Day 6 – Clocks, Alert, Web Server, Sharding

1. **HTML Structure (index.html):**

- Sets up a basic webpage with a div to display the current time, and three buttons: "Alert", "Confirm", and "Prompt".

2. **JavaScript Logic (index.js):**

- **Clock Functionality:**
    - **renderTime**() function: Updates the content of the div with the id current-time to display the current date and time, formatted using **toLocaleString**().
    - **setInterval**(): Calls **renderTime**() every 1000 milliseconds (1 second) to update the time dynamically.
- **Button Event Listeners:**
    - **Alert Button:** When clicked, displays a simple alert box with the message "This is an alert!".
    - **Confirm Button:** When clicked, displays a confirmation dialog with the message "Do you confirm this action?". The result (true/false) is logged to the console.
    - **Prompt Button:** When clicked, displays a prompt dialog asking the user to enter their name, with "Default Name" as the default value. The entered value (or null if canceled) is logged to the console.

3. **Playwright Tests:**

- **Test Suite:** A test suite named "Clock and Alert Tests" is defined.
- **Test 1: "Test with predefined time"**
    - Navigates to the specified HTML page.
    - Uses page.evaluate() to override the global Date.now() function, effectively freezing time at a specific date and time (2024-02-02T10:00:00).
    - Asserts that the text content of the element with the data-testid attribute current-time matches the expected time ("2/2/2024, 10:00:00 AM").
    - Changes the mocked Date.now() to a new time (2024-02-02T11:30:00).

- Asserts that the time displayed is updated to the new mocked time ("2/2/2024, 11:30:00 AM").

```
test.describe('Clock and Alert Tests', () => {
  test('Test with predefined time', async ({ page }) => {
    await
page.goto('http://127.0.0.1:5500/Playwright_UST/Day_6_Playwright_Commands/src/
index.html');
    await page.evaluate(() => {
      const fixedDate = new Date('2024-02-02T10:00:00');
      Date.now = () => fixedDate.getTime();
    });
    await expect(page.getByTestId('current-time')).toHaveText('2/2/2024,
10:00:00 AM');
    await page.evaluate(() => {
      const fixedDate = new Date('2024-02-02T11:30:00');
      Date.now = () => fixedDate.getTime();
    });
    await expect(page.getByTestId('current-time')).toHaveText('2/2/2024,
11:30:00 AM');
  });
```

- **Test 2: "Test alert, confirm, and prompt buttons"**
  - Navigates to the HTML page.
  - Sets up a dialog event listener to handle alert, confirm, and prompt dialogs.
  - **Dialog Handling:** Inside the dialog event listener:
    - Checks the dialog.type() to determine the type of dialog (alert, confirm, or prompt).
    - Asserts that the dialog.message() matches the expected message for each dialog type.
    - Calls dialog.accept() to close the alert and confirm dialogs, or dialog.accept('Playwright User') to close the prompt dialog and enter "Playwright User" as the input.
  - Clicks the "Alert", "Confirm", and "Prompt" buttons in sequence.

```
test('Test alert, confirm, and prompt buttons', async ({ page }) => {
    await
page.goto('http://127.0.0.1:5500/Playwright_UST/Day_6_Playwright_Commands/src/
index.html');
    page.on('dialog', async (dialog) => {
      if (dialog.type() === 'alert') {
```

```
        expect(dialog.message()).toBe('This is an alert!');
        await dialog.accept();
      } else if (dialog.type() === 'confirm') {
        expect(dialog.message()).toBe('Do you confirm this action?');
        await dialog.accept();
      } else if (dialog.type() === 'prompt') {
        expect(dialog.message()).toBe('Please enter your name:');
        await dialog.accept('Playwright User');
      }
    });
    await page.getByTestId('alert-button').click();
    await page.getByTestId('confirm-button').click();
    await page.getByTestId('prompt-button').click();
  });
});
```

4. **Host the web server in local system and run the test automation projects locally.**

```
webServer: {
    command: 'npm run start',
    url: 'http://127.0.0.1:5500',
    reuseExistingServer: !process.env.CI,
  },
```

5. **Sharding**

- Can further scale Playwright test execution by running tests on multiple machines simultaneously.
- We call this mode of operation "sharding". Sharding in Playwright means splitting your tests into smaller parts called "shards".
- Each shard is like a separate job that can run independently. The whole purpose is to divide your tests to **speed up test runtime.**
- **npx playwright test --shard=1/4**
- **npx playwright test --shard=2/4**
- **npx playwright test --shard=3/4**
- **npx playwright test --shard=4/4**