

def GetAvailableVehicles(self, Availability=1):
    cursor = None
    try:
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("SELECT * FROM vehicle WHERE Availability = %s", (Availability,))
        vehicles = cursor.fetchall()
        return [
            Vehicle(
                vehicle_id=v["vehicle_id"],
                Model=v["Model"],
                Make=v["Make"],
                Year=v["Year"],

```

```

        color=v["color"],
        registration_number=v["registration_number"],
        daily_rate=float(v["daily_rate"]),
        Availability=v["Availability"]
    )
    for v in vehicles
]
except Error as e:
    print("Database error", e)
    raise
finally:
    if cursor:
        cursor.close()

def close(self):
    if hasattr(self, 'con') and self.con:
        self.con.close()

```

- **IReservationService:**
 - GetReservationById(reservationId)
 - GetReservationsByCustomerId(customerId)
 - CreateReservation(reservationData)
 - UpdateReservation(reservationData)
 - CancelReservation(reservationId)

IReservationservice.py

```
from abc import ABC, abstractmethod
```

```

class IReservationService(ABC):
    @abstractmethod
    def getReservationById(self, reservation_id):
        pass
    @abstractmethod
    def getReservationsByCustomerId(self, customer_id):
        pass
    @abstractmethod
    def createReservation(self, reservation_data):
        pass
    @abstractmethod
    def updateReservation(self, reservation_data):
        pass

    @abstractmethod
    def cancelReservation(self, reservation_id):
        pass
    @abstractmethod
    def _check_booking_overlap(self, vehicle_id, start_date, end_date):

```

```

        pass
    @abstractmethod
    def getAllReservations(self):
        pass

```

- **ReservationService (implements IReservationService):**
 - Methods: **GetReservationById, GetReservationsByCustomerId, CreateReservation, UpdateReservation, CancelReservation**

ReservationService.py

```

from service.IReservationService import IReservationService
from entity.Reservation import Reservation
from Exceptions import ReservationException, VehicleNotFoundException,
OverlappingBookingException
import mysql.connector
from mysql.connector import Error
from datetime import datetime
from decimal import Decimal

class ReservationService(IReservationService):
    def __init__(self):
        self.con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Zuhi743#",
            database="carconnect",
            port=3306
        )

    def _check_booking_overlap(self, vehicle_id: int, start_date: datetime, end_date:
datetime):
        cursor = self.con.cursor()
        cursor.execute("""
            SELECT 1 FROM reservation
            WHERE vehicle_id = %s AND status = 'confirmed'
            AND (start_date < %s AND end_date > %s)
            """, (vehicle_id, end_date, start_date))
        if cursor.fetchone():
            raise OverlappingBookingException("Vehicle is already booked for selected times.")
        cursor.close()

    def createReservation(self, reservation_data):
        cursor = None
        try:

```



```

customer_id = reservation_data["customer_id"]
vehicle_id = reservation_data["vehicle_id"]
start_date = reservation_data["start_date"]
end_date = reservation_data["end_date"]
status = reservation_data.get("status", "confirmed")

overlap_cursor = self.con.cursor()
overlap_cursor.execute("""
    SELECT 1 FROM reservation
    WHERE vehicle_id = %s
    AND status = 'confirmed'
    AND (start_date < %s AND end_date > %s)
""", (vehicle_id, end_date, start_date))
if overlap_cursor.fetchone():
    overlap_cursor.close()
    raise ReservationException("Vehicle is already booked for the selected date/time.")
overlap_cursor.close()

vehicle_cursor = self.con.cursor(dictionary=True)
vehicle_cursor.execute("SELECT * FROM vehicle WHERE vehicle_id = %s",
(vehicle_id,))
vehicle = vehicle_cursor.fetchone()
vehicle_cursor.close()

if not vehicle:
    raise VehicleNotFoundException("Vehicle not found.")

hours = (end_date - start_date).total_seconds() / 3600
if hours <= 0:
    raise ReservationException("Invalid time range.")

hourly_rate = float(vehicle["daily_rate"]) / 24
subtotal = hours * hourly_rate
tax = 0.18 * subtotal
total_cost = subtotal + tax

cursor = self.con.cursor()
cursor.execute("""
    INSERT INTO reservation (customer_id, vehicle_id, start_date, end_date,
total_cost, status)
    VALUES (%s, %s, %s, %s, %s, %s)
""", (customer_id, vehicle_id, start_date, end_date, total_cost, status))
self.con.commit()

```

```

        print(f'Reservation created successfully. ID: {cursor.lastrowid}')
        return cursor.lastrowid

    except Error as e:
        self.con.rollback()
        raise
    finally:
        if cursor:
            cursor.close()

    def cancelReservation(self, reservation_id: int) -> float:
        reservation = self.getReservationById(reservation_id)
        if not reservation:
            raise ReservationException("Reservation not found.")

        days_left = (reservation.start_date.date() - datetime.now().date()).days
        fee = 0
        if days_left < 1:
            fee = float(reservation.total_cost) * 0.2
            print(f'Late cancellation fee: ₹{fee:.2f}')

        cursor = self.con.cursor()
        cursor.execute("UPDATE reservation SET status = 'cancelled' WHERE reservation_id = %s", (reservation_id,))
        self.con.commit()
        cursor.close()
        return fee

    def getReservationById(self, reservation_id):
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("SELECT * FROM reservation WHERE reservation_id = %s", (reservation_id,))
        data = cursor.fetchone()
        cursor.close()
        if data:
            return Reservation(**data)
        return None

    def getReservationsByCustomerId(self, customer_id):
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("SELECT * FROM reservation WHERE customer_id = %s", (customer_id,))
        reservations = [Reservation(**row) for row in cursor.fetchall()]
        cursor.close()
        return reservations

```

```

def updateReservation(self, reservation_data):
    cursor = None
    try:
        reservation_id = reservation_data["reservation_id"]
        reservation = self.getReservationById(reservation_id)
        if not reservation:
            raise ReservationException("Reservation not found.")

        cursor = self.con.cursor()
        cursor.execute("""
            UPDATE reservation SET
                start_date = %s,
                end_date = %s,
                status = %s
            WHERE reservation_id = %s
        """, (
            reservation_data.get("start_date", reservation.start_date),
            reservation_data.get("end_date", reservation.end_date),
            reservation_data.get("status", reservation.status),
            reservation_id
        ))
        self.con.commit()
        print("Reservation updated.")
        return True
    except Error as e:
        self.con.rollback()
        raise
    finally:
        if cursor:
            cursor.close()

def getAvailableVehiclesForDates(self, start_date, end_date):
    cursor = self.con.cursor(dictionary=True)
    query = """
        SELECT * FROM vehicle
        WHERE vehicle_id NOT IN (
            SELECT vehicle_id FROM reservation
            WHERE status = 'confirmed'
            AND (start_date < %s AND end_date > %s)
        )
    """
    cursor.execute(query, (end_date, start_date))
    available = cursor.fetchall()
    cursor.close()
    return available

```

```

def getAllReservations(self):
    cursor = self.con.cursor(dictionary=True)
    cursor.execute("SELECT * FROM reservation")
    data = cursor.fetchall()
    cursor.close()
    return data

```

- **IAdminService:**
 - GetAdminById(adminId)
 - GetAdminByUsername(username)
 - RegisterAdmin(adminData)
 - UpdateAdmin(adminData)
 - DeleteAdmin(adminId)

AdminService.py

