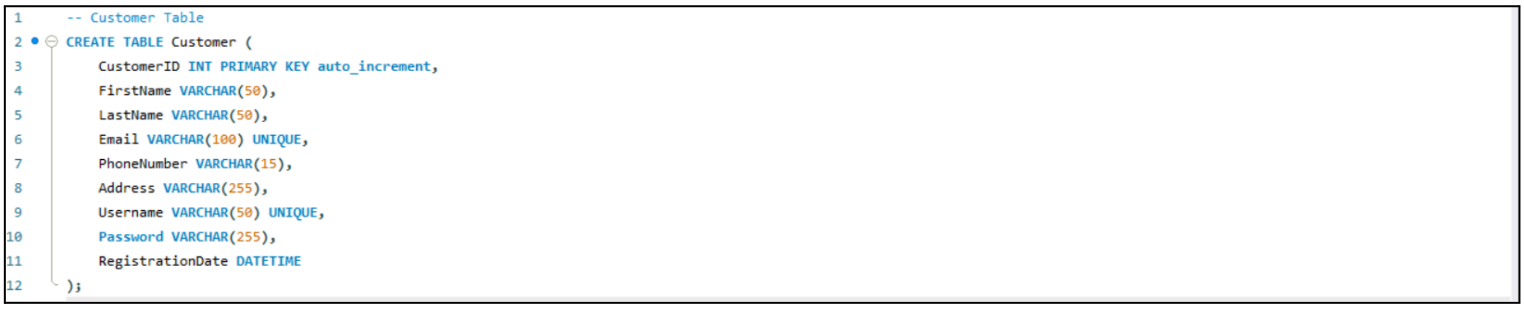
CaseStudy - CarConnect

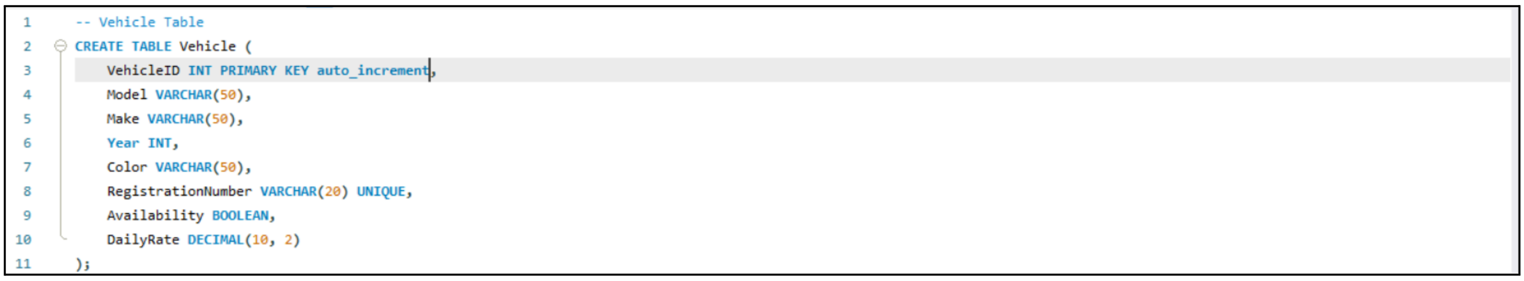
SQL Tables

Customer Table



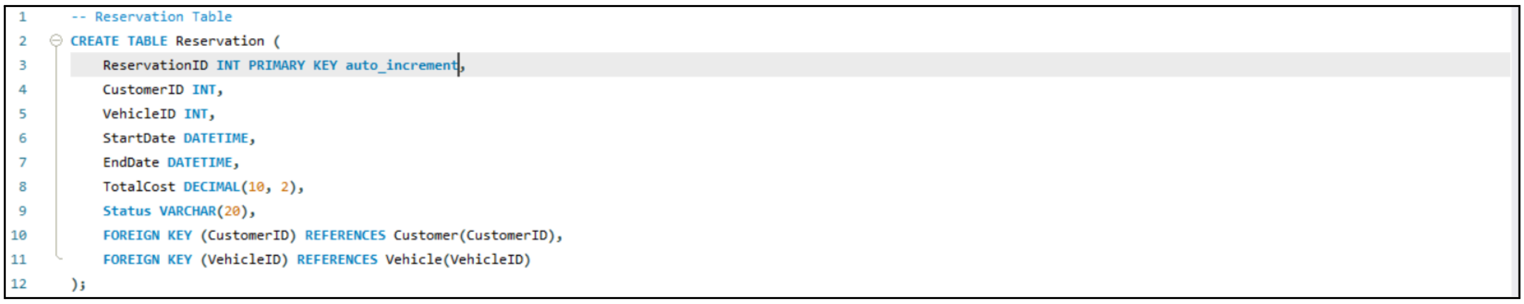
```sql
-- Customer Table
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY auto_increment,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100) UNIQUE,
    PhoneNumber VARCHAR(15),
    Address VARCHAR(255),
    Username VARCHAR(50) UNIQUE,
    Password VARCHAR(255),
    RegistrationDate DATETIME
);
```

Vehicle Table



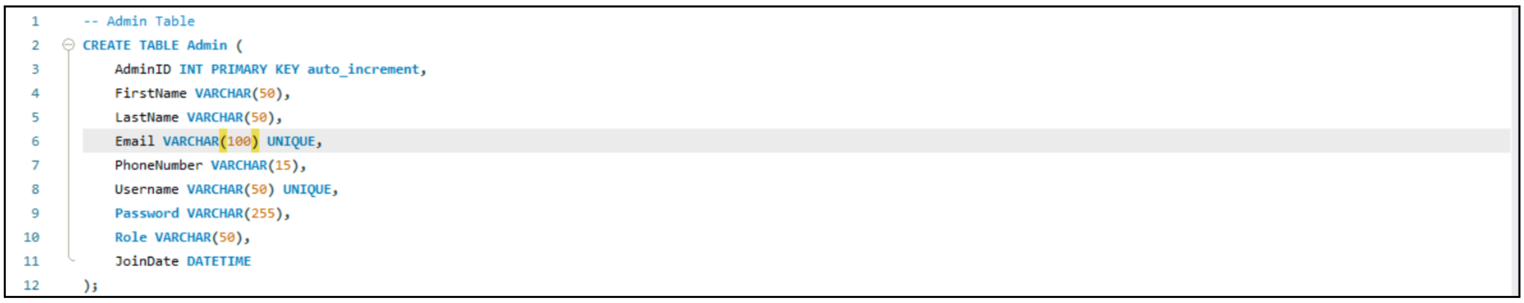
```sql
-- Vehicle Table
CREATE TABLE Vehicle (
    VehicleID INT PRIMARY KEY auto_increment,
    Model VARCHAR(50),
    Make VARCHAR(50),
    Year INT,
    Color VARCHAR(50),
    RegistrationNumber VARCHAR(20) UNIQUE,
    Availability BOOLEAN,
    DailyRate DECIMAL(10, 2)
);
```

Reservation Table



```sql
-- Reservation Table
CREATE TABLE Reservation (
    ReservationID INT PRIMARY KEY auto_increment,
    CustomerID INT,
    VehicleID INT,
    StartDate DATETIME,
    EndDate DATETIME,
    TotalCost DECIMAL(10, 2),
    Status VARCHAR(20),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID)
);
```

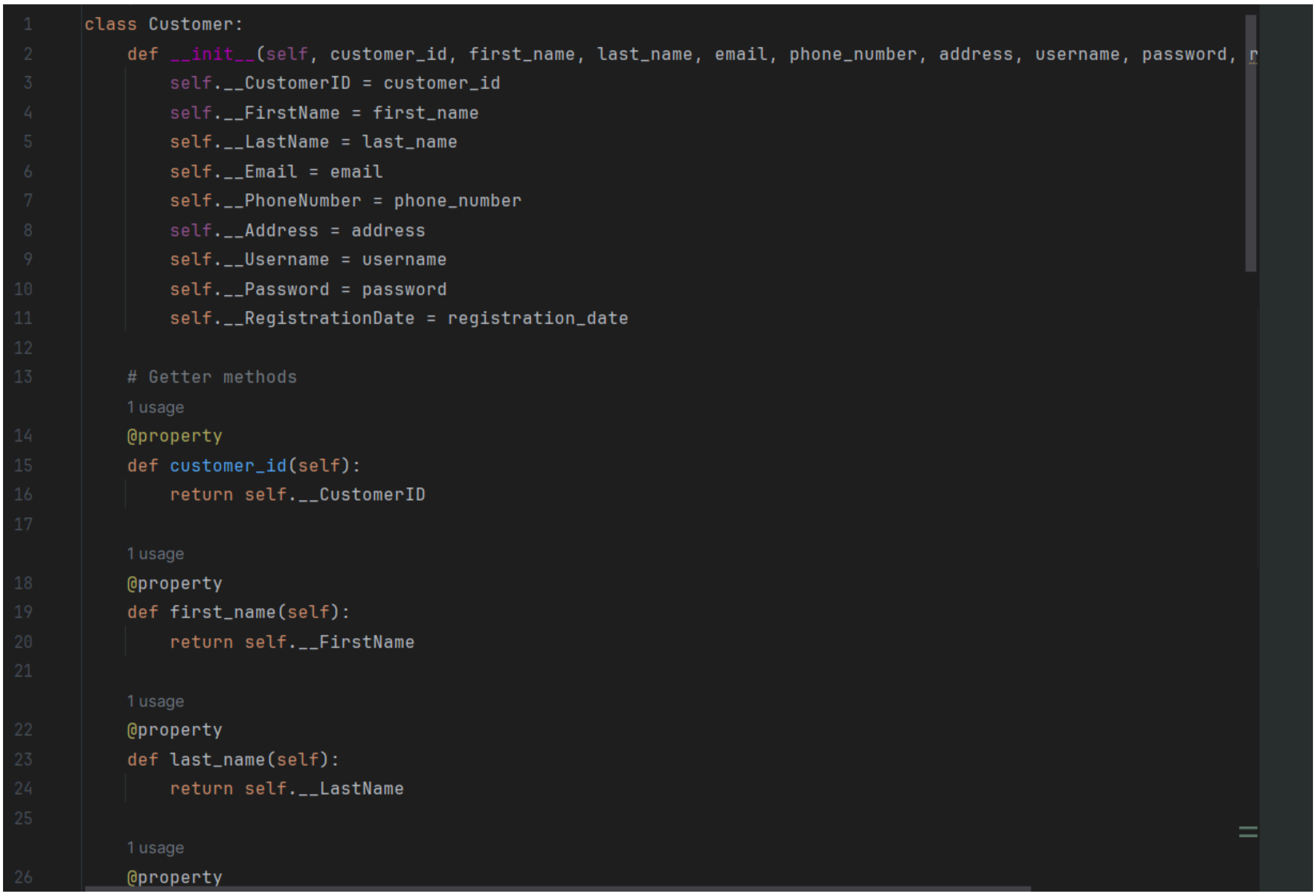
Admin Table



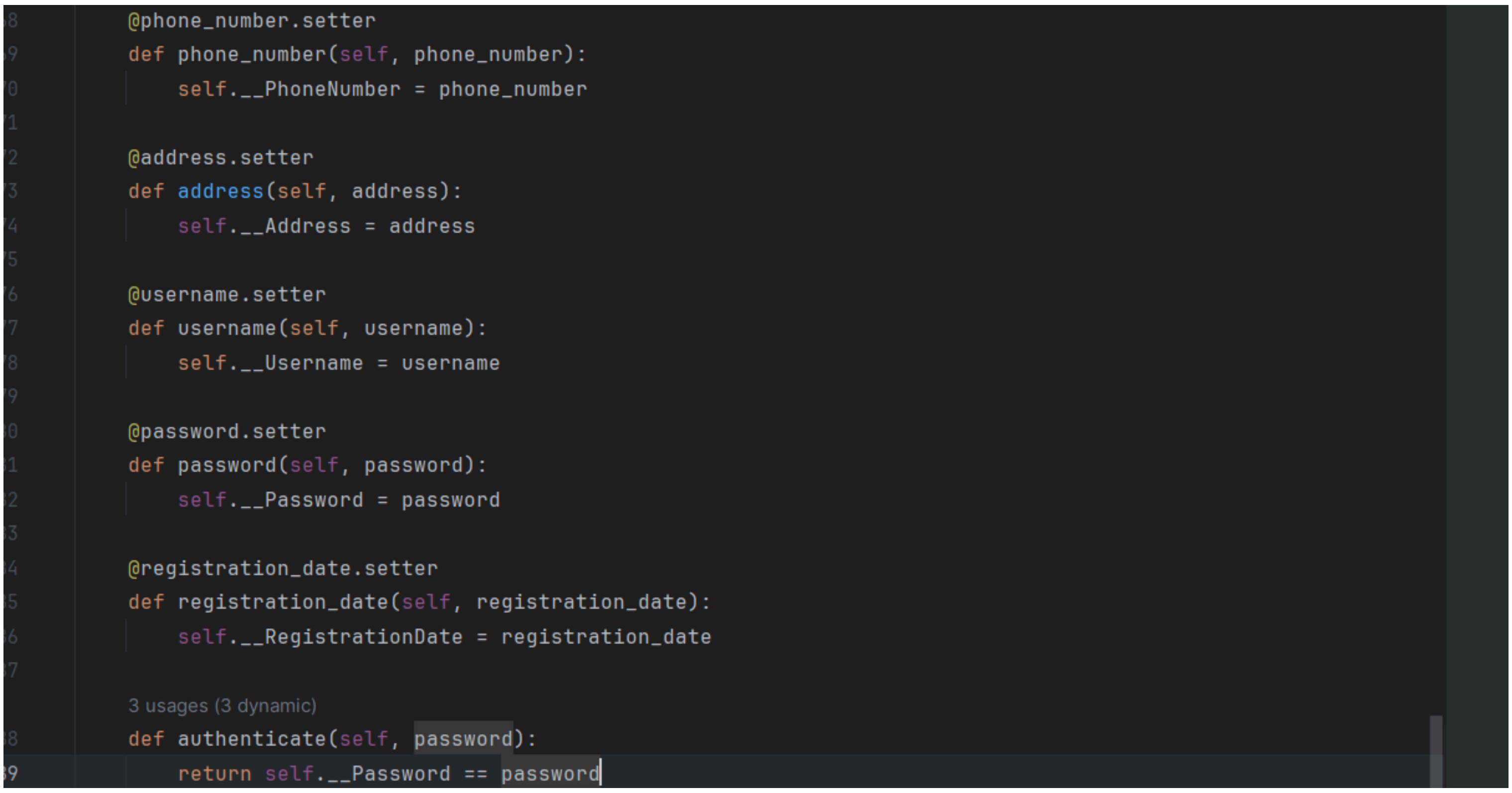
```sql
-- Admin Table
CREATE TABLE Admin (
    AdminID INT PRIMARY KEY auto_increment,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100) UNIQUE,
    PhoneNumber VARCHAR(15),
    Username VARCHAR(50) UNIQUE,
    Password VARCHAR(255),
    Role VARCHAR(50),
    JoinDate DATETIME
);
```

Classes

Customer

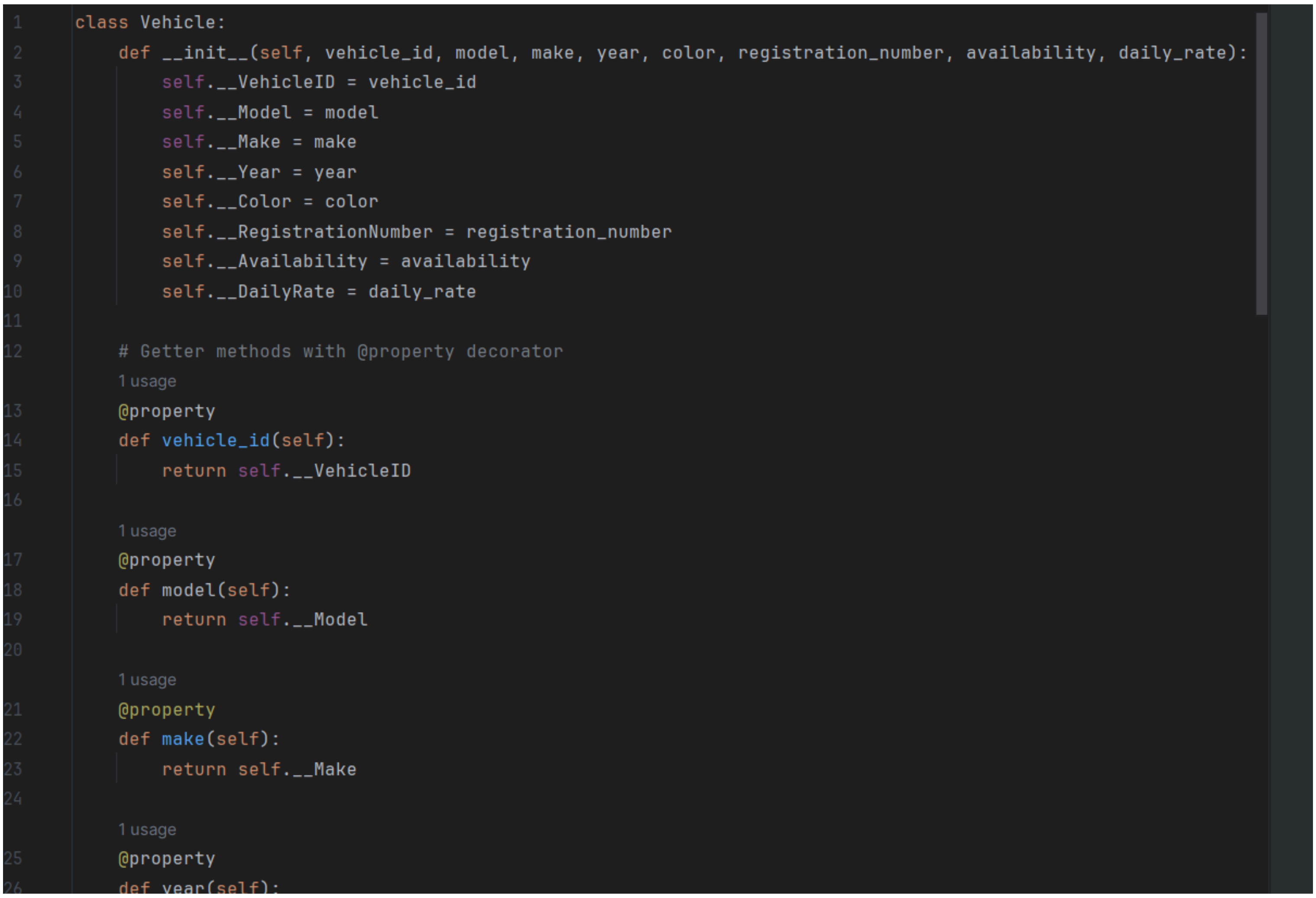


```python
class Customer:
    def __init__(self, customer_id, first_name, last_name, email, phone_number, address, username, password, r
        self.__CustomerID = customer_id
        self.__FirstName = first_name
        self.__LastName = last_name
        self.__Email = email
        self.__PhoneNumber = phone_number
        self.__Address = address
        self.__Username = username
        self.__Password = password
        self.__RegistrationDate = registration_date

    # Getter methods
    1 usage
    @property
    def customer_id(self):
        return self.__CustomerID

    1 usage
    @property
    def first_name(self):
        return self.__FirstName

    1 usage
    @property
    def last_name(self):
        return self.__LastName

    1 usage
    @property
```

