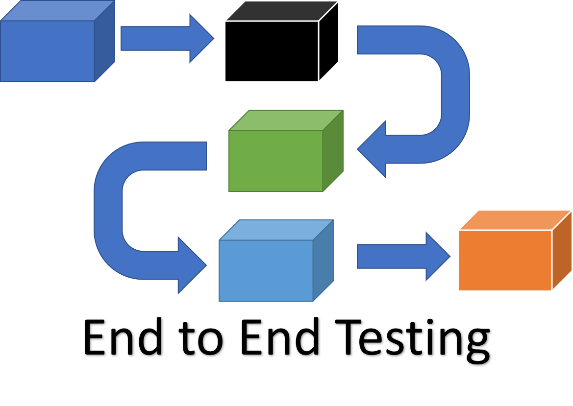
# Workshop: Lifecycle Management and Automation



The **EShop** is a comprehensive **e-commerce application** with multiple components, including **user management**, **product management**, **order processing**, and more. Here's an overview of the key aspects and functionalities of the project:

### 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 **EShop.zip** file, which contains all the necessary files.
2. **Unzip** the **EShop.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**

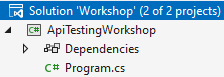
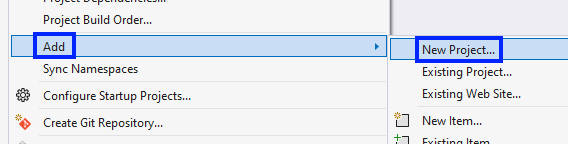
API documentation is available at <http://localhost:5000/api-docs>.

## Setting Up the Project

### Create a New C# Project

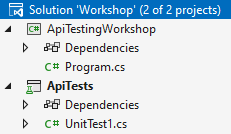
Open Visual Studio and create a new C# Console Application project named ApiTestingWorkshop.

Right-click the solution in Solution Explorer and select **Add** 🡪 **New Project**.

 🡪

Choose **Unit Test Project** and name it ApiTests.

Your project should have this architecture for now:



### Install NUnit and RestSharp via NuGet

Right-click on the ApiTests project and select **Manage NuGet Packages**.

Search for and install the following packages:

* NUnit
* NUnit3TestAdapter
* Microsoft.NET.Test.Sdk
* RestSharp

### Configure the Project for Unit Testing

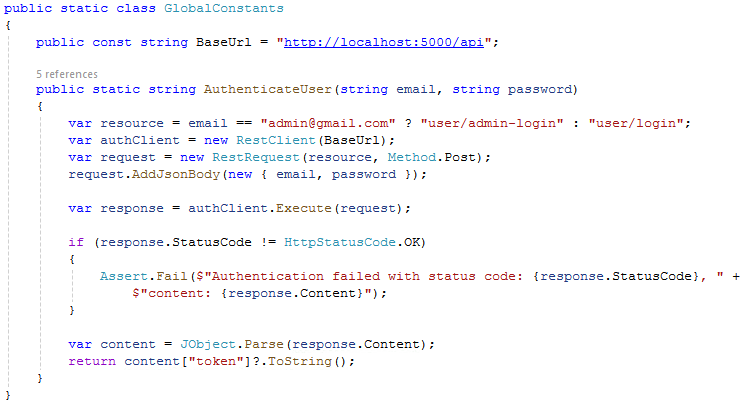
Ensure that the ApiTests project references NUnit and is set up to run tests. Check that the NUnit3TestAdapter is properly installed.

### Add Global Constants

Create new class GlobalConstants.cs. This class is a static utility class that contains essential **constants** and **methods** for the application. It includes:

* **BaseUrl:** A constant string that defines the API's base URL ("<http://localhost:5000/api>").
* **AuthenticateUser Method:** A method that handles user authentication by sending a POST request to the API. It returns an **authentication token** if successful, or fails the test with an error message if authentication fails.

This class centralizes key **API configurations** and **authentication logic** for easy reuse across the application.

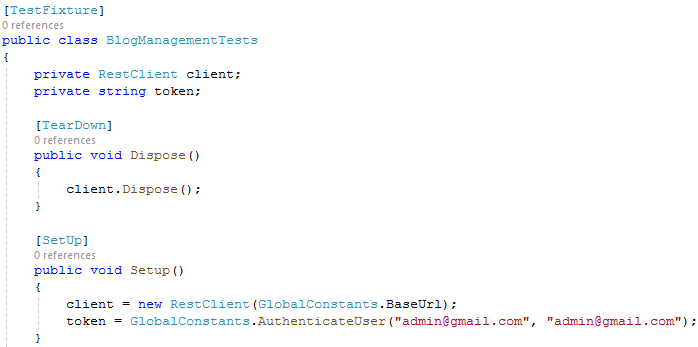


## Blog Management Tests

Create a new test class named **BlogManagementTests**.cs in the ApiTests project.

