Individual Class Exercise: Exploring Testing Tools for ASP.NET MVC Apps

Course: APDP201
Date: 10 September 2025
Student Number: 22103651
Name: Mfundo Mahlaba

1. Introduction

This report explores three pivotal testing tools for ensuring the quality, functionality, and security of ASP.NET MVC applications. It includes a review of each tool, practical use cases, and a demonstration with xUnit.net and Selenium WebDriver.

2. Tool Reviews & Demos

Tool 1: xUnit.net (Unit Testing)

Use Case in ASP.NET MVC: Perfect for testing the logic of controller actions, model validation, and service layers in isolation.
Key Advantages: Modern, fast, and the community standard for .NET unit testing.
Key Limitations: Designed for isolated unit tests, not end-to-end flows.
Demo: Unit Testing a Controller Method

Sample Code:

```
public class HomeControllerTests
{
    [Fact]
    public void Index_Returns_ViewResult_With_Welcome_Message()
    {
        var controller = new HomeController();
        var result = controller.Index();
        var viewResult = Assert.IsType<ViewResult>(result);
        Assert.Equal("Welcome to APDP201!", viewResult.ViewData["Title"]);
    }
}
```

(Screenshot Placeholder: Test Explorer with passed test.)

Tool 2: Selenium WebDriver (Functional / End-to-End Testing)

Use Case in ASP.NET MVC: Automates browser interactions to test complete user workflows, such as logging into the application, ensuring UI elements work, and validating post-login redirects.
Key Advantages: Tests the application from a real user's perspective and works across multiple browsers.
Key Limitations: Tests are slower and can be brittle if reliant on unstable HTML selectors.
Demo: Automating a Login Page

Sample Code:

```
public class LoginUITests : IDisposable
{
    private readonly IWebDriver _driver;
    public LoginUITests() => _driver = new ChromeDriver();
    public void Dispose() => _driver.Quit();

    [Fact]
    public void Login_With_Valid_Credentials_Redirects_To_Home()
    {
        _driver.Navigate().GoToUrl("https://localhost:7234/Account/Login");
        _driver.FindElement(By.Id("Email")).SendKeys("student@example.com");
```

```
            _driver.FindElement(By.Id("Password")).SendKeys("SecurePassword123!");
            _driver.FindElement(By.TagName("button")).Click();
            System.Threading.Thread.Sleep(2000);
            Assert.Contains("Welcome", _driver.PageSource);
        }
    }
```

(Screenshot Placeholder: Chrome browser showing automated login.)

Tool 3: OWASP ZAP (Zed Attack Proxy) (Security Testing / DAST)

Use Case in ASP.NET MVC: Proxies browser traffic to actively scan a running MVC application for common vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).
Steps to Get Started:
1. Download and run the ZAP executable from OWASP.
2. Configure your web browser to use ZAP as a proxy (localhost:8080).
3. Manually browse your site or use the 'Automated Scan' feature while ZAP records and analyzes the traffic.
4. Review the generated security report with alerts ranked by risk.
Key Advantages: Free, open-source, and crucial for identifying security holes.
Key Limitations: Can generate false positives and requires the app to be running to test it.

3. Conclusion

Utilizing a combination of unit tests (xUnit), end-to-end UI tests (Selenium), and security scans (ZAP) provides a robust testing strategy for any ASP.NET MVC project. This multi-layered approach ensures code correctness, functional reliability, and protection against common security threats.