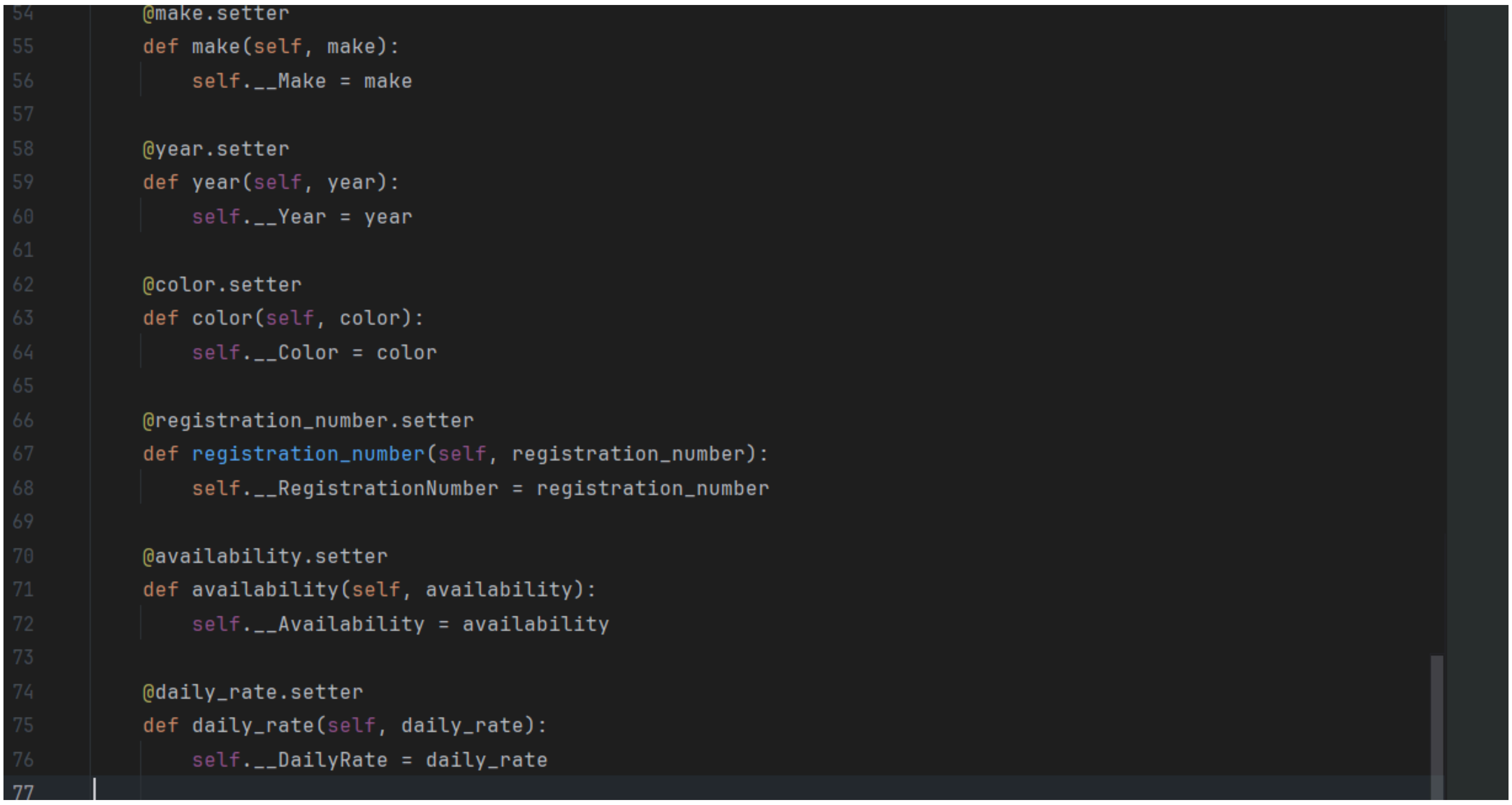


```python
    @phone_number.setter
    def phone_number(self, phone_number):
        self.__PhoneNumber = phone_number

    @address.setter
    def address(self, address):
        self.__Address = address

    @username.setter
    def username(self, username):
        self.__Username = username

    @password.setter
    def password(self, password):
        self.__Password = password

    @registration_date.setter
    def registration_date(self, registration_date):
        self.__RegistrationDate = registration_date

    3 usages (3 dynamic)
    def authenticate(self, password):
        return self.__Password == password
```

Vehicle

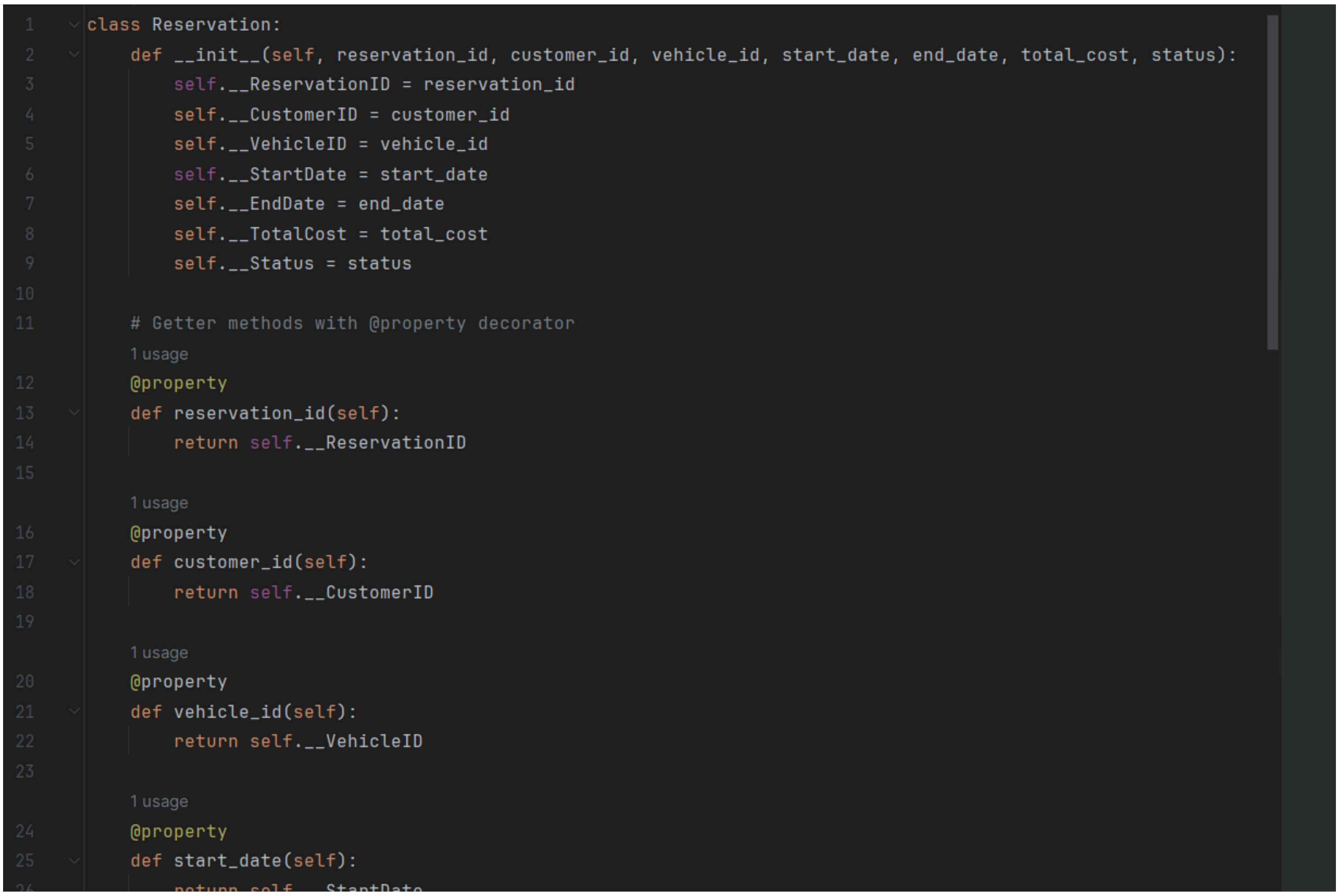


```python
class Vehicle:
    def __init__(self, vehicle_id, model, make, year, color, registration_number, availability, daily_rate):
        self.__VehicleID = vehicle_id
        self.__Model = model
        self.__Make = make
        self.__Year = year
        self.__Color = color
        self.__RegistrationNumber = registration_number
        self.__Availability = availability
        self.__DailyRate = daily_rate

    # Getter methods with @property decorator
    1 usage
    @property
    def vehicle_id(self):
        return self.__VehicleID

    1 usage
    @property
    def model(self):
        return self.__Model

    1 usage
    @property
    def make(self):
        return self.__Make

    1 usage
    @property
    def year(self):
```

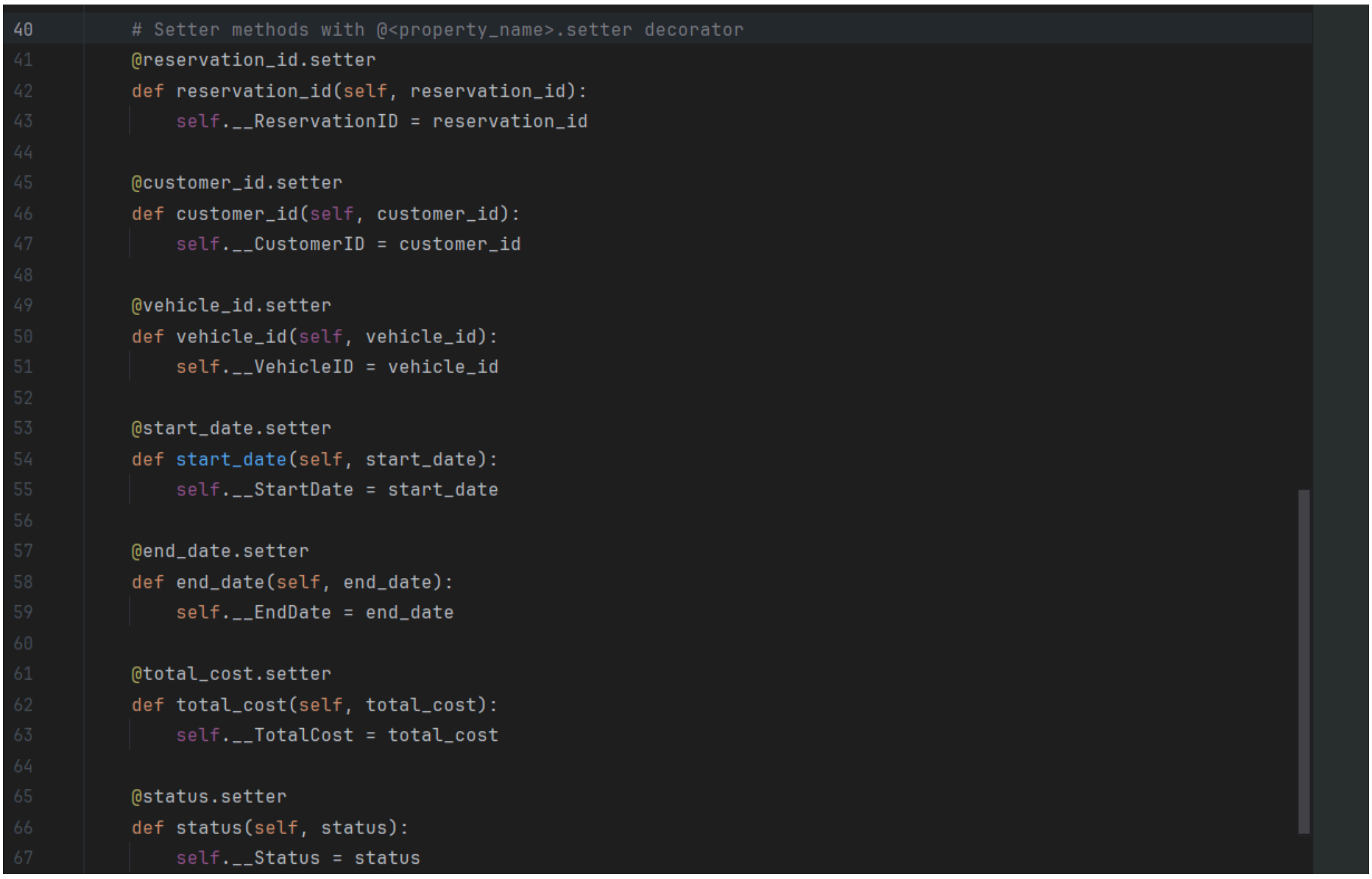


```python
    @make.setter
    def make(self, make):
        self.__Make = make

    @year.setter
    def year(self, year):
        self.__Year = year

    @color.setter
    def color(self, color):
        self.__Color = color

    @registration_number.setter
    def registration_number(self, registration_number):
        self.__RegistrationNumber = registration_number

    @availability.setter
    def availability(self, availability):
        self.__Availability = availability

    @daily_rate.setter
    def daily_rate(self, daily_rate):
        self.__DailyRate = daily_rate

```

Reservation

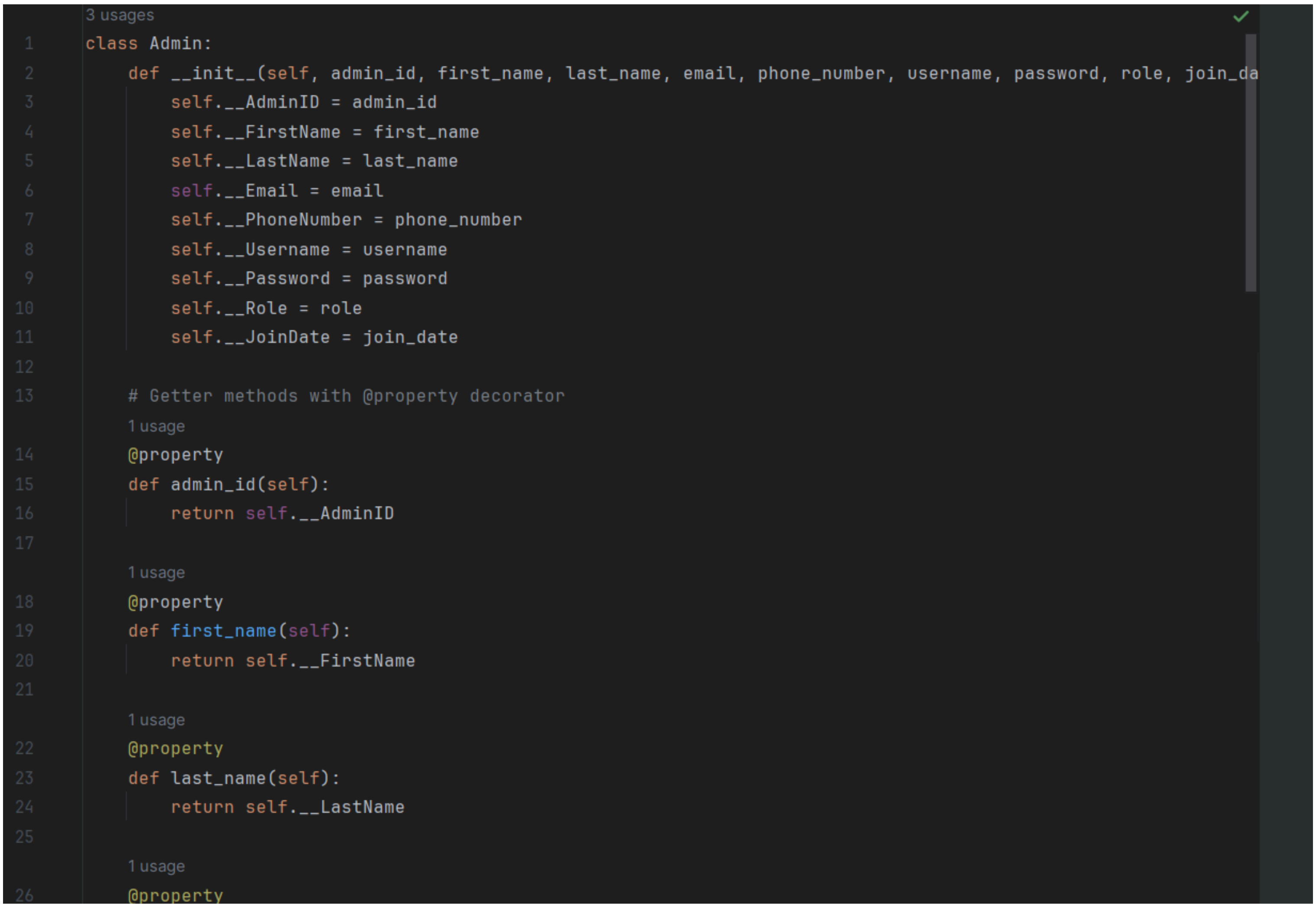


```python
class Reservation:
    def __init__(self, reservation_id, customer_id, vehicle_id, start_date, end_date, total_cost, status):
        self.__ReservationID = reservation_id
        self.__CustomerID = customer_id
        self.__VehicleID = vehicle_id
        self.__StartDate = start_date
        self.__EndDate = end_date
        self.__TotalCost = total_cost
        self.__Status = status

    # Getter methods with @property decorator
    @property
    def reservation_id(self):
        return self.__ReservationID

    @property
    def customer_id(self):
        return self.__CustomerID

    @property
    def vehicle_id(self):
        return self.__VehicleID

    @property
    def start_date(self):
        return self.__StartDate
```

