Workshop: Basic API Functional Testing with NUnit and RestSharp

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:

Project Overview

Technologies Used:

- **Node.js**: A JavaScript runtime used for server-side development.
- **Express.js**: A web application framework for Node.js, used to build the API.
- **MongoDB**: A NoSQL database used to store data.
- **Mongoose**: An ODM (Object Data Modeling) library for MongoDB and Node.js.
- **bcrypt**: A library to hash passwords for security.
- **dotenv**: A module to load environment variables from a .env file.

Key Functionalities:

- **User Management**: Handles user registration, login, and role-based access control.
- **Product Management**: Manages product information, including CRUD operations.
- **Order Processing**: Manages the order lifecycle, including creation, updating, and deletion.
- **Cart Management**: Handles shopping cart operations for users.
- **Coupon Management**: Manages discount coupons for promotional purposes.
- **Enquiries**: Handles customer enquiries and their statuses.
- **Blog Management**: Manages blog posts and categories.

SoftUni
© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

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.

4. **Access** the API

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

5. API **Documentation**

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

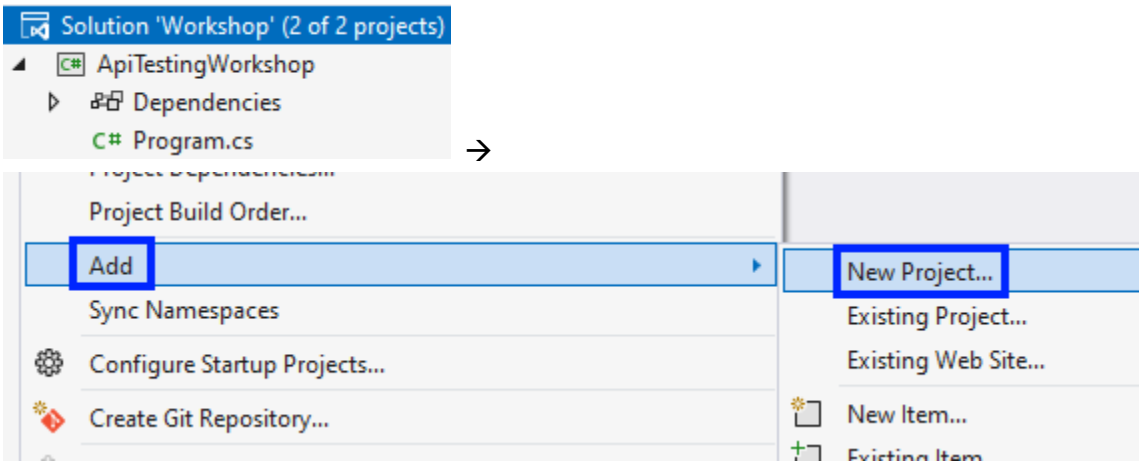
You will learn how to use **NUnit** and **RestSharp** to perform **basic functional testing** on API endpoints.

1. Setting Up the Project:

Create a New C# Project

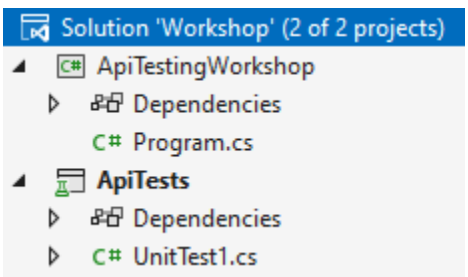
Open Visual Studio and create a new C# Console Application project named **`ApiTestingWorkshop`**.

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:



© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

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.

```
public static class GlobalConstants
{
    public const string BaseUrl = "http://localhost:5000/api";

    5 references
    public static string AuthenticateUser(string email, string password)
    {
        var authClient = new RestClient(BaseUrl);
        var request = new RestRequest("user/admin-login", Method.Post);
        request.AddJsonBody(new { email, password });

        var response = authClient.Execute(request);

        if (response.StatusCode != HttpStatusCode.OK)
        {
            Assert.Fail($"Authentication failed with status code: {response.StatusCode}, " +
                $"content: {response.Content}");
        }

        var content = JObject.Parse(response.Content);
        return content["token"]?.ToString();
    }
}
```

2. CRUD Operations for Products Testing

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

The **ProductApiTests** class is a test fixture, marked with the **[TestFixture]** attribute, designed to test the **Product API**. It includes a **RestClient** for making **API requests** and implements the **IDisposable** interface to

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

ensure proper cleanup of resources. The **`Dispose()` method** disposes of the **`RestClient`** instance to prevent resource leaks, maintaining a clean test environment.

```
[TestFixture]
0 references
public class ProductApiTests : IDisposable
{
    private RestClient client;
    private string token;

    0 references
    public void Dispose()
    {
        client?.Dispose();
    }
```

The **`Setup` method** is a pre-test initialization method that runs before each test. It creates a **`RestClient`** using the **base API URL** and authenticates the user by obtaining an **authentication token**. The method also includes an assertion to ensure that the **token is valid** (**not null or empty**), **setting up a secure and consistent environment for the tests**.

```
    [SetUp]
    0 references
    public void Setup()
    {
        client = new RestClient(GlobalConstants.BaseUrl);
        token = GlobalConstants.AuthenticateUser("admin@gmail.com", "admin@gmail.com");

        Assert.That(token, Is.Not.Null.Or.Empty, "Authentication token should not be null or empty");
    }
```

GET Request Testing:

The **`Test_GetAllProducts` method** is a unit test that verifies the functionality of the **API endpoint** responsible for retrieving **all products**. It sends a **GET request to the "product" endpoint**, confirms the response **status code** is **200 OK** and ensures the **response content is not empty**.

```
    [Test]
    0 references
    public void Test_GetAllProducts()
    {
        var request = new RestRequest("product", Method.Get);
        var response = client.Execute(request);

        Assert.Multiple(() =>
        {
            Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK));
            Assert.That(response.Content, Is.Not.Empty);
```