The **BlogManagementTests** class is a test fixture, marked with the [TestFixture] attribute, designed to test the **Blog API**. It includes a RestClient for making **API requests**. The Dispose() **method** disposes of the RestClient instance to prevent resource leaks, maintaining a clean test environment.

The **Setup** method initializes the **RestClient** and **authenticates** the user **before each test**, ensuring a secure and consistent testing environment by **validating** the **authentication token**.

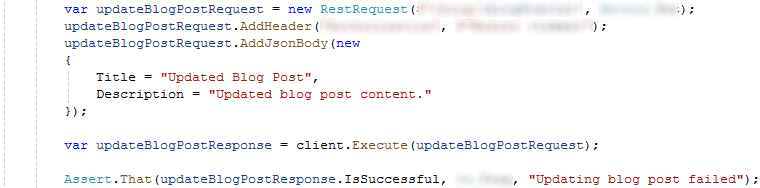


### Blog Post Lifecycle Test

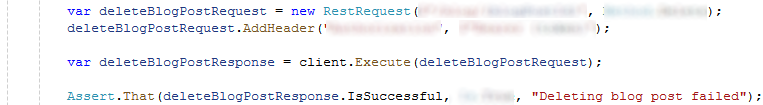
The BlogPostLifecycleTest **method** is an automated test that verifies the complete lifecycle of a blog post through a series of **API requests**. The test begins by **creating** a **new blog post** using a **POST request**, where it checks that the **post** was **successfully** **created** and that a **valid blog post ID is returned**. Don’t forget to attach **authorization token** to the header for all requests.



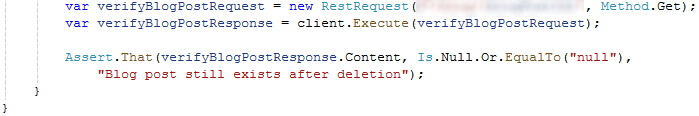
The test **updates the existing blog post** via a **PUT request** and asserts that the **update operation is successful**.



Following this, the **blog post** is **deleted** with a **DELETE request**, and the test ensures that this **deletion is successful**.



Finally, the test verifies that the **deleted blog post no longer exists** by sending a **GET request**, confirming that the **content is null** or the blog post has been **removed**.

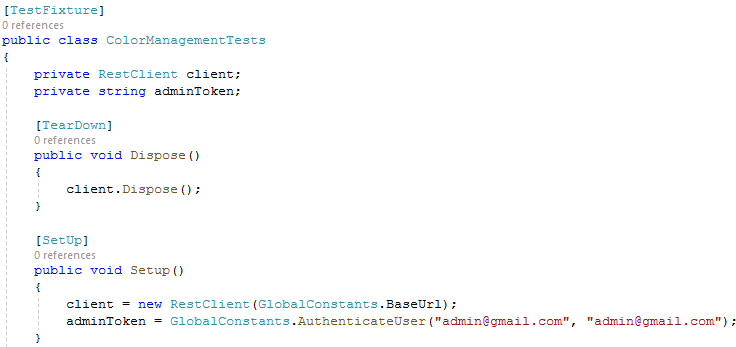


## Color Management Tests

Create a new test class named **ColorManagementTests**.cs in the ApiTests **project**.

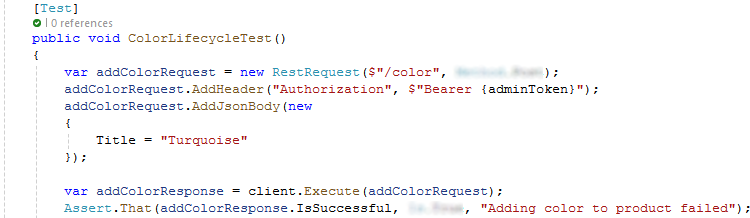
The ColorManagementTestsclass is a test fixture, marked with the **[TestFixture]** attribute, designed to test the **Color API**. It includes a **RestClient** for making **API requests**. The **Dispose()** **method** disposes of the **RestClient** instance to prevent resource leaks, maintaining a clean test environment.

The **Setup** method initializes the **RestClient** and **authenticates** the user **before each test**, ensuring a secure and consistent testing environment by **validating** the **authentication token**.

