**Back-End Test**

## The "Story Spoiler" System

**"Story Spoiler"** is an interactive web application for **sharing and managing story spoilers**. It is accessible through a dedicated URL; the platform is designed for story enthusiasts to connect and share. It offers a seamless experience with features like **story spoiler** **creation** and **story spoiler management**. Story Spoiler is also referred as Story.

**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://d3s5nxhwblsjbi.cloudfront.net**](https://d3s5nxhwblsjbi.cloudfront.net)

### API Endpoints

**"Story Spoil"** exposes a **RESTful API**, available at**:**   
[**https://d3s5nxhwblsjbi.cloudfront.net/api**](https://d3s5nxhwblsjbi.cloudfront.net/api)

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

**<https://d3s5nxhwblsjbi.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",  
  "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:  
  **{**"userName": "string",   
  **"password": "1234567",   
  "accessToken": "eyJhbGciOiJ…"  
  }**

**NB! Access token is needed for all story spoiler requests.**

### 3. Story Spoiler

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

* **GET /api/Story/All** – list all story spoilers (empty request body).
* **GET /api/****Story/Search** – search spoilers by their name.  
  Requires **query parameter: ?keyword=storyTitle**
  + **POST** **/api/Story/Create** – create a new story spoiler.  
    Include a JSON object in the request body (title and description are mandatory, url is optional):

**{  
"title": "string",   
"description": "string",**  **"url": ""**

**}**

* PUT /api/Story/Edit/storyId – replace the existing story spoiler with a new one.  
  Include a JSON object in the request body (title and description are mandatory, url is optional):   
  **{  
  "title": "string",   
  "description": "string",   
  "url": ""**

**}**

* DELETE **/api/Story/Delete/storyId** – delete existing story spoiler.

## RESTful API: RestSharp API Tests

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

### 1.3. Create a New Story Spoiler with the Required Fields

* Create a test to send a **POST request** to **add a new story**.
* **Assert** that the response status code is Created (201).
* **Assert** that the **StoryId** is **returned in the response**.
* **Assert** that the **response message** indicates the story was **"Successfully created!".**
* **Store the StoryId** as a static member of the **static member of the test class to maintain its value between test runs**

### 1.4. Edit the Story Spoiler that you Created

* Create a test that **sends a PUT request** to edit the story using the **StoryId** from the **story creation test as a path variable**.
* **Assert** that the **response status code is OK (200).**
* **Assert** that the **response message** indicates the story was **"Successfully edited".**

### 1.4. Get All Story Spoilers

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

### 1.6. Delete a Story Spoiler

* Create test that sends a **DELETE request** using the **StoryId** from the created story.
* **Assert** that the response status code is OK (200).
* **Assert** that the response message **is "Deleted successfully!".**

### 1.7. Try to Create a Story Spoiler without the Required Fields

* Write a test that attempts to **create a story 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 Story Spoiler

* Write a test to **send a PUT request to edit a story with a StoryId that does not exist**.
* **Assert** that the response status code is **NotFound (404).**
* **Assert** that the response message indicates **"No spoilers...".**

### 1.9. Delete a Non-existing Story Spoiler

* Write a test to **send a DELETE request to edit a story with a StoryId that does not exist**.
* **Assert** that the response status code is **Bad request (400).**
* **Assert** that the response message indicates **"Unable to delete this story spoiler!".**

### 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.

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.