

# **Web Service Project REST Implementation**

**Sze Phing Ho**

# Table of Contents

Introduction .....	1
Business impact .....	1
Business Utility .....	2
Business Processes .....	3
Web Services .....	4
REST (WebCarSalesRESTProject) .....	4
SOAP (WebCarSalesServiceProject) .....	5
Data Structure .....	6
Data Inputs .....	6
Data Outputs .....	6
HTML forms .....	7
Client Application .....	8
Screen shots .....	9

# Introduction

The Car Sales Management project is a comprehensive web service that aims to facilitate the car sales process and enhance the overall user experience. It provides both RESTful and SOAP endpoints to cater to various user requirements. The project's business utility revolves around managing the car catalog, allowing users to view available cars, search for specific cars, add new cars, delete existing cars, and update car information. Additionally, it offers functionalities related to retrieving and searching car sales orders.

## Business impact

1. **Improved customer experience:** The project provides a user-friendly web interface for customers to browse and search for cars. By offering multiple formats (HTML, plain text, JSON) for displaying the car catalog, customers can choose their preferred format, enhancing their overall experience.
2. **Increased sales and revenue:** By providing an online platform to showcase the car catalog and enabling customers to search for cars based on specific criteria (car ID, car make), the project helps attract potential buyers and facilitates the sales process. This can lead to increased sales and revenue for the car dealership.
3. **Time and cost efficiency:** The project automates the process of displaying the car catalog, eliminating the need for manual catalog creation and updates. It provides a centralized system for managing car information, making it easier and more efficient for the dealership to showcase and promote their inventory. This can save time and reduce operational costs.
4. **Scalability and flexibility:** The project utilizes RESTful web services, which are highly scalable and can accommodate a growing number of users and increasing data volumes. It also allows for easy integration with other systems or applications, providing flexibility for future enhancements or integrations.
5. **Competitive advantage:** By leveraging modern web technologies and offering a convenient and interactive car catalog, the project can give the car dealership a competitive edge in the market. It demonstrates the dealership's commitment to digital innovation and customer satisfaction, attracting tech-savvy customers and differentiating the business from competitors.

# Business Utility

The business utility of the project is to provide a RESTful and SOAP web service for car sales. Up to now, the web service offers the following functionality:

**Displaying the car catalog:** The `display_car_catalog` method is a business utility that provides functionality to display the car catalog to potential customers. This utility allows users to view the available cars, their details, and potentially make a purchase decision.

**Searching for cars in the catalog:** The `search_cars` method is a business utility that enables users to search for specific cars based on their preferences or criteria. This utility assists customers in finding the cars that match their requirements quickly and efficiently.

**Adding a new car to the catalog:** The `add_car` method is a business utility that allows car dealers or administrators to add new cars to the catalog. This utility supports the process of expanding the inventory and making new vehicles available for potential customers.

**Deleting a car from the catalog:** The `delete_car` method is a business utility that facilitates the removal of a car from the catalog. This utility is useful when a car is no longer available for sale, such as when it's sold or withdrawn from the inventory.

**Updating car information in the catalog:** The `update_car_info` method is a business utility that enables the modification of car details in the catalog. This utility can be used to update information such as price, mileage, availability, or any other relevant attributes of a car.

**Order information retrieval:** Clients can retrieve information about all available car sales orders, enabling them to get an overview of the orders in the system.

**Order search:** Clients can search for specific car sales orders based on their order IDs, making it convenient to access detailed information about individual orders.

**Add an order:** Clients can add a car order by providing the car ID, and the specific car will be removed from the car catalog.

# Business Processes

**Displaying the car catalog:** This process involves retrieving the car catalog data and presenting it to potential customers in a user-friendly format. Customers can browse through the available cars, view their details, and make informed decisions about potential purchases. The utility method `display_car_catalog` facilitates this process by rendering the catalog data and presenting it to the customers.

**Searching for cars:** This process enables customers to search for specific cars based on their preferences. Customers can input criteria such as car make, model, year, price range, or any other relevant attributes. The `search_cars` utility method processes the search query and returns a list of cars that match the specified criteria. This process enhances the customer's ability to find suitable cars efficiently.

**Adding a new car to the catalog:** This process involves car dealers or administrators adding new cars to the catalog. When a new car becomes available for sale, the car details, such as make, model, year, price, mileage, and other relevant information, need to be added to the catalog. The `add_car` utility method supports this process by allowing the necessary information to be input and added to the catalog.

**Deleting a car from the catalog:** This process deals with removing a car from the catalog. When a car is sold, no longer available for sale, or needs to be removed from the inventory for any other reason, the car entry must be deleted from the catalog. The `delete_car` utility method supports this process by removing the specified car from the catalog.

**Updating car information in the catalog:** This process involves modifying the details of a car in the catalog. Car information, such as price updates, mileage changes, or other attribute modifications, may occur over time. The `update_car_info` utility method allows authorized personnel to update the relevant information in the catalog, ensuring that it reflects the most accurate and up-to-date data.

**Order Display Process:** The service provides the `displayOrderInfo()` method, which retrieves and returns information about all available car sales orders. Users can call this method to obtain a list of order details for display or further processing.

**Order Search Process:** The service offers the `searchOrderInfo(int orderId)` method, allowing clients to search for a specific car sales order using its unique order ID. Users can pass an order ID as input to the method, and the service will return the corresponding order details if a match is found.