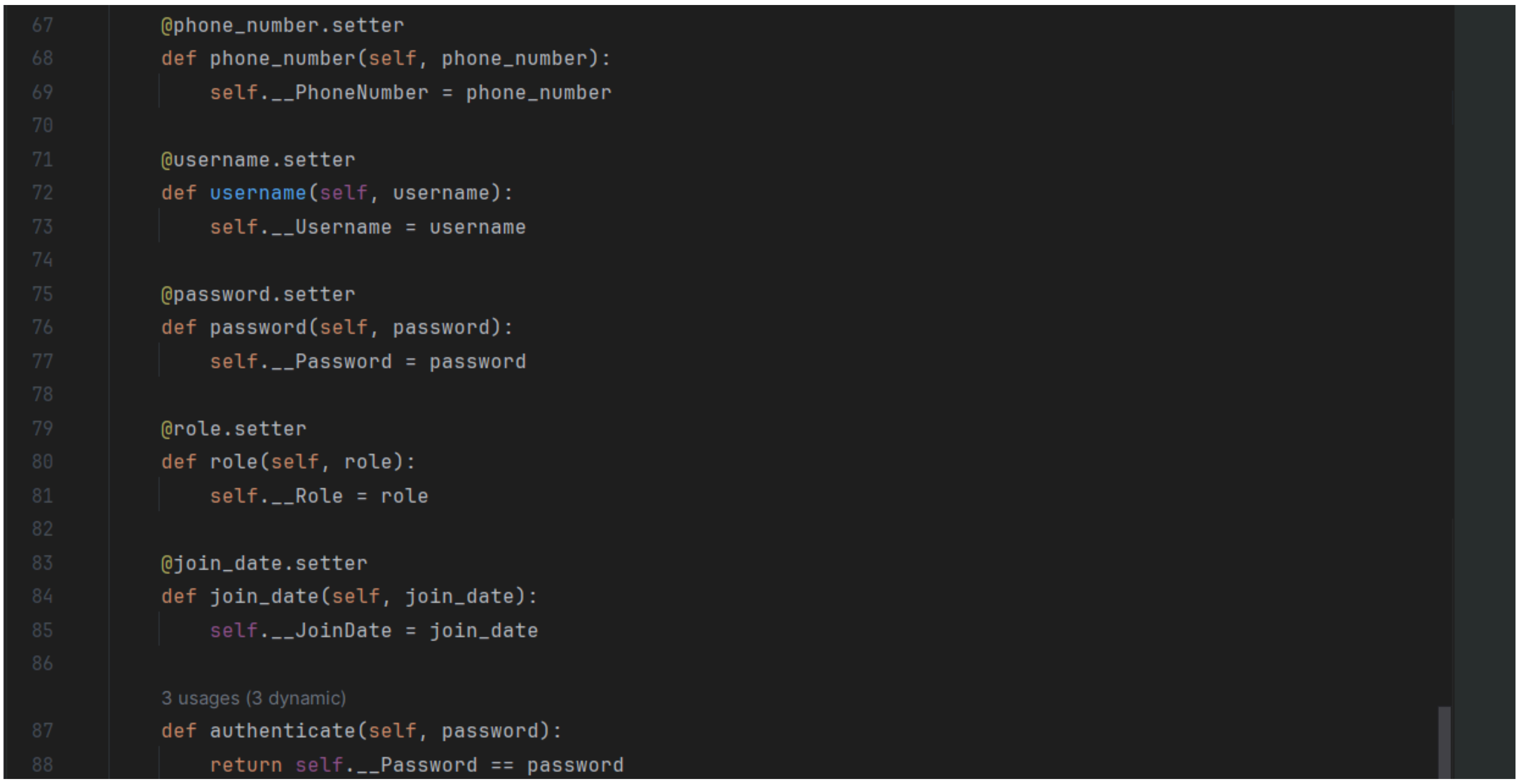


```python
    # Setter methods with @<property_name>.setter decorator
    @reservation_id.setter
    def reservation_id(self, reservation_id):
        self.__ReservationID = reservation_id

    @customer_id.setter
    def customer_id(self, customer_id):
        self.__CustomerID = customer_id

    @vehicle_id.setter
    def vehicle_id(self, vehicle_id):
        self.__VehicleID = vehicle_id

    @start_date.setter
    def start_date(self, start_date):
        self.__StartDate = start_date

    @end_date.setter
    def end_date(self, end_date):
        self.__EndDate = end_date

    @total_cost.setter
    def total_cost(self, total_cost):
        self.__TotalCost = total_cost

    @status.setter
    def status(self, status):
        self.__Status = status
```

Admin



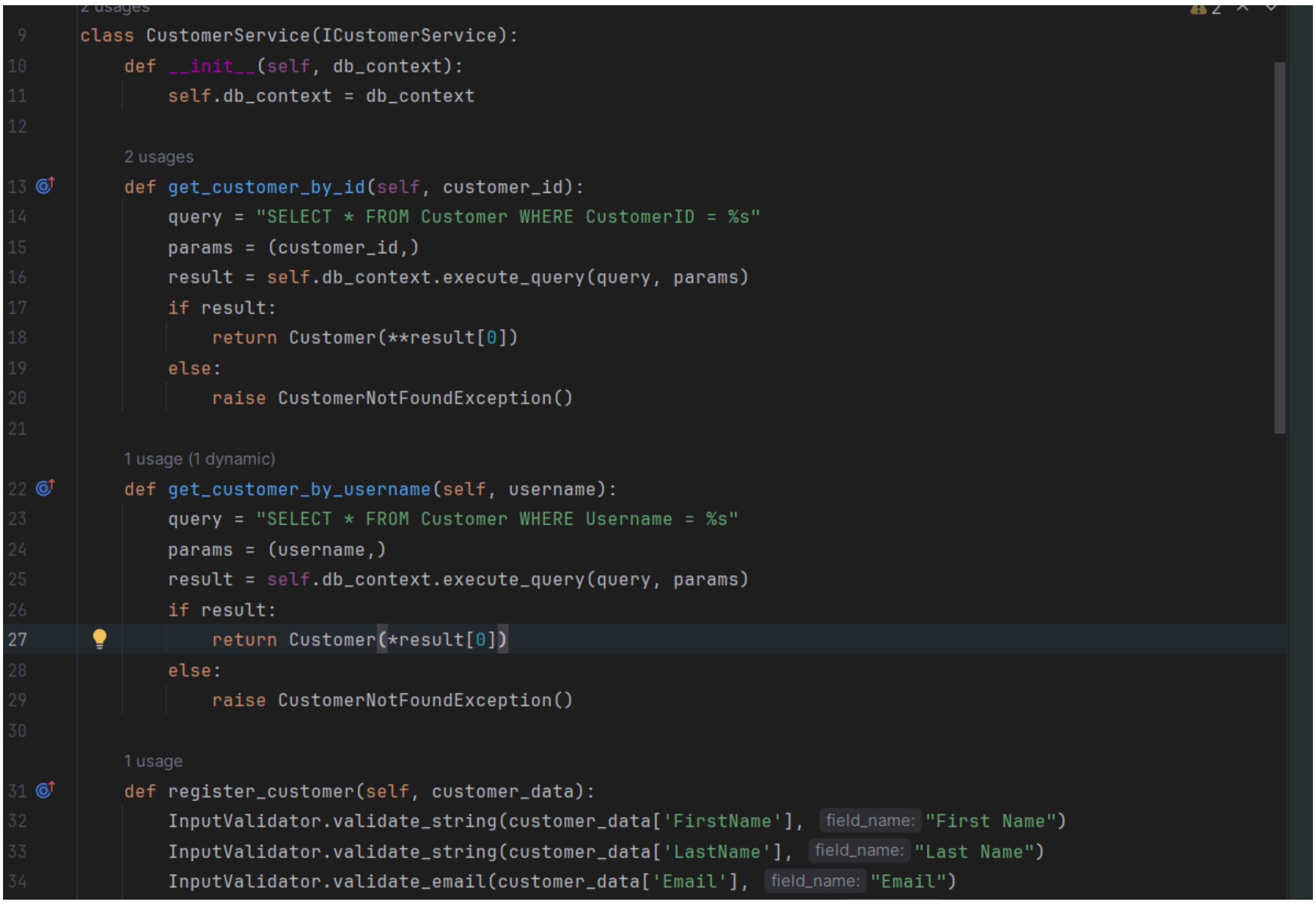
```
3 usages
class Admin:
    def __init__(self, admin_id, first_name, last_name, email, phone_number, username, password, role, join_da
        self.__AdminID = admin_id
        self.__FirstName = first_name
        self.__LastName = last_name
        self.__Email = email
        self.__PhoneNumber = phone_number
        self.__Username = username
        self.__Password = password
        self.__Role = role
        self.__JoinDate = join_date

    # Getter methods with @property decorator
    1 usage
    @property
    def admin_id(self):
        return self.__AdminID

    1 usage
    @property
    def first_name(self):
        return self.__FirstName

    1 usage
    @property
    def last_name(self):
        return self.__LastName

    1 usage
    @property
```



```
    @phone_number.setter
    def phone_number(self, phone_number):
        self.__PhoneNumber = phone_number

    @username.setter
    def username(self, username):
        self.__Username = username

    @password.setter
    def password(self, password):
        self.__Password = password

    @role.setter
    def role(self, role):
        self.__Role = role

    @join_date.setter
    def join_date(self, join_date):
        self.__JoinDate = join_date

    3 usages (3 dynamic)
    def authenticate(self, password):
        return self.__Password == password
```

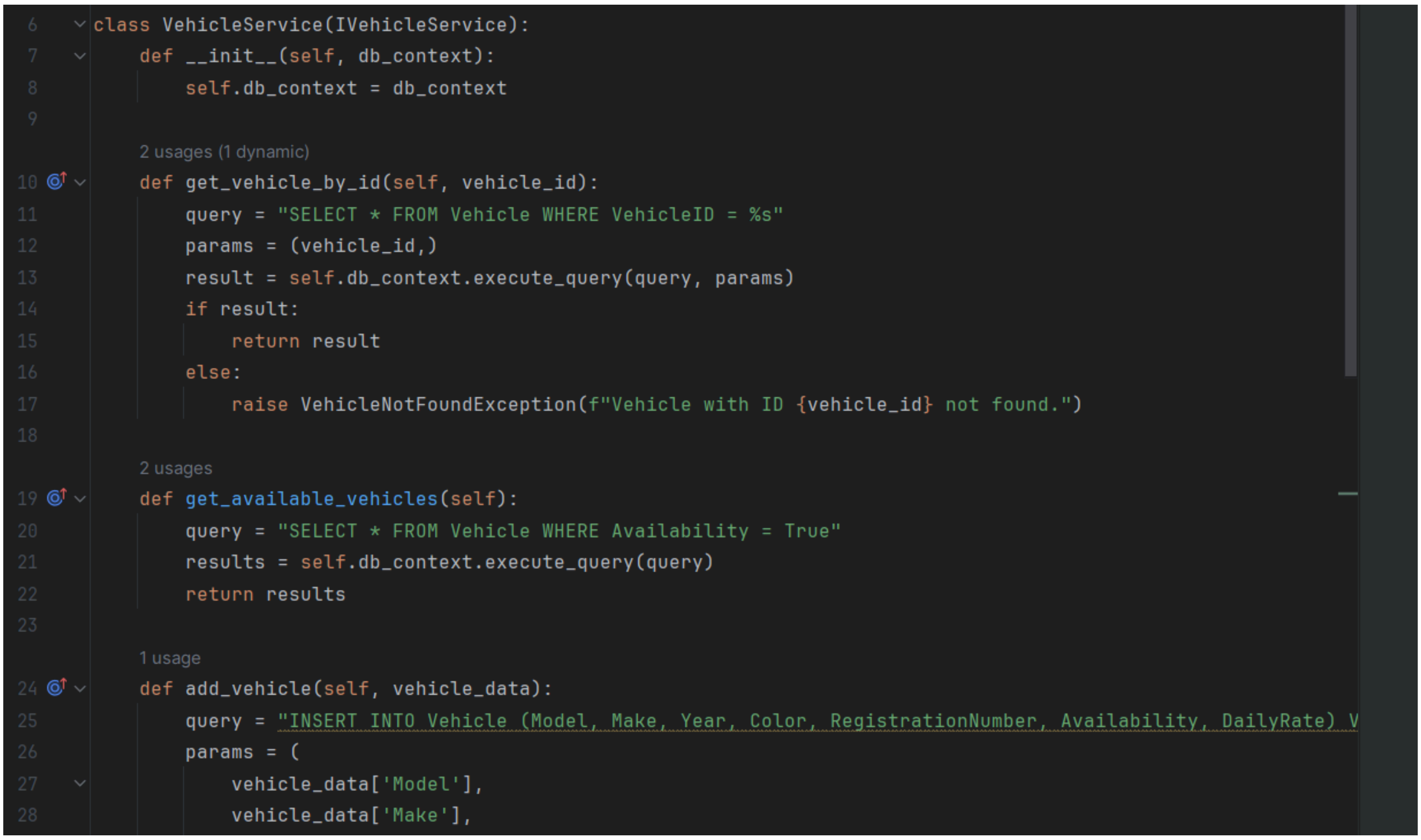
Services

CustomerService (implements ICustomerService)



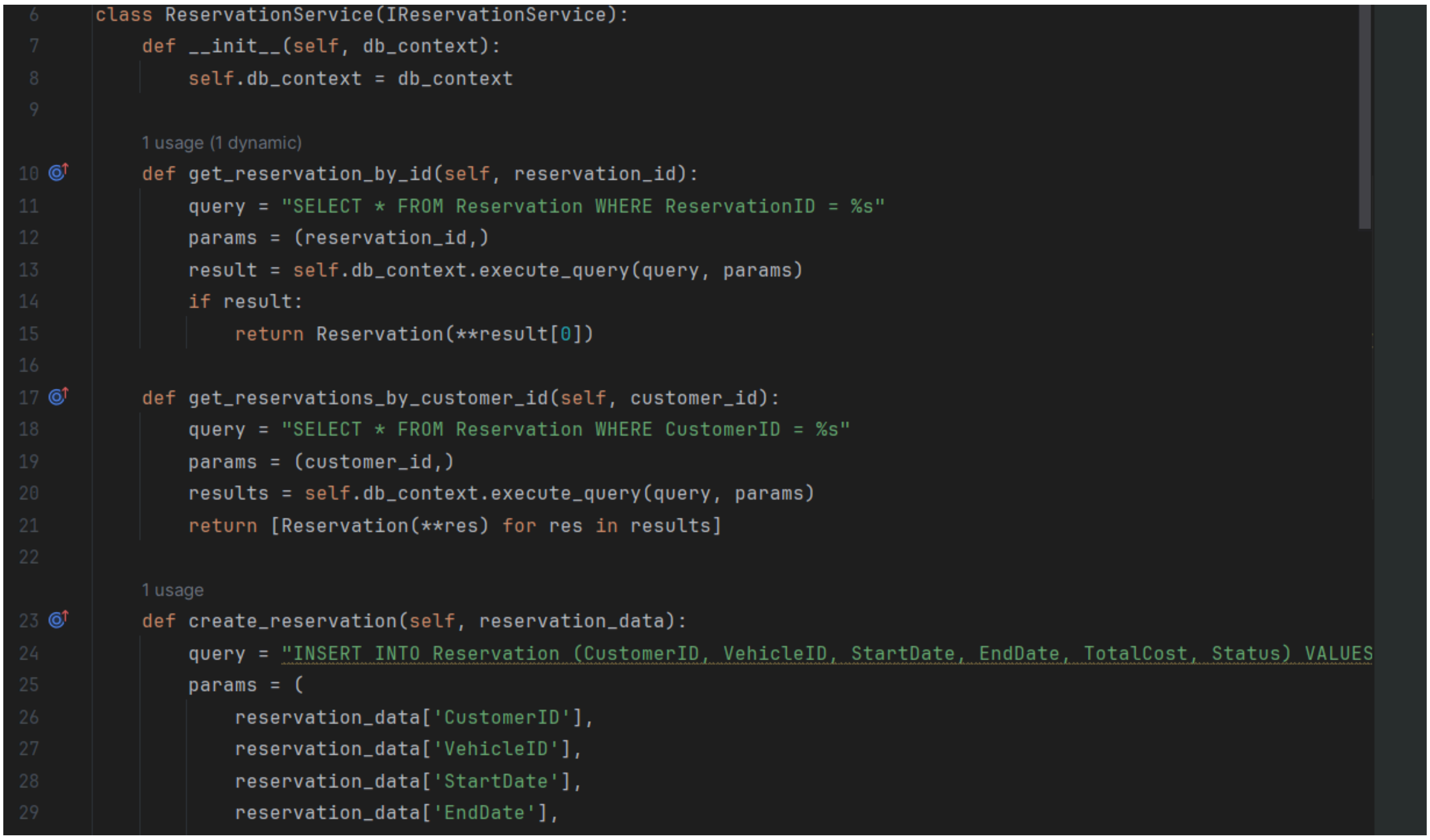
```python
class CustomerService(ICustomerService):
    def __init__(self, db_context):
        self.db_context = db_context

    def get_customer_by_id(self, customer_id):
        query = "SELECT * FROM Customer WHERE CustomerID = %s"
        params = (customer_id,)
        result = self.db_context.execute_query(query, params)
        if result:
            return Customer(**result[0])
        else:
            raise CustomerNotFoundException()

    def get_customer_by_username(self, username):
        query = "SELECT * FROM Customer WHERE Username = %s"
        params = (username,)
        result = self.db_context.execute_query(query, params)
        if result:
            return Customer(*result[0])
        else:
            raise CustomerNotFoundException()

    def register_customer(self, customer_data):
        InputValidator.validate_string(customer_data['FirstName'], "First Name")
        InputValidator.validate_string(customer_data['LastName'], "Last Name")
        InputValidator.validate_email(customer_data['Email'], "Email")
```

VehicleService (implements IVehicleService)