```

from Exceptions import AdminNotFoundException, AuthenticationException
from service.IAdminService import IAdminService
from entity.Admin import Admin
from mysql.connector import connect, Error
import bcrypt

```

```

class AdminService(IAdminService):

```

```

    def __init__(self):
        self.con = self._get_connection()

```

```

    def _get_connection(self):
        try:
            return connect(
                host="localhost",
                user="root",
                password="Zuhi743#",
                database="carconnect",
                port=3306
            )
        except Error as e:
            print("Database connection failed:", e)
            raise

```

```

    def registerAdmin(self, admin_data):
        cursor = None

```

```

try:
    cursor = self.con.cursor()
    insert_query = """
        INSERT INTO admin (first_name, last_name, email, phone_number, username,
password, role, join_date)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
    """

    hashed_password = bcrypt.hashpw(admin_data["password"].encode(),
bcrypt.gensalt()).decode()
    cursor.execute(insert_query, (
        admin_data["first_name"],
        admin_data["last_name"],
        admin_data["email"],
        admin_data["phone_number"],
        admin_data["username"],
        hashed_password,
        admin_data["role"],
        admin_data["join_date"]
    ))
    self.con.commit()
    print("Admin registered successfully.")
    return True
except Error as e:
    self.con.rollback()
    print("Database Error:", e)
    return False
finally:
    if cursor:
        cursor.close()

def getAdminByUsername(self, username):
    cursor = None
    try:
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("SELECT * FROM admin WHERE username = %s", (username,))
        admin_data = cursor.fetchone()
        if admin_data:
            return Admin(
                admin_id=admin_data["admin_id"],
                first_name=admin_data["first_name"],
                last_name=admin_data["last_name"],
                email=admin_data["email"],
                phone_number=admin_data["phone_number"],
                username=admin_data["username"],
                password=admin_data["password"],
                role=admin_data["role"],

```

```

        join_date=admin_data["join_date"]
    )
    else:
        return None
except Error as e:
    print("Database error:", e)
    raise
finally:
    if cursor:
        cursor.close()

def getAdminById(self, admin_id):
    cursor = None
    try:
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("SELECT * FROM admin WHERE admin_id = %s", (admin_id,))
        admin_data = cursor.fetchone()
        if admin_data:
            return Admin(
                admin_id=admin_data["admin_id"],
                first_name=admin_data["first_name"],
                last_name=admin_data["last_name"],
                email=admin_data["email"],
                phone_number=admin_data["phone_number"],
                username=admin_data["username"],
                password=admin_data["password"],
                role=admin_data["role"],
                join_date=admin_data["join_date"]
            )
        else:
            return None
    except Error as e:
        print("Database error:", e)
        raise
    finally:
        if cursor:
            cursor.close()

def updateAdmin(self, admin_data):
    cursor = None
    try:
        admin_id = admin_data.get("admin_id")
        if not admin_id:
            print("Admin ID is required.")
            return False
        existing_admin = self.getAdminById(admin_id)

```

```

    if not existing_admin:
        print("Admin not found.")
        return False
    new_password = admin_data.get("password", existing_admin.password)
    if new_password != existing_admin.password:
        new_password = bcrypt.hashpw(new_password.encode(),
bcrypt.gensalt()).decode()
    cursor = self.con.cursor()
    update_query = """
        UPDATE admin SET
            first_name = %s,
            last_name = %s,
            email = %s,
            phone_number = %s,
            username = %s,
            password = %s,
            role = %s,
            join_date = %s
        WHERE admin_id = %s
    """
    cursor.execute(update_query, (
        admin_data.get("first_name", existing_admin.first_name),
        admin_data.get("last_name", existing_admin.last_name),
        admin_data.get("email", existing_admin.email),
        admin_data.get("phone_number", existing_admin.phone_number),
        admin_data.get("username", existing_admin.username),
        new_password,
        admin_data.get("role", existing_admin.role),
        admin_data.get("join_date", existing_admin.join_date),
        admin_id
    ))
    self.con.commit()
    print("Admin updated successfully.")
    return True
except Error as e:
    self.con.rollback()
    print("Database error:", e)
    return False
finally:
    if cursor:
        cursor.close()

def deleteAdmin(self, admin_id):
    cursor = None
    try:
        cursor = self.con.cursor()

```

```

        cursor.execute("SELECT admin_id FROM admin WHERE admin_id = %s",
(admin_id,))
        if not cursor.fetchone():
            print("Admin not found.")
            return False
        cursor.execute("DELETE FROM admin WHERE admin_id = %s", (admin_id,))
        self.con.commit()
        print(f"Admin ID {admin_id} deleted successfully.")
        return True
    except Error as e:
        self.con.rollback()
        print("Database error:", e)
        return False
    finally:
        if cursor:
            cursor.close()

```

```

def migrate_admin_passwords(self):
    cursor = None
    try:
        cursor = self.con.cursor(dictionary=True)
        cursor.execute("SELECT admin_id, password FROM admin")
        admins = cursor.fetchall()
        for admin in admins:
            pwd = admin["password"]
            if pwd.startswith("$2b$"):
                continue
            hashed = bcrypt.hashpw(pwd.encode(), bcrypt.gensalt()).decode()
            cursor.execute(
                "UPDATE admin SET password=%s WHERE admin_id=%s",
                (hashed, admin["admin_id"])
            )
        self.con.commit()
        print(f"Updated {len(admins)} admin passwords to hashed format.")
    except Error as e:
        self.con.rollback()
        print(f"Migration failed: {e}")
        raise
    finally:
        if cursor:
            cursor.close()

```


Connect your application to the SQL database:

- Create a connection string that includes the necessary information to connect to your SQL Server database. This includes the server name, database name, authentication credentials, and any other relevant settings.
- Use the **SqlConnection** class to establish a connection to the SQL Server database.
- Once the connection is open, you can use the **SqlCommand** class to execute SQL queries.

```
def _get_connection(self):
    try:
        return connect(
            host="localhost",
            user="root",
            password="Zuhi743#",
            database="carconnect",
            port=3306
        )
    except Error as e:
        print("Database connection failed:", e)
        raise
```

EXCEPTIONS

AuthenticationException:

- Thrown when there is an issue with user authentication.
- Example Usage: Incorrect username or password during customer or admin login.

ReservationException:

- Thrown when there is an issue with reservations.
- Example Usage: Attempting to make a reservation for a vehicle that is already reserved.

VehicleNotFoundException:

- Thrown when a requested vehicle is not found.
- Example Usage: Trying to get details of a vehicle that does not exist.

AdminNotFoundException:

- Thrown when an admin user is not found.
- Example Usage: Attempting to access details of an admin that does not exist.

InvalidInputException:

- Thrown when there is invalid input data.
- Example Usage: When a required field is missing or has an incorrect format.

Exception.py

```
class AuthenticationException(Exception):
```

```
    def __init__(self, message="Invalid credentials"):
        self.message = message
        super().__init__(self.message)
```

