THE FIX IS IN

Understanding Playwright Fixtures

Amber Race
Adobe

ABOUT ME

- Started tech career at Microsoft in 1998
- Moved into SDET role in 2005
- Been at Adobe since 2021
- Converted test automation from WebDriverIO to Playwright in 2023

AGENDA

- What is a fixture?
- How do fixtures work?
- Combining fixtures
- Fixture options
- Fixture design

WHAT IS A FIXTURE

"Playwright Test is based on the concept of test fixtures. Test fixtures are used to establish the environment for each test, giving the test everything it needs and nothing else. Test fixtures are isolated between tests. With fixtures, you can group tests based on their meaning, instead of their common setup."

https://playwright.dev/docs/test-fixtures#introduction

ESTABLISHING THE ENVIRONMENT

The test only needs the checkout page

```
test.beforeEach(async ({ page, loginPage }) => {
    await page.goto(Routes.LOGIN);
    await loginPage.login(Usernames.STANDARD);
    await page.goto(Routes.CHECKOUT);
});
test("should show error when first name is missing", async ({
    checkoutPage
}) => {
    await checkoutPage.fillLastName("Doe");
    await checkoutPage.fillPostalCode("12345");
    await checkoutPage.continue();
    await validateCheckoutError(
        checkoutPage,
        CheckoutErrors.MISSING_FIRST_NAME
    );
});
```

The setup only needs the login page

FIXTURE ISOLATION

Each test get its own instance of the login page it a completely separate browser context

```
test("should show error when username and password are invalid", async ({
    loginPage
}) => {
    await loginPage.login("invalid_user", "invalid_password");
    await validateLoginError(
        loginPage,
        LoginErrors.INVALID_CREDENTIALS
    );
});
test("should show error when username is locked out", async ({
    loginPage
}) => {
    await loginPage.login(Usernames.LOCKED_OUT);
    await validateLoginError(loginPage, LoginErrors.LOCKED_OUT_USER);
});
```

GROUP TESTS BASED ON MEANING

Because fixture instances are isolated, two tests in the same block can have totally different setups

```
test.describe("Sauce Errors", () => {
    test("should show error on login page when username is missing", async ({
        loginPage
    }) => {
        await loginPage.login("");
        await validateSauceError(loginPage, LoginErrors.MISSING_USERNAME);
    });

    test("should show error on checkout page when first name is missing", async ({
        checkoutPage
    }) => {
        await checkoutPage.fillLastName("Doe");
        await checkoutPage.fillPostalCode("12345");
        await checkoutPage.continue();
        await validateSauceError(
            checkoutPage,
            CheckoutErrors.MISSING_FIRST_NAME
        );
    });
});
```

FIXTURES VS PAGE OBJECTS

- A page object can be a fixture, but fixtures are not limited to just page objects. Any object or constant can be made into a fixture
- Fixtures include everything Playwright needs to create an instance of the target object, including setup and tear down
- Playwright handles the management of fixtures

WHAT IS A FIXTURE, REALLY??

A fixture is an object combined with all the necessary setup and teardown code that Playwright requires to create a specific instance.

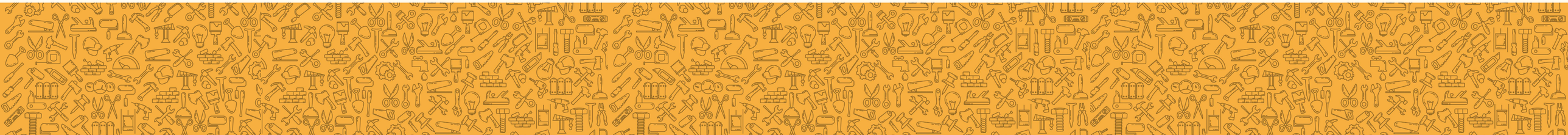
HOW FIXTURES WORK

THE FIXTURE POOL

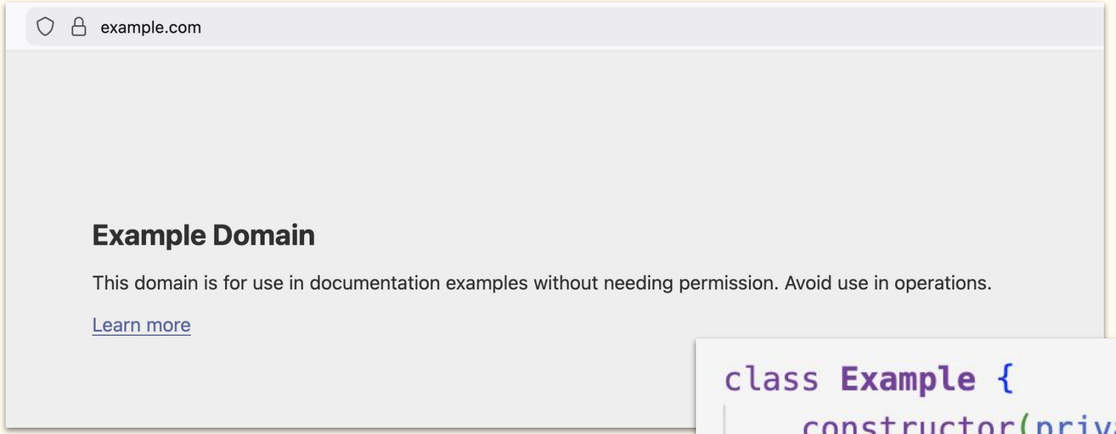
```
const playwrightFixtures = {
    //...other fixtures
    actionTimeout: [0, { option: true, box: true }],
    testIdAttribute: ["data-testid", { option: true, box: true }],
    navigationTimeout: [0, { option: true, box: true }],
    baseURL: [
        async ({}, use) => {
            await use(process.env.PLAYWRIGHT_TEST_BASE_URL);
        },
        { option: true, box: true }
    ]
    //...other fixtures
};
```

- Playwright comes with a default pool of fixtures such as *page*, *baseUrl*, *isMobile*, etc.
- The pool is a mapping of the fixture name to the value or function that creates the instance
- Extending a test with a new fixture adds to this pool

https://github.com/microsoft/playwright/blob/main/packages/playwright/src/index.ts



CREATING NEW FIXTURES




```
class Example {
    constructor(private readonly page: Page) {}
    learnMore = async (): Promise<void> => {
        await this.page.getByRole("link", { name: "Learn More" }).click();
    };
    url = (): string => this.page.url();
}
```

CREATING NEW FIXTURES

```
import { test as base } from "@playwright/test";

export const test = base.extend<{
    example: Example;
}>({
    example: async ({ page, baseURL }, use) => {
        console.log(`opening ${baseURL}`);
        await page.goto("/");
        await use(new Example(page));
        console.log("finished with example");
        await page.close();
    }
});
```

Start by extending the base Playwright test with your new mapping of name to type. Note that this is the type that will get passed to the test

Then pass in the fixture name with the full instantiation function, including any setup and teardown

FIXTURE PROCESSING

```
test.use({ baseURL: "https://www.example.com" });
test.describe("Example Test", () => {
    test("test example link", async ({ example }) => {
        await example.learnMore();
        expect(example.url()).toContain("www.iana.org");
    });
});
```

```
export const test = base.extend<{
    example: Example;
}>({
    example: async ({ page, baseURL }, use) => {
        console.log(`opening ${baseURL}`);
        await page.goto("/");
        await use(new Example(page));
        console.log("finished with example");
        await page.close();
    }
});
```

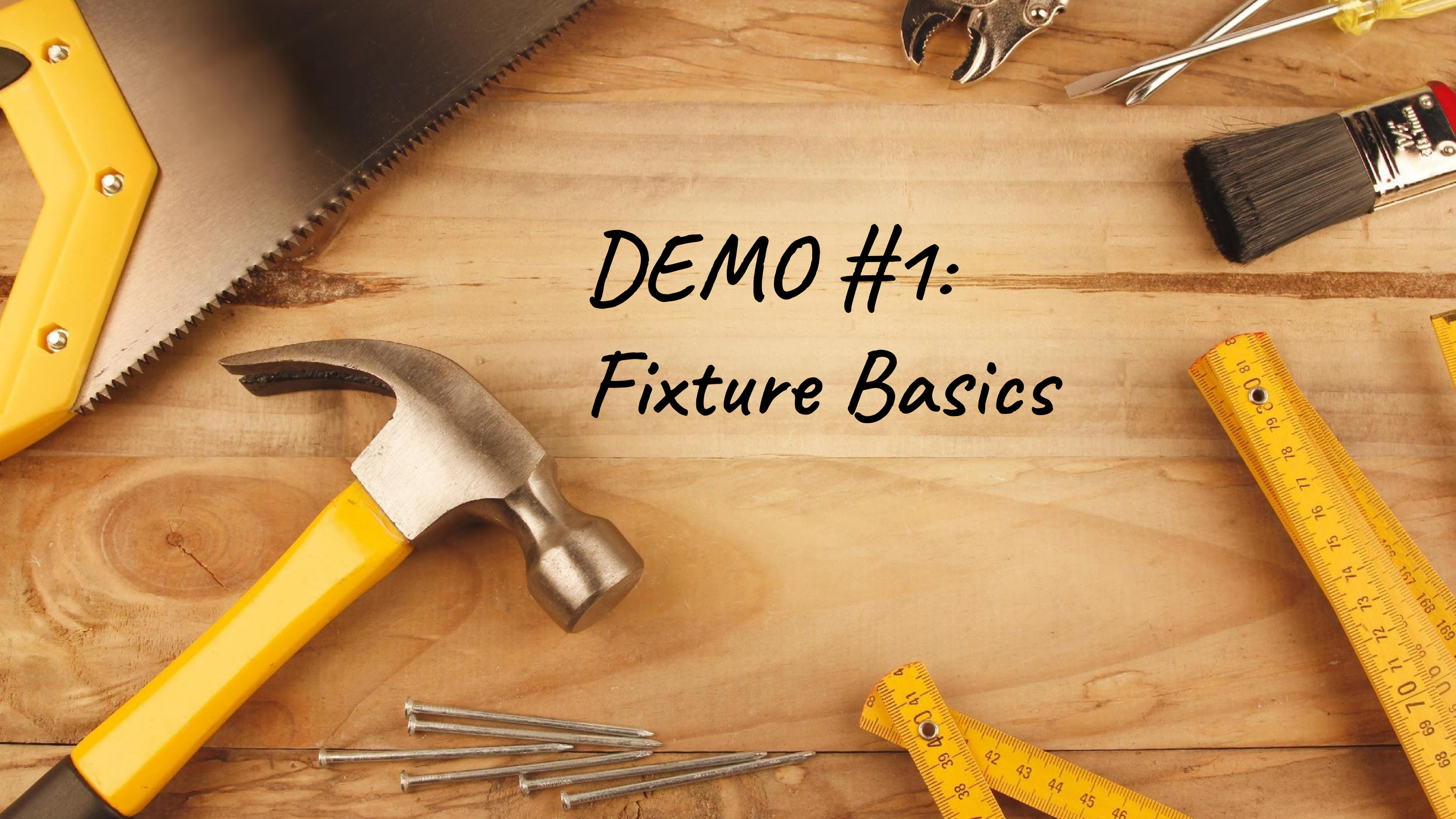
Extract fixture names from the test parameters

Determine dependencies and process required fixtures in order

Pass only the requested objects to the test

Test!