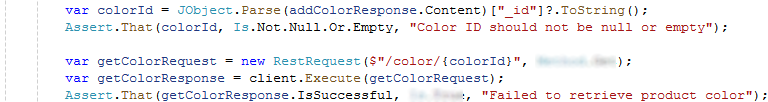


### Color Lifecycle Test

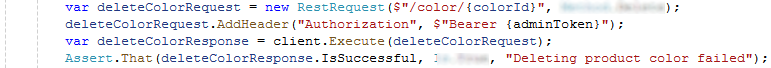
The test method named ColorLifecycleTest that **validates** **the** **lifecycle** of a "**color**" entity in a product management system. A **POST request** is sent to **add a new color** (in this case, "**Turquoise**") to the system. The request includes an **authorization header** and a **JSON body**. The test **asserts** that the color was **successfully added**.

****

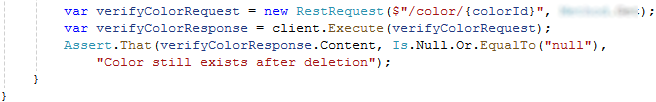
The **color ID** returned from the previous step is **extracted from the response**, and a **GET request** is made to **retrieve the color details**. The test **asserts** that the **retrieval was successful**.



A **DELETE request** is sent to **remove the color using its ID**, and the test checks that the **deletion was successful**.



A final **GET request** checks whether the **color still exists** in the **system after deletion**. The test **asserts** that the **color no longer exists**.



### Color Lifecycle Negative Test

To create a **negative unit test** for the ColorLifecycleTest method, we need to simulate conditions where the API **calls fai**l and assert that the **expected failure scenarios are handled correctly**. This helps ensure that your code can gracefully handle errors, such as **invalid inputs** or **server** **issues**. The ColorLifecycleNegativeTest ensures that API requests for **adding**, **retrieving**, and **deleting** a **color** **fail** when using an **invalid token** or **invalid** color **ID**. It verifies that an attempt to add a color with an invalid token results in an InternalServerError, as does attempting to retrieve or **delete** a color with an **invalid** color **ID**. The test is designed to confirm that the API responds appropriately to **invalid inputs**, by returning **error statuses** instead of **succeeding** in these scenarios.

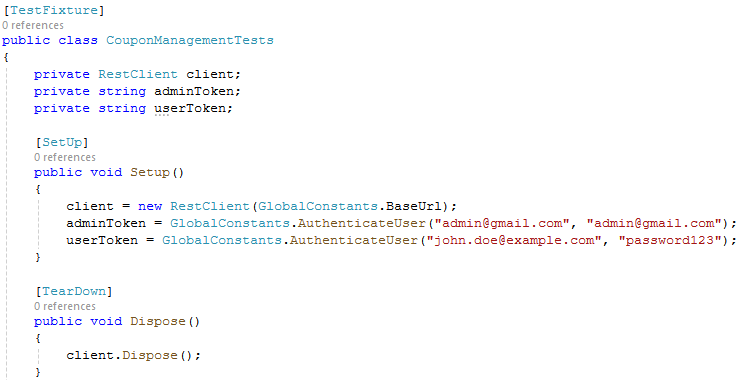


## Coupon Management Tests

Create a new test class named **CouponManagementTests**.cs in the ApiTests **project**.

The CouponManagementTestsclass is a test fixture, marked with the **[TestFixture]** attribute, designed to test the **Coupon API**. It includes a **RestClient** for making **API requests**. The **Dispose()** **method** disposes of the **RestClient** instance to prevent resource leaks, maintaining a clean test environment.

The **Setup** method initializes the **RestClient** and **authenticates** the user **before each test**, ensuring a secure and consistent testing environment by **validating** the **authentication token**.

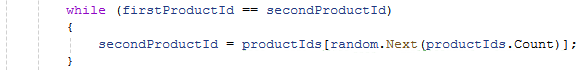


### Coupon Lifecycle Test

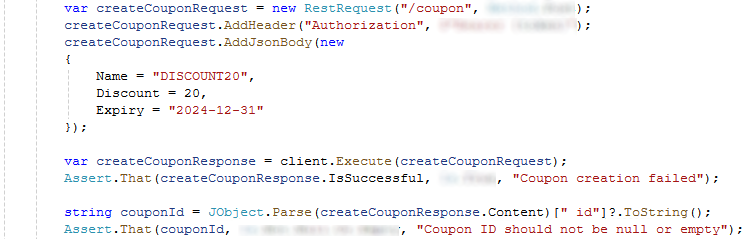
The CouponLifecycleTest **method** begins by retrieving a list of **product IDs** from the database and **randomly** selecting **two distinct product IDs** for use in the test.