Verifies that specific product titles, such as "**Smartphone Alpha**" and "**Wireless Headphones**," **are included in the response**. Checks that the **prices** of these products match expected values. The test uses **`Assert.Multiple`** to **execute all assertions together**, ensuring comprehensive validation of the API's product retrieval feature.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
        var content = JArray.Parse(response.Content);

        var productTitles = new[]
        {
            "Smartphone Alpha", "Wireless Headphones", "Gaming Laptop",
            "4K Ultra HD TV", "Smartwatch Pro"
        };

        foreach (var title in productTitles)
        {
            Assert.That(content.ToString(), Does.Contain(title));
        }

        var expectedPrices = new Dictionary<string, decimal>
        {
            { "Smartphone Alpha", 999 },
            { "Wireless Headphones", 199 },
            { "Gaming Laptop", 1499 },
            { "4K Ultra HD TV", 899 },
            { "Smartwatch Pro", 299 }
        };

        foreach (var product in content)
        {
            var title = product["title"].ToString();
            if (expectedPrices.ContainsKey(title))
            {
                Assert.That(product["price"].Value<decimal>(),
                    Is.EqualTo(expectedPrices[title]));
            }
        }
    });
}
```

POST Request Testing:

The **`Test_AddProduct` method** is a unit test that validates the API's ability to add a new product. It sends a **POST request** to the "product" endpoint with the necessary product details, including title, slug, description, price, category, brand, and quantity. The request is authenticated using a Bearer token.Then **verifies the response**, ensuring that the **status code** is **200 OK** and the response content accurately reflects the data sent in the request.

```
[Test]
0 references
public void Test_AddProduct()
{
    var request = new RestRequest("product", Method.Post);
    request.AddHeader("Authorization", $"Bearer {token}");
    request.AddJsonBody(new
    {
        title = "New Test Product",
        slug = "new-test-product",
        description = "New Product Description",
        price = 99.99,
        category = "test",
        brand = "test",
        quantity = 100,
    });

    var response = client.Execute(request);

    Assert.Multiple(() =>
    {
        Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK));
        Assert.That(response.Content, Is.Not.Empty);
```


© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

After all it asserts multiple product attributes, such as title, price, and quantity, to confirm that the product was correctly added to the system. This test ensures that the API properly processes and stores new product data.

```
            var content = JObject.Parse(response.Content);

            Assert.That(content["title"].ToString(), Is.EqualTo("New Test Product"));
            Assert.That(content["slug"].ToString(), Is.EqualTo("new-test-product"));
            Assert.That(content["description"].ToString(), Is.EqualTo("New Product Description"));
            Assert.That(content["price"].Value<decimal>(), Is.EqualTo(99.99));
            Assert.That(content["category"].ToString(), Is.EqualTo("test"));
            Assert.That(content["brand"].ToString(), Is.EqualTo("test"));
            Assert.That(content["quantity"].Value<int>(), Is.EqualTo(100));
        });
    }
```

PUT Request Testing:

This test case, **`Test_UpdateProduct_InvalidProductId`**, is a **negative test** designed to validate the behavior of the API when an invalid or non-existent product ID is used to update a product. The test sends an HTTP PUT request with an invalid product ID and expects the API to respond with a **400 Bad Request** or **500 Internal Server Error**, depending on the server's error handling. Additionally, the test checks for an **appropriate error message**, such as "**This id is not valid or not Found**" in the response content, indicating that the provided product ID is invalid.

```
[Test]
0 references
public void Test_UpdateProduct_InvalidProductId()
{
    var invalidProductId = "invalidProductId12345";

    var updateRequest = new RestRequest("product/{id}", Method.Put);
    updateRequest.AddUrlSegment("id", invalidProductId);
    updateRequest.AddHeader("Authorization", $"Bearer {token}");
    updateRequest.AddJsonBody(new
    {
        title = "Invalid Product Update",
        description = "This should fail due to invalid product ID",
        price = 99.99,
        brand = "InvalidBrand",
        quantity = 10
    });

    var updateResponse = client.Execute(updateRequest);

    Assert.Multiple(() =>
    {
        Assert.That(updateResponse.StatusCode,
            Is.EqualTo(HttpStatusCode.InternalServerError).Or.EqualTo(HttpStatusCode.BadRequest),
            "Expected 404 Not Found or 500 Bad Request for invalid product ID");

        Assert.That(updateResponse.Content,
            Does.Contain("This id is not valid or not Found").Or.Contain("Invalid ID"),
            "Expected an error message indicating the product ID is invalid or not found");
    });
}
```

DELETE Request Testing:

The **`Test_DeleteProduct` method** is a unit test that validates the API's capability to **delete a specific product**. The test begins by **retrieving the list of products** and **searching for the product with the slug** "**electric-bike**". Once identified, it sends a **DELETE request** to **remove the product**, ensuring the request is **authenticated**. After confirming the deletion with a **200 OK status code**, the test performs an **additional check** by attempting to retrieve the deleted product to verify that it has indeed been removed from the database.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
[Test]
0 references
public void Test_DeleteProduct()
{
    var getRequest = new RestRequest("product", Method.Get);

    var getResponse = client.Execute(getRequest);

    Assert.That(getResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK));
    Assert.That(getResponse.Content, Is.Not.Empty);

    var products = JArray.Parse(getResponse.Content);
    var productToDelete = products.FirstOrDefault(p => p["slug"]?.ToString() == "electric-bike");

    Assert.That(productToDelete, Is.Not.Null, "Product with slug 'electric-bike' not found");

    var productId = productToDelete["_id"]?.ToString();

    var deleteRequest = new RestRequest("product/{id}", Method.Delete);
    deleteRequest.AddUrlSegment("id", productId);
    deleteRequest.AddHeader("Authorization", $"Bearer {token}");

    var deleteResponse = client.Execute(deleteRequest);

    Assert.That(deleteResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK));

    var verifyGetRequest = new RestRequest("product/{id}", Method.Get);
    verifyGetRequest.AddUrlSegment("id", productId);
    var verifyGetResponse = client.Execute(verifyGetRequest);

    Assert.That(verifyGetResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK));
}
```

3. CRUD Operations for Blogs Testing

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

The **`BlogApiTests`** class is a test fixture, marked with the **`[TestFixture]`** attribute, designed to test the **Blog API**. It includes a **`RestClient`** for making **API requests** and implements the **`IDisposable`** interface to **ensure proper cleanup of resources**. The **`Dispose()` method** disposes of the **`RestClient`** instance to prevent resource leaks, maintaining a clean test environment.

```
[TestFixture]
0 references
public class BlogApiTests : IDisposable
{
    private RestClient client;
    private string token;

    0 references
    public void Dispose()
    {
        client?.Dispose();
    }
```

The **Setup method** is a pre-test initialization method that runs before each test. It creates a **`RestClient`** using the **base API URL** and authenticates the user by obtaining an **authentication token**. The method also includes an assertion to ensure that the **token is valid** (**not null or empty**), **setting up a secure and consistent environment for the tests**.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
[SetUp]
0 references
public void Setup()
{
    client = new RestClient(GlobalConstants.BaseUrl);
    token = GlobalConstants.AuthenticateUser("admin@gmail.com", "admin@gmail.com");

    Assert.That(token, Is.Not.Null.Or.Empty, "Authentication token should not be null or empty");
}
```

GET Request Testing:

The **`Test_GetAllBlogs`** method verifies the functionality of the API endpoint responsible for retrieving **all blog** posts. It begins by sending a **GET request** to the "**blog**" endpoint and then performs several assertions to **ensure the response is valid**. The test checks that the response status code is **200 OK** and that the content **is not empty**. It further validates that the response content is a JSON array **containing at least one blog** post.