```
class InvalidEmailException(Exception):
```

```
    def __init__(self, message="Email must be a valid @gmail.com address"):
        self.message = message
        super().__init__(self.message)
```

```
class WeakPasswordException(Exception):
```

```
    def __init__(self, message="Password must be at least 8 characters with one special
character"):
        self.message = message
        super().__init__(self.message)
```

```
class InvalidPhoneNumberException(Exception):
```

```
    def __init__(self, message="Phone number must be valid"):
        self.message = message
        super().__init__(self.message)
```

```
class CustomerNotFoundException(Exception):
```

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

```
class DuplicateCustomerException(Exception):
```

```
    def __init__(self, message="Customer ID already exists"):
        self.message = message
        super().__init__(self.message)
```

```
class VehicleNotFoundException(Exception):
```

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

```
class DuplicateVehicleException(Exception):
```

```
    def __init__(self, message="Vehicle ID already exists"):
        self.message = message
        super().__init__(self.message)
```

```
class InvalidYearException(Exception):
```

```
    def __init__(self, message="Year must be between 1900 and current year"):
        super().__init__(message)
```

```
class InvalidRegNumberException(Exception):
```

```
    def __init__(self, message="Registration number must be in format ABC-1234"):
        super().__init__(message)
```

```
class InvalidDailyRateException(Exception):
```

```
    def __init__(self, message="Daily rate must be positive"):
        super().__init__(message)
```

```
class OverlappingBookingException(Exception):
```

```
    def __init__(self, message="Vehicle already booked for selected dates"):
        super().__init__(message)
```

```
class ReservationException(Exception):
```

```
    def __init__(self, message="Reservation conflict or invalid"):
        self.message = message
        super().__init__(self.message)
```

```
class AdminNotFoundException(Exception):
```

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

```
class InvalidInputException(Exception):
```

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

```
class DatabaseConnectionException(Exception):
```

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

TESTING

```
import pytest
from service.CustomerService import CustomerService
from service.VehicleService import VehicleService
from Exceptions import (
    AuthenticationException
)
```

```
@pytest.fixture
def customer_service():
    service = CustomerService()
    try:
        yield service
    finally:
        service.close()
```

```
@pytest.fixture
def sample_customer_data():
    return {
        "first_name": "Haritha",
        "last_name": "Hari",
        "email": "harithah.2020@gmail.com",
        "phone_number": "1234567890",
        "address": "123 Test St",
        "username": "Haritha",
        "password": "Haritha@2003"
    }
```

```
def test_invalid_credentials(customer_service, sample_customer_data):

    customer_service.registerCustomer(sample_customer_data)

    try:
        customer_service.authenticate_customer(
            sample_customer_data["username"],
            "wrong"
        )
        assert False, "Authentication failed with wrong password"
    except AuthenticationException:
        assert True
```

```

try:
    customer_service.authenticate_customer(
        "Hari",
        sample_customer_data["password"]
    )
    assert False, "Authentication failed with wrong username"
except AuthenticationException:
    assert True

```

```

def test_update_customer_information(customer_service, sample_customer_data):

```

```

    customer =
customer_service.getCustomerByUsername(sample_customer_data["username"])

    updates = {
        "customer_id": customer.customer_id,
        "phone_number": "9876543210",
        "address": "456 Updated St",
        "password": "Haritha@2003"
    }
    result = customer_service.updateCustomer(updates)

    assert result is True

    updated_customer = customer_service.getCustomerById(customer.customer_id)
    assert updated_customer.phone_number == "9876543210"
    assert updated_customer.address == "456 Updated St"

```

```

@pytest.fixture
def vehicle_service():
    service = VehicleService()
    try:
        yield service
    finally:
        service.close()

```

```

@pytest.fixture
def sample_vehicle_data():
    return {
        "Model": "Hero",
        "Make": "Honda",
        "Year": "2023",
        "color": "Black",
        "registration_number": "DL-1122",
    }

```

```
    "daily_rate": 1800.00,  
    "Availability": True  
}
```

```
def test_add_new_vehicle(vehicle_service, sample_vehicle_data):
```

```
    result = vehicle_service.AddVehicle(sample_vehicle_data)  
    assert result is True  
    vehicles = vehicle_service.GetAvailableVehicles(1)
```

```
    test_vehicle = None  
    for v in vehicles:  
        if v.registration_number == sample_vehicle_data["registration_number"]:  
            test_vehicle = v  
            break
```

```
    assert test_vehicle is not None, "Added vehicle not found in available vehicles"
```

```
    if test_vehicle:  
        vehicle_service.RemoveVehicle(test_vehicle.vehicle_id)
```

```
def test_get_available_vehicles(vehicle_service, sample_vehicle_data):
```

```
    available_vehicles = vehicle_service.GetAvailableVehicles(1)  
    print("Available Vehicles:")  
    for i, vehicle in enumerate(available_vehicles, 1):  
        print(f'{i}. Model- {vehicle.Model}, Make- {vehicle.Make}, "  
            f'Availability- {vehicle.Availability}')
```

```
    assert isinstance(available_vehicles, list)  
    assert len(available_vehicles) > 0  
    assert all(v.Availability == 1 for v in available_vehicles)
```

MAIN FUNCTION

```
from datetime import datetime
from service.CustomerService import CustomerService
from service.VehicleService import VehicleService
from service.AdminService import AdminService
from service.ReservationService import ReservationService
from Exceptions import *

def main_menu():
    print("Welcome to CarConnect - A Car Rental Platform!")
    print("1.Customer Portal")
    print("2.Admin Portal")
    print("3.Exit")
    return int(input("Enter your choice 1-3: "))

def customer_portal(c_service, v_service, r_service):
    print("Customer Portal")
    print("1.Register")
    print("2.Login")
    print("3.Main Menu")
    choice = int(input("Enter your choice 1-3: "))
    if choice == 1:
        register_customer(c_service)
    elif choice == 2:
        customer = login_customer(c_service)
        if customer:
            customer_dashboard(customer, v_service, r_service)
    elif choice == 3:
        return
    else:
        print("Choose between 1-3")

def register_customer(c_service):
    c_data = {
        "first_name": input("First Name: "),
        "last_name": input("Last Name: "),
        "email": input("Email: "),
        "phone_number": input("Phone number: "),
        "address": input("Address: "),
        "username": input("Username: "),
        "password": input("Password: ")
    }
    try:
        c_service.registerCustomer(c_data)
        print("Registration successful!")
```



```
except Exception as e:  
    print(f"Registration failed: {e}")
```

```
def login_customer(c_service):  
    username = input("Username: ")  
    password = input("Password: ")  
    try:  
        if c_service.authenticate_customer(username, password):  
            return c_service.getCustomerByUsername(username)  
    except AuthenticationException:  
        print("Invalid username or password.")  
    return None
```

```
def customer_dashboard(customer, v_service, r_service):  
    while True:  
        print(f"\nWelcome {customer.first_name}")  
        print("1.View Available Vehicles")  
        print("2.Make Reservation")  
        print("3.My Reservations")  
        print("4.Cancel Reservation")  
        print("5.Logout")  
        choice = int(input("Enter your choice 1-5: "))  
        if choice == 1:  
            view_available_vehicles(v_service, r_service)  
        elif choice == 2:  
            make_reservation(customer, v_service, r_service)  
        elif choice == 3:  
            my_reservations(customer, r_service)  
        elif choice == 4:  
            cancel_reservation(customer, r_service)  
        elif choice == 5:  
            print("Logged out.")  
            break  
        else:  
            print("Choose between 1-5")
```

