Class 2

Step 1: **Generate Code Using codegen**

First, run the following Playwright command in your terminal to generate the test code. This will open the browser and record your actions.

npx playwright codegen

- When you run this command, Playwright will start recording your actions in the browser.
- As you interact with the page, the code will be generated and output in real-time in the terminal.

After interacting with the page (e.g., clicking buttons, filling forms), you can stop the recording, and the generated code will be available for you to modify further.

Step 2: **Code Explanation**

```
// Import necessary methods from Playwright's testing library

import { expect, test } from '@playwright/test';

// Set up the test to run with the browser visible (not headless)

test.use({ headless: false });

test('Login and Checkout Test', async ({ page }) => {



  // Step 1: Navigate to the website

  // This command opens the Saucedemo login page

  await page.goto('https://www.saucedemo.com/');

  // Step 2: Fill in the login form

  // Click on the username field and enter the username 'standard_user'

  await page.locator('[data-test="username"]').click();

  await page.locator('[data-test="username"]').fill('standard_user');
```

```
// Click on the password field and enter the password 'secret_sauce'
await page.locator('[data-test="password"]').click();
await page.locator('[data-test="password"]').fill('secret_sauce');

// Step 3: Click on the login button to submit the login form
await page.locator('[data-test="login-button"]').click();
// Step 4: Take a screenshot of the page after login (for debugging or verification)
await page.screenshot({ path: 'image.png' });
// Step 5: Add items to the shopping cart
// Click on the 'Add to Cart' button for the backpack
await page.locator('[data-test="add-to-cart-sauce-labs-backpack"]').click();

// Click on the 'Add to Cart' button for the bike light
await page.locator('[data-test="add-to-cart-sauce-labs-bike-light"]').click();
// Step 6: Verify the cart contains 2 items
const product = await page.locator("#shopping_cart_container").innerText();
console.log(product); // Logs the contents of the cart to the console
// Assert that the number of items in the cart is exactly 2
await expect(product).toBe("2");
// Step 7: Proceed to the checkout page
await page.locator('[data-test="shopping-cart-link"]').click();
```

```
  await page.locator('[data-test="checkout"]').click();
  // Step 8: Fill in the checkout form
  // Enter the first name 'qa'
  await page.locator('[data-test="firstName"]').click();
  await page.locator('[data-test="firstName"]').fill('qa');

  // Enter the last name 'melody'
  await page.locator('[data-test="lastName"]').click();
  await page.locator('[data-test="lastName"]').fill('melody');

  // Enter the postal code '429'
  await page.locator('[data-test="postalCode"]').click();
  await page.locator('[data-test="postalCode"]').fill('429');

  // Step 9: Continue to the next step in the checkout process
  await page.locator('[data-test="continue"]').click();
  // Step 10: Finish the checkout process
  await page.locator('[data-test="finish"]').click();
  // Step 11: Return to the product page after completing the checkout
  await page.locator('[data-test="back-to-products"]').click();
});
```

Key Concepts and How the `codegen` Fits in:

- **Generate Code with `codegen`**: By running `npx playwright codegen https://www.saucedemo.com/`, Playwright will record your browser interactions. This is a helpful tool for beginners who may not be familiar with selectors or how to perform automated actions. The code generated by `codegen` can then be fine-tuned for specific tests.

- **Interacting with Web Elements**:

    - **Locators**: Playwright uses `locator` to identify elements on the page, such as buttons, text fields, or links. In the example, we use `[data-test="username"]` to identify the username input field. This is a data attribute that uniquely identifies elements in your application.
    - **Actions**: Methods like `.click()` and `.fill()` simulate user interactions such as clicking and typing.

- **Assertions**:

    - The line `await expect(product).toBe("2");` is an assertion, meaning it checks if the number of items in the cart is exactly 2. Assertions are essential for verifying that your web application is behaving as expected.

- **Screenshots**:

    - The command `await page.screenshot({ path: 'image.png' });` helps capture a screenshot of the browser at any point in the test, which is useful for debugging or visual verification.

Tips for Students:

- **Using `codegen`**: This tool helps generate boilerplate code that you can later refine to match your testing goals.
- **Selectors**: Use data attributes (like `data-test="username"`) for more stable and readable tests.
- **Debugging**: Running tests with `headless: false` allows you to watch the tests run, which is valuable for identifying issues or understanding the flow.