```
[Test]
| 0 references
public void Test_GetAllBlogs()
{
    var request = new RestRequest("blog", Method.Get);

    var response = client.Execute(request);

    Assert.Multiple(() =>
    {
        Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code OK (200)");
        Assert.That(response.Content, Is.Not.Empty, "Response content should not be empty");

        var blogs = JArray.Parse(response.Content);

        Assert.That(blogs.Type, Is.EqualTo(JTokenType.Array),
            "Expected response content to be a JSON array");
        Assert.That(blogs.Count, Is.GreaterThan(0), "Expected at least one blog in the response");
```

For each blog in the array, the test asserts that critical fields such as **title**, **description**, **author**, and **category are present and not empty**. This ensures that the API returns **complete and correctly formatted blog data**.

```
        foreach (var blog in blogs)
        {
            Assert.That(blog["title"]?.ToString(), Is.Not.Null.And.Not.Empty,
                "Blog title should not be null or empty");
            Assert.That(blog["description"]?.ToString(), Is.Not.Null.And.Not.Empty,
                "Blog description should not be null or empty");
            Assert.That(blog["author"]?.ToString(), Is.Not.Null.And.Not.Empty,
                "Blog author should not be null or empty");
            Assert.That(blog["category"]?.ToString(), Is.Not.Null.And.Not.Empty,
                "Blog category should not be null or empty");
        }
    });
}
```

POST Request Testing:

The **`Test_AddBlog` method** sends a **POST request** to the "**blog**" endpoint with the necessary blog details, including the **title**, **description**, and **category**, and uses a **Bearer token for authentication**. The test then checks that the response status code is **200 OK** and that the response **content is not empty**. It further asserts that the **returned blog data matches the input values** for the title, description, and category, and ensures that the author field is present and not empty.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
[Test]
0 references
public void Test_AddBlog()
{
    var request = new RestRequest("blog", Method.Post);
    request.AddHeader("Authorization", $"Bearer {token}");
    request.AddJsonBody(new
    {
        title = "New Blog",
        description = "New Blog Description",
        category = "Technology"
    });

    var response = client.Execute(request);

    Assert.Multiple(() =>
    {
        Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code Created (201)");
        Assert.That(response.Content, Is.Not.Empty, "Response content should not be empty");

        var content = JObject.Parse(response.Content);

        Assert.That(content["title"]?.ToString(), Is.EqualTo("New Blog"),
            "Blog title should match the input");
        Assert.That(content["description"]?.ToString(), Is.EqualTo("New Blog Description"),
            "Blog description should match the input");
        Assert.That(content["category"]?.ToString(), Is.EqualTo("Technology"),
            "Blog category should match the input");
        Assert.That(content["author"]?.ToString(), Is.Not.Null.And.Not.Empty,
            "Blog author should not be null or empty");
    });
}
```

PUT Request Testing:

The **`Test_UpdateBlog` method** first **retrieves all blogs** with a **GET request** and identifies the blog titled "**10 Tips for a Healthier Lifestyle**" for updating. This title is included in seeded data. After ensuring the blog exists, the test extracts its ID and sends a **PUT request to update** the blog's **title**, **description**, and **category**, using a **Bearer token for authentication**.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
[Test]
0 references
public void Test_UpdateBlog()
{
    var getRequest = new RestRequest("blog", Method.Get);

    var getResponse = client.Execute(getRequest);

    Assert.That(getResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK), "Failed to retrieve blogs");
    Assert.That(getResponse.Content, Is.Not.Empty, "Get blogs response content is empty");

    var blogs = JArray.Parse(getResponse.Content);
    var blogToUpdate = blogs.FirstOrDefault(b => b["title"]?.ToString() == "10 Tips for a Healthier Lifestyle");

    Assert.That(blogToUpdate, Is.Not.Null, "Blog with title '10 Tips for a Healthier Lifestyle' not found");

    var blogId = blogToUpdate["_id"].ToString();

    var updateRequest = new RestRequest("blog/{id}", Method.Put);
    updateRequest.AddHeader("Authorization", $"Bearer {token}");
    updateRequest.AddUrlSegment("id", blogId);
    updateRequest.AddJsonBody(new
    {
        title = "Updated Blog",
        description = "Updated Description",
        category = "Lifestyle"
    });

    var updateResponse = client.Execute(updateRequest);
```

The test then verifies that the response status code is **200 OK** and that the response content is not empty. It further **asserts** that the updated blog's title, description, and category **match the new values** provided, and that the author field remains valid.

```
    Assert.Multiple(() =>
    {
        Assert.That(updateResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code OK (200)");
        Assert.That(updateResponse.Content, Is.Not.Empty, "Update response content should not be empty");

        var content = JObject.Parse(updateResponse.Content);

        Assert.That(content["title"]?.ToString(), Is.EqualTo("Updated Blog"),
            "Blog title should match the updated value");
        Assert.That(content["description"]?.ToString(), Is.EqualTo("Updated Description"),
            "Blog description should match the updated value");
        Assert.That(content["category"]?.ToString(), Is.EqualTo("Lifestyle"),
            "Blog category should match the updated value");
        Assert.That(content["author"]?.ToString(), Is.Not.Null.And.Not.Empty,
            "Blog author should not be null or empty");