```
def view_available_vehicles(v_service, r_service):  
    print("\nAvailable Vehicles:")  
    try:  
        vehicles = v_service.GetAvailableVehicles(1)  
        available_found = False  
        for v in vehicles:  
            reservations = r_service.getAllReservations()  
            is_reserved = False  
            for res in reservations:  
                if res["vehicle_id"] == v.vehicle_id and res["status"] == "confirmed":  
                    is_reserved = True  
                    break
```

```

        if not is_reserved:
            available_found = True
            print(f'{v.vehicle_id}. {v.Make} {v.Model} ({v.color}) - ₹{v.daily_rate}/day")
        if not available_found:
            print("No available vehicles found.")
    except Exception as e:
        print(f'Error: {e}')

def make_reservation(customer, v_service, r_service):
    print("\nMake a Reservation")
    try:
        start = input("Enter start datetime (YYYY-MM-DD HH:MM): ")
        end = input("Enter end datetime (YYYY-MM-DD HH:MM): ")

        start_date = datetime.strptime(start, "%Y-%m-%d %H:%M")
        end_date = datetime.strptime(end, "%Y-%m-%d %H:%M")

        vehicles = r_service.getAvailableVehiclesForDates(start_date, end_date)
        if not vehicles:
            print("No vehicles available for your selected time range.")
            return

        print("\nVehicles available for your dates:")
        for v in vehicles:
            print(f'{v["vehicle_id"]} - {v["Make"]} {v["Model"]} ({v["color"]}) - ₹{v["daily_rate"]}/day")

        vehicle_id = int(input("Enter vehicle ID to reserve from above list: "))

        reservation_data = {
            "customer_id": customer.customer_id,
            "vehicle_id": vehicle_id,
            "start_date": start_date,
            "end_date": end_date
        }
        reservation_id = r_service.createReservation(reservation_data)
        reservation = r_service.getReservationById(reservation_id)
        if reservation:
            print(reservation.generate_invoice())
        else:
            print("Reservation created but not found afterwards.")
    except Exception as e:
        print(f'Reservation failed: {e}')

def my_reservations(customer, r_service):
    print("\nMy Reservations")
    try:
        reservations = r_service.getReservationsByCustomerId(customer.customer_id)
        for res in reservations:
            print(f'{res.reservation_id} - {res.vehicle_id} - {res.start_date} to {res.end_date} - ₹{res.total_cost:.2f} ({res.status})")
    except Exception as e:

```

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

```
def cancel_reservation(customer, r_service):  
    my_reservations(customer, r_service)  
    try:  
        res_id = int(input("Enter reservation ID to cancel: "))  
        fee = r_service.cancelReservation(res_id)  
        print(f'Reservation cancelled. Cancellation fee: ₹ {fee:.2f}')  
    except Exception as e:  
        print(f'Cancellation failed: {e}')
```

```
def admin_portal(a_service, v_service, r_service):  
    print("\nAdmin Portal")  
    print("1.Login")  
    print("2.Main Menu")  
    choice = int(input("Enter your choice 1-2: "))  
    if choice == 1:  
        admin = login_admin(a_service)  
        if admin:  
            admin_dashboard(admin, v_service, r_service)  
    elif choice == 2:  
        return  
    else:  
        print("Choose between 1-2")
```

```
def login_admin(a_service):  
    username = input("Admin Username: ")  
    password = input("Password: ")  
    try:  
        admin = a_service.getAdminByUsername(username)  
        if admin and admin.authenticate(password):  
            return admin  
    except Exception as e:  
        print(f'Login failed: {e}')  
    return None
```

```
def admin_dashboard(admin, v_service, r_service):  
    while True:  
        print(f'\nAdmin Dashboard ({admin.role})')  
        print("1.Add Vehicle")  
        print("2.View Vehicles")  
        print("3.Update Vehicle")  
        print("4.Remove Vehicle")  
        print("5.View Reservations")  
        print("6.Logout")  
        choice = int(input("Enter your choice 1-6: "))  
        if choice == 1:
```

```

        add_vehicle(v_service)
    elif choice == 2:
        view_all_vehicles(v_service)
    elif choice == 3:
        update_vehicle(v_service)
    elif choice == 4:
        remove_vehicle(v_service)
    elif choice == 5:
        view_all_reservations(r_service)
    elif choice == 6:
        print("Admin logged out.")
        break
    else:
        print("Choose between 1-6")

```

```

def add_vehicle(v_service):
    v_data = {
        "Model": input("Model: "),
        "Make": input("Make: "),
        "Year": input("Year: "),
        "color": input("Color: "),
        "registration_number": input("Reg No: "),
        "daily_rate": float(input("Daily Rate: ")),
        "Availability": 1
    }
    try:
        v_service.AddVehicle(v_data)
        print("Vehicle added.")
    except Exception as e:
        print(f"Add failed: {e}")

```

```

def view_all_vehicles(v_service):
    try:
        vehicles = v_service.GetAllVehicles(1)
        for v in vehicles:
            status = "Available" if v.Availability else "Booked"
            print(f"{v.vehicle_id} - {v.Make} {v.Model} ({status})")
    except Exception as e:
        print(f"Error: {e}")

```

```

def update_vehicle(v_service):
    view_all_vehicles(v_service)
    try:
        vehicle_id = int(input("Vehicle ID to update: "))
        v_data = {
            "vehicle_id": vehicle_id,
            "Model": input("New Model: "),
            "Make": input("New Make: "),

```

```

        "Year": input("New Year: "),
        "color": input("New Color: "),
        "registration_number": input("New Reg No: "),
        "daily_rate": input("New Daily Rate: ")
    }
    v_service.UpdateVehicle(v_data)
    print("Vehicle updated.")
except Exception as e:
    print(f"Update failed: {e}")

```

```

def remove_vehicle(v_service):
    view_all_vehicles(v_service)
    try:
        vehicle_id = int(input("Vehicle ID to remove: "))
        v_service.RemoveVehicle(vehicle_id)
        print("Vehicle removed.")
    except Exception as e:
        print(f"Removal failed: {e}")

```

```

def view_all_reservations(r_service):
    try:
        reservations = r_service.getAllReservations()
        for res in reservations:
            print(f'{res["reservation_id"]} - {res["customer_id"]} - {res["vehicle_id"]} - {res["status"]}')
    except Exception as e:
        print(f"Error: {e}")

```

```

def main():
    c_service = CustomerService()
    v_service = VehicleService()
    a_service = AdminService()
    r_service = ReservationService()

    while True:
        choice = main_menu()
        if choice == 1:
            customer_portal(c_service, v_service, r_service)
        elif choice == 2:
            admin_portal(a_service, v_service, r_service)
        elif choice == 3:
            print("Thank you for using CarConnect!")
            break
        else:
            print("Invalid choice, please retry.")

```

```

if __name__ == "__main__":
    main()

```

CONSOLE APPLICATION OUTPUT

Customer Login

