Test Matrix

QA Course Project (Front End)

21/05/2022

Ligia Samara Diaz Hirashi

Version: 1.0

Created: 21/05/2022

Last Updated: 4/07/2022

Location: Monterrey, N.L., México.

Revision and Sign-off Sheet

Document History-

Version	Date	Author	Description of Change
1	20/05/2022	Samara Diaz Hirashi	Draft

Reference Documents-

Version	Date	Document Name
1.0	19/05/2022	QA Course Project (Front End)

1. Test Cases

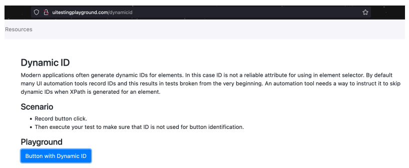
1.1. Dynamic ID (QPK-227)

1.1.1 Use Case | Simple Dynamic ID

“As a tester I want to make sure that I can design and implement a test on the UI Testing Playground website using a clickable button without using a dynamic ID to make sure dynamic IDs of elements are not being recorded.”

1.1.2 Test Case | Dynamic ID

Description:
“Modern applications often generate dynamic IDs for elements. In this case, ID is not a reliable attribute for using in element selector. By default many UI automation tools record IDs and this results in tests broken from the very beginning. An automation tool needs a way to instruct it to skip dynamic IDs when XPath is generated for an element. Record button click and then execute your test to make sure that ID is not used for button identification.”

Summary	Steps	Expected Result	Actual Result	Status
The test should record a button click making sure that dynamic IDs are not being used in the test process.	1.- Go to http://uitestingplayground.com/dynamicid. 2.- Check if there’s a button on page. 3.- Click the button. 4.- Verify that the dynamic ID of the button is not being used for button identification.	There should be a button on the page, it should be clickable and no dynamic IDs should be used while testing its functionality.	There was a clickable button on the website and its dynamic ID was not used while testing.  uitestingplayground.com/dynamicid Resources Dynamic ID Modern applications often generate dynamic IDs for elements. In this case ID is not a reliable attribute for using in element selector. By default many UI automation tools record IDs and this results in tests broken from the very beginning. An automation tool needs a way to instruct it to skip dynamic IDs when XPath is generated for an element. Scenario • Record button click. • Then execute your test to make sure that ID is not used for button identification. Playground Button with Dynamic ID	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/sCJSX

1.2. Class Attribute (QPK-228)

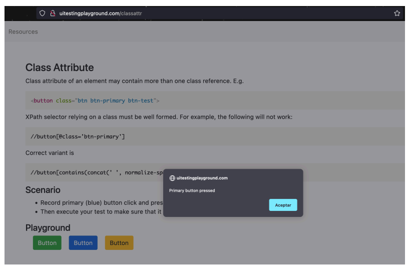
1.2.1. Use Case | Simple Class Attribute

“As a tester I want to be able to design and implement a test on the UI Testing Playground website using a clickable button identifying it by a class attribute to verify that the class attribute based XPath is well formed.”

1.2.2. Test Case | Class Attribute

Description:
“Class attributes of an element may contain more than one class reference, XPath selector relying on a class must be well formed. Execute a test making sure that it can identify a button using btn-primary class. Record primary(blue) button click and press ok in alert popup and then execute your test to make sure that it can identify the button using btn-primary class.”

Summary	Steps	Expected Result	Actual Result	Status

The test should record a button click and press ok in alert popup using a class attribute to identify it.	1.- Go to http://uitestingplayground.com/classattr.
	2.- Check if there’s a button on page.
	3.- Click the primary (blue) button.
	4.- Validate that a popup appears.
	5.- Press ok in alert popup.
	6.- Verify that the button is being correctly identified by it’s class attribute.
		There should be a button on the page, it should be clickable and it should display an alert popup when clicked. Class attribute of the button should be used to identify it.	There’s a clickable button on the page that displays an alert popup when clicked. The identification of this when testing button is done by class name.
			
			uitestingplayground.com/classattr
			Resources
			Class Attribute
			Class attribute of an element may contain more than one class reference. E.g.
			`<button class="btn btn-primary btn-test">`
			XPath selector relying on a class must be well formed. For example, the following will not work:
			`//button[@class='btn-primary']`
			Correct variant is
			`//button[contains(concat(' ', normalize-sp`
			Scenario
			• Record primary (blue) button click and pres
			• Then execute your test to make sure that it
			Playground
			uitestingplayground.com
			Primary button pressed
			
				Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/vGX56

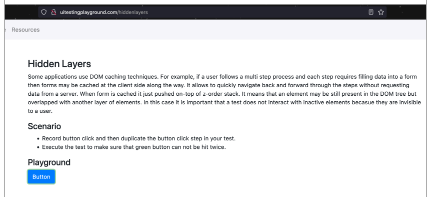
1.3. Hidden Layers (QPK-230)

1.3.1. Use Case | Hidden Layers

“As a tester I want to design and implement a test on the UI Testing Playground website using a clickable button that can only be clicked once to verify that the test does not interact with invisible elements.”

1.3.2. Test Case | Hidden Layers

Description:
“Some applications use DOM caching techniques. For example, if a user follows a multi-step process and each step requires filling data into a form then forms may be cached at the client side along the way. It allows you to quickly navigate back and forward through the steps without requesting data from a server. When the form is cached it's just pushed on top of the z-order stack. It means that an element may be still present in the DOM tree but overlapped with another layer of elements. In this case, it is important that a test does not interact with inactive elements because they are invisible to a user. Record button click and then duplicate the button step in your test and then execute the test to make sure that green button can not be hit twice.”

Summary	Steps	Expected Result	Actual Result	Status
The test should record a button click and duplicate it to make sure that it cannot be pressed twice.	1.- Go to http://uitestingplayground.com/hiddenlayers. 2.- Check if there’s a button on page. 3.- Click the button. 4.- Validate that it changes color to blue. 5.- Click on the button. 6.- Verify that the button is still blue.	There should be a button on the page, it should be clickable and it should change color when clicked, when the click is duplicated it shouldn’t change color again.	There’s a clickable button on the page that turns to blue when clicked for the first time.  uitestingplayground.com/hiddenlayers Resources Hidden Layers Some applications use DOM caching techniques. For example, if a user follows a multi step process and each step requires filling data into a form then forms may be cached at the client side along the way. It allows to quickly navigate back and forward through the steps without requesting data from a server. When form is cached it just pushed on-top of z-order stack. It means that an element may be still present in the DOM tree but overlapped with another layer of elements. In this case it is important that a test does not interact with inactive elements becasue they are invisible to a user. Scenario • Record button click and then duplicate the button click step in your test. • Execute the test to make sure that green button can not be hit twice. Playground Button	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/deV79

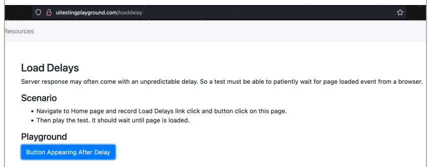
1.4. Load Delay (QPK-231)

1.4.1. Use Case | Load Delay

“As a tester I want to design and implement a test on the UI Testing Playground website using a link that sends me to another website with an unpredictable delay to ensure that the test is capable of waiting for a page to load before clicking an element.”

1.4.2. Test Case | Load Delay

Description:
“Server response may often come with an unpredictable delay. So a test must be able to patiently wait for a page loaded event from a browser. Navigate to Home page and record Load Delay link click and button click on this page and then play the test, it should wait until page is loaded.”

Summary	Steps	Expected Result	Actual Result	Status
The test should click a link to another page and click another button once the page is loaded.	1.- Go to http://uitestingplayground.com. 2.- Check if there’s a “Load Delays” link on the page. 3.- Click the link. 4.- Validate that the link redirects to http://uitestingplayground.com/loaddelay. 5.- Click on the button. 6.- Verify that the test waits until the page is loaded to click on the button.	There should be a link to “Load Delays” on the page that takes you to the “Load Delays” page which contains a button that should be clicked only after the website is loaded.	There’s a link to “Load Delays” on the main page that redirects to “Load Delays” which contains a clickable button.  uitestingplayground.com/loaddelay Resources Load Delays Server response may often come with an unpredictable delay. So a test must be able to patiently wait for page loaded event from a browser. Scenario • Navigate to Home page and record Load Delays link click and button click on this page. • Then play the test. It should wait until page is loaded. Playground Button Appearing After Delay	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/tuvCH

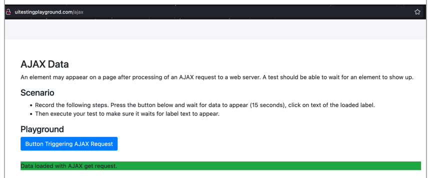
1.5. AJAX Data (QPK-232)

1.5.1. Use Case | AJAX Data

“As a tester I want to design and implement a test on the UI Testing Playground website that clicks a button and waits 15 seconds for data to appear then clicks on a label to ensure that some elements appear on the page after loading data with an AJAX Request.”

