Day 3 – Playwright

1. **Fixture Setup**

   Two types of Fixtures:

- Custom
- Bulit In – page, context, browser

Custom fixtures are declared to inject instances of each page class (LoginPage, ProductsPage, etc.). Fixtures act as a **bridge** between the test logic and the application's UI by providing **reusable setup** and context (like **page objects**, user sessions, etc.) to tests.

```
type SaucedemoFixtures = {
    loginPage: LoginPage,
    productsPage: ProductsPage,
    productDetailsPage: ProductDetailsPage,
    cartPage: CartPage,
    checkoutPage: CheckoutPage,
    orderConfirmationPage: OrderConfirmationPage
};
export const test = base.extend<SaucedemoFixtures>({
    loginPage: async ({ page }, use) => {
        const loginPage = new LoginPage(page);
        await loginPage.goto();
        await use(loginPage);
    },
```

2. **Login in beforeEach Hook**

Automatically logs in a user before every test

```
test.beforeEach(async ({ loginPage }) => {
  await loginPage.goto();
  await loginPage.login("standard_user", "secret_sauce");
});
```

3. **Parallelized End-To-End Testing**

```
test.describe.configure({ mode: 'parallel' }); //serial
```

Using this to run tests faster by adding necessary workers. If you want to run in serial use serial instead of parallel.

Or In playwright.config.ts file set the parallelism, it will test the testcase fully parallel mode.

```
fullyParallel: true,
```

Or we can achieve parallelism by specifying the number of workers required to achieve it in the CLI like this:

npx playwright test --project chromium -workers 3

4. **Parameterized Testing / Data-Driven Testing**

Iterates over different sets of user details specified by us to verify the functionality and test flow.

The same test logic is executed multiple times with **different sets of input data**. This helps verify **how the application behaves with various user inputs or scenarios**, improving test coverage and **reducing code duplication.**

```
[
  { firstName: 'Aaryan', lastName: 'ust'},
  { firstName: 'Abden', lastName: 'ust'},
  { firstName: 'Deepak', lastName: 'ust'},
].forEach(({ firstName}) => {
  test.only(`End to end test for cart functionality of ${firstName}`, async ({
productsPage }) => {

  await checkoutPage.enterDetails(firstName, lastName, pin);
  });
})
```

5. **npx playwright test --max-failures=2**

Runs all Playwright tests, but stops the test run after 2 test failures. Useful for saving time during debugging or CI runs by not continuing after multiple failures.

Or we can specify it in the playwright.config.ts in **defineConfig:**

```
maxFailures: 2,
```

6. An End-To-End Scenario

```
test.describe.configure({ mode: 'parallel' });

test.beforeEach(async ({ loginPage }) => {
  await loginPage.goto();
  await loginPage.login("standard_user", "secret_sauce");
});

[
  { firstName: 'Aaryan', lastName: 'ust', pin: '12345', orderConfirmMsg:
'Thank you for your order!' },
  { firstName: 'Abden', lastName: 'raj', pin: '12345', orderConfirmMsg: 'Thank
you for your order!' },
  { firstName: 'Deepak', lastName: 'Antony', pin: '12345', orderConfirmMsg:
'Thank you for your order!' },

].forEach(({ firstName, lastName, pin, orderConfirmMsg }) => {
  test.only(`End to end test for cart functionality of ${firstName}`, async ({
productsPage, productDetailsPage, cartPage, checkoutPage,
orderConfirmationPage }) => {
    await productsPage.clickFirstItem();
    await productDetailsPage.addToCart();
    const cartQuantity = await productDetailsPage.getCartQuantity();
    expect(cartQuantity).toBe("1");
    console.log("Cart quantity is 1");
    await productDetailsPage.gotocheckout();
    await cartPage.gotocheckout();
    await checkoutPage.enterDetails(firstName, lastName, pin);
    await checkoutPage.finishOrder();
    await orderConfirmationPage.verifyOrderConfirmation(orderConfirmMsg);
    await orderConfirmationPage.backToHome();
    await productsPage.verifyProductPageIsDisplayed();
  });
})
```