```
D:\Python_Project\venv\Scripts\python.exe D:\Python_Project\mainfunc.py
Welcome to CarConnect - A Car Rental Platform!
1.Customer Portal
2.Admin Portal
3.Exit
Enter your choice 1-3:1
Customer-Portal
1.Are you new to CarConnect?-Register
2.Login
3.Mainmenu
Enter your choice 1-3:2
Customer Login
Username:alexj
Password:johnson@123
Authentication successfull
```

New Customer Registration

```
D:\Python_Project\venv\Scripts\python.exe D:\Python_Project\mainfunc.py
Welcome to CarConnect - A Car Rental Platform!
1.Customer Portal
2.Admin Portal
3.Exit
Enter your choice 1-3:1
Customer-Portal
1.Are you new to CarConnect?-Register
2.Login
3.Mainmenu
Enter your choice 1-3:1
Register your details
First Name:Yamuna
Last Name:Muthumanickam
Email:yamu873@gmail.com
Phone number:8790561234
Address:Thozhilpettai
Username:yamu
Password:Yamu@123
Customer 8 registered successfully!
Registered Sucessfully,Now you can login with your credentials
```

Viewing all Vehicles

```
Enter your choice 1-6: 2
1 - Maruti Suzuki Swift (Available)
2 - Hyundai Creta (Booked)
3 - Tata Nexon EV (Available)
4 - Toyota Innova Crysta (Available)
5 - Maruti Suzuki WagonR (Available)
6 - Hyundai Verna (Available)
7 - Camry Toyota (Available)
9 - Skoda Kodiaq (Available)
10 - Mahindra Thar (Available)
```

Making Reservation

```
Enter your choice 1-5:2
Make a Reservation
Enter start datetime (YYYY-MM-DD HH:MM): 2025-07-20 09:00
Enter end datetime (YYYY-MM-DD HH:MM): 2025-07-22 09:00

Vehicles available for your dates:
1 - Maruti Suzuki Swift (Pearl White) - 1500.00/day
4 - Toyota Innova Crysta (Silver) - 4000.00/day
5 - Maruti Suzuki WagonR (Red) - 1200.00/day
6 - Hyundai Verna (White) - 1600.00/day
```

Cancelling Reservation

```
Viewing Customer Dashboard
1.View Available Vehicles
2.Make Reservation
3.My Reservation
4.Cancel Reservation
5.Logout
Enter your choice 1-5:4
Your Reservations
Reservation ID: 6
Vehicle ID: 5, Dates: 2025-07-25 08:00:00 to 2025-07-27 08:00:00
Status: completed, Total Cost: 2832.00

Reservation ID: 9
Vehicle ID: 3, Dates: 2025-07-10 00:00:00 to 2025-07-12 00:00:00
Status: confirmed, Total Cost: 6608.00

Enter reservation ID to cancel: 9
Reservation cancelled. Cancellation fee: 0.00
```

Admin Login

```
Logged out
Welcome to CarConnect - A Car Rental Platform!
1.Customer Portal
2.Admin Portal
3.Exit
Enter your choice 1-3:2
Admin Portal
1.Login
2.Mainmenu
Enter your choice 1-2: 1
Admin Login
Username: admin1
Password: Admin@123
Admin Dashboard (superadmin)
```

Adding Vehicle

```
Admin Dashboard (superadmin)
1.Add New Vehicle
2.View All Vehicles
3.Update Vehicle
4.Remove Vehicle
5.View All Reservations
6.Logout
Enter your choice 1-6: 1
Add New Vehicle
Model: Kodiah
Make: Skoda
Year: 2023
Color: Black
Registration Number (example: TN-1234): HR-9999
Daily Rate: 3500.00
Availability(0-Rented,1-Available):1
Vehicle 7 registered successfully
Vehicle added successfully!
```


Vehicle Updation

```
Enter vehicle ID to update: 6
Leave blank for no changes
New Model:
New Make:
New Year:
New Color: Black
New Registration Number:
New Daily Rate:
Vehicle Updated Successfully
```

Displaying All Reservations

```
All Reservations
Reservation ID: 1
Customer ID: 1, Vehicle ID: 1
Dates: 2025-07-01 10:00:00 to 2025-07-05 10:00:00
Status: confirmed, Total:7080.00

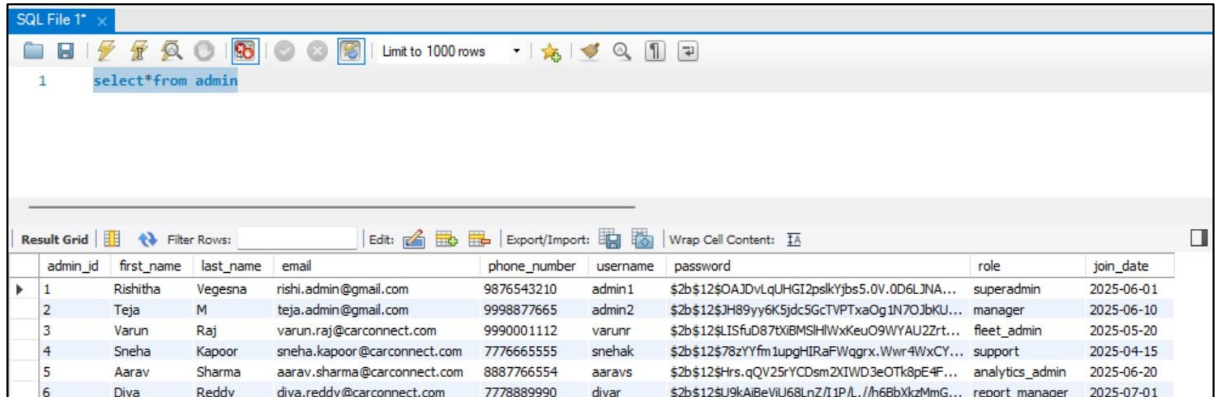
Reservation ID: 2
Customer ID: 2, Vehicle ID: 3
Dates: 2025-07-06 09:00:00 to 2025-07-08 09:00:00
Status: completed, Total:6608.00

Reservation ID: 3
Customer ID: 3, Vehicle ID: 5
Dates: 2025-07-10 11:00:00 to 2025-07-11 11:00:00
Status: cancelled, Total:1416.00

