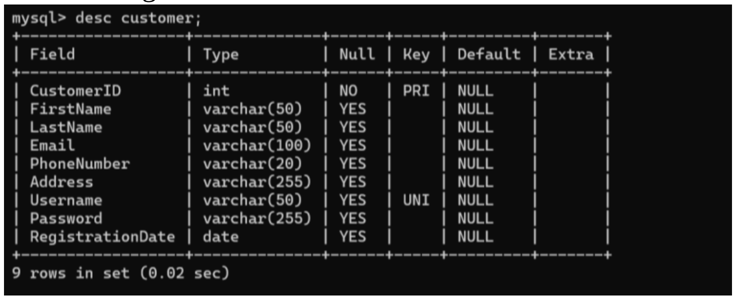
Casestudy-CarConnect

Name:Soundarya V

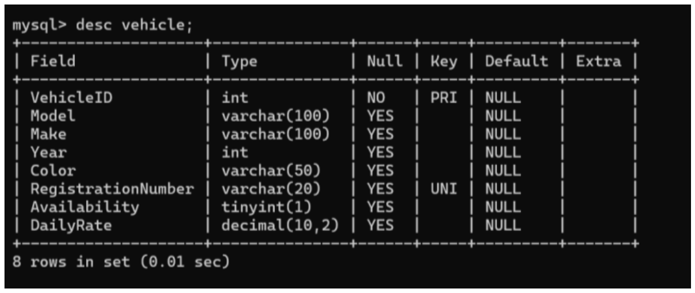
Create following tables in SQL Schema with appropriate class and write the unit test case for the application. SQL Tables:

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.



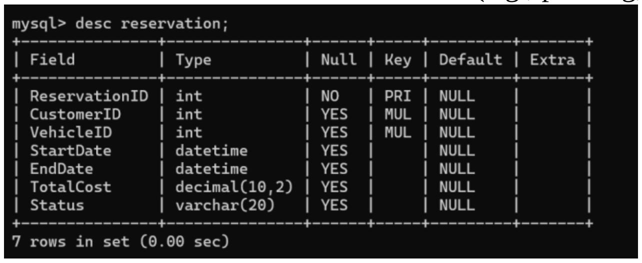
```
mysql> desc customer;
+------------------+--------------+------+-----+---------+-------+
| Field            | Type         | Null | Key | Default | Extra |
+------------------+--------------+------+-----+---------+-------+
| CustomerID       | int          | NO   | PRI | NULL    |       |
| FirstName        | varchar(50)  | YES  |     | NULL    |       |
| LastName         | varchar(50)  | YES  |     | NULL    |       |
| Email            | varchar(100) | YES  |     | NULL    |       |
| PhoneNumber      | varchar(20)  | YES  |     | NULL    |       |
| Address          | varchar(255) | YES  |     | NULL    |       |
| Username         | varchar(50)  | YES  | UNI | NULL    |       |
| Password         | varchar(255) | YES  |     | NULL    |       |
| RegistrationDate | date         | YES  |     | NULL    |       |
+------------------+--------------+------+-----+---------+-------+
9 rows in set (0.02 sec)
```

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.



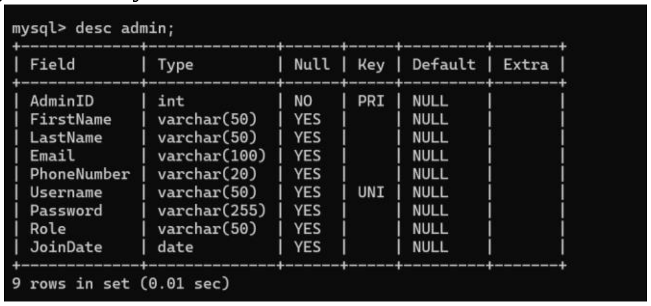
```
mysql> desc vehicle;
+--------------------+---------------+------+-----+---------+-------+
| Field              | Type          | Null | Key | Default | Extra |
+--------------------+---------------+------+-----+---------+-------+
| VehicleID          | int           | NO   | PRI | NULL    |       |
| Model              | varchar(100)  | YES  |     | NULL    |       |
| Make               | varchar(100)  | YES  |     | NULL    |       |
| Year               | int           | YES  |     | NULL    |       |
| Color              | varchar(50)   | YES  |     | NULL    |       |
| RegistrationNumber | varchar(20)   | YES  | UNI | NULL    |       |
| Availability       | tinyint(1)    | YES  |     | NULL    |       |
| DailyRate          | decimal(10,2) | YES  |     | NULL    |       |
+--------------------+---------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
```

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).



```
mysql> desc reservation;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| ReservationID | int           | NO   | PRI | NULL    |       |
| CustomerID    | int           | YES  | MUL | NULL    |       |
| VehicleID     | int           | YES  | MUL | NULL    |       |
| StartDate     | datetime      | YES  |     | NULL    |       |
| EndDate       | datetime      | YES  |     | NULL    |       |
| TotalCost     | decimal(10,2) | YES  |     | NULL    |       |
| Status        | varchar(20)   | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

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.



```
mysql> desc admin;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| AdminID     | int          | NO   | PRI | NULL    |       |
| FirstName   | varchar(50)  | YES  |     | NULL    |       |
| LastName    | varchar(50)  | YES  |     | NULL    |       |
| Email       | varchar(100) | YES  |     | NULL    |       |
| PhoneNumber | varchar(20)  | YES  |     | NULL    |       |
| Username    | varchar(50)  | YES  | UNI | NULL    |       |
| Password    | varchar(255) | YES  |     | NULL    |       |
| Role        | varchar(50)  | YES  |     | NULL    |       |
| JoinDate    | date         | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
9 rows in set (0.01 sec)
```

Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors (default and parametrized) and getters,setters

Classes:

• Customer:

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

```
class Customer:
    def __init__(self, customer_id, first_name, last_name, email, phone_number, 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_number = phone_number
        self.address = address
        self.username = username
        self.password = password
        self.registration_date = registration_date

    def authenticate(self,username, password):
        if not CustomerService.is_valid_credentials(username, password):
            raise AuthenticationException("Incorrect username or password")

        return self.password == password
```

• Vehicle:

• Properties: VehicleID, Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate

```
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
```

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

```
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):
            delta = self.end_date - self.start_date
            rental_days = delta.days
            self.total_cost = rental_days * self.daily_rate
```

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

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

    def authenticate(self,username, password):
        if not ReservationService.is_valid_credentials(username, password):
            raise AuthenticationException("Incorrect username or password")

        return self.password == password
```

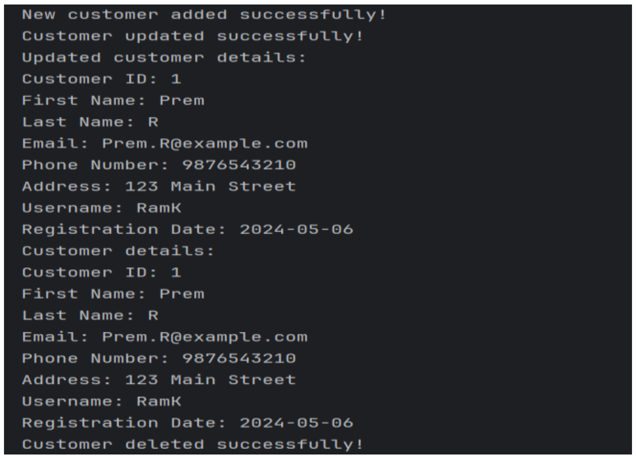
CustomerService (implements ICustomerService):
• Methods: GetCustomerById,GetCustomerByUsername, RegisterCustomer, UpdateCustomer, DeleteCustomer

```
from carconnect import Customer
from Exception import InvalidInputException
from Interfaces import ICustomerService
class CustomerService(ICustomerService):
    def __init__(self):
        self.customers = {}
        self.credentials = {}
        self.next_customer_id = 1

    def get_customer_by_id(self, CustomerID):
        customer = self.customers.get(CustomerID, None)
        if customer:
            return customer
        else:
            print(f"Customer with ID {CustomerID} not found")
            return None

    def get_customer_by_username(self, username):
        for customer in self.customers.values():
            if customer.username == username:
                return customer
        return None

    def add_customer(self, first_name, last_name, email, phone_number, address, username,
password, registration_date):
        if username in self.credentials:
            raise InvalidInputException("Username already exists")
        customer = Customer(self.next_customer_id, first_name, last_name, email, phone_number,
address, username, password, registration_date)
        self.customers[self.next_customer_id] = customer
        self.next_customer_id += 1
        self.credentials[username] = password
        return customer

    def update_customer(self, customer_id, **kwargs):
        customer = self.get_customer_by_id(customer_id)
        if customer:
            for key, value in kwargs.items():
                #print(f"Setting {key} to {value} for customer {customer_id}")
                setattr(customer, key, value)
            return True
        else:
            print(f"Customer with ID {customer_id} not found")
        return False

    def delete_customer(self, customer_id):
```

```python
        if customer_id in self.customers:
            del self.customers[customer_id]
            return True
        return False

    def is_valid_credentials(self, username, password):
        if username in self.credentials:
            if self.credentials[username] == password:
                return True
        return False
```

Implementation:

```python
from carconnect import Customer
from CustomerService import CustomerService
from Exception import InvalidInputException


customer_service = CustomerService()

try:
    new_customer = customer_service.add_customer(
        first_name="Ram",
        last_name="Kumar",
        email="ram.kumar@example.com",
        phone_number="987654321",
        address="123 Main Street",
        username="RamK",
        password="Ram123",
        registration_date="2024-05-06"
    )
    print("New customer added successfully!")
except InvalidInputException as e:
    print("Error:", e)

customer_id_to_update = 1
update_data = {
    "first_name": "Prem",
    "last_name": "R",
    "email": "Prem.R@example.com",
    "phone_number": "9876543210",
}
if customer_service.update_customer(customer_id_to_update, **update_data):
    print("Customer updated successfully!")
    updated_customer = customer_service.get_customer_by_id(customer_id_to_update)
    print("Updated customer details:")
    print(f"Customer ID: {updated_customer.customer_id}")
    print(f"First Name: {updated_customer.first_name}")
    print(f"Last Name: {updated_customer.last_name}")
    print(f"Email: {updated_customer.email}")
    print(f"Phone Number: {updated_customer.phone_number}")
    print(f"Address: {updated_customer.address}")
    print(f"Username: {updated_customer.username}")
    print(f"Registration Date: {updated_customer.registration_date}")
else:
    print(f"Failed to update customer with ID {customer_id_to_update}")


customer_id_to_get = 1
customer = customer_service.get_customer_by_id(customer_id_to_get)

if customer:
        print("Customer details:")
        print(f"Customer ID: {customer.customer_id}")
        print(f"First Name: {customer.first_name}")
        print(f"Last Name: {customer.last_name}")
        print(f"Email: {customer.email}")
        print(f"Phone Number: {customer.phone_number}")
        print(f"Address: {customer.address}")
        print(f"Username: {customer.username}")
        print(f"Registration Date: {customer.registration_date}")
else:
    print(f"Customer with ID {customer_id_to_get} not found")


customer_id_to_delete = 1
if customer_service.delete_customer(customer_id_to_delete):
    print("Customer deleted successfully!")
else:
    print(f"Failed to delete customer with ID {customer_id_to_delete}")
```



```
New customer added successfully!
Customer updated successfully!
Updated customer details:
Customer ID: 1
First Name: Prem
Last Name: R
Email: Prem.R@example.com
Phone Number: 9876543210
Address: 123 Main Street
Username: RamK
Registration Date: 2024-05-06
Customer details:
Customer ID: 1
First Name: Prem
Last Name: R
Email: Prem.R@example.com
Phone Number: 9876543210
Address: 123 Main Street
Username: RamK
Registration Date: 2024-05-06
Customer deleted successfully!
```

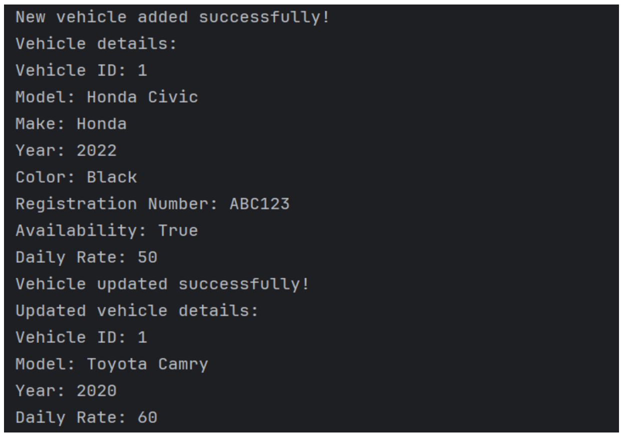
• **VehicleService (implements IVehicleService**):
 • Methods: GetVehicleById, GetAvailableVehicles, AddVehicle, UpdateVehicle, RemoveVehicle

```python
from carconnect import Vehicle  # Import the Vehicle class
from Exception import VehicleNotFoundException
from Interfaces import IVehicleService
class VehicleService(IVehicleService):
    def __init__(self):
        self.vehicles = {}
        self.next_vehicle_id = 1

    def get_vehicle_by_id(self, vehicle_id):
        vehicle = self.vehicles.get(vehicle_id, None)
        if vehicle:
            return vehicle
        else:
            raise VehicleNotFoundException()
            return None

    def get_available_vehicles(self):
        available_vehicles = []
        for vehicle in self.vehicles.values():
            if vehicle.availability:
                available_vehicles.append(vehicle)
        return available_vehicles

    def add_vehicle(self, model, make, year, color, registration_number, availability, daily_rate):
        vehicle = Vehicle(self.next_vehicle_id, model, make, year, color, registration_number,
availability, daily_rate)
        self.vehicles[self.next_vehicle_id] = vehicle
        self.next_vehicle_id += 1
        return vehicle

    def update_vehicle(self, vehicle_id, **kwargs):
        vehicle = self.get_vehicle_by_id(vehicle_id)
        if vehicle:
            for key, value in kwargs.items():
                setattr(vehicle, key, value)
            return True
        else:
            print(f"Vehicle with ID {vehicle_id} not found")
        return False

    def remove_vehicle(self, vehicle_id):
        if vehicle_id in self.vehicles:
            del self.vehicles[vehicle_id]
            return True
        return False

    def is_vehicle_available(self, vehicle_id):
        # Check if the vehicle exists
        if vehicle_id not in self.vehicles:
            raise VehicleNotFoundException("Vehicle not found")
        # Check if the vehicle is available for reservation
        vehicle = self.vehicles[vehicle_id]
        return vehicle.availability
```

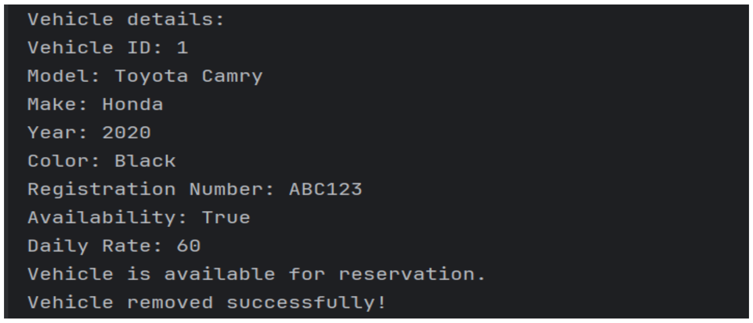
Implementation:

```
from carconnect import Vehicle  # Import the Vehicle class
from Exception import VehicleNotFoundException
from VehicleService import VehicleService

vehicle_service = VehicleService()

# Add a new vehicle
new_vehicle = vehicle_service.add_vehicle(
    model="Honda Civic",
    make="Honda",
    year=2022,
    color="Black",
    registration_number="ABC123",
    availability=True,
    daily_rate=50
)
print("New vehicle added successfully!")
print("Vehicle details:")
print(f"Vehicle ID: {new_vehicle.vehicle_id}")
print(f"Model: {new_vehicle.model}")
print(f"Make: {new_vehicle.make}")
print(f"Year: {new_vehicle.year}")
print(f"Color: {new_vehicle.color}")
print(f"Registration Number: {new_vehicle.registration_number}")
print(f"Availability: {new_vehicle.availability}")
print(f"Daily Rate: {new_vehicle.daily_rate}")

# Update an existing vehicle
vehicle_id_to_update = 1  # Assuming vehicle ID 1 exists
update_data = {
    "model": "Toyota Camry",
    "year": 2020,
    "daily_rate": 60
}
if vehicle_service.update_vehicle(vehicle_id_to_update, **update_data):
    print("Vehicle updated successfully!")
    # Print the updated vehicle
    updated_vehicle = vehicle_service.get_vehicle_by_id(vehicle_id_to_update)
    print("Updated vehicle details:")
    print(f"Vehicle ID: {updated_vehicle.vehicle_id}")
    print(f"Model: {updated_vehicle.model}")
    print(f"Year: {updated_vehicle.year}")
    print(f"Daily Rate: {updated_vehicle.daily_rate}")
else:
    print(f"Failed to update vehicle with ID {vehicle_id_to_update}")
```

```python
# Get vehicle by ID
vehicle_id_to_get = 1
try:
    vehicle = vehicle_service.get_vehicle_by_id(vehicle_id_to_get)
    print("Vehicle details:")
    print(f"Vehicle ID: {vehicle.vehicle_id}")
    print(f"Model: {vehicle.model}")
    print(f"Make: {vehicle.make}")
    print(f"Year: {vehicle.year}")
    print(f"Color: {vehicle.color}")
    print(f"Registration Number: {vehicle.registration_number}")
    print(f"Availability: {vehicle.availability}")
    print(f"Daily Rate: {vehicle.daily_rate}")
except VehicleNotFoundException:
    print(f"Vehicle with ID {vehicle_id_to_get} not found")

# Check if a vehicle is available
vehicle_id_to_check = 1
try:
    if vehicle_service.is_vehicle_available(vehicle_id_to_check):
        print("Vehicle is available for reservation.")
    else:
        print("Vehicle is not available for reservation.")
except VehicleNotFoundException:
    print(f"Vehicle with ID {vehicle_id_to_check} not found")

# Remove a vehicle
vehicle_id_to_remove = 1
if vehicle_service.remove_vehicle(vehicle_id_to_remove):
    print("Vehicle removed successfully!")
else:
    print(f"Failed to remove vehicle with ID {vehicle_id_to_remove}")
```



```
New vehicle added successfully!
Vehicle details:
Vehicle ID: 1
Model: Honda Civic
Make: Honda
Year: 2022
Color: Black
Registration Number: ABC123
Availability: True
Daily Rate: 50
Vehicle updated successfully!
Updated vehicle details:
Vehicle ID: 1
Model: Toyota Camry
Year: 2020
Daily Rate: 60
```



```
Vehicle details:
Vehicle ID: 1
Model: Toyota Camry
Make: Honda
Year: 2020
Color: Black
Registration Number: ABC123
Availability: True
Daily Rate: 60
Vehicle is available for reservation.
Vehicle removed successfully!
```

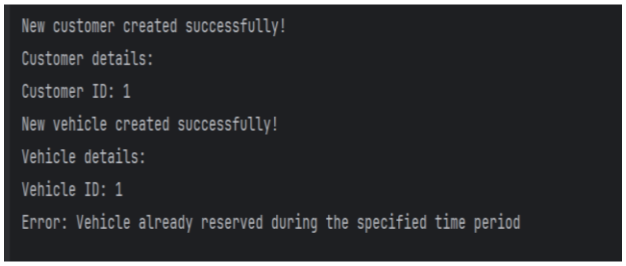
• **ReservationService (implements IReservationService):**
• Methods: GetReservationById, GetReservationsByCustomerId, CreateReservation, UpdateReservation, CancelReservation

```
from carconnect import Reservation  # Import the Reservation class
from Exception import ReservationException
from VehicleService import VehicleService
from Interfaces import IReservationService
class ReservationService(IReservationService):
    def __init__(self):
        self.reservations = {}
        self.credentials = {}
        self.next_reservation_id = 1

    def get_reservation_by_id(self, reservation_id):
        reservation = self.reservations.get(reservation_id, None)
        if reservation:
            return reservation
        else:
            print(f"Reservation with ID {reservation_id} not found")
            return None

    def get_reservations_by_customer_id(self, customer_id):
        customer_reservations = []
        for reservation in self.reservations.values():
            if reservation.customer_id == customer_id:
                customer_reservations.append(reservation)
        return customer_reservations

    def create_reservation(self, customer_id, vehicle_id, start_date, end_date, total_cost, status):
        if not VehicleService.is_vehicle_available(vehicle_id):
            raise ReservationException("Vehicle already reserved")
        reservation = Reservation(self.next_reservation_id, customer_id, vehicle_id, start_date,
end_date, total_cost, status)
        self.reservations[self.next_reservation_id] = reservation
        self.next_reservation_id += 1
        return reservation

    def update_reservation(self, reservation_id, **kwargs):
        reservation = self.get_reservation_by_id(reservation_id)
        if reservation:
            for key, value in kwargs.items():
                setattr(reservation, key, value)
            return True
        else:
            print(f"Reservation with ID {reservation_id} not found")
        return False

    def cancel_reservation(self, reservation_id):
        if reservation_id in self.reservations:
            del self.reservations[reservation_id]
            return True
        return False

    def is_valid_credentials(self, username, password):
        if username in self.credentials:
            if self.credentials[username] == password:
                return True
        return False
```

Implementation:

```
from Exception import ReservationException
from ReservationService import ReservationService
from CustomerService import CustomerService
from VehicleService import VehicleService

reservation_service = ReservationService()

customer_service = CustomerService()

vehicle_service = VehicleService()

try:
    new_customer = customer_service.add_customer(
        first_name="John",
        last_name="Doe",
        email="john.doe@example.com",
        phone_number="1234567890",
        address="123 Main Street",
        username="johndoe",
        password="password",
        registration_date="2024-05-06"
    )
    print("New customer created successfully!")
    print("Customer details:")
    print(f"Customer ID: {new_customer.customer_id}")
except Exception as e:
```

```python
    print("Error:", e)

try:
    new_vehicle = vehicle_service.add_vehicle(
        model="Toyota Camry",
        make="Toyota",
        year=2023,
        color="Blue",
        registration_number="XYZ456",
        availability=True,
        daily_rate=60
    )
    print("New vehicle created successfully!")
    print("Vehicle details:")
    print(f"Vehicle ID: {new_vehicle.vehicle_id}")
except Exception as e:
    print("Error:", e)

try:
    new_reservation = reservation_service.create_reservation(
        customer_id=new_customer.customer_id,  # Use the CustomerID of the new customer
        vehicle_id=new_vehicle.vehicle_id,   # Use the VehicleID of the new vehicle
        start_date="2024-05-07",
        end_date="2024-05-10",
        total_cost=150,
        status="confirmed"
    )
    print("New reservation created successfully!")
    print("Reservation details:")
    print(f"Reservation ID: {new_reservation.reservation_id}")
    print(f"Customer ID: {new_reservation.customer_id}")
    print(f"Vehicle ID: {new_reservation.vehicle_id}")
    print(f"Start Date: {new_reservation.start_date}")
    print(f"End Date: {new_reservation.end_date}")
    print(f"Total Cost: {new_reservation.total_cost}")
    print(f"Status: {new_reservation.status}")
except ReservationException as e:
    print("Error:", e)
```



```
New customer created successfully!
Customer details:
Customer ID: 1
New vehicle created successfully!
Vehicle details:
Vehicle ID: 1
Error: Vehicle already reserved during the specified time period
```

• **AdminService (implements IAdminService):**
• Methods: GetAdminById, GetAdminByUsername, RegisterAdmin, UpdateAdmin, DeleteAdmin

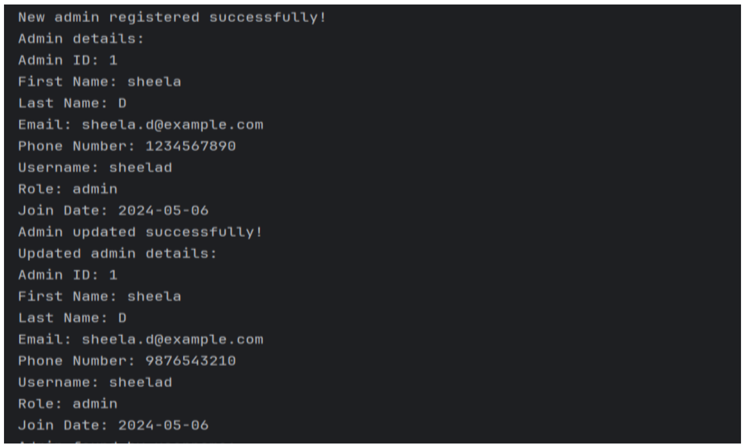
```
from carconnect import Admin
from Exception import AdminNotFoundException
from Interfaces import IAdminService
class AdminService(IAdminService):
    def __init__(self):
        self.admins = {}
        self.next_admin_id = 1

    def get_admin_by_id(self, admin_id):
        admin = self.admins.get(admin_id, None)
        if admin:
            return admin
        else:
            raise AdminNotFoundException()
            return None

    def get_admin_by_username(self, username):
        for admin in self.admins.values():
            if admin.username == username:
                return admin
        return None

    def register_admin(self, first_name, last_name, email, phone_number, username, password, role,
join_date):
        admin = Admin(self.next_admin_id, first_name, last_name, email, phone_number, username,
password, role, join_date)
        self.admins[self.next_admin_id] = admin
        self.next_admin_id += 1
        return admin

    def update_admin(self, admin_id, **kwargs):
        admin = self.get_admin_by_id(admin_id)
        if admin:
            for key, value in kwargs.items():
                setattr(admin, key, value)
            return True
        else:
            print(f"Admin with ID {admin_id} not found")
        return False

    def delete_admin(self, admin_id):
        if admin_id in self.admins:
            del self.admins[admin_id]
            return True
        return False
```

Implementation:

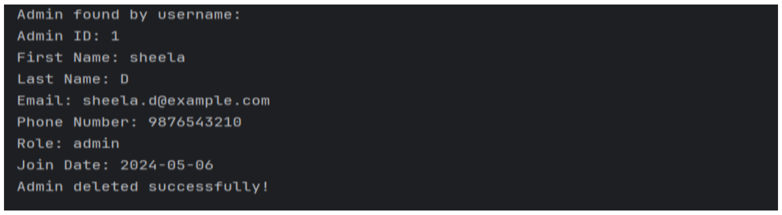
```
from carconnect import Reservation
from carconnect import Admin
from Exception import AdminNotFoundException
```

```python
from AdminService import AdminService

admin_service = AdminService()
new_admin = admin_service.register_admin(
    first_name="sheela",
    last_name="D",
    email="sheela.d@example.com",
    phone_number="1234567890",
    username="sheelad",
    password="sheela1",
    role="admin",
    join_date="2024-05-06"
)
print("New admin registered successfully!")
print("Admin details:")
print(f"Admin ID: {new_admin.admin_id}")
print(f"First Name: {new_admin.first_name}")
print(f"Last Name: {new_admin.last_name}")
print(f"Email: {new_admin.email}")
print(f"Phone Number: {new_admin.phone_number}")
print(f"Username: {new_admin.username}")
print(f"Role: {new_admin.role}")
print(f"Join Date: {new_admin.join_date}")

admin_id_to_update = new_admin.admin_id
update_data = {
    "phone_number": "9876543210"
}
if admin_service.update_admin(admin_id_to_update, **update_data):
    print("Admin updated successfully!")

    updated_admin = admin_service.get_admin_by_id(admin_id_to_update)
    print("Updated admin details:")
    print(f"Admin ID: {updated_admin.admin_id}")
    print(f"First Name: {updated_admin.first_name}")
    print(f"Last Name: {updated_admin.last_name}")
    print(f"Email: {updated_admin.email}")
    print(f"Phone Number: {updated_admin.phone_number}")
    print(f"Username: {updated_admin.username}")
    print(f"Role: {updated_admin.role}")
    print(f"Join Date: {updated_admin.join_date}")
else:
    print(f"Failed to update admin with ID {admin_id_to_update}")

username_to_get_admin = "sheelad"
admin_by_username = admin_service.get_admin_by_username(username_to_get_admin)
if admin_by_username:
    print("Admin found by username:")
    print(f"Admin ID: {admin_by_username.admin_id}")
    print(f"First Name: {admin_by_username.first_name}")
    print(f"Last Name: {admin_by_username.last_name}")
    print(f"Email: {admin_by_username.email}")
    print(f"Phone Number: {admin_by_username.phone_number}")
    print(f"Role: {admin_by_username.role}")
    print(f"Join Date: {admin_by_username.join_date}")
else:
    print(f"No admin found with username {username_to_get_admin}")

# Delete an admin
admin_id_to_delete = new_admin.admin_id  # Use the ID of the newly registered admin
if admin_service.delete_admin(admin_id_to_delete):
    print("Admin deleted successfully!")
else:
    print(f"Failed to delete admin with ID {admin_id_to_delete}")
```

Implementation:



```
New admin registered successfully!
Admin details:
Admin ID: 1
First Name: sheela
Last Name: D
Email: sheela.d@example.com
Phone Number: 1234567890
Username: sheelad
Role: admin
Join Date: 2024-05-06
Admin updated successfully!
Updated admin details:
Admin ID: 1
First Name: sheela
Last Name: D
Email: sheela.d@example.com
Phone Number: 9876543210
Username: sheelad
Role: admin
Join Date: 2024-05-06
```



```
Admin found by username:
Admin ID: 1
First Name: sheela
Last Name: D
Email: sheela.d@example.com
Phone Number: 9876543210
Role: admin
Join Date: 2024-05-06
Admin deleted successfully!
```

Interfaces:

• ICustomerService:

• GetCustomerById(customerId) • GetCustomerByUsername(username) • RegisterCustomer(customerData) • UpdateCustomer(customerData) • DeleteCustomer(customerId)

```
class ICustomerService(ABC):
    @abstractmethod
    def get_customer_by_id(self, customer_id):
        pass

    @abstractmethod
    def get_customer_by_username(self, username):
        pass

    @abstractmethod
    def add_customer(self, first_name, last_name, email, phone_number, address, username,
password, registration_date):
        pass

    @abstractmethod
    def update_customer(self, customer_id, **kwargs):
        pass

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

• IVehicleService:
• GetVehicleById(vehicleId) • GetAvailableVehicles() • AddVehicle(vehicleData) •
UpdateVehicle(vehicleData) • RemoveVehicle(vehicleId)

