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 testing tools relevant for ASP.NET MVC application development. For each tool, I will describe its purpose, its application in an MVC project, and the steps to get started. I will also provide a demo for two of the selected tools.

2. Tool Reviews

Tool 1: xUnit.net (Unit Testing)

Use Case in ASP.NET MVC: xUnit is ideal for testing the business logic within your application. In an MVC project, this is perfect for testing controller actions, model validation, and service layers.
Steps to Get Started:
1. Installation: Create an xUnit Test Project or add xunit packages via NuGet.
2. Configuration: Add a project reference to your MVC project.
3. Writing your first test: Use [Fact] attribute and xUnit assertions.

Key Advantages:
- Clean, modern, extensible.
- Strong community support, standard for .NET.
- Very fast execution for isolated code.

Key Limitations:
- Designed mainly for unit tests, not integration or UI tests.

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

Use Case in ASP.NET MVC: Selenium automates browsers to test applications from a user's perspective. It helps test complete flows like login, registration, and form submissions.
Steps to Get Started:
1. Installation: Install Selenium.WebDriver and Selenium.Support packages.
2. Configuration: Download the browser driver (e.g., ChromeDriver).
3. Writing tests: Automate actions (Click, SendKeys) and assert page content.

Key Advantages:
- Works across multiple browsers.
- Enables realistic, end-to-end tests.
- Can integrate with xUnit.

Key Limitations:
- Slower execution.
- Brittle tests if selectors are unstable.
- Requires graphical or headless environment for CI/CD.

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

Use Case in ASP.NET MVC: ZAP is used to detect vulnerabilities in a running web app, including SQL Injection, XSS, CSRF, and misconfigurations.
Steps to Get Started:
1. Installation: Download and run the ZAP executable.
2. Configuration: Set browser proxy to ZAP (localhost:8080).
3. Running tests: Browse app or use Automated Scan, review reports.

Key Advantages:
- Free, open-source, powerful for scanning.
- Generates reports ranked by risk.

Key Limitations:
- Can generate false positives.
- Requires running app to test.

3. Demo

Demo 1: xUnit.net - Testing a Controller Method

```
using Microsoft.AspNetCore.Mvc;
using MyAspNetMvcApp.Controllers;
using Xunit;

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: Visual Studio Test Explorer showing test passed.)

Demo 2: Selenium WebDriver - Automating a Login Page

```
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using Xunit;

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: Chrome browser showing automated login with fields populated.)