# QA Back-End Technologies Basics

## Automotive Data Organization Challenge

Welcome to the **Automotive Enthusiast Project**! Our mission is to gather and analyze data on various car models to assist enthusiasts and potential buyers in making informed decisions. Unfortunately, our comprehensive car database has encountered some disorganization issues. **Structured and detailed data** are vital for accurately comparing car models and understanding their key features. The scrambled data includes essential information about different car models.

You will be provided information on **7 car models**. Each entry includes the **model's ID, name, year of manufacture, fuel efficiency, and a list of features**. The details are presented in a sentence format:

1. "Model ID **501**: **Tesla Model 3**, manufactured in **2020**, achieves a fuel efficiency of **130.1** MPGe. Key features include **Autopilot**, **All-Electric**, **Touchscreen Display**."
2. "Model ID 502: Honda Civic Type R, 2021 model, has a fuel efficiency of 28.2 MPG. Features: Turbocharged Engine, Sport Seats, Adaptive Suspension."
3. "Model ID 503: Ford F-150, year 2021, with a fuel efficiency of 24.3 MPG, features: Hybrid Powertrain, Pro Trailer Backup Assist, Touchscreen Interface."
4. "Model ID 504: Porsche 718 Boxster, manufactured in 2021, offers 22.4 MPG fuel efficiency. It features a Mid-Engine Layout, Convertible Top, Porsche Communication Management."
5. "Model ID 505: Chevrolet Corvette, 2020 model, with 19.5 MPG fuel efficiency, boasts features such as Mid-Engine Design, Leather Seating, Performance Data Recorder."
6. "Model ID 506: BMW 330i, from 2021, has a fuel efficiency of 30.6 MPG. Its features include TwinPower Turbo Engine, iDrive System, Dynamic Cruise Control."
7. "Model ID 507: Mazda MX-5 Miata, 2021 model, offers a fuel efficiency of 29.7 MPG and features like Rear-Wheel Drive, Soft Top, Lane Departure Warning System."

**Convert** the scrambled data into **structured JSON format manually:**

* **Use a text or a code editor** to write the JSON document. We recommend **Notepad++ or VS Code**.
* **Extract relevant details** from each cars' description.
* **Organize the data** into a structured JSON format.
* **Each car** **record** in the JSON document should include **the following attributes**:
* **modelId: Integer** (A unique identifier for each car model)
* **carName: String** (The name of the car model)
* **yearOfManufacture:** Integer (The year the car model was manufactured)
* **fuelEfficiency:** Double (The fuel efficiency of the car model, in MPG or MPGe)
* **features:** Array of Strings (A list of describing three key features of the car model)

You are provided with a **JSON** **parser application**. Use it to **parse and validate** the JSON file you have created.

* **Replace the content of Cars.json** with the JSON data you created.
* After pasting your JSON data into the coresponding JSON file, **make sure to save any changes**.
* **Run the parser** application within your IDE.
* **The parser will process the chosen JSON file** and display the extracted data **in the console**.
* Carefully review the output in the console.
* If the parser displays an error message, check your JSON file for any syntax errors or formatting issues.
* Ensure all required keys are present and correctly named.
* **Save the JSON file, because this would be the file that you have to submit as a solution for this task.**

## The "Revue Crafters" System

**"Revue Crafters"** is an interactive web application for **sharing and managing revues**. It is accessible through a dedicated URL; the platform is designed for people to connect and share their experinces with movies, books, products, etc.. It offers a seamless experience with features like **revue** **creation** and **revue management**.

**Your task** is focused on **using Postman**, **Newman and RestSharp** to conduct **API tests**, ensuring the application's functionality works as expected.  
You can find the Web App here:

[**https://d2925tksfvgq8c.cloudfront.net**](https://d2925tksfvgq8c.cloudfront.net)

### API Endpoints

**"Revue Crafters"** exposes a **RESTful API**, available at**:**   
[**https://d2925tksfvgq8c.cloudfront.net/api**](https://d2925tksfvgq8c.cloudfront.net/api)

**The API is directly available through your browser and you can see all the supported methods.**

The **supported API endpoints** and **the interactive documentation** can be found at:

[**https://d2925tksfvgq8c.cloudfront.net/swagger/index.html**](https://d2925tksfvgq8c.cloudfront.net/swagger/index.html)

For your convenience, here is a **brief overview of the most important endpoints** below as well:

### 1. User

* POST /api/User/Create - create a new user. Post a JSON object in the request body:  
  {   
  "userName": "string",   
  "email": "user@example.com",   
  "password": "string",   
  "rePassword": "string",   
  "acceptedAgreement": true  
  }
* POST /api/User/Authentication - log in an existing user. Post a JSON object in the request body:  
  {  
  "email": "user@example.com",  
  "password": "string"  
  }

### 2. Access Token

* When a user logs in, the response format is JSON object:  
  **{**"email": " user@example.com",  
  **"password": "string",  
  "accessToken": "eyJhbGciOiJ…"  
  }**

**NB! Access token is needed for all revue requests. It should be placed under the Authorization tab, Bearer Token option.**

### 3. Revue

All of the **following requests require Authorization**!

* **GET /api/Revue/All** – list all revues (empty request body).
* **POST** **/api/Revue/Create** – create a new revue.  
  Include a JSON object in the request body (title and description are mandatory, url is optional):   
  **{  
  "title": "string",  
  "url": "",  
  "description": "string"  
  }**
* PUT /api/Revue/Edit – replace the existing revue with the new one.  
  Include a JSON object in the request body (title and description are mandatory, url is optional):   
  **{  
  "title": "string",   
  "url": " ",   
  "description": "string"  
  };**

Requires **queryParameter: ?revueId={revueId}**

* DELETE **/api/Revue/Delete** – delete existing revue;   
  **Requires queryParameter: ?revueId={revueId}**

## RESTful API: Postman API Tests

Your task is to write **API tests** with Postman for certain **RESTful API endpoints**. Organize your tests within a collection, **use collection variables** and **pre-request scripts** to **guarantee successful execution on every run**. **It's important to use collection variables**, **NOT ENVIRONMENT VARIABLES**, to maintain the integrity and portability of the test suite.

### Prerequisites

First you need to **register a new user**. **Registration** of a **new user** is **a mandatory step** that you must complete prior to conducting your API tests. You have the **flexibility to register** either through the [**web UI**](https://d3s5nxhwblsjbi.cloudfront.net/) or **by making a request via Postman**. Please note that this **initial registration process is not included in the scope of your assignment and will not contribute to your final score**. However, it is essential as you will **need an active user account** for all subsequent API requests that form the core of your test cases.   
**If you decide to register via Postman**, **remove this request from your collection.**

### 2.1. Base Setup

* Add the base URL [**https://d2925tksfvgq8c.cloudfront.net**](https://d2925tksfvgq8c.cloudfront.net) as a collection variable **{baseURL}**.
* Ensure all requests use this **{baseURL}**.

### 2.2. Login and Authentication

* Send a **POST request** for **user authentication**.
* **Assert a 200 status** code for success.
* **Assert** that the **response body includes** the attributes **email**, **password**, and **accessToken**. The objective is not to confirm the specific content of these fields but to ensure that they are present in the response.
* Save the **accessToken** as a **collection variable** **{{token}}** for **Bearer Token authorization in subsequent requests**.

### 2.3. Create a New Revue

* Use a **pre-request script** to **generate a random title for the revue** (**a word followed by up to three digits**).
* Store this title as a **{{randomTitle}}** collection variable.
* **Send a POST request** with **{{randomTitle}}** and a **description** (description can be added manually).
* **Assert a 200 status** code.
* **Assert the "Successfully created!" message** in the response body.
* **Assert** that the **title** of the **created revue** in the response matches the expected title.

### 2.4. List all Revues

* **Send a GET request to receive a list of all created revues**.
* **Assert a 200 status** code.
* **Assert** that the **response is an array** and that it contains **at least one item**.
* Extract the **id of the last created revue** from the response body and **store it as a collection variable** **{{lastRevueId}}.**

### 2.5. Edit the Created Revue

* **Send a PUT request** to **modify the revue** identified by **{{lastRevueId}}.** **Change its title** (you can do this manually, no need for scripting).
* **Assert a 200 status** code.
* **Assert** the **"Edited successfully"** message.
* **Assert** that the **title in the response matches the new title you provided.**

### 2.6. Delete the Edited Revue

* Send a **DELETE request to delete the edited Revue** identified by **{{lastRevueId}}.**
* **Assert a 200 status** code.
* **Assert** the **"The revue is deleted!"** message.

### 2.7.Final Steps

1. Make sure that your collection contains all the requests needed:

* **Login**
* **Create New Revue**
* **List All Revues**
* **Edit the Created Revue**
* **Delete the Created Revue**

1. Make sure that the collection can be executed successfully on each run.

**IMPORTANT:**

* Export your collection of requests in Collection v.2.1 format. The export will create a JSON file.
* Make sure that the exported collection can be imported back, e.g. it opens with Postman without problems.
* Do not delete or alter the data in the exported JSON file.

## 3. Newman with htmlextra Reporter

1. **Run** the exported **collection** that you created via Postman in **Newman**.
2. Use **htmlextra as a reporter.**
3. Add the **generated html report** to the archive with your other tasks.