1.5.2. Test Case | AJAX Data

Description:
“An element may appear on a page after processing of an AJAX request to a web server. A test should be able to wait for an element to show up. Record the following steps and press the button below and wait for data to appear (15 seconds), click on the text of the loaded label and then execute your test to make sure it waits for label text to appear.”

Summary	Steps	Expected Result	Actual Result	Status
The test should click on a button and wait 15 seconds for data to appear then click on a label.	1.- Go to http://uitestingplayground.com/ajax. 2.- Check if there’s a button. 3.- Click the button. 4.- Wait 15 seconds. 5.- Validate that new data appears. 6.- Click on the text of the loaded label.	There should be a button on the website and after 15 seconds of clicking it, new data should appear.	There’s a clickable button on the website and after clicking it, new data appears.  uitestingplayground.com/ajax AJAX Data An element may appaear on a page after processing of an AJAX request to a web server. A test should be able to wait for an element to show up. Scenario • Record the following steps. Press the button below and wait for data to appear (15 seconds), click on text of the loaded label. • Then execute your test to make sure it waits for label text to appear. Playground Button Triggering AJAX Request Data loaded with AJAX get request.	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/opruP

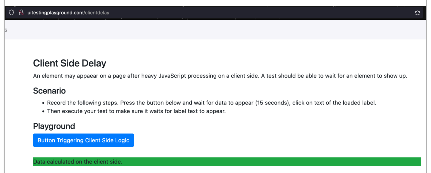
1.6. Client Side Delay (QPK-233)

1.6.1. Use Case | Client Side Delay

“As a tester I want to design and implement a test on the UI Testing Playground website that clicks a button and waits 15 seconds for data to appear then clicks on a label to ensure that some elements appear on the page after heavy JavaScript processing on a client side.”

1.6.2. Test Case | Client Side Delay

Description:
“An element may appear on a page after heavy JavaScript processing on a client-side. A test should be able to wait for an element to show up. Record the following steps, press the button and wait for data to appear (15 seconds), and click on the text of the loaded label. Then execute your test to make sure it waits for label text to appear.”

Summary	Steps	Expected Result	Actual Result	Status
The test should click on a button and wait 15 seconds for data to appear then click on a label.	1.- Go to http://uitestingplayground.com/clientdelay. 2.- Check if there’s a button. 3.- Click the button. 4.- Wait 15 seconds. 5.- Validate that new data appears. 6.- Click on the text of the loaded label.	There should be a button on the website and after 15 seconds of clicking it, new data should appear.	There’s a clickable button on the website and new data appears 15 seconds after clicking it.  uitestingplayground.com/clientdelay Client Side Delay An element may appaear on a page after heavy JavaScript processing on a client side. A test should be able to wait for an element to show up. Scenario • Record the following steps. Press the button below and wait for data to appear (15 seconds), click on text of the loaded label. • Then execute your test to make sure it waits for label text to appear. Playground Data calculated on the client side.	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/ol178

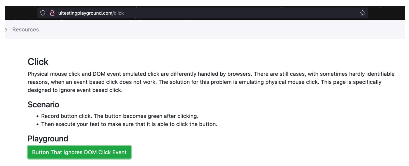
1.7. Click (QPK-234)

1.7.1. Use Case | Click

“As a tester I want to design and implement a test on the UI Testing Playground website that emulates a button click and verifies that it ignores a DOM click event.”

1.7.2. Test Case | Click

Description:
“Physical mouse click and DOM event emulated click are differently handled by browsers. There are still cases, with sometimes hardly identifiable reasons, when an event based click does not work. The solution for this problem is emulating physical mouse clicks. This page is specifically designed to ignore event based clicks. Record button click and the button becomes green after clicking then execute your test to make sure that it is able to click the button.”

