First Assignment

**Part 1: API Testing in C#**

**1. Setting Up the Project:**

**To get started with API testing in C#:**

1. **Create a new project in Visual Studio:**
   * **Choose a Unit Test project type (NUnit, MSTest, or XUnit based on preference).**
2. **Install required packages:**
   * **Use RestSharp or HttpClient to handle HTTP requests, and Newtonsoft.Json for JSON serialization/deserialization if necessary.**
   * **If you’re using NUnit, install NUnit and NUnit3TestAdapter through the NuGet package manager for seamless test execution.**
3. **Set up the API Client:**
   * **This client will handle requests to each endpoint in https://fakestoreapi.com. Using a single API client method across tests can help ensure code reusability.**

**2. Writing Tests for Each Endpoint:**

**Here’s how you can tackle testing each endpoint with example code snippets for clarity:**

**a. GET /products - Retrieving Products**

**[Test]**

**public void GetAllProducts\_ShouldReturnProducts()**

**{**

**// Arrange**

**var client = new RestClient("https://fakestoreapi.com");**

**var request = new RestRequest("/products", Method.GET);**

**// Act**

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

**// Assert**

**Assert.AreEqual(HttpStatusCode.OK, response.StatusCode, "Expected 200 OK response.");**

**Assert.IsNotEmpty(response.Content, "Expected non-empty response content.");**

**}**

**This verifies that the GET request to fetch all products returns a status code of 200 and that the response contains data.**