**Adding a car order:** The service offers the `addOrder(int orderId, String carId, String carModel, double carPrice)` method, allowing clients to add a record of car order using car's unique car ID. Users can pass the argument to create a new order, and the service will return the corresponding order details and delete the specific car from the car catalog based on the car ID.

# Web Services

## REST (WebCarSalesRESTProject)

**displayHTMLCarCatalog():** This method displays the car catalog in HTML format.

**displayTextCarCatalog():** This method displays the car catalog in plain text format.

**displayJSONCarCatalog():** This method displays the car catalog in JSON format.

**searchJSONCarId(String carId):** This method searches for a car by car ID using @PathParam and returns the matching cars in JSON format.

**searchQueryHTMLCarId(String carId):** This method searches for a car by car ID using @QueryParam and returns the matching cars in HTML format.

**searchJSONCarMake(String carMake):** This method searches for cars by car make using @PathParam and returns the matching cars in JSON format.

**searchQueryHTMLCarMake(String carMake):** This method searches for cars by car make using @QueryParam and returns the matching cars in HTML format.

**searchJSONCarModel(String carModel):** This method searches for cars by car model using @PathParam and returns the matching cars in JSON format.

**searchQueryHTMLCarModel(String carModel):** This method searches for cars by car model using @QueryParam and returns the matching cars in HTML format.

**searchJSONCarPrice(double price1, double price2):** This method searches for cars within a price range using @PathParam and returns the matching cars in JSON format.

**searchQueryJSONCarPrice(double price1, double price2):** This method searches for cars within a price range using @QueryParam and returns the matching cars in HTML format.

**addQueryJSONCar(String carId, String carMake, String carModel, int carYear, String carDesc, double carPrice, String carImageURL):** This method adds a car to the catalog using @QueryParam and returns the updated car catalog in JSON format.

**addJSONCar(String carId, String carMake, String carModel, int carYear, String carDesc, double carPrice, String carImageURL):** This method adds a car to the catalog using @FormParam and returns the updated car catalog in JSON format.

**addCarHTML(String carId, String carMake, String carModel, int carYear, String carDesc, double carPrice, String carImageURL):** This method adds a car to the catalog using @FormParam and returns the updated car catalog in HTML format.

**removeJSONClientCar(String carId):** This method removes a car from the catalog using @QueryParam and returns the updated car catalog in JSON format.

**removeJSONCar(String carId):** This method removes a car from the catalog using @QueryParam and returns the updated car catalog in JSON format.

**updateJSONCar(String carId, double carPrice):** This method updates the price of a car using @QueryParam and returns the updated car catalog in JSON format.

## **SOAP (WebCarSalesServiceProject)**

The WebCarSales web service is implemented using the Jakarta API for web services (jakarta.jws.WebService) and uses SOAP binding style RPC (jakarta.jws.soap.SOAPBinding.Style.RPC). The service exposes two main methods: displayOrderInfo() and searchOrderInfo(int orderId).

**displayCarCatalog():** This method retrieves information of all available car and returns a string representation of the cars.

**displayOrderInfo():** This method retrieves information about all available car sales orders and returns a string representation of the orders and return a string containing information about all available car sales orders.

**searchOrderInfo(int orderId):** This method allows clients to search for a specific car sales order based on its order ID. If a matching order is found with the provided orderId, the method returns the Order object representing that order. If no matching order is found, it returns null.

**addOrder(int orderId, String carId, String carModel, double carPrice):** This method allows clients to add a new record of order placed on a specific car based on the car ID. The record of new order will be created in order list and the specific car will be removed from the car catalog.

# Data Structure

The data structure used to store the car information is a `HashMap<String, Car>`. The key of the map is the car ID (String), and the value is an instance of the Car class. The data structure used to store the order information is a `ArrayList<Order>`.

## Data Inputs

- `searchJSONCarId(carId)`: Path parameter 'carId' representing the car ID to search for.
- `searchJSONCarMake(carMake)`: Path parameter 'carMake' representing the car make to search for.
- `searchJSONCarModel(carModel)`: Path parameter 'carModel' representing the car model to search for.
- `searchJSONCarPrice(price1, price2)`: Path parameter 'price1' representing the lower bound of the price range and path parameter 'price2' representing the higher bound of the price range.
- `addJSONCar(carId, carMake, carModel, carYear, carDesc, carPrice)`: Form parameters 'carId', 'carMake', 'carModel', 'carYear', 'carDesc', 'carPrice', representing the attributes of a Car object to be added to the carHashMap.
- `removeJSONCar(carId)`: Query parameter 'carId' representing the ID of car to be deleted.
- `updateJSONCar(carId, carPrice)`: Query parameters 'carId' representing the ID of the car to be updated, and 'carPrice' representing the new price for the car to be updated.
- `searchOrderInfo(int orderId)`: The unique identifier of the car sales order that the user wants to search for.
- `addOrder(int orderId, String carId, String carModel, double carPrice)`: The unique identifier of the car sales order, car ID, car model and car price of the order.