    });
}
```

DELETE Request Testing:

The **`Test_DeleteBlog` method** begins by sending a **GET request** to retrieve all blogs, then identifies the blog titled "**The Evolution of Entertainment in the Digital Age**" for deletion. After confirming the blog exists, the test extracts its ID and sends a **DELETE request**, using a **Bearer token for authentication**.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
[Test]
0 references
public void Test_DeleteBlog()
{
    var getRequest = new RestRequest("blog", Method.Get);
    var getResponse = client.Execute(getRequest);

    Assert.That(getResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK), "Failed to retrieve blogs");
    Assert.That(getResponse.Content, Is.Not.Empty, "Get blogs response content is empty");

    var blogs = JArray.Parse(getResponse.Content);
    var blogToDelete = blogs.FirstOrDefault(b => b["title"]?.ToString() ==
    "The Evolution of Entertainment in the Digital Age");

    Assert.That(blogToDelete, Is.Not.Null,
        "Blog with title 'The Evolution of Entertainment in the Digital Age' not found");

    var blogId = blogToDelete["_id"].ToString();

    var deleteRequest = new RestRequest("blog/{id}", Method.Delete);
    deleteRequest.AddHeader("Authorization", $"Bearer {token}");
    deleteRequest.AddUrlSegment("id", blogId);

    var deleteResponse = client.Execute(deleteRequest);
```

The test checks that the response **status code** is **200 OK**, indicating a **successful deletion**. The test attempts to retrieve the deleted blog again to **confirm that it has been removed**, ensuring the content is either **null or empty**.

```
    Assert.Multiple(() =>
    {
        Assert.That(deleteResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code Ok");

        var verifyGetRequest = new RestRequest("blog/{id}", Method.Get);
        verifyGetRequest.AddUrlSegment("id", blogId);

        var verifyGetResponse = client.Execute(verifyGetRequest);

        Assert.That(verifyGetResponse.Content, Is.Null.Or.EqualTo("null"),
            "Verify get response content should be empty");
    });
}
```

4. CRUD Operations for Brands Testing:

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**.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
[TestFixture]
0 references
public class BrandApiTests : IDisposable
{
    private RestClient client;
    private string token;

    0 references
    public void Dispose()
    {
        client?.Dispose();
    }

    [SetUp]
    0 references
    public void Setup()
    {
        client = new RestClient(GlobalConstants.BaseUrl);
        token = GlobalConstants.AuthenticateUser("admin@gmail.com", "admin@gmail.com");
        Assert.That(token, Is.Not.Null.Or.Empty, "Authentication token should not be null or empty");
    }
```

GET Request Testing:

This method sends a **GET request** to the "**brand**" endpoint and performs several assertions to **ensure the response** is valid. The test checks that the status code is **200 OK** and that the **response content is not empty**. It confirms that the response is a **JSON array containing at least one brand** and **verifies that specific brand names**, such as "**TechCorp**" and "**GameMaster**," are included in the list. Additionally, the test ensures that each brand has a **non-null and non-empty ID and title**. The test also expects that **more than five brands are returned**.

```
    [Test]
    0 references
    public void Test_GetAllBrands()
    {
        var request = new RestRequest("brand", Method.Get);

        var response = client.Execute(request);

        Assert.Multiple(() =>
        {
            Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK),
                "Expected status code OK (200)");
            Assert.That(response.Content, Is.Not.Empty, "Response content should not be empty");

            var brands = JArray.Parse(response.Content);
            Assert.That(brands.Type, Is.EqualTo(JTokenType.Array),
                "Expected response content to be a JSON array");
            Assert.That(brands.Count, Is.GreaterThan(0), "Expected at least one brand in the response");

            var firstBrand = brands.FirstOrDefault();
            Assert.That(firstBrand, Is.Not.Null, "Expected at least one brand in the response");

            var brandNames = brands.Select(b => b["title"]?.ToString()).ToList();
            Assert.That(brandNames, Does.Contain("TechCorp"), "Expected brand title 'TechCorp'");
            Assert.That(brandNames, Does.Contain("GameMaster"), "Expected brand title 'GameMaster'");

            foreach (var brand in brands)
            {
                Assert.That(brand["_id"]?.ToString(), Is.Not.Null.And.Not.Empty,
                    "Brand ID should not be null or empty");
                Assert.That(brand["title"]?.ToString(), Is.Not.Null.And.Not.Empty,
                    "Brand title should not be null or empty");
            }

            Assert.That(brands.Count, Is.GreaterThan(5), "Expected more than 5 brands in the response");
        });
    }
```


© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

POST Request Testing:

The **`Test_AddBrand` method** sends a **POST request** to the "**brand**" endpoint, including a JSON body with the brand title "**New Brand**" and an **authorization token in the header**.

```
[Test]
0 references
public void Test_AddBrand()
{
    var request = new RestRequest("brand", Method.Post);
    request.AddHeader("Authorization", $"Bearer {token}");
    request.AddJsonBody(new { title = "New Brand" });

    var response = client.Execute(request);
```

The test then checks that the **response status** code is **200 OK** and that the **response content is not empty**. It further asserts that the response contains a valid, **non-null brand ID** and that the **brand title** in the response **matches the input**.

```
    Assert.Multiple(() =>
    {
        Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code OK (200)");
        Assert.That(response.Content, Is.Not.Empty, "Response content should not be empty");

        var content = JObject.Parse(response.Content);

        Assert.That(content["_id"]?.ToString(), Is.Not.Null.And.Not.Empty,
            "Brand ID should not be null or empty");
        Assert.That(content["title"]?.ToString(), Is.EqualTo("New Brand"),
            "Brand title should match the input");
    });
}
```

PUT Request Testing:

The next method begins by sending a **GET request** to **retrieve all brands**, then identifies the brand titled "**GameMaster**" for **updating**. After confirming the brand exists, the test extracts its ID and **sends a PUT request** to update the brand's title to "**Updated Brand**," using a **Bearer token for authentication**.

```
[Test]
0 references
public void Test_UpdateBrand()
{
    var getRequest = new RestRequest("brand", Method.Get);

    var getResponse = client.Execute(getRequest);

    Assert.That(getResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
        "Failed to retrieve brands");
    Assert.That(getResponse.Content, Is.Not.Empty, "Get brands response content is empty");

    var brands = JArray.Parse(getResponse.Content);
    var brandToUpdate = brands.FirstOrDefault(b => b["title"]?.ToString() == "GameMaster");

    Assert.That(brandToUpdate, Is.Not.Null, "Brand with title 'GameMaster' not found");

    var brandId = brandToUpdate["_id"]?.ToString();

    var updateRequest = new RestRequest("brand/{id}", Method.Put);
    updateRequest.AddHeader("Authorization", $"Bearer {token}");
    updateRequest.AddUrlSegment("id", brandId);
    updateRequest.AddJsonBody(new { title = "Updated Brand" });

    var updateResponse = client.Execute(updateRequest);
```


© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

The test checks that the **response status code** is **200 OK** and that the **response content is not empty**. It also verifies that the brand ID in the response matches the original ID, the title has been **updated correctly**, and that both **`createdAt`** and **`updatedAt`** fields are present, with the **`updatedAt`** value reflecting the **recent update**.

```
        Assert.Multiple(() =>
        {
            Assert.That(updateResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
                "Expected status code OK (200)");
            Assert.That(updateResponse.Content, Is.Not.Empty, "Response content should not be empty");

            var content = JObject.Parse(updateResponse.Content);

            Assert.That(content["_id"]?.ToString(), Is.EqualTo(brandId),
                "Brand ID should match the updated brand's ID");
            Assert.That(content["title"]?.ToString(), Is.EqualTo("Updated Brand"),
                "Brand title should be updated correctly");

            Assert.That(content.ContainsKey("createdAt"), Is.True, "Brand should have a createdAt field");
            Assert.That(content.ContainsKey("updatedAt"), Is.True, "Brand should have an updatedAt field");

            Assert.That(content["updatedAt"]?.ToString(), Is.Not.EqualTo(content["createdAt"]?.ToString()),
                "updatedAt should be different from createdAt after an update");
        });
    }
```

DELETE Request Testing:

The **`Test_DeleteBrand`** method begins by sending a **GET request** to **retrieve all brands** and identifies the brand titled "**`ViewTech`**" for deletion. After confirming that the brand exists, the test extracts its ID and sends a **DELETE request** to **remove the brand**, using a **Bearer token for authentication**. The test checks that the deletion response status code is **200 OK**, indicating the brand was **successfully deleted**.

```
    [Test]
    0 references
    public void Test_DeleteBrand()
    {
        var getRequest = new RestRequest("brand", Method.Get);

        var getResponse = client.Execute(getRequest);

        Assert.That(getResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Failed to retrieve brands");
        Assert.That(getResponse.Content, Is.Not.Empty, "Get brands response content is empty");

        var brands = JArray.Parse(getResponse.Content);
        var brandToDelete = brands.FirstOrDefault(b => b["title"]?.ToString() == "ViewTech");

        Assert.That(brandToDelete, Is.Not.Null, "Brand with title 'ViewTech' not found");

        var brandId = brandToDelete["_id"]?.ToString();

        var deleteRequest = new RestRequest("brand/{id}", Method.Delete);
        deleteRequest.AddHeader("Authorization", $"Bearer {token}");
        deleteRequest.AddUrlSegment("id", brandId);

        var deleteResponse = client.Execute(deleteRequest);
```

Finally, the test attempts to retrieve the **deleted brand to verify that it is no longer available**, ensuring the response content is **empty** or "**null**".

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
    Assert.Multiple(() =>
    {
        Assert.That(deleteResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code NoContent (204) after deletion");

        var verifyGetRequest = new RestRequest("brand/{id}", Method.Get);
        verifyGetRequest.AddUrlSegment("id", brandId);
        var verifyGetResponse = client.Execute(verifyGetRequest);

        Assert.That(verifyGetResponse.Content, Is.Empty.Or.EqualTo("null"),
            "Verify get response content should be empty or 'null'");
    });
}
```

5. CRUD Operations for Colors Testing:

Create a new test class named **`ColorApiTests.cs`** in the **`ApiTests`** project. This class is a test fixture designed to test the Color 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**.

```
[TestFixture]
0 references
public class ColorApiTests : IDisposable
{
    private RestClient client;
    private string token;

    [SetUp]
    0 references
    public void Setup()
    {
        client = new RestClient(GlobalConstants.BaseUrl);
        token = GlobalConstants.AuthenticateUser("admin@gmail.com", "admin@gmail.com");
        Assert.That(token, Is.Not.Null.Or.Empty, "Authentication token should not be null or empty");
    }
```

The **`Order` attribute** in this test class is being to specify the sequence in which the tests should be executed. Normally, **NUnit** (and most test frameworks) run tests in a random or **non-deterministic order**, since tests should ideally be independent of one another. However, in this case, we are going to use the [**Order**] attribute to explicitly **set the order of the tests**, which suggests that the tests may have some **dependencies** on one another, or that the order of execution is significant to the business logic being tested.

GET Request Testing:

The **`Test_GetAllColors` method** is a unit test that verifies the API's ability to **retrieve all colors**. It sends **a GET request** to the "**color**" endpoint and performs several assertions to ensure the **response is correct**. The test checks that the **response status code** is **200 OK** and that the response **content is not empty**.

```
[Test, Order(1)]
0 references
public void Test_GetAllColors()
{
    var request = new RestRequest("color", Method.Get);

    var response = client.Execute(request);

    Assert.Multiple(() =>
    {
        Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code OK (200)");
        Assert.That(response.Content, Is.Not.Empty, "Response content should not be empty");
```


© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

It confirms that the content is a **JSON array** containing **at least one color** and specifically checks for the presence of common colors like "**Black**," "**White**," and "**Red**." The test also verifies that **each color has a valid**, **non-null ID and title**. Finally, it asserts that exactly **10 colors** are returned in the response, ensuring the completeness and accuracy of the API's color retrieval functionality.

```
        var colors = JArray.Parse(response.Content);

        Assert.That(colors.Type, Is.EqualTo(JTokenType.Array),
            "Expected response content to be a JSON array");
        Assert.That(colors.Count, Is.GreaterThan(0), "Expected at least one color in the response");

        var colorTitles = colors.Select(c => c["title"]?.ToString()).ToList();
        Assert.That(colorTitles, Does.Contain("Black"), "Expected color 'Black'");
        Assert.That(colorTitles, Does.Contain("White"), "Expected color 'White'");
        Assert.That(colorTitles, Does.Contain("Red"), "Expected color 'Red'");

        foreach (var color in colors)
        {
            Assert.That(color["_id"]?.ToString(), Is.Not.Null.And.Not.Empty,
                "Color ID should not be null or empty");
            Assert.That(color["title"]?.ToString(), Is.Not.Null.And.Not.Empty,
                "Color title should not be null or empty");
        }

        Assert.That(colors.Count, Is.EqualTo(10), "Expected exactly 10 colors in the response");
    });
}
```

