Ensek API Testing Strategy/Feedback

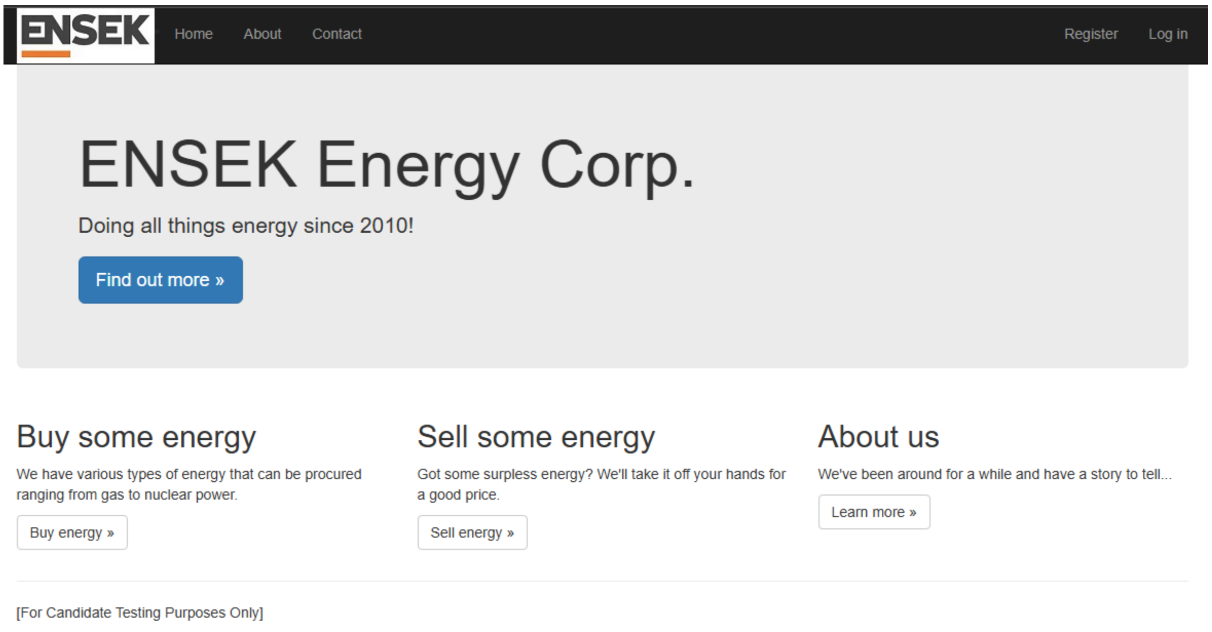
Below is a test strategy and my approach to testing the front end provided in the testing task. In this task, I will undertake exploratory testing, identify implemented functionality, and write a UI test to cover a basic test scenario.

Exploratory Testing

My first step in designing a test strategy for this application was to perform exploratory testing to discover what functionality currently exists within it and identify key areas to focus testing on for the rest of this task.

Upon opening the link the user has several buttons and options available to them, I started by selecting all the header links to determine which pages had been implemented and might require future testing. After this, I went through the remaining buttons presented to the user and noted what I found.

Home page - This button takes you to the landing page that is currently being viewed



ENSEK Energy Corp.

Doing all things energy since 2010!

Buy some energy

We have various types of energy that can be procured ranging from gas to nuclear power.

Sell some energy

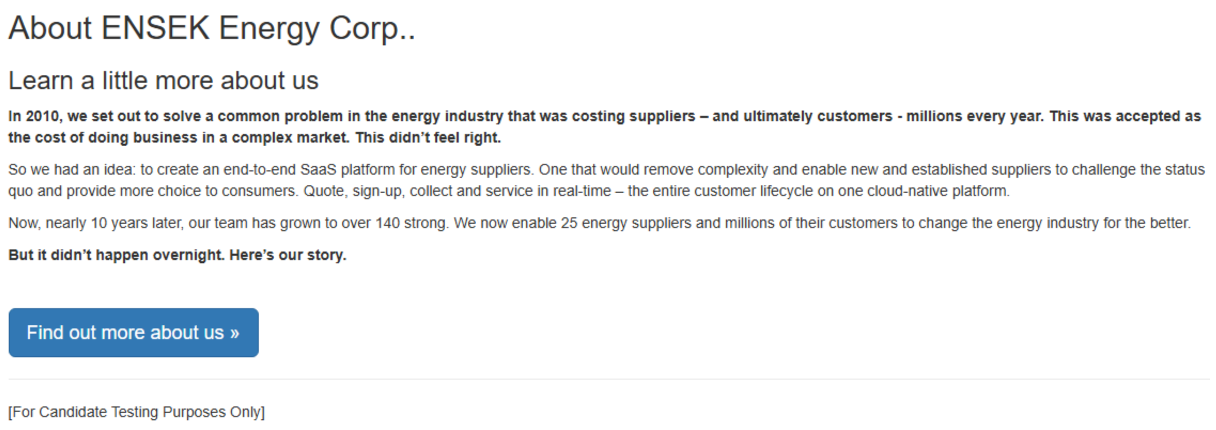
Got some surpless energy? We'll take it off your hands for a good price.