Clean up fixtures in reverse order

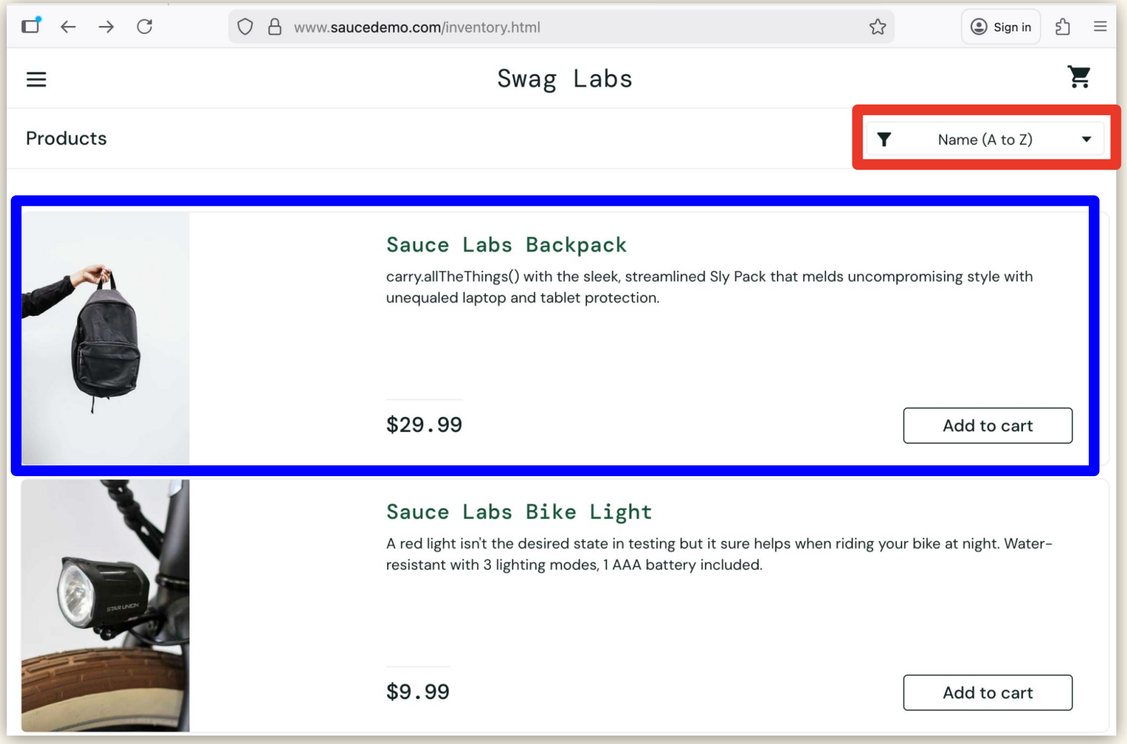


DEMO #1:
Fixture Basics

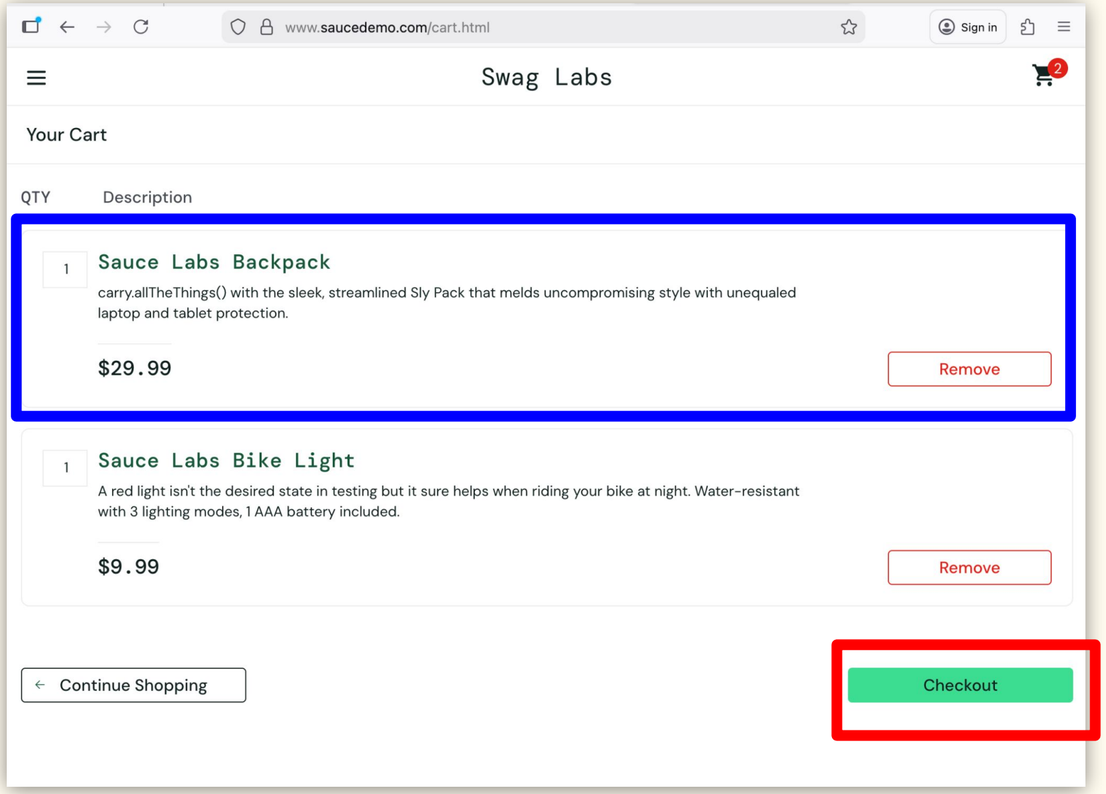
COMBINING FIXTURES

FIXTURE COMPOSITION

Inventory Page



Cart Page



FIXTURE COMPOSITION

```ts
type Item = {
    name: string;
    price: string;
    description: string;
};
class Items {
    constructor(private readonly page: Page) {}
    item = (): Locator => this.page.getByTestId("inventory-item");
    allItems = async (): Promise<Item[]> => {
        const items = (await this.item().all()) ?? [];
        return await Promise.all(…
        );
    };
}
```

```ts
class InventoryPage {
    constructor(private readonly page: Page, private readonly items: Items) {}
    allItems = async (): Promise<Item[]> => {
        return await this.items.allItems();
    };
    sortDropdown = (): Locator =>
        this.page.getByTestId("product-sort-container");
    sortByPriceLowToHigh = async (): Promise<void> => {
        await this.sortDropdown().selectOption("lohi");
    };
}
```

```ts
class CartPage {
    constructor(private readonly page: Page, private readonly items: Items) {}
    contents = async (): Promise<Item[]> => {
        return await this.items.allItems();
    };
    checkoutButton = (): Locator => this.page.getByTestId("checkout");
    checkout = async (): Promise<void> => {
        await this.checkoutButton().click();
    };
}
```