Reservation ID: 4
Customer ID: 4, Vehicle ID: 4
Dates: 2025-07-12 10:00:00 to 2025-07-15 10:00:00
Status: confirmed, Total:14160.00
```

Database Outputs

Admin Table:



SQL File 1* x

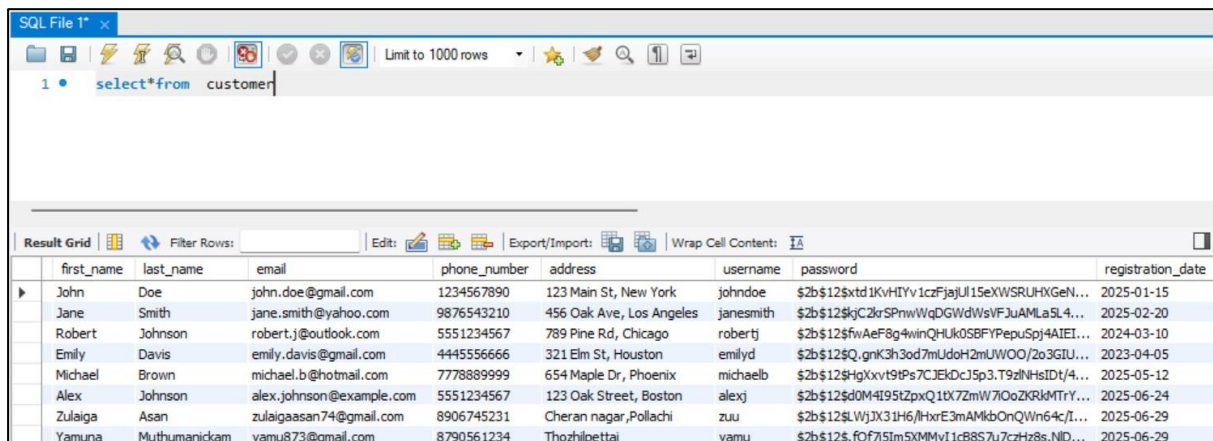
Limit to 1000 rows

1 select * from admin

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	admin_id	first_name	last_name	email	phone_number	username	password	role	join_date
▶	1	Rishitha	Vegesna	rishi.admin@gmail.com	9876543210	admin1	\$2b\$12\$0AJDvLqUHG12pskYjbs5.0V.0D6LJNA...	superadmin	2025-06-01
	2	Teja	M	teja.admin@gmail.com	9998877665	admin2	\$2b\$12\$3H89yy6K5jdc5GcTVPTxaOg1N70JbKU...	manager	2025-06-10
	3	Varun	Raj	varun.raj@carconnect.com	9990001112	varunr	\$2b\$12\$ISfUD87xIBMSHfWxKeuO9WYAUZ2rt...	fleet_admin	2025-05-20
	4	Sneha	Kapoor	sneha.kapoor@carconnect.com	7776665555	snehak	\$2b\$12\$78zYfjm1upgHIRaFWqgrx.Wwr4WxCY...	support	2025-04-15
	5	Aarav	Sharma	aarav.sharma@carconnect.com	8887766554	aaravs	\$2b\$12\$Hrs.qQV25rYCDsm2XtWD3eOTk8p4F...	analytics_admin	2025-06-20
	6	Diva	Reddy	diva.reddy@carconnect.com	7778889990	divar	\$2b\$12\$I9kAReVIL68LnZ7I1P1A.../h6BbXkzMMG...	report_manager	2025-07-01

Customer Table



SQL File 1* x

Limit to 1000 rows

1 • select * from customer

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	first_name	last_name	email	phone_number	address	username	password	registration_date
▶	John	Doe	john.doe@gmail.com	1234567890	123 Main St, New York	johndoe	\$2b\$12\$xttd1KvHTYv1czFajUl15eXWSRUHXGeN...	2025-01-15
	Jane	Smith	jane.smith@yahoo.com	9876543210	456 Oak Ave, Los Angeles	janesmith	\$2b\$12\$gC2krSPnwWqDGWdWsfJuAMLaSL4...	2025-02-20
	Robert	Johnson	robert.j@outlook.com	5551234567	789 Pine Rd, Chicago	robertj	\$2b\$12\$fwAeF8g4winQHUKoSBFYpPuSpj4AIEI...	2024-03-10
	Emily	Davis	emily.davis@gmail.com	4445556666	321 Elm St, Houston	emilyd	\$2b\$12\$Q.gnK3h3od7mUdoH2mUWOO/zo3GIU...	2023-04-05
	Michael	Brown	michael.b@hotmail.com	7778889999	654 Maple Dr, Phoenix	michaelb	\$2b\$12\$H9Xcvt9tPs7CJEKdcJ5p3.T9zINHsIDt/4...	2025-05-12
	Alex	Johnson	alex.johnson@example.com	5551234567	123 Oak Street, Boston	alexj	\$2b\$12\$d0M4t95tZpxQ1tX7ZmW7iOoZKRkMTry...	2025-06-24
	Zulaiga	Asan	zulaigaasan74@gmail.com	8906745231	Cheran nagar, Pollachi	zuu	\$2b\$12\$LWjJX31H6/lHxrE3mAMkbOnQWn64c/I...	2025-06-29
	Yamuna	Muthumanickam	yamu873@gmail.com	8790561234	Thozhilpettai	yamu	\$2b\$12\$.fQf7l5lm5XMMVyl1cB8S7u7czHz8s.NID...	2025-06-29

Reservation Table:

SQL File 1* x

Limit to 1000 rows

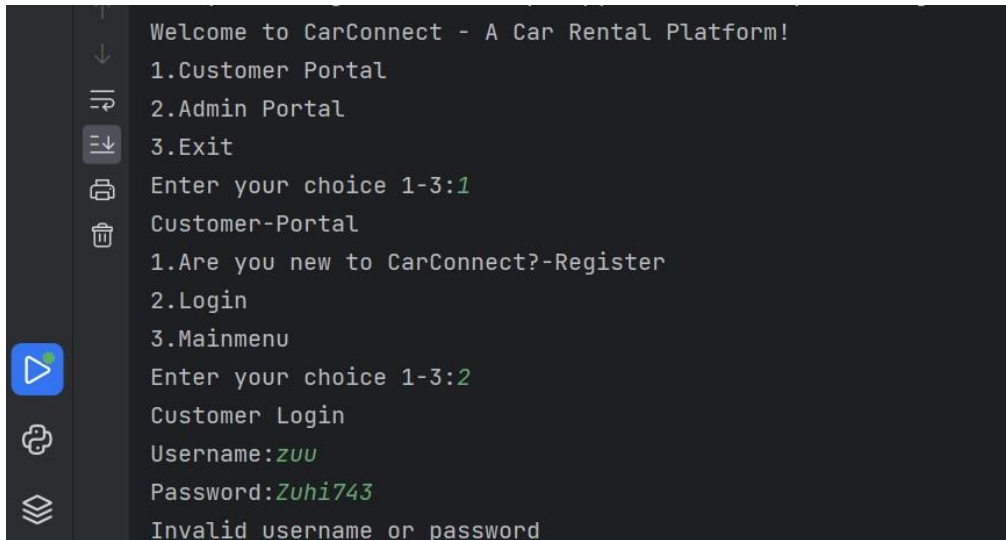
1 • `select*from reservation`

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:

	reservation_id	customer_id	vehicle_id	start_date	end_date	total_cost	status
	2	2	3	2025-07-06 09:00:00	2025-07-08 09:00:00	6608.00	completed
	3	3	5	2025-07-10 11:00:00	2025-07-11 11:00:00	1416.00	cancelled
	4	4	4	2025-07-12 10:00:00	2025-07-15 10:00:00	14160.00	confirmed
	5	5	1	2025-07-20 09:00:00	2025-07-22 09:00:00	3540.00	cancelled
	6	6	5	2025-07-25 08:00:00	2025-07-27 08:00:00	2832.00	completed
	7	7	2	2025-07-20 09:00:00	2025-07-22 09:00:00	8260.00	confirmed
	8	6	3	2025-07-20 09:00:00	2025-07-22 09:00:00	6608.00	confirmed

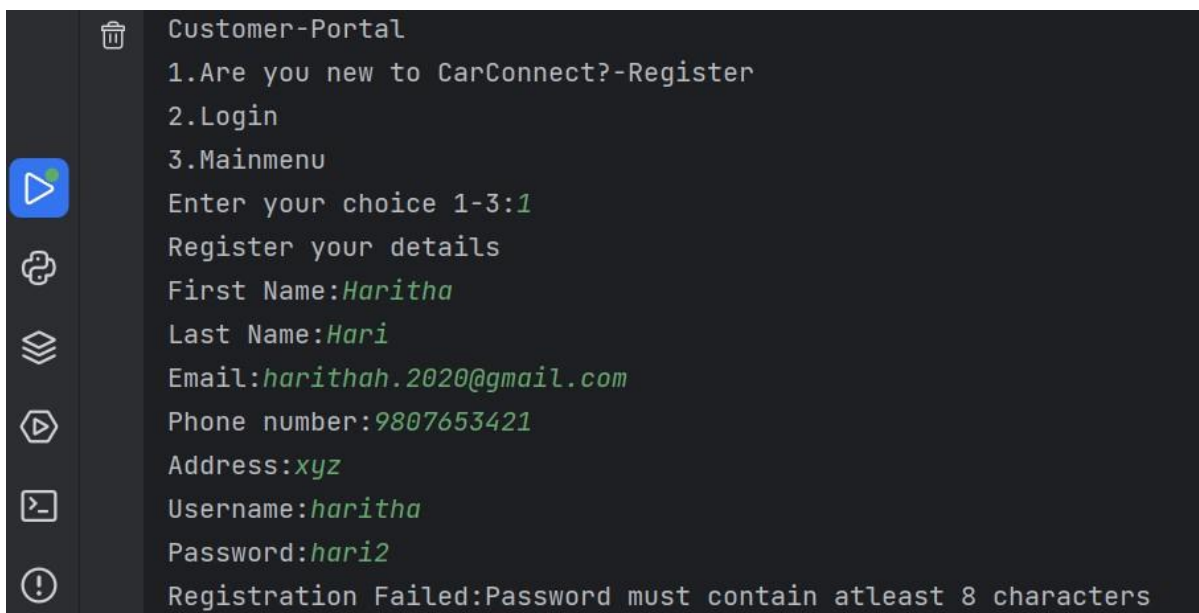
EXCEPTIONS RESULTS

Invalid Username/Password – Authentication Exception