## Data Outputs

- `displayHTMLCarCatalog()`: Returns an HTML string representing the car catalog in table format.
- `displayTextCarCatalog()`: Returns a plain text string containing the car catalog information.
- `displayJSONCarCatalog()`: Returns a list of Car objects representing the car catalog in JSON format.
- `searchJSONCarId(carId)`: Returns a list of Car objects representing the matching car(s) for the provided car ID in JSON format.
- `searchJSONCarMake(carMake)`: Returns a list of Car objects representing the matching car(s) for the provided car make in JSON format.
- `searchJSONCarModel(carModel)`: Returns a list of Car objects representing the matching car(s) for the provided car model in JSON format.
- `searchJSONCarPrice(price1, price2)`: Returns a list of Car objects representing the matching car(s) within the price range in JSON format.
- `addJSONCar(carId, carMake, carModel, carYear, carDesc, carPrice)`: Returns the HashMap of the Car objects after adding the new record of car.
- `removeJSONCar(carId)`: Returns the HashMap of Car objects after deleting a car from it.
- `updateJSONCar(carId, carPrice)`: Returns the HashMap of Car objects after updating the price of a Car object.
- `displayOrderInfo()`: Returns a string containing information about all available car sales orders. Each order's details are formatted with the order ID and car ID.
- `searchOrderInfo(int orderId)`: If a matching order is found with the provided orderId, the method returns the Order object representing that order. If no matching order is found, it returns null.
- `addOrder(int orderId, String carId, String carModel, double carPrice)`: Returns an object of the new order.



# HTML forms

1. addCarForm.html: Allows users to add a new car by providing car information such as car ID, make, model, year, description (new/used), and price. The form submits a POST request to the `'../WebCarSalesRESTProject/rest/WebCarSales/addCar'` endpoint. The form fields have the following attributes:
  - Car ID: `<input type="text" name="car_id" size="20" />`
  - Make: `<input type="text" name="car_make" size="20" />`
  - Model: `<input type="text" name="car_model" size="20" />`
  - Year: `<input type="text" name="car_year" size="20" />`
  - Description (New/Used): `<input type="text" name="car_desc" size="20" />`
  - Price (\$): `<input type="text" name="car_price" size="20" />`
  - Image URL: `<input type="text" name="car_image_url" size="20" />`
2. searchCarForm.html: Allows users to search for a car by its ID, Make, Model, and Price Range. Users enter the car ID, Make, Model, Minimum Price and Maximum Price in the input fields, and the forms submit GET request to the endpoints of `'../WebCarSalesRESTProject/rest/WebCarSales/searchCarId'`, `'../WebCarSalesRESTProject/rest/WebCarSales/searchCarMake'`, `'../WebCarSalesRESTProject/rest/WebCarSales/searchCarModel'`, `'../WebCarSalesRESTProject/rest/WebCarSales/searchCarPrice'`.
3. searchCarByIdForm.html: Allows users to search for a car by its ID. Users enter the car ID in the input field, and the form submits a GET request to the `'../WebCarSalesRESTProject/rest/WebCarSales/searchCarId'` endpoint. The form field has the following attribute:
  - Car ID: `<input type="text" name="carId" size="20" />`
4. searchCarByCarMakeForm.html: Allows users to search for cars by their make. Users enter the car make in the input field, and the form submits a GET request to the `'../WebCarSalesRESTProject/rest/WebCarSales/searchCarMake'` endpoint. The form field has the following attribute:
  - Car Make: `<input type="text" name="carMake" size="20" />`
5. searchCarByModelForm.html: Allows users to search for cars by their model. Users enter the car model in the input field, and the form submits a GET request to the `'../WebCarSalesRESTProject/rest/WebCarSales/searchCarModel'` endpoint. The form field has the following attribute:
  - Car Model: `<input type="text" name="carModel" size="20" />`
6. searchCarByPriceForm.html: Allows users to search for cars within a specific price range. Users enter the lower bound and upper bound of the price range in the input fields, and the form submits a GET request to the `'../WebCarSalesRESTProject/rest/WebCarSales/searchCarPrice'` endpoint. The form fields have the following attributes:
  - Lower bound of price range: `<input type="text" name="price1" size="20" />`
  - Higher bound of price range: `<input type="text" name="price2" size="20" />`

# Client Application

The ClientCarSalesREST application interacts with a RESTful web service for car sales. It provides an interface to search for cars based on different criteria such as ID, make, model, and price range.

The client application begins by establishing a connection to the REST service by specifying the URL of the service. It then creates a client object using the JAX-RS ClientBuilder and configures it with the necessary settings using ClientConfig.

The application allows users to perform the following actions:

