**Introduction**

My suite is developed using SpecFlow with Selenium C# and nUnit framework for the assertions.

IDE Visual Studio.

The solution is designed in Page Object Model (POM). To make it simpler I have all the steps in one class (ProjectSteps) and all the methods in another one (Actions), but in a real situation I would have different classes for different pages (one for homepage, another one for login, register….)

19/04: In the first version, the POM framework referred only to the fact that the business and the tech facing were separated, I kept it simple due to the time restrictions. To implement that separation, in this version, I’ve created a structure project with classes for different webs in the application, which makes them totally reusable (see for instance the use of the method ClickAgreeTerms from checkout page used in a method in LogIn page)

**• Identify five high risk areas of the application**

**• Select two (or more) of the risks identified and create automated tests for them**

In a real-life situation, with 4 hours to have the confidence to deploy, I would go for manual testing, but then, where would be the fun?

My identified 5 high risk areas are:

1.- Navigation and presence of critical elements which must be checked.

The correct navigation is checked in the background step in the feature (Given the user has navigated to 'https://demowebshop.tricentis.com/').

Critical elements could be checked in a similar to the way I’m checking that the title element is present to check if navigation to the page has been successful. We can create a specific scenario with a list of elements to find.

2.- Register and Login are ok.

Once the page is working, being able to make a purchase, register and login are essential.

Scenario: S1 Login

Given the user clicks Log in menu option

When the user introduces 'amanda.fields@mail.com' in 'Email'

And the user introduces 'amandafields' in 'Password'

And the user clicks Log in button

Then the user is logged in as 'amanda.fields@mail.com'

3.- Navigation through the: catalog, products, poll… For instance, we should create a few scenarios to test the links to BOOKS, COMPUTERS, ELECTRONICS… and check there some elements. This would be quite easy to do as all the tests would be similar, with the same steps. Choosing the correct locators we only would need to change the parameters.

4.- Shopping cart

At least a complete workflow but separated scenarios too, to be able to isolate possible bugs, adding and removing items to the Cart

5.- Checkout.

This functionality by itself could be a suite. It’s crucial too so I’m creating a little scenario there to test the workflow.

Scenario: S2 Checkout

Given the user selects the catalogue 'books'

And the user adds to cart the book

And the user navigates to the Shopping cart

And the user agrees with the terms of service

When the user clicks Checkout button

And the user is logged as 'amanda.fields@mail.com' with password 'amandafields'

And the user clicks Continue in Billing Address

And the user selects In-Store Pickup

And the user clicks Continue in Shipping Address

And the user clicks Continue in Payment Method

And the user clicks Continue in Payment Information

And the user confirms order

Then the user gets the message an Order number

**NB**: As I was almost running out of time, a couple of locators are not working fine (InStorePickupCheckLocator and NextButtonLocator) They would need more time.

**Why you identified the five high risk areas**

They are the areas without which the web page wouldn’t make sense. That’s the reason not to have the Wishing list functionality.

**Why you chose to automate the test cases selected**

Making automated testing in a Register functionality is a little tricky, as you only can create once a user. So I’ve created a user and a Scenario to Log in (hopefully my user will be there when you run the tests)

Checkout is a vital functionality in a commercial web, even when this one is the simplest scenario.

**Detailed description of how to build and run**

I’m sending a solution so the build and the run can be done in VS.

As the build is created in the solution, you can run the suite from Command line too going directory in which the .dll is, and running the line:

dotnet vstest Assessment\_OUTsurance.dll --logger:trx

Text

Description automatically generated

It will indicate where the result logs are. In the project, you have some examples of the generated log.

Anyway, this would be a temporary solution. The best way to run a suite of automated tests is to have unattended jobs. Currently, I have my suites run by a DevOps pipeline which runs Jenkins tasks nightly.

**Provide any insights as to how you might go about improving them given more time.**

First thing would be to fix those locators (18/04, fixed. Pushed new commit).

Next thing I would create some BE(backend) steps to create users and elements in the shopping cart and teardowns with methods to delete them.

That would guarantee us the tests are completely replicable, independent from existing data in DB, and identical each time they run. At the same time, we wouldn’t have rubbish in the DB. Even being a test DB we don’t want that.

Then I would continue making more than 2 happy path scenarios (the application does what it means to do) and negative scenarios (the application doesn’t do what it doesn’t mean to do) in each risk area I listed. Later, risks that are not on that list (like the whishing list).

A comprehensive test plan is not in the scope of this assessment but it would be essential. I would need work sessions with QA, PO and developers to define and prioritise the suite.

I would recommend including a mechanism to take screenshots, at least in failing tests.

It is important to note that apart from having a scenario like my second one to test the workflow, which may be too fragile as it does too many things, I would create smaller and more robust scenarios, so I can identify and report better the bugs. I’ve tried to do something similar with a lot of checks but they are mainly focused on the locators, we should have more checks as the price of the book is correctly presented, the contact data are correctly extracted from DB…

To avoid having a crazy amount of automated tests, I wouldn’t include checks for mandatory fields and the use of correct types of data. That should be done with unit tests by developers.