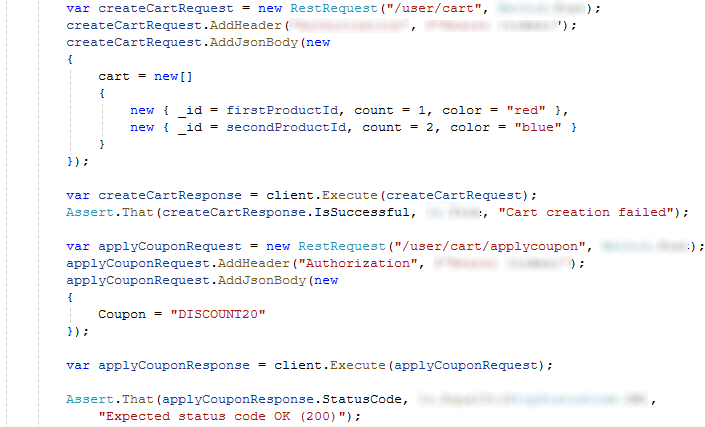
The while loop checks whether the two selected product IDs (firstProductId and secondProductId) are the same. If they are the same, the loop continues to randomly select a new secondProductId **until it is different** from firstProductId.



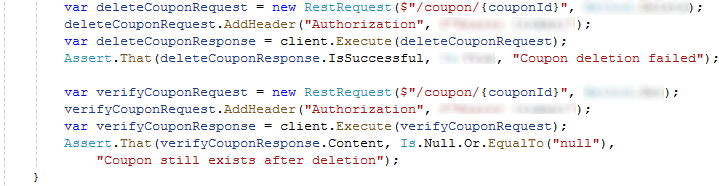
It then creates a **new coupon** with specific **details** such as a **discount percentage** and **expiry date**, **ensuring** the coupon is **successfully created** and its ID is stored for further operations. For authorization use **admin token**.



Next, the test **creates** a **shopping cart**, **adds the selected products** to it, and **applies the created coupon to the cart**, verifying that the coupon application is successful by checking for an **HTTP 200 OK status code**.

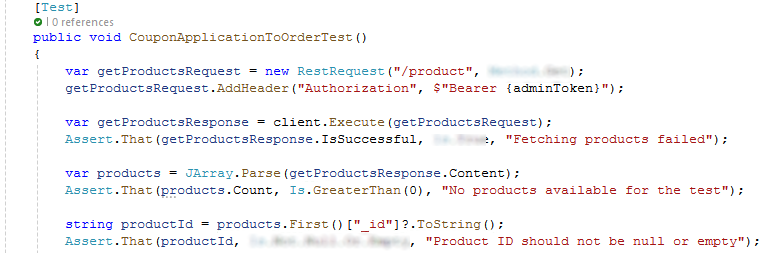


Finally, the test **deletes the coupon** and **verifies that the coupon has been properly removed** from the system by attempting to retrieve it and ensuring the response indicates that the **coupon no longer exists**.

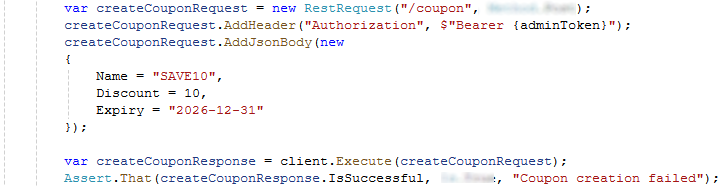


### Coupon Application to Order Test

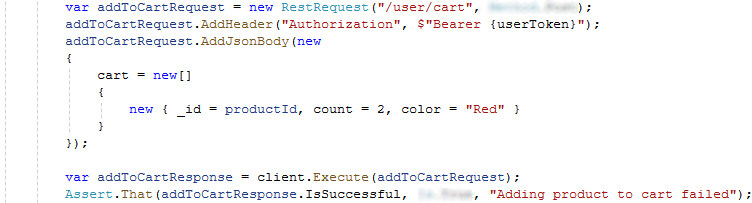
The CouponApplicationToOrderTest **method** is a **unit test** that validates the process of **applying a coupon** to an **order**. The test begins by **fetching a list of available products** to **ensure** that **products exist** in the system.