TEST COMPOSITION

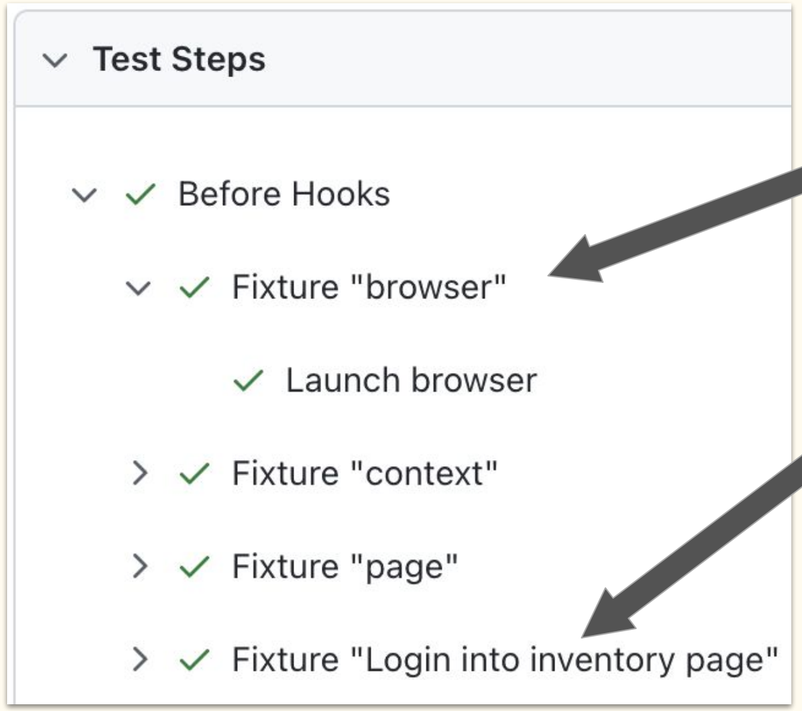
```ts
export const test = base.extend<{
    loginForm: LoginForm;
    items: Items;
    inventoryPage: InventoryPage;
    cartPage: CartPage;
}>({
    loginForm: async ({ page }, use) => {
        await page.goto(Routes.LOGIN);
        await use(new LoginForm(page));
    },
    items: async ({ page }, use) => {
        await use(new Items(page));
    },
    inventoryPage: async ({ page, items, loginForm }, use) => {
        await loginForm.login();
        await use(new InventoryPage(page, items));
    },
    cartPage: async ({ page, items }, use) => {
        await use(new CartPage(page, items));
    }
});
```

```ts
test.describe("show items in different pages", () => {
    test("should sort items by price low to high", async ({
        inventoryPage
    }) => {
        await inventoryPage.sortByPriceLowToHigh();
        const items = await inventoryPage.allItems();
        const prices = items.map((item) =>
            parseFloat(item.price.replace("$", ""))
        );
        const sortedPrices = [...prices].sort((a, b) => a - b);
        expect(prices).toEqual(sortedPrices); — 1ms
    });
    test("should display items in cart", async ({
        inventoryPage
        cartPage
    }) => {
        const items = await inventoryPage.allItems();
        const targetItem = items[3];
        await inventoryPage.addItemToCart(targetItem.name);
        await cartPage.open();
        const cartItems = await cartPage.contents();
        expect(cartItems).toHaveLength(1); — 0ms
        expect(cartItems[0]).toEqual(targetItem); — 0ms
    });
});
```

DEMO #2:
Combining Fixtures

FIXTURE OPTIONS

TITLE AND BOX OPTIONS



Use `{ box: true }` to hide potentially confusing setup steps

Set `{ title: "My Title" }` to display a more user-friendly description in the test report

AUTO ADDED FIXTURES

Set `{ auto: "true" }` when extending tests to add fixtures automatically. This is useful for features you want to be in every test

- Telemetry setup
- Metadata handling
- Log collection
- Document cleanup

```
owner: "",
ownerCheck: [
    async ({ owner }, use, testInfo: TestInfo) => {
        if (owner.length === 0) {
            throw new Error("Owner is required");
        }

        testInfo.annotations.push({
            type: "owner",
            description: owner
        });
        await use(owner);
    },
    { auto: true }
],
```

WORKER FIXTURES

- Playwright divides test execution across multiple worker threads.
- Fixtures can be added to workers directly to act on all the tests executed by that thread

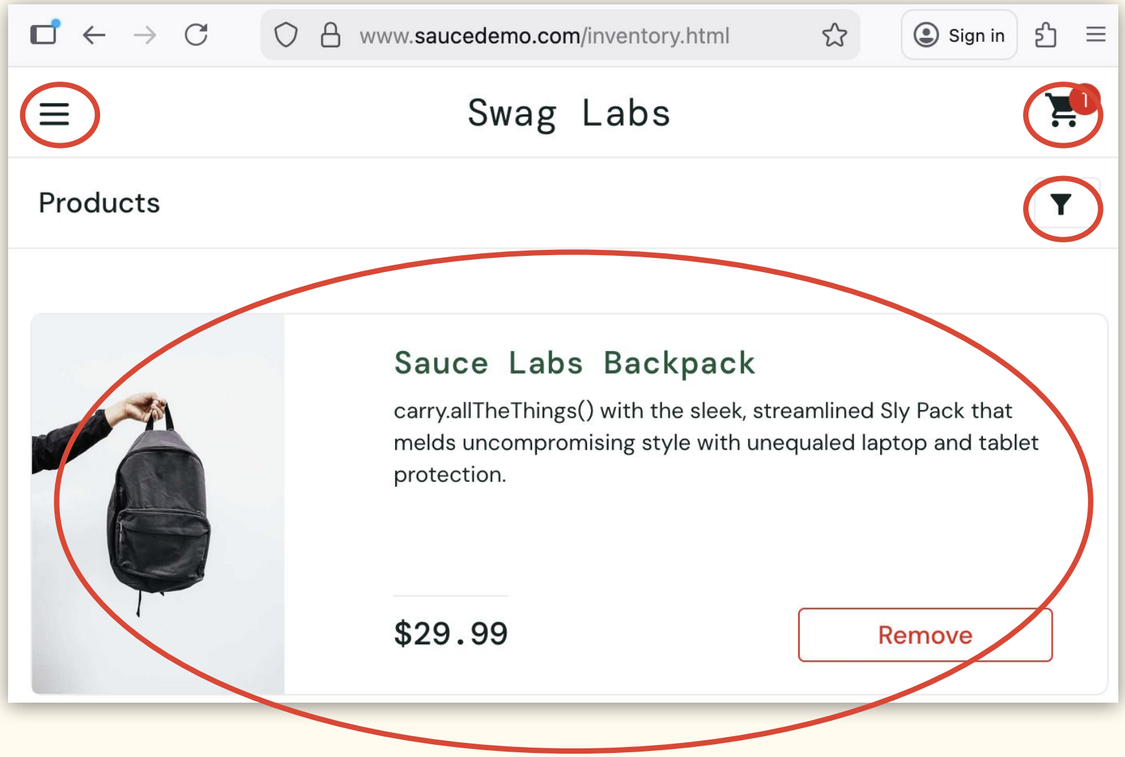
```
username: [
    async ({}, use, workerInfo: WorkerInfo) => {
        const usernames = Object.values(Usernames);
        const username =
            usernames[workerInfo.workerIndex % usernames.length];
        await use(username);
    },
    { scope: "worker" }
],
```


DEMO #3:
Fixture Options

FIXTURE DESIGN

KEEP FIXTURES FOCUSED

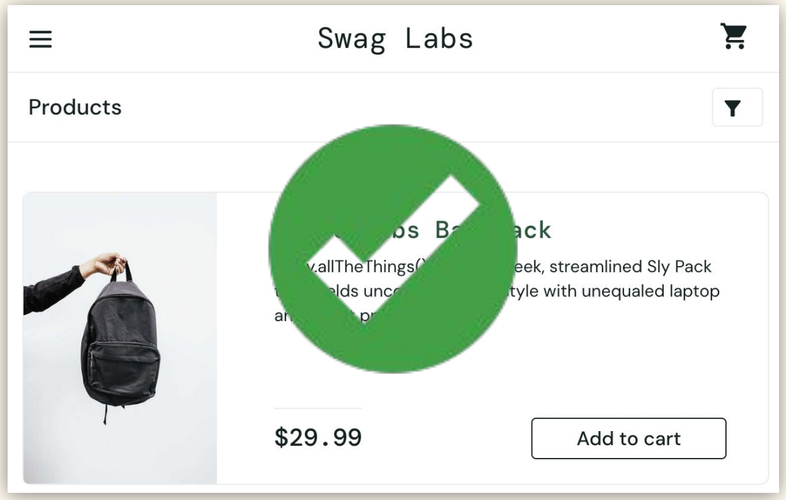
- Using a component-based fixture design provides more flexibility for creating tests and dealing with app changes
- Try to limit unique locators to one specific fixture
- Smaller fixtures can always be combined into larger ones if needed