Summary	Steps	Expected Result	Actual Result	Status
The test should emulate a button click and make sure that the test ignores the DOM click event.	1.- Go to http://uitestingplayground.com/click. 2.- Check if there’s a button. 3.- Click the button. 5.- Validate that the button changes color. 6.- Verify that the test ignores the DOM click event.	There should be a button on the website and it should emulate a button click that changes the color of the button without using a DOM Click Event.	There’s a clickable button on the website that changes color when clicked without a DOM event.  uitestingplayground.com/click Resources Click Physical mouse click and DOM event emulated click are differently handled by browsers. There are still cases, with sometimes hardly identifiable reasons, when an event based click does not work. The solution for this problem is emulating physical mouse click. This page is specifically designed to ignore event based click. Scenario • Record button click. The button becomes green after clicking. • Then execute your test to make sure that it is able to click the button. Playground Button That Ignores DOM Click Event	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/admxC

1.8. Text Input (QPK-235)

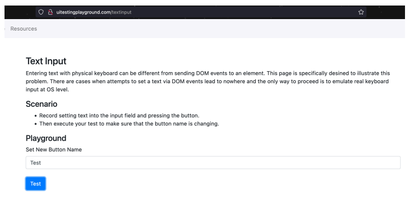
1.8.1. Use Case | Text Input

“As a tester I want to design and implement a test on the UI Testing Playground website that enters text into an edit field to change a button name.”

1.8.2. Test Case | Text Input

Description:

“Entering text with a physical keyboard can be different from sending DOM events to an element. This page is specifically designed to illustrate this problem. There are cases when attempts to set a text via DOM events lead to nowhere and the only way to proceed is to emulate real keyboard input at the OS level. Record-setting text into the input field and pressing the button and then execute your test to make sure that the button name is changing.”

Summary	Steps	Expected Result	Actual Result	Status
The test should record setting text into the input field and pressing the button to change the button’s name.	1.- Go to http://uitestingplayground.com/textinput. 2.- Check if there’s a button. 3.- Check if there’s a text input field. 4.- Write text on the input field. 5.- Click the button. 6.- Verify that the button changes its text.	There should be a button and a text field on the website and the button should change its name to the one introduced into the text field.	There’s a clickable button on the website and an input field. The button changes its text to the one in the input field when clicked.  uitestingplayground.com/textinput Resources **Text Input** Entering text with physical keyboard can be different from sending DOM events to an element. This page is specifically desined to illustrate this problem. There are cases when attempts to set a text via DOM events lead to nowhere and the only way to proceed is to emulate real keyboard input at OS level. **Scenario** • Record setting text into the input field and pressing the button. • Then execute your test to make sure that the button name is changing. **Playground** Set New Button Name Test Test	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/lLW14

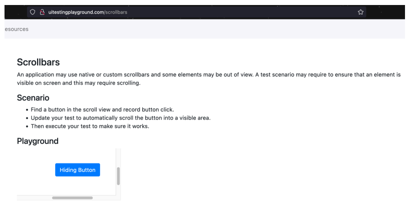
1.9. Scrollbars (QPK-240)

1.9.1. Use Case | Scrollbars

“As a tester I want to design and implement a test on the UI Testing Playground website that finds a button using a scrollbar and clicks on it.”

1.9.2. Test Case | Scrollbars

Description:
“An application may use native or custom scrollbars and some elements may be out of view. A test scenario may require to ensure that an element is visible on screen and this may require scrolling. Find a button in the scroll view and record the button click. Update your test to automatically scroll the button into a visible area and then execute your test to make sure it works.”

Summary	Steps	Expected Result	Actual Result	Status
The test should find a button while scrolling and click on it.	1.- Go to http://uitestingplayground.com/scrollbars. 2.- Use the scrollbars to find a button. 3.- Click on the button.	There should be scrollbars and a clickable button.	There are scrollbars and a clickable button on the page.  uitestingplayground.com/scrollbars esources Scrollbars An application may use native or custom scrollbars and some elements may be out of view. A test scenario may require to ensure that an element is visible on screen and this may require scrolling. Scenario • Find a button in the scroll view and record button click. • Update your test to automatically scroll the button into a visible area. • Then execute your test to make sure it works. Playground 	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/bktuN

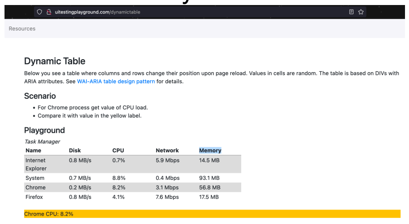
1.10. Dynamic Table (QPK-242)

1.10.1. Use Case | Dynamic Table

“As a tester I want to design and implement a test on the UI Testing Playground website that gets the value of a CPU Load and compares it with a text.”

