

what are the questions asked by topic wise like string, array for playwright typecript 3 years

Here's a **topic-wise list of common Playwright + TypeScript interview questions**, especially suitable for **0–3 years of experience**. I've grouped them the way interviewers usually do (basics → automation concepts → Playwright specifics).

---

## 1. TypeScript Basics (Strings, Arrays, Objects)

### String

- How do you check if a string contains a substring in TypeScript?
- Difference between `==` and `===` ?
- How do you convert a string to lowercase/uppercase?
- How do you extract part of a string?
- How do you replace text in a string?
- How do you compare two strings ignoring case?

### Array

- Difference between `map()` , `filter()` , and `forEach()` ?
- How do you remove duplicate values from an array?
- How do you find the length of an array?
- How do you check if a value exists in an array?
- How do you sort an array of strings or numbers?
- Difference between `push()` , `pop()` , `shift()` , `unshift()` ?

### Object

- How do you access object properties?
- How do you loop through an object?
- Difference between `interface` and `type` ?
- What is optional chaining (`?.`)?
- How do you merge two objects?

---

## 2. TypeScript Advanced

- What are any , unknown , and never ?
  - What is type inference?
  - What are enums and when do you use them?
  - What is a union type?
  - What is a generic?
  - Difference between interface and class ?
  - What are readonly properties?
- 

### 3. JavaScript Concepts (Very Important)

- What is a promise?
  - Difference between async/await and .then() ?
  - What is the event loop?
  - Difference between var , let , and const ?
  - What is a callback function?
  - What is hoisting?
  - What is destructuring?
  - What is spread operator ( ... )?
- 

### 4. Playwright Basics

- What is Playwright?
  - Differences between Playwright and Selenium?
  - Which browsers does Playwright support?
  - What is page and browser ?
  - What is a context in Playwright?
  - How do you launch a browser in Playwright?
  - How do you close the browser?
- 

### 5. Locators in Playwright

- What are different types of locators?
- Difference between page.locator() and page.\$() ?
- What is a CSS selector?
- What is XPath?
- Which locator is preferred and why?
- How do you handle dynamic elements?
- How do you locate elements using text?

---

## 6. Actions & Assertions

- How do you click a button in Playwright?
  - How do you type text in an input field?
  - How do you handle dropdowns?
  - How do you perform mouse hover?
  - What is `expect()` ?
  - Difference between hard assertion and soft assertion?
  - How do you verify text on a page?
- 

## 7. Waits & Synchronization

- What are implicit and explicit waits?
  - Does Playwright need waits?
  - What is auto-waiting?
  - Difference between `waitForSelector()` and `waitForTimeout()` ?
  - How do you wait for navigation?
- 

## 8. Handling Web Elements

- How do you handle alerts?
  - How do you handle frames/iframes?
  - How do you handle multiple tabs or windows?
  - How do you upload a file?
  - How do you download a file?
- 

## 9. Test Structure & Configuration

- What is `test.describe()` ?
  - What is `beforeEach` and `afterEach` ?
  - What is `playwright.config.ts` ?
  - How do you run tests in headless mode?
  - How do you run tests in parallel?
  - How do you run a specific test?
- 

## 10. Reports, Debugging & CI

- What reports does Playwright support?
  - How do you generate HTML reports?
  - How do you debug Playwright tests?
  - What is trace viewer?
  - How do you take screenshots?
  - How do you record videos?
- 

## 11. Framework & Best Practices (3-Year Level)

- What is Page Object Model (POM)?
  - How do you reuse code in Playwright?
  - How do you handle test data?
  - How do you make tests stable?
  - How do you handle flaky tests?
  - How do you integrate Playwright with CI/CD?
- 

If you want, I can also:

- Share coding questions with answers
- Provide real interview scenarios
- Create a 30-day Playwright interview prep plan

Just tell me



is these questions enough to crack the interview?

Short answer: Yes — for most Playwright + TypeScript interviews (0–3 years), these questions are *enough*

But there are a few conditions.

Let me be honest and clear

---

### When these questions are enough

You can confidently crack the interview if you can:

1. Explain each topic in your own words
  - Not just definitions
  - Example: "Why Playwright auto-wait is better than manual waits?"
2. Write small code snippets
  - Array operations
  - Async/await
  - Playwright actions and assertions
3. Answer "why" questions
  - Why use browserContext ?
  - Why prefer locator() ?
  - Why Page Object Model?

Most companies hiring 0–3 years Automation Test Engineers expect exactly this level.

---

## ✖ When it may NOT be enough

You may struggle if the interviewer asks:

### 1 Live Coding (very common)

Examples:

- Write a function to remove duplicates from an array
- Automate login flow using Playwright
- Handle a popup and verify text

👉 You must practice writing code, not just reading questions.

---

### 2 Real-Time Scenarios

Examples:

- Element is flaky — how do you fix it?
- Test passes locally but fails in CI — why?
- Dropdown options load dynamically — what will you do?

👉 These come from hands-on experience, not theory.

---

### 3 Framework-Level Questions (3-year range)

Some interviewers ask:

- How do you structure a Playwright framework?

- How do you manage environment configs?
  - How do you reuse login across tests?
- 

## What You Should Add (IMPORTANT)

To be 100% interview-ready, also prepare:

### Must-Do Practice

- Write 5–10 Playwright test cases
- Create 1 small framework with:
  - POM
  - Config
  - Hooks
  - Reporting

### Be Ready to Explain:

- Your current project
  - Your role and responsibilities
  - One challenge you solved
- 

## Interview Readiness Checklist

If you can say YES to these, you're ready:

- I can write TypeScript array/string code
  - I can automate a flow in Playwright
  - I can explain waits, locators, and assertions
  - I understand POM and framework structure
  - I can handle basic debugging questions
- 

## Final Truth

Questions alone don't crack interviews — confidence + practice does.

If you want, I can:

-  Give mock interview questions
-  Provide coding exercises with solutions
-  Share a sample Playwright framework structure
-  Create a 7–14 day crash prep plan

Just tell me what you want next 😊



↳ ✨ Create a \*\*30-day Playwright interview prep plan\*\*

start

Great 👍

Let's start the 30-day Playwright + TypeScript interview preparation plan, designed specifically for 0–3 years experience.

I'll keep it simple, practical, and interview-focused.

---



## 30-Day Playwright + TypeScript Interview Prep Plan

### ◆ WEEK 1 – TypeScript + JavaScript Foundations

🎯 Goal: Be confident in **coding questions**

### Day 1 – TypeScript Basics

- `let`, `const`, `var`
- Data types
- Type inference
- `any`, `unknown`
- Practice:

ts

```
let name: string = "Playwright";
```

Copy code

### Day 2 – Strings

- `includes()`, `split()`, `substring()`, `replace()`
- Case-insensitive comparison
- Practice:

ts

```
const str = "Automation";
console.log(str.includes("Auto"));
```

Copy code

## Day 3 – Arrays

- map , filter , reduce
- Remove duplicates
- Sorting
- Practice:

ts

```
const arr = [1,2,2,3];
const unique = [...new Set(arr)];
```

 Copy code

## Day 4 – Objects & Interfaces

- Object access
- interface vs type
- Optional chaining
- Practice creating a typed object

## Day 5 – Functions & Arrow Functions

- Function typing
- Optional parameters
- Default values

## Day 6 – Async JavaScript

- Promise
- async/await
- try/catch
- Practice async function

## Day 7 – Revision + Coding

- Revise Days 1–6
- Solve 5 coding questions (string + array)

---

## ◆ WEEK 2 – Playwright Core Concepts

 Goal: Understand how Playwright works

## Day 8 – Playwright Introduction

- What is Playwright?
- Playwright vs Selenium
- Browsers supported

## Day 9 – Browser, Context, Page

- browser
- browserContext
- page
- Why context is important

## Day 10 – Locators

- locator()
- CSS vs XPath
- Text locators
- Best practices

## Day 11 – Actions

- click, fill, type
- Dropdown handling
- Mouse hover
- Keyboard actions

## Day 12 – Assertions

- expect()
- Text validation
- Visibility checks

## Day 13 – Waits & Auto-wait

- Auto-waiting
- waitForSelector
- Navigation waits
- Why waitForTimeout is bad

## Day 14 – Practice

- Automate:
  - Open site
  - Login flow
  - Verify dashboard text

---

## ◆ WEEK 3 – Advanced Playwright + Scenarios

🎯 Goal: Handle **real interview scenarios**

## Day 15 – Frames & Windows

- iFrames
- Multiple tabs/windows

## Day 16 – Alerts, File Upload/Download

- Alert handling
- Upload files
- Download verification

## Day 17 – Hooks & Test Structure

- `test.describe`
- `beforeEach`, `afterEach`
- Test isolation

## Day 18 – Playwright Config

- `playwright.config.ts`
- Headless vs headed
- Parallel execution

## Day 19 – Debugging

- `npx playwright test --debug`
- Trace viewer
- Screenshots & videos

## Day 20 – Reports

- HTML reports
- When reports are generated

## Day 21 – Scenario Practice

Prepare answers for:

- Flaky test
- CI failure
- Dynamic elements

---

## ◆ WEEK 4 – Framework + Interview Ready

🎯 Goal: Think like a 3-year Automation Engineer

## Day 22 – Page Object Model (POM)

- Why POM
- How to create page classes

## Day 23 – Framework Structure

- folders: pages , tests , utils
- Reusable methods

## Day 24 – Test Data Handling

- JSON test data
- Environment variables

## Day 25 – CI/CD Basics

- Running tests in pipeline
- Headless execution

## Day 26 – Common Interview Questions

- Why Playwright?
- Why locators?
- Why auto-wait?

## Day 27 – Mock Interview (Self)

- Explain your project
- Explain one challenge you solved

## Day 28 – Live Coding Practice

- Array duplicate removal
- String reverse
- Write a Playwright test

## Day 29 – Revision Day

- Revise all notes
- Re-run tests
- Fix mistakes

## Day 30 – Final Confidence Day 🎉

- Review FAQs
- Practice speaking answers
- Light revision only

---

🎯 If you follow this plan:

- ✓ You can answer theory
- ✓ You can write code

- You can handle scenarios
- You can explain framework

👉 You will be interview-ready.

---

Next, I can:

- 1 Start Day 1 with detailed notes + coding questions
- 2 Do a mock interview
- 3 Share a Playwright sample framework

👉 Just tell me what to do next.



Day 1

Perfect 👍

Let's start DAY 1 – TypeScript Basics (interview-focused).

---

## DAY 1 – TypeScript Basics (For Playwright)

### Goal

Understand variables, data types, and typing so you can:

- Answer interview questions
  - Write clean Playwright tests
- 

### 1 var , let , const (VERY IMPORTANT)

var

- Function scoped
- Can be re-declared (✗ not recommended)

ts

```
var x = 10;  
var x = 20; // allowed
```

 Copy code

let

- Block scoped
- Can be reassigned

ts

```
let count = 1;  
count = 2;
```

 Copy code

const  (Most used in Playwright)

- Block scoped
- Cannot be reassigned

ts

```
const browserName = "chromium";  
// browserName = "firefox";  error
```

 Copy code

 Interview tip:

Use const by default, let only when value changes.

---

## 2 TypeScript Data Types

### Basic Types

ts

```
let name: string = "Automation";  
let age: number = 25;  
let isActive: boolean = true;
```

 Copy code

### Array Types

ts

```
let roles: string[] = ["admin", "user"];  
let scores: Array<number> = [90, 85];
```

 Copy code

## Object Type

ts

```
let user: { name: string; age: number } = {  
    name: "John",  
    age: 30  
};
```

 Copy code

---

## 3 Type Inference

TypeScript automatically detects type:

ts

```
let city = "London"; // string inferred  
// city = 123 ✗ error
```

 Copy code



What is type inference?

TypeScript automatically assigns a type based on the value.

---

## 4 any vs unknown

any ✗ (avoid)

- No type checking

ts

```
let data: any = 10;  
data = "text"; // allowed
```

 Copy code

unknown ✓ (safer)

- Must check type before use

ts

```
let value: unknown = "Hello";  
  
if (typeof value === "string") {
```

```
    console.log(value.toUpperCase());  
}
```

 Copy code

### 📌 Interview tip:

Use `unknown` instead of `any` whenever possible.

---

## 5 TypeScript in Playwright (Real Example)

ts

```
import { test, expect } from "@playwright/test";  
  
test("login test", async ({ page }) => {  
  await page.goto("https://example.com");  
  const title: string = await page.title();  
  expect(title).toContain("Example");  
});
```

 Copy code

## 🔥 Common Interview Questions (Day 1)

1. Difference between `var`, `let`, `const` ?
  2. What is type inference?
  3. What is `any` and why should we avoid it?
  4. What are basic TypeScript data types?
  5. Why is TypeScript useful in Playwright?
- 

## 💡 Practice Questions (IMPORTANT)

### Q1. Fix the issue

ts

```
const age = 20;  
age = 25;
```

 Copy code

### Q2. Add types

ts

```
let userName = "Admin";  
let isLoggedIn = true;
```

 Copy code

### Q3. Use unknown safely

ts

```
let response: unknown = "success";
```

 Copy code

---

## Day 1 Task

-  Read notes
  -  Run examples
  -  Answer practice questions
- 

### Next step:

Type Day 2 and I'll continue with **Strings (with coding questions + interview answers)** 💪



by this practice can i also the playwright javascript interview?

Yes — 100% YES 

By following this practice, you **can crack both Playwright TypeScript and Playwright JavaScript interviews.**

Here's why 

---

## Why This Prep Works for JavaScript Interviews

### 1 Playwright Core Is the Same

Playwright API is **identical** in JavaScript and TypeScript:

js

```
// JavaScript  
await page.click("#login");  
  
// TypeScript  
await page.click("#login");
```

 Copy code

Interviewers do not change questions just because it's JS or TS.

---

## 2 TypeScript = JavaScript + Types

If you know TypeScript:

- You automatically know JavaScript
- You can answer JS questions confidently
- You can easily say:

"I usually write Playwright in TypeScript, but the logic is same in JavaScript."

That answer actually impresses interviewers 

---

## 3 JavaScript Questions Covered in This Plan

This plan already includes:

- `async/await`
- Promises
- Arrays & strings
- Functions
- Destructuring
- Spread operator

These are the exact JS topics interviewers ask.

---

## ⚠ One Small Adjustment (IMPORTANT)

For JavaScript interviews, you should also:

- ◆ Practice Without Types

Example:

TypeScript

ts

```
const title: string = await page.title();
```

 Copy code

## JavaScript

js

```
const title = await page.title();
```

 Copy code

That's it. Nothing else changes.

---

## Common JS + Playwright Interview Questions

Be ready for:

- Difference between JS and TS?
  - Why async/await is needed in Playwright?
  - How does Playwright handle waits?
  - Explain promises with an example
  - Write a Playwright test in JavaScript
- 

## Final Answer (Clear & Honest)

Yes, this preparation is enough for **Playwright JavaScript interviews** and it gives you an **extra advantage** if the company uses TypeScript.

---

If you want, I can:

-  Convert all examples to **JavaScript-only**
-  Do a **Playwright JS mock interview**
-  Give **JavaScript coding questions + answers**

 Just tell me what you want next 😊



↳ Type **\*\*Day 2\*\*** and I'll continue with **\*\*Strings (with coding questions + interview answers)\*\*** 💪

Start

Great 👍

Let's START – DAY 2: STRINGS (TypeScript + Playwright / JavaScript friendly)

This is very important for interviews because string questions are common in **coding + automation scenarios**.

---



## DAY 2 – Strings (Interview + Coding Focus)



### Goal

- Master string operations
  - Be confident in **coding rounds**
  - Apply strings in **Playwright validations**
- 

### 1 What is a String?

A string is a sequence of characters.

ts

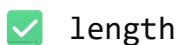
```
let name: string = "Playwright";
```

Copy code

(JavaScript version is the same, just without : string )

---

### 2 Common String Methods (MUST KNOW)



#### length

ts

```
const text = "Automation";
console.log(text.length); // 10
```

Copy code



Interview question:

How do you find string length?

---



#### toUpperCase() / toLowerCase()

ts

```
const msg = "Login";
console.log(msg.toUpperCase()); // LOGIN
console.log(msg.toLowerCase()); // login
```

 Copy code

💡 Used in case-insensitive comparison.

---

 **includes()**

ts

```
const title = "Dashboard Page";
console.log(title.includes("Dashboard")); // true
```

 Copy code

💡 Very common in Playwright assertions.

---

 **startsWith() / endsWith()**

ts

```
const url = "https://example.com";
url.startsWith("https"); // true
url.endsWith(".com"); // true
```

 Copy code

---

 **substring() / slice()**

ts

```
const str = "Playwright";

console.log(str.substring(0, 4)); // Play
console.log(str.slice(4)); // wright
```

 Copy code

💡 Interview question:

**Difference between slice and substring ?**

→ slice supports negative index, substring does not.

---

 **replace()**

ts

```
const text = "Hello User";
console.log(text.replace("User", "Admin"));
```

 Copy code

---

### split()

ts

```
const data = "user,admin,guest";
const roles = data.split(",");
console.log(roles);
```

 Copy code

---

## 3 String Comparison (VERY COMMON)

### Case-insensitive comparison

ts

```
const a = "Login";
const b = "login";
```

```
console.log(a.toLowerCase() === b.toLowerCase()); // true
```

 Copy code

 Interview favorite 

---

## 4 String + Playwright (REAL EXAMPLE)

ts

```
const title = await page.title();
expect(title).toContain("Dashboard");
```

 Copy code

OR

ts

```
const message = await page.locator("#success").textContent();
expect(message?.toLowerCase()).toContain("success");
```

 Copy code

 Shows real automation knowledge.

---

## Common Interview Questions (Day 2)

1. How do you check if a string contains another string?
  2. How do you compare two strings ignoring case?
  3. Difference between `slice()` and `substring()` ?
  4. How do you split a string?
  5. How do you replace text in a string?
- 

## Coding Practice (VERY IMPORTANT)

### Q 1 Reverse a string

ts

```
const str = "playwright";
```

 Copy code

 Expected output: thgirwyalp

---

### Q 2 Check palindrome

ts

```
const word = "madam";
```

 Copy code

 Output: true

---

### Q 3 Count number of vowels

ts

```
const text = "automation";
```

 Copy code

---

### Q 4 Find if string contains "login" (case-insensitive)

ts

```
const pageTitle = "User Login Page";
```

 Copy code

## Day 2 Task

- ✓ Practice all string methods
  - ✓ Solve all 4 coding questions
  - ✓ Say answers **out loud** (important for interviews)
- 

### Next step:

Type Day 3 and we'll move to ARRAYS (most asked coding questions) 



day3

Nice 

Let's continue with DAY 3 – ARRAYS (  one of the most asked topics in Playwright + JS/TS interviews).