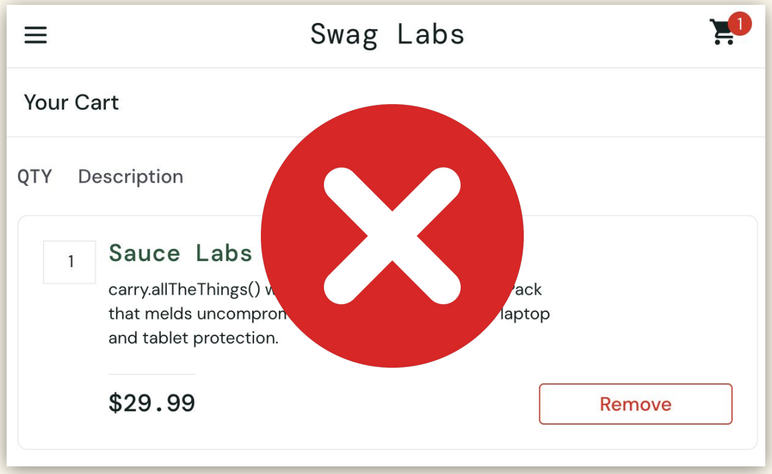


FIXTURES ARE NOT TESTS

- Fixtures are best used for actions, not assertions
- Puting assertions into fixtures makes them less flexible
- Fixtures set up the environment for the test to validate

```
validateItem = async (item: SauceItem): Promise<void> => {
    expect(item.name).toBeTruthy();
    expect(item.description).toBeTruthy();
    expect(item.price).toMatch(USDollarRegex);
    expect(item.priceNumber).toBeGreaterThan(0);
    expect(item.itemImage).toBeDefined();
    await expect(item.itemImage!).toBeVisible();
    expect(item.addButton).toBeDefined();
    await expect(item.addButton!).toBeVisible();
};
```





KEEP TEST EXTENSIONS FOCUSED

```ts
export const test = base.extend<{
    username: string;
    loginPage: LoginPage;
    inventoryPage: InventoryPage;
    cartPage: CartPage;
    summaryPage: SummaryPage;
    completePage: CompletePage;
    checkoutPage: CheckoutPage;
    detailPage: DetailPage;
    inventoryItem: InventoryItem;
    cart: Cart;
    sideMenu: SideMenu;
    sorter: Sorter;
    overview: Overview;
    sauceError: SauceError;
    loginForm: LoginForm;
    checkoutForm: CheckoutForm;
}>({
    username: [Usernames.STANDARD, { option: true }],
    loginPage,
    inventoryPage,
    cartPage,
    summaryPage,
    completePage,
    checkoutPage,
    detailPage,
    inventoryItem,
    cart,
    sideMenu,
    sorter,
    overview,
    sauceError,
    loginForm,
    checkoutForm
});
```

Don’t try to stuff your entire application into a single test extension

Extend tests based on target workflows or ownership areas for easier maintenance

```ts
export const test = base.extend<{
    username: string;
    items: InventoryItem;
    cart: Cart;
    sorter: Sorter;
    loginForm: LoginForm;
}>({
    username: Usernames.STANDARD,
    loginForm: async ({ page }, use) => {
        await page.goto(Routes.LOGIN);
        await use(new LoginForm(page));
    },
    items: async ({ page, loginForm, username }, use) => {
        await loginForm.login(username);
        await use(new InventoryItem(page));
    },
    cart: async ({ page }, use) => {
        await use(new Cart(page));
    },
    sorter: async ({ page }, use) => {
        await use(new Sorter(page));
    }
});
```

WORK WITH PLAYWRIGHT NOT AGAINST IT

Fixtures are weird.
The code looks weird.
Embrace the weird.

```
export type TestAccount = { username: string };
export const workerTest = test.extend<
    {
        username: string;
    },
    {
        testAccount: TestAccount;
    }
>({
    testAccount: [
        async ({}, use, workerInfo: WorkerInfo) => {
            const usernames = Object.values(Usernames);
            const testAccount =
                usernames[workerInfo.workerIndex % usernames.length];
            await use({ username: testAccount } as TestAccount);
        },
        { scope: "worker" }
    ],
    username: async ({ testAccount }, use) => {
        await use(testAccount!.username);
    }
});
```

TEST AUTOMATION IS SOFTWARE

- Principles of good software design apply to test automation
- Write fixtures with usability and maintenance in mind
- Test automation engineers are software developers

THANK YOU!