About us

We've been around for a while and have a story to tell...

[For Candidate Testing Purposes Only]

About button - This button takes you to the page in the picture below, on this page the only clickable button takes you to the main Ensek website which is not part of the testing scope for this task.



About ENSEK Energy Corp..

Learn a little more about us

In 2010, we set out to solve a common problem in the energy industry that was costing suppliers – and ultimately customers - millions every year. This was accepted as the cost of doing business in a complex market. This didn't feel right.

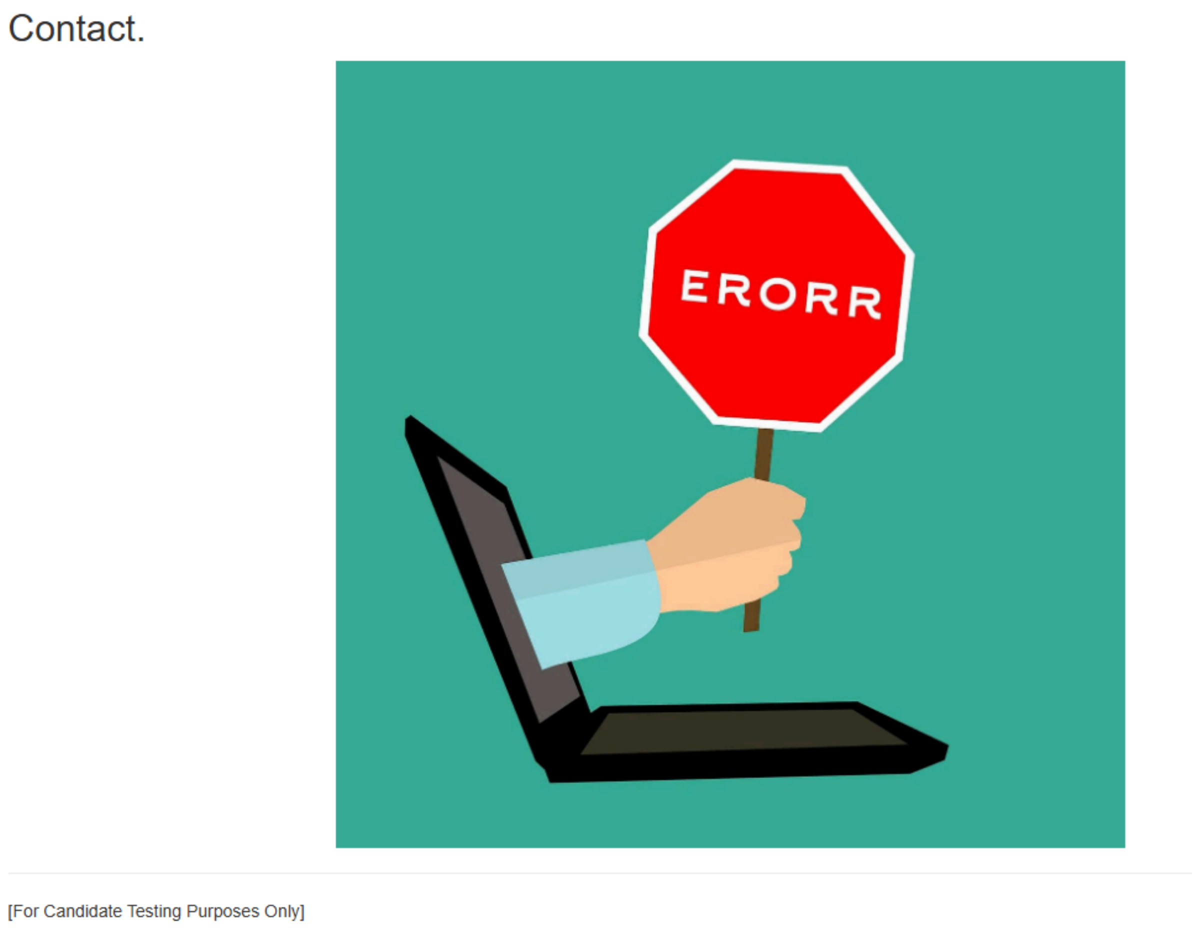
So we had an idea: to create an end-to-end SaaS platform for energy suppliers. One that would remove complexity and enable new and established suppliers to challenge the status quo and provide more choice to consumers. Quote, sign-up, collect and service in real-time – the entire customer lifecycle on one cloud-native platform.

