Jesse Fulcher

CSCI 392

Fall Independent Study

Final Submission

Project Summary:

For my Fall Independent Study I decided I wanted to learn more about Automated Testing and more specifically the tools offered inside of Visual Studio 2013 that helped take away the manual process of testing a web application. When I started this adventure I did not have any knowledge of what was out there or available and a very limited undertanding of ASP.NET and web applications in general. To start my project I began doing research into the field of automated testing in general, and to my surprise I found that automated testing is sort of a new tool that business had just started really utilizing recently.

Inside of Visual Studio I quickly found that there were three main ways that you could test; Unit Tests, Automated UI Tests and Load Testing. I was really only able to get into the first two, but from my research I found quite a bit of information on the subject.

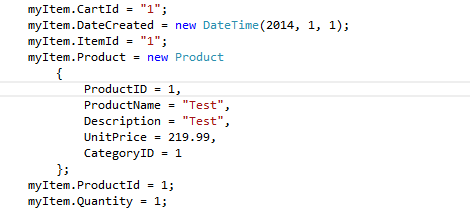
When I first started my project I wanted to come up with an idea that was simple, yet testable. If I were to just create a blog or a 1-way site for users to log into there wasn’t as much testable material so I knew that I needed to focus on an application that utilized user input so that I could test what the user is entering and test that against my logic built into the site. This is why I settled on building an ecommerce/shopping cart application. The concept seemed easy enough for me to follow and it had plenty of testable material that I could learn how to write testable code on. The second step was to pick the subject. There were lots of tutorials online that had a variety of toys/apparel/etc but I wanted to focus on a subject that had my interest, so I created an online hockey gear store. From this idea came, Jesse’s Hockey Shop.

Not having much experience in buildiing ASP.NET web applicaitons from scratch I decided to follow some tutorials online to bulild a simular design and I settled on this one I found on the ASP.NET web site, a very powerful resoure for anybody wanting to learn ASP.NET and its tools. I followed the steps in this tutorial to begin building my ecommerce site from scratch:

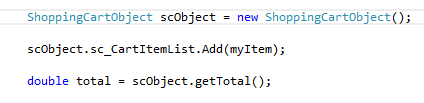
<http://www.asp.net/web-forms/overview/getting-started/getting-started-with-aspnet-45-web-forms/introduction-and-overview>

With this groundwork in place and me starting to tweak the design to match what I was going for it was time to get into the testing phase. I decided to start with the Unit Tests as this seemed to be a very valuable tool and the idea of being able to test my test against itself seemed very beneficial. Through this process I quickly found that in order to test your code with Unit Tests you need to be able to have testable code. Due to the structure of my code and how it was accessing the database it was hard to write Unit Tests. All of the examples and tutorials online were giving me good information but without having a separate BLL to test against it was really hard. It was at this point I realized that I needed to separate out my logic form my code behind in order to write my tests.

One common idea that I kept finding while research Unit Tests was this idea of the 3 A’s (Arrange, Act, Assert). What this means you you need to first setup your environment, then take your action and finally assert your manually entered values to what your code gives you to see if your test will fail. When putting this to action, this was the hardest part. You need to basically build out an Object and hardcode in values. So for example I wanted to test that when my shopping cart had only one item in it, my total was showing the correct price for that one item. In order to accomplish this I need to build up a item in code. This was part of the Arrange Step. This lead to me writing something like this:



With this new item built I could then create a new instance of the shopping cart and add it item to it to get the total:



This completed the Act action.

Finally, in order to compare these to each other I would asses by using the Assert class and calling the AreEqual() method:



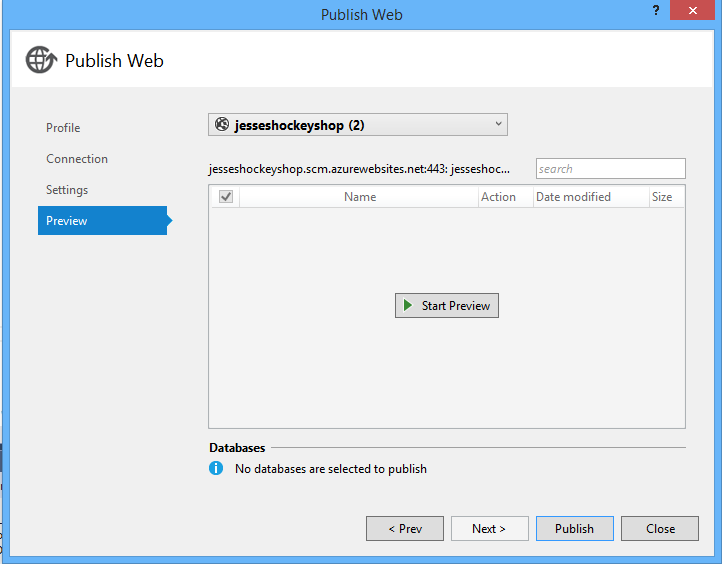
When I run this test in test explorer it would take the Unit price and test it against my hard coded value to tell me if the total was returning the correct value. I then repeated this step with several different cases I used if I had multiple quantities of the same item, or if I had multiple different items. I could use the same logic, but with different cases. Now, if I go in and make changes to my code, I can run this unit test to see if I’m still getting the correct total. I can run this test before I publish my site or after any changes without having to manually launch the site, put in different values, and manually checking to see if they are correct.