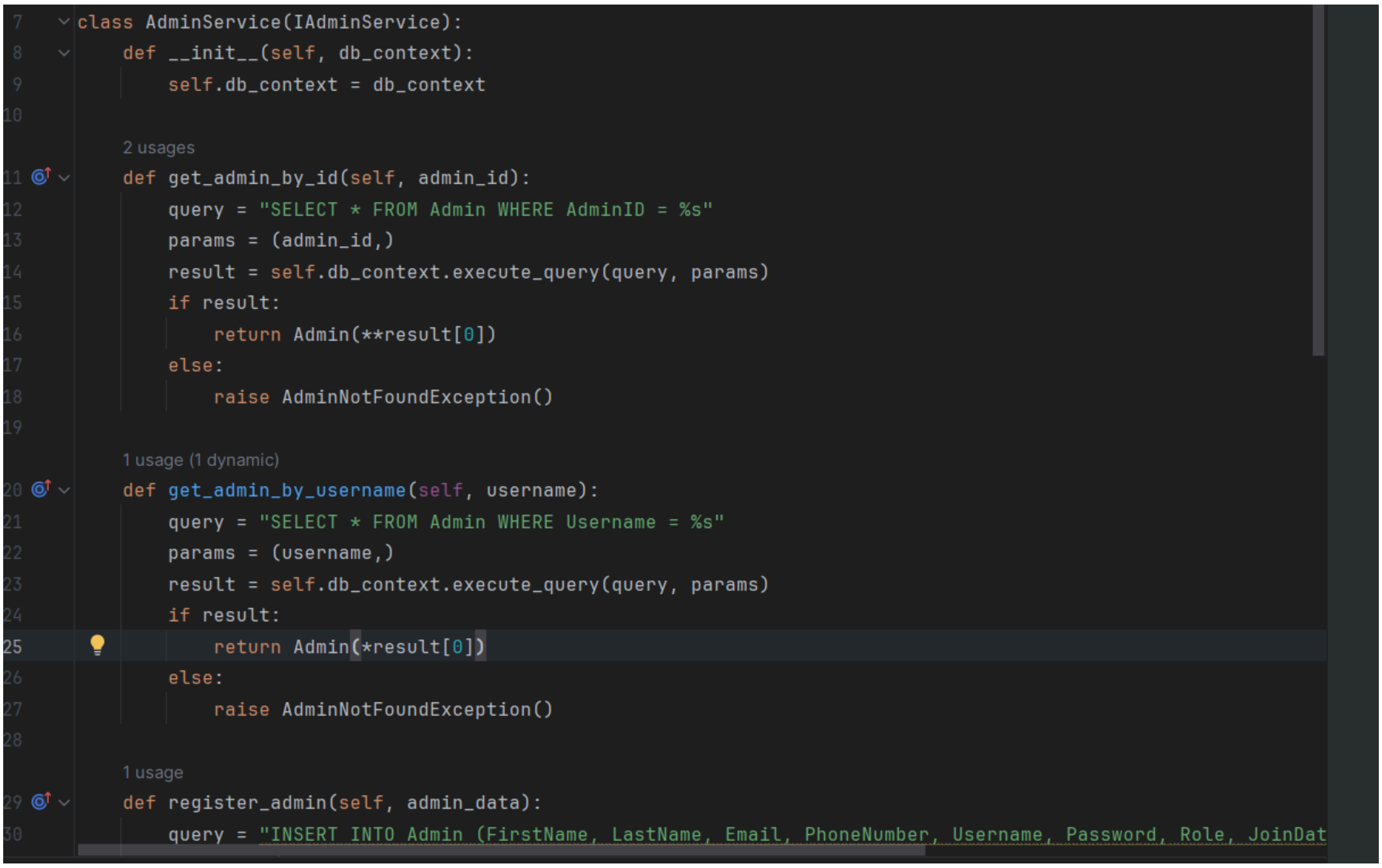
```python
class VehicleService(IVehicleService):
    def __init__(self, db_context):
        self.db_context = db_context

    def get_vehicle_by_id(self, vehicle_id):
        query = "SELECT * FROM Vehicle WHERE VehicleID = %s"
        params = (vehicle_id,)
        result = self.db_context.execute_query(query, params)
        if result:
            return result
        else:
            raise VehicleNotFoundException(f"Vehicle with ID {vehicle_id} not found.")

    def get_available_vehicles(self):
        query = "SELECT * FROM Vehicle WHERE Availability = True"
        results = self.db_context.execute_query(query)
        return results

    def add_vehicle(self, vehicle_data):
        query = "INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate) V
        params = (
            vehicle_data['Model'],
            vehicle_data['Make'],
```

ReservationService (implements IReservationService)



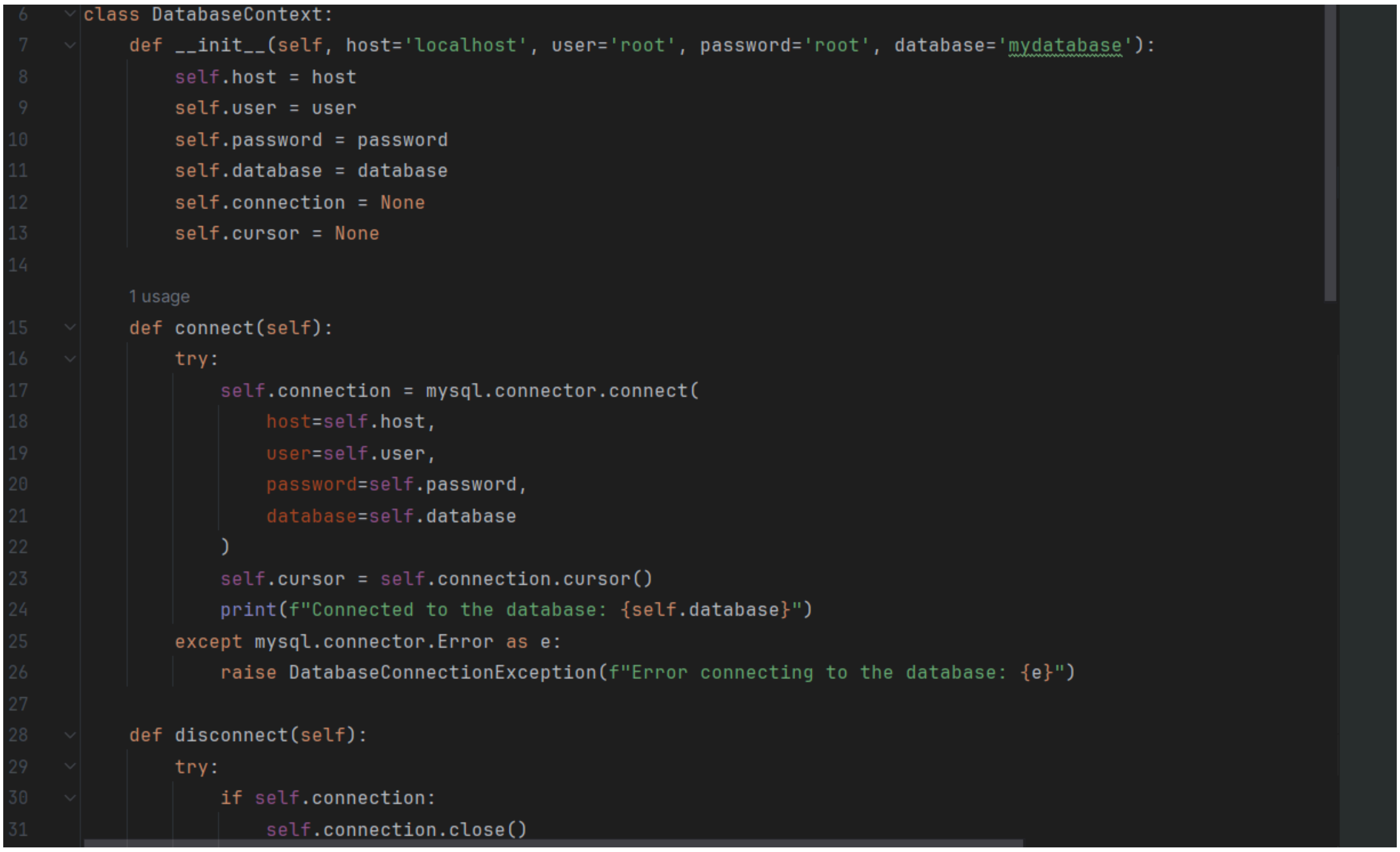
```
class ReservationService(IReservationService):
    def __init__(self, db_context):
        self.db_context = db_context

    1 usage (1 dynamic)
    def get_reservation_by_id(self, reservation_id):
        query = "SELECT * FROM Reservation WHERE ReservationID = %s"
        params = (reservation_id,)
        result = self.db_context.execute_query(query, params)
        if result:
            return Reservation(**result[0])

    def get_reservations_by_customer_id(self, customer_id):
        query = "SELECT * FROM Reservation WHERE CustomerID = %s"
        params = (customer_id,)
        results = self.db_context.execute_query(query, params)
        return [Reservation(**res) for res in results]

    1 usage
    def create_reservation(self, reservation_data):
        query = "INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status) VALUES
        params = (
            reservation_data['CustomerID'],
            reservation_data['VehicleID'],
            reservation_data['StartDate'],
            reservation_data['EndDate'],
```

AdminService (implements IAdminService)



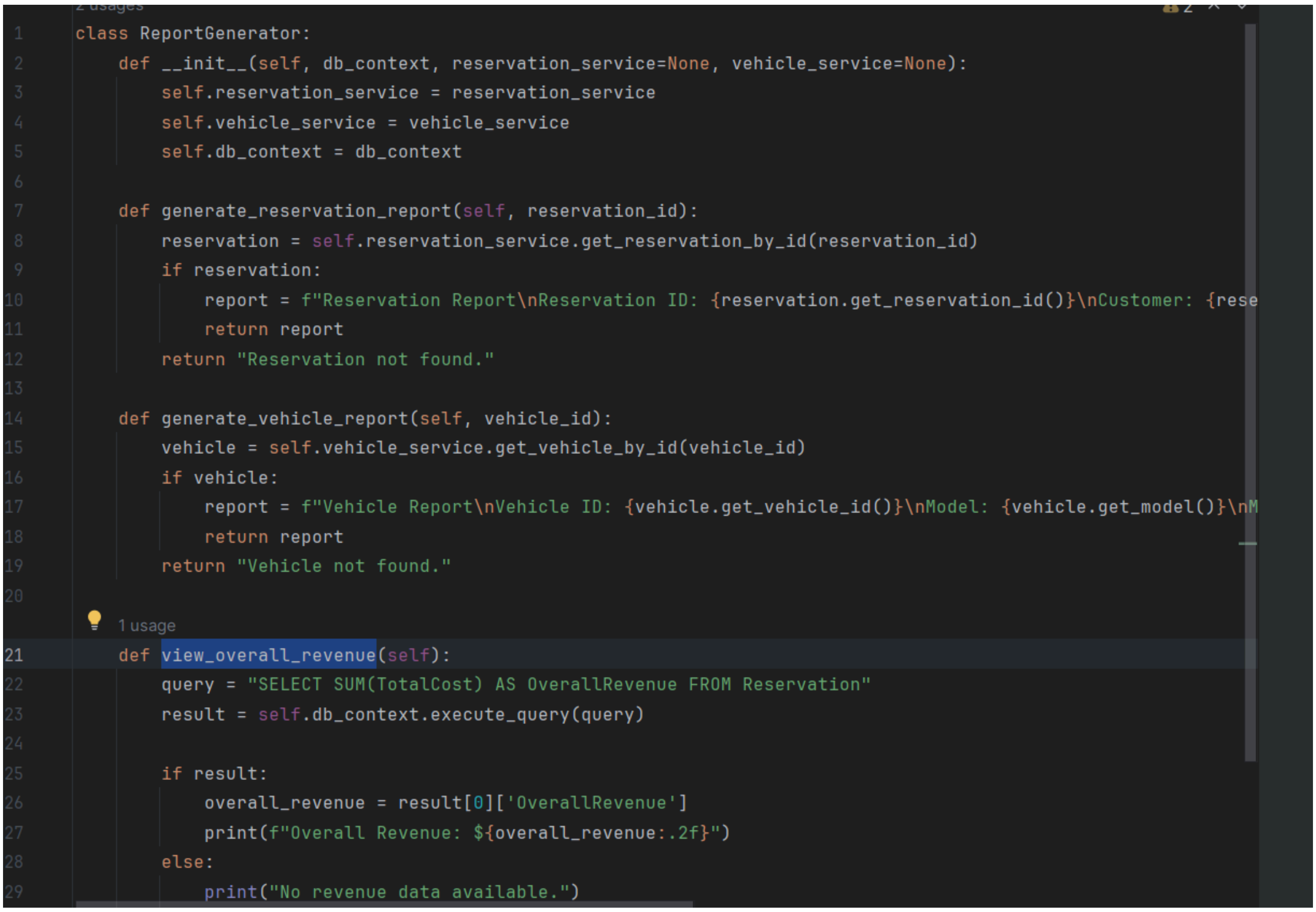
```
class AdminService(IAdminService):
    def __init__(self, db_context):
        self.db_context = db_context

    2 usages
    def get_admin_by_id(self, admin_id):
        query = "SELECT * FROM Admin WHERE AdminID = %s"
        params = (admin_id,)
        result = self.db_context.execute_query(query, params)
        if result:
            return Admin(**result[0])
        else:
            raise AdminNotFoundException()

    1 usage (1 dynamic)
    def get_admin_by_username(self, username):
        query = "SELECT * FROM Admin WHERE Username = %s"
        params = (username,)
        result = self.db_context.execute_query(query, params)
        if result:
            return Admin(*result[0])
        else:
            raise AdminNotFoundException()

    1 usage
    def register_admin(self, admin_data):
        query = "INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDat
```

DatabaseContext



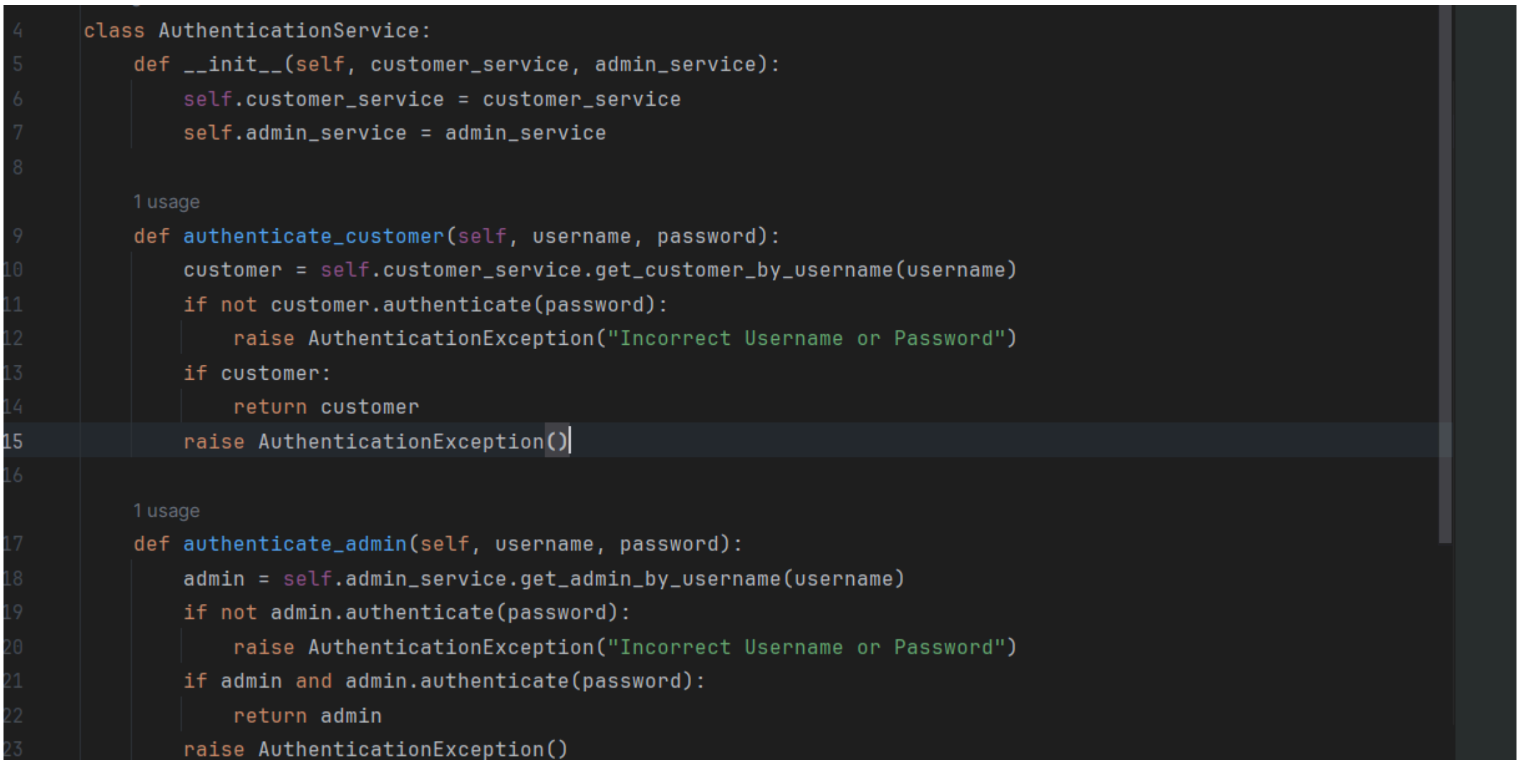
```python
class DatabaseContext:
    def __init__(self, host='localhost', user='root', password='root', database='mydatabase'):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None
        self.cursor = None

    1 usage
    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
            self.cursor = self.connection.cursor()
            print(f"Connected to the database: {self.database}")
        except mysql.connector.Error as e:
            raise DatabaseConnectionException(f"Error connecting to the database: {e}")

    def disconnect(self):
        try:
            if self.connection:
                self.connection.close()
```

ReportGenerator



```python
class ReportGenerator:
    def __init__(self, db_context, reservation_service=None, vehicle_service=None):
        self.reservation_service = reservation_service
        self.vehicle_service = vehicle_service
        self.db_context = db_context

    def generate_reservation_report(self, reservation_id):
        reservation = self.reservation_service.get_reservation_by_id(reservation_id)
        if reservation:
            report = f"Reservation Report\nReservation ID: {reservation.get_reservation_id()}\nCustomer: {rese
            return report
        return "Reservation not found."

    def generate_vehicle_report(self, vehicle_id):
        vehicle = self.vehicle_service.get_vehicle_by_id(vehicle_id)
        if vehicle:
            report = f"Vehicle Report\nVehicle ID: {vehicle.get_vehicle_id()}\nModel: {vehicle.get_model()}\nM
            return report
        return "Vehicle not found."

    1 usage
    def view_overall_revenue(self):
        query = "SELECT SUM(TotalCost) AS OverallRevenue FROM Reservation"
        result = self.db_context.execute_query(query)

        if result:
            overall_revenue = result[0]['OverallRevenue']
            print(f"Overall Revenue: ${overall_revenue:.2f}")
        else:
            print("No revenue data available.")
```