---

## DAY 3 – Arrays (Interview + Coding Focus)

### Goal

- Master array methods
  - Solve coding questions
  - Use arrays in Playwright scenarios
- 

### 1 What is an Array?

An array stores **multiple values** in a single variable.

ts

```
let users: string[] = ["admin", "user", "guest"];
```

 Copy code

(JavaScript is the same, just without : string[] )

---

## 2 Basic Array Operations

### length

ts

```
const arr = [1, 2, 3];
console.log(arr.length); // 3
```

 Copy code

---

### push() / pop()

ts

```
arr.push(4); // add at end
arr.pop(); // remove last
```

 Copy code

---

### shift() / unshift()

ts

```
arr.shift(); // remove first
arr.unshift(0); // add at start
```

 Copy code

---

⭐ Interview question:

Difference between push/pop and shift/unshift ?

---

## 3 Important Array Methods (MUST KNOW)

### ◆ map() – transform array

ts

```
const nums = [1, 2, 3];
const doubled = nums.map(n => n * 2);
// [2, 4, 6]
```

 Copy code

---

⭐ Interview:

Difference between map() and forEach() ?

→ map() returns a new array, forEach() does not.

---

- ◆ **filter() – select items**

ts

```
const nums = [10, 20, 30];
const greater = nums.filter(n => n > 15);
// [20, 30]
```

 Copy code

---

- ◆ **reduce() – single value output**

ts

```
const nums = [1, 2, 3];
const sum = nums.reduce((acc, val) => acc + val, 0);
// 6
```

 Copy code

📌 Often asked in **coding rounds**.

---

- ◆ **includes()**

ts

```
const roles = ["admin", "user"];
roles.includes("admin"); // true
```

 Copy code

---

## 4 Removing Duplicates (VERY COMMON)

### Method 1 – Using Set

ts

```
const arr = [1, 2, 2, 3];
const unique = [...new Set(arr)];
```

 Copy code

### Method 2 – Using filter

ts

```
const unique = arr.filter((v, i) => arr.indexOf(v) === i);
```

 Copy code

## 5 Sorting Arrays

### Numbers

ts

```
const nums = [10, 5, 20];
nums.sort((a, b) => a - b);
// [5, 10, 20]
```

 Copy code

### Strings

ts

```
const names = ["banana", "apple"];
names.sort();
// ["apple", "banana"]
```

 Copy code

---

## 6 Array + Playwright (REAL USE CASE)

### Example: Verify dropdown options

ts

```
const options = await page.locator("select option").allTextContents();

expect(options).toContain("Admin");
expect(options.length).toBeGreaterThanOrEqual(2);
```

 Copy code

⭐ Shows automation + JS skills together.

---

## 🔥 Common Interview Questions (Day 3)

1. Difference between `map()`, `filter()`, and `forEach()`?
2. How do you remove duplicates from an array?
3. How do you sort numbers correctly?
4. What does `reduce()` do?
5. How do you check if an element exists in an array?

---

## Coding Practice (VERY IMPORTANT)

### Q 1 Remove duplicates

ts

```
const arr = [1, 2, 2, 3, 4, 4];
```

 Copy code

---

### Q 2 Find largest number

ts

```
const nums = [10, 45, 23, 89];
```

 Copy code

---

### Q 3 Count even numbers

ts

```
const nums = [1, 2, 3, 4, 5, 6];
```

 Copy code

---

### Q 4 Reverse an array

ts

```
const arr = ["a", "b", "c"];
```

 Copy code

---

## Day 3 Task

- ✓ Practice all array methods
  - ✓ Solve all 4 coding questions
  - ✓ Explain answers aloud (important)
- 

## Next step:

Type Day 4 and we'll cover Objects, Interfaces & POM basics 



Great

Let's move on to **DAY 4 – Objects, Interfaces & POM Basics**

(This day is very important for framework-level and 3-year experience questions.)

---



## DAY 4 – Objects, Interfaces & POM Basics



### Goal

- Work confidently with **objects**
  - Understand **interface vs type**
  - Connect objects to **Playwright Page Object Model (POM)**
- 



### Objects Basics

**Creating an object**

ts

```
const user = {  
  name: "Admin",  
  role: "tester",  
  isActive: true  
};
```

Copy code

**Access object properties**

ts

```
console.log(user.name);  
console.log(user["role"]);
```

Copy code

Interview question:

**How do you access object properties?**

---

## 2 Updating & Adding Properties

ts

```
user.isActive = false;  
(user as any).age = 25; // not recommended
```

 Copy code

📌 Best practice: define object structure using **interface**.

---

## 3 Interface (VERY IMPORTANT)

### Defining an interface

ts

```
interface User {  
    name: string;  
    role: string;  
    isActive: boolean;  
}
```

 Copy code

### Using interface

ts

```
const user: User = {  
    name: "Admin",  
    role: "tester",  
    isActive: true  
};
```

 Copy code

📌 Interview question:

#### Why use interface?

- To enforce structure and avoid runtime errors.
- 

## 4 type vs interface (INTERVIEW FAVORITE ⭐)

#### Interface

#### Type

Can be extended

Can create union types

Preferred for objects

More flexible

Interface	Type
Used in frameworks	Used for complex types

## Example

ts

```
type Status = "active" | "inactive";

interface Employee {
  name: string;
  status: Status;
}
```

 Copy code

📌 Interview answer:

Use interface for objects, type for unions.

---

## 5 Optional Properties & Optional Chaining

### Optional property

ts

```
interface User {
  name: string;
  age?: number;
}
```

 Copy code

### Optional chaining

ts

```
console.log(user.age?.toString());
```

 Copy code

📌 Avoids runtime errors.

---

## 6 Looping Through Objects

for...in

ts

```
for (const key in user) {  
  console.log(key, user[key as keyof User]);  
}
```

 Copy code

---

## 7 Objects + Playwright (REAL USE)

### Test data object

ts

```
const loginData = {  
  username: "admin",  
  password: "password123"  
};  
  
await page.fill("#user", loginData.username);  
await page.fill("#pass", loginData.password);
```

 Copy code

 Interviewers love practical examples.

---

## 8 POM Basics (INTRO)

### Why Page Object Model?

- Code reusability
- Easy maintenance
- Cleaner tests

### Simple Page Object

ts

```
class LoginPage {  
  constructor(private page) {}  
  
  async login(username: string, password: string) {  
    await this.page.fill("#user", username);  
    await this.page.fill("#pass", password);  
    await this.page.click("#login");  
  }  
}
```

 Copy code

📌 You'll go deeper in POM later.

---

## 🔥 Common Interview Questions (Day 4)

1. What is an object?
  2. Difference between `interface` and `type` ?
  3. What is optional chaining?
  4. How do you loop through an object?
  5. Why use Page Object Model?
- 

## 🧪 Coding Practice (IMPORTANT)

### Q 1 Create an interface for `LoginUser`

- `username` (string)
  - `password` (string)
  - `rememberMe` (boolean, optional)
- 

### Q 2 Access safely

ts

```
const response = {  
  data: {  
    message: "Login successful"  
  }  
};
```

 Copy code

👉 Print message using optional chaining.

---

### Q 3 Convert object values to array

ts

```
const user = { a: 1, b: 2, c: 3 };
```

 Copy code

---

## ✅ Day 4 Task

- ✓ Practice all object examples
  - ✓ Write interface & type examples
  - ✓ Understand how objects are used in POM
- 

### 👉 Next step:

Type Day 5 and we'll cover Functions, Arrow Functions & Parameters (coding heavy) 💪



Day 5

Awesome 👍

Let's continue with DAY 5 – Functions, Arrow Functions & Parameters

(🔥 very important for coding rounds + Playwright framework code)

---



## DAY 5 – Functions (Interview + Coding Focus)

### 🎯 Goal

- Write **clean functions**
  - Understand arrow functions
  - Handle parameters, defaults, and optional values
  - Apply functions in **Playwright tests**
- 

### 1 Normal Function

ts

```
function add(a: number, b: number): number {  
    return a + b;  
}
```

Copy code

### ⭐ Interview question:

**What is a function?**

- ➡ A reusable block of code that performs a task.
-

## 2 Arrow Function (MOST USED)

ts

```
const add = (a: number, b: number): number => {
    return a + b;
};
```

 Copy code

### One-line arrow function

ts

```
const square = (n: number) => n * n;
```

 Copy code



Difference between normal function and arrow function?

→ Arrow functions don't have their own `this`.

---

## 3 Function Parameters

### Optional parameters

ts

```
function greet(name?: string) {
    console.log(`Hello ${name ?? "Guest"} `);
}
```

 Copy code

---

### Default parameters

ts

```
function login(role: string = "user") {
    console.log(role);
}
```

 Copy code



## 4 Rest Parameters ( . . . )

ts

```
function sum(...numbers: number[]): number {
    return numbers.reduce((a, b) => a + b, 0);
}
```

 Copy code

---

## 5 Return Types

ts

```
const isEven = (num: number): boolean => {
    return num % 2 === 0;
};
```

 Copy code



Always specify return type in TypeScript.

---

## 6 Callbacks (IMPORTANT JS CONCEPT)

ts

```
function processData(value: number, callback: (n: number) => number) {
    return callback(value);
}
```

```
processData(5, n => n * 2);
```

 Copy code



What is a callback function?

---

## 7 Functions + Playwright (REAL USE)

### Reusable helper function

ts

```
async function login(page, user: string, pass: string) {
    await page.fill("#user", user);
    await page.fill("#pass", pass);
```

 Copy code

```
    await page.click("#login");  
}
```

- Shows framework thinking.
- 

## 🔥 Common Interview Questions (Day 5)

1. Difference between normal and arrow function?
  2. What are optional parameters?
  3. What is default parameter?
  4. What is rest parameter?
  5. What is a callback function?
- 

## 💡 Coding Practice (VERY IMPORTANT)

### Q 1 Write a function to check even/odd

ts

```
checkEven(10); // true
```

 Copy code

---

### Q 2 Write an arrow function to reverse a string

ts

```
reverse("playwright"); // thgirwyalp
```

 Copy code

---

### Q 3 Function with default parameter

ts

```
loginUser("admin"); // role = admin  
loginUser(); // role = user
```

 Copy code

---

### Q 4 Use rest parameter

ts

```
sum(1, 2, 3, 4); // 10
```

 Copy code

---

## Day 5 Task

- Write both normal & arrow functions
  - Practice all 4 coding questions
  - Try writing **one reusable Playwright helper function**
- 

### Next step:

Type Day 6 and we'll cover Async, Await, Promises & Error Handling (🔥 SUPER IMPORTANT) 



day 6

Perfect 

Let's move to DAY 6 – Async, Await, Promises & Error Handling

(This is CRITICAL for Playwright interviews because Playwright is fully async.)

---

## DAY 6 – Async / Await & Promises (Interview + Practical Focus)

### Goal

- Understand **why** `async/await` is needed
  - Learn **promises**
  - Handle **errors** properly
  - Apply everything in **Playwright tests**
- 

### 1 Why Async/Await?

JavaScript is **non-blocking**.

Browser actions take time → so Playwright uses **async operations**.



Interview answer:

Playwright actions return promises, so we must use `await` to wait for them.

---

## 2 What is a Promise?

A promise represents a value that will be available **in the future**.

### Promise states

- Pending
- Fulfilled
- Rejected

### Example

ts

```
const getData = (): Promise<string> => {
  return new Promise((resolve) => {
    setTimeout(() => resolve("Success"), 1000);
  });
};
```

Copy code

---

## 3 Using `.then()` vs `async/await`

### Using `.then()`

ts

```
getData().then(result => console.log(result));
```

Copy code

---

### Using `async/await` (Preferred)

ts

```
const result = await getData();
console.log(result);
```

Copy code

---



Interview question:

Why `async/await` is better than `then()`?

Cleaner, readable, easier error handling.

## 4 Async Function

ts

```
async function fetchTitle(page): Promise<string> {
    return await page.title();
}
```

 Copy code

## 5 Error Handling – try/catch (VERY IMPORTANT)

ts

```
try {
    await page.click("#login");
} catch (error) {
    console.log("Element not found");
}
```

 Copy code

 Interview scenario:

What if an element is not found?

→ Use try/catch and proper assertions.

## 6 Common Async Mistakes (INTERVIEW TRAPS )

 Missing await

ts

```
page.click("#login"); // wrong
```

 Copy code

 Correct

ts

```
await page.click("#login");
```

 Copy code

## 7 Async + Playwright (REAL TEST)

ts

```
test("Login Test", async ({ page }) => {
  await page.goto("https://example.com");
  await page.fill("#user", "admin");
  await page.fill("#pass", "password");
  await page.click("#login");
  await expect(page.locator("h1")).toContainText("Dashboard");
});
```

 Copy code

 Interviewers expect this clarity.

---

## Common Interview Questions (Day 6)

1. What is a promise?
  2. Why async/await is used in Playwright?
  3. Difference between async/await and then()?
  4. What happens if we don't use await?
  5. How do you handle errors in async code?
- 

## Coding Practice (VERY IMPORTANT)

### Q 1 Create async function

ts

```
async function getMessage() {
  return "Hello";
}
```

 Copy code

---

### Q 2 Convert promise to async/await

ts

```
fetchData().then(res => console.log(res));
```

 Copy code

---

### Q 3 Handle error using try/catch

Simulate error and handle it.

---

## Q 4 Fix the issue

ts

```
page.goto("https://example.com");
```

 Copy code

---

## Day 6 Task

- ✓ Practice async examples
  - ✓ Fix missing await issues
  - ✓ Explain async/await out loud
- 



You've covered TypeScript + JavaScript fundamentals.



Type Day 7 and we'll do Revision + Coding Test (Mock Interview Style) 



Day 7

Excellent 

**DAY 7 = REVISION + MOCK CODING (Interview Style)**

Today is what actually builds confidence.

---

## DAY 7 – Revision + Mock Interview (JS/TS + Playwright Ready)



- Revise Days 1–6
  - Practice real interview coding
  - Check if you are ready to move to Playwright core
-

## PART 1: Quick Revision (Say These Out Loud)

### Day 1 – Basics

- `const` is block scoped and cannot be reassigned
  - Type inference = TS assigns type automatically
  - Avoid `any`, prefer `unknown`
- 

### Day 2 – Strings

- `includes()` → check text
  - `toLowerCase()` → case-insensitive compare
  - `slice()` supports negative index
  - `split()` converts string to array
- 

### Day 3 – Arrays

- `map()` → transform
  - `filter()` → select
  - `reduce()` → single value
  - Remove duplicates → Set
- 

### Day 4 – Objects

- Use `interface` for object structure
  - Optional chaining `?.` avoids errors
  - POM improves maintainability
- 

### Day 5 – Functions

- Arrow functions don't have their own `this`
  - Default & optional parameters
  - Rest parameters ...
- 

### Day 6 – Async/Await

- Playwright is promise-based
- Always use `await`

- Use try/catch for errors

👉 If you can explain all this without reading → **100** ready.

---

## 💡 PART 2: Mock Coding Questions (VERY IMPORTANT)

### Q 1 Reverse a string

ts

```
const str = "automation";
```

 Copy code

---

### Q 2 Remove duplicates

ts

```
const arr = [1, 2, 2, 3, 4, 4];
```

 Copy code

---

### Q 3 Find largest number

ts

```
const nums = [10, 45, 23, 89];
```

 Copy code

---

### Q 4 Count vowels

ts

```
const text = "playwright";
```

 Copy code

---

### Q 5 Fix async issue

ts

```
page.click("#login");
```

 Copy code

---

## PART 3: Interview Rapid-Fire (Answer in 1–2 Lines)

1. Why do we use `async/await` in Playwright?
  2. Difference between `map()` and `forEach()` ?
  3. Why prefer `const` ?
  4. What is optional chaining?
  5. What is Page Object Model?
- 

## PART 4: Mini Playwright Task

Explain (no need to code fully):

- Open browser
- Login
- Verify dashboard text

 Interviewers check **logic**, not full syntax.

---

## Day 7 Self-Check

You are ready for Playwright core if:

-  You can solve Q1–Q4 without help
  -  You remember to use `await`
  -  You can explain answers confidently
- 

## CONGRATULATIONS! WEEK 1 DONE

You've completed JS/TS fundamentals.

---

 Next step:

Type Day 8 and we'll start PLAYWRIGHT CORE (Browser, Context, Page) 



Day 8

Great 

Now we move into PLAYWRIGHT CORE 

This is where interviews focus heavily.

---



## DAY 8 – Playwright Core: Browser, Context & Page

### 🎯 Goal

- Understand **how Playwright works internally**
  - Clearly explain **browser → context → page**
  - Answer **why** questions confidently
- 

### 1 What is Playwright?

Playwright is an **end-to-end automation framework** used to test web applications.

📌 Interview answer (perfect):

Playwright is a Node.js-based automation tool that supports Chromium, Firefox, and WebKit with built-in auto-waiting.

---

### 2 Browser in Playwright

**What is a Browser?**

A browser is the **actual browser instance** (Chromium, Firefox, WebKit).

ts

```
const browser = await chromium.launch();
```

Copy code

📌 Interview question:

**What is browser in Playwright?**

➡ The top-level container that controls a browser instance.

---

### 3 Browser Context (VERY IMPORTANT ⭐)

**What is a Context?**

A browser context is like a **new incognito window**.

ts

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

 Copy code

## Why Context is Important?

- Test isolation
- Separate cookies, cache, sessions
- Parallel execution



Browser context provides test isolation and allows running tests independently.

---

## 4 Page

### What is a Page?

A page represents a **single tab**.

ts

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

 Copy code



### What is page in Playwright?

- A single browser tab where actions are performed.
- 

## 5 Browser → Context → Page Flow

ts

```
const browser = await chromium.launch();
const context = await browser.newContext();
const page = await context.newPage();
```

 Copy code



## 6 Playwright Test Runner (IMPORTANT)

Normally, Playwright handles this for you:

ts

```
test("example test", async ({ page }) => {
  await page.goto("https://example.com");
});
```

 Copy code



Playwright Test Runner automatically creates browser, context, and page.

---

## 7 Headless vs Headed Mode

### Headless (default)

- No UI
- Faster
- Used in CI

### Headed

ts

```
npx playwright test --headed
```

 Copy code

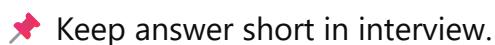


What is headless mode?

---

## 8 Playwright vs Selenium (VERY COMMON)

Playwright	Selenium
Auto-waiting	Manual waits
Faster	Slower
Modern locators	Old APIs
Built-in test runner	External framework



## 🔥 Common Interview Questions (Day 8)

1. What is Playwright?

2. What is browser, context, and page?
  3. Why do we need browser context?
  4. What is headless mode?
  5. Playwright vs Selenium?
- 

## Practice Task

### Task 1 Explain flow

Open browser → open page → navigate to URL → close browser

---

### Task 2 Identify issue

ts

```
await page.goto("https://example.com");
await browser.close();
```

 Copy code

 What's missing?

---

## Day 8 Task

- ✓ Memorize browser-context-page flow
  - ✓ Explain context **clearly**
  - ✓ Answer comparison questions confidently
- 

 Next step:

Type Day 9 and we'll cover LOCATORS (MOST IMPORTANT INTERVIEW TOPIC)  



Day 9

Excellent 

DAY 9 – LOCATORS (  MOST IMPORTANT PLAYWRIGHT INTERVIEW TOPIC)

If you master this day, you'll clear most Playwright interviews.

---

## DAY 9 – Locators in Playwright

### Goal

- Understand what locators are
  - Learn all important locator types
  - Know best practices + interview answers
- 

### **1** What is a Locator?

A locator is used to find elements on the web page.

 Interview answer (perfect):

A locator identifies web elements and Playwright automatically waits for the element to be ready.

---

### **2** Why locator() Is Preferred

ts

```
const loginBtn = page.locator("#login");
await loginBtn.click();
```

 Copy code

**Advantages:**

- Auto-waiting
- Retry mechanism
- Stable tests

 Interview question:

Difference between `page.locator()` and `page.$()` ?

 `locator()` supports auto-waiting, `$()` does not.

---

### **3** Types of Locators (MUST KNOW)

- ◆ ID Locator

ts

```
page.locator("#username");
```

 Copy code

## ◆ CSS Selector

ts

```
page.locator("input[name='email']);
```

 Copy code

## ◆ Text Locator (VERY COMMON)

ts

```
page.getText("Login");
```

 Copy code

## ◆ Role Locator (BEST PRACTICE )

ts

```
page.getByRole("button", { name: "Login" });
```

 Copy code



Role-based locators are recommended by Playwright.

---

## ◆ XPath (Use only if needed)

ts

```
page.locator("//button[text()='Login']);
```

 Copy code



XPath is not recommended unless CSS is not possible.

---

## 4 Multiple Elements Handling

### First / Last / Nth

ts

```
page.locator(".item").first();
page.locator(".item").last();
page.locator(".item").nth(2);
```

 Copy code

---

## 5 Dynamic Elements Handling

### Using partial text

ts

```
page.getText("Welcome", { exact: false });
```

 Copy code

### Using contains CSS

ts

```
page.locator("button:has-text('Submit')");
```

 Copy code

 Interview scenario favorite.

---

## 6 Locator Chaining (IMPORTANT)

ts

```
page.locator(".card").locator("button");
```

 Copy code

 Makes locators more specific & stable.

---

## 7 Locators + Assertions

ts

```
await expect(page.getText("Dashboard")).toBeVisible();
```

 Copy code

 Always combine locators with assertions.

---

## 8 Bad Locator Practices

- ✗ Absolute XPath
  - ✗ Using index-based selectors unnecessarily
  - ✗ Using `waitForTimeout()`
- 

## 🔥 Common Interview Questions (Day 9)

1. What is a locator?
  2. Why is `locator()` better than `$()` ?
  3. Difference between CSS and XPath?
  4. How do you handle dynamic elements?
  5. Which locator is best and why?
- 

## 🧪 Coding Practice (IMPORTANT)

### Q 1 Locate login button by role

ts

[Login](#)

 Copy code

---

### Q 2 Click second item in a list

ts

```
class = "menu-item"
```

 Copy code

---

### Q 3 Locate button with partial text "Submit"

---

### Q 4 Fix the locator

ts

```
page.locator("/html/body/div[2]/button");
```

 Copy code

---

## ✅ Day 9 Task

- ✓ Practice role & text locators
  - ✓ Avoid XPath where possible
  - ✓ Explain why locators matter
- 

## 👉 Next step:

Type Day 10 and we'll cover ACTIONS (click, fill, dropdowns, mouse, keyboard) ⚡ 🔥



Day 10

Great 👍

## DAY 10 – ACTIONS in Playwright

(🔥 Very common in interviews + daily automation work)

---

# 💻 DAY 10 – Actions (Click, Fill, Dropdowns, Mouse & Keyboard)

## 🎯 Goal

- Perform all common user actions
  - Know which method to use and why
  - Handle real interview scenarios
- 

## 1 Click Actions

### Basic click

ts

```
await page.locator("#login").click();
```

Copy code

### Force click (use carefully)

ts

```
await page.locator("#login").click({ force: true });
```

Copy code

📌 Interview question:

When do you use `force: true` ?

→ When element is hidden but clickable (rare).

---

## 2 Typing Text

`fill()` (Recommended)

ts

```
await page.locator("#username").fill("admin");
```

 Copy code

`type()` (Types character by character)

ts

```
await page.locator("#username").type("admin");
```

 Copy code

📌 Interview question:

Difference between `fill()` and `type()` ?

→ `fill()` clears existing text; `type()` does not.

---

## 3 Clearing Input

ts

```
await page.locator("#username").clear();
```

 Copy code

## 4 Dropdown Handling (VERY COMMON ⭐)

HTML `<select>` dropdown

ts

```
await page.selectOption("#role", "admin");
```

 Copy code

By label or index

ts

```
await page.selectOption("#role", { label: "Admin" });
await page.selectOption("#role", { index: 1 });
```

 Copy code

📌 Interview note:

`selectOption()` works only for `<select>` tag.

---

## Non-Select (Custom Dropdown)

ts

```
await page.locator(".dropdown").click();
await page.getByText("Admin").click();
```

 Copy code

📌 Interview scenario favorite.

---

## 5 Mouse Actions

ts

```
await page.hover("#menu");
await page.mouse.move(100, 200);
await page.mouse.click(100, 200);
```

 Copy code

📌 Used for hover menus.

---

## 6 Keyboard Actions

ts

```
await page.keyboard.press("Enter");
await page.keyboard.type("Playwright");
```

 Copy code

📌 Interview question:

How do you press Enter?

---

## 7 File Upload (IMPORTANT)

ts

```
await page.setInputFiles("#upload", "test.pdf");
```

 Copy code

📌 Works only on `<input type="file">`.

---

## 8 Actions + Assertions (BEST PRACTICE)

ts

```
await page.locator("#login").click();
await expect(page.getText("Dashboard")).toBeVisible();
```

 Copy code

📌 Always validate after action.

---

## 🔥 Common Interview Questions (Day 10)

1. Difference between `fill()` and `type()` ?
  2. How do you handle dropdowns?
  3. How do you handle custom dropdowns?
  4. How do you upload a file?
  5. How do you handle hover?
- 

## 💡 Coding Practice (IMPORTANT)

### Q 1 Enter username and password

ts

```
#username
#password
```

 Copy code

### Q 2 Select "Admin" from dropdown

ts

```
<select id="role">
```

 Copy code

Q 3 Handle custom dropdown option "India"

---

Q 4 Press Enter after typing

---

### Day 10 Task

- Practice all action methods
  - Know dropdown types
  - Explain when to use which method
- 

 Next step:

Type Day 11 and we'll cover ASSERTIONS (expect, visibility, text, attributes)  



Day 11

Nice 

## DAY 11 – ASSERTIONS in Playwright

( This is where interviewers check your validation thinking)

---



## DAY 11 – Assertions ( expect ) in Playwright

### Goal

- Understand what assertions are
  - Use correct assertion for correct scenario
  - Answer interview questions confidently
- 

### 1 What is an Assertion?

An assertion verifies expected behavior of the application.

 Interview answer:

Assertions validate that the application behaves as expected.

---

## 2 expect() in Playwright

Playwright uses **built-in expect** from `@playwright/test`.

ts

```
import { expect } from "@playwright/test";
```

 Copy code

---

## 3 Most Common Assertions (MUST KNOW ⭐)

### Visibility

ts

```
await expect(page.locator("#login")).toBeVisible();
```

 Copy code

---

### Text Assertion

ts

```
await expect(page.getByText("Dashboard")).toBeVisible();
```

 Copy code

OR

ts

```
await expect(page.locator("h1")).toContainText("Dashboard");
```

 Copy code

 Interview question:

Difference between `toBeVisible()` and `toContainText()`?