1.10.2. Test Case | Dynamic Table

Description:
“Below you see a table where columns and rows change their position upon page reload. Values in cells are random. The table is based on DIVs with ARIA attributes. See WAI-ARIA table design pattern for details. For Chrome process the value of the CPU load and compare it with the value in the yellow label.”

Summary	Steps	Expected Result	Actual Result	Status
The test should get the value of a CPU Load and compare it with a text.	1.- Go to http://uitestingplayground.com/dynamictable. 2.- Get the value of a CPU Load. 3.- Compare it with the value in the yellow label.	There should be a table with CPU values and a text in a yellow label.	There’s a table with CPU values and a text in a yellow label. 	Pass ✅

uitestingplayground.com/dynamictable

Resources

Dynamic Table

Below you see a table where columns and rows change their position upon page reload. Values in cells are random. The table is based on DIVs with ARIA attributes. See WAI-ARIA table design pattern for details.

Scenario

- For Chrome process get value of CPU load.
- Compare it with value in the yellow label.

Playground

Task Manager

Name	Disk	CPU	Network	Memory
Internet Explorer	0.8 MB/s	0.7%	5.9 Mbps	14.5 MB
System	0.7 MB/s	8.8%	0.4 Mbps	93.1 MB
Chrome	0.2 MB/s	8.2%	3.1 Mbps	56.8 MB
Firefox	0.8 MB/s	4.1%	7.6 Mbps	17.5 MB

Chrome CPU: 8.2%

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/epyTU

1.11. Verify Text (QPK-243)

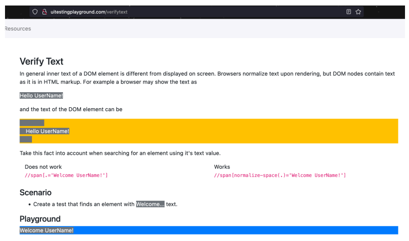
1.11.1. Use Case | Verify Text

“As a tester I want to design and implement a test on the UI Testing Playground website that finds an element with a “Welcome…” text.”

1.11.2. Test Case | Verify Text

Description:

“In general the inner text of a DOM element is different from that displayed on screen. Browsers normalize text upon rendering, but the DOM nodes contain text as it is in HTML markup. Create a test that finds an element with “Welcome…” text.”

Summary	Steps	Expected Result	Actual Result	Status
The test should find an element with “Welcome” text.	1.- Go to http://uitestingplayground.com/verifytext. 2.- Verify that there’s an element on the page. 3.- Validate that the element contains “Welcome…”.	There should be an element on the page that contains “Welcome…”.	There’s an element on the page that contains “Welcome” but its missing “…”.  uitestingplayground.com/verifytext Resources Verify Text In general inner text of a DOM element is different from displayed on screen. Browsers normalize text upon rendering, but DOM nodes contain text as it is in HTML markup. For example a browser may show the text as Hello UserName! and the text of the DOM element can be Hello UserName! Take this fact into account when searching for an element using it's text value. Does not work `//span[.='Welcome UserName!']` Works `//span[normalize-space(.)='Welcome UserName!']` Scenario • Create a test that finds an element with Welcome... text. Playground Welcome UserName!	Fail 🚫

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/lDNVW

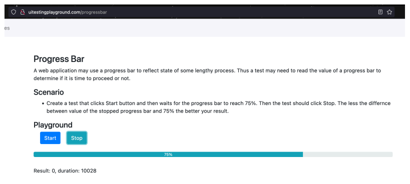
1.12. Progress Bar (QPK-245)

1.12.1. Use Case | Progress Bar

“As a tester I want to design and implement a test on the UI Testing Playground website that clicks a button to start a progress bar and then stops it at 75%.”

