Test Plan

Introduction

The Test Plan has been created to communicate the test approach to team members. It includes the scope, assumptions/risks, automation tools and the framework architecture. This document will clearly identify what the test deliverables will be and what is deemed in and out of scope.

1. Scope

The webpage that is to be tested is: https://the-internet.herokuapp.com/challenging_dom. At the end of phase one, a tester must have:

1. Written at least 10 automated test cases, to test the webpage UI.
2. Created a test automation framework using open source tools that is scalable and maintainable.
3. Implemented an automated report with failure screenshots and video records.
4. Integrated the test automation framework into a CI pipeline.

API testing is not in scope as they have not been developed yet and should be tested in isolation first, however there are placeholders in the UI that can be tested. Non functional testing, such as Visual testing, is not in scope for phase one.

2. Assumptions/Risks

2.1 Assumptions

1. The bug that is in the url path, is a known bug that has been fixed. Testers will use a work around for now, but test will be written so that when the bug fix is implemented the framework can be updated quickly to reflect the change.
2. The button labels are made up of a random combination of four words, qux, foo, baz, bar. These labels will change every time one of the buttons is clicked.
3. The functionality around the edit/delete buttons, has not been implemented yet, but a placeholder has. This will be tested until the functionality in the url path has been implemented.
4. Visual testing is not in scope for automation in phase one, due to the changes on the UI. Automation will check that the answer exists, but cannot check its format due to it being a canvas element.
5. The scope is limited to the https://the-internet.herokuapp.com/challenging_dom, third party pages are not to be tested.

2.2 Risks

1. The webpage has not been developed in consultation with automation testers, therefore the page has not been built with test automation in mind. This means that unique element locators may be difficult to find. This can lead to flaky tests and false fails.

2. The webpage has not had all functionality (API) implemented yet. Tests will have to be extended when this is implemented.
3. The requirements have not been made available to the automation testers, so the automated test cases are to be developed using the testers experience of the webpage.
4. Scope creep, there are third party webpages that are linked to the webpage, these are not in scope.
5. Automated testing does not replace manual testing. Automated tests check business flows and carry out smoke/regression tests but manual exploratory testing will also be required.

3. Tools/Architecture

3.1 Tools

The tool that has been chosen for test automation is cypress.io. This is a very powerful all in one, open source framework that is written in javascript. The main reason for selecting cypress is that it runs in the browser, in the same loop as the application. It allows for greater control over the DOM and enables testers to use query for element location. This will help testers overcome the problem of flaky tests due to the application being developed without test automation in mind. Cypress also has a large degree of documentation to aid testers in its use.

There are some limitations with cypress.io however. Cypress only supports a limited number of browsers, chrome, Firefox and electron. It also only supports javascript, meaning testers must have a knowledge of this language.

The CI tool that has been elected is Jenkins. It was selected because of its ease of use, configurability and open source.

3.2 Architecture

The architectural pattern that will be used in the framework is page object pattern. This has been chosen for two main reasons:

1. **Code reusability** – The same page class can be used in different tests and all the locators and their methods can be re-used across various test cases. Cypress also allows the tester to create re-useable command for standard elements, eg. tables. These are not page specific and can be re-used many times in the framework.
2. **Code maintainability** – There is a clean separation between test code which can be our functional scenarios and page-specific code such as locators and methods. So, if some structural change happens on the web page, it will only impact the page object and will not have any impact on the test scripts.

4. Test Environment

The application has already been pushed to production, so the automation will have to be run in a production environment.

5. Test Descriptions

Requirement	Requirement Description	Testcase ID
A	Each button (button, button alert, button submit), changes the button labels.	TC002, TC003, TC004
B	Table is formatted in 10 rows, with buttons to edit/delete each row.	TC005, TC006, TC007, TC008, TC009, TC010, TC011, TC012, TC013, TC014
C	The Answer is present on the webpage.	TC001