# **Apply Activity:**

## **Objectives**

* + Your task is to examine the back-end code for an ecommerce site and to test its functionality.
  + Although the code works, there may be logical errors that need to be tested.
  + Your objective is to test the code, analyse the results, and correct any errors found using unit testing.
  + You will also create test scenarios for the critical parts of the application and report any errors discovered, along with suggested fixes.
  + Finally, you will submit your unit tests and completed code.

## **Functions performed by the site.**

Please read the following requirements which are performed by the site and devise as many unit tests as are needed to test the code.

We have provided two sets of classes for your project in C# and Java.   
Please choose one and write your test using MS-Test (for C#) or JUnit (for Java).

Unit tests should be about testing the units of code and not about testing the engagement with external dependencies such as databases (not Integration testing!).

The site makes extensive use of databases to store product details and to keep a record customer details (name, password, address…) and you therefore will need to make a simple Mock of this database using a mocking framework. Alternatively, you may choose to create a hand-written stub for the database.

The choice is yours, but you will need to justify the reason for your choice.

If you elect to create a mock, then you may choose any mocking framework, but it is highly advisable to choose either Moq (for C#) and Mokito (for Java).

* **The site allows users to register with the site with a valid username,   
  password and address.**
* **Usernames** must follow these rules:
  + Usernames are not case sensitive.
  + All white spaces such as tabs and spaces are removed by the UI and therefore there is no need to trim a username during your tests.
  + Must not be Null.
  + Should be greater than 2 characters.
  + Contain only alphabetic characters (no digits).
  + Must not contain illegal characters like **~** < **>** @ **%** $ **£**
  + Must be unique in the database of users.
* **Passwords** must follow these rules:
  + Passwords are case sensitive.
  + All white spaces such as tabs and space chars are removed by the UI and therefore there is no need to trim a password during your tests.
  + Must have at least one number.
  + Must not contain the following characters or sequence of characters  
    **< > ; = 'or' '--'**
  + Must contain at least one of these special character   
    **~ @ % $ £**
* The **address** is composed of three parts.
* House or flat number   
  which is a string with numeric format, but it may contain a dash character   
  like 11-15**.**
  + House number does not contain any other alphabetic character.
  + Address line
    - There are no restrictions currently set on the address line in terms of type of characters, but it must not be Null.
  + Post code
    - Although the provided code utilises regular expressions to validate UK postcodes, it may require testing due to the intricate nature of UK postcodes!
  + City name
    - Must be a recognised city within UK in which the company operates. The company is set up in a few cities and their names are recorded in a database. We only accept customers from these cities.
* **Existing users can login using their username and password.**
* If a user tries to login more than three time, then a **TooManyLoginTriesException** would result (see below about custom exceptions)

# **Custom Exceptions**

The provided starter code throws a generic Exception when it finds an invalid username, password etc. Please create and use custom exceptions like:

* **LoginException** is thrown if username or password is not valid
* **TooManyLoginTriesException** when a user tries to login more than **3** times
* **InvalidUKAddressException** when a user offers an incorrect   
  house number, city or post code.
* Feel free to create more to suit your application  
    
  **Tip:** Here is a simple code for the **LoginException** custom exception type:  
  **C#**

**public class LoginException : Exception {**

**public LoginException() : base("Incorrect user name or password"){**

**}**

**}**

**Java:  
public class LoginException extends Exception {**

**public LoginException(){  
 super("Incorrect user name or password")**

**}**

**}**

# **Unit tests**

* Please choose either C# or Java for creating your unit tests.
* You’re provided with the starter classes which you can include in your own project.  
  Complete projects are not provided for simplicity.

We have provided a minimal amount of code to get your project started. Feel free to enhance or modify these classes if you like or create extra classes.

* Your task is to unit test the login and registry classes and test their methods.
* You will create either mocks or write hand-written stubs for any dependency such as a database (see the **IUserAccout** interface)

# **Optional exercise: Backend code for purchasing a product and editing a shopping basket.**

**Objective:**

Please create a code (using OO classes) that allows customers to select a product and add it to their shopping basket. Customers should also be able to edit their basket and delete items. Furthermore, the code should be designed to increase the quantity of an item if it is purchased again.

To ensure the quality and reliability of the code, please consider writing unit tests concurrently as you develop this part. Alternatively, an even better practice would be to employ Test-Driven Development (**TDD**) techniques to ensure that your code meets the necessary requirements and passes all tests before proceeding with further development.

The eventual application will enable the user to select from a range of products on offer.

1. Users will have the chance to add items to their basket and enter the quantity which they required.
2. Users can view their shopping baskets and are then given the chance to edit the quantity ordered or remove an item from a basket

**\*\* End**