

Submit your work as a **single zip / rar / 7z archive** holding your solutions for each problem at SoftUni Website.

Please refer to the **end of this document** for instructions on **how to submit your work.**

**The "Foody" System**

**"Foody"** is an **interactive platform** designed for users to **share their favourite dishes and culinary ideas**. It's a space for food lovers to engage, share, and **manage culinary delights** The platform, includes key features like user registration, food submission, and management.

Your task is to conduct **API tests using Postman, Newman, and RestSharp**, ensuring the application's functionalities perform as expected.

**Access "Foody"** **Web App** through its dedicated URL:

[**http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:85/**](http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:85/)

**API Endpoints**

**"Foody"** exposes a **RESTful API**, available at**:**   
[**http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:86/api/**](http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:86/api/) Keep in mind that the API is not directly available trough your browser.You can see all the **supported methods** on the **following URL**:

[**http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:86/api/Info/Methods**](http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:86/api/Info/Methods)

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

[**http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:86/swagger/index.html**](http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:86/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",  
  "firstName": "string",   
  "midName": "string",   
  "lastName": "string",   
  "email": "user@example.com",   
  "password": "string",   
  "rePassword": "string"  
  }**
* **POST /api/User/Authentication** - log in an existing user. Post a JSON object in the request body: **{  
  "userName": "string",   
  "password": "string"  
  }**

**2. Access Token**

* When a user logs in, the response format is JSON object:   
  **{  
  "email": "test@gmail.com",   
  "password": "1234567",   
  "accessToken": "eyJhbGciOiJ…"  
  }**

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

**3. Food**

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

* **GET /api/Food/All** – list all foods (empty request body)
* **GET /api/Food/Search** – search foods by their name.   
  Requires **queryParameter: ?keyword=foodName**
* **POST** **/api/Food/Create** – create a new food.   
  Include a JSON object in the request body (title and description are mandatory, url is optional):  
  **{  
  "name": "string",  
  "description": "string",  
  "url": ""  
  }**
* **PATCH /api/Food/Edit/foodId** – change **the title** of existing food.  
  Include a JSON object in the request body:   
  **[  
   {  
   "path": "/name",  
   "op": "replace",  
   "value": "string"  
   }  
  ]**  
  You only have to change the value, with the new title, **leave the path** and **op as they are**. **Use Square brackets as well as curly brackets!**
* **DELETE /api/Food/Delete/foodId** – delete existing food.
* **RESTful API: Postman API Tests (35 points)**

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**](http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:85/) 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.**

* **Base Setup**
* Add the base URL [**http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:86**](http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com:86/) as a collection variable **{baseURL}**.
* Ensure all requests use this **{baseURL}**.
* **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 **username**, **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 value of the **accessToken** as a **collection variable** **{{token}}** for **Bearer Token authorization in subsequent requests**.
* **Create a New Food**
* Use a **pre-request script** to **generate a random food title** (**a word followed by up to three digits**).
* Store this title as a **{{randomFood}}** collection variable.
* **Send a POST request** with **{{randomFood}}** and a **description** (description can be added manually).
* **Assert a 201 status** code.
* **Assert** the response body contains a **foodId** property.
* Save the **foodId** as a **collection variable {{foodId}}**
* **Search Food by Name**
* **Send a GET request to search** for the food that you created **by name**.
* **Use {{randomFood}}** variable as a query parameter.
* **Assert a 200 status** code.
* **Assert** that the **response is an array** and that it contains **the food name that you searched for**.
* **Edit the Name of the Food that you Created**
* **Send a PATCH request** to **change the name of the food you created.**
* **Use {{foodId}}** as a path variable**.**
* **Change the name of the food** (you can do this manually, no need for scripting). \*Check the endpoint's requirements
* **Assert a 200 status** code.
* **Assert** the **"Successfully edited"** message.
* **Delete the Edited Food**
* Send a **DELETE request to delete the edited food**
* Use **{{FoodId}} as path variable.**
* **Assert a 200 status** code.
* **Assert** that the **response message** is **"Deleted successfully!"**.
* **Final Steps**
* Make sure that your collection contains all the requests needed:
* **Login**
* **Create New Food**
* **Search by Food Name**
* **Edit the Name of the Food that you created**
* **Delete the Food that you created**
* Make sure that the **collection** can be **executed successfully on each run**.

Export and save your collection in a **single JSON file**.

* **Newman with htmlextra Reporter (15 points)**
* **Run** the exported **collection** that you created via Postman in **Newman**.
* Use **htmlextra as a reporter.**
* Add the **generated html report** to the archive with your other tasks.
* **RESTful API: RestSharp API Tests (50 points)**

**In this task**, you will demonstrate your ability to interact with a **RESTful API** using **RestSharp** within a **.NET test project**. Your primary goal is to create a set of **automated tests from scratch** that validate the key functionalities of the **Foody API**. You will be **assessed** on your ability to configure a **test project**, **utilize RestSharp** to **make API requests**, and **assert** the expected **responses using NUnit**.

**3.0. Prerequisites**

First, you are required to **set up a new NUnit Test Project** in your Visual Studio. Ensure you **install all necessary packages**, including **RestSharp**, to create a functional API testing suite. This project will serve as the foundation for your subsequent testing tasks.

**3.1. Base Setup**

* **Initialize a RestClient** with the **base URL of the API**.
* Since you already have an account**, authenticate** with **your credentials**, and **store** the received **JWT token**.
* **Configure** the **RestClient with an Authenticator using the stored JWT token**.

**3.2. Data Transfer Objects (DTOs)**

**Before you begin writing your tests**, it's important to **create Data Transfer Objects (DTOs).** Given that you are **familiar** with the **structure of both the requests and responses**, you have the flexibility to **create as many DTOs as you need**. However, these **two DTOs should be sufficient** for the scope of your task:

* **ApiResponseDTO** - this DTO will be used to parse common response structures from the API. It should include the following properties:
* **Msg** of **type string** to capture response messages.
* **FoodId** of **type string** to capture the unique identifier of a food. This field may be null for responses that do not include food ID.
* **FoodDTO** - representing the structure of a food for creation and editing purposes. It should include the following properties:
* **Name** of **type string** for the food's name.
* **Description** of **type string** for the food's description.
* An **optional Url** of **type string** representing a link to the food's picture, if applicable.

**3.3. Create a New Food with the Required Fields**

* **Create a test** to send a **POST request** to **add a new food**.
* **Assert** that the response **status code is Created (201).**
* **Assert** that the **response** body contains a **foodId** property.
* **Store** the **foodId** of the **created food** in a **static member of the test class to maintain its value between test runs**.

**3.4. Edit the Title of the Food that you Created**

* Create a test that **sends a PATCH request** to edit the title of the food
* Use the **foodId** that you **stored in the previous request as a path variable**.
* **Assert** that the **response status code is OK (200).**
* **Assert** that the **response message** indicates the food was **"Successfully edited".**

**3.5. Get All Foods**

* **Create a test to send a GET request to list all foods.**
* **Assert that** the response **status code is OK (200).**
* **Assert that** the response contains a **non-empty array.**

**3.6. Delete the Food that you Edited**

* Create a test that **sends a DELETE request**.
* Use the Use the **foodId** that you **stored as a path variable**.
* **Assert that** the response **status code is OK (200).**
* **Confirm** that the response message is **"Deleted successfully!".**

**3.7. Try to Create a Food without the Required Fields**

* Write a test that attempts to **create a food with missing required fields** (Name, Description).
* Send the **POST reques**t with **the incomplete data**.
* Assert that the response status code is **BadRequest (400).**

**3.8. Try to Edit a Non-existing Food**

* Write a test to **send a PUT request to edit an Food with a foodId that does not exist**.
* **Assert** that the response status code is **NotFound (404).**
* **Assert** that the response message is **"No food revues...".**

**3.9. Try to Delete a Non-existing Food**

* Write a test to **send a DELETE request to edit a food with a foodId that does not exist**.
* **Assert** that the response status code is **BadRequest (400).**
* **Assert** that the response message is **"Unable to delete this food revue!".**

**3.10. Final Steps**

* Ensure that each test is correctly **ordered to maintain the required sequence of actions. Use [Order( )]**
* Verify that tests are designed to **run successfully in on each run.**
* **Delete bin and obj folders** from your solution folder.
* **How to submit your exam**You should have a single **zip / rar / 7z** archive containing all of your tasks

Upload your archive at SoftUni website, into [Regular Exam section](https://softuni.bg/trainings/4399/back-end-test-automation-february-2024).

* The Postman collection should be exported in a single **JSON** file.
* You also need to export the **html file** obtained **from the htmlextra reporter in Newman.**
* Your **RestSharp API Test** project should be **in a folder**.

At the end, the content of your archive should look similar:



Before archiving, please make sure that you **deleted all bin and obj folders from your RestSharp Test project.**