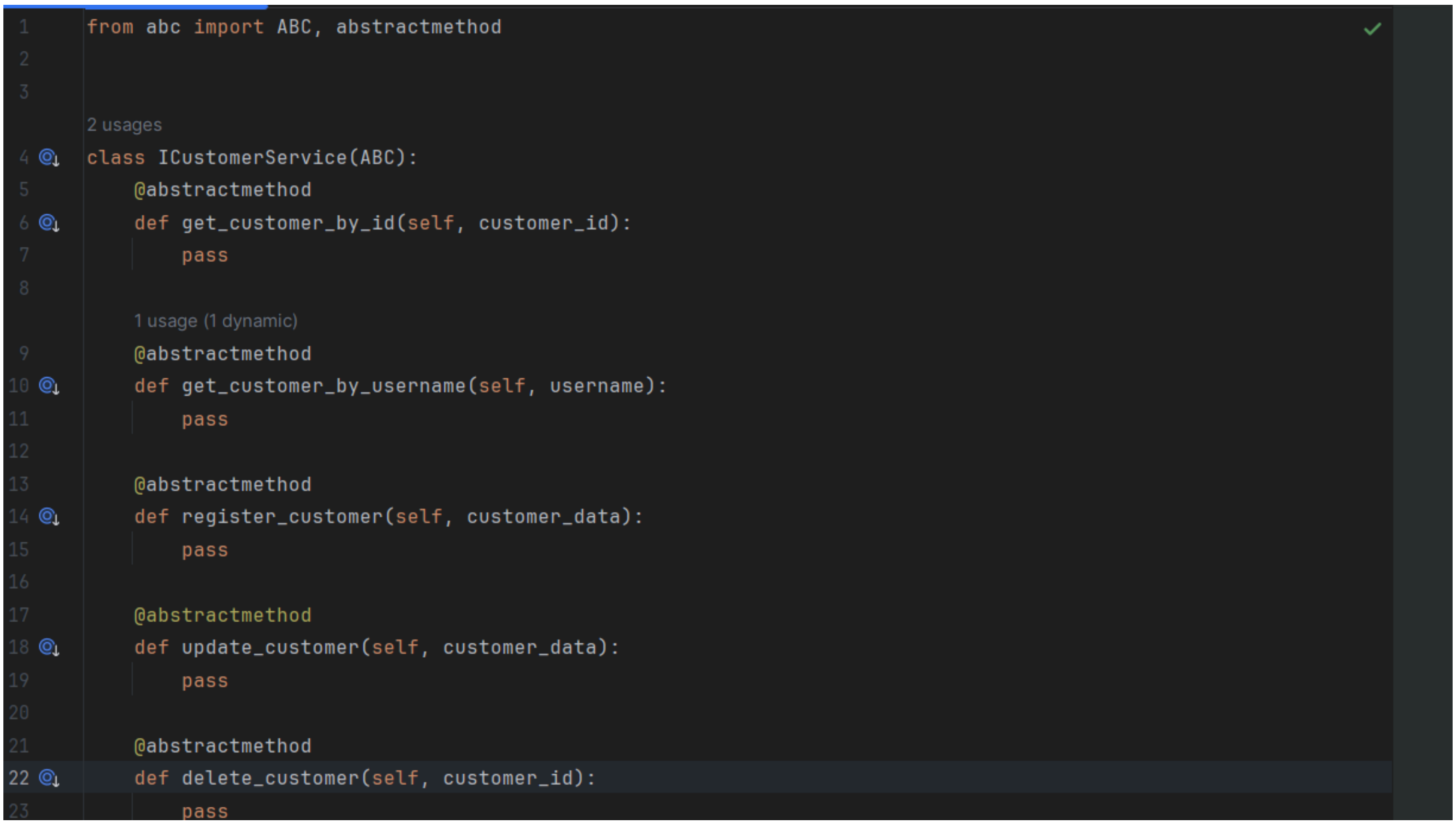
Authentication



```
class AuthenticationService:
    def __init__(self, customer_service, admin_service):
        self.customer_service = customer_service
        self.admin_service = admin_service

    1 usage
    def authenticate_customer(self, username, password):
        customer = self.customer_service.get_customer_by_username(username)
        if not customer.authenticate(password):
            raise AuthenticationException("Incorrect Username or Password")
        if customer:
            return customer
        raise AuthenticationException()

    1 usage
    def authenticate_admin(self, username, password):
        admin = self.admin_service.get_admin_by_username(username)
        if not admin.authenticate(password):
            raise AuthenticationException("Incorrect Username or Password")
        if admin and admin.authenticate(password):
            return admin
        raise AuthenticationException()
```

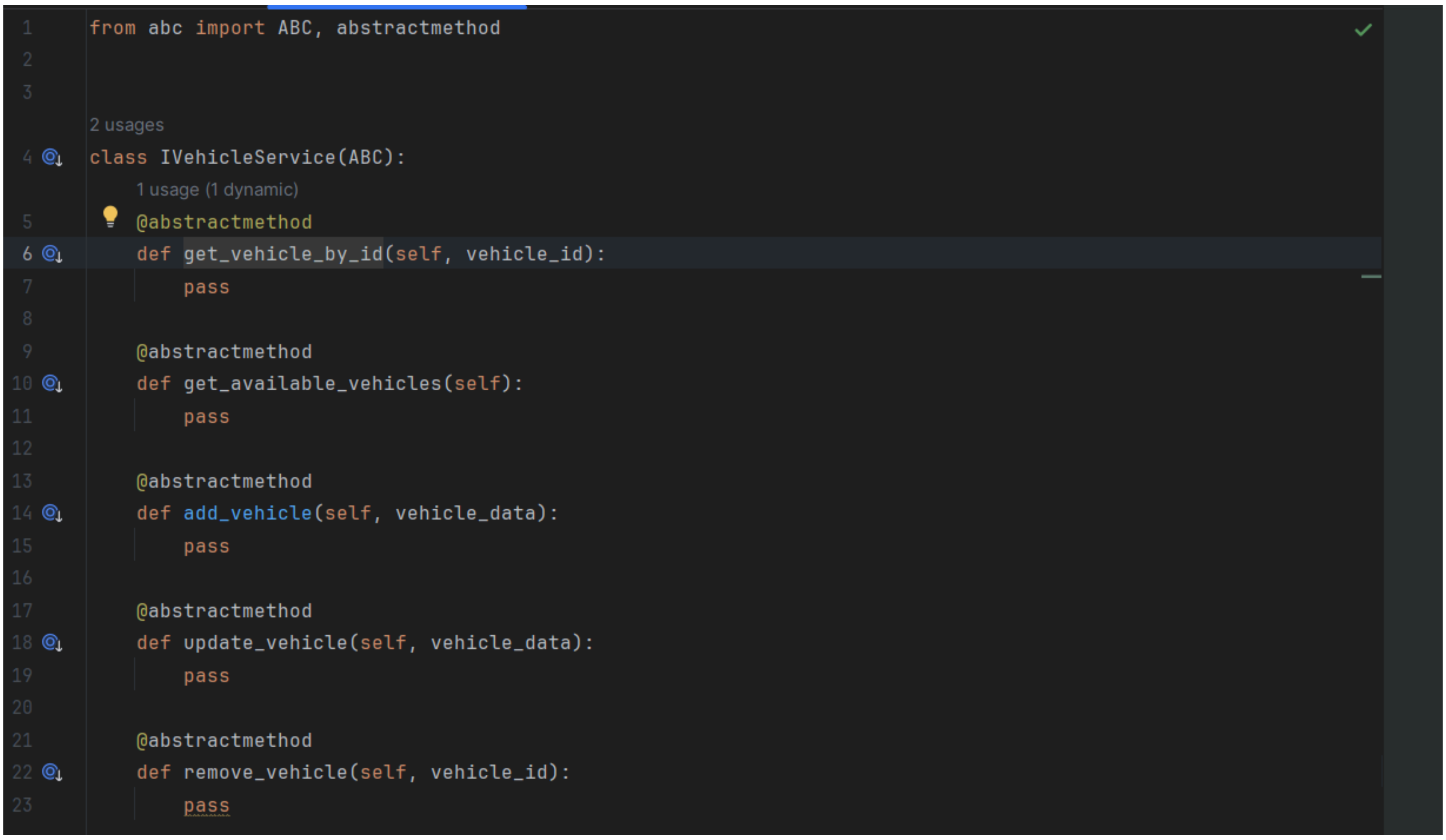
Interfaces

ICustomerService



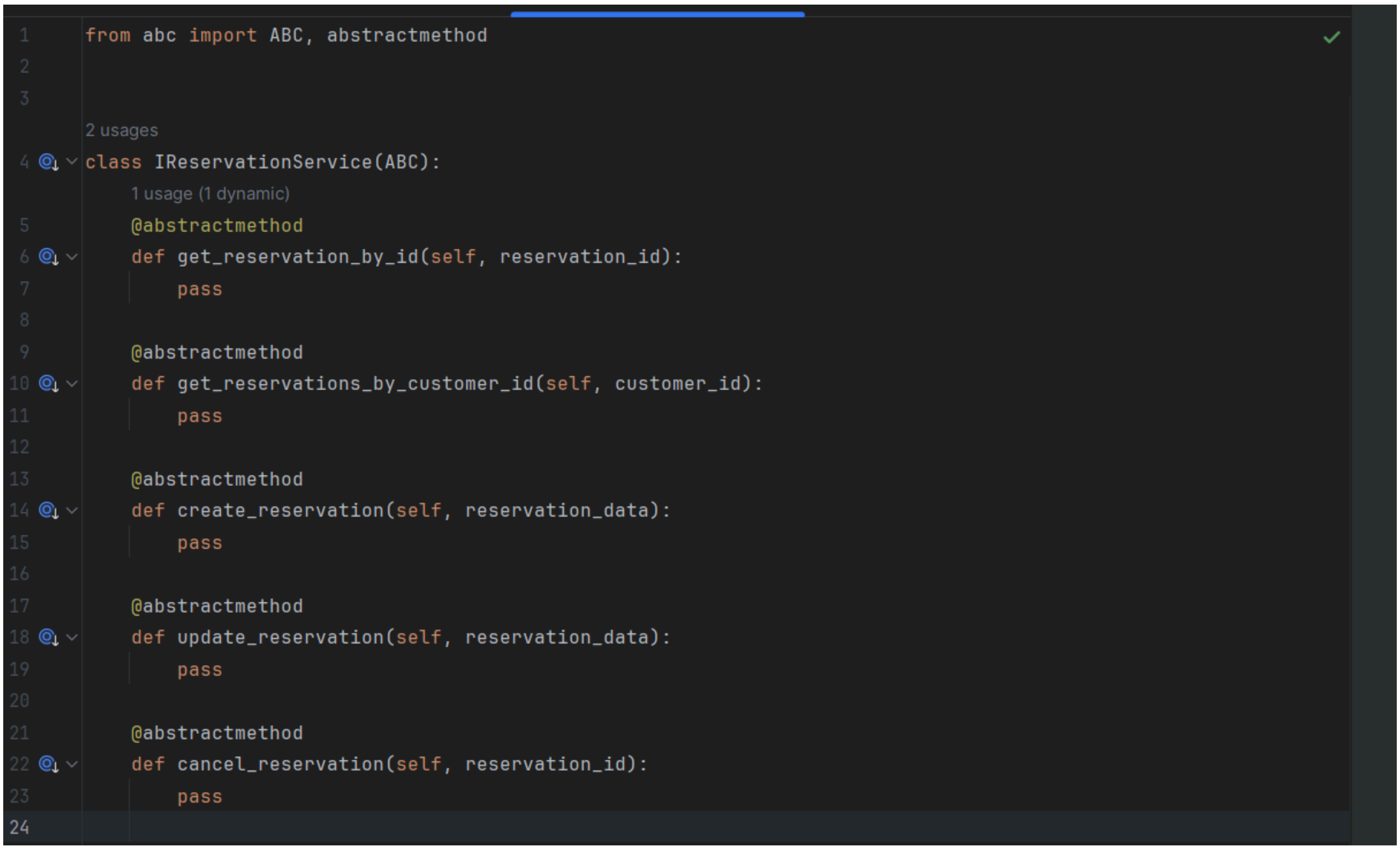
```
from abc import ABC, abstractmethod


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

    1 usage (1 dynamic)
    @abstractmethod
    def get_customer_by_username(self, username):
        pass

    @abstractmethod
    def register_customer(self, customer_data):
        pass

    @abstractmethod
    def update_customer(self, customer_data):
        pass

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

IVehicleService



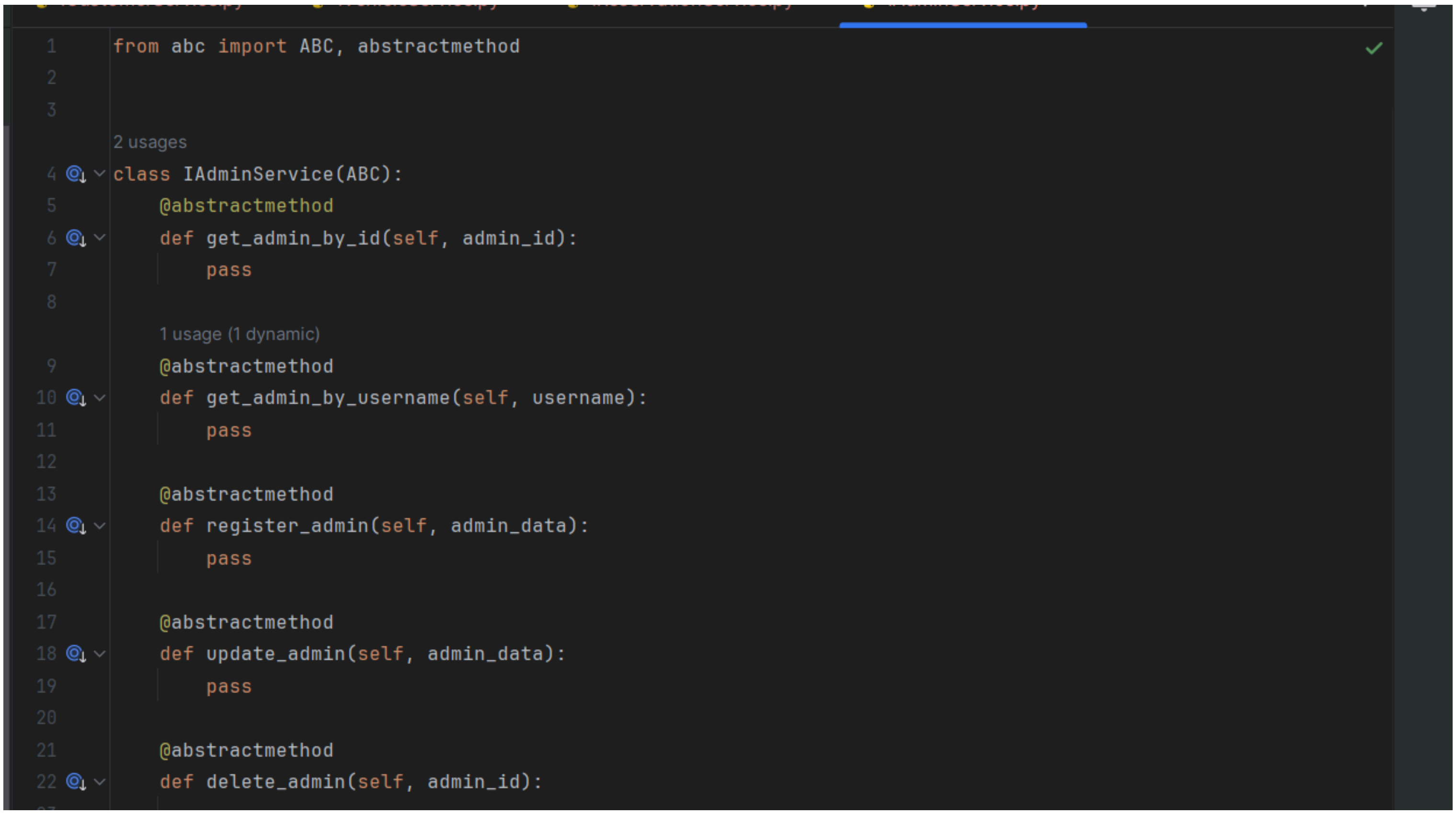
```
from abc import ABC, abstractmethod


2 usages
class IVehicleService(ABC):
    1 usage (1 dynamic)
    @abstractmethod
    def get_vehicle_by_id(self, vehicle_id):
        pass

    @abstractmethod
    def get_available_vehicles(self):
        pass

    @abstractmethod
    def add_vehicle(self, vehicle_data):
        pass

    @abstractmethod
    def update_vehicle(self, vehicle_data):
        pass

    @abstractmethod
    def remove_vehicle(self, vehicle_id):
        pass
```

IReservationService



```
from abc import ABC, abstractmethod


2 usages
class IReservationService(ABC):
    1 usage (1 dynamic)
    @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, reservation_data):
        pass

