**QA Back-End Automation – Retake Exam 05 March 2023**



**Exam** assignment for the ["QA Automation Back-End" Course @ SoftUni](https://softuni.bg/trainings/4021/qa-automation-back-end-january-2023).

Submit your work as a single zip / rar / 7z archive holding the source code for each problem.

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

**C# Unit Tests**

You are given a zip archive, that contains the skeleton for your C# tasks. When you unzip it, open the **"Bank"** folder and open the **"Bank.sln"** file. This will open the Bank solution in your Visual Studio. It contains two projects: BankAccount and BankAccountTests. **BankAccount project** is a **C# project** (a set of several C# classes), which implements certain **functionalities,** with **logic described below**. Your task is to **write unit tests** in C# to assert the project (or certain part of it) works as expected. Write your **unit tests** in **BankAccountTests project**.

**Bank App: Overview**

The **"Bank" app** is a simple bank account simulator designed to keep records of the deposits and withdrawals.

**Input / Output Data**

When opened the **"Bank" app** waits for an account to be **created** with the **initial amount**. Next, it **prompts** for the amount to **deposit**. After the deposit, it **waits** for the **withdrawal amount**. Finally, it **calculates** the **amount** **left in the account**. If the withdrawal sum is **<= 1000, a fee of 0.02 is added**, if the withdrawal sum is **> 1000, the fee is 0.05.**

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | |
| 1000  500  800 | Enter the deposit amount:  Amount after deposit: 1500  Enter the withdrawal amount:  Amount after withdrawal: 684.00 | |
| **Comments** | | |
| **First line:** 1000 is the initial amount in the account  **Second line:** Console asks to enter the deposit amount  **Third line:** 500 is the deposit  **Fourth line:** Console returns the sum from the initial amount and the deposit 🡪1500  **Fifth line:** Console asks to enter the withdrawal amount  **Sixth line:** 800 is the amount to withdraw  **Seventh line:** Console returns the amount left in the account. The withdrawal sum is less than 1000 so the calculated fee is 0.02 🡪 1500 – 800 – 800\*0.02 = 684 | | |
| **Input** | | **Output** |
| -1000 | | Amount cannot be negative! |
| **Comments** | | |
| **First line:** The initial amount of the account cannot be negative | | |
| **Input** | | **Output** |
| 1000  1000  3000 | | Enter the deposit amount:  Amount after deposit: 2000  Enter the withdrawal amount:  Withdrawal exceeds account balance! |
| **Comments** | | |
| **First line:** 1000 is the initial amount in the account  **Second line:** Console asks to enter the deposit amount  **Third line:** 1000 is the deposit  **Fourth line:** Console returns the sum from the initial amount and the deposit 🡪2000  **Fifth line:** Console asks to enter the withdrawal amount  **Sixth line:** 3000 is the amount to withdraw  **Seventh line:** Exception is thrown, because the withdrawal amount is bigger that the available amount 🡪 2000 – 3000 = -1000 | | |
| **Input** | | **Output** |
| 1000  1000  -1000 | | Enter the deposit amount:  Amount after deposit: 2000  Enter the withdrawal amount:  "Withdrawal cannot be negative or zero!" |
| **Comments** | | |
| **First line:** 1000 is the initial amount in the account  **Second line:** Console asks to enter the deposit amount  **Third line:** 1000 is the deposit  **Fourth line:** Console returns the sum from initial amount and the deposit 🡪2000  **Fifth line:** Console asks to enter the withdrawal amount  **Sixth line:** The amount to withdraw is negative number 🡪 -1000  **Seventh line:** Exception is thrown, because the withdrawal amount is negative | | |

**Bank: Unit Tests to Write**

**"Bank"** consists of **two classes: BankAccount** and **BankAccountProcessor**.

Your task is to write unit tests for the **BankAccount** **class and its methods.** (**35 points**):

* public decimal **Amount –** think of **positive, negative** and **border cases** (10 points)
* public void **Deposit -** think of **positive, negative** and **border** **cases** (10 points)
* public void **Withdraw -** think of **positive, negative** and **border cases**, consider **cases about the fee** (15 points)

You should write at **least 2 tests** for the **Amount** and **Deposit** methods, and **3 tests** for the **Withdraw** method.