1.12.2. Test Case | Progress Bar

Description:
“A web application may use a progress bar to reflect state of some lengthy process. Thus a test may need to read the value of a progress bar to determine if it is time to proceed or not. Create a test that clicks the start button and then waits for the progress bar to reach 75%. Then the test should click stop. The less the difference between value of the stopped progress bar and 75% the better your result.”

Summary	Steps	Expected Result	Actual Result	Status
The test should emulate the click of a button to start a progress bar and then click another button to stop the progress at 75%.	1.- Go to http://uitestingplayground.com/progressbar. 2.- Verify that there are two buttons on the page and a progress bar. 3.- Click on the start button. 4.- Click on the stop button.	There should be two buttons and a progress bar on the site and the tester should be able to stop the progress at 75%.	There are two buttons and a progress bar on the website and the progress was stopped at 75%.  uitestingplayground.com/progressbar Progress Bar A web application may use a progress bar to reflect state of some lengthy process. Thus a test may need to read the value of a progress bar to determine if it is time to proceed or not. Scenario • Create a test that clicks Start button and then waits for the progress bar to reach 75%. Then the test should click Stop. The less the differnce between value of the stopped progress bar and 75% the better your result. Playground Result: 0, duration: 10028	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/koyJU

1.13. Visibility (QPK-246)

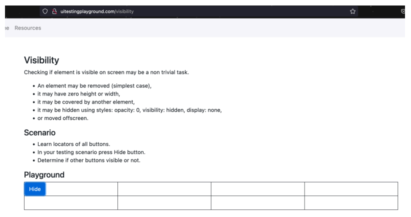
1.13.1. Use Case | Visibility

"As a tester I want to design and implement a test on the UI Testing Playground website that checks if an element is visible on screen."

1.13.2. Test Case | Visibility

Description:

"Checking if an element is visible on screen may be a non trivial task. An element may be removed (simplest case), it may have zero height or width, it may be covered by another element, it may be hidden using styles: opacity: 0, visibility: hidden, display: none, or moved offscreen. Learn locators of all buttons, in your testing scenario press the hide button and determine if other buttons are visible or not."

Summary	Steps	Expected Result	Actual Result	Status
The test should check if an element is visible through different scenarios.	1.- Go to http://uitestingplayground.com/visibility. 2.- Verify that there are buttons. 3.- Learn the locator of all buttons. 4.- Click the hide button. 5.- Determine if other buttons are visible or not.	There should be several buttons on the page and they should be hidden or displayed when pressing the hide button.	There are several buttons on the page and they're hidden when pressing the hide button.  uitestingplayground.com/visibility Resources Visibility Checking if element is visible on screen may be a non trivial task. • An element may be removed (simplest case), • it may have zero height or width, • it may be covered by another element, • it may be hidden using styles: opacity: 0, visibility: hidden, display: none, • or moved offscreen. Scenario • Learn locators of all buttons. • In your testing scenario press Hide button. • Determine if other buttons visible or not. Playground Hide	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/ADJ08

1.14. Sample App (QPK-251)

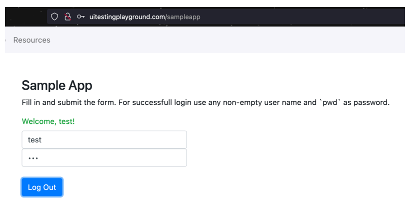
1.14.1. Use Case | Sample App

“As a tester I want to design and implement a test on the UI Testing Playground website that fills and submit a form simulating a demo application.”

1.14.2. Test Case | Sample App

Description:

“Fill in and submit the form. For successful login use any non-empty user name and ‘pwd’ as password.”

Summary	Steps	Expected Result	Actual Result	Status
The test should fill and submit a form with pre-specified data to successfully log in a user.	1.- Go to http://uitestingplayground.com/sampleapp. 2.- Verify that there’s a form. 3.- Fill the form with any user and password ‘pwd’. 4.- Click the log in button. 5.- Verify that the user is successfully logged in.	There should be a form in the website and the user should be logged in after filling the form with the specified data.	There’s a form and the user is logged in after filling the form with the specified data.  uitestingplayground.com/sampleapp Resources Sample App Fill in and submit the form. For successfull login use any non-empty user name and \`pwd\` as password. Welcome, test! test ••• Log Out	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/aeA09

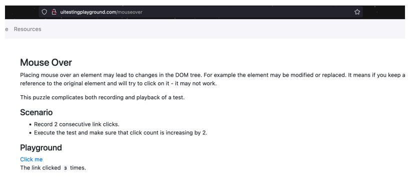
1.15. Mouse Over (QPK-252)

1.15.1. Use Case | Mouse Over

“As a tester I want to design and implement a test on the UI Testing Playground website that places the mouse over an element to change the DOM and makes the element unavailable.”