```
Welcome to CarConnect - A Car Rental Platform!
1.Customer Portal
2.Admin Portal
3.Exit
Enter your choice 1-3:1
Customer-Portal
1.Are you new to CarConnect?-Register
2.Login
3.Mainmenu
Enter your choice 1-3:2
Customer Login
Username:zUU
Password:Zuhi743
Invalid username or password
```

Weak Password Exception



```
Customer-Portal
1.Are you new to CarConnect?-Register
2.Login
3.Mainmenu
Enter your choice 1-3:1
Register your details
First Name:Haritha
Last Name:Hari
Email:harithah.2020@gmail.com
Phone number:9807653421
Address:xyz
Username:haritha
Password:hari2
Registration Failed:Password must contain atleast 8 characters
```

Invalid Phonenumber Exception

```
Customer-Portal
1.Are you new to CarConnect?-Register
2.Login
3.Mainmenu
Enter your choice 1-3:1
Register your details
First Name:Haritha
Last Name:Hari
Email:harithah.2020@gmail.com
Phone number:1234567
Address:xyz
Username:haritha
Password:Haritha@2003
Registration Failed:Phone number must be valid
```

Invalid Registernumber Exception

```
1.Add New Vehicle
2.View All Vehicles
3.Update Vehicle
4.Remove Vehicle
5.View All Reservations
6.Logout
Enter your choice 1-6: 1
Add New Vehicle
Model: Verna
Make: Hyundai
Year: 2023
Color: Black
Registration Number (example: TN-1234): ABCD-54321
Daily Rate: 1200.00
Availability(0-Rented,1-Available):1
Failed to add vehicle: Registration number must be in format ABC-1234
```

Invalid Email Exception

```
Enter your choice 1-3:1
Customer-Portal
1.Are you new to CarConnect?-Register
2.Login
3.Mainmenu
Enter your choice 1-3:1
Register your details
First Name:Haritha
Last Name:Hari
Email:haritha2020gmailcom
Phone number:8976541234
Address:xyz
Username:haritha
Password:Haritha@2003
Registration Failed:Email must be a valid @gmail.com address
```

Vehicle Not Found Exception

```
5.Logout
Enter your choice 1-5:2
Make a Reservation
3 - Tata Nexon EV
5 - Maruti Suzuki WagonR
Enter vehicle ID to reserve: 9
Start (YYYY-MM-DD HH:MM): 2025-07-10 09:30
End (YYYY-MM-DD HH:MM): 2025-07-11 09:30
Reservation failed: Vehicle not found.
```

PYTEST RESULTS

Tested with invalid credentials

RunPython tests for test_carconnect.test_invalid_credentials

Tests passed: 1 of 1 test - 880 ms

Test Results880 ms

D:\Python_Project\venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2023.1.2/plugins/p

Testing started at 08:35 ...

Launching pytest with arguments test_carconnect.py::test_invalid_credentials --no-header --no-summary -q in D:\Pyt

===== test session starts =====

collecting ... collected 1 item

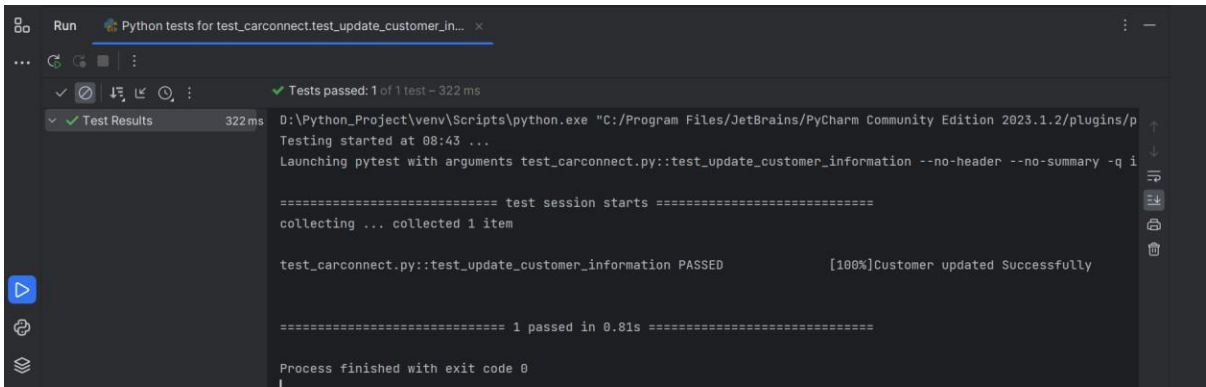
test_carconnect.py::test_invalid_credentials PASSED [100%]Customer 8 registered successfully!

===== 1 passed in 1.18s =====










Process finished with exit code 0

SQL File 1*									
Limit to 1000 rows									
1	select*from customer;								
2	-- select*from vehicle;								
3	-- select*from admin;								
4	-- select*from reservation;								
Result Grid									
	customer_id	first_name	last_name	email	phone_number	address	username	password	registra
3	Robert	Johnson	robert.j@outlook.com	5551234567	789 Pine Rd, Chicago	robertj	\$2b\$12\$nfQIn1f8piZz4614Jes5sextkd.qV/g.tG...	2024-03	
4	Emily	Davis	emily.davis@gmail.com	4445556666	321 Elm St, Houston	emilyd	\$2b\$12\$QxLFWgrHoWlXueKgU6wEObsh456C...	2023-04	
5	Michael	Brown	michael.b@hotmail.com	7778889999	654 Maple Dr, Phoenix	michaelb	\$2b\$12\$mrVm8On8CMxpWQWRGthObOjSSu...	2025-05	
6	Alex	Johnson	alex.johnson@example.com	5551234567	123 Oak Street, Boston	alexj	\$2b\$12\$I3K0xsiz4LDHJaaWUmNRXOyrHzLKmu...	2025-06	
7	Zulaiga	Asan	zulaigaasan74@gmail.com	9087654321	Cheran nagar,Pollachi	zuu	\$2b\$12\$Z1DmY29D6dwg1mYEnZ3yfuQ65p6Wx...	2025-06	
8	Haritha	Hari	harithah.2020@gmail.com	1234567890	123 Test St	Haritha	\$2b\$12\$WtufyzhDBmdLO.oBkbC4upEb/b1W1...	2025-06	






Tested by updated customer details



SQL File 1* x








Limit to 1000 rows