**No need** to **write tests** for the **BankAccountProcessor class.**

## **The "URL Shortener" System**

The **"URL Shortener"** holds a **collection of URLs**, accessible by a **short code**. It supports the following operations:

* **Add a new URL** -> generates a short URL by given URL + short code
* **Redirect** from a short URL to the original URL
* View **statistics**: URL | Short URL | Date Created | Visits

### Installing and Running the App

To avoid conflicts, it is highly recommended that you **fork the project** for this app from:

[**https://replit.com/@SoftUniQA/Shorturl**](https://replit.com/@SoftUniQA/Shorturl) into your **own repl.it account** and run it from there

Alternatively, you can **install** and **run** the app on your **local machine, assuming that you have Node.js installed**:

|  |
| --- |
| git clone https://github.com/QA-Automation-Testing-Demo/ShortURL  cd ShortURL  npm install  npm start |

### Resetting the App

The app **does not have a persistent database** storage, so you can **reset it** by a simple **restart** (stop & start).

* After restart, you will lose all changes and the default sample data will be populated automatically.

### API Endpoints

**"URL Shortener"** exposes a **RESTful API**, available at:

[**https://shorturl.softuniqa.repl.co/api**](https://shorturl.softuniqa.repl.co/api)or in your case **http://{yoursite}/api**

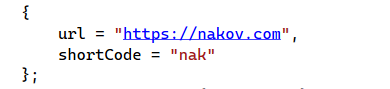
* GET /api – list all API endpoints.
* GET /api/urls – list all shout URLs.
* GET /api/urls/shortCode – finds short URL by given shortCode.
* POST /api/urls – create a new URL short code.
  + Post a JSON objects in the request body, e. g.  
    {"url":"https://cnn.com", "shortCode":"cnn"}
* DELETE /api/urls/shortCode – delete short URL by given shortCode.
* POST /api/urls/visit/shortCode – visit short URL by given shortCode (increases the visits count).

This is a sample output from an API call to /api/urls:



**2. RESTful API: RestSharp API Tests**

Your task is to write **automated tests** in C# for certain RESTful API endpoints. You should implement the following automated tests (**35 points**):

* **List all short URLs** and assert that the returned array is not empty (5 score)
* **Find URL** by short code "nak" and assert that the returned object holds url: "https://nakov.com" (6 score)
* **Search URL** by invalid short code and assert that the response equals "NotFound" (6 score)
* **Create new** short URL and assert the response status code is "OK" and the returning messagge is "Short code added." (10 score)
* **Try to create duplicated short URL** and assert the response code is "BadRequest" (8 score). Example for existing object:  
  

**3. RESTful API: Postman API Tests**

Your task is to write **API tests** with Postman for certain RESTful API endpoints. **(30 points)**You should **organize your tests** in a **collection** that can be run **without failing on the second run**. Use **variables** and **environment**.

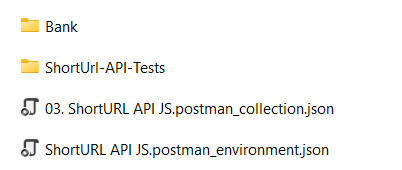
* **List all short URLs** 🡪 assert that the response code is 200, assert that the response type is JSON. (5 points)
* **Create new URL** 🡪 assert the returned code is 200, save URL's shortcode and use it as variable for the next requests (8 points)
* **Find the created URL by shortcode** 🡪 find the URL you just created by shortcode, assert it is the expected URL (7 points)
* Try to **create duplicate URL** 🡪 assert the returned code is 400 (5 points). Example for existing object:



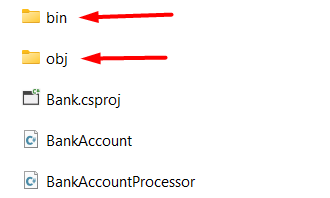
* **Delete** existing URL 🡪 delete the URL you created; assert it's been deleted (5 points)

## **How to submit your exam** You should attach a single **zip / rar / 7z** archive containing all of your tasks. The Postman collection should be exported in a single file. You also need to export the Environment in separate file.

## This is how the files in your archive should look like.



Before archiving, please make sure that you **deleted all bin and obj folders**



Every VS project has its own obj and bin folders so you should delete all of these from **Bank,** **BankTests** and **RestSharpAPITests.**