    @abstractmethod
    def update_reservation(self, reservation_data):
        pass

    @abstractmethod
    def cancel_reservation(self, reservation_id):
        pass

```

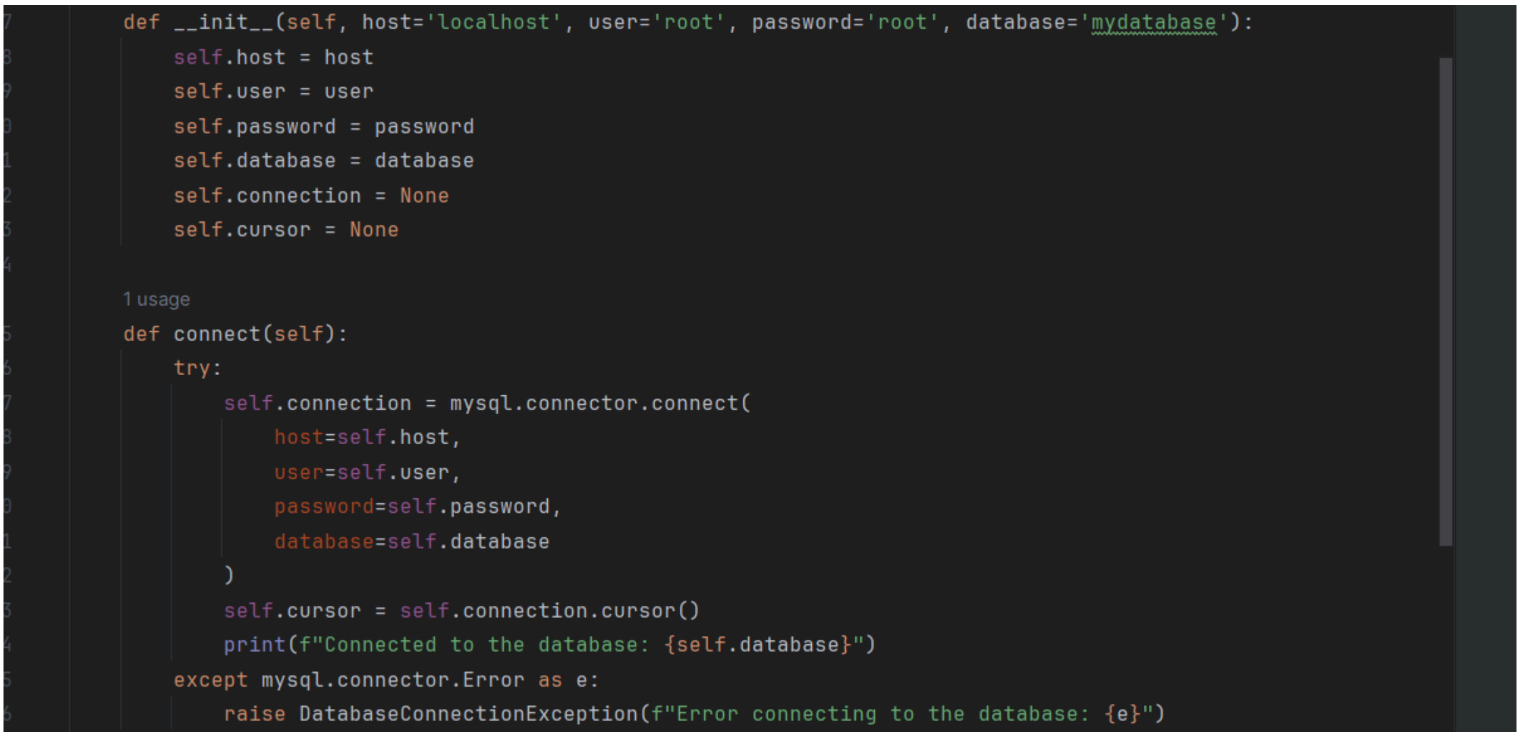
IAdminService



```python
from abc import ABC, abstractmethod


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

    1 usage (1 dynamic)
    @abstractmethod
    def get_admin_by_username(self, username):
        pass

    @abstractmethod
    def register_admin(self, admin_data):
        pass

    @abstractmethod
    def update_admin(self, admin_data):
        pass

    @abstractmethod
    def delete_admin(self, admin_id):
```

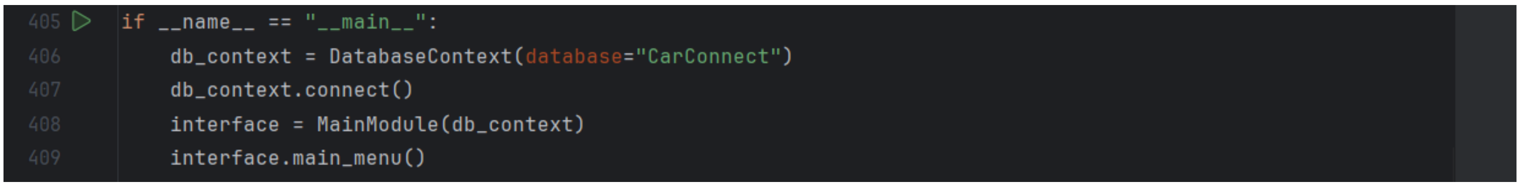
Connect your application to the SQL database

Database connection is done through mysql-python-connector.
DatabaseContext.py



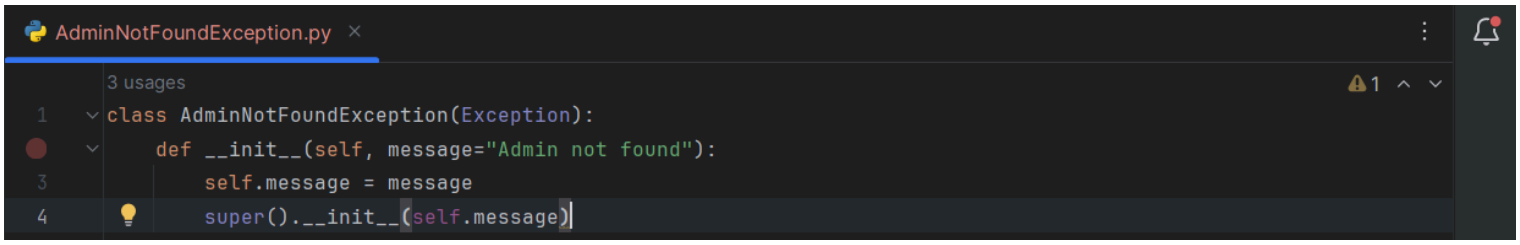
```python
    def __init__(self, host='localhost', user='root', password='root', database='mydatabase'):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None
        self.cursor = None

    1 usage
    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
            self.cursor = self.connection.cursor()
            print(f"Connected to the database: {self.database}")
        except mysql.connector.Error as e:
            raise DatabaseConnectionException(f"Error connecting to the database: {e}")
```

Initialized the connection before the menu display
MainModult.py

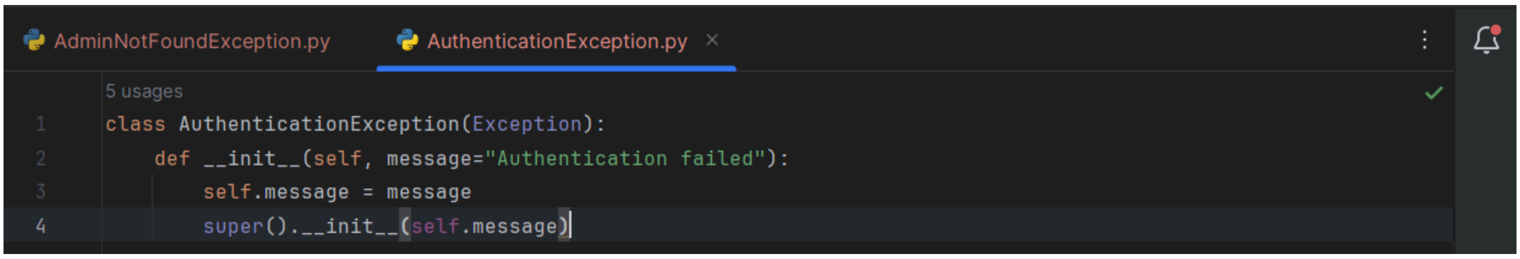


```python
if __name__ == "__main__":
    db_context = DatabaseContext(database="CarConnect")
    db_context.connect()
    interface = MainModule(db_context)
    interface.main_menu()
```

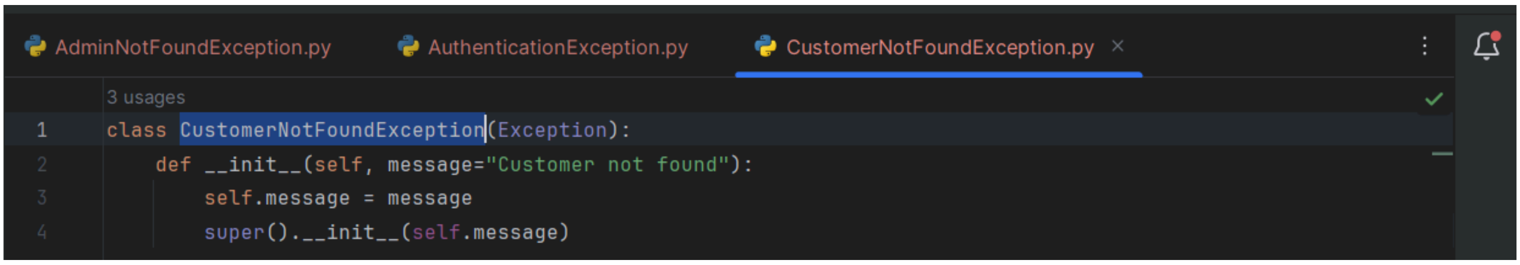
Exceptions


AdminNotFoundException.py

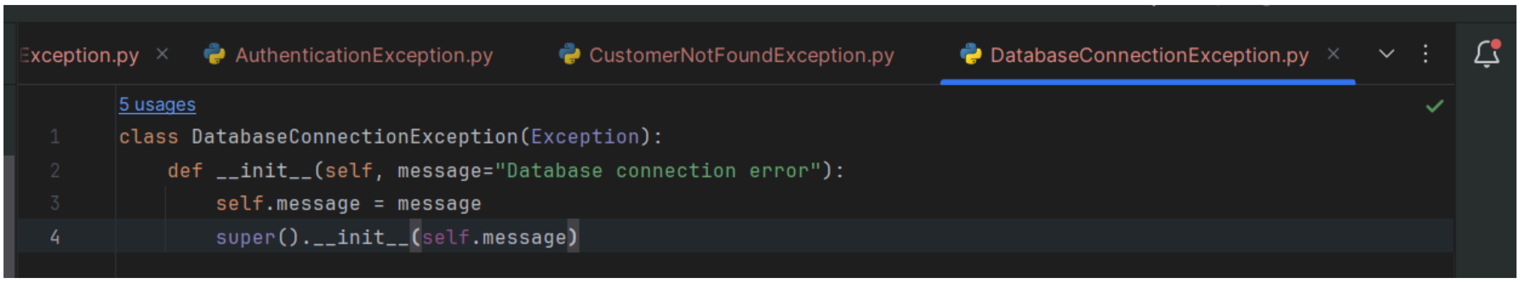
3 usages
```python
class AdminNotFoundException(Exception):
    def __init__(self, message="Admin not found"):
        self.message = message
        super().__init__(self.message)
```


AdminNotFoundException.py AuthenticationException.py

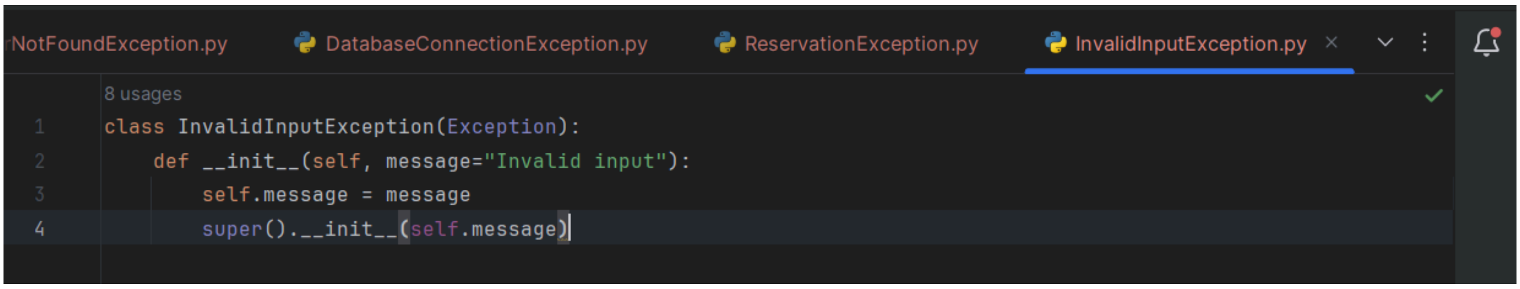
5 usages
```python
class AuthenticationException(Exception):
    def __init__(self, message="Authentication failed"):
        self.message = message
        super().__init__(self.message)
```


AdminNotFoundException.py AuthenticationException.py CustomerNotFoundException.py

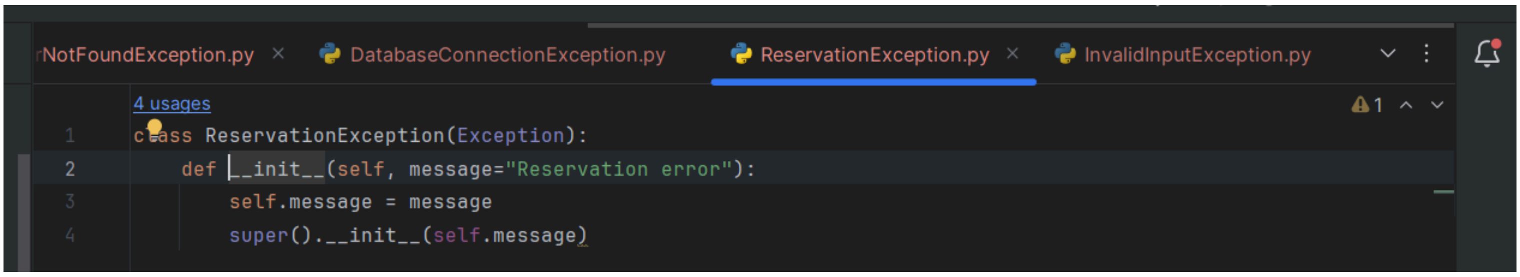
3 usages
```python
class CustomerNotFoundException(Exception):
    def __init__(self, message="Customer not found"):
        self.message = message
        super().__init__(self.message)
```


Exception.py AuthenticationException.py CustomerNotFoundException.py DatabaseConnectionException.py

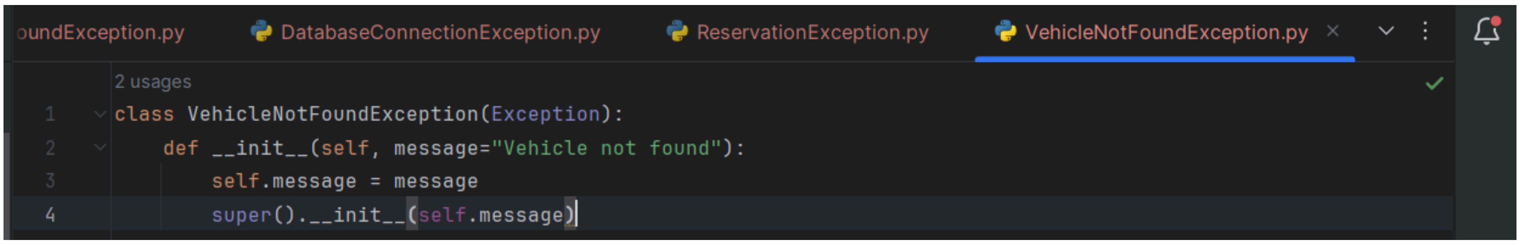
5 usages
```python
class DatabaseConnectionException(Exception):
    def __init__(self, message="Database connection error"):
        self.message = message
        super().__init__(self.message)
```


NotFoundException.py DatabaseConnectionException.py ReservationException.py InvalidInputException.py

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


NotFoundException.py DatabaseConnectionException.py ReservationException.py InvalidInputException.py

4 usages
```python
class ReservationException(Exception):
    def __init__(self, message="Reservation error"):
        self.message = message
        super().__init__(self.message)
```

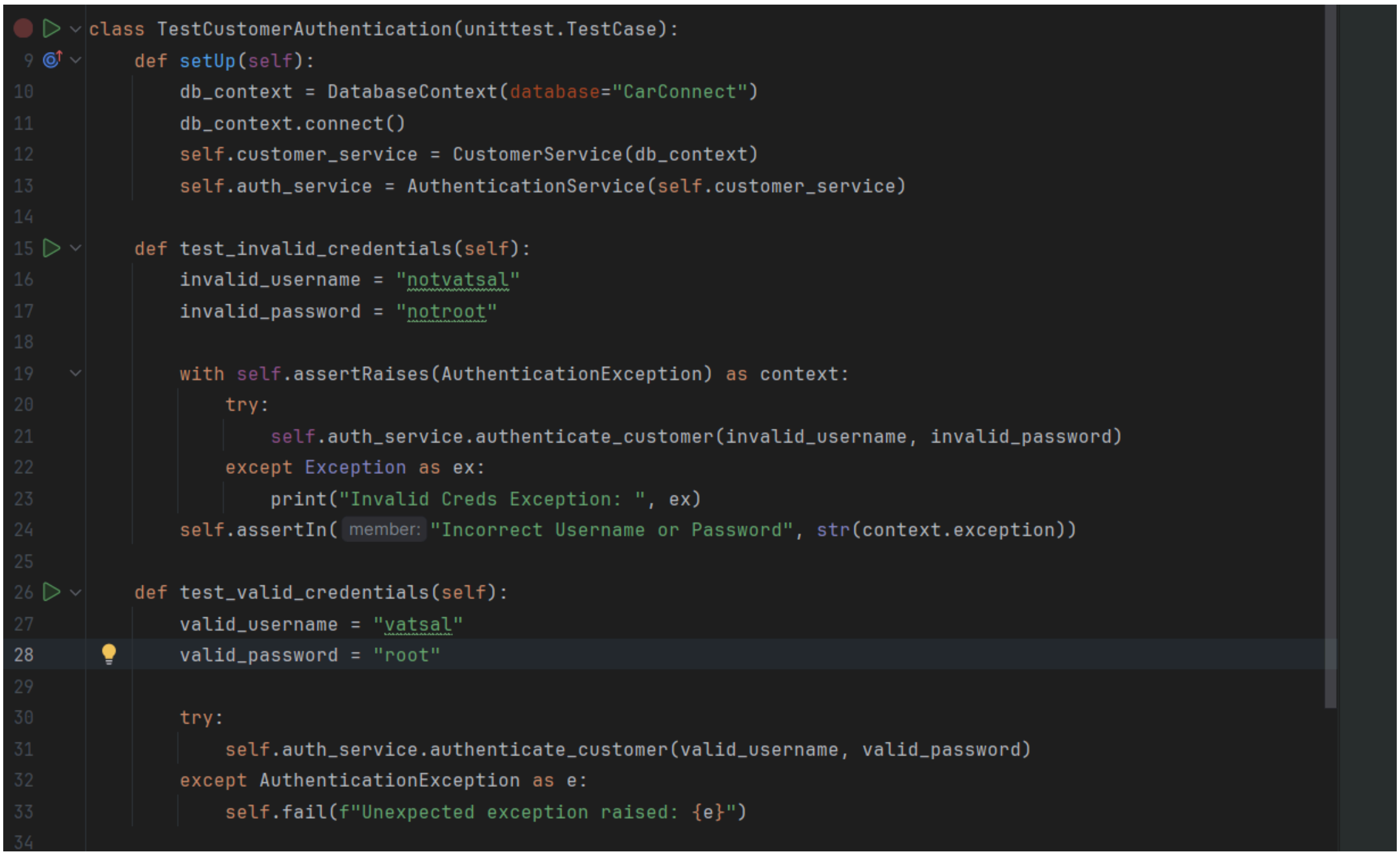

oundException.py DatabaseConnectionException.py ReservationException.py VehicleNotFoundException.py

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

Used all exceptions in appropriate places in the code.

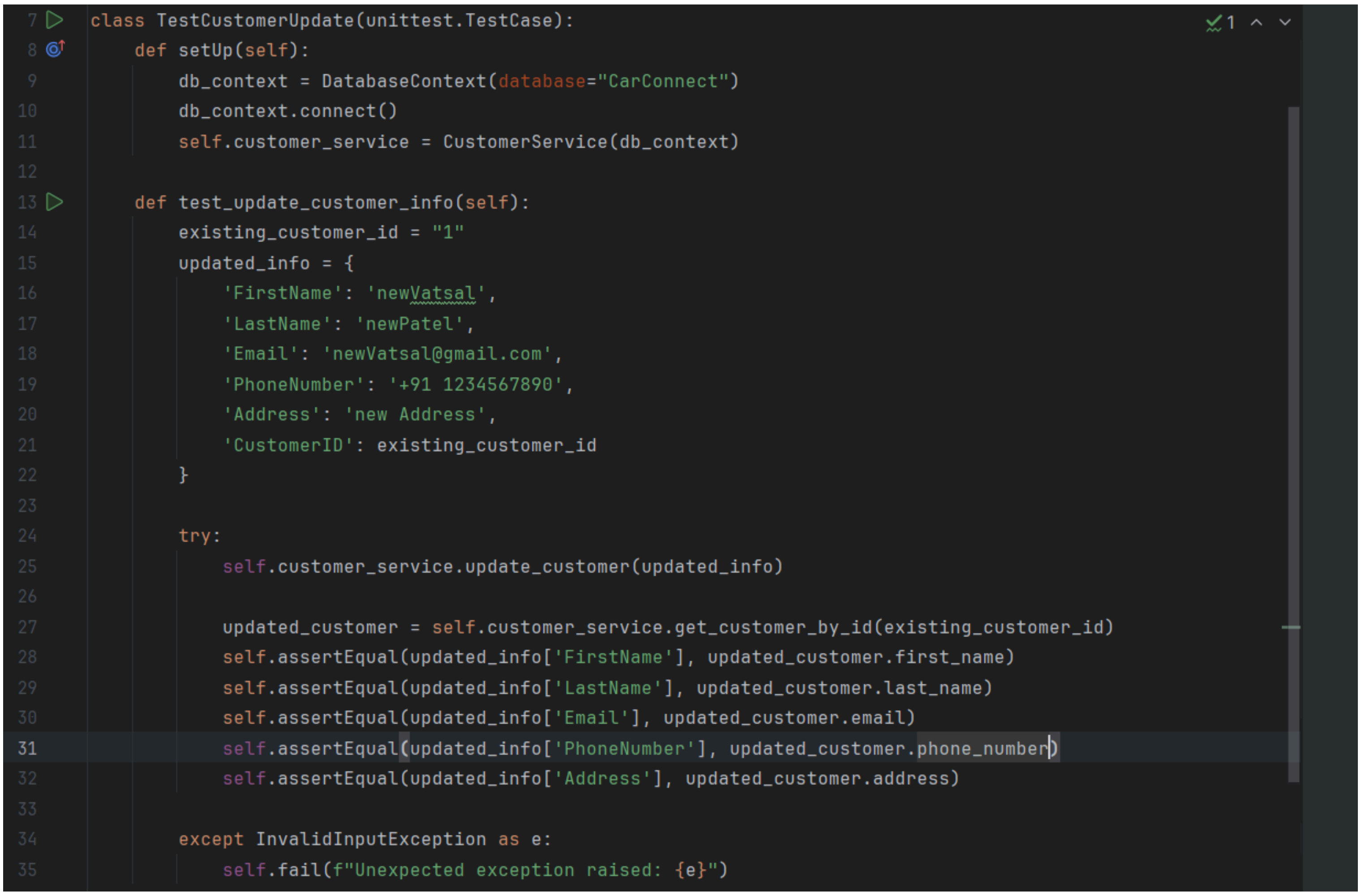
Unit Testing

1. Test customer authentication with invalid credentials.



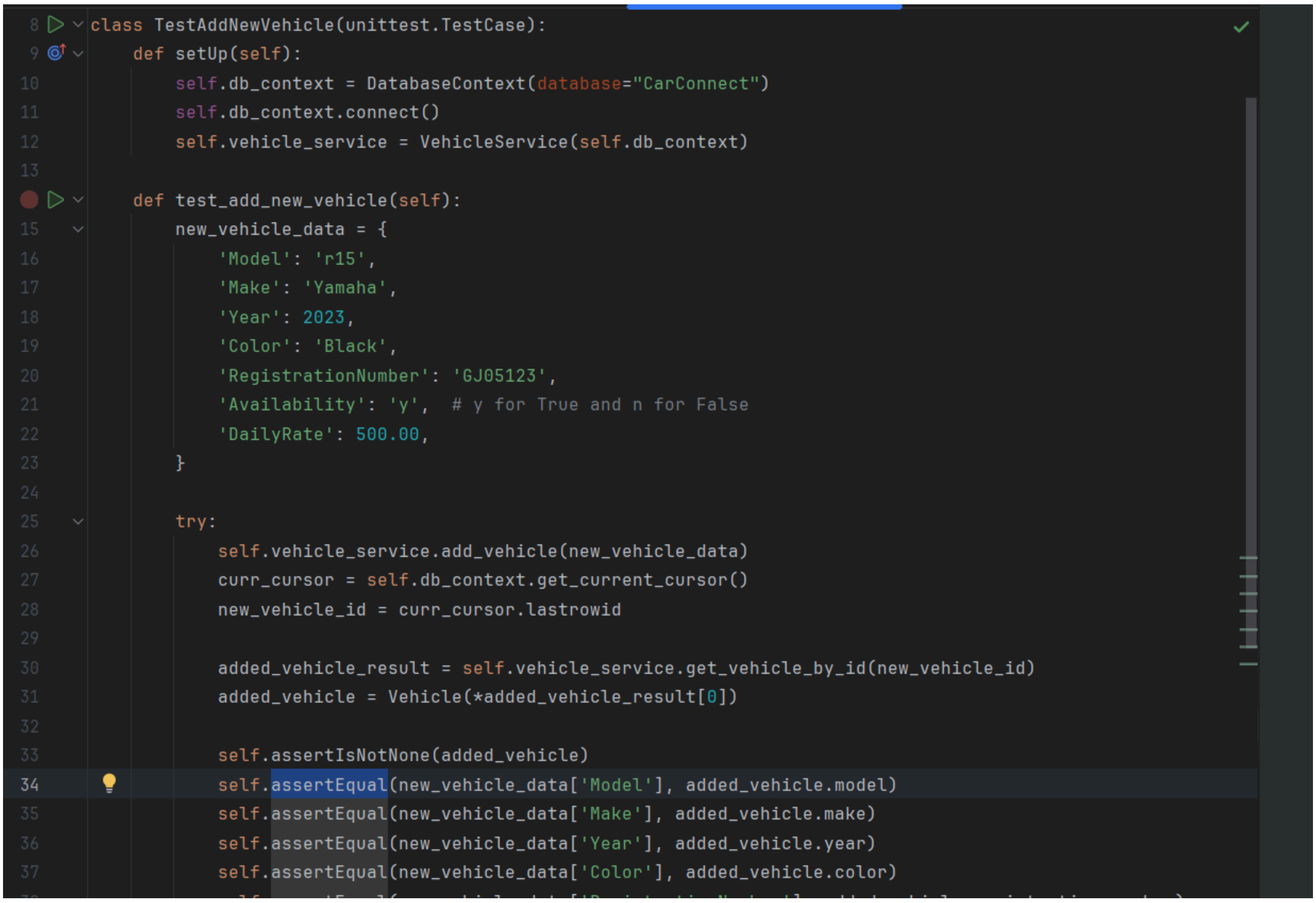
```
class TestCustomerAuthentication(unittest.TestCase):
    def setUp(self):
        db_context = DatabaseContext(database="CarConnect")
        db_context.connect()
        self.customer_service = CustomerService(db_context)
        self.auth_service = AuthenticationService(self.customer_service)

