



## **Playwright Automation Day03**

---



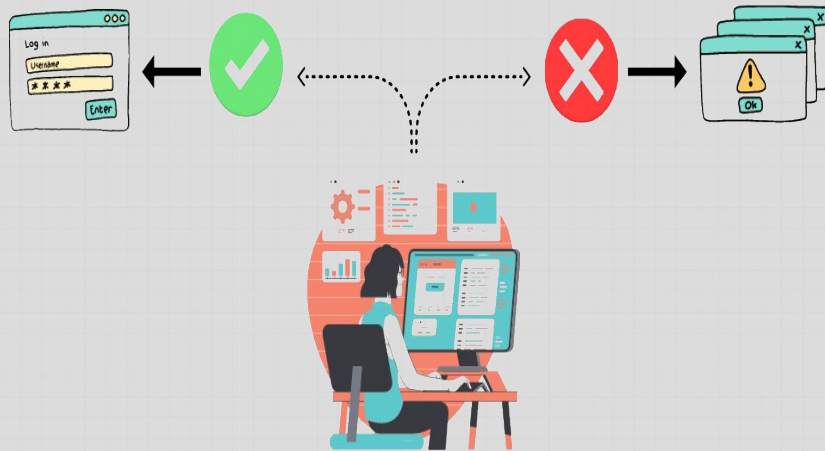
# Content

- Playwright Assertions
- iFrames
- Alerts



# Playwright Assertions

- Assertions are statements in our test code that check if a condition is true.
- Helps us to validate that the software behaves as expected.
- Assertions catch errors and bugs early in the development process
- Playwright provides built-in assertion methods to make tests more reliable:



# Assertion Methods – with values

Methods Name	Description
<code>expect(value).toBe(expected)</code>	Checks if value is strictly equal to expected
<code>expect(value).toBeTruthy()</code>	Checks if value is truthy
<code>expect(value).toBeFalsy()</code>	Checks if value is falsy
<code>expect(value).toBeGreaterThan(number)</code>	Checks if value is greater than number
<code>expect(value).toBeGreaterThanOrEqual(number)</code>	Checks if value is greater than or equal to number
<code>expect(value).toBeLessThan(number)</code>	Checks if value is less than number
<code>expect(value).toBeLessThanOrEqual(number)</code>	Checks if value is less than or equal to number.
<code>expect(value).toBeNull()</code>	Checks if value is null
<code>expect(value).toContain(substring)</code>	Checks if value contains substring
<code>expect(value).toEqual(expected)</code>	checks if two values have the same structure and content, but not necessarily the same reference



# Assertion Methods – with Elements

Methods Name	Description
<b>expect(element).toBeVisible()</b>	Verifies that the element is visible.
<b>expect(element).toBeHidden()</b>	Verifies that the element is hidden.
<b>expect(element).toBeEnabled()</b>	Verifies that the element is enabled.
<b>expect(element).toBeDisabled()</b>	Verifies that the element is disabled.
<b>expect(element).toBeEmpty()</b>	Verifies that the element is empty.
<b>expect(element).toBeChecked()</b>	Verifies that the element is checked. (checkbox, radio button)
<b>expect(element).toContainText(text)</b>	Verifies that the element contains the specified text.
<b>expect(element).toHaveText(text)</b>	Verifies that the element's text is equal to the specified text.
<b>expect(element).toHaveAttribute(name, value)</b>	Verifies that the element has the specified attribute with the specified value.
<b>expect(element).toHaveCSS(name, value)</b>	Verifies that the element has the specified CSS property with the specified value.



# iFrames

- HTML elements that embed another HTML document within the current document.
- Allows to display content from a different source or domain.
- The iframes isolate the content from the main page, they have their own context.
- The iframes isolate the content from the main page, they have their own context.