POST Request Testing:

The next method sends a **POST request** to the "**color**" endpoint with a JSON body containing the **new color title**, "**New Color**" and includes an **authorization token in the header**. The test checks that the response status code is **200 OK** and that the response content is not empty.

```
[Test, Order(2)]
0 references
public void Test_AddColor()
{
    var request = new RestRequest("color", Method.Post);
    request.AddHeader("Authorization", $"Bearer {token}");
    request.AddJsonBody(new { title = "New Color" });

    var response = client.Execute(request);

    Assert.Multiple(() =>
    {
        Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code OK (200)");
        Assert.That(response.Content, Is.Not.Empty, "Response content should not be empty");
```

It then parses the response to ensure the newly created color has a **valid**, **non-null ID** and that the **title matches the input**. The test also verifies that the response includes both **`createdAt`** and **`updatedAt`** fields, confirming they are in a valid date-time format and that these timestamps are **identical upon creation**.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
        var content = JObject.Parse(response.Content);

        Assert.That(content["_id"]?.ToString(), Is.Not.Null.And.Not.Empty,
            "Color ID should not be null or empty");
        Assert.That(content["title"]?.ToString(), Is.EqualTo("New Color"),
            "Color title should match the input");

        Assert.That(content.ContainsKey("createdAt"), Is.True, "Color should have a createdAt field");
        Assert.That(content.ContainsKey("updatedAt"), Is.True, "Color should have an updatedAt field");

        Assert.That(DateTime.TryParse(content["createdAt"]?.ToString(), out _), Is.True,
            "createdAt should be a valid date-time format");
        Assert.That(DateTime.TryParse(content["updatedAt"]?.ToString(), out _), Is.True,
            "updatedAt should be a valid date-time format");

        Assert.That(content["createdAt"]?.ToString(), Is.EqualTo(content["updatedAt"]?.ToString()),
            "createdAt and updatedAt should be the same on creation");
    });
}
```

PUT Request Testing:

The test begins by sending a **GET request** to **retrieve all available colors** and identifies the color titled "**Red**" for **updating**. After confirming that the color exists, the test extracts its ID and **sends a PUT request to update the color's title** to "**Updated Red**" using a **Bearer token for authentication**.

```
[Test, Order(3)]
0 references
public void Test_UpdateColor()
{
    var getRequest = new RestRequest("color", Method.Get);

    var getResponse = client.Execute(getRequest);

    Assert.That(getResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
        "Failed to retrieve colors");
    Assert.That(getResponse.Content, Is.Not.Empty, "Get colors response content is empty");

    var colors = JArray.Parse(getResponse.Content);
    var colorToUpdate = colors.FirstOrDefault(c => c["title"]?.ToString() == "Red");

    Assert.That(colorToUpdate, Is.Not.Null, "Color with title 'Red' not found");

    var colorId = colorToUpdate["_id"]?.ToString();

    var updateRequest = new RestRequest("color/{id}", Method.Put);
    updateRequest.AddHeader("Authorization", $"Bearer {token}");
    updateRequest.AddUrlSegment("id", colorId);
    updateRequest.AddJsonBody(new { title = "Updated Red" });

    var updateResponse = client.Execute(updateRequest);
```

The test checks that the **response status code** is **200 OK** and that the **response content** is **not empty**. It further verifies that the color's ID remains the same, the title has been **updated correctly**, and both **`createdAt`** and **`updatedAt`** fields are present and in valid date-time format. The test also ensures that the **`updatedAt`** timestamp reflects the **recent update**, differing from the **`createdAt`** timestamp.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
    Assert.Multiple(() =>
    {
        Assert.That(updateResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code OK (200)");
        Assert.That(updateResponse.Content, Is.Not.Empty, "Response content should not be empty");

        var content = JObject.Parse(updateResponse.Content);

        Assert.That(content["_id"]?.ToString(), Is.EqualTo(colorId),
            "Color ID should match the updated color's ID");
        Assert.That(content["title"]?.ToString(), Is.EqualTo("Updated Red"),
            "Color title should be updated correctly");
        Assert.That(content.ContainsKey("createdAt"), Is.True, "Color should have a createdAt field");
        Assert.That(content.ContainsKey("updatedAt"), Is.True, "Color should have an updatedAt field");
        Assert.That(DateTime.TryParse(content["createdAt"]?.ToString(), out _), Is.True,
            "createdAt should be a valid date-time format");
        Assert.That(DateTime.TryParse(content["updatedAt"]?.ToString(), out _), Is.True,
            "updatedAt should be a valid date-time format");
        Assert.That(content["updatedAt"]?.ToString(), Is.Not.EqualTo(content["createdAt"]?.ToString()),
            "updatedAt should be different from createdAt after an update");
    });
}
```

DELETE Request Testing:

The **`Test_DeleteColor` method** begins by sending a **GET request** to retrieve the list of colors and identifies the color titled "**Black**" for deletion. After confirming that the color **exists**, the test extracts its ID and **sends** a **DELETE request**, using a **Bearer token for authentication**.

```
[Test, Order(4)]
0 references
public void Test_DeleteColor()
{
    var getRequest = new RestRequest("color", Method.Get);

    var getResponse = client.Execute(getRequest);

    Assert.That(getResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
        "Failed to retrieve colors");
    Assert.That(getResponse.Content, Is.Not.Empty, "Get colors response content is empty");

    var colors = JArray.Parse(getResponse.Content);
    var colorToDelete = colors.FirstOrDefault(c => c["title"]?.ToString() == "Black");

    Assert.That(colorToDelete, Is.Not.Null, "Color with title 'Black' not found");

    var colorId = colorToDelete["_id"]?.ToString();

    var deleteRequest = new RestRequest("color/{id}", Method.Delete);
    deleteRequest.AddHeader("Authorization", $"Bearer {token}");
    deleteRequest.AddUrlSegment("id", colorId);

    var deleteResponse = client.Execute(deleteRequest);
```

