**Back-End Test Automation – Retake Exam**



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 "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 **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.**

### 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: 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 **RevueCrafters 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**.

### 1.0. 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.

### 1.1. 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**.
* If you don’t have an account yet, you can create one however you prefer – either via the web interface or by sending a request to the /api/User/Create endpoint.

**Note:** Account creation is **not part of the exam and will not be evaluated**. You are free to use whichever method is easier for you. The important part is that your tests use a **valid token** obtained after login.

* **Configure** the **RestClient with an Authenticator using the stored JWT token**.

### 1.2. 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.
* **RevueId** of **type string** to capture the unique identifier of a revue. This field may be null for responses that do not include revue id.
* **RevueDTO** - representing the structure of a revue for creation and editing purposes. It should include the following properties:
  + **Title** of **type string** for the revue's title.
  + An **optional Url** of **type string** representing a link to the revue's picture, if applicable.
  + **Description** of **type string** for the revue's description.

### 1.3. Create a New Revue with the Required Fields

* **Create a test** to send a **POST request** to **add a new revue**.
* **Assert** that the response **status code is OK (200)**
* **Assert** that the **response message** indicates the revue was **"Successfully created!"**

### 1.4. Get All Revues

* **Create a test to send a GET request to list all revues.**
* **Assert that** the response **status code is OK (200)**
* **Assert that** the response contains a **non-empty array.**
* **Store** the **id** of the **last created revue** in a **static member of the test class to maintain its value between test runs**.

### 1.5. Edit the Last Revue that you Created

* Create a test that **sends a PUT request** to edit the revue.
* Use the **id** that you **stored in the previous request, as a query parameter**.
* **Assert** that the **response status code is OK (200)**
* **Assert** that the **response message** indicates the revue was **"Edited successfully"**

### 1.6. Delete the Revue that you Edited

* Create a test that **sends a DELETE request**.
* Use the **id** that you **stored in the "Get All Revues" request as a query parameter.**
* **Assert that** the response **status code is OK (200)**
* **Confirm** that the response message is **"The revue is deleted!"**

### 1.7. Try to Create a Revue without the Required Fields

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

### 1.8. Edit a Non-existing Revue

* Write a test to **send a PUT request to edit a revue with a RevueId that does not exist**.
* **Assert** that the response status code is **BadRequest (400).**
* **Assert** that the response message indicates **"There is no such revue!"**

### 1.9. Delete a Non-existing Revue

* Write a test to **send a DELETE request to edit a revue with a RevueId that does not exist**.
* **Assert** that the response status code is **BadRequest (400)**
* **Assert** that the response message indicates **"There is no such revue!"**

### 1.10. 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.

## CI Workflow

Create a GitHub Actions workflow that automatically restores dependencies, builds the project and executes the tests every time code is pushed to the main branch.

### Requirements

The workflow must be commited in the .github/workflows/ci.yml file.

The workflow must trigger on push to main.

The workflow must have the following steps:

* + Checkout the repo;
  + Install the needed SDK;
  + Restore the packages;
  + Build the solution;
  + Run all tests.

Each step should have a meaningful name.

Use environment variables.

Code quality will be taken into account, meaning that the code should be tidy – no unnecessary empty lines, but still the code must be ordered so it is easy to read it.

## How to submit your exam

#### 3.1. Prepare Your Archive for Upload

You need to create a **single .zip, .rar, or .7z archive** that contains:

1. **Your RestSharp API Test project folder**
   * Make sure the folder includes all tests described in the task (from 1.3 to 1.9).
   * Organize your test files clearly and consistently.

* **Delete the bin and obj folders before compressing the project.** These are unnecessary and will make your archive too large.

1. **A separate .TXT file**
   * Inside this file, paste the **URL of your public GitHub repository**   
     (e.g., https://github.com/username/revue-craft-tests).
   * **This file is mandatory**. Without it, your second task won't be evaluated.

#### 3.2. Your Public GitHub Repository

Your GitHub repository must be:

* **Public**, so we can access it without requesting permission;
* Show the **latest successful GitHub Action run** in the **Actions** tab.

#### 3.3. Upload to SoftUni

* Go to the Regular Exam section on the SoftUni platform.
* Upload your compressed archive (.zip, .rar, or .7z).

3.4. Submission Checklist

* All tests (1.3 to 1.9) are implemented correctly;
* Tests are ordered using [Order()] and reusable on every run;
* bin/ and obj/ folders are **deleted**;
* GitHub Repo URL .TXT file is included in the archive;
* The link inside the .TXT file leads to a **public repository**;
* GitHub repo contains .github/workflows/ci.yml file;
* A GitHub Actions workflow was triggered and completed successfully (visible in the Actions tab);
* Environment variables are used in the workflow;
* Archive is uploaded to **SoftUni’s Retake Exam section**.