Playwright TypeScript
User Events & Actions — Student Checklist + Examples

Trainer handout • Updated January 08, 2026 • Instructor: MD Zaman

How to use this handout: Keep this as your daily reference. Each item is something you should be able to demonstrate in code during labs and the capstone.

Master Checklist

Navigation & Page Lifecycle	■ Open a page and navigate with waiting strategy ■ Back / forward / reload ■ Popups (new tab/window) and multiple pages ■ Iframes (frameLocator)
Locators & User Interactions	■ Locate by role/label/testid/text ■ Click / dblclick / right-click; hover ■ Fill / clear; type with delay ■ Keyboard keys (Enter/Tab/Ctrl/Cmd shortcuts) ■ Select dropdown; check/uncheck; radio ■ Upload and download files ■ Drag and drop; scroll into view
Assertions & Validation	■ Visibility/hidden; enabled/disabled; editable ■ Text; value; attribute; URL; title ■ Soft assertions (when appropriate)
Waiting & Stability	■ Use auto-wait (locators + expect) ■ Wait for URL/navigation; wait for response ■ Avoid hard sleeps (waitForTimeout)
Dialogs, Modals, Toasts	■ Alert/confirm/prompt dialogs ■ Modal confirm/cancel flows ■ Toast/status message assertions
Auth & Session	■ Login once; reuse storageState ■ Session expiry and logout validation
Network & API	■ APIRequestContext for setup/validation ■ API setup -> UI verify ■ Route mocking/interception for controlled scenarios
Structure & Evidence	■ test.describe + hooks + tags (smoke/regression) ■ Screenshots/video/trace on failure ■ HTML (and optionally JUnit) reporting

Definitions + One Example Each

These are the core actions you'll use repeatedly. In labs, you should be able to explain what each does and why it's used.

Navigate and wait

Go to a URL and wait for the correct ready state.

```
await page.goto("https://the-internet.herokuapp.com/", { waitUntil: "domcontentloaded" });
```

Accessible locator (recommended)

Find elements using role/label/testid for stability.

```
await page.getByRole("link", { name: "Form Authentication" }).click();
```

Fill + click + assert

Perform a user input action and verify the outcome.

```
await page.getByLabel("Username").fill("tomsmith");
await page.getByLabel("Password").fill("SuperSecretPassword!");
await page.getByRole("button", { name: "Login" }).click();
await expect(page.locator("#flash")).toContainText("You logged into a secure area!");
```

Popup / new window

Capture a new page opened by a user action.

```
const [popup] = await Promise.all([
  page.context().waitForEvent("page"),
  page.getByRole("link", { name: "Click Here" }).click(),
]);
await popup.waitForLoadState();
```

Iframes

Interact with content inside an iframe using frameLocator.

```
await page.frameLocator("#mce_0_ifr")
  .locator("body")
  .fill("Hello from Playwright!");
```

File upload

Upload a file via input element.

```
await page.setInputFiles('input[type="file"]', "tests/fixtures/sample.txt");
```

Download

Wait for and validate a download triggered by user action.

```
const download = await Promise.all([
  page.waitForEvent("download"),
  page.getByRole("link", { name: "some-file.txt" }).click(),
```

```
]).then(([d]) => d);
expect(await download.path()).toBeTruthy();
```

Dialog

Accept or dismiss browser dialogs (alert/confirm/prompt).

```
page.once("dialog", async (d) => await d.accept());
await page.getByRole("button", { name: "Click for JS Alert" }).click();
```

Wait for response

Synchronize with backend by waiting for a specific response.

```
const res = await page.waitForResponse(r => r.url().includes("/posts/1") && r.status() === 200);
```

API setup

Create/prepare data via APIRequestContext (fast and reliable).

```
const res = await request.get("https://jsonplaceholder.typicode.com/posts/1");
expect(res.ok()).toBeTruthy();
```

Network mocking

Stub a response to make the UI deterministic.

```
await page.route("**/api/profile", route => route.fulfill({
  status: 200,
  contentType: "application/json",
  body: JSON.stringify({ name: "Mock User" }),
}));
```

Evidence on failure (config)

Capture screenshots/videos/traces to debug failures quickly.

```
// playwright.config.ts
use: {
  screenshot: "only-on-failure",
  trace: "on-first-retry",
  video: "retain-on-failure",
}
```

Capstone: End-to-End Practice Suite

During the course, we'll build a capstone suite that exercises the full set of user actions. The sample pack uses a stable public demo site for UI practice plus a public API for request validation.

Item	What students will do
UI Demo Site (Practice)	https://the-internet.herokuapp.com/ (login, checkboxes, dropdown, frames, alerts, uploads, downloads, drag-drop, windows)
API Demo (Practice)	https://jsonplaceholder.typicode.com/ (APIRequestContext examples)
Suite Design	Multiple focused tests (UI auth, forms, windows/frames, file ops, alerts) + an API-only test + optional mocking demo.
Success Criteria	All tests pass locally; reports generated; failures produce evidence (trace/screenshot).

Trainer note: Keep each test small and reliable. Prefer role/label/testid locators, avoid hard waits, and capture traces on first retry.