    def test_invalid_credentials(self):
        invalid_username = "notvatsal"
        invalid_password = "notroot"

        with self.assertRaises(AuthenticationException) as context:
            try:
                self.auth_service.authenticate_customer(invalid_username, invalid_password)
            except Exception as ex:
                print("Invalid Creds Exception: ", ex)
        self.assertIn( member: "Incorrect Username or Password", str(context.exception))

    def test_valid_credentials(self):
        valid_username = "vatsal"
        valid_password = "root"

        try:
            self.auth_service.authenticate_customer(valid_username, valid_password)
        except AuthenticationException as e:
            self.fail(f"Unexpected exception raised: {e}")
```

2. Test updating customer information.

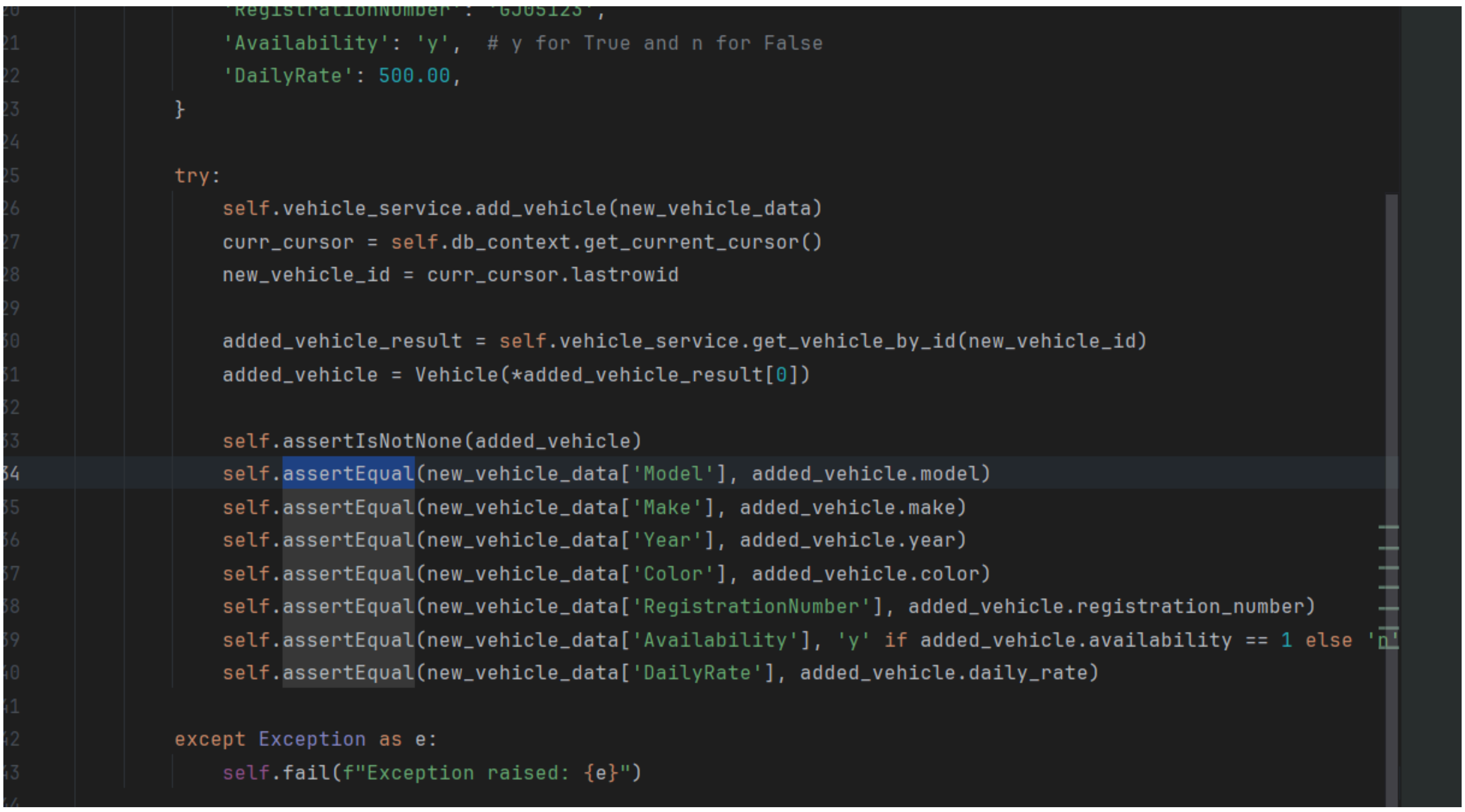


```
class TestCustomerUpdate(unittest.TestCase):
    def setUp(self):
        db_context = DatabaseContext(database="CarConnect")
        db_context.connect()
        self.customer_service = CustomerService(db_context)

    def test_update_customer_info(self):
        existing_customer_id = "1"
        updated_info = {
            'FirstName': 'newVatsal',
            'LastName': 'newPatel',
            'Email': 'newVatsal@gmail.com',
            'PhoneNumber': '+91 1234567890',
            'Address': 'new Address',
            'CustomerID': existing_customer_id
        }

        try:
            self.customer_service.update_customer(updated_info)

            updated_customer = self.customer_service.get_customer_by_id(existing_customer_id)
            self.assertEqual(updated_info['FirstName'], updated_customer.first_name)
            self.assertEqual(updated_info['LastName'], updated_customer.last_name)
            self.assertEqual(updated_info['Email'], updated_customer.email)
            self.assertEqual(updated_info['PhoneNumber'], updated_customer.phone_number)
            self.assertEqual(updated_info['Address'], updated_customer.address)

        except InvalidInputException as e:
            self.fail(f"Unexpected exception raised: {e}")
```

3. Test adding a new vehicle.



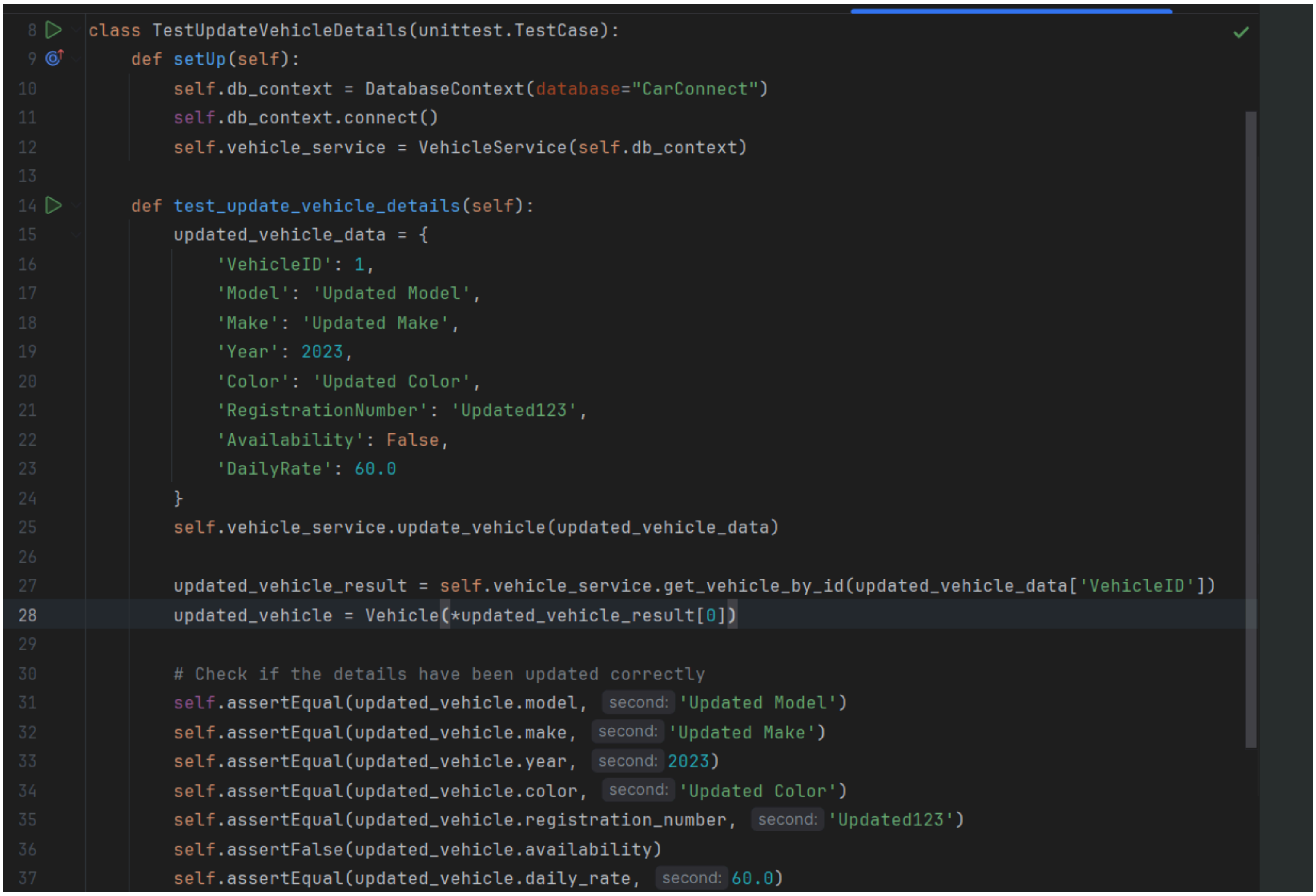
```python
class TestAddNewVehicle(unittest.TestCase):
    def setUp(self):
        self.db_context = DatabaseContext(database="CarConnect")
        self.db_context.connect()
        self.vehicle_service = VehicleService(self.db_context)

    def test_add_new_vehicle(self):
        new_vehicle_data = {
            'Model': 'r15',
            'Make': 'Yamaha',
            'Year': 2023,
            'Color': 'Black',
            'RegistrationNumber': 'GJ05123',
            'Availability': 'y',  # y for True and n for False
            'DailyRate': 500.00,
        }

        try:
            self.vehicle_service.add_vehicle(new_vehicle_data)
            curr_cursor = self.db_context.get_current_cursor()
            new_vehicle_id = curr_cursor.lastrowid

            added_vehicle_result = self.vehicle_service.get_vehicle_by_id(new_vehicle_id)
            added_vehicle = Vehicle(*added_vehicle_result[0])

            self.assertIsNotNone(added_vehicle)
            self.assertEqual(new_vehicle_data['Model'], added_vehicle.model)
            self.assertEqual(new_vehicle_data['Make'], added_vehicle.make)
            self.assertEqual(new_vehicle_data['Year'], added_vehicle.year)
            self.assertEqual(new_vehicle_data['Color'], added_vehicle.color)
```