```
1 select*from customer;
2 -- select*from vehicle;
3 -- select*from admin;
4 -- select*from reservation;
```

Result Grid

 Filter Rows:

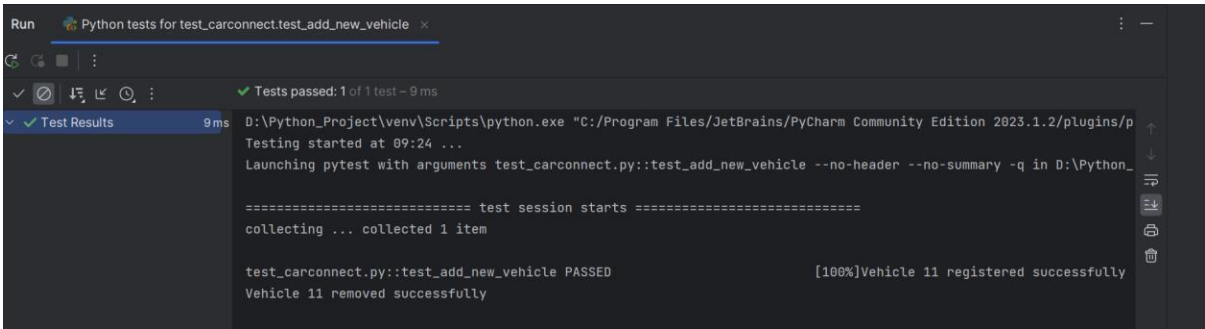
Edit: 

Export/Import: 

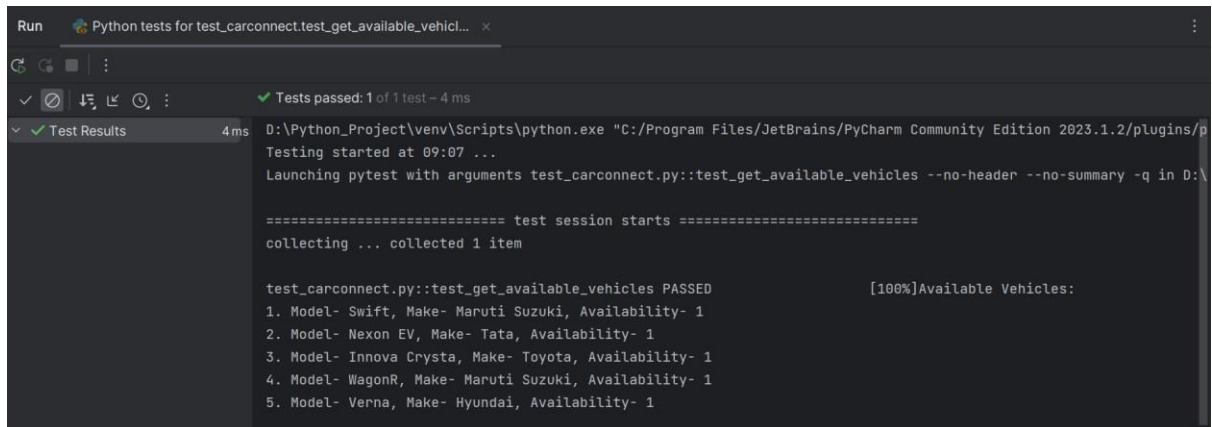
Wrap Cell Contents: 

	customer_id	first_name	last_name	email	phone_number	address	username	password	registra
3	Robert	Johnson	robert.j@outlook.com	5551234567	789 Pine Rd, Chicago	robertj	\$2b\$12\$nfQInI8pZz4614Ues5sextkd.qV/g.tG...	2024-03	
4	Emily	Davis	emily.davis@gmail.com	4445556666	321 Elm St, Houston	emilyd	\$2b\$12\$nQxLFWgrHoWLxueKgU6wEObsh456C...	2023-04	
5	Michael	Brown	michael.b@hotmail.com	7778889999	654 Maple Dr, Phoenix	michaelb	\$2b\$12\$mrVm8On8CMxpWwQWRGtjhObOjSSu...	2025-05	
6	Alex	Johnson	alex.johnson@example.com	5551234567	123 Oak Street, Boston	alexj	\$2b\$12\$I3K0xsiz4LDHJaaWUmNRXOyrHzLKmu...	2025-06	
7	Zulaiga	Asan	zulaigaasan74@gmail.com	9087654321	Cheran nagar, Pollachi	zuu	\$2b\$12\$Z1DmY29D6dwg1mYEnZ3yfuQ65p6Wx...	2025-06	
8	Haritha	Hari	harithah.2020@gmail.com	9876543210	456 Updated St	Haritha	\$2b\$12\$VpLE5z7TKqy0oS4ORBGBgeh/R9ndw2...	2025-06	

Tested adding new vehicle:



Tested getting available vehicle



The screenshot shows the PyCharm Run console with the following output:

```
Run Python tests for test_carconnect.test_get_available_vehicl... x
✓ Tests passed: 1 of 1 test - 4 ms
D:\Python_Project\venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2023.1.2/plugins/p
Testing started at 09:07 ...
Launching pytest with arguments test_carconnect.py::test_get_available_vehicles --no-header --no-summary -q in D:\

===== test session starts =====
collecting ... collected 1 item

test_carconnect.py::test_get_available_vehicles PASSED [100%]Available Vehicles:
1. Model- Swift, Make- Maruti Suzuki, Availability- 1
2. Model- Nexon EV, Make- Tata, Availability- 1
3. Model- Innova Crysta, Make- Toyota, Availability- 1
4. Model- WagonR, Make- Maruti Suzuki, Availability- 1
5. Model- Verna, Make- Hyundai, Availability- 1
```