Now, nearly 10 years later, our team has grown to over 140 strong. We now enable 25 energy suppliers and millions of their customers to change the energy industry for the better.

But it didn't happen overnight. Here's our story.

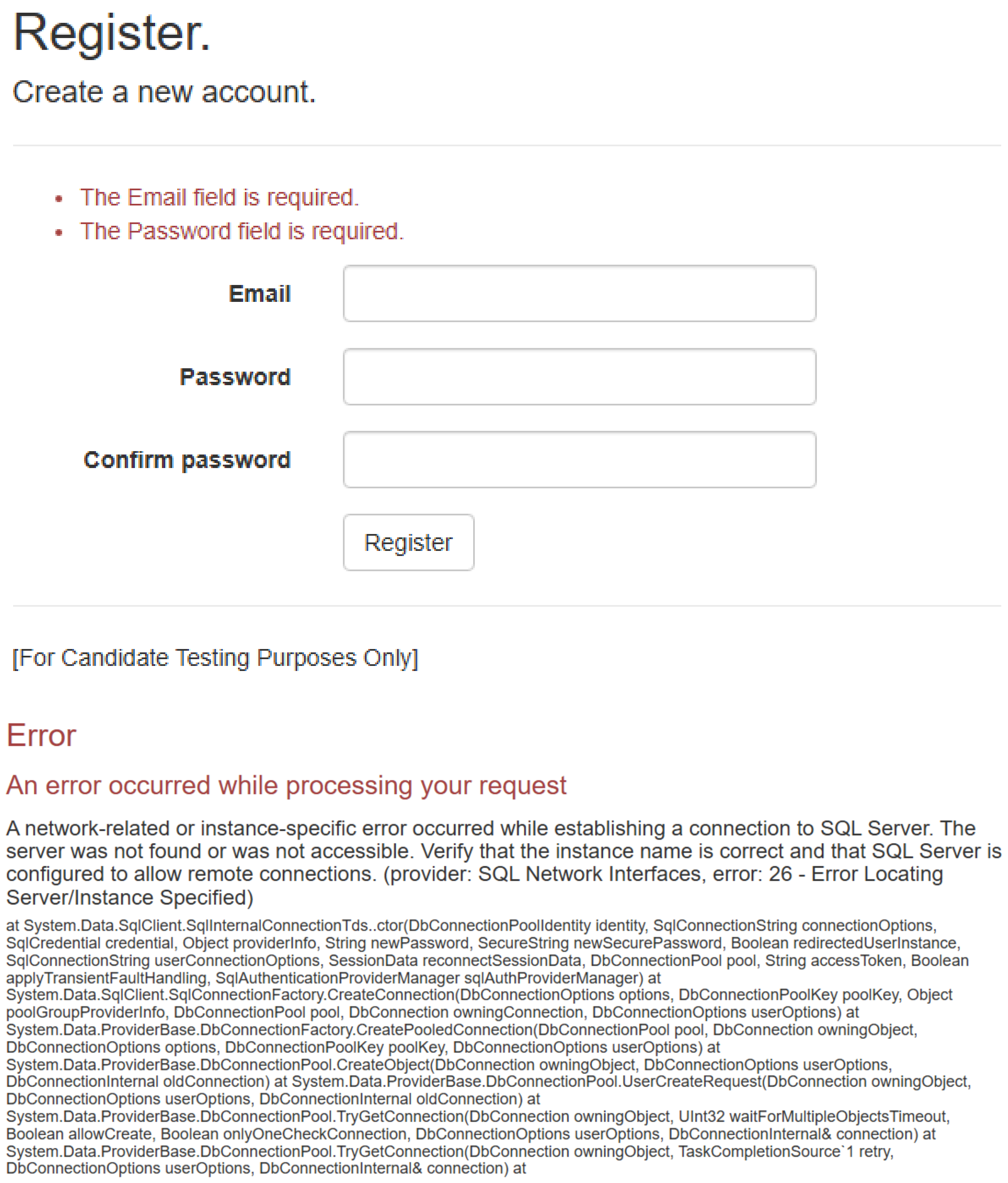
[For Candidate Testing Purposes Only]

Contact button - This button takes the user to a page that displays an error, this button should be removed (in a public-facing application) as it is either not implemented or currently broken.



[For Candidate Testing Purposes Only]

Register Button - This button takes the user to a registration form that has basic validation implemented requiring the user to input a valid email and a password that meets the security requirements set out. However, even if the form meets all the requirements the request is rejected with a 500 error and a big SQL error is shown on the screen. Ideally, these errors should not be shown to a user on a public-facing system.



Register.

Create a new account.

- The Email field is required.
- The Password field is required.

Email	
Password	
Confirm password	

[For Candidate Testing Purposes Only]

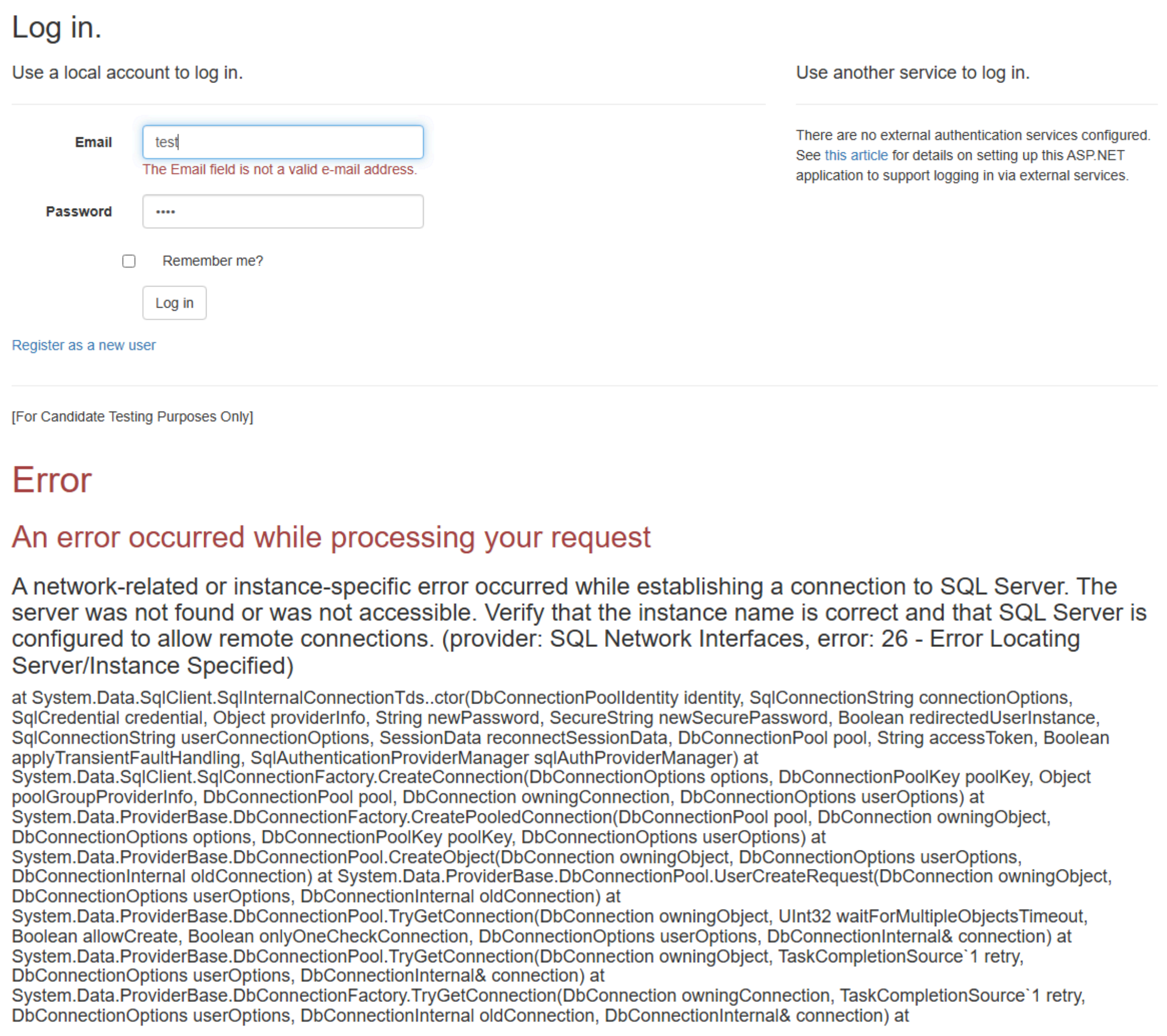
Error

An error occurred while processing your request

A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: SQL Network Interfaces, error: 26 - Error Locating Server/Instance Specified)

```
at System.Data.SqlClient.SqlInternalConnectionTds..ctor(DbConnectionPoolIdentity identity, SqlConnectionString connectionOptions,
SqlCredential credential, Object providerInfo, String newPassword, SecureString newSecurePassword, Boolean redirectedUserInstance,
SqlConnectionString userConnectionOptions, SessionData reconnectSessionData, DbConnectionPool pool, String accessToken, Boolean
applyTransientFaultHandling, SqlAuthenticationProviderManager sqlAuthProviderManager) at
System.Data.SqlClient.SqlConnectionFactory.CreateConnection(DbConnectionOptions options, DbConnectionPoolKey poolKey, Object
poolGroupProviderInfo, DbConnectionPool pool, DbConnection owningConnection, DbConnectionOptions userOptions) at
System.Data.ProviderBase.DbConnectionFactory.CreatePooledConnection(DbConnectionPool pool, DbConnection owningObject,
DbConnectionOptions options, DbConnectionPoolKey poolKey, DbConnectionOptions userOptions) at
System.Data.ProviderBase.DbConnectionPool.CreateObject(DbConnection owningObject, DbConnectionOptions userOptions,
DbConnectionInternal oldConnection) at System.Data.ProviderBase.DbConnectionPool.UserCreateRequest(DbConnection owningObject,
DbConnectionOptions userOptions, DbConnectionInternal oldConnection) at
System.Data.ProviderBase.DbConnectionPool.TryGetConnection(DbConnection owningObject, UInt32 waitForMultipleObjectsTimeout,
Boolean allowCreate, Boolean onlyOneCheckConnection, DbConnectionOptions userOptions, DbConnectionInternal& connection) at
System.Data.ProviderBase.DbConnectionPool.TryGetConnection(DbConnection owningObject, TaskCompletionSource`1 retry,
DbConnectionOptions userOptions, DbConnectionInternal& connection) at
```

Login Button - This button takes the user to a login form that also has basic validation for the email field. Similar to the register form when a request is submitted the user is presented with a SQL error. Ideally, these errors should not be shown to a user on a public-facing system.



The Email field is not a valid e-mail address.

There are no external authentication services configured. See this article for details on setting up this ASP.NET application to support logging in via external services.

[For Candidate Testing Purposes Only]

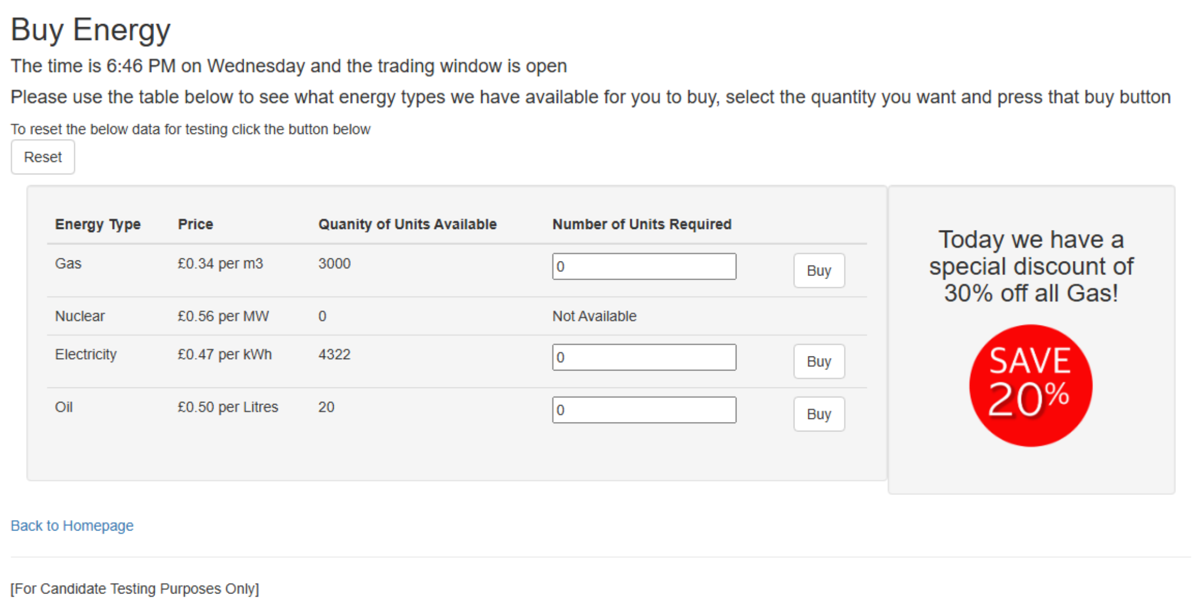
Error

An error occurred while processing your request

A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: SQL Network Interfaces, error: 26 - Error Locating Server/Instance Specified)

```
at System.Data.SqlClient.SqlInternalConnectionTds..ctor(DbConnectionPoolIdentity identity, SqlConnectionString connectionOptions,
SqlCredential credential, Object providerInfo, String newPassword, SecureString newSecurePassword, Boolean redirectedUserInstance,
SqlConnectionString userConnectionOptions, SessionData reconnectSessionData, DbConnectionPool pool, String accessToken, Boolean
applyTransientFaultHandling, SqlAuthenticationProviderManager sqlAuthProviderManager) at
System.Data.SqlClient.SqlConnectionFactory.CreateConnection(DbConnectionOptions options, DbConnectionPoolKey poolKey, Object
poolGroupProviderInfo, DbConnectionPool pool, DbConnection owningConnection, DbConnectionOptions userOptions) at
System.Data.ProviderBase.DbConnectionFactory.CreatePooledConnection(DbConnectionPool pool, DbConnection owningObject,
DbConnectionOptions options, DbConnectionPoolKey poolKey, DbConnectionOptions userOptions) at
System.Data.ProviderBase.DbConnectionPool.CreateObject(DbConnection owningObject, DbConnectionOptions userOptions,
DbConnectionInternal oldConnection) at System.Data.ProviderBase.DbConnectionPool.UserCreateRequest(DbConnection owningObject,
DbConnectionOptions userOptions, DbConnectionInternal oldConnection) at
System.Data.ProviderBase.DbConnectionPool.TryGetConnection(DbConnection owningObject, UInt32 waitForMultipleObjectsTimeout,
Boolean allowCreate, Boolean onlyOneCheckConnection, DbConnectionOptions userOptions, DbConnectionInternal& connection) at
System.Data.ProviderBase.DbConnectionPool.TryGetConnection(DbConnection owningObject, TaskCompletionSource`1 retry,
DbConnectionOptions userOptions, DbConnectionInternal& connection) at
System.Data.ProviderBase.DbConnectionFactory.TryGetConnection(DbConnection owningConnection, TaskCompletionSource`1 retry,
DbConnectionOptions userOptions, DbConnectionInternal oldConnection, DbConnectionInternal& connection) at
```

Buy Energy Button - This button takes the user to a simple buying form where the user is able to input a quantity of Energy to purchase. If the user orders something on this page they are redirected to a confirmation page where the order is confirmed. This is the most functionally complete part of the application so most of my testing should be focused around it.



Buy Energy

The time is 6:46 PM on Wednesday and the trading window is open

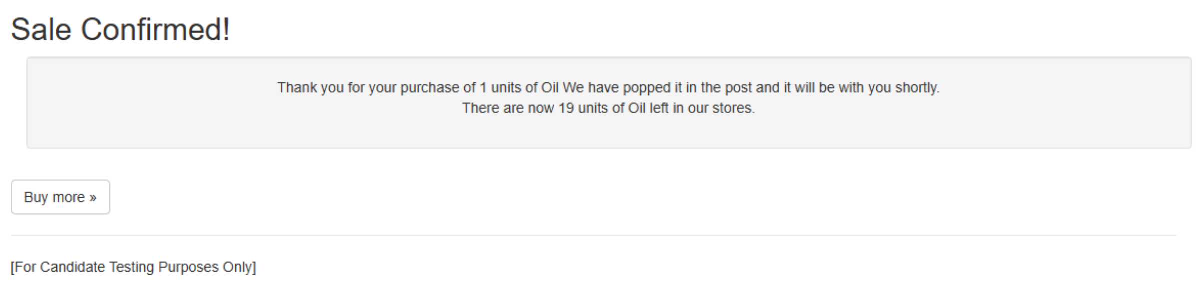
Please use the table below to see what energy types we have available for you to buy, select the quantity you want and press that buy button

To reset the below data for testing click the button below

Energy Type	Price	Quanity of Units Available	Number of Units Required	
Gas	£0.34 per m3	3000	0	Buy
Nuclear	£0.56 per MW	0	Not Available	
Electricity	£0.47 per kWh	4322	0	Buy
Oil	£0.50 per Litres	20	0	Buy

Today we have a special discount of 30% off all Gas!

[For Candidate Testing Purposes Only]

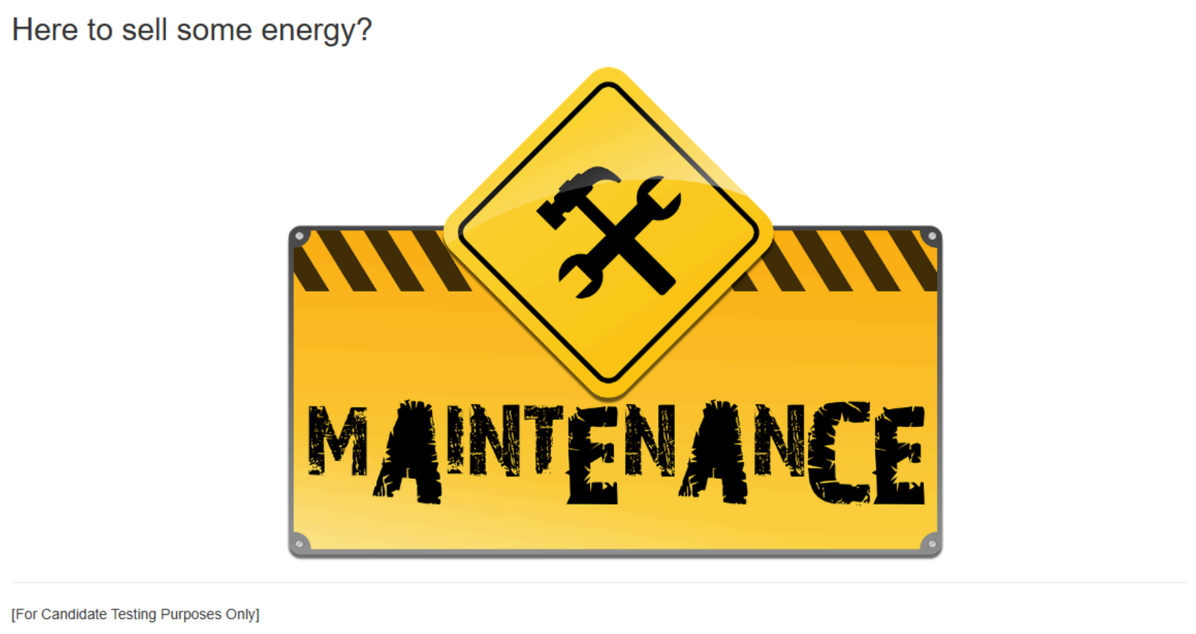


Sale Confirmed!

Thank you for your purchase of 1 units of Oil We have popped it in the post and it will be with you shortly.
There are now 19 units of Oil left in our stores.

[For Candidate Testing Purposes Only]

Sell Energy Button - This button takes the user to a different page where they are presented with a maintenance image. This functionality is clearly not implemented yet so will not be the focus of my testing efforts.



Here to sell some energy?

[For Candidate Testing Purposes Only]

About Us Button (main page) - This button is functionally identical to the button that can be used in the headers.

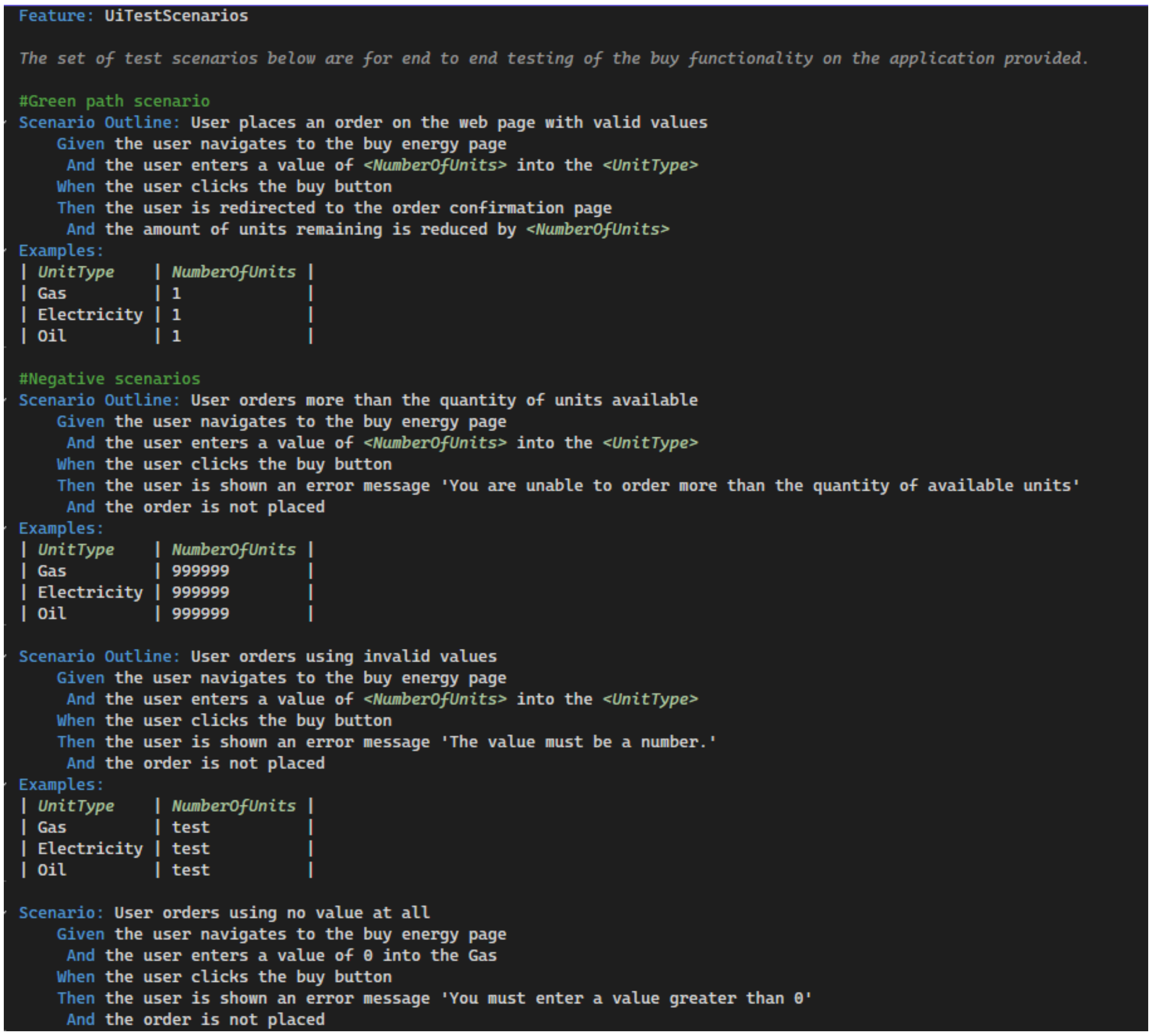
<u>Exploratory Testing Conclusion</u>

After exploring all of the options on the test website I can conclude that the majority of my testing efforts (given the time constraints) should be focused on the testing of the buy energy page as it is the most functionally complete part of the application.

Creating Testing Scenarios

The next step in my testing process was to create test scenarios for the buying functionality of the application. This involved outlining all the green path scenarios (things the user should be able to do) as well as outlining all of the negative scenarios (what the user should be unable to do). I wrote my test scenarios in BDD (or Gherkin) as I would for any feature that was being developed. These scenarios are usually written at the conception part of the feature development so that the developer knows what test scenarios the feature must pass to be considered complete.

Below are the scenarios I wrote, I will only be implementing one of the test scenarios in the UI test framework as a proof of concept.



```
Feature: UiTestScenarios

The set of test scenarios below are for end to end testing of the buy functionality on the application provided.

#Green path scenario
Scenario Outline: User places an order on the web page with valid values
    Given the user navigates to the buy energy page
     And the user enters a value of <NumberOfUnits> into the <UnitType>
    When the user clicks the buy button
    Then the user is redirected to the order confirmation page
     And the amount of units remaining is reduced by <NumberOfUnits>
Examples:
| UnitType    | NumberOfUnits |
| Gas         | 1             |
| Electricity | 1             |
| Oil         | 1             |

#Negative scenarios
Scenario Outline: User orders more than the quantity of units available
    Given the user navigates to the buy energy page
     And the user enters a value of <NumberOfUnits> into the <UnitType>
    When the user clicks the buy button
    Then the user is shown an error message 'You are unable to order more than the quantity of available units'
     And the order is not placed
Examples:
| UnitType    | NumberOfUnits |
| Gas         | 999999        |
| Electricity | 999999        |
| Oil         | 999999        |

Scenario Outline: User orders using invalid values
    Given the user navigates to the buy energy page
     And the user enters a value of <NumberOfUnits> into the <UnitType>
    When the user clicks the buy button
    Then the user is shown an error message 'The value must be a number.'
     And the order is not placed
Examples:
| UnitType    | NumberOfUnits |
| Gas         | test          |
| Electricity | test          |
| Oil         | test          |

Scenario: User orders using no value at all
    Given the user navigates to the buy energy page
     And the user enters a value of 0 into the Gas
    When the user clicks the buy button
    Then the user is shown an error message 'You must enter a value greater than 0'
     And the order is not placed
```

<u>**Creating a UI test for the Green Path Scenario**</u>

As the final part of my testing, I will be creating a simple green path test using Selenium and C#. Usually, I would link the steps in this test with the BDD scenarios I have written above but for the sake of simplicity, I will not be doing that and writing it purely in C# code. The code for this task is in GitHub and can be run if the code is pulled and built on a local machine.

<u>Observations</u>

Whilst writing the test I noticed a lack of unique identifiers on the HTML elements on the page. This means a tester would have to use CSS selection and the FindByText function to select the element on the screen. This is generally poor practice as it is possible to have multiple page elements with the same text or CSS class. Instead, I would add ‘data-testid’ tags onto the HTML elements to improve reliability.

I have used Xpaths to select certain elements on the buy page like the buy buttons, this is not a reliable method of selecting elements so unique ID’s should be added instead to avoid breaking the tests if elements are moved or changed.

The ID’s that are supplied to the text fields on the buy page are also poor as they are not unique meaning you have to select which element you want.