1.15.2. Test Case | Mouse Over

Description:
“Placing the mouse over an element may lead to changes in the DOM tree. For example, the element may be modified or replaced. It means if you keep a reference to the original element and will try to click on it - it may not work. Record two consecutive link clicks. Execute the test and make sure that click count is increased by 2.”

Summary	Steps	Expected Result	Actual Result	Status
The test should click on an element and make sure that another element in the DOM tree is changing.	1.- Go to http://uitestingplayground.com/mouseover. 2.- Verify that there are two elements. 3.- Click on “Click me”. 4.- Verify that the counter changes when the link is clicked.	There should be a “Click me” link and a counter on the website and the counter should increase when the link is clicked.	There’s a “Click me” link and a counter on the website that increases its number when clicked.  uitestingplayground.com/mouseover Mouse Over Placing mouse over an element may lead to changes in the DOM tree. For example the element may be modified or replaced. It means if you keep a reference to the original element and will try to click on it - it may not work. This puzzle complicates both recording and playback of a test. Scenario • Record 2 consecutive link clicks. • Execute the test and make sure that click count is increasing by 2. Playground Click me The link clicked 3 times.	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/dknvG

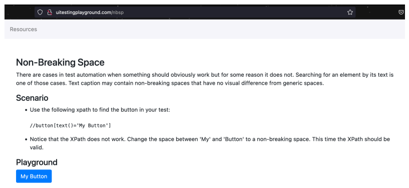
1.16. Non-Breaking Space (QPK-257)

1.16.1. Use Case | Non-Breaking Space

“As a tester I want to design and implement a test on the UI Testing Playground website that finds a button using a specific xpath.”

1.16.2. Test Case | Non-Breaking Space

Description:
“There are cases in test automation when something should obviously work but for some reason, it does not. Searching for an element by its text is one of those cases. A text caption may contain non-breaking spaces that have no visual difference from generic spaces. Use the following XPath to find the button in your test: ‘button[text()=‘My Button’] notice that the XPath does not work. Change the space between ‘My’ and ‘Button’ to a non-breaking space. This time the XPath should be valid.”

Summary	Steps	Expected Result	Actual Result	Status
The test should use a specified Xpath to find a button testing a valid and invalid scenario.	1.- Go to http://uitestingplayground.com/nbsp. 2.- Verify that there’s a button on the website. 3.-Use the xpath “button[text()=‘My Button’] to find the button. 4.- Verify that the test doesn’t find the button. 5.- Use the XPath “button[text()=‘MyButton’] to find the button. 6.- Validate that the test finds the button.	There should be a button on the website and the test should validate a valid and invalid scenario when trying to find the button.	There’s a button on the website that is found during testing when implementing its content with a non-breaking space.  uitestingplayground.com/nbsp Resources **Non-Breaking Space** There are cases in test automation when something should obviously work but for some reason it does not. Searching for an element by its text is one of those cases. Text caption may contain non-breaking spaces that have no visual difference from generic spaces. **Scenario** • Use the following xpath to find the button in your test: `//button[text()='My Button']` • Notice that the XPath does not work. Change the space between 'My' and 'Button' to a non-breaking space. This time the XPath should be valid. **Playground** My Button	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/uCR01

1.17. Overlapped Element (QPK-259)

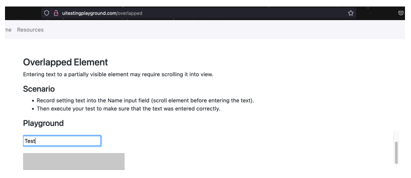
1.17.1. Use Case | Overlapped Element

“As a tester I want to design and implement a test on the UI Testing Playground website that scrolls to find a visible element and enters text to it.”

1.17.2. Test Case | Overlapped Element

Description:

“Entering text to a partially visible element may require scrolling it into view. Record setting text into the Name input field (scroll element before entering the text). Then execute your test to make sure that the text was entered correctly.”