The test checks that the response **status code** is **200 OK**, indicating the **deletion was successful**. To ensure the color has been properly deleted, the test **sends** a **GET request for the deleted color by ID**, verifying that the response content is **empty** or "**null**". Finally, the test retrieves the list of colors again to confirm that "**Black**" **no longer exists in the list**.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
        Assert.Multiple(() =>
        {
            Assert.That(deleteResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
                "Expected status code OK (200) after deletion");

            var verifyGetRequest = new RestRequest("color/{id}", Method.Get);
            verifyGetRequest.AddUrlSegment("id", colorId);
            var verifyGetResponse = client.Execute(verifyGetRequest);

            Assert.That(verifyGetResponse.Content, Is.Empty.Or.EqualTo("null"),
                "Verify get response content should be empty or 'null'");

            var refreshedGetResponse = client.Execute(getRequest);
            var refreshedColors = JArray.Parse(refreshedGetResponse.Content);
            var colorExists = refreshedColors.Any(c => c["title"]?.ToString() == "Black");

            Assert.That(colorExists, Is.False,
                "Color with title 'Black' should no longer exist in the list");
        });
    }
```

6. CRUD Operations for Coupons Testing:

Create a new test class named **CouponApiTests.cs** in the **ApiTests** project. This class is a test fixture designed to test the Coupon 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**.

```
[TestFixture]
0 references
public class CouponApiTests : IDisposable
{
    private RestClient client;
    private string token;

    0 references
    public void Dispose()
    {
        client?.Dispose();
    }

    [SetUp]
    0 references
    public void Setup()
    {
        client = new RestClient(GlobalConstants.BaseUrl);
        token = GlobalConstants.AuthenticateUser("admin@gmail.com", "admin@gmail.com");
        Assert.That(token, Is.Not.Null.Or.Empty, "Authentication token should not be null or empty");
    }
```

GET Request Testing:

The **Test_GetAllCoupons method** sends a **GET request** to the "**coupon**" **endpoint**, including an **authorization token in the header**, and performs several assertions to ensure the response is correct. The test checks that the **status code** is **200 OK** and that the **response content** is **not empty**. It confirms that the content is a **JSON array** containing at least one coupon and specifically checks for the presence of expected coupon names like "**SUMMER21**", "**WINTER21**", and "**BLACKFRIDAY**".

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
[Test]
0 references
public void Test_GetAllCoupons()
{
    var request = new RestRequest("coupon", Method.Get);
    request.AddHeader("Authorization", $"Bearer {token}");

    var response = client.Execute(request);

    Assert.Multiple(() =>
    {
        Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code OK (200)");
        Assert.That(response.Content, Is.Not.Empty, "Response content should not be empty");

        var coupons = JArray.Parse(response.Content);

        Assert.That(coupons.Type, Is.EqualTo(JTokenType.Array),
            "Expected response content to be a JSON array");
        Assert.That(coupons.Count, Is.GreaterThan(0),
            "Expected at least one coupon in the response");

        var couponNames = coupons.Select(c => c["name"]?.ToString()).ToList();
        Assert.That(couponNames, Does.Contain("SUMMER21"), "Expected coupon 'SUMMER21'");
        Assert.That(couponNames, Does.Contain("WINTER21"), "Expected coupon 'WINTER21'");
        Assert.That(couponNames, Does.Contain("BLACKFRIDAY"), "Expected coupon 'BLACKFRIDAY'");
```

The test further verifies that **each coupon has a valid**, **non-null ID**, **name**, and **expiry date**, and that the **discount value** is a **positive integer**. Additionally, the test ensures that each coupon's expiry date is in a valid date-time format and is **set to a future date**. This comprehensive test ensures that the API correctly **retrieves** and **returns detailed coupon information**.

```
        foreach (var coupon in coupons)
        {
            Assert.That(coupon["_id"]?.ToString(), Is.Not.Null.And.Not.Empty,
                "Coupon ID should not be null or empty");
            Assert.That(coupon["name"]?.ToString(), Is.Not.Null.And.Not.Empty,
                "Coupon name should not be null or empty");
            Assert.That(coupon["expiry"]?.ToString(), Is.Not.Null.And.Not.Empty,
                "Coupon expiry date should not be null or empty");
            int discountValue = coupon["discount"].Value<int>();
            Assert.That(discountValue, Is.GreaterThan(0),
                "Coupon discount should be a positive integer");
        }

        foreach (var coupon in coupons)
        {
            Assert.That(DateTime.TryParse(coupon["expiry"]?.ToString(), out var expiryDate),
                Is.True, "Coupon expiry should be a valid date-time format");
            Assert.That(expiryDate, Is.GreaterThan(DateTime.Now),
                "Coupon expiry should be in the future");
        }
    });
}
```

POST Request Testing:

The test sends a **POST request** to the "**coupon**" **endpoint** with a **JSON body** containing the coupon's **name**, **discount value**, and **expiry date**. An **authorization token** is included in the **request header** to ensure proper authentication. The test then checks that the response status code is **200 OK** and that the **response content** is **not empty**. It parses the response to confirm that the newly created coupon has a **valid**, **non-null ID**, and that the **coupon's name**, **discount**, and **expiry date match** the **input values**. Additionally, the test verifies that the **expiry date** is in a **valid date-time** format and set in the future.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

```
[Test]
0 references
public void Test_AddCoupon()
{
    var request = new RestRequest("coupon", Method.Post);
    request.AddHeader("Authorization", $"Bearer {token}");
    request.AddJsonBody(new { name = "New Coupon", discount = 20, expiry = "2026-12-31" });

    var response = client.Execute(request);

    Assert.Multiple(() =>
    {
        Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Expected status code OK (200)");
        Assert.That(response.Content, Is.Not.Empty, "Response content should not be empty");

        var content = JObject.Parse(response.Content);

        Assert.That(content["_id"]?.ToString(), Is.Not.Null.And.Not.Empty,
            "Coupon ID should not be null or empty");
        Assert.That(content["name"]?.ToString(), Is.EqualTo("NEW COUPON"),
            "Coupon name should match the input");
        Assert.That(content["discount"]?.Value<int>(), Is.EqualTo(20),
            "Coupon discount should match the input");
        Assert.That(content["expiry"]?.ToString(), Is.EqualTo("12/31/2026 12:00:00 AM"),
            "Coupon expiry should match the input");

        Assert.That(DateTime.TryParse(content["expiry"]?.ToString(), out var expiryDate),
            Is.True, "Expiry should be a valid date-time format");
        Assert.That(expiryDate, Is.GreaterThan(DateTime.Now),
            "Coupon expiry date should be in the future");
    });
}
```