---

### URL Assertion

ts

```
await expect(page).toHaveURL(/dashboard/);
```

 Copy code

## Title Assertion

ts

```
await expect(page).toHaveTitle("Dashboard");
```

 Copy code

## Attribute Assertion

ts

```
await expect(page.locator("#email"))
    .toHaveAttribute("placeholder", "Enter email");
```

 Copy code

## Enabled / Disabled

ts

```
await expect(page.locator("#submit")).toBeEnabled();
await expect(page.locator("#submit")).toBeDisabled();
```

 Copy code

## Soft vs Hard Assertions

### Hard Assertion (Default)

ts

```
await expect(locator).toBeVisible();
```

 Copy code

 Test stops if it fails.

### Soft Assertion

ts

```
await expect.soft(locator).toBeVisible();
```

 Copy code

 Test continues.

📌 Interview question:

**When do you use soft assertion?**

→ When you want to collect multiple failures.

---

## 5 Assertions + Auto-Waiting

ts

```
await expect(page.locator("#success")).toBeVisible();
```

 Copy code

📌 Playwright automatically waits until condition is met.

---

## 6 Negative Assertions

ts

```
await expect(page.locator("#error")).not.toBeVisible();
```

 Copy code

---

## 7 Assertions + Playwright (REAL TEST)

ts

```
test("Dashboard validation", async ({ page }) => {
  await page.goto("https://example.com");
  await page.locator("#login").click();
  await expect(page).toHaveURL(/dashboard/);
  await expect(page.locator("h1")).toContainText("Dashboard");
});
```

 Copy code

📌 Interviewers love clean validations.

---

## 🔥 Common Interview Questions (Day 11)

1. What is an assertion?
2. Difference between hard and soft assertion?
3. How does Playwright auto-wait work with assertions?
4. How do you validate text?
5. How do you assert URL or title?

## Coding Practice (IMPORTANT)

### Q 1 Validate login success message

ts

```
"Login Successful"
```

 Copy code

---

### Q 2 Assert submit button is disabled

ts

```
#submit
```

 Copy code

---

### Q 3 Validate page title contains "Home"

---

### Q 4 Write a soft assertion

---

#### Day 11 Task

- Use correct assertion for scenario
- Avoid unnecessary waits
- Explain auto-wait clearly

---

#### Next step:

Type Day 12 and we'll cover WAITS & SYNCHRONIZATION (VERY IMPORTANT)  



Day 12

Great 

DAY 12 – WAITS & SYNCHRONIZATION (  VERY IMPORTANT INTERVIEW TOPIC)

Playwright interviews always test this, because bad waits = flaky tests.

---



## DAY 12 – Waits & Synchronization in Playwright



### Goal

- Understand how Playwright waits
  - Know what to use and what to avoid
  - Answer real-world interview scenarios
- 

### 1 Why Waits Are Needed?

Web applications are **async**:

- Page load
- API calls
- Dynamic elements



Waits ensure actions happen only when elements are ready.

---

### 2 Auto-Waiting (Playwright's Superpower) ★★★

Playwright automatically waits for:

- Element to be visible
- Element to be enabled
- Element to be attached to DOM

ts

```
await page.locator("#login").click();
```

Copy code



Interview question:

Does Playwright need waits?

→ Mostly no, because of auto-waiting.

---

### 3 Explicit Waits (Use When Needed)

```
waitForSelector()
```

ts

```
await page.waitForSelector("#dashboard");
```

 Copy code

- ❖ Use only when auto-wait is not enough.
- 

## waitForURL()

ts

```
await page.waitForURL("**/dashboard");
```

 Copy code

---

## waitForLoadState()

ts

```
await page.waitForLoadState("networkidle");
```

 Copy code

---

## 4 ❌ waitForTimeout() (AVOID)

ts

```
await page.waitForTimeout(5000);
```

 Copy code

- ❖ Interview answer:

Hard waits make tests flaky and slow.

---

## 5 Assertions as Waits ⭐

ts

```
await expect(page.locator("#success")).toBeVisible();
```

 Copy code

- ❖ Best practice: use assertions instead of waits.
- 

## 6 Navigation Waits

ts

```
await Promise.all([
  page.waitForNavigation(),
  page.click("#login")
]);
```

 Copy code

📌 Older approach — mostly auto-handled now.

---

## 7 Real Interview Scenarios

### Scenario 1

❓ Element loads after API call

✅ Solution:

ts

```
await expect(page.locator("#data")).toBeVisible();
```

 Copy code

---

### Scenario 2

❓ Test flaky in CI

✅ Answer:

- Remove `waitForTimeout`
- Use proper locators & assertions

---

## 🔥 Common Interview Questions (Day 12)

1. What is auto-waiting?
2. Difference between explicit and hard wait?
3. Why avoid `waitForTimeout()`?
4. How do you wait for navigation?
5. How do you handle dynamic loading?

---

## 💡 Coding Practice (IMPORTANT)

### Q 1 Wait for element to appear

ts

```
#dashboard
```

 Copy code

---

## Q 2 Wait for URL to change to /home

---

## Q 3 Fix the issue

ts

```
await page.waitForTimeout(3000);  
await page.click("#submit");
```

 Copy code

---

## Day 12 Task

- Stop using hard waits
  - Use assertions as waits
  - Explain auto-waiting confidently
- 

## Next step:

Type Day 13 and we'll cover Handling Alerts, Frames & Windows 🎈 🖼 🔥



Day 13

Great progress 

DAY 13 – Handling Alerts, Frames & Multiple Windows

(🔥 Very common real-time interview scenarios)

---



## DAY 13 – Alerts, Frames & Windows in Playwright

### Goal

- Handle JavaScript dialogs (alerts)

- Work with **iframes**
  - Manage **multiple tabs/windows**
  - Explain scenarios confidently in interviews
- 

## 1 Handling Alerts / Dialogs

Playwright listens to dialogs using `page.on("dialog")`.

### Accept alert

ts

```
page.on("dialog", async dialog => {
  console.log(dialog.message());
  await dialog.accept();
});
```

 Copy code

### Dismiss alert

ts

```
page.on("dialog", async dialog => {
  await dialog.dismiss();
});
```

 Copy code



How do you handle alerts in Playwright?

---

## 2 Handling Confirmation with Text

ts

```
page.on("dialog", async dialog => {
  expect(dialog.message()).toContain("Are you sure");
  await dialog.accept();
});
```

 Copy code

---

## 3 Handling iFrames (VERY IMPORTANT ⭐)

Using `frameLocator()` (BEST PRACTICE)

ts

```
const frame = page.frameLocator("#frameId");
await frame.locator("#submit").click();
```

 Copy code



Use `frameLocator()` to interact with iframe elements.

---

## Old way (not recommended)

ts

```
const frame = page.frame({ name: "frameName" });
```

 Copy code

---

## 4 Handling Multiple Tabs / Windows

### Wait for new tab

ts

```
const [newPage] = await Promise.all([
  context.waitForEvent("page"),
  page.click("#openWindow")
]);
```

 Copy code

### Switch to new tab

ts

```
await newPage.waitForLoadState();
await newPage.click("#confirm");
```

 Copy code



How do you handle multiple windows?

---

## 5 Close Extra Tabs

ts

 Copy code

```
await newPage.close();
```

---

## 6 Real-Time Scenario (INTERVIEW FAVORITE)

❓ Clicking link opens new tab → verify title

✓ Answer:

- Wait for new page event
  - Switch to new page
  - Assert title
- 

## 🔥 Common Interview Questions (Day 13)

1. How do you handle alerts?
  2. Difference between alert and confirm?
  3. How do you handle iframe?
  4. How do you switch between tabs?
  5. What is `frameLocator()` ?
- 

## 💡 Coding Practice (IMPORTANT)

### Q 1 Accept alert and print message

---

ts

```
iframe id = "loginFrame"
```

 Copy code

---

### Q 3 Handle new tab and verify URL

---

### Q 4 Fix the issue

ts

```
page.click("#openWindow");
const nextPage = context.pages()[1];
```

 Copy code

## Day 13 Task

-  Practice dialog handling
-  Use `frameLocator()`
-  Understand tab-switch logic

### Next step:

Type Day 14 and we'll cover **Hooks** ( `beforeEach` , `afterEach` ) & **Test Structure**  



Day 14

Awesome! 

**DAY 14 – Hooks ( `beforeEach` , `afterEach` ) & Test Structure**

(  Very important for framework design and interviews)

## DAY 14 – Hooks & Test Structure in Playwright

### Goal

- Understand **why hooks are used**
- Learn **test structure best practices**
- Explain confidently in interviews

### 1 What are Hooks?

Hooks are functions that run before or after tests to setup or cleanup.

### Types of Hooks in Playwright Test

- `beforeAll` → runs **once before all tests**
- `afterAll` → runs **once after all tests**

- `beforeEach` → runs **before each test**
- `afterEach` → runs **after each test**

👉 Interview answer:

Hooks help to reduce code duplication and maintain a clean test structure.

