CA1 Report

# Source code - including tests:

**BloodPressure.cs**

namespace BPCalculator

{

// BP categories

public enum BPCategory

{

[Display(Name = "Low Blood Pressure")] Low,

[Display(Name = "Normal Blood Pressure")] Normal,

[Display(Name = "Pre-High Blood Pressure")] PreHigh,

[Display(Name = "High Blood Pressure")] High,

[Display(Name = "Outside of measurable ranges")] Abnormal

};

public class BloodPressure

{

public const int SystolicMin = 70;

public const int SystolicMax = 190;

public const int DiastolicMin = 40;

public const int DiastolicMax = 100;

[Display(Name = "Systolic blood pressure")]

[Range(SystolicMin, SystolicMax, ErrorMessage = "Invalid Systolic Value")]

public int Systolic { get; set; } // mmHG

[Display(Name = "Diastolic blood presure")]

[Range(DiastolicMin, DiastolicMax, ErrorMessage = "Invalid Diastolic Value")]

public int Diastolic { get; set; } // mmHG

// calculate BP category

[Display(Name = "Calculate blood presure")]

public BPCategory Category

{

get

{

/\*application insight in azure for telemetry

release as a container or azure app service

acceptence testing bdd, business tesing tests, can be written in visual studio, automate user input see whats comming back, write selenium tests to simulate this process

perfromance test in azure portal 50 users over 2 minutes build either in vs or azure, if perfromnce does not meet req hold the release

\*/

if (IsWithinRange(Systolic, 140, 190) && IsWithinRange(Diastolic, 90, 100))

{ return BPCategory.High; }

else if(IsWithinRange(Systolic, 120, 140) && IsWithinRange(Diastolic, 80, 90))

{ return BPCategory.PreHigh; }

else if(IsWithinRange(Systolic, 90, 120) && IsWithinRange(Diastolic, 60, 80))

{ return BPCategory.Normal; }

else if(IsWithinRange(Systolic, 70, 90) && IsWithinRange(Diastolic, 40, 60))

{ return BPCategory.Low; }

else

{ return BPCategory.Abnormal; }

}

}

/// <summary>

/// Takes category and returns class to allow for text colour change.

/// </summary>

/// <param name="category">blood pressure category</param>

/// <returns>css class</returns>

public string GetDivClass(BPCategory category)

{

string cssClass = string.Empty;

switch (category)

{

case BPCategory.Low:

cssClass = "cssLow";

break;

case BPCategory.Normal:

cssClass = "cssNormal";

break;

case BPCategory.PreHigh:

cssClass = "cssPreHigh";

break;

case BPCategory.High:

cssClass = "cssHigh";

break;

default:

cssClass = "cssAbnormal";

break;

}

return cssClass;

}

/// <summary>

/// Calculate if blood preasure is within ranges

/// </summary>

/// <param name="value">bp value</param>

/// <param name="minimum">min range</param>

/// <param name="maximum">max range</param>

/// <returns></returns>

public bool IsWithinRange(int value, int minimum, int maximum)

{

return value >= minimum && value <= maximum;

}

}

}

**BloodPressure.cshtml**

<div class="@Model.BP.GetDivClass(Model.BP.Category)">

<span id="bpCategoryResult" > @Html.DisplayFor(model => model.BP.Category, new { htmlAttributes = new { @class = "form-control" } })</span>

</div>

**Site.min.css**

/\*Css class for changing text colour\*/

.cssLow, .form-bottom {

color: #6495ED;

}

.cssNormal {

color: #228B22;

}

.cssPreHigh {

color: #FF7F50;

}

.cssHigh {

color: #DC143C;

}

.cssAbnormal {

color: #B22222;

}

SeleniumTest.cs

namespace SeleniumTest

{

[TestClass]

public class SeleniumTest1

{

private TestContext testContextInstance;

private IWebDriver driver;

private string appURL;

public SeleniumTest1()

{ }

[TestMethod]

[TestCategory("Chrome")]

public void HighPressureTest()

{

driver.Navigate().GoToUrl(appURL + "/BloodPressure");

//Clear the input fields

driver.FindElement(By.Id("BP\_Systolic")).Clear();

//insert some test data

driver.FindElement(By.Id("BP\_Systolic")).SendKeys(systolicHigh);

driver.FindElement(By.Id("BP\_Diastolic")).Clear();

driver.FindElement(By.Id("BP\_Diastolic")).SendKeys(diastolicHigh);

driver.FindElement(By.Id("BP\_Systolic")).Click();

string resultText = driver.FindElement(By.Id("bpCategoryResult")).Text;

Assert.IsTrue(resultText.Equals("High Blood Pressure"));

}

[TestMethod]

[TestCategory("Chrome")]

public void PreHighTest()

{

driver.Navigate().GoToUrl(appURL + "/BloodPressure");

//Clear the input fields

driver.FindElement(By.Id("BP\_Systolic")).Clear();

//insert some test data

driver.FindElement(By.Id("BP\_Systolic")).SendKeys(systolicPreHigh);

driver.FindElement(By.Id("BP\_Diastolic")).Clear();

driver.FindElement(By.Id("BP\_Diastolic")).SendKeys(diastolicPreHigh);

driver.FindElement(By.Id("BP\_Systolic")).Click();

string resultText = driver.FindElement(By.Id("bpCategoryResult")).Text;

