**QA Back-End Test Automation**

**Exam Preparation 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 must have installed **Node.js** – see detailed instructions here <https://nodejs.org/en/download/prebuilt-installer>

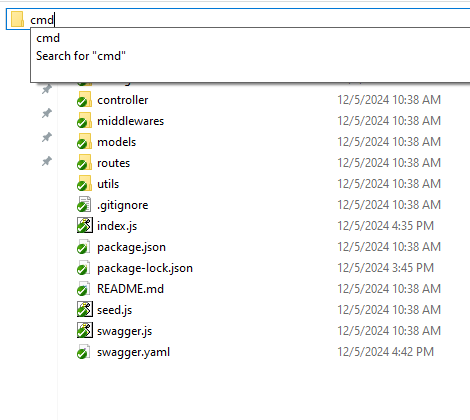
Follow these steps to get the application running locally.

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.

A red rectangle with black text

Description automatically generated

1. Navigate to the unzipped folder in File Explorer and **Open in Terminal** there.
   1. Either using right-click **Open in Terminal**A screenshot of a computer

      Description automatically generated
   2. Or by writing **cmd** in the File Explorer path and pressing Enter 
2. Execute the command **npm install** and wait for it to install the needed packages.
3. Execute the command **npm run start** and wait for it to start the application.
   1. **NOTE: Leave the cmd Terminal open, as closing it will close the server**.
4. Access the **API Documentation** available at <http://localhost:5000/api-docs>.
5. Keep in mind that Tests can affect the state of the data on the server, to ensure you’re working with the base data, you can reset the server by closing the Terminal and repeating steps **3** and **5**
   1. An example of a test that can affect the server data is one that deletes an object on the server, running it a second time will fail as that object would no longer be available.
   2. If you’re testing with swagger (<http://localhost:5000/api-docs>), keep in mind that it might be necessary to refresh the page after restarting the server, to avoid errors due to old data/examples.

## Solution Skeleton and Authentication

Unzip the Skeleton.zip file. You must write your tests inside the methods in DestinationTests.cs (Task 1) and **CategoryTests.cs (Task 2)**. Be careful not to change the names of the methods.

You are provided with a **Setup** function. Its role is to login to the application and save 2 variables you must use for your tests:

* **client** – initialized RestClient ready to be used for your tests. The base url for the API endpoints is already saved. In your tests you need to only add the name of the endpoint as first parameter.
* **token** – initialized authentication token ready to be used for your tests. When you need to execute request to an endpoint that requires authentication you need to add the token to the header like this:



## CRUD Operations for Destinations Testing (150 Points)

### 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.
  + Each destination’s **attractions** should be a JSON array.
  + Each destination’s **bestTimeToVisit** should not be null or empty.

### Get Destination by Name (30 Points)

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

Conditions to Test:

* **Response Assertions:**
  + The HTTP response status code should be **200 OK**.
  + The response content should **not be empty**.
* **Data Assertions:**
  + Verify that a Destination with the name "**New York City**" is returned in the received destinations.
* **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.

To create a new Destination, you need to create a valid object with **name**, **location**, **description**, **bestTimeToVisit**, **attractions** and **category**. The **category** property must contain a valid **id** of one of the existing categories. To do that you need to retrieve the list of categories and set the category property to have the value of the id of one of the categories returned.

Conditions to Test:

* **Get all categories**
* **Create a new Destination**
* **Response Assertions:**
  + The HTTP response status code should be **200 OK**.
  + The response content should **not** be **empty**.
* **Retrieve the id of the created Destination from the response**
* **Get the details of the Destination**
* **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.
  + The **location** in the response should match the input value.
  + The **description** in the response should match the input value.
  + The **bestTimeToVisit** in the response should match the input value.
  + The **category** should not be empty.
    - The **id** of the category should match the input value (keep in mind that here the category is returned as an object, which has the **\_id** property you need).
  + The **attractions** should be a **JSON array**.
    - The array should have the same **number of elements** as the input value for **attractions**.
    - The **values** of the elements should be the same as the input values for **attractions.**

### Update Destination (30 Points)

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

Conditions to Test:

* **Get all Destinations**
* **Get Request Assertions:**
  + The HTTP response status code for the GET request should be **200 O**K.
  + The GET request response content should not be empty.
  + The Destination with the name "**Machu Picchu**" should exist in the response.
* **Get the id of the Destination.**
* **Update** the Destination withnew **name** and **bestTimeToVisit.**
* **Update** **Response Assertions:**
  + The HTTP response status code for the PUT request should be **200 OK**.
  + The PUT response content should not be empty.
* **Updated Fields Assertions:**
  + The updated **name** should match the input value.
  + The updated **bestTimeToVisit** should match the input value.

### Remove Destination by Id (30 Points)

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

Conditions to Test:

* **Get all Destinations**
* **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.
* **Get the Id of the destination**
* **Delete the Destination**
* **Delete Response 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 content "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**.

#### Step 5: 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 6: 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**.

#### Step 7: Verify the deleted category cannot be found

* **Method: GET /category/{categoryId}**
* **Description: Attempt to retrieve the deleted category to confirm that it has been removed.**
* **Assertions:**
  + The response content should be **“null”**.

## 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.