1

2

3

Start Application

Payment plan


Review


iframe 521.5 x 283.32


Card number

1234 1234 1234 1234

VISA








Expiration

MM / YY

CVC

CVC



Country

United States

ZIP

12345

By providing your card information, you allow CYDEO to charge your card for future payments in accordance with their terms.

```

    <div _ngcontent-ng-c1610074379 id=
    payment-element class= stripeElement
    t">
      <div class= "__PrivateStripeElement" style="margin: -4px 0px !importan
    t; padding: 0px !important; border: none !important; display: block !
    important; background: transparent !important; position: relative !im
    portant; opacity: 1 !important; clear: both !important; transition: h
    eight 0.35s ease 0s !important;">
        ...
        ▶ <iframe name= "__privateStripeFrame1564" frameborder= "0"
        allowtransparency= "true" scrolling= "no" role= "presentation" allow=
        "payment *; publickey-credentials-get *" src= "https://js.stripe.co
        m/v3/elements-inner-payment-4401281___Fcheckout.cydeo.com%2Fsoc-us00
        66controllerId= _privateStripeController1561" title= "Secure payment
        input frame" style= "border: 0px !important; margin: -4px; padding:
        0px !important; width: calc(100% + 8px); min-width: 100% !importan
        t; overflow: hidden !important; display: block !important; user-sel
        ect: none !important; transform: translate(0px) !important; color-s
        cheme: light only !important; height: 283.32px; opacity: 1; transit
        ion: height 0.35s ease 0s, opacity 0.4s ease 0.1s;"> ...
        </iframe> == $0
      </div>
    </div>
    <div _ngcontent-ng-c1610874379 id= "error-message"></div>
  </form>
  </app-payment-test>
</div>
▶ <div _ngcontent-ng-c1716864982 class= "panel-content-holder ng-star-inserte
d"> ... </div> (flex)

```



# Handlings iframes in Playwright

- The `frameLocator()` method of the `page` object is used for locating the iframes.
- The iframes can be located by using the following locators:

- ID
- Name
- CSS
- XPath

```
// Locate iFrame by ID
const iframe1 = page.frameLocator('#frameID');

// Locate iFrame by name
const iframe2 = page.frameLocator('iframe[name="myFrame"]');

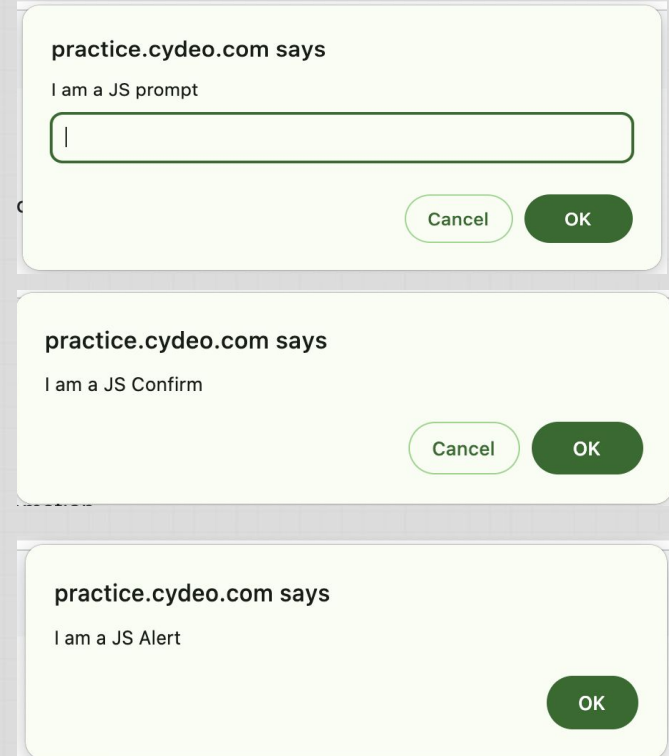
// Locate iFrame using CSS selector
const frame3 = page.frameLocator('iframe.classname');

// Locate iFrame using XPath
const iframe4 = page.frameLocator('//iframe[@class="value"]');
```



# JavaScript Alerts

- Alerts are browser dialogs that interrupt user interaction.
- Common types include alerts, confirmation dialogs, and prompts.
- There are three types of alerts:
  - **alert**: Simple message with an **OK** button
  - **confirm**: Message with **OK** and **Cancel** buttons
  - **prompt**: Message with a **text input**, **OK**, and **Cancel** buttons





# Handlings alerts in Playwright

- The `on()` method of the `page` object is used for handling alerts
- The `on()` method needs to be called **before** the alert is triggered.
  - `accept()`: accepts the dialog
  - `dismiss()`: dismisses the dialog
  - `accept(input)`: provides input and accepts the dialog

```
page.on('dialog', async dialog => {  
    await dialog.accept();  
});
```

```
page.on('dialog', async dialog => {  
    await dialog.dismiss();  
});
```

```
page.on('dialog', async dialog => {  
    await dialog.accept('Inputs');  
});
```