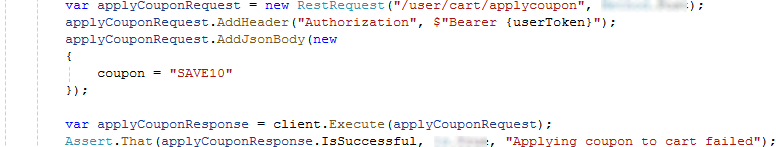
It then **creates a new coupon** with a specified discount using a **POST request**.



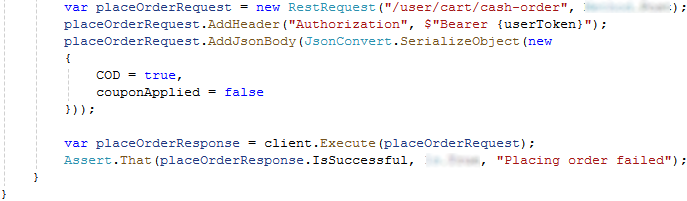
After **successfully creating** the coupon, the test **adds a specific product** **to the user's cart**.



Once the product is in the cart, the test **applies** the **created coupon** to the cart via another **POST request**.

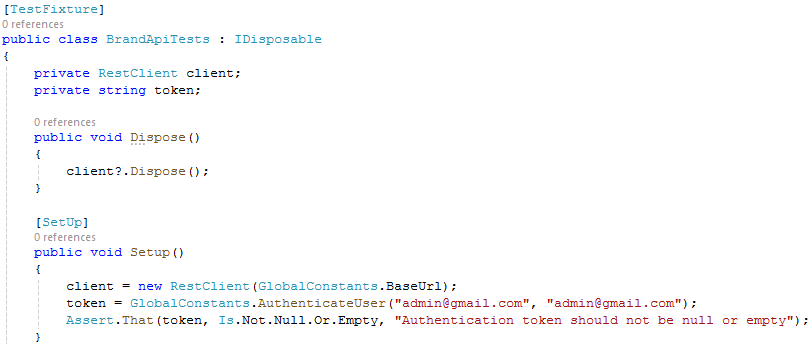


Finally, the test attempts to place an order with the applied coupon and a **Cash on Delivery** (**COD**) option.



## Order Management Tests

Create a new test class named BrandApiTests.cs in the ApiTests project. This class is a test fixture designed to test the Brand API, using a **RestClient** for API requests. It implements IDisposable to clean up resources and prevent leaks. The Setup method initializes the **RestClient** and **authenticates** the user **before each test**, ensuring a secure and consistent testing environment by **validating** the **authentication token**.



### Complex Order Lifecycle Test

The ComplexOrderLifecycleTest begins by **fetching available products**, then **adds** a selected **product** to the **user's cart**.

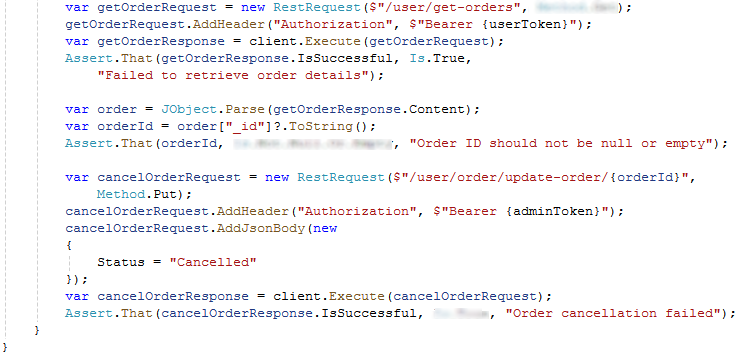




A **coupon is applied to the cart**, and an **order is placed using Cash on Delivery** (**COD**).



The test then retrieves the order details and **finally updates** the **order status** to "**Cancelled**". Throughout the process, assertions are made to ensure **each** **step is successful**, verifying that the system behaves as expected in **handling complex** **order scenarios**.



## Product Management Tests

Create a new test class named **ProductManagementTests**.cs in the ApiTests project.

The **ProductManagementTests** class, marked with [**TestFixture**], tests the User API using a **RestClient** for API requests. The **Dispose()** **method** cleans up the **RestClient** to prevent resource leaks. The **Setup** method initializes the **RestClient** and **authenticates the user before each test**, ensuring consistent and secure testing.

