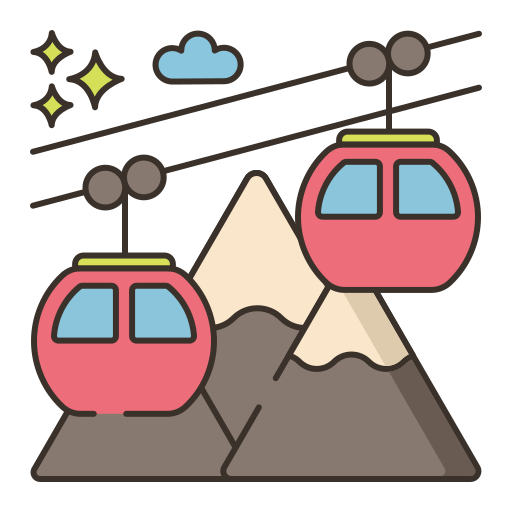
**QA Back-End Automation – Regular Exam 26 Feb 2023**

****

**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**.

1. **C# Unit Tests**

You are given a zip archive, that contains the skeleton for your C# tasks. When you unzip it, open the **"Lift"** folder and open the **"Lift.sln"** file. This will open the Lift solution in your Visual Studio. It contains two projects: Lift and LiftTests. **Lift project** is a **C# project** (a set of several C# classes), which implements certain **app,** 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 **LiftTests project**.

### Lift App: Overview

The **"Lift" app** is designed to **find a place for waiting tourists on a lift. The lift consists of cabins.** Every cabin has **a maximum capacity of 4 people.** If the cabin is full, you should direct the people to **the next one with empty seats** available. The "Lift" app reads from the console the **number of waiting people** in the queue and the **input cabins' state**, calculates the **output cabins' state** and returns what will happen with the queue, when all cabins are processed.

### Input Data

* From **the first line,** the app reads **how many people** are waiting in the queue to get **on the lift**.
* From **the second line**, the app reads the **current state of the lift cabins**, each separated by a single space.

### Output Data

Depending on the lift state and the queue state after all cabins have passed, the app returns the following:

* If there are no more people in the queue and the lift has empty spots:

**"The lift has {count} empty spots!  
{cabins separated by ' '}"**

* If there are still people waiting in the queue and no more available space:

**"There isn't enough space! {people} people in a queue!  
{cabins separated by ' '}"**

* If the lift is full and there are no more people in the queue:

**"All people placed and the lift is full.  
{cabins separated by ' '}"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 15  0 0 0 0 | The lift has 1 empty spots!  4 4 4 3 |
| **Comments** | |
| **First line:** 15 people waiting  **Second line:** 4 empty cabins  **First cabin comes:** 4 0 0 0 -> 11 people left  **Second cabin comes:** 4 4 0 0 -> 7 people left  **Third cabin comes:** 4 4 4 0 -> 3 people left  **Forth cabin comes:** 4 4 4 3 -> 0 people left (1 free spots in the last cabin left) | |
| **Input** | **Output** |
| 20  0 2 0 | There isn't enough space! 10 people in a queue!  4 4 4 |
| **Comments** | |
| **First line:** 20 people waiting  **Second line:** 3 cabins, first – empty, second – with 2 two people on it, third - empty  **First cabin comes:** 4 2 0 -> 16 people left  **Second cabin comes:** 4 4 0 -> 14 people left  **Third cabin comes:** 4 4 4 -> 10 people left, but there are no more cabins. | |
| **Input** | **Output** |
| 3  2 4 3 | All people placed and the lift is full.  4 4 4 |
| **Comments** | |
| **First line:** 3 people waiting  **Second line:** 3 cabins, first – with 2 two people on it, second – full, third - with 3 two people on it  **First cabin comes:** 4 4 0 -> 1 person left  **Second cabin comes:** 4 4 0 -> and it's full, 1 person left  **Third cabin comes:** 4 4 4 -> there are no more waiting people | |

### Lift App: Unit Tests to Write

Write unit tests to cover the method FitPeopleOnTheLift(…) from LiftSimilator Class (**20 points**):

* Check with **valid input** and assert that method works as expected (5 point)
* Check with **invalid people count** (еg. waiting people are zero or negative number) and assert that exception isthrown (5 points)
* Check with **invalid lift size** (eg. cabin size is negative or above 4) and assert that exception is thrown (5 points)
* Check with **invalid lift state** (eg. cabin array is null or cabins count is zero)and assert that exception is thrown (5 points)

Write unit tests to cover the method FitPeopleOnTheLiftAndGetResult(…) **from LiftSimulator Class** (**15 points**):

* Assert message when there is **not enough space on the lift** (5 points)
* Assert message when **lift has empty spaces** (5 points)
* Assert message when **lift is full** (5 points)

Do not write unit tests for the Main() method.

## The "Contact Book" System

The "**Contact Book**" is a simple information system for managing **contacts**. Each contact consists of first name, last name, email, phone and comments. Users can **view** the contacts, **search** by keyword, and **add** new contacts.

You are given the RESTful **API** client for the task board system. Your assignment is to write **API tests** for the system using **RestSharp** and **Postman**.

### 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/contactbook>** 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/ContactBook  cd ContactBook  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

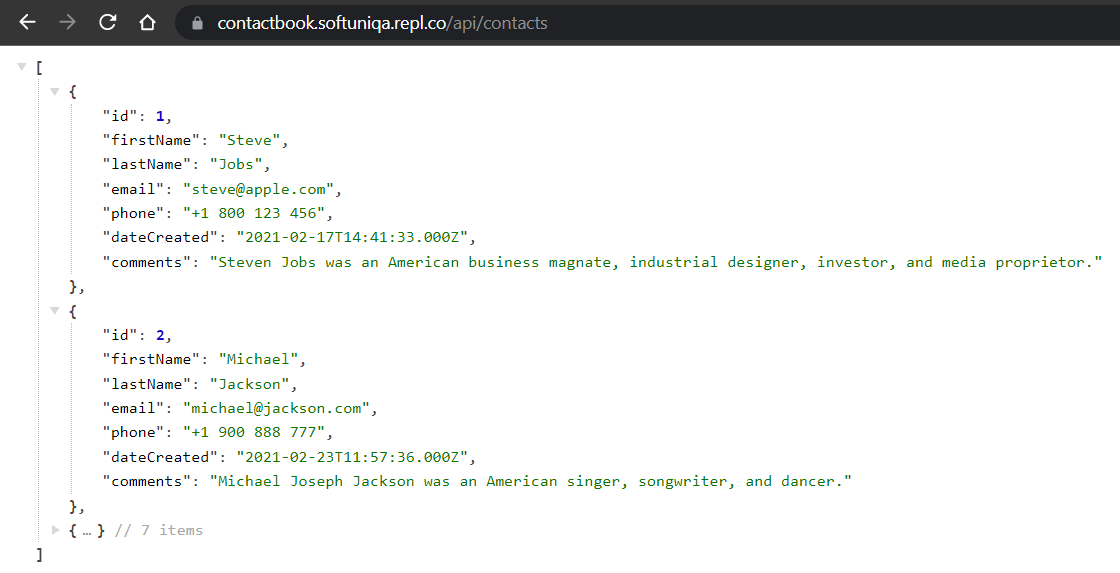
**"ContactBook"** exposes a **RESTful API**, available at:

**<https://contactbook.softuniqa.repl.co/api>** or in your case **http://{yoursite}/api**

The following **API endpoints** are supported:

* GET /api – list all API endpoints
* GET /api/contacts – list all contacts (returns JSON array of contacts)
* GET /api/contacts/id – view contact details by id (returns JSON object)
* GET /api/contacts/search/keyword – find contact by keyword
* POST /api/contacts – create a new contact
  + Post a JSON object in the request body, e. g.  
    {"firstName":"Marie", "lastName":"Curie", "email":"marie67@gmail.com", "phone":"+1 800 200 300", "comments":"Old friend"}
  + Fields firstName, lastName and phone are mandatory
* PATCH /api/contacts/id – edit existing contact. If there is no request body, should return an error.
* DELETE /api/contacts/id – delete a contact by id

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



## 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 contacts** and assert that the first returned object holds first name "Steve" and last name "Jobs" (8 points).
* **Find contacts** by keyword "albert" and assert that first returned object holds first name "Albert" and last name "Einstein" (5 points).
* **Find contacts** by keyword "missing{*random\_number*}" and assert that the results are empty (5 points).
* Try to **create a new contact**, holding invalid data, and assert an error is returned. You may have to check for object error message. (5 points).
* Create a **new contact**, holding valid data, and assert that the status **code is 201 (Created),** returning message is "**Contact added",** and that the **contact is created with all given properties** (12 points)

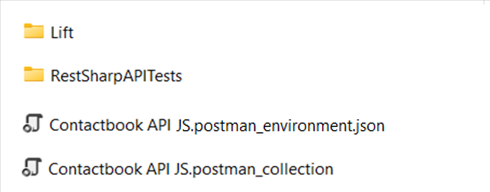
## 3. RESTful API: Postman API Tests

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.

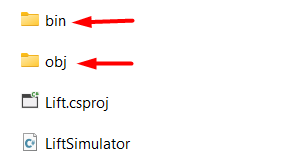
* **List all contacts** 🡪 assert that the response code is 200, assert that the response type is JSON. (5 points)
* **Find contact by keyword** 🡪 find contact with name Michael and assert that the last name is Jackson (6 points)
* **Create contact** 🡪 assert the returned code is 201 or 202, save contact's id and use it as variable for the next request (8 points)
* **Try to edit created contact** 🡪 try to change the name of the contact you just created and assert that the error code is 404 (5points)
* **Delete existing contact** 🡪 delete the contact you created; assert it’s been deleted (6 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 **Lift,** **LiftTests** and **RestSharpAPITests.**

