**Assignment – Code Examination and Functionality Test | SDL4 RS A3.2 |**

**Robert Morgan**

For this task I will be writing unit tests for an ecommerce website. As the site in question makes extensive use of databases, I will be creating Moqs to test these. Testing is important for all projects to ensure correct functionality. On this site there are a lot of requirements for inputs that need to be tested, for example passwords requiring more than 8 characters, at least one number and one special character.

Although I will be completing this project on my own, I have the support of my line manager and wider team if I ever need it.

I am using NCrunch to help with testing. It is useful as it runs tests all the time allowing me to see live while I code if a test passes or fails. To the left of the code, right of the line numbers there will be green or red symbols, showing if a test passes or fails

I started by testing the Address class. My first test was for the isValidUKPostCode() method, passing in a valid postcode and asserting that the method returns true. I noticed that the test fails when a C is used as one of the last 2 characters in the post code.

Text

Description automatically generated

Checking the regex showed that C had been missed out.

Text

Description automatically generated

I do not know if this is correct for valid postcodes, but I assume this was deliberate as it would have been easier to use [A-Z], as they had for previous characters in the same regex expression, than it would have been to write out the alphabet excluding C.

Changing the C to a different letter makes the test pass as expected. Graphical user interface, text

Description automatically generated

While testing the is isValidHouseNumber() method I noticed that it does not verify properly, allowing for multiple trailing letters rather than just one, and allowing for the numeric part to exceed 1000.

Text

Description automatically generated

To test the isValidCity() method I needed to create a Mock of a database, as the method takes a parameter of IUserAccount userAccountDb. I chose to use Moq for this, and setup the Mock inside the Setup section of my tests.

Text

Description automatically generated

After writing tests to check that it returns true if a city is in the database and false if not, both tests pass

A screenshot of a computer

Description automatically generated with medium confidence

I noticed that there was no method for validating addressLine, so this will be ignored. The validation for this was supposed to be “The address line can combine letters and numbers but should not exceed 250 chars” but there is no code for this.

Next, I moved onto testing the UserAuthentication class. I started by testing the isValidPasswordFormat method. I noticed that testing the method using a valid password failed.

A screenshot of a computer

Description automatically generated with medium confidence

Investigating this revealed that the method hasExtraPasswordChars in the Utils class only searches for @,$ and £. A screenshot of a computer

Description automatically generated with medium confidence

This should be changed to include all possible special characters.

Changing the ! in my test to one of the characters in searches causes the test to pass

Text

Description automatically generated

I continued to write tests to check that the method returns false if any of the mentioned requirements are not met.

Text

Description automatically generated

I then tested the isValidUserNameFormat method. Testing for a valid username passed, however one of the requirements was that spaces or tabs before or after entry were ignored. This fails for this method. Text

Description automatically generated

However a very similar test for passwords passes.

Text

Description automatically generated

Investigating the code revealed that the method for validating passwords uses the string method .Trim() whereas the method for validating the username is missing this.

Text

Description automatically generated

Another error was found when testing the Username length. The specification states that username cannot be less than 2 characters or more than 50, however as seen in the screenshot above, the code only checks for a length that is more than 1, there is no check if the length is more than 50, causing the test for more than 50 characters to fail.

Text

Description automatically generated

The final class I can test is the UserAccountManager class.

Graphical user interface, text, chat or text message

Description automatically generated

The Utils class does not need its own tests, as all methods were covered by the tests for UserAuthentication as shown below.

Text

Description automatically generated

The QAExceptions class declares the custom user defined exceptions and so will be tested when these exceptions are thrown by the tests of UserAccountManager.

Text

Description automatically generated

So the final class I will be testing is the UserAccountManager class. This class is also the most complicated to test. Like with the tests for the Address class I will need to use Moq create a Mock of the database, however this time I will need to setup the Mock for all the methods in the IUserAccount interface, whereas I only has to setup the getCityNames() method for the testing of the Address class. Below is how I set up the Mock of the IUserAccount interface.

Text

Description automatically generated

I setup the register method to return true for any parameters it is given, as it is only called in one place after all the validation has been done, and does not affect the result of the method that it is called in at all. 

I then setup the mock for the isRegisteredUser method which returns a Boolean, so I setup the mock to return true if given the username “registered” and false if given username of “notregistered”. I used the same setup of getCityNames that I did in the tests for the Address class, and finally I setup isAnExistingUser in a similar way to isRegisteredUser, allowing the testing for when the method returns both true and false.

I then used the mockDb to create a new instance of UserAccountManager which will be used by the tests.

First, I wrote tests for the login method. The first test was simple as it was testing that calling the method as a registered user would return true. The next test was for a user that was not registered. This test was a little more complicated as a non-registered user causes an exception to be thrown. This was a user defined exception, so I tested that both the type of exception and the exception message match with what I expected. The final thing I tested for the login method was to test that a TooManyLoginTriesException was thrown if a user failed to login 3 times. To test this I had to put each login attempts prior to the one which will throw the exception in a try catch block, or else the test would fail as the login attempts throw a LoginException.

Text

Description automatically generated

I then moved onto testing the final method, register. I started by writing a test for a successful register, asserting that the result is true. For this I created a valid Address, including a city I setup in the mock as a valid city, and used the username “notexisting” which I setup in the mock, along with a password that meets the requirements. Due to the fact that there is not a method which validates addressLine I did not include it which is a problem with the code. Because multiple tests will use a valid address I moved the declaration of validAddress to the top of the class, outside of the tests.

Text

Description automatically generated

A picture containing text

Description automatically generated

After writing a test for successfully registering, I will be writing tests with different data to hit each if statement within the register method as shown below.

Text

Description automatically generated

Shown below are tests for attempting to register an existing user, and for attempting to register with an incorrect format for both username and password. All tests pass as expected, with the register method throwing the expected Exception of LoginException with the correct corresponding method for each.

Text

Description automatically generated

And finally the last 3 tests, testing for invalid: house number, postcode, and city. As mentioned before there should be validation for address line but this does not exist.

For each test I made a copy of validAddress, changing the thing I want to cause an exception. Each test passed as expected, with the register method throwing the expected Exception of InvalidUKAddressException with the correct corresponding method for each.

Other than the issue of there being no validation for addressLine there were no problems discovered by testing the UserAccountManager. All tests passed as expected.

In conclusion, this exercise gave me a lot of practise with unit testing and mocking. Although I had a reasonable amount of experience prior to this exercise, more is always better. I have learnt a few things through completing the exercise, for example using Assert to verify a specific exception was thrown and verifying that exception also had the expected message. What went well was that I successfully tested the entire solution, writing 37 unit tests which achieves almost 100% code coverage, and using Moq to create a mock of a database. I was able to find multiple bugs in the code, highlighting them and making suggestions as comments in the code. I will include screenshots of all of these below. I cannot think of anything that did not go well, or that I would do differently next time.

Errors/Bugs and how they could be improved:

Text

Description automatically generated

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated