First off, this creation was born in an air-gapped network with no access to the page under test. However, this is not a problem as I was able to inspect the page with Firepath on an internet enable machine.

If you open up the sln in Visual Studio and build it, it should update all the missing references automatically from Nuget. Additionally if you install the Chromedriver in the location specified in the BeforeFeature method of the StartStop.cs.

The methodology of element mapping I have used here, enables the SDET to create the Gherkin style test scenarios and the coded step definitions without the need for direct access to the page under test. I have spent a great deal of time over the last 3 years working on sites that offer large and complex questionnaires, consequently there is a great deal of branching of questions and so elements may not be available all the time, especially if the page model is loaded at run time.

The element location details (CSS in this case) are stored in a separate file and the files contents are referenced as the element under test is interacted upon. Whilst in this version the entire file is read and stored as part of the BeforeFeature action, I would normally read the single element details as and when required as so keeping the resource footprint of the test tool to a minimum.

Whilst in the solution I have referenced each summary row and it’s details as individual mapped items, I would normally have performed the “findElement” action on a dynamically created CssSelector value. The row values all end “-1, -2, -3” etc. My passing in through the Gherkin statement the row number, this could be used to create the CssSelector data-test value dynamically as so reduce the mapping required to 1 set rather than 5, this is also true of the details section (hourly details), reducing by a similar ratio (1 set of 8 hourly increments)

It was refreshing to the use of a custom attribute in the page (data-test). This type of design for test attribute is so sorely missing in many of todays development projects. From my own experience when you find such implementations it usually means that development have bought into the test automation process and are working hand in hand with the tests (either in a scrum team or outwith).

Another enhancement I would like to see in this type of testing is the levering of webservices. With all forms of testing there must be a point at which you must decide something is deemed to be correct. In this case the stored data and the webservices implemention that is used to populate the web page under test. In previous projects I have used the webservices and the returned data (in Json format) to help validate the data displayed on the screen. Assuming that the structure of the returned data is in a set format it should be fairly easy to parse the Json data and use it to confirm that the page is displaying values in the correct order in the correct fields. I have used this previously to confirm pages created from Content Delivery systems as well as the data received through 3rd party interfaces / api’s

Would also be better if the page was up to date as the date does not seem to change.

Type forward on the City field would be useful for the user as there is no indication as to whether the inputted City is valid until either the error message or valid data is displayed.

Certain elements on the page like the wind direction svg would require further discussions with development to establish an easier way to manage their direction.

Obviously I would look to tying the test solution into some form of CI (Jenkins or TeamCity for example). Not just to trigger the tests off the back of a successful dev deployment but also the deployment of the test environment.

Reporting is also another area that would require enhancing. Html Nunit/Junit style reports are ok at a high level but when the test scenarios contain significant details test reporting should be enhanced, possibly writing scenario results to a database.

Some form of test data management should be implemented.

Beyond the scope of the test I would have also considered the following options.

* Multi browser testing
* Multi OS testing
* Mobile/tablet testing (responsive web) I did have a look on a couple of Samsung mobiles and the site does appear to responsive to screen size changes etc.
* Interface testing. The data does seem to come from an 3rd party source
* Load and Performance testing (minimum owasp top 10)
* AA testing minimum although a weather website would be used by the entire population spectrum for AAA would be preferable.

… Could talk for hours on this!!!