```

class IVehicleService(ABC):
    @abstractmethod
    def get_vehicle_by_id(self, vehicle_id):
        pass

    @abstractmethod
    def get_available_vehicles(self):
        pass

    @abstractmethod
    def add_vehicle(self, model, make, year, color, registration_number, availability, daily_rate):
        pass

    @abstractmethod
    def update_vehicle(self, vehicle_id, **kwargs):
        pass

    @abstractmethod
    def remove_vehicle(self, vehicle_id):
        pass

    @abstractmethod
    def is_vehicle_available(self, vehicle_id,start_date,end_date):
        pass
```

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

```
class IReservationService(ABC):
    @abstractmethod
    def get_reservation_by_id(self, reservation_id):
        pass

    @abstractmethod
    def get_reservations_by_customer_id(self, customer_id):
        pass

    @abstractmethod
    def create_reservation(self, customer_id, vehicle_id, start_date, end_date, total_cost, status):
        pass

    @abstractmethod
    def update_reservation(self, reservation_id, **kwargs):
        pass

    @abstractmethod
    def cancel_reservation(self, reservation_id):
        pass
```

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

```
class IAdminService(ABC):
    @abstractmethod
    def get_admin_by_id(self, admin_id):
        pass

    @abstractmethod
    def get_admin_by_username(self, username):
        pass

    @abstractmethod
    def register_admin(self, first_name, last_name, email, phone_number, username, password,
role, join_date):
        pass

    @abstractmethod
    def update_admin(self, admin_id, **kwargs):
        pass
    @abstractmethod
    def delete_admin(self, admin_id):
        pass
```

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.

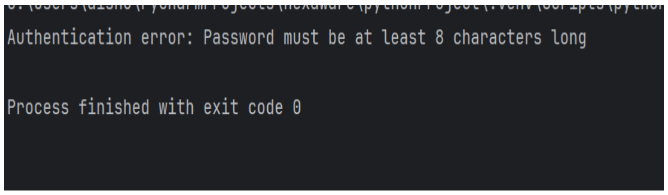
```
import mysql.connector
from CustomerService import CustomerService
from VehicleService import VehicleService
from ReservationService import ReservationService
from AdminService import AdminService
from Exception import DatabaseConnectionException,VehicleNotFoundException
class SQLCommand:
    def __init__(self, connection):
        self.connection = connection

    def get_customer_by_id_from_db(self, customer_id):
        query = "SELECT * FROM Customer WHERE CustomerID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (customer_id,))
            customer = cursor.fetchone()
            return customer

    def get_customer_by_username_from_db(self, username):
        query = "SELECT * FROM Customer WHERE Username = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (username,))
            customer = cursor.fetchone()
            return customer

    def add_customer_to_db(self, first_name, last_name, email, phone_number, address, username,
password, registration_date):
        query = "INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address,
Username, Password, RegistrationDate) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (first_name, last_name, email, phone_number, address, username,
password, registration_date))
            self.connection.commit()
            return cursor.lastrowid

    def update_customer_to_db(self, customer_id, **kwargs):
        placeholders = ", ".join([f"{key} = %s" for key in kwargs.keys()])
        values = tuple(kwargs.values()) + (customer_id,)
        query = f"UPDATE Customer SET {placeholders} WHERE CustomerID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, values)
            self.connection.commit()
            return cursor.rowcount > 0

    def delete_customer_from_db(self, customer_id):
        query = "DELETE FROM Customer WHERE CustomerID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (customer_id,))
            self.connection.commit()
            return cursor.rowcount > 0

    def get_available_vehicles_from_db(self):
        query = "SELECT * FROM Vehicle WHERE Availability = 1"
        with self.connection.cursor() as cursor:
            cursor.execute(query)
            vehicles = cursor.fetchall()
            return vehicles

    def get_vehicle_by_id_from_db(self, vehicle_id):
        query = "SELECT * FROM Vehicle WHERE VehicleID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (vehicle_id,))
            vehicle = cursor.fetchone()
            return vehicle

    def get_available_vehicles_from_db(self):
        query = "SELECT * FROM Vehicle WHERE Availability = 1"
        with self.connection.cursor() as cursor:
            cursor.execute(query)
            vehicles = cursor.fetchall()
            return vehicles

    def add_vehicle_to_db(self, model, make, year, color, registration_number, availability, daily_rate):
        query = "INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability,
DailyRate) VALUES (%s, %s, %s, %s, %s, %s, %s)"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (model, make, year, color, registration_number, availability, daily_rate))
            self.connection.commit()
            return cursor.lastrowid

    def update_vehicle_to_db(self, vehicle_id, **kwargs):
        placeholders = ", ".join([f"{key} = %s" for key in kwargs.keys()])
        values = tuple(kwargs.values()) + (vehicle_id,)
        query = f"UPDATE Vehicle SET {placeholders} WHERE VehicleID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, values)
            self.connection.commit()
            return cursor.rowcount > 0

    def remove_vehicle_from_db(self, vehicle_id):
        query = "DELETE FROM Vehicle WHERE VehicleID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (vehicle_id,))
            self.connection.commit()
            return cursor.rowcount > 0

    def is_vehicle_available_in_db(self, vehicle_id):
        query = "SELECT Availability FROM Vehicle WHERE VehicleID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (vehicle_id,))
            result = cursor.fetchone()
            if result:
                return bool(result[0])
            else:
                raise VehicleNotFoundException("Vehicle not found")

    def get_reservation_by_id_from_db(self, reservation_id):
        query = "SELECT * FROM Reservation WHERE ReservationID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (reservation_id,))
            reservation = cursor.fetchone()
            return reservation

    def get_reservations_by_customer_id_from_db(self, customer_id):
        query = "SELECT * FROM Reservation WHERE CustomerID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (customer_id,))
            reservations = cursor.fetchall()
            return reservations

    def create_reservation_in_db(self, customer_id, vehicle_id, start_date, end_date, total_cost, status):
        query = "INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost,
Status) VALUES (%s, %s, %s, %s, %s, %s)"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (customer_id, vehicle_id, start_date, end_date, total_cost, status))
            self.connection.commit()
            reservation_id = cursor.lastrowid
            return reservation_id

    def update_reservation_in_db(self, reservation_id, **kwargs):
        placeholders = ", ".join([f"{key} = %s" for key in kwargs.keys()])
        values = tuple(kwargs.values()) + (reservation_id,)
        query = f"UPDATE Reservation SET {placeholders} WHERE ReservationID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, values)
            self.connection.commit()
            return cursor.rowcount > 0

    def cancel_reservation_in_db(self, reservation_id):
        query = "DELETE FROM Reservation WHERE ReservationID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (reservation_id,))
            self.connection.commit()
            return cursor.rowcount > 0

    def get_admin_by_id_from_db(self, admin_id):
        query = "SELECT * FROM Admin WHERE AdminID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (admin_id,))
            admin = cursor.fetchone()
            return admin

    def get_admin_by_username_from_db(self, username):
        query = "SELECT * FROM Admin WHERE Username = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (username,))
            admin = cursor.fetchone()
            return admin

    def register_admin_in_db(self, first_name, last_name, email, phone_number, username, password, role,
join_date):
        query = "INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username,
Password, Role, JoinDate) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (first_name, last_name, email, phone_number, username, password, role,
join_date))
            self.connection.commit()
            admin_id = cursor.lastrowid
            return admin_id

    def update_admin_in_db(self, admin_id, **kwargs):
        placeholders = ", ".join([f"{key} = %s" for key in kwargs.keys()])
        values = tuple(kwargs.values()) + (admin_id,)
        query = f"UPDATE Admin SET {placeholders} WHERE AdminID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, values)
            self.connection.commit()
            return cursor.rowcount > 0

    def delete_admin_from_db(self, admin_id):
        query = "DELETE FROM Admin WHERE AdminID = %s"
        with self.connection.cursor() as cursor:
            cursor.execute(query, (admin_id,))
            self.connection.commit()
            return cursor.rowcount > 0

def establish_database_connection():
    con = mysql.connector.connect(
        host="localhost",
        user="root",
        password="root",
        port="3306",
        database="carconnect"
    )
    if not is_database_connected(con):
        raise DatabaseConnectionException("Unable to establish database connection")
    return con
```

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

```
class AuthenticationException(Exception):
    """Exception raised for authentication issues."""

    def __init__(self, message="Authentication failed"):
        self.message = message
        super().__init__(self.message)
```

Implementation:

```
try:
    db_connection = establish_database_connection()
    sql_command = SQLCommand(db_connection)
    customer_id = sql_command.add_customer_to_db("John", "Doe", "john@example.com",
"123456789", "123 Main St", "johndoe", "123", "2024-05-06")

    print("Customer added with ID:", customer_id)
except AuthenticationException as e:
    print("Authentication error:", e)
except Exception as e:
    print("Error:", e)
finally:
    if db_connection.is_connected():
        db_connection.close()
```

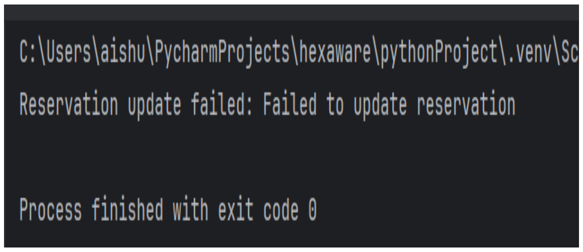


```
Authentication error: Password must be at least 8 characters long

Process finished with exit code 0
```

ReservationException: • Thrown when there is an issue with reservations. • Example Usage: Attempt into make a reservation for a vehicle that is already reserved.

```

class ReservationException(Exception):
    """Exception raised for reservation issues."""

    def __init__(self, message="Reservation failed"):
        self.message = message
        super().__init__(self.message)
```

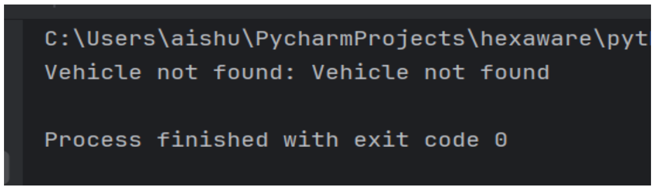
```
try:
    db_connection = establish_database_connection()
    sql_command = SQLCommand(db_connection)
    sql_command.update_reservation_in_db(123, status='confirmed')

    print("Reservation updated successfully")
except ReservationException as e:
    print("Reservation update failed:", e)
except Exception as e:
    print("Error:", e)
finally:
    if db_connection.is_connected():
        db_connection.close()
```



```
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Sc
Reservation update failed: Failed to update reservation


Process finished with exit code 0
```

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

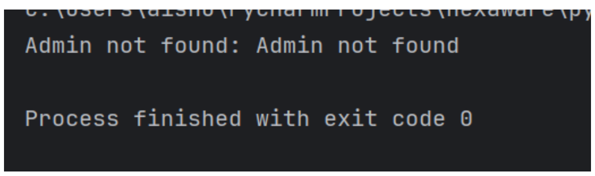
```
try:
    db_connection = establish_database_connection()
    sql_command = SQLCommand(db_connection)
    vehicle = sql_command.get_vehicle_by_id_from_db(123)
    print("Vehicle found:", vehicle)
except VehicleNotFoundException as e:
    print("Vehicle not found:", e)
except Exception as e:
    print("Error:", e)
finally:

    if db_connection.is_connected():
        db_connection.close()
```



```
C:\Users\aishu\PycharmProjects\hexaware\pyt
Vehicle not found: Vehicle not found

