# QA Back-End Automation - Exam Prep



**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 **"Towns"** folder and open the **"Towns.sln"** file. This will open the Towns solution in your Visual Studio. It contains two projects: **Towns** and **TownsTests**. **Towns 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 **TownsTests project**.

### Towns App: Overview

The **"Towns" app** is designed to **create and keep records of towns. It reads the input from the console and supports the following commands: CREATE, ADD, REMOVE, REVERSE, PRINT, END**

### Input / Output Data

**CREATE – creates a collection of towns' names. If followed by the towns' names, separated by coma, it creates a collection with the given towns. If empty, creates empty collection of towns. In both cases, the App prints "Successfully created collection of towns." on the console.**

**ADD – adds a town to an existing collection. If collection doesn't exist, creates a new one with the name(s) of the current town(s). If empty or duplicate, prints "Cannot add: "**

**REMOVE – removes a town from the collection by a given index. If index is empty or invalid, prints "Invalid operation."**

**REVERSE - reverses the order** of the collection of towns and prints **"Collection of towns reversed."** If the collection is empty or has only one record in it, prints **"Cannot reverse a collection of towns with less than 2 items."**

**PRINT** – prints the collection's current state.

**END** – ends the program

If an **invalid command** is given, **"Invalid command: {name of the command}"** is printed.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| CREATE Sofia, Plovdiv, Varna  PRINT  ADD Veliko Tarnovo  PRINT  ADD Veliko Tarnovo  PRINT  ADD  REMOVE 1  PRINT  REVERSE  PRINT  REMOVE 3  REMOVE  PRINT  CREATE Sofia  PRINT  JUMP  PRINT  END | Successfully created collection of towns.  Towns: Sofia, Plovdiv, Varna  Successfully added: Veliko Tarnovo  Towns: Sofia, Plovdiv, Varna, Veliko Tarnovo  Cannot add: Veliko Tarnovo  Towns: Sofia, Plovdiv, Varna, Veliko Tarnovo  Cannot add:  Successfully removed from index: 1  Towns: Sofia, Varna, Veliko Tarnovo  Collection of towns reversed.  Towns: Veliko Tarnovo, Varna, Sofia  Invalid operation.  Invalid operation.  Towns: Veliko Tarnovo, Varna, Sofia  Successfully created collection of towns.  Towns: Sofia  Ivalid command: JUMP  Towns: Sofia |

### Towns App: Unit Tests to Write

**"Towns" app consists of two classes: TownsCollection and TownProcessor. Your task is to write unit tests for each of them.**

Write unit tests for the methods of **TownsCollection** class (**20 points**):

Write one unit test **for each of the following:**

* public **TownsCollection;** (5 points)
* public bool **Add;** (5 points)
* public void **RemoveAt;** (5 points)
* public void **Reverse;** (5 points)
* Think about **valid** and **invalid** input data.
* You should **write at least 1 test for each of the methods**.

Write unit tests to cover the **TownProcessor** class(**15 points**):

* Write unit tests for the different commands: **CREATE, ADD, REVERSE, REMOVE, PRINT and assert returned messages** are correct
* **Do not write tests for the END command.**
* No need to write tests for each command, write **a total of 3 unit tests for 3 different commands.**

## The "Task Board" System

**"Task Board**" is a simple information system for managing **tasks in a task board**. Each task consists of **title** + **description**. Tasks are organized in **boards**, which are displayed as columns (sections): Open, In Progress, Done. Users can **view** the task board with the tasks, **search** for tasks by keyword, **view** task details, **create** new tasks and **edit** existing tasks (and move existing tasks from one board to another).

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/TaskBoardJS>** into your **own repl.it account and run it from there.**

/**How to run the app on Repl.it** - ref. Running-The-Exam-App-On-Repl-it.docx/

Alternatively, you can **install** and **run** the app on your **local machine**:

|  |
| --- |
| git clone https://github.com/QA-Automation-Testing-Demo/TaskBoard-JS  cd TaskBoard-JS  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

TaskBoard exposes a **RESTful API**, available at:

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

The following endpoints are supported:

* GET /api – list all API endpoints
* GET /api/tasks – list all tasks (returns JSON array of tasks)
* GET /api/tasks/id – returns a task by given id
* GET /api/tasks/search/keyword – list all tasks matching given keyword
* GET /api/tasks/board/boardName – list tasks by board name
* POST /api/tasks – create a new task (post a JSON object in the request body, e.g. {"title":"Add Tests", "description":"API + UI tests", "board":"Open"})
* PATCH /api/tasks/id – edit task by id (send a JSON object in the request body, holding the fields to modify, e.g. {"title":"changed title", "board":"Done"})
* DELETE /api/tasks/id – delete task by id
* GET /api/boards – list all boards

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

## 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**):

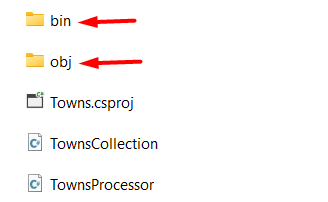
* **List the tasks** and assert that the first task from board "Done" has title "Project skeleton" (8 points).
* **Find tasks** by keyword "home" and assert that the first result has title "Home page" (5 points).
* **Find tasks** by keyword "missing{*randnum*}" and assert that the results are empty (5 points).
* Try to **create a new task**, holding invalid data, and assert an error is returned (5 points).
* Create a **new task**, holding valid data, and assert the **new task is successfully created with respective properties** (12 points).

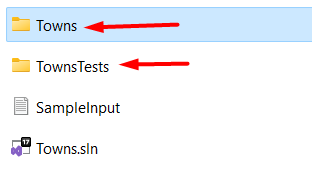
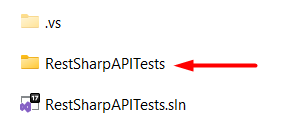
## RESTful API: Postman API Tests

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 boards** 🡪 assert that the response code is 200, assert that the returned response type is json. (5 points)
* **List the tasks by board name "Done"** 🡪 assert the returned code is 200, assert that all tasks are from the board "Done" (6 points)
* **Create task** 🡪 assert the returned code is 201 or 202, save task's id and use it as variable for the next two requests (8 points)
* **Edit created task** 🡪 change the name of the task you just created and assert it's not empty string (6 points)
* **Delete existing task** 🡪 delete the task you patched; 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 project has its own obj and bin folders so you should delete all of these from **Towns,** **TownsTests** and **RestSharpAPITests.**