Browser, BrowserContext, and Page

1. Browser

- Represents an actual **browser instance** (like Chromium, Firefox, or WebKit).
- Created using playwright.chromium.launch(), firefox.launch(), or webkit.launch().
- It's a **heavyweight object**—ideally created once per test suite.
- Supports **headless** or **headed** mode.

Example:

```
import { chromium, Browser } from 'playwright';

const browser: Browser = await chromium.launch({ headless: false });
```

2. BrowserContext

- Think of it as a **new, isolated user session** (like an incognito profile).
- It **shares the underlying browser instance** but has separate cookies, cache, local storage, etc.
- You can have **multiple contexts in one browser**—ideal for multi-user testing.

Why use it?

- Isolation between tests.
- Efficient parallel test execution using the same browser process.

Example:

```
import { BrowserContext } from 'playwright';

const context: BrowserContext = await browser.newContext();
```

3. Page

- Represents a **tab** inside a browser context.
- Most interactions (click, fill, navigate, etc.) happen through the Page object.
- You can create **multiple pages per context**.

Example:

```
import { Page } from 'playwright';

const page: Page = await context.newPage();

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

await page.click('text=Login');
```

Hierarchy Overview

Browser

└── BrowserContext (1..n)

  └── Page (1..n)

context.waitForEvent('page') and page.waitForEvent('popup')

These methods help you **wait for new pages or popups** triggered by user actions, such as clicking on a link or button that opens a new tab or window.

context.waitForEvent('page')

- Waits for a **new tab or window (Page)** to be opened from the current **BrowserContext**.
- Useful when the app opens an external link in a **new browser tab** (not just a popup from the same page).
- Returns a Page object representing the new tab.

Example:

```
const newPage = await context.waitForEvent('page');
```

Common Use Case:

Used with links that open new browser tabs or windows, such as external sites or "open in new tab" links.

page.waitForEvent('popup')

- Waits for a **popup**.
- The popup is **child of the current page**.

Example:

```
const popup = await page.waitForEvent('popup');
```

Common Use Case:

Used when clicking a button or link opens a **popup window** (e.g., login popup, payment gateway, etc.).

Why Promise.all([...])?

When opening a new page or popup, you must wait **simultaneously** for:

1. The event (popup/page),
2. The user action (click).

❗ **Mistake to avoid:**

Don't do this:

```
await page.click('#PopUp'); // this may complete before the popup is detected
const popup = await page.waitForEvent('popup'); // too late!
```

Instead, use Promise.all([...]) to **synchronize** both actions.

Example 1: Handle Popup from a Page

```
await Promise.all([
  page.waitForEvent('popup'),       // Wait for popup
  page.locator('#PopUp').click()    // Click triggers popup
]);
```

Example 2: Handle New Tab from a Context

```
await Promise.all([
  context.waitForEvent('page'),    // Wait for new page/tab
  parentPage.locator("a:has-text('OrangeHRM, Inc')").click()  // Click triggers new tab
]);
```