Process finished with exit code 0
```

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

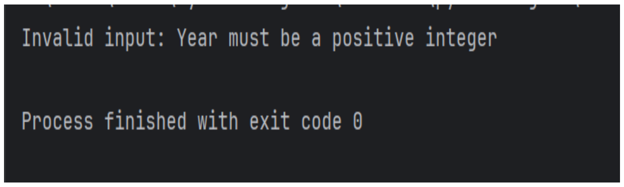
```
def is_database_connected(connection):
    return connection is not None

try:
    db_connection = establish_database_connection()
    sql_command = SQLCommand(db_connection)
    admin = sql_command.get_admin_by_id_from_db(123)


    print("Admin found:", admin)
except AdminNotFoundException as e:
    print("Admin not found:", e)
except Exception as e:
    print("Error:", e)
finally:
    if db_connection.is_connected():
        db_connection.close()
```



```
C:\Users\aisho\PycharmProjects\hexaware\py
Admin not found: Admin not found

Process finished with exit code 0
```

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

```
def is_database_connected(connection):
    return connection is not None

try:
    db_connection = establish_database_connection()
    sql_command = SQLCommand(db_connection)
    vehicle_id = sql_command.add_vehicle_to_db("Toyota", "Corolla", -2020, "Red", "ABC123", 1, 50.0)


    print("Vehicle added with ID:", vehicle_id)
except InvalidInputException as e:
    print("Invalid input:", e)
except Exception as e:
    print("Error:", e)
finally:
    if db_connection.is_connected():
        db_connection.close()
```



```
Invalid input: Year must be a positive integer


Process finished with exit code 0
```

DatabaseConnectionException: • Thrown when there is an issue with the database connection. • Example Usage: Unable to establish a connection to the database.

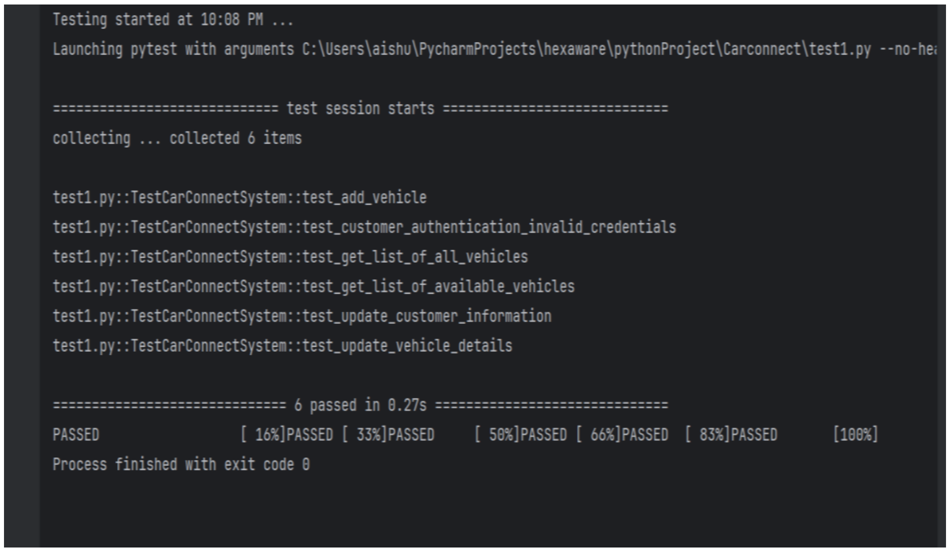
```
def establish_database_connection():
    con = mysql.connector.connect(
        host="localhost",
        user="root",
        password="root",
        port="3306",
        database="carconnect"
    )
    if not is_database_connected(con):
        raise DatabaseConnectionException("Unable to establish database connection")
    return con
```

Unit Testing:
Create NUnit test cases for car rental System are essential to ensure the correctness and reliability of your system. Below are some example questions to guide the creation of NUnit test cases for various components of the system:

1. Test customer authentication with invalid credentials.
2. Test updating customer information.
3. Test adding a new vehicle.
4. Test updating vehicle details.
5. Test getting a list of available vehicles.
6. Test getting a list of all vehicles

```
import unittest
from .database import SQLCommand, establish_database_connection
from .Exception import AuthenticationException

class TestCarConnectSystem(unittest.TestCase):
    def setUp(self):
        self.connection = establish_database_connection()
        self.sql_command = SQLCommand(self.connection)

    def tearDown(self):
        self.connection.close()

    def test_customer_authentication_invalid_credentials(self):
        invalid_password = '12'
        with self.assertRaises(AuthenticationException):
```

```python
        self.sql_command.authentication_invalid(invalid_password)

    def test_update_customer_information(self):
        customer_id = 1
        updated_info = {'FirstName': 'Rajeshh', 'LastName': 'Kumarr'}
        self.assertTrue(self.sql_command.update_customer_to_db(customer_id, **updated_info))

    def test_add_vehicle(self):
        vehicle_id = self.sql_command.add_vehicle_to_db(
            'Toyota', 'Camry', 2022, 'Black', 'ABC13', 1, 50.00
        )

        self.assertTrue(vehicle_id)

    def test_update_vehicle_details(self):
        vehicle_id = 1
       updated_details = {'Color': 'Bluee'}
        self.assertTrue(self.sql_command.update_vehicle_to_db(vehicle_id, **updated_details))

    def test_get_list_of_available_vehicles(self):
        available_vehicles = self.sql_command.get_available_vehicles_from_db()
        self.assertTrue(len(available_vehicles) > 0)

    def test_get_list_of_all_vehicles(self):
        all_vehicles = self.sql_command.get_available_vehicles_from_db()
        self.assertTrue(len(all_vehicles) > 0)

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



```
Testing started at 10:08 PM ...
Launching pytest with arguments C:\Users\aishu\PycharmProjects\hexaware\pythonProject\Carconnect\test1.py --no-hea

============================ test session starts ============================
collecting ... collected 6 items

test1.py::TestCarConnectSystem::test_add_vehicle
test1.py::TestCarConnectSystem::test_customer_authentication_invalid_credentials
test1.py::TestCarConnectSystem::test_get_list_of_all_vehicles
test1.py::TestCarConnectSystem::test_get_list_of_available_vehicles
test1.py::TestCarConnectSystem::test_update_customer_information
test1.py::TestCarConnectSystem::test_update_vehicle_details

============================== 6 passed in 0.27s ==============================
PASSED                  [ 16%]PASSED [ 33%]PASSED     [ 50%]PASSED [ 66%]PASSED  [ 83%]PASSED       [100%]
Process finished with exit code 0
```