**b. POST /products - Creating a Product**

**Creating a new product allows us to validate both the response status and the structure of the response.**

**[Test]**

**public void CreateProduct\_ShouldReturnCreatedProduct()**

**{**

**// Arrange**

**var client = new RestClient("https://fakestoreapi.com");**

**var request = new RestRequest("/products", Method.POST);**

**request.AddJsonBody(new {**

**title = "Test Product",**

**price = 29.99,**

**category = "electronics",**

**description = "A sample test product"**

**});**

**// Act**

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

**// Assert**

**Assert.AreEqual(HttpStatusCode.Created, response.StatusCode, "Expected 201 Created response.");**

**Assert.IsTrue(response.Content.Contains("Test Product"), "Response should contain the product title.");**

**}**

**c. PUT /products/{productId} - Updating a Product**

**Updating a product includes a positive test (valid productId) and a negative test (invalid productId).**

**[Test]**

**public void UpdateProduct\_ShouldReturnUpdatedProduct()**

**{**

**// Arrange**

**int productId = 1; // Assuming this product exists**

**var client = new RestClient("https://fakestoreapi.com");**

**var request = new RestRequest($"/products/{productId}", Method.PUT);**

**request.AddJsonBody(new {**

**title = "Updated Product Title",**

**price = 35.99**

**});**

**// Act**

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

**// Assert**

**Assert.AreEqual(HttpStatusCode.OK, response.StatusCode, "Expected 200 OK response.");**

**Assert.IsTrue(response.Content.Contains("Updated Product Title"), "Response should contain updated product title.");**

**}**

**[Test]**

**public void UpdateProduct\_InvalidId\_ShouldReturnNotFound()**

**{**

**// Arrange**

**int invalidProductId = 9999;**

**var client = new RestClient("https://fakestoreapi.com");**

**var request = new RestRequest($"/products/{invalidProductId}", Method.PUT);**

**request.AddJsonBody(new {**

**title = "Invalid Update"**

**});**

**// Act**

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

**// Assert**

**Assert.AreEqual(HttpStatusCode.NotFound, response.StatusCode, "Expected 404 Not Found response.");**

**}**

**d. DELETE /products/{productId} - Deleting a Product**

**[Test]**

**public void DeleteProduct\_ShouldReturnSuccess()**

**{**

**// Arrange**

**int productId = 1; // Assuming this product exists**

**var client = new RestClient("https://fakestoreapi.com");**

**var request = new RestRequest($"/products/{productId}", Method.DELETE);**

**// Act**

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

**// Assert**

**Assert.AreEqual(HttpStatusCode.OK, response.StatusCode, "Expected 200 OK response.");**

**}**

**e. GET /products/categories - Retrieving Categories**

**[Test]**

**public void GetCategories\_ShouldReturnCategories()**

**{**

**// Arrange**

**var client = new RestClient("https://fakestoreapi.com");**

**var request = new RestRequest("/products/categories", Method.GET);**

**// Act**

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

**// Assert**

**Assert.AreEqual(HttpStatusCode.OK, response.StatusCode, "Expected 200 OK response.");**

**Assert.IsNotEmpty(response.Content, "Expected non-empty list of categories.");**

**}**

**f. GET /products?sort=asc - Sorting Products**

**[Test]**

**public void GetSortedProducts\_ShouldReturnSortedProducts()**

**{**

**// Arrange**

**var client = new RestClient("https://fakestoreapi.com");**

**var request = new RestRequest("/products?sort=asc", Method.GET);**

**// Act**

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

**// Assert**

**Assert.AreEqual(HttpStatusCode.OK, response.StatusCode, "Expected 200 OK response.");**

**}**

**2.Second Assignment**

**Project Setup in Visual Studio**

1. **Create a New Project**: Open your existing Visual Studio solution and add a new project for UI testing:
   * Go to **File > Add > New Project**.
   * Select **Class Library (.NET Core)**, and name it SauceDemoUITests.
2. **Install Dependencies**:
   * Right-click the new project in **Solution Explorer** and select **Manage NuGet Packages**.
   * Install the following packages:
     + Selenium.WebDriver
     + Selenium.WebDriver.ChromeDriver (or EdgeDriver, depending on your preferred browser)
     + NUnit
     + NUnit3TestAdapter (for running NUnit tests in Visual Studio)
3. **Organize Project Structure**:
   * Create a folder structure in the project:
     + **Pages** – to store page object classes
     + **Tests** – to store the test classes
     + **Utilities** – for helper classes (e.g., configuration management)

**Implementing the Page Object Model (POM)**

The POM pattern organizes the elements and actions for each page in separate classes, making tests easier to maintain. Here’s how to implement it for Sauce Demo.

**1. LoginPage.cs (in the Pages folder)**

using OpenQA.Selenium;

namespace SauceDemoUITests.Pages

{

public class LoginPage

{

private IWebDriver \_driver;

// Constructor

public LoginPage(IWebDriver driver)

{

\_driver = driver;

}

// Elements

private IWebElement UsernameField => \_driver.FindElement(By.Id("user-name"));

private IWebElement PasswordField => \_driver.FindElement(By.Id("password"));

private IWebElement LoginButton => \_driver.FindElement(By.Id("login-button"));

// Actions

public void Login(string username, string password)

{

UsernameField.SendKeys(username);

PasswordField.SendKeys(password);

LoginButton.Click();

}

}

}

**2. ProductsPage.cs (in the Pages folder)**

using OpenQA.Selenium;

namespace SauceDemoUITests.Pages

{

public class ProductsPage

{

private IWebDriver \_driver;

public ProductsPage(IWebDriver driver)

{

\_driver = driver;

}

private IWebElement BackpackAddToCartButton => \_driver.FindElement(By.Id("add-to-cart-sauce-labs-backpack"));

private IWebElement BikeLightAddToCartButton => \_driver.FindElement(By.Id("add-to-cart-sauce-labs-bike-light"));

private IWebElement CartLink => \_driver.FindElement(By.ClassName("shopping\_cart\_link"));

public void AddBackpackToCart() => BackpackAddToCartButton.Click();

public void AddBikeLightToCart() => BikeLightAddToCartButton.Click();

public void OpenCart() => CartLink.Click();

}

}

**3. CartPage.cs (in the Pages folder)**

using OpenQA.Selenium;

using NUnit.Framework;

namespace SauceDemoUITests.Pages

{

public class CartPage

{

private IWebDriver \_driver;

public CartPage(IWebDriver driver)

{

\_driver = driver;

}

private IWebElement CheckoutButton => \_driver.FindElement(By.Id("checkout"));

private IReadOnlyCollection<IWebElement> CartItems => \_driver.FindElements(By.ClassName("cart\_item"));

public void ValidateCartItems(int expectedItemCount)

{

Assert.AreEqual(expectedItemCount, CartItems.Count, "Item count in the cart does not match.");

}

public void Checkout() => CheckoutButton.Click();

}

}

**Test Implementation (using NUnit)**

**1. SauceDemoTests.cs (in the Tests folder)**

using NUnit.Framework;

using OpenQA.Selenium;

using OpenQA.Selenium.Chrome;

using SauceDemoUITests.Pages;

namespace SauceDemoUITests.Tests

{

public class SauceDemoTests

{

private IWebDriver \_driver;

private LoginPage \_loginPage;

private ProductsPage \_productsPage;

private CartPage \_cartPage;

[SetUp]

public void Setup()

{

\_driver = new ChromeDriver();

\_driver.Manage().Window.Maximize();

\_driver.Navigate().GoToUrl("https://www.saucedemo.com/");

\_loginPage = new LoginPage(\_driver);

\_productsPage = new ProductsPage(\_driver);

\_cartPage = new CartPage(\_driver);

}

[Test]

public void SauceDemoCompleteOrderTest()

{

// Login

\_loginPage.Login("standard\_user", "secret\_sauce");

// Add Products to Cart

\_productsPage.AddBackpackToCart();

\_productsPage.AddBikeLightToCart();

\_productsPage.OpenCart();

// Validate Cart

\_cartPage.ValidateCartItems(2);

// Checkout Process

\_cartPage.Checkout();

Assert.Pass("Order placed successfully.");

}

[TearDown]

public void TearDown()

{

\_driver.Quit();

}

}

}