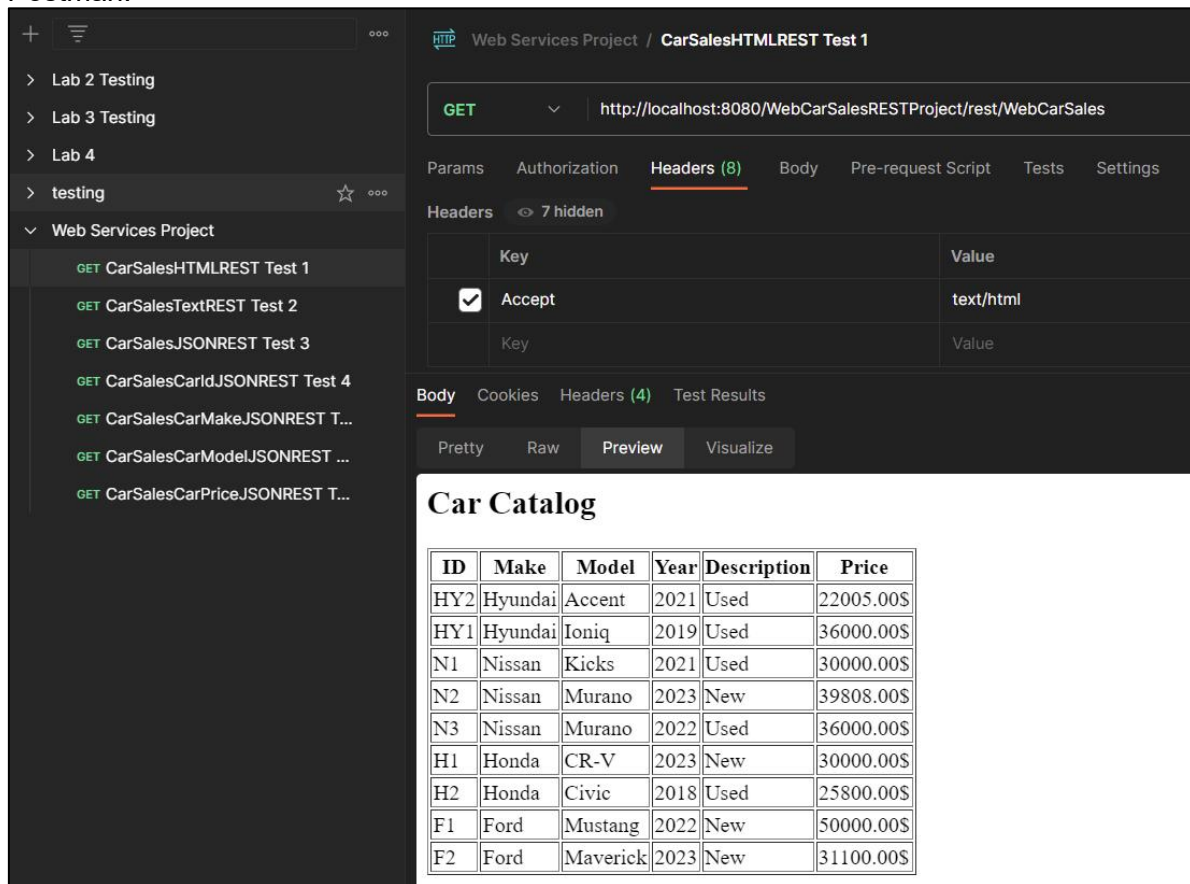
1. Retrieve a list of all cars: The client sends a GET request to the REST service and retrieves a JSON response containing a list of cars. The response is then displayed on the console.
2. Search for a car by ID: The user is prompted to enter a car ID, and the client sends a GET request with the ID as a path parameter. The response, containing information about the car with the specified ID, is displayed on the console.
3. Search for a car by make: The user is prompted to enter a car make, and the client sends a GET request with the make as a path parameter. The response, containing information about the cars with the specified make, is displayed on the console.
4. Search for a car by model: The user is prompted to enter a car model, and the client sends a GET request with the model as a path parameter. The response, containing information about the cars with the specified model, is displayed on the console.
5. Search for cars within a price range: The user is prompted to enter a minimum and maximum price, and the client sends a GET request with the price range as path parameters. The response, containing information about the cars within the specified price range, is displayed on the console.
6. Adding a Car: The code prompts the user to enter car details (ID, make, model, year, description, price, image URL) and sends a POST request to add the car to the REST service. The response is stored in the response variable and displayed to the console.
7. Deleting a Car: The code prompts the user to enter a car ID and sends a DELETE request to remove the car from the REST service. The response is stored in the response variable and displayed to the console.
8. Updating a Car's Price: The code prompts the user to enter a car ID and a new price, and sends a PUT request to update the car's price in the REST service. The response is stored in the response variable and displayed to the console.

The application utilizes the JAX-RS client API to interact with the REST service and retrieves responses in JSON format. It uses the Scanner class to obtain user input for searching criteria.

# Screen shots

## CarSalesHTMLREST Test 1: Display car catalog in HTML

Postman:

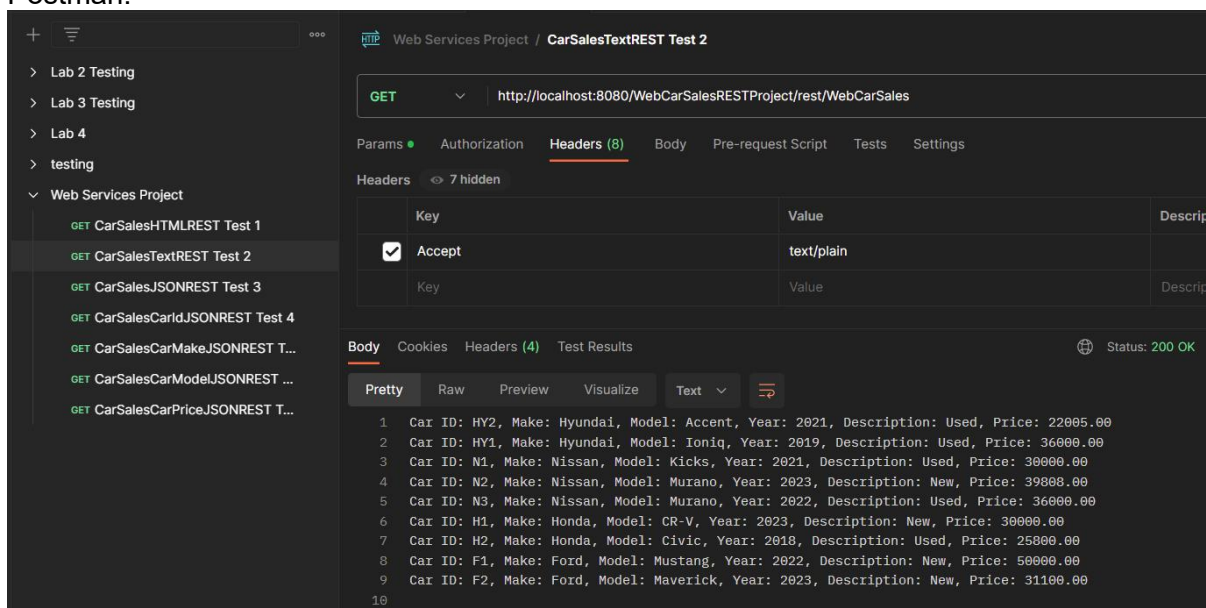


The screenshot shows the Postman interface for a REST client. The left sidebar displays a list of tests under the 'Web Services Project' folder. The main panel shows the details for 'CarSalesHTMLREST Test 1'. The request is a GET to 'http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales'. The 'Headers' tab is active, showing an 'Accept' header set to 'text/html'. The 'Body' tab is also active, displaying a 'Car Catalog' table with 10 rows of car data.

ID	Make	Model	Year	Description	Price
HY2	Hyundai	Accent	2021	Used	22005.00\$
HY1	Hyundai	Ioniq	2019	Used	36000.00\$
N1	Nissan	Kicks	2021	Used	30000.00\$
N2	Nissan	Murano	2023	New	39808.00\$
N3	Nissan	Murano	2022	Used	36000.00\$
H1	Honda	CR-V	2023	New	30000.00\$
H2	Honda	Civic	2018	Used	25800.00\$
F1	Ford	Mustang	2022	New	50000.00\$
F2	Ford	Maverick	2023	New	31100.00\$

## CarSalesTextREST Test 2: Display car catalog in plain text

Postman:



The screenshot shows the Postman interface for a REST client. The left sidebar displays a list of tests under the 'Web Services Project' folder. The main panel shows the details for 'CarSalesTextREST Test 2'. The request is a GET to 'http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales'. The 'Headers' tab is active, showing an 'Accept' header set to 'text/plain'. The 'Body' tab is also active, displaying the car catalog data as a plain text list. The status bar indicates a 200 OK response.

```
1 Car ID: HY2, Make: Hyundai, Model: Accent, Year: 2021, Description: Used, Price: 22005.00
2 Car ID: HY1, Make: Hyundai, Model: Ioniq, Year: 2019, Description: Used, Price: 36000.00
3 Car ID: N1, Make: Nissan, Model: Kicks, Year: 2021, Description: Used, Price: 30000.00
4 Car ID: N2, Make: Nissan, Model: Murano, Year: 2023, Description: New, Price: 39808.00
5 Car ID: N3, Make: Nissan, Model: Murano, Year: 2022, Description: Used, Price: 36000.00
6 Car ID: H1, Make: Honda, Model: CR-V, Year: 2023, Description: New, Price: 30000.00
7 Car ID: H2, Make: Honda, Model: Civic, Year: 2018, Description: Used, Price: 25800.00
8 Car ID: F1, Make: Ford, Model: Mustang, Year: 2022, Description: New, Price: 50000.00
9 Car ID: F2, Make: Ford, Model: Maverick, Year: 2023, Description: New, Price: 31100.00
10
```

## CarSalesJSONREST Test 3: Display car catalog in JSON

Postman:

Web Services Project / CarSalesJSONREST Test 3

GET <http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales>

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers 7 hidden

Key	Value	Description
Accept	application/json	
Key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "carDesc": "Used",
4     "carId": "HY2",
5     "carMake": "Hyundai",
6     "carModel": "Accent",
7     "carPrice": 22005.0,
8     "carYear": 2021
9   },
10  {
11    "carDesc": "Used",
12    "carId": "HY1",
13    "carMake": "Hyundai",
14    "carModel": "Ioniq",
15    "carPrice": 36000.0,
16    "carYear": 2019
17  },
18  {
19    "carDesc": "Used",
```

## CarSalesCardIdJSONREST Test 4: Search a car by Car ID and return the output in JSON

Postman:

Web Services Project / CarSalesCardIdJSONREST Test 4

GET <http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales/searchCardId/HY1>

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers 7 hidden

Key	Value	Description
Accept	application/json	
Key	Value	Description

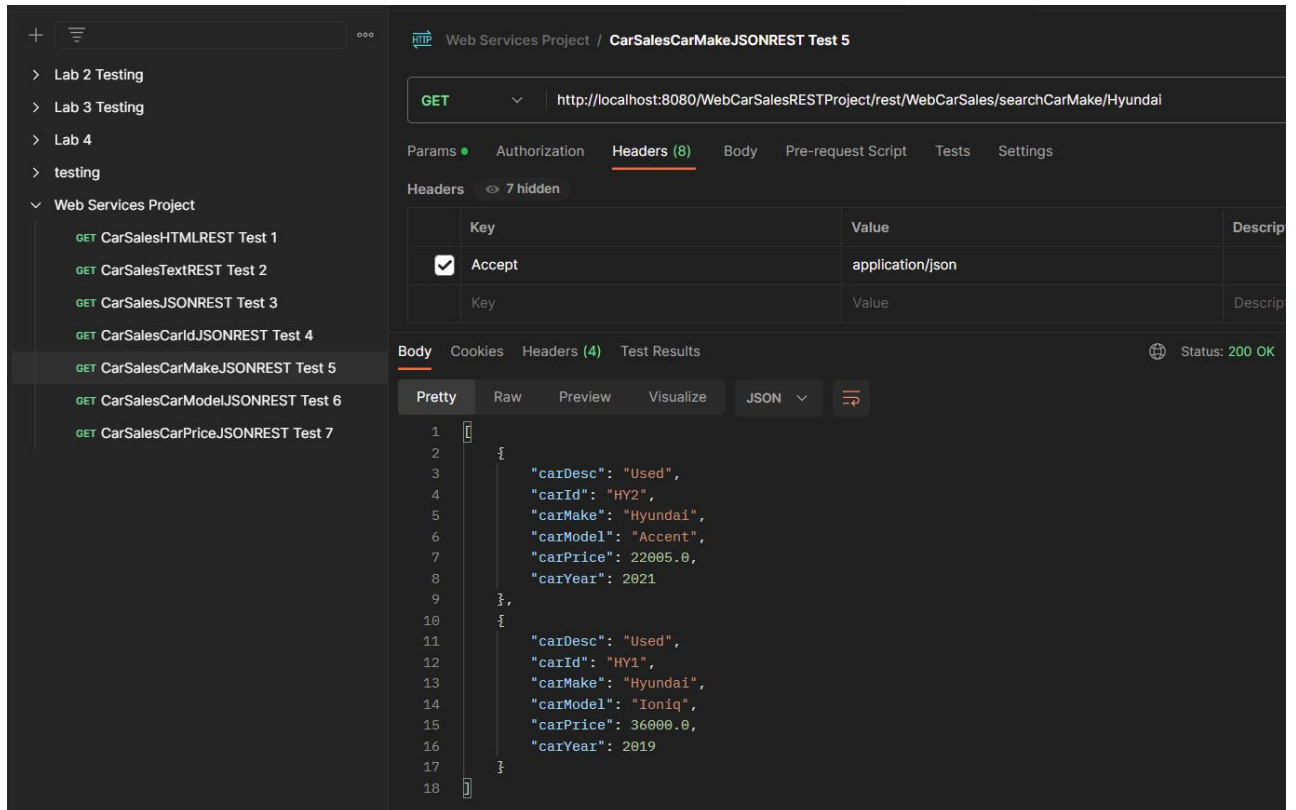
Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "carDesc": "Used",
4     "carId": "HY1",
5     "carMake": "Hyundai",
6     "carModel": "Ioniq",
7     "carPrice": 36000.0,
8     "carYear": 2019
9   }
10 }
```

## CarSalesCarMakeJSONREST Test 5: Search cars by Car Make and return the output in JSON

Postman:



The screenshot shows the Postman interface for a REST client. The left sidebar lists a project named 'Web Services Project' with several tests. The main panel displays the details for 'CarSalesCarMakeJSONREST Test 5'. The URL is 'http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales/searchCarMake/Hyundai'. The method is 'GET'. The 'Headers' tab is active, showing an 'Accept' header with the value 'application/json'. The 'Body' tab is also active, showing a JSON response in 'Pretty' format. The status is '200 OK'.

Web Services Project / CarSalesCarMakeJSONREST Test 5

GET http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales/searchCarMake/Hyundai

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers 7 hidden

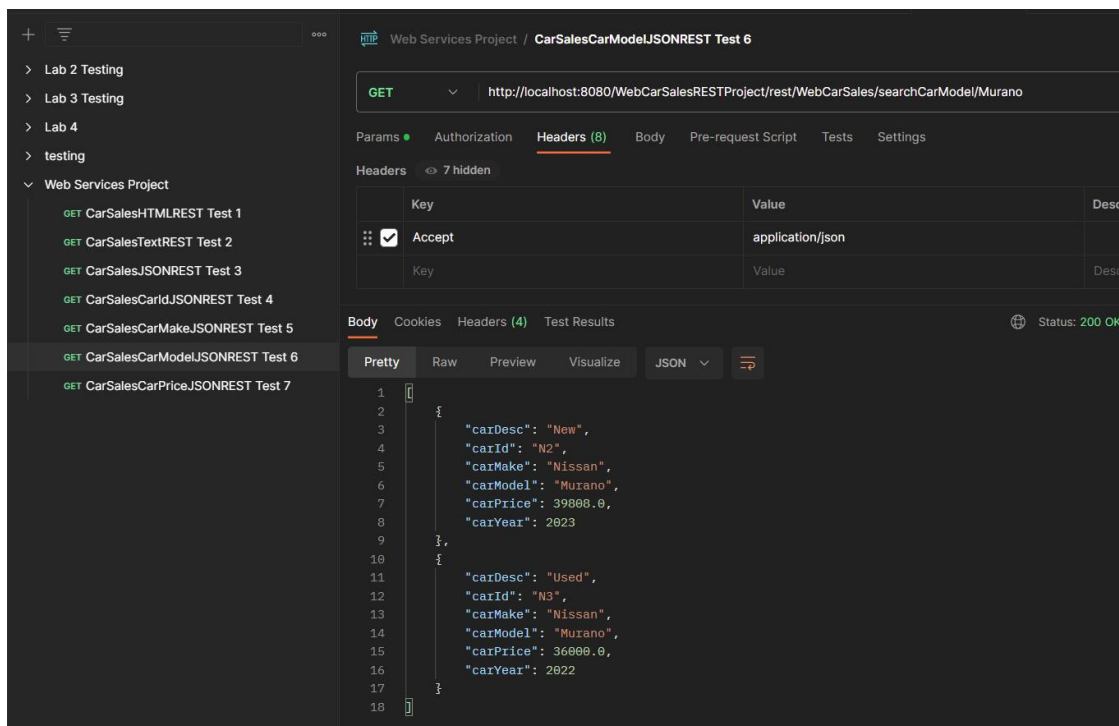
Key	Value	Description
Accept	application/json	
Key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "carDesc": "Used",
4     "carId": "HY2",
5     "carMake": "Hyundai",
6     "carModel": "Accent",
7     "carPrice": 22005.0,
8     "carYear": 2021
9   },
10  {
11    "carDesc": "Used",
12    "carId": "HY1",
13    "carMake": "Hyundai",
14    "carModel": "Ioniq",
15    "carPrice": 36000.0,
16    "carYear": 2019
17  }
18 }
```

## CarSalesCarModelJSONREST Test 6: Search cars by Car Model and return output in JSON



The screenshot shows the Postman interface for a REST client. The left sidebar lists a project named 'Web Services Project' with several tests. The main panel displays the details for 'CarSalesCarModelJSONREST Test 6'. The URL is 'http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales/searchCarModel/Murano'. The method is 'GET'. The 'Headers' tab is active, showing an 'Accept' header with the value 'application/json'. The 'Body' tab is also active, showing a JSON response in 'Pretty' format. The status is '200 OK'.

Web Services Project / CarSalesCarModelJSONREST Test 6

GET http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales/searchCarModel/Murano

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers 7 hidden

Key	Value	Description
Accept	application/json	
Key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "carDesc": "New",
4     "carId": "N2",
5     "carMake": "Nissan",
6     "carModel": "Murano",
7     "carPrice": 39000.0,
8     "carYear": 2023
9   },
10  {
11    "carDesc": "Used",
12    "carId": "N3",
13    "carMake": "Nissan",
14    "carModel": "Murano",
15    "carPrice": 36000.0,
16    "carYear": 2022
17  }
18 }
```

