CIS 3374-5274 – Quality Assurance and Testing
Lab 8

Assignment: Create an API and its unit tests

Assigned Date:	March 10, 2022
Due Date:	March 15, 2022 by the end of the lab

Overview
This assignment will show you how to create your own simple API along with some simple unit tests to test your API.

Instructions

1. Go to the following tutorial: https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api to see how to set up and create your own API. Note that when selecting the template for your API, please check the boxes for Web API and Add Unit Tests.
   a. **PLEASE NOTE: For some users there might be additional required dependencies in Visual Studio for you to create and access the API Templates required for you to begin and complete this lab. Please reach out to me if you have any questions, 484-855-5518.**
   b. You can use any version of Microsoft Visual Studio.
   c. Please follow all instructions in the tutorial.
   d. Copy and paste the code as the tutorial states, as the goal is to have a functioning API with no errors.
   e. Please stop following the tutorial at the section: Calling the Web API with JavaScript and jQuery.
   f. Make the following change to the code that was copied and pasted.
      i. Go to ProductsController.cs.
      ii. Under the array of products, add the following code:

```
public ProductsController()
    {

    }

public ProductsController(Product[] products)
    {
        this.products = products;
    }
```

      iii. This section of code is needed to add Unit Tests in the next section of this lab and will allow us to load our Controller that we will create in our Unit Tests with our own Products.
2. Once you have made your API for the first time, you would normally start an API testing tool like Postman, start your API locally and send a request to it through Postman to verify it behaves as expected. Here, if you followed the tutorial correctly, we can assume that your API functions as it should.
3. Next, you'll create unit tests for the Controller methods that you just created. Go to https://docs.microsoft.com/en-us/aspnet/web-api/overview/testing-and-debugging/unit-testing-with-aspnet-web-api, to learn how to add unit tests to your code.

a. If you did not check the Add Unit Tests checkbox when first creating your project, please refer to the Create application with unit test project heading and then look at the Add unit test project to existing application section on how to do it after initial project creation.
b. Otherwise, you can skip the Create application with unit test project section.
c. Skip the Set up the Web API 2 application section.
d. Follow the instructions in the Install Nuget packages in test project section.
e. Follow the instructions under the Create tests section.
f. Once you've copied and pasted the code as instructed, we need to make some modifications to fix the errors.
g. Locate the lines: using StoreApp.Controllers; and using StoreApp.Models; and replace StoreApp with the name of your API, which should be ProductsApp if you named it according to the tutorial; then change the namespace from StoreApp.Tests to ProductsApp.Tests or whatever you named your API.
h. Remove the following Test Methods (as they have not been created):
    i. public async Task GetAllProductsAsync_ShouldReturnAllProducts()
    ii. public async Task GetProductAsync_ShouldReturnCorrectProduct()
i. Modify the method at the bottom called GetTestProducts() because this originally was set up to work with a List of Products but our API is set up to use an array of Products.
    i. Replace the entire method GetTestProducts with the following:

```
private Product[] GetTestProducts()
{
    var testProducts = new Product[]
    {
        new Product { Id = 1, Name = "Demo1", Price = 1 },
        new Product { Id = 2, Name = "Demo2", Price = 3.75M },
        new Product { Id = 3, Name = "Demo3", Price = 16.99M },
        new Product { Id = 4, Name = "Demo4", Price = 11.00M }
    };
    return testProducts;
 }
```

j. Please make the following adjustments to the Test Methods:
    i. public void GetAllProducts_ShouldReturnAllProducts()

```
{
        var testProducts = GetTestProducts();
        var controller = new ProductsController(testProducts);

        var result = controller.GetAllProducts() as Product[];
        Assert.AreEqual(testProducts.Length, result.Length);
 }
```

    ii. public void GetProduct_ShouldReturnCorrectProduct()

```
{
        var testProducts = GetTestProducts();
        var controller = new ProductsController(testProducts);

        var result = controller.GetProduct(4) as
OkNegotiatedContentResult<Product>;
        Assert.IsNotNull(result);
        Assert.AreEqual(testProducts[3].Name, result.Content.Name);
 }
```

    iii. public void GetProduct_ShouldNotFindProduct()

```
{
```

```
                var controller = new ProductsController(GetTestProducts());

                var result = controller.GetProduct(999);
                Assert.IsInstanceOfType(result, typeof(NotFoundResult));
            }
```

   k. These adjustments change the name of the Controller we are referencing and updating the .Count that was being used previously on result and testProducts to .Length since it's compatible with arrays.

4. Run your tests. Refer to the top of the application window and look for the Tests dropdown. Click it and select Run, then All Tests. A side window will open and start to run your tests. If they all pass then you are done with the creation of the API for this lab,
5. The next and final step in this lab is to upload your solution to GitHub. Follow these steps if you don't know how to upload content to GitHub:
   a. Go to github.com and sign in. If you don't have an account create one with your temple.edu email address.
   b. After signing in, locate the section of the site that says Repositories on the left hand side. Click New.
   c. Name your Repository ProductAPIRepo.
   d. Make the Repository **PUBLIC**.
      i. **Please do not make the repository private, this will result in a lost point.**
   e. Press Create Repository.
   f. You will now have a new repository created and it will present you with a setup page.
   g. Download Git Bash to add these files to your newly created repo or if you already know how to add a solution/project to a Git repo then you do not need to follow these instructions. You can download it here: https://git-scm.com/download/win
   h. This download comes loaded with Git Bash, which is a command line interface to work with git repositories.
   i. Launch Git Bash.
   j. Create a folder for your repository to be stored, e.g., a folder named git is appropriate.
   k. Open Git Bash and navigate to the folder you just created and issue the following command: git clone {your GitHub repo URL} (it can be found on the quick setup page after you create your new repository). You might have to add your credentials to get that command to work.
   l. Once your empty repo has been cloned to that folder, change the directory of Git Bash to that folder, e.g., "cd ProductsApp".
   m. In another File Explorer copy and paste the entire ProductsApp folder (mine was found at C:\Users\{my_user}\source\repos\) to the ProductAPIRepo folder which was created when you cloned your repository.
   n. In Git Bash, issue this command: git status, and you should see a red line as a response with the name of the folder you just added. This means git doesn't recognize this folder and we should add it.
   o. To add it issue this command: git add *. This will add everything included in that folder structure and will result in a long list of green text.
   p. Next issue this command: git commit -m "initial commit", which creates a local commit and attaches a commit message to it.
   q. Finally, to get this code local to your remote git repository, issue this command: git push, which makes all files added in the commit available to anyone with access to the repository.

6. Now that your API solution containing your API project and Unit Test project have been uploaded to GitHub, this lab is finished.

Submission: Submit the link to your GitHub repo in Canvas.