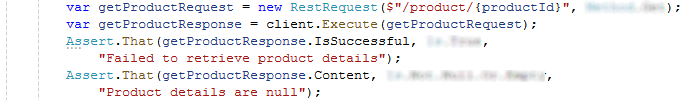


### Product Lifecycle Test

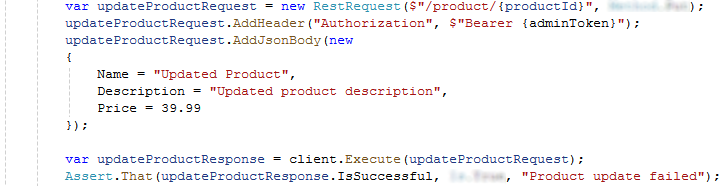
The ProductLifecycleTest **method** **verifies the complete lifecycle of a product** in an API, from creation to deletion. The test starts by sending a **POST request** to **create a new product**, including details like **title**, **description**, **price**, **category**, **brand**, and **quantity**. It checks that the product is **successfully** **created** and **retrieves** **the product ID** **from the** **response**.



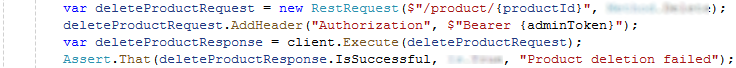
The test then sends a **GET request** to retrieve the details of the **newly created product**. It verifies that the **product** **details** are **successfully retrieved** and the **response contains the expected data**.



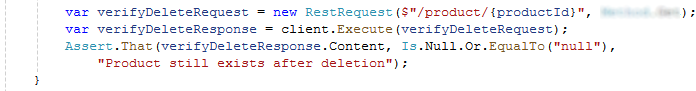
Next, the test updates the product by sending a **PUT request** to modifying attributes like **name**, **description**, and **price**. It ensures the **update** is **successful by checking the response**.



The test then deletes the product by sending a **DELETE request** and verifies that the **deletion process is successful**.

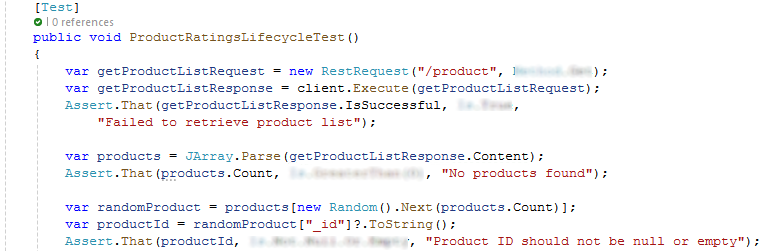


Finally, the test attempts to retrieve the product again with a **GET request** to confirm that the **product** **has been** **deleted**. It checks that the **response** **indicates the product no longer exists**.

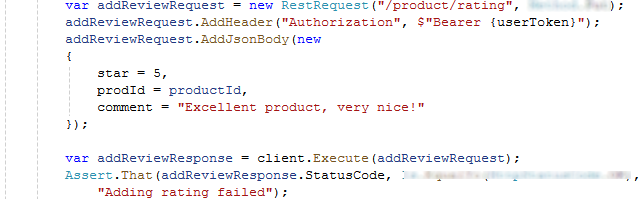


### Product Rating Lifecycle Test

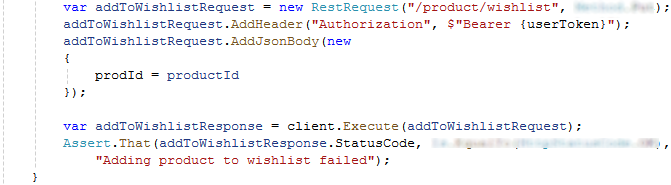
The test begins by enthusiastically retrieving a **random product ID** **from** **the** **product list**, ensuring the list is not only available but filled with a variety of options. It then **randomly selects a product**, ensuring the chosen product **ID** is **valid and not null**, which sets the stage for the next steps.



The second step is where the excitement builds, as the test joyfully **adds a glowing 5-star review to the randomly selected product**, accompanied by a positive and encouraging **comment**. The test carefully checks that this **review** is **successfully added**, confirming that the API correctly handles user feedback and ratings.

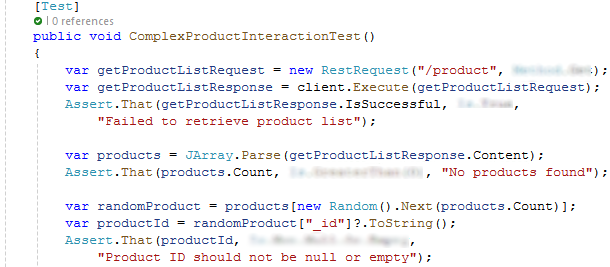


Finally, in a satisfying conclusion, the test adds the **cherished product to the user's wishlist**, ensuring the product is securely **stored for future enjoyment**.