A major part of Unit Testing is to ensure that your Unit Tests are correct and the best way to do this is to write your test to fail. When I first wrote this test, in the Assert call I first put in a value of 0.00 knowing that this is incorrect. Now when I ran the test it failed so I know that my test is working correctly.

After successfully getting some unit tests done, I decided it was time for me to look into Automated Unit Tests. From my research into the UI tests I determined that I actually needed to have my site up and running on the web so before I could do these test I needed to learn how to publish my site and this was a learning experience all in itself.

In order to publish I first needed to find a good host. Finding a host was a task in that there are many that wanted me to pay, some that were free with limited space and some that just seemed not reliable or fake. After researching different options for a host I decided to go with a free one called Somee. Somee bragged of easy ASP.NET hosting and free domain names and it seemed reputable, however once I actually created an account it was a nightmare. There was very little support and I couldn't’ find any help. Their marketing site was very well done and look professional, however their internal application looked like it was created 10 years ago and the instructions were how to launch your APS.NET applicaiton only had instructions for Visual Studio 2010. Needless to say, I was never able to get my site working for Somee and after many nights of trial and error I just couldn’t get it to work.

At this point I turned my focus to Azure. Azure was expensive but they had a trial period that I could put my site up and host for a limited time. Azure was full of features and through other various Microsoft sites there was tons of support. As soon as I registered there were tutorials and help to get my site up. Because Azure is with Microsoft they even had plugins I could download to publish directly from Visual Studio. Once I had the Azure SDK I could publish my site with the click of a button:

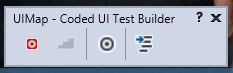


Internally there were tons of tools from Azure to monitor my sites activity, upload and change connection strings and overall it was a very easy experience to learn.

Now that I finally had my site published to the web and had a web address:

<http://jesseshockeyshop.azurewebsites.net/>

I was able to begin writing my UI tests. The UI Tests were pretty easy and I think due to them being almost completely graphical in nature helps in understanding how they work. To begin you would use the UI Test Tool inside of your UI Test project to record actions. This gives you a tool like this:



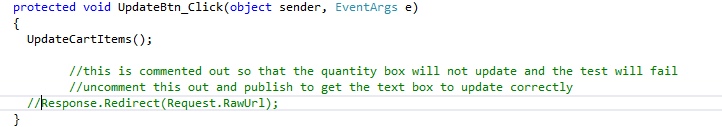
to start recording actions.

It’s important to note that you must begin with the actual launching of your application. This means you need to click on IE from your desktop, type in the name of your web address and then you can begin your testing. You cannot run locally, and you cannot just begin recording from your site. When you actually run your tests, it takes control of your computers mouse and opens the site application just like you were doing it your self. Its actually kind of weird to watch because its going through the actions on your computer without you doing it.

After going through the same process of opening my web browser and navigating to my site over and over I decided there had to be an easier way. This is when I decided to make the launching of IE and navigating to my site its own method. I could reuse this method over and over in each of my tests without having to re record this step. After coming to this realization I decided I could do that with other methods as well. For example, I wanted to test going to the shopping cart and make sure items were being added and counted correctly. Instead of recording an action for each of these I could write one method that would navigate to the products page, add items, and then update the items. Each test would test different values, but the one method would get us to where we needed to be. This cut down on the work quite a bit.

Once you have navigated to where you need to be you also have to assert the values in your controls. For example on my shopping cart if I had a couple of different items and I wanted to update the quantity, and check to ensure the total was adding up correctly, after clicking the Update you have to stop, create the code for this step and then start a new method with the assert button. Then drag that onto your browser to compare the values and ensure they are correct. Because of this assert method this can lead to many different steps, although your asses should be your last step.

Much like the Unit Tests I wanted to make sure that my UI Tests were working properly, so I removed code from my project to make it break. When you clicked the update button I made it not update the Quantity box. I changed this in my project:



By not doing a redirect, it wouldn’t update the quantity in the text box, therefore my test should fail. When I actually ran my first UI test, it passed. Why? Well, when looking at my first Asses Method I was comparing the control name to itself and not the value. I quickly learned that by default it is comparing the name of the control and not the values in it. When I did change my test to test the values in the box instead of the name of the control it failed as it should. This meant that my test was working properly.

-------------------------------------------------------------------

Reflection:

Overall I feel like this project has been very beneficial to my learning in the field of automated testing. Not only did I learn about how to write different types of tests within Visual Studio but as a side effect I learned how to create an ASP.NET web application shopping cart from scratch, build a more testable code structure using BLL’s and separating out the logic from the code behind, how to publishe my website to the web using an online host, and I also learned how beneficial automated testing in weather that be testing the logic of your code or how well the UI controls on your site work. I believe that from this Independent Study I have a much better understanding of Automated Testing and can speak to the process and some of the tools available, which in turn makes me a more valuable employee or applicant in the future.