Summary	Steps	Expected Result	Actual Result	Status
The test should scroll an element into view and enter text to it.	1.- Go to http://uitestingplayground.com/overalapped. 2.- Scroll until the input field “Name” is visible. 3.- Enter text into the field. 4.- Verify that text was entered correctly.	There should be two input fields and scrollbars on the website and one of the input fields should be partially visible. The input field “Name” should be able to accept text.	There are two input fields and a scrollbar on ther website, the input field “Name” is visible when you scroll down and it accepts text.  uitestingplayground.com/overlapped Resources Overlapped Element Entering text to a partially visible element may require scrolling it into view. Scenario • Record setting text into the Name input field (scroll element before entering the text). • Then execute your test to make sure that the text was entered correctly. Playground Test	Pass ✅

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/hqQ69

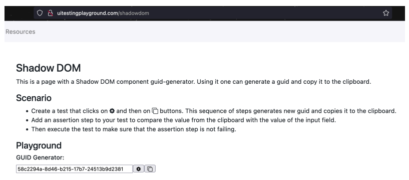
1.18. Shadow DOM (QPK-260)

1.18.1. Use Case | Shadow DOM

“As a tester I want to design and implement a test on the UI Testing Playground website that looks inside a Shadow DOM component to generate a guid and copy it to the clipboard.”

1.18.2. Test Case | Shadow DOM

Description:
“This is a page with a Shadow DOM component guid-generator. Using it one can generate a guid and copy it to the clipboard. Create a test that clicks on the generator and then on the copy buttons. This sequence of steps generates a new guid and copies it to the clipboard. Add an assertion step to your test to compare the value from the clipboard with the value of the input field. Then execute the test to make sure that the assertion step is not failing.”

Summary	Steps	Expected Result	Actual Result	Status
The test should generate a guid and copy it to the clipboard and then assert the value.	1.- Go to http://uitestingplayground.com/shadowdom. 2.- Verify that there’s a GUID Generator on the website. 3.- Click on the button to generate a new GUID. 4.- Click on the button to copy the GUID into the clipboard. 5.- Validate that the value on the input field and the value on the clipboards are the same.	There should be a GUID Generator on the website with two buttons and an input field that generates a GUID and copies to the clipboard. The text copied into the clipboard should be the same as the one in the input field.	There’s a GUID generator on the website with two buttons and an input field. One button generates a GUID but the other one doesn’t copy it to the clipboard. 	Fail 🚫

uitestingplayground.com/shadowdom

Resources

Shadow DOM

This is a page with a Shadow DOM component guid-generator. Using it one can generate a guid and copy it to the clipboard.

Scenario

- Create a test that clicks on ⚙ and then on ⧉ buttons. This sequence of steps generates new guid and copies it to the clipboard.
- Add an assertion step to your test to compare the value from the clipboard with the value of the input field.
- Then execute the test to make sure that the assertion step is not failing.

Playground

GUID Generator:

58c2294a-8d46-b215-17b7-24513b9d2381

Software version: 2.0

Browser: Firefox version 100.0.1.

Jira: shorturl.at/uxCHI

2. Test Matrix

Test Case	Mozilla Firefox 100.0.1	Google Chrome 103.0.5060.53
Dynamic ID (QPK-227)	🟩	🟩
Class Attribute (QPK-228)	🟩	🟩
Hidden Layers (QPK-230)	🟩	🟩
Load Delay (QPK-231)	🟩	🟩
AJAX Data (QPK-232)	🟩	🟩
Client Side Delay (QPK-233)	🟩	🟩
Click (QPK-234)	🟩	🟩
Text Input (QPK-235)	🟩	🟩
Scrollbars (QPK-240)	🟩	🟩
Dynamic Table (QPK-242)	🟩	🟩
Verify Text (QPK-243)	🟥	🟥
Progress Bar (QPK-244)	🟩	🟩
Visibility (QPK-246)	🟩	🟩
Sample App (QPK-251)	🟩	🟩
Mouse Over (QPK-252)	🟩	🟩
Non-Breaking Space (QPK-257)	🟩	🟩
Overlapped Element (QPK-259)	🟩	🟩
Shadow DOM (QPK-260)	🟥	🟥

- **Failed:** 11.11%
- **Passed:** 88.88 %