**Writeup for Phase-4 Project:**

**Name: Nileena James.**

Github link: <https://github.com/niljames/Phase-4-Assessment.git>

Url: http://phase-4.azurewebsites.net/

Question:

DESCRIPTION

Build an ASP.NET website comprising of the following three pages:

* Pizza selection page
* Order checkout page
* Order confirmation page

Create test suites to test the pages for end-user testing.

**Background of the problem statement:**

Joe’s Pizza has created the UI for an online ordering portal. They want the pages to be properly tested with a testing framework using NUnit, Selenium and Specflow. The prototype will then be deployed via Jenkins to an Azure VM at scheduled intervals.

**You must use the following:**

* Visual Studio ASP.NET Core Project
* NUnit
* Selenium
* SpecFlow
* Jenkins
* Windows Azure

**You have been hired to perform the following tasks:**

* Create an ASP.NET Core web application containing three cshtml pages:
  + A Pizza selection page which will allow users to choose which type of pizza they want to order.
  + An Order Checkout page which will show the selected pizza with quantity and pricing. Clicking on Checkout will take them to the confirmation page.
  + Order Confirmation page which will display a message with an order id, amount and the type of pizza that is ordered.
* Create a Windows Class Library project.
* Add a project reference to the first project.
* Configure NUnit, Selenium and Specflow.
* Write test cases to test the web app.
* Run the test cases using Test Runner.
* Create an Azure VM for hosting the prototype app.
* Set up Jenkins to create a build at scheduled intervals in the day and deploy it to Azure.

**Explanation:**

* I created a database in Azure named as PizzaDB, where I created a table named Products and entered the respective details into the table.
* Using this database, I used the connection string of the database for the MVC application to retrieve the details from the table.
* For testing, I used both Nunit as well as Selenium to test all the functionalities of the three pages as mentioned in the question.

![Graphical user interface, website

Description automatically generated]()**Fig 1**: Homepage (Pizza Selection Page)

* The first page as seen in the picture is the Pizza Selection Page, where I have extracted the different products from the cloud database “PizzaDB” and represented the products in the form of the table.
* Clicking on View details will show details of the particular product and add to cart will add that particular product to the cart.

![Graphical user interface, website

Description automatically generated]() **Fig 2**: Homepage (Pizza Selection Page)

![A picture containing text, pizza, dish, food

Description automatically generated]()

**Fig 3**: View Details Page

Table

Description automatically generated **Fig 4**: Order Checkout Page

* Represented in the form of cart where all the items selected are added into the cart for purchase.
* Clicking on checkout will direct you to the payment page, where the total amount to pay will be displayed.
* The payment gateway integration was done via stripe.

Graphical user interface, application

Description automatically generated**Fig 5**: Payment Gateway page.

Graphical user interface, application

Description automatically generated

* The remaining images contain the code for the testing using Nunit as well as Selenium.

![Text

Description automatically generated]()**Fig 7.1**: Selenium Testing Code

![Text

Description automatically generated]()**Fig 7.2**: Selenium Testing Code

![Text

Description automatically generated]()**Fig 7.3**: Selenium Testing Code

![Text

Description automatically generated]()**Fig 7.4**: Selenium Testing Code

* As seen in the code, the functionalities have been written to test all the three pages.
* Explanation of the code:
* First, the selenium.WebDriver package has to be installed.
* WebDriver has to be installed in your system and must be called from within the code through the path.
* Along with the WebDriver, the ChromeDriver must also be used in order to perform the tests in the Chrome Browser.
* Then the url of the site created must be passed, so that it can be accessed.
* Once the url is opened successfully, either the Id or the name of the fields are passed to retrieve certain fields or buttons and sendKeys() function is used to fill the values in the field or to click the button.
* Also the current window handle is stored for future reference to retrieve that page (to access the parent window from the child window).
* Thread.sleep() is used to pause the automation for a certain time limit.
* The test code is run by starting a new debug instance keeping the MVC app running in the background.
* The remaining images as seen below are the images for Nunit testing.

![Text

Description automatically generated]()**Fig 8.1**: Nunit Testing Code

![Text

Description automatically generated]() **Fig 8.2**: Nunit Testing Code ![Text

Description automatically generated]() **Fig 8.3**: Nunit Testing Code

![Text

Description automatically generated]() **Fig 8.4**: Nunit Testing Code

![Graphical user interface, text

Description automatically generated]() **Fig 8.5**: Nunit Testing Code

* Explanation of the code for Nunit Testing:
* The code explanation is almost like that of the Selenium Testing but with few exceptions.
* There are two functions setup() and Test1().
* In the Setup() function, the drivers are created and mentioned and the urls for navigation is given.
* Meanwhile the Test1() function, contains the code for testing the functionalities of all the three pages as mentioned in the Selenium Testing.
* The testExplorer is used to run the test while keeping the mvc app running on the other project.
* As seen in 8.5, the test cases have passed successfully.

![Text

Description automatically generated]()

**Fig 9**: Azure Cloud Database

* The database created in Azure is PizzaDB and the table used is Product.
* The screenshot shows the execution of the query “Select \* from Product” which retrieved all the data in the table.
* The same table can be accessed in the SSMS (Sql Server Management Studio) by entering the required credentials.
* After all this, the project is then deployed in Azure via App Services, which can be accessed by clicking on this url: http://phase-4.azurewebsites.net/