## CarSalesCarPriceJSONREST Test 7: Search cars by a range of price and return the output based on price in ascending order in JSON

Web Services Project / CarSalesCarPriceJSONREST Test 7

GET <http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales/searchCarPrice/30000/50000>

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers 7 hidden

Key	Value
Accept	application/json
Key	Value

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "carDesc": "Used",
4     "carId": "N1",
5     "carMake": "Nissan",
6     "carModel": "Kicks",
7     "carPrice": 30000.0,
8     "carYear": 2021
9   },
10  {
11    "carDesc": "New",
12    "carId": "H1",
13    "carMake": "Honda",
14    "carModel": "CR-V",
15    "carPrice": 30000.0,
16    "carYear": 2023
17  },
18  {
19    "carDesc": "New",
20    "carId": "F2",
21    "carMake": "Ford",
22    "carModel": "Maverick",
23    "carPrice": 31100.0,
24    "carYear": 2023
25  },
26  {
27    "carDesc": "Used",
28    "carId": "HY1",
29    "carMake": "Hyundai",
```

## HTML Forms Data Inputs, Data Outputs

### searchCarForm.html

AutoMart - Car Sales

Search Car By Make

Car Make:

Submit Car Make

Cancel

Search Car By Car ID

Car ID:

Submit Car ID

Cancel

Search Car By Car Model

Car Model:

Submit Car Model

Cancel

Search Car By Price Range


Minimum price: Maximum price:

Submit Price Range


Cancel




Price range: \$30000.00 - \$50000.00




Make: Nissan  
Model: Kicks  
Year: 2021  
Description: Used  
Price: \$30000.00




Make: Honda  
Model: CR-V  
Year: 2023  
Description: New  
Price: \$30000.00




Make: Ford  
Model: Maverick  
Year: 2023  
Description: New  
Price: \$31100.00




Make: Hyundai  
Model: Ioniq  
Year: 2019  
Description: Used  
Price: \$36000.00



Make: Nissan  
Model: Murano  
Year: 2022  
Description: Used  
Price: \$36000.00



Make: Nissan  
Model: Murano  
Year: 2023  
Description: New  
Price: \$39808.00



Make: Honda  
Model: Civic  
Year: 2019  
Description: Used  
Price: \$45000.00

## addCarForm.html

### AutoMart - Car Sales

#### Car Input Information

Car ID:

Make:

Model:










Year:

Description (New/Used):

Price (\$):

Image URL:

## Car Catalog

Car ID	Make	Model	Year	Description	Price (\$)	Image
F1	Ford	Mustang	2022	New	50000.00	
F2	Ford	Maverick	2023	New	31100.00	
H1	Honda	CR-V	2023	New	30000.00	
H2	Honda	Civic	2018	Used	25800.00	
H3	Honda	Civic	2019	Used	45000.00	
HY1	Hyundai	Ioniq	2019	Used	36000.00	
HY2	Hyundai	Accent	2020	Used	21995.00	
HY3	Hyundai	Accent	2021	Used	22005.00	
N1	Nissan	Kicks	2021	Used	30000.00	

## removeCar

The screenshot shows the Postman interface for a DELETE request named 'CarSalesDeleteCarPJSONREST Test 9'. The URL is `http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales/removeCar?car_id=F1`. The 'Query Params' section shows a single parameter: `car_id` with value `F1`. The 'Body' tab is selected, showing a JSON body in 'Pretty' format:

```
35 {
36   "carDesc": "Used",
37   "carId": "N3",
38   "carMake": "Nissan",
39   "carModel": "Murano",
40   "carPrice": 36000.0,
41   "carYear": 2022
42 },
43 "H1": {
44   "carDesc": "New",
```

## updateCar

The screenshot shows the Postman interface for a PUT request named 'CarSalesUpdateCarJSONREST Test 10'. The URL is `http://localhost:8080/WebCarSalesRESTProject/rest/WebCarSales/updateCar?car_id=F1&car_price=45000`. The 'Query Params' section shows two parameters: `car_id` with value `F1` and `car_price` with value `45000`. The 'Body' tab is selected, showing a JSON body in 'Pretty' format:

```
46 {
47   "carModel": "CR-V",
48   "carPrice": 30000.0,
49   "carYear": 2023
50 },
51 "H2": {
52   "carDesc": "Used",
53   "carId": "H2",
54   "carMake": "Honda",
55   "carModel": "Civic",
56   "carPrice": 25800.0,
57   "carYear": 2018
58 },
59 "F1": {
60   "carDesc": "New",
61   "carId": "F1",
62   "carMake": "Ford",
63   "carModel": "Mustang",
64   "carPrice": 45000.0,
65   "carYear": 2022
66 }
```



## displayCarCatalog

Web Services Project / CarSalesDisplayCarsSOAP Test 1

POST http://localhost:8080/WebCarSalesServiceProject/ws/webCarSales

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL XML

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
3   xmlns:tns="http://webCarSalesService/"
4   <S:Body>
5     <tns:displayCarCatalog>
6     </tns:displayCarCatalog>
7   </S:Body>
8 </S:Envelope>
```

Body Cookies Headers (4) Test Results Status: 200 OK Time: 13 ms Size:

Pretty Raw Preview Visualize XML

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
3   <S:Body>
4     <ns2:displayCarCatalogResponse xmlns:ns2="http://webCarSalesService/"
5       <return>Car ID: HY2, Make: Hyundai, Model: Accent, Year: 2020, Description: Used, Price: 21995.00
6 Car ID: HY1, Make: Hyundai, Model: Ioniq, Year: 2019, Description: Used, Price: 36000.00
7 Car ID: N1, Make: Nissan, Model: Kicks, Year: 2021, Description: Used, Price: 30000.00
8 Car ID: HY3, Make: Hyundai, Model: Accent, Year: 2021, Description: Used, Price: 22005.00
9 Car ID: N2, Make: Nissan, Model: Murano, Year: 2023, Description: New, Price: 39800.00
10 Car ID: N3, Make: Nissan, Model: Murano, Year: 2022, Description: Used, Price: 36000.00
11 Car ID: H1, Make: Honda, Model: CR-V, Year: 2023, Description: New, Price: 30000.00
12 Car ID: H2, Make: Honda, Model: Civic, Year: 2018, Description: Used, Price: 25800.00
13 Car ID: F2, Make: Ford, Model: Maverick, Year: 2023, Description: New, Price: 31100.00
14 </return>
15 </ns2:displayCarCatalogResponse>
16 </S:Body>
```

## displayOrderInfo

Web Services Project / CarSalesDisplayOrdersSOAP TEST 1

POST http://localhost:8080/WebCarSalesServiceProject/ws/webCarSales

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL XML

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
3   xmlns:tns="http://webCarSalesService/"
4   <S:Body>
5     <tns:displayOrderInfo>
6     </tns:displayOrderInfo>
7   </S:Body>
8 </S:Envelope>
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize XML

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
3   <S:Body>
4     <ns2:displayOrderInfoResponse xmlns:ns2="http://webCarSalesService/"
5       <return>
6 Order ID: 1, Car ID: F1
7 Order ID: 2, Car ID: H1
8 Order ID: 3, Car ID: N1</return>
9 </ns2:displayOrderInfoResponse>
10 </S:Body>
11 </S:Envelope>
```

## searchOrderInfo(int orderId)

Web Services Project / CarSalesSearchOrdersSOAP Test 3

POST http://localhost:8080/WebCarSalesServiceProject/ws/webCarSales

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL XML

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
3   xmlns:tns="http://webCarSalesService/">
4   <S:Body>
5     <tns:searchOrderInfo>
6       <arg0>1</arg0>
7     </tns:searchOrderInfo>
8   </S:Body>
9 </S:Envelope>
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize XML

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
3   <S:Body>
4     <ns2:searchOrderInfoResponse xmlns:ns2="http://webCarSalesService/">
5       <return>
6         <carId>F1</carId>
7         <carModel>Mustang</carModel>
8         <carPrice>50000.0</carPrice>
9         <orderId>1</orderId>
10      </return>
11    </ns2:searchOrderInfoResponse>
12  </S:Body>
13 </S:Envelope>
```

## addOrder (int orderId, String carId, String carModel, double carPrice)

Web Services Project / CarSalesAddOrdersSOAP Test 4

POST http://localhost:8080/WebCarSalesServiceProject/ws/webCarSales

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL XML

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
3   xmlns:tns="http://webCarSalesService/">
4   <S:Body>
5     <tns:addOrder>
6       <arg0>1</arg0>
7       <arg1>F1</arg1>
8       <arg2>Mustang</arg2>
9       <arg3>50000.0</arg3>
10    </tns:addOrder>
11  </S:Body>
12 </S:Envelope>
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize XML

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
3   <S:Body>
4     <ns2:addOrderResponse xmlns:ns2="http://webCarSalesService/">
5       <return>
6         <carId>F1</carId>
7         <carModel>Mustang</carModel>
8         <carPrice>50000.0</carPrice>
9         <orderId>1</orderId>
10      </return>
11    </ns2:addOrderResponse>
12  </S:Body>
13 </S:Envelope>
```