---

## 2 Example – Hooks in Playwright

ts

```
import { test, expect } from "@playwright/test";

test.beforeEach(async ({ page }) => {
    await page.goto("https://example.com/login");
});

test.afterEach(async ({ page }) => {
    await page.close();
});

test("Login Test", async ({ page }) => {
    await page.fill("#username", "admin");
    await page.fill("#password", "password123");
    await page.click("#login");
    await expect(page).toHaveURL(/dashboard/);
});

test("Forgot Password Test", async ({ page }) => {
    await page.click("#forgotPassword");
    await expect(page.locator("#reset")).toBeVisible();
});
```

 Copy code

## 3 When to Use Each Hook

Hook	When to use
<code>beforeAll</code>	Setup test data or global login
<code>afterAll</code>	Cleanup data, close browser
<code>beforeEach</code>	Navigate to page before each test

Hook	When to use
afterEach	Clear session, reset state after test

## 4 Test Organization

### Using describe block (GROUPING TESTS)

ts

```
test.describe("Login Tests", () => {
  test("Valid Login", async ({ page }) => { /*...*/ });
  test("Invalid Login", async ({ page }) => { /*...*/ });
});
```

 Copy code

⭐ Interview question:

Why use describe blocks?

→ To group related tests and improve readability.

## 5 Reusable Code with Hooks

- Avoid repeating login in each test
- Use beforeEach to handle common setup
- Use afterEach for cleanup like logout

## 6 Real Interview Scenario

❓ Multiple tests require login → what's best practice?

✓ Use beforeEach to login instead of repeating login steps in every test.

## 🔥 Common Interview Questions (Day 14)

1. What is a hook in Playwright?
2. Difference between beforeEach and beforeAll ?
3. When to use afterEach vs afterAll ?
4. How do you group tests?
5. Why are hooks important for framework design?

## Coding Practice (IMPORTANT)

### Q 1 Setup beforeEach for login

- Use page.goto and fill login fields
- 

### Q 2 Use afterEach to clear session / logout

---

### Q 3 Group login tests in a describe block

---

### Q 4 Explain: How would you avoid code duplication in 10+ tests?

---

## Day 14 Task

- ✓ Implement hooks in your tests
  - ✓ Practice grouping tests
  - ✓ Explain hooks confidently for interviews
- 

## Next step:

Type Day 15 and we'll cover Screenshots, Videos & Reporting in Playwright   



Day 15

Awesome! 

DAY 15 – Screenshots, Videos & Reporting in Playwright

(🔥 Very common in real-time projects + interview scenario questions)

---



## DAY 15 – Screenshots, Videos & Reporting

### Goal

- Learn how to capture screenshots and videos

- Generate test reports
  - Explain usage confidently in interviews
- 

## 1 Screenshots in Playwright

### Take screenshot of entire page

ts

```
await page.screenshot({ path: "fullPage.png", fullPage: true });
```

 Copy code

### Take screenshot of element

ts

```
await page.locator("#login").screenshot({ path: "loginButton.png" })
```

 Copy code



When would you take a screenshot in tests?

- To capture UI state on failures for debugging.
- 

## 2 Videos in Playwright

### Enable video recording for a test

ts

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

test("Video Test", async ({ page }) => {
  await page.goto("https://example.com");
}, {
  video: "on"
});
```

 Copy code

### Configure default video for all tests (playwright.config.ts)

ts

```
use: {
  video: "on-first-retry"
```

 Copy code

```
}
```

📌 Interview question:

**Why use video recording?**

➡ Helps debug failures, especially flaky tests.

---

## 3 Reporting in Playwright

**Generate HTML report**

bash

```
npx playwright test --reporter=html
```

 Copy code

**Open HTML report**

bash

```
npx playwright show-report
```

 Copy code

**Multiple reporters**

ts

```
reporter: [[ "list" ], [ "html", { open: "always" } ]]
```

 Copy code

📌 Interview question:

**How do you share test results with the team?**

➡ Use HTML or JUnit reports, optionally integrate with CI/CD.

---

## 4 Screenshots / Videos + Hooks (BEST PRACTICE)

ts

```
test.afterEach(async ({ page }, testInfo) => {
  if (testInfo.status !== testInfo.expectedStatus) {
    await page.screenshot({ path: `screenshots/${testInfo.title}.png` });
  }
});
```

 Copy code

📌 Saves screenshot only on test failure.

---

## 5 Real-Time Scenarios (INTERVIEW FAVORITE)

1.  ? Flaky UI test → solution?  
 Capture screenshot/video on failure for debugging.
  2.  ? How to share results in CI?  
 Generate HTML or JUnit reports, attach artifacts.
- 

## 🔥 Common Interview Questions (Day 15)

1. How do you take a screenshot in Playwright?
  2. How do you record video for tests?
  3. How do you generate reports?
  4. How do you capture failure screenshots?
  5. Difference between screenshots, video, and report?
- 

## 🧪 Coding Practice (IMPORTANT)

Q 1 Take screenshot of login page

---

Q 2 Record video of test execution

---

Q 3 Save screenshot only if test fails

---

Q 4 Generate HTML report

---

## ✓ Day 15 Task

- ✓ Practice screenshots & videos
  - ✓ Configure reports in `playwright.config.ts`
  - ✓ Explain reporting workflow confidently
- 

## 👉 Next step:

Type Day 16 and we'll cover Data-Driven Testing + Fixtures ✨ 🔥



Perfect! 👍

## DAY 16 – Data-Driven Testing & Fixtures in Playwright

(🔥 Critical for framework design and interview scenarios)

---



## DAY 16 – Data-Driven Testing & Fixtures



### Goal

- Learn how to run same test with multiple data sets
  - Understand fixtures for reusable setup
  - Explain clearly in interviews
- 

1

### What is Data-Driven Testing?

Data-driven testing means running the same test multiple times with different input values.

2

Interview answer:

Data-driven testing increases coverage and reduces code duplication by feeding multiple inputs to the same test.

---

2

### Simple Data-Driven Example

ts

```
import { test, expect } from "@playwright/test";

const users = [
  { username: "admin", password: "admin123" },
  { username: "user", password: "user123" },
];
```

```
for (const user of users) {  
  test(`Login test for ${user.username}`, async ({ page }) => {  
    await page.goto("https://example.com/login");  
    await page.fill("#username", user.username);  
    await page.fill("#password", user.password);  
    await page.click("#login");  
    await expect(page).toHaveURL(/dashboard/);  
  });  
}
```

 Copy code



## Why use data-driven testing?

- To test multiple scenarios without writing separate tests.
- 

## 3 Using `test.describe()` + `test.each()` (BETTER APPROACH)

ts

```
const loginData = [  
  ["admin", "admin123"],  
  ["user", "user123"]  
];  
  
test.describe("Login Tests", () => {  
  test.each(loginData)("Login with %s", async (username, password, { page }) => {  
    await page.goto("https://example.com/login");  
    await page.fill("#username", username);  
    await page.fill("#password", password);  
    await page.click("#login");  
    await expect(page).toHaveURL(/dashboard/);  
  });  
});
```

 Copy code



## 4 Fixtures in Playwright (VERY IMPORTANT ⭐)

Fixtures are reusable setup objects for tests.

### Built-in fixture example

ts

```
test("Login Test", async ({ page }) => {
  await page.goto("https://example.com/login");
});
```

 Copy code

Here, `page` is a **fixture** provided by Playwright.

---

## Custom Fixture Example

ts

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

type MyFixtures = {
  loginPage: any;
};

export const test = base.extend<MyFixtures>({
  loginPage: async ({ page }, use) => {
    await page.goto("https://example.com/login");
    await use(page);
  },
});

test("Login using fixture", async ({ loginPage }) => {
  await loginPage.fill("#username", "admin");
  await loginPage.fill("#password", "admin123");
  await loginPage.click("#login");
});
```

 Copy code

⭐ Interview question:

**Why use fixtures?**

➡ To avoid repeating setup code, maintain clean and reusable tests.

---

## 5 Real Interview Scenarios

1. **?** Multiple login credentials → solution?
  - Use data-driven testing.
2. **?** Repeating login in 20 tests → solution?
  - Use custom fixture for login setup.

## Common Interview Questions (Day 16)

1. What is data-driven testing?
  2. How do you feed multiple data sets to a test?
  3. What is a fixture?
  4. Difference between built-in and custom fixture?
  5. Why are fixtures important for framework design?
- 

## Coding Practice (IMPORTANT)

Q 1 Login test for 3 users using loop

---

Q 2 Use `test.each()` for login scenarios

---

Q 3 Create a custom fixture for login

---

Q 4 Explain how fixtures reduce code duplication

---

## Day 16 Task

- ✓ Practice multiple login scenarios with different data
  - ✓ Implement and reuse fixtures
  - ✓ Be ready to explain **why and when to use them**
- 

## Next step:

Type Day 17 and we'll cover Playwright Locators Advanced + Shadow DOM & Nested Elements  



Awesome! 👍

## DAY 17 – Advanced Locators: Shadow DOM & Nested Elements

(🔥 This is often asked in mid-level Playwright interviews)

---

# DAY 17 – Advanced Locators: Shadow DOM & Nested Elements

## 🎯 Goal

- Learn how to handle **nested elements**
  - Learn how to work with **Shadow DOM**
  - Be able to answer **interview scenarios confidently**
- 

## 1 Nested Elements (Parent → Child)

Sometimes you need to locate **child elements inside a parent**.

**Example:**

ts

```
const card = page.locator(".card"); // parent
await card.locator("button").click(); // child button inside card
```

 Copy code

### 📌 Interview question:

**How do you click a button inside a specific card?**

→ Use locator chaining.

---

## Another Example – Complex nesting

ts

```
await page.locator(".form").locator("input[name='email']").fill("test@example.com")
```

 Copy code

## 2 Shadow DOM Elements (VERY IMPORTANT ⭐)

Some modern web apps use **Shadow DOM** which prevents direct access using normal locators.

## Playwright solution

Use `pierce` or `>>` selector:

ts

```
await page.locator("css=custom-element >> shadow=input").fill("Hello")  
Copy code
```

OR using `shadowRoot`:

ts

```
const shadowInput = page.locator("custom-element").locator("input");  
await shadowInput.fill("Hello")  
Copy code
```



**How do you handle elements inside Shadow DOM?**

- Use `locator("shadow-selector")` or `>>` syntax.
- 

## 3 Advanced Text Locators

### Partial text

ts

```
await page.getByText("Submit", { exact: false }).click()  
Copy code
```

### Role + Name

ts

```
await page.getByRole("button", { name: "Submit" }).click()  
Copy code
```

- ★ Best practice: Role locators are stable for accessibility compliance.
- 

## 4 Handling Dynamic Nested Elements

ts

```
await page.locator(".card").filter({ hasText: "Active" }).locator("button").click()  
Copy code
```

- ★ Useful when multiple similar elements exist.

## 5 Real Interview Scenarios

1. ? Click a button inside the 3rd card → solution?  
✓ `page.locator(".card").nth(2).locator("button").click();`
  2. ? Fill an input inside Shadow DOM → solution?  
✓ Use `>> shadow=` selector or nested locator.
- 

## 🔥 Common Interview Questions (Day 17)

1. How do you click child elements inside a parent?
  2. What is Shadow DOM and why is it tricky?
  3. How do you access Shadow DOM elements in Playwright?
  4. Difference between normal and role-based locators?
  5. How to handle dynamic nested elements?
- 

## 🧪 Coding Practice (IMPORTANT)

Q 1 Click the button inside the second `.card` element

---

Q 2 Fill input inside Shadow DOM element `custom-element`

---

Q 3 Filter `.item` that has text "Active" and click child button

---

Q 4 Use role locator to click "Submit" button

---

## ✓ Day 17 Task

- ✓ Master nested locators and Shadow DOM
  - ✓ Practice `filter()`, `nth()` and locator chaining
  - ✓ Explain Shadow DOM solution clearly
- 

## 👉 Next step:

Type Day 18 and we'll cover Handling Tables & Dynamic Web Tables in Playwright 📊🔥



Awesome! 🤘

## DAY 18 – Handling Tables & Dynamic Web Tables in Playwright

(🔥 Very common in automation projects and interviews)

---



## DAY 18 – Tables & Dynamic Web Tables

### 🎯 Goal

- Learn how to **locate tables**
  - Extract **row & column data**
  - Handle **dynamic tables**
  - Answer related interview questions confidently
- 

### 1 Locating Tables

Tables in HTML usually use `<table>` , `<tr>` , `<td>` .

ts

```
const table = page.locator("table#userTable");
```

Copy code

📌 Interview question:

**How do you locate a table in Playwright?**

➡️ Use `page.locator("table#id")` or unique CSS/XPath.

---

### 2 Counting Rows & Columns

ts

```
const rows = await table.locator("tr").count();
const cols = await table.locator("tr").first().locator("td").count();
```

Copy code

📌 Useful for iterating over data.

## 3 Accessing Cell Data

ts

```
const cell = table.locator("tr").nth(1).locator("td").nth(2);
console.log(await cell.textContent());
```

 Copy code



Print 2nd row, 3rd column data

- Use `nth()` on `tr` and `td`.

## 4 Iterating Over Table Rows

ts

```
const rowCount = await table.locator("tr").count();

for (let i = 0; i < rowCount; i++) {
  const row = table.locator("tr").nth(i);
  const username = await row.locator("td").nth(0).textContent();
  const role = await row.locator("td").nth(1).textContent();
  console.log(username, role);
}
```

 Copy code



How do you extract all table data dynamically?

## 5 Dynamic Tables / Filtering

- Sometimes rows load dynamically via JS/Ajax
- Use assertions or wait for rows:

ts

```
await expect(table.locator("tr")).toHaveLength(10); // waits until 10 rows appear
```

 Copy code

- Click based on text:

ts

 Copy code

```
await table.locator("tr").filter({ hasText: "Admin" }).locator("button").click();
```

## 6 Real Interview Scenarios

1. ? Click “Edit” button for user “John” → solution?

ts

```
await table.locator("tr").filter({ hasText: "John" }).locator("button").click();
```

 Copy code

2. ? Get data from the last row → solution?

ts

```
const lastRow = table.locator("tr").last();
```

 Copy code

## 🔥 Common Interview Questions (Day 18)

1. How do you count rows and columns in a table?
2. How do you access a specific cell?
3. How do you iterate through all rows dynamically?
4. How do you click a button in a specific row?
5. How do you handle dynamic tables loaded via JS?

## 💡 Coding Practice (IMPORTANT)

- Q 1 Print username & role for all rows in a table

- Q 2 Click the “Delete” button for user “Alice”

- Q 3 Print data from the last row

- Q 4 Assert the table has at least 5 rows

ts

```
table.locator("tr")
```

 Copy code

---

## Day 18 Task

- ✓ Practice row & column iteration
  - ✓ Filter rows based on text
  - ✓ Handle dynamic table data
- 

### Next step:

Type Day 19 and we'll cover Mouse & Keyboard Advanced Actions (Drag & Drop, Scroll, Hover, Multi-Select)  



Day 19

Perfect! 

**DAY 19 – Advanced Mouse & Keyboard Actions (Drag & Drop, Scroll, Hover, Multi-Select)**

( Often asked in interviews and required in complex UI automation)

---



## DAY 19 – Advanced Mouse & Keyboard Actions

### Goal

- Master mouse actions beyond click & hover
  - Use keyboard actions
  - Handle drag-drop, scrolling, multi-select
  - Be confident in interview scenarios
- 

### 1 Hover Over Elements

ts

 Copy code

 Copy code

```
await page.locator("#menu").hover();
```

 Interview question:

**When would you use hover?**

- To reveal dropdown menus, tooltips, or dynamic elements.
- 

## 2 Drag & Drop

ts

```
await page.locator("#source").dragTo(page.locator("#target"));
```

 Copy code

**Alternate (manual mouse actions)**

ts

```
const source = page.locator("#source");
const target = page.locator("#target");
```

```
const sourceBox = await source.boundingBox();
const targetBox = await target.boundingBox();
```

```
await page.mouse.move(sourceBox.x + sourceBox.width / 2, sourceBox.y + sourceBox.height / 2);
await page.mouse.down();
await page.mouse.move(targetBox.x + targetBox.width / 2, targetBox.y + targetBox.height / 2);
await page.mouse.up();
```

 Copy code

 Interview tip: Use `dragTo()` for simplicity.

---

## 3 Scrolling the Page

**Scroll to element**

ts

```
await page.locator("#footer").scrollIntoViewIfNeeded();
```

 Copy code

**Scroll by pixels**

ts

```
await page.mouse.wheel(0, 500);
```

 Copy code

⭐ Interview question:

**How do you scroll to element if hidden?**

→ Use `scrollIntoViewIfNeeded()`.

---

## 4 Keyboard Actions

**Press single key**

ts

```
await page.keyboard.press("Enter");
```

 Copy code

**Type text with delay**

ts

```
await page.keyboard.type("Playwright", { delay: 100 });
```

 Copy code

**Shortcut keys (Ctrl, Shift)**

ts

```
await page.keyboard.down("Control");
await page.keyboard.press("A");
await page.keyboard.up("Control");
```

 Copy code

⭐ Useful for select-all, copy-paste, multi-select.

---

## 5 Multi-Select Dropdown / List

ts

```
await page.locator("select#fruits").selectOption([
  { value: "apple" },
  { value: "orange" },
]);
```

 Copy code

**Multi-Select Custom List (Checkboxes)**

ts

```
await page.locator(".checkbox").filter({ hasText: "Option1" }).click();
await page.locator(".checkbox").filter({ hasText: "Option3" }).click();
```

 Copy code

---

## 6 Real Interview Scenarios

1.  Hover menu and click submenu → solution?

ts

```
await page.locator("#menu").hover();
await page.locator("#submenu").click();
```

 Copy code

2.  Drag item to target → solution?

ts

```
await page.locator("#item").dragTo(page.locator("#target"));
```

 Copy code

3.  Scroll to hidden element → solution?

ts

```
await page.locator("#hidden").scrollIntoViewIfNeeded();
```

 Copy code

---

## 🔥 Common Interview Questions (Day 19)

1. How do you hover over an element?
  2. How do you perform drag & drop?
  3. How do you scroll to a hidden element?
  4. How do you press keyboard shortcuts?
  5. How do you select multiple items from a list?
- 

## 💡 Coding Practice (IMPORTANT)

- Q 1 Hover over menu → click submenu
- 

- Q 2 Drag element #box1 to #box2
-

## Q 3 Scroll to footer element

---

## Q 4 Select multiple options from multi-select dropdown

---

### Day 19 Task

- Practice hover, drag-drop, scroll
  - Practice keