**QA Back-End Test Automation**

**Exam Preparation I**

You can check your solutions in [Judge](https://judge.softuni.org/Contests/4698/Exam-Prep-I).

The **Travelers** app focuses on providing functionality for managing **travel-related data**, such as **destinations** and **categories**. **Before running the tests do not forget to start your API**.

## How to Run the Project

You should have installed **Docker**.

Follow these steps to get the application running in a Docker container.

1. **Download** the **Travelers**.zip file, which contains all the necessary files.
2. **Unzip** the **Travelers**.zip file into your preferred directory on your machine.
3. **Build** and **Run the Docker Containers**.

Ensure you have **Docker** and **Docker Compose** installed. Then, run the following command to build and start the containers:

**docker-compose up --build**

This command will load the Docker image into your local Docker environment.

1. **Access** the API

Once the containers are up and running, you can access the API at <http://localhost:5000/api>.

1. API **Documentation**

API documentation is available at <http://localhost:5000/api-docs>.

## CRUD Operations for Destinations Testing (150 Points)

Open Skeleton.zip file. You can write your test inside the methods in ApiTests.cs. Be careful not to change the names of the following methods.

### Get All Destinations (30 Points)

Write a unit test for the **Test\_GetAllDestinations**() method.

Conditions to Test:

* Ensure that the **category** filter returns an array of the **correct destinations**.
* Verify that the **output format matches the specified string format**.
* **Response Assertions:**
  + The HTTP response status code should be 200 OK.
  + The response content should not be empty.
* **Data Structure Assertions:**
  + The response content should be a JSON array.
  + The JSON array should contain at least one destination.
* **Destination Fields Assertions (for each destination):**
  + Each destination's name should not be null or empty.
  + Each destination's location should not be null or empty.
  + Each destination's description should not be null or empty.
  + Each destination's category should not be null or empty.

### Get Destination by Name (30 Points)

Write a unit test for the **Test\_GetDestinationByName**() method.

Conditions to Test:

* Verify that the **correct destination** is returned **based on the name** for example "**New York City**".
* **Response Assertions:**
  + The HTTP response status code should be 200 OK.
  + The response content should not be empty.
* **Destination Fields Assertions:**
  + The location of the destination with the name "New York City" should be "New York, USA".
  + The description of the destination with the name "New York City" should match the given description.

### Add Destination (30 Points)

Write a unit test for the **Test\_AddDestination**() method.

Conditions to Test:

* Verify that the **new destination** is **added** **correctly** to the array.
* Ensure that the **destination** has the **correct ID**, **name**, **location**, **category** and **description**.
* **Response Assertions:**
  + The HTTP response status code should be 200 OK.
  + The response content should not be empty.
* **Destination Fields Assertions:**
  + The name in the response should match the input value ("Maui Beach").
  + The location in the response should match the input value ("Hawaii, USA").
  + The category in the response should not be null or empty.
  + The description in the response should not be null or empty.

### Update Destination (30 Points)

Write a unit test for the **Test\_UpdateDestination**() method.

Conditions to Test:

* Verify that the **destination's name = "Summer in Machu Picchu", best time to visit description** is **updated correctly**.
* Ensure the method handles non-existent destination IDs properly.
* **Get Request Assertions:**
  + The HTTP response status code for the GET request should be 200 OK.
  + The GET request response content should not be empty.
  + The destination with the name "Machu Picchu" should exist in the response.
* **Update Request Assertions:**
  + The HTTP response status code for the PUT request should be 200 OK.
  + The PUT request response content should not be empty.
* **Updated Fields Assertions:**
  + The updated name should be "Summer in Machu Picchu".
  + The updated bestTimeToVisit should be "Summer!".
  + The updated description should not be null or empty.

### Remove Destination by Id (30 Points)

Write a unit test for the **Test\_DeleteDestination**() method.

Conditions to Test:

* Ensure that the destination is **removed successfully**.
* Test for **non-existent destination IDs** and ensure appropriate error handling.
* **Get Request Assertions:**
  + The HTTP response status code for the initial GET request should be 200 OK.
  + The GET request response content should not be empty.
  + The destination with the name "Yellowstone National Park" should exist in the response.
* **Delete Request Assertions:**
  + The HTTP response status code for the DELETE request should be 200 OK.
* **Verification Assertions:**
  + A subsequent GET request for the deleted destination should return a response with either null content or the string "null".

## Category Management Tests (150 Points)

You can write your test inside the methods in CategoryTests.cs. Be careful not to change the name of the following method.

### Test\_CategoryLifecycle (150 Points)

This test case ensures that the entire **lifecycle of a category** can be performed **successfully**. The lifecycle includes **creating** a category, **retrieving** it (both by listing all categories and by ID), **editing** the category, and finally **deleting** it.

#### Step 1: Create a new category

* **Method:** POST /category
* **Description:** A request is made to create a new category with the name "Test Category."
* **Expected Status Code:** 200 OK
* **Assertions:**
  + The HTTP response code should be 200 OK.
  + The response body should contain a **non-null**, **non-empty** category ID (**\_id**).

#### Step 2: Get all categories

* **Method:** GET /category
* **Description:** Fetch all categories to ensure that the newly created category appears in the list of categories.
* **Expected Status Code:** 200 OK
* **Assertions:**
  + The HTTP response code should be 200 OK.
  + The response content should not be empty.
  + The response should be a JSON array.
  + The array should contain at least one category.

#### Step 3: Get category by ID

* **Method:** GET /category/{categoryId}
* **Description:** Retrieve the category by the ID that was returned during the creation step.
* **Expected Status Code:** 200 OK
* **Assertions:**
  + The HTTP response code should be 200 OK.
  + The response content should not be empty.
  + The returned category should have the same ID as the one created.
  + The category name should be "**Test Category**".

#### Step 4: Edit the category

* **Method:** PUT /category/{categoryId}
* **Description:** Update the category name to "**Updated Test Category**" using a PUT request.
* **Expected Status Code:** 200 OK
* **Assertions:**
  + The HTTP response code should be **200 OK**.

#### Verification (after edit):

* **Method**: GET /category/{categoryId}
* **Description:** Fetch the updated category to ensure the name has been changed.
* **Expected Status Code:** 200 OK
* **Assertions:**
  + The HTTP response code should be 200 OK.
  + The response content should not be empty.
  + The category name should be updated to "**Updated Test Category**".

#### Step 5: Delete the category

* **Method:** DELETE /category/{categoryId}
* **Description:** Delete the category by its ID using a DELETE request.
* **Expected Status Code:** 200 OK
* **Assertions:**
  + The HTTP response code should be **200 OK**.

## How to Submit Your Work

You need to submit your work on the SoftUni website in the Exam Section.

1. Archive the folder that contains your solution.
2. Upload the archive to the SoftUni website in the course section for your exam.