Assert.IsTrue(resultText.Equals("Pre-High Blood Pressure"));

}

[TestMethod]

[TestCategory("Chrome")]

public void NormalPressureTest()

{

driver.Navigate().GoToUrl(appURL + "/BloodPressure");

//Clear the input fields

driver.FindElement(By.Id("BP\_Systolic")).Clear();

//insert some test data

driver.FindElement(By.Id("BP\_Systolic")).SendKeys(systolicNormal);

driver.FindElement(By.Id("BP\_Diastolic")).Clear();

driver.FindElement(By.Id("BP\_Diastolic")).SendKeys(diastolicNormal);

driver.FindElement(By.Id("BP\_Systolic")).Click();

string resultText = driver.FindElement(By.Id("bpCategoryResult")).Text;

Assert.IsTrue(resultText.Equals("Normal Blood Pressure"));

}

[TestMethod]

[TestCategory("Chrome")]

public void LowPressureTest()

{

driver.Navigate().GoToUrl(appURL + "/BloodPressure");

//Clear the input fields

driver.FindElement(By.Id("BP\_Systolic")).Clear();

//insert some test data

driver.FindElement(By.Id("BP\_Systolic")).SendKeys(systolicLow);

driver.FindElement(By.Id("BP\_Diastolic")).Clear();

driver.FindElement(By.Id("BP\_Diastolic")).SendKeys(diastolicLow);

driver.FindElement(By.Id("BP\_Systolic")).Click();

string resultText = driver.FindElement(By.Id("bpCategoryResult")).Text;

Assert.IsTrue(resultText.Equals("Low Blood Pressure"));

}

/// <summary>

///Gets or sets the test context which provides

///information about and functionality for the current test run.

///</summary>

public TestContext TestContext

{

get

{

return testContextInstance;

}

set

{

testContextInstance = value;

}

}

[TestInitialize()]

public void SetupTest()

{

appURL = "https://bpcalcapp-bpcalcqa.azurewebsites.net";

string browser = "Chrome";

switch (browser)

{

case "Chrome":

driver = new ChromeDriver(Environment.GetEnvironmentVariable("ChromeWebDriver"));

break;

case "Firefox":

driver = new FirefoxDriver(Environment.GetEnvironmentVariable("GeckoWebDriver"));

break;

case "IE":

driver = new InternetExplorerDriver(Environment.GetEnvironmentVariable("IEWebDriver"));

break;

default:

driver = new ChromeDriver(Environment.GetEnvironmentVariable("ChromeWebDriver"));

break;

}

}

[TestCleanup()]

public void MyTestCleanup()

{

driver.Quit();

}

//High pressure test parametrs

private const String systolicHigh = "160";

private const String diastolicHigh = "100";

// Pre High pressure test parameters

private const String systolicPreHigh = "120";

private const String diastolicPreHigh = "80";

//Normal pressure test parametrs

private const String systolicNormal = "100";

private const String diastolicNormal = "80";

//Normal pressure test parametrs

private const String systolicLow = "80";

private const String diastolicLow = "60";

}

}

# Report describing the build pipeline (tasks, design philosophy, yaml), likewise for the release pipeline except maybe include a screenshot since no yaml for release just yet

**BPCalculator1**

Design philosophy: Build and release web application to Azure, the web application consists of two additional stages. Stage one serves QA this is where first automated test are running to validate stability of the code release, once tests pass deployment moves to next step and runs load testing on a “just like production” environment, this assures that all performance tests are run on production mirror ensuring that the application will be able to perform as expected. Once the tests on stage 1 and 2 pass successfully, release is deemed production ready. The deployment was designed to utilise automated provisioning of applications for required deployment and tests, environment is deleted afterwards to reduce the cost.

Build:

1. Use NuGet 4.4.1 – downloads and installs nuget 4.4.1 on the host
2. Nuget restore - Runs nuget restore on the .sln within the workspace
3. Build solution – builds the .sln within the workspace
4. VsTest – runs unit tests, targets files containing UnitTest in their name
5. Copy Files – copies all artifacts to the staging directory, targets specific file extensions, such as: .dll, .xml, .exe for automated testing, .json and .ps1 for provisioning
6. Publish Artifacts – publishes artifacts to shared repository within azure devops

Release:

Stage 1

1. Provision web app resources in Azure – creates application service plan and web application, including two stages
2. Deploy Azure App Service: Deploys the application in Azure on provisioned web app
3. VsTest – runs automated tests

Stage 2

1. Deploy Azure App Service – Deploys application on provisioned stage.
2. Web Performance test – runs load test
3. Azure PowerShell script: InlineScript – Runs powershell script to destroy all resources to save cost

# Tell me about the new feature (e.g. a user story) added

The new feature changes the colour of the text depending on the blood pressure level, it highlights to the user if the result is threatening or not in colour. To achieve this new feature additional css was added to existing css file sit.min.css (cssLow, cssNormal, cssPreHigh, cssHigh, cssAbnormal)

And the BloodPressure.cshtml file has been adjusted to apply new style depending on the result printed to the page, a new method was added to the business layer GetDivClass(BPCategory category) this method takes in category as a parameter and returns name of the css class to be used for display.