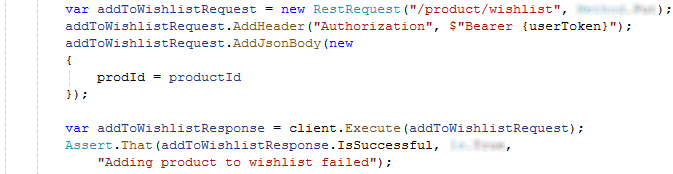


### Complex Product Interaction Test

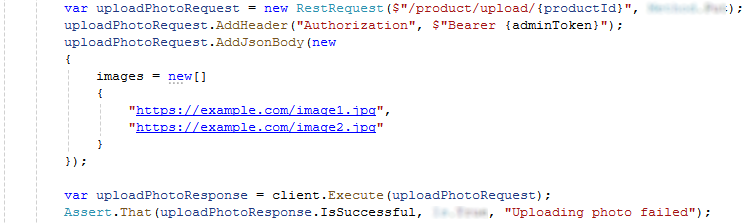
The test starts by sending a **GET request** to fetch a **list of available products**. It ensures that the **product list** is **successfully retrieved** and that it contains at least one product. From the retrieved product list, the test **randomly selects one product** and **extracts its ID**. This product ID is used in subsequent operations.



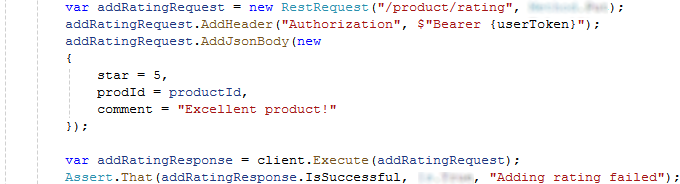
The test adds the selected product to the user's wishlist by sending a **PUT request**. It **includes** **the** **product ID** in the request and **verifies that the product is successfully added to the wishlist**.

****

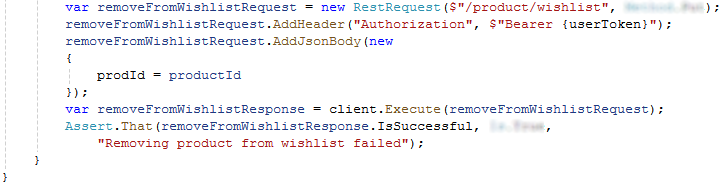
The test uploads new photos for the selected product by sending a **PUT request**. The request includes a list of **image URLs**, and the test checks that the **photo upload is successful**.

****

The test **adds** a 5-star **rating** and a **positive review** for the selected product by sending a **PUT request**. It verifies that the **rating** **is successfully added**.



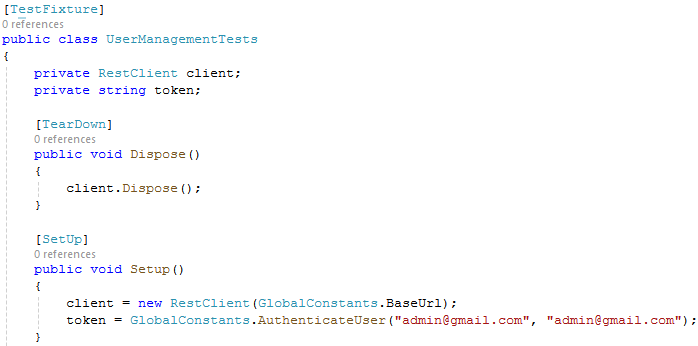
Finally, the test removes the product from the user's wishlist by sending another **PUT request**. It confirms that the product is **successfully removed from the wishlist**.



## User Management Tests

Create a new test class named **UserManagementTests**.cs in the ApiTests project.

The UserManagementTests class, marked with [TestFixture], tests the User API using a RestClient for API requests. The Dispose() **method** cleans up the RestClient to prevent resource leaks. The Setup method initializes the RestClient and **authenticates the user before each test**, ensuring consistent and secure testing.