PUT Request Testing:

The **`Test_UpdateCoupon`** method begins by sending a **GET request** to **retrieve all coupons**, including an **authorization token** in the header for authentication. It then identifies the coupon titled "**FLASHSALE**" for updating. After confirming that the coupon exists, the test extracts its ID and sends a **PUT request to update** the coupon's **name**, **discount value**, and **expiry date**.

```
[Test]
0 references
public void Test_UpdateCoupon()
{
    var getRequest = new RestRequest("coupon", Method.Get);
    getRequest.AddHeader("Authorization", $"Bearer {token}");

    var getResponse = client.Execute(getRequest);

    Assert.That(getResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
        "Failed to retrieve coupons");
    Assert.That(getResponse.Content, Is.Not.Empty, "Get coupons response content is empty");

    var coupons = JArray.Parse(getResponse.Content);
    var couponToUpdate = coupons.FirstOrDefault(c => c["name"]?.ToString() == "FLASHSALE");

    Assert.That(couponToUpdate, Is.Not.Null, "Coupon with name 'FLASHSALE' not found");

    var couponId = couponToUpdate["_id"]?.ToString();

    var updateRequest = new RestRequest("coupon/{id}", Method.Put);
    updateRequest.AddHeader("Authorization", $"Bearer {token}");
    updateRequest.AddUrlSegment("id", couponId);
    updateRequest.AddJsonBody(new { name = "Updated Coupon", discount = 25, expiry = "2026-12-31" });

    var updateResponse = client.Execute(updateRequest);
```


© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

The test checks that the response status code is **200 OK** and that the response content is **not empty**. It verifies that the coupon's **ID remains unchanged**, ensuring that the update did not inadvertently create a new coupon. The test also confirms that the updated fields—**name**, **discount**, and **expiry date**—**match** the **new values** provided. Additionally, it ensures that the **expiry date** is in a **valid date-time format** and **set to a future date**.

```
        Assert.Multiple(() =>
        {
            Assert.That(updateResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
                "Expected status code OK (200) after update");
            Assert.That(updateResponse.Content, Is.Not.Empty,
                "Response content should not be empty after update");

            var updatedContent = JObject.Parse(updateResponse.Content);

            Assert.That(updatedContent["_id"]?.ToString(), Is.EqualTo(couponId),
                "Coupon ID should not change after update");
            Assert.That(updatedContent["name"]?.ToString(), Is.EqualTo("UPDATED COUPON"),
                "Coupon name should be updated");
            Assert.That(updatedContent["discount"]?.Value<int>(), Is.EqualTo(25),
                "Coupon discount should be updated");
            Assert.That(updatedContent["expiry"]?.ToString(), Is.EqualTo("12/31/2026 12:00:00 AM"),
                "Coupon expiry should be updated");

            Assert.That(DateTime.TryParse(updatedContent["expiry"]?.ToString(), out var expiryDate),
                Is.True, "Expiry should be a valid date-time format");
            Assert.That(expiryDate, Is.GreaterThan(DateTime.Now),
                "Coupon expiry date should be in the future");
        });
    }
```

DELETE Request Testing:

The test starts by sending a **GET request** to retrieve the list of coupons, using a **Bearer token for authentication**. It identifies the coupon titled "**SPRING21**" for deletion. After confirming that the **coupon exists**, the test extracts its ID and **sends** a **DELETE request to remove the coupon**.

```
    [Test]
    0 references
    public void Test_DeleteCoupon()
    {
        var getRequest = new RestRequest("coupon", Method.Get);
        getRequest.AddHeader("Authorization", $"Bearer {token}");

        var getResponse = client.Execute(getRequest);

        Assert.That(getResponse.StatusCode, Is.EqualTo(HttpStatusCode.OK),
            "Failed to retrieve coupons");
        Assert.That(getResponse.Content, Is.Not.Empty, "Get coupons response content is empty");

        var coupons = JArray.Parse(getResponse.Content);
        var couponToDelete = coupons.FirstOrDefault(c => c["name"]?.ToString() == "SPRING21");

        Assert.That(couponToDelete, Is.Not.Null, "Coupon with name 'SPRING21' not found");

        var couponId = couponToDelete["_id"]?.ToString();

        var deleteRequest = new RestRequest("coupon/{id}", Method.Delete);
        deleteRequest.AddHeader("Authorization", $"Bearer {token}");
        deleteRequest.AddUrlSegment("id", couponId);

        var deleteResponse = client.Execute(deleteRequest);
```


© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Following the deletion, the test performs **additional checks** to ensure that the coupon has been **successfully removed**. It sends a **GET request** for the specific coupon by its ID and also **retrieves the list of all coupons again**. The test then verifies that the **deleted coupon is no longer present in the list**, confirming that the **deletion** was **effective**.

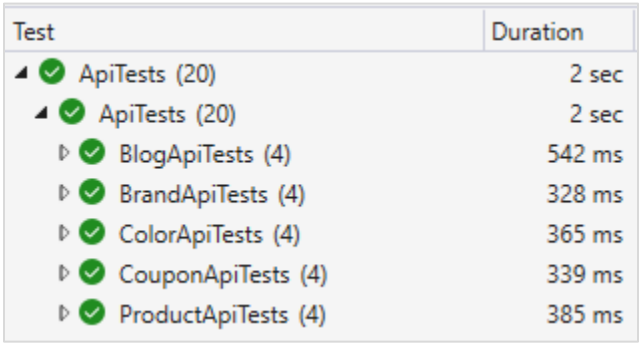
```
        Assert.Multiple(() =>
        {
            var verifyGetRequest = new RestRequest("coupon/{id}", Method.Get);
            verifyGetRequest.AddHeader("Authorization", $"Bearer {token}");
            verifyGetRequest.AddUrlSegment("id", couponId);

            var verifyGetResponse = client.Execute(verifyGetRequest);

            var verifyListResponse = client.Execute(getRequest);

            var updatedCoupons = JArray.Parse(verifyListResponse.Content);
            var deletedCoupon = updatedCoupons
            .FirstOrDefault(c => c["_id"]?.ToString() == couponId);

            Assert.That(deletedCoupon, Is.Null,
                "Deleted coupon should not be present in the list of coupons");
        });
    }
}
```

7. 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 ☺

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.