```python
            'RegistrationNumber': 'GJ05123',
            'Availability': 'y',  # y for True and n for False
            'DailyRate': 500.00,
        }

        try:
            self.vehicle_service.add_vehicle(new_vehicle_data)
            curr_cursor = self.db_context.get_current_cursor()
            new_vehicle_id = curr_cursor.lastrowid

            added_vehicle_result = self.vehicle_service.get_vehicle_by_id(new_vehicle_id)
            added_vehicle = Vehicle(*added_vehicle_result[0])

            self.assertIsNotNone(added_vehicle)
            self.assertEqual(new_vehicle_data['Model'], added_vehicle.model)
            self.assertEqual(new_vehicle_data['Make'], added_vehicle.make)
            self.assertEqual(new_vehicle_data['Year'], added_vehicle.year)
            self.assertEqual(new_vehicle_data['Color'], added_vehicle.color)
```

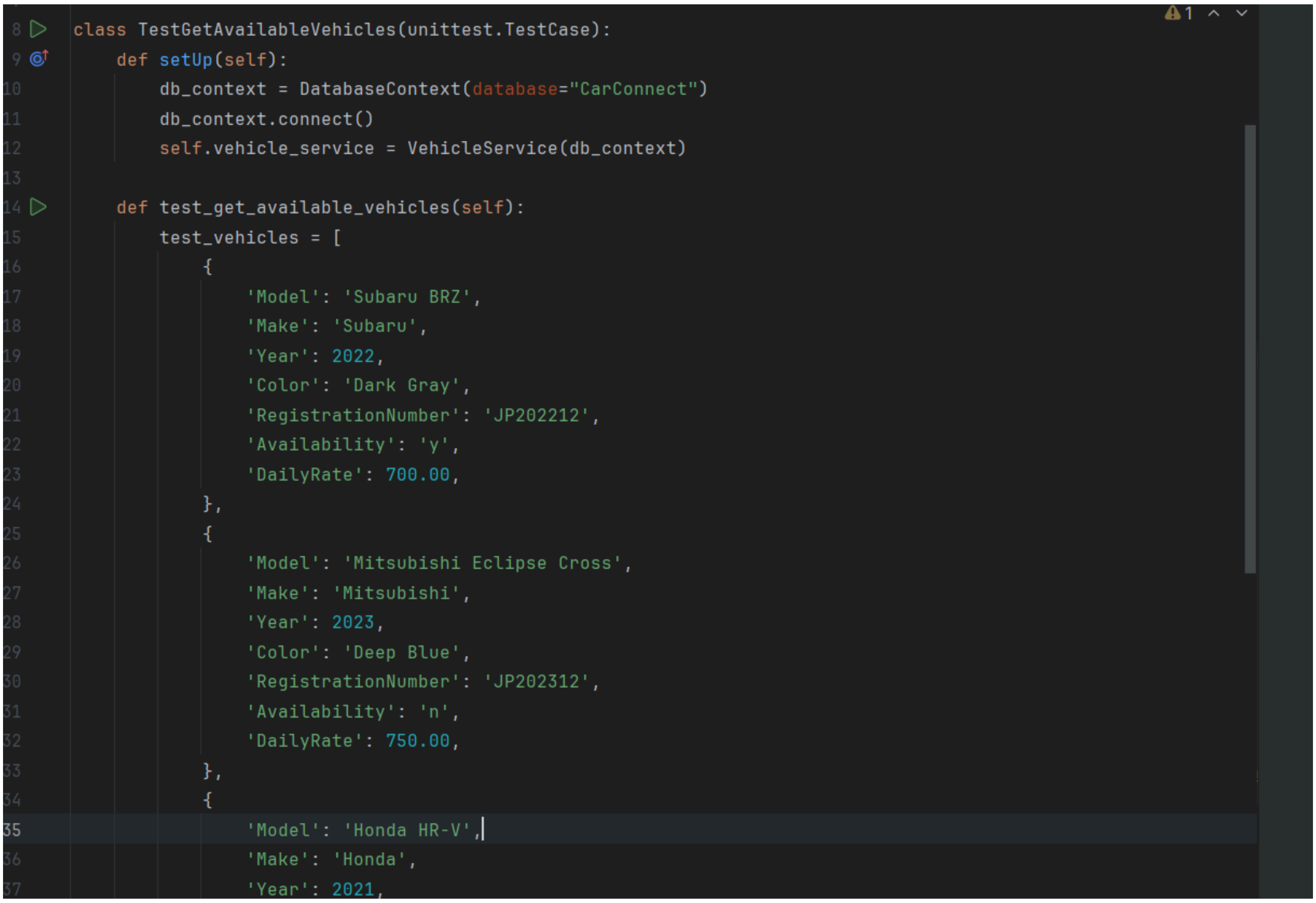
```python
            self.assertEqual(new_vehicle_data['RegistrationNumber'], added_vehicle.registration_number)
            self.assertEqual(new_vehicle_data['Availability'], 'y' if added_vehicle.availability == 1 else 'n'
            self.assertEqual(new_vehicle_data['DailyRate'], added_vehicle.daily_rate)

        except Exception as e:
            self.fail(f"Exception raised: {e}")
```

4. Test updating vehicle details.

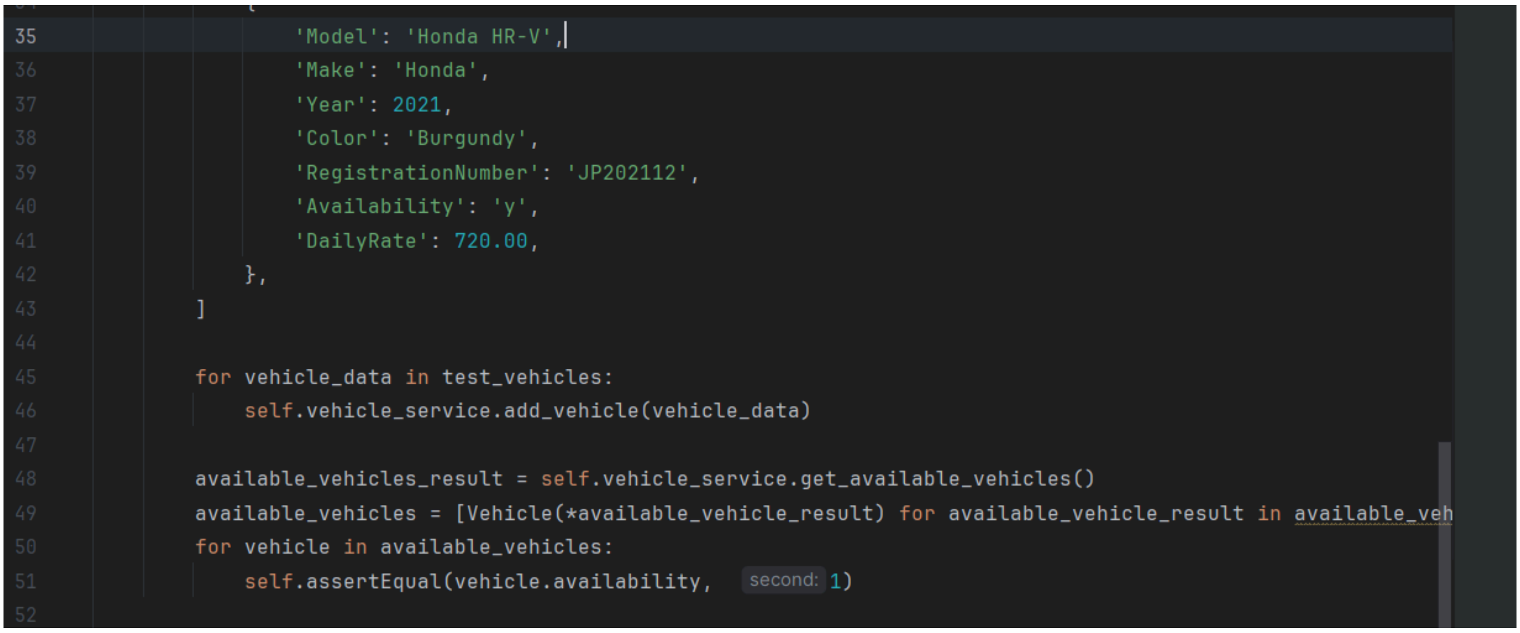


```
class TestUpdateVehicleDetails(unittest.TestCase):
    def setUp(self):
        self.db_context = DatabaseContext(database="CarConnect")
        self.db_context.connect()
        self.vehicle_service = VehicleService(self.db_context)

    def test_update_vehicle_details(self):
        updated_vehicle_data = {
            'VehicleID': 1,
            'Model': 'Updated Model',
            'Make': 'Updated Make',
            'Year': 2023,
            'Color': 'Updated Color',
            'RegistrationNumber': 'Updated123',
            'Availability': False,
            'DailyRate': 60.0
        }
        self.vehicle_service.update_vehicle(updated_vehicle_data)

        updated_vehicle_result = self.vehicle_service.get_vehicle_by_id(updated_vehicle_data['VehicleID'])
        updated_vehicle = Vehicle(*updated_vehicle_result[0])

        # Check if the details have been updated correctly
        self.assertEqual(updated_vehicle.model, second: 'Updated Model')
        self.assertEqual(updated_vehicle.make, second: 'Updated Make')
        self.assertEqual(updated_vehicle.year, second: 2023)
        self.assertEqual(updated_vehicle.color, second: 'Updated Color')
        self.assertEqual(updated_vehicle.registration_number, second: 'Updated123')
        self.assertFalse(updated_vehicle.availability)
        self.assertEqual(updated_vehicle.daily_rate, second: 60.0)
```

5. Test getting a list of available vehicles.

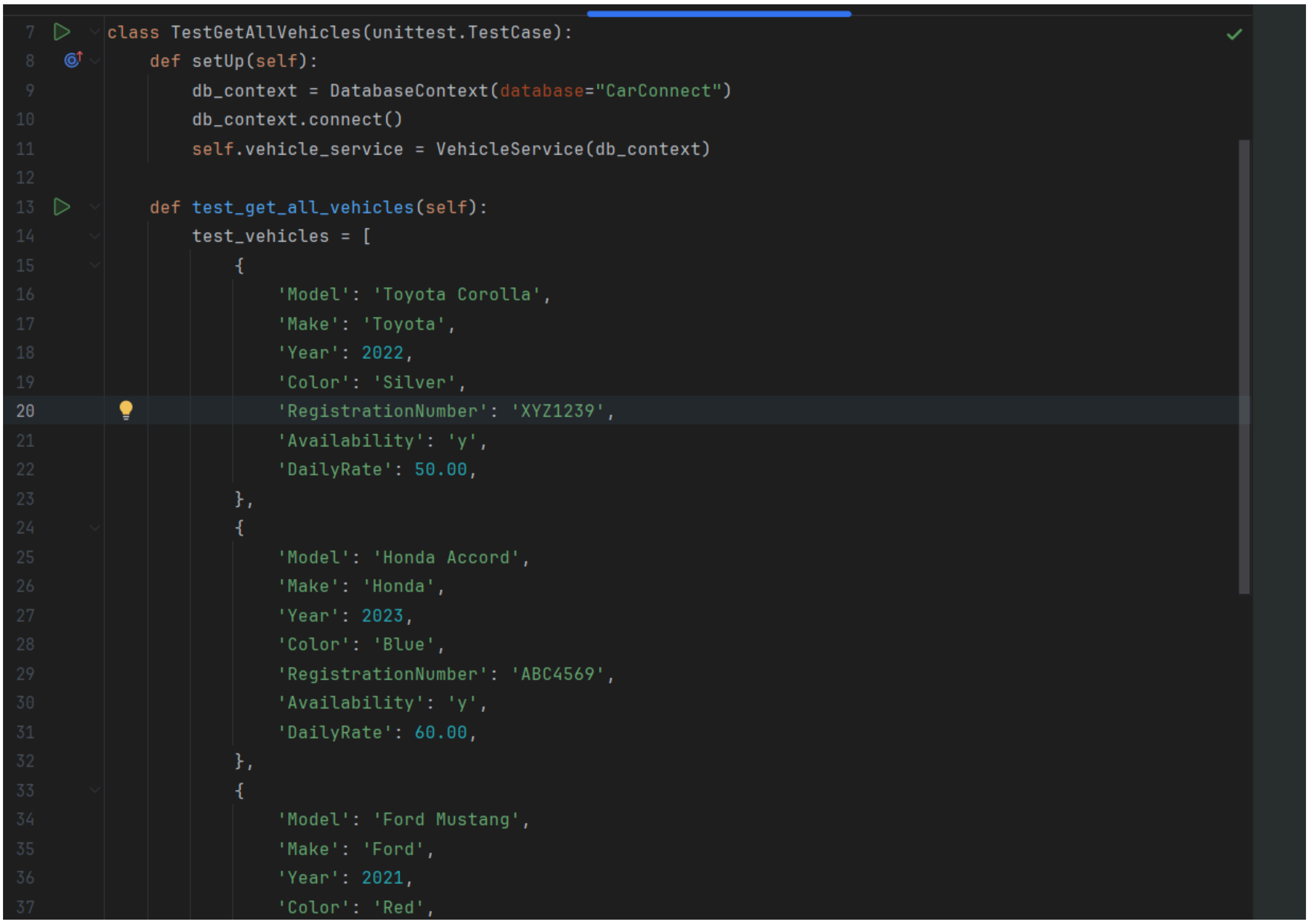
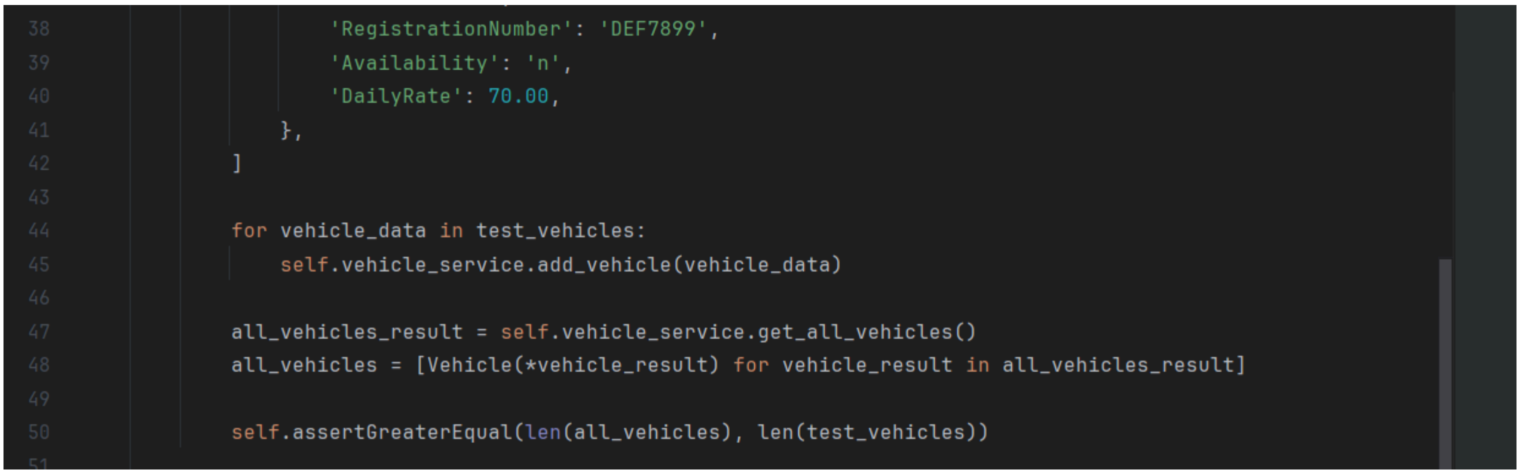


```python
class TestGetAvailableVehicles(unittest.TestCase):
    def setUp(self):
        db_context = DatabaseContext(database="CarConnect")
        db_context.connect()
        self.vehicle_service = VehicleService(db_context)

    def test_get_available_vehicles(self):
        test_vehicles = [
            {
                'Model': 'Subaru BRZ',
                'Make': 'Subaru',
                'Year': 2022,
                'Color': 'Dark Gray',
                'RegistrationNumber': 'JP202212',
                'Availability': 'y',
                'DailyRate': 700.00,
            },
            {
                'Model': 'Mitsubishi Eclipse Cross',
                'Make': 'Mitsubishi',
                'Year': 2023,
                'Color': 'Deep Blue',
                'RegistrationNumber': 'JP202312',
                'Availability': 'n',
                'DailyRate': 750.00,
            },
            {
                'Model': 'Honda HR-V',
                'Make': 'Honda',
                'Year': 2021,
```



```python
                'Model': 'Honda HR-V',
                'Make': 'Honda',
                'Year': 2021,
                'Color': 'Burgundy',
                'RegistrationNumber': 'JP202112',
                'Availability': 'y',
                'DailyRate': 720.00,
            },
        ]

        for vehicle_data in test_vehicles:
            self.vehicle_service.add_vehicle(vehicle_data)

        available_vehicles_result = self.vehicle_service.get_available_vehicles()
        available_vehicles = [Vehicle(*available_vehicle_result) for available_vehicle_result in available_veh
        for vehicle in available_vehicles:
            self.assertEqual(vehicle.availability,  second: 1)
```

6. Test getting a list of all vehicles.

```python
class TestGetAllVehicles(unittest.TestCase):
    def setUp(self):
        db_context = DatabaseContext(database="CarConnect")
        db_context.connect()
        self.vehicle_service = VehicleService(db_context)

    def test_get_all_vehicles(self):
        test_vehicles = [
            {
                'Model': 'Toyota Corolla',
                'Make': 'Toyota',
                'Year': 2022,
                'Color': 'Silver',
                'RegistrationNumber': 'XYZ1239',
                'Availability': 'y',
                'DailyRate': 50.00,
            },
            {
                'Model': 'Honda Accord',
                'Make': 'Honda',
                'Year': 2023,
                'Color': 'Blue',
                'RegistrationNumber': 'ABC4569',
                'Availability': 'y',
                'DailyRate': 60.00,
            },
            {
                'Model': 'Ford Mustang',
                'Make': 'Ford',
                'Year': 2021,
                'Color': 'Red',
                'RegistrationNumber': 'DEF7899',
                'Availability': 'n',
                'DailyRate': 70.00,
            },
        ]

        for vehicle_data in test_vehicles:
            self.vehicle_service.add_vehicle(vehicle_data)

        all_vehicles_result = self.vehicle_service.get_all_vehicles()
        all_vehicles = [Vehicle(*vehicle_result) for vehicle_result in all_vehicles_result]

        self.assertGreaterEqual(len(all_vehicles), len(test_vehicles))
```

Conclusion
Overall it is a full-fledged backend and database connection implementation. I recommend you check the project file by file to see all the features and things implemented.

Thank You!

****