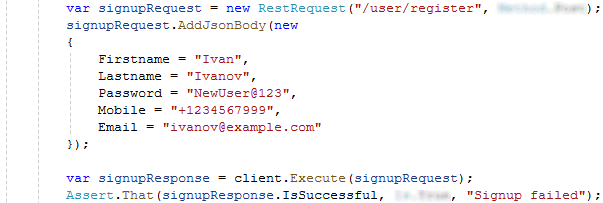


### User Login Test

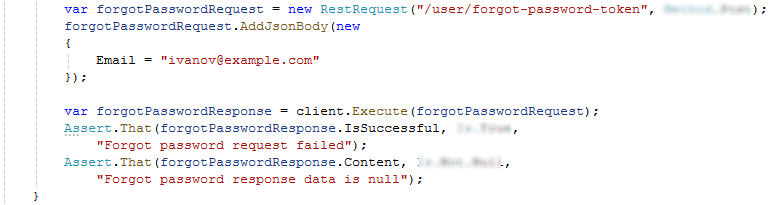
This method that verifies the functionality of **user-related API endpoints**, including **login**, **registration**, and **password reset**. It first logs in a user by sending a **POST request** **with** **credentials**, **checks** for a **successful response**, and **extracts a token**.



Then, it **registers a new user** by sending **POST request** with user details and ensures the **registration is successful**.



Finally, it tests the **password reset** functionality by sending a **POST** **request** to trigger a **password reset** and verifies that the **response** is **successful** and **contains valid data**.



### User Signup Login Update and Delete Test

This test method first tests user **registration** by **signing up a new user** and **checking the response** for **success** and **valid data**.



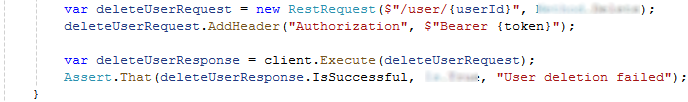
Then, it logs in with the **new user’s credentials**, **confirming successful authentication** and **extracting the token** and **user ID**.



With this token, it tests updating the **user's details** and verifies the **update is successful**.



Finally, the test deletes the user account, ensuring the **deletion process** is **completed successfully**, **validating** **each** **step** **of the user lifecycle**.

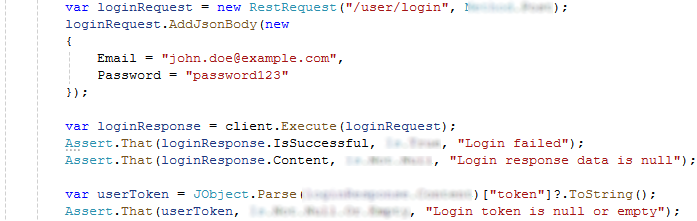


### Product And User Cart Test

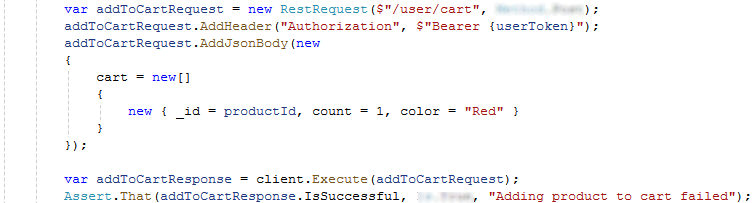
The ProductAndUserCartTest begins by **creating** a **new product** via a **POST request**, including product details such as **title**, **description**, **price**, **category**, and **quantity**. It verifies the product is **successfully** **created** by checking the response and extracting the product ID.



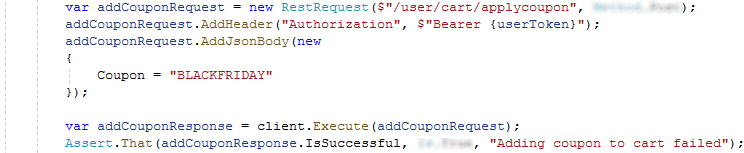
The test then **logs in a user** using a **POST request** to the **/user/login endpoint** with the user's **email** and **password**. It confirms the **login** is **successful** and retrieves the user **authentication token** from the **response**.



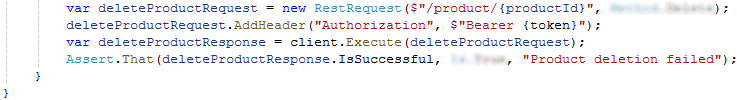
With the obtained **token**, the test adds the newly created product to the user's cart by sending a **POST request**. It checks that the product is **successfully added to the cart**.



The test **applies a coupon to the cart** by sending another **POST request**, using the **same token**. It verifies that the **coupon application** is **successful**.



Finally, the test deletes the created product by sending a **DELETE request** to the endpoint. It ensures the product **deletion process** is **successful**.

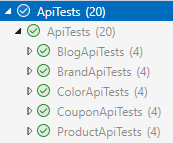


## Run the Tests

In Visual Studio, open the **Test Explorer** (Test 🡪 Windows 🡪 Test Explorer).

Build the solution to discover the tests.

Run all tests to ensure they pass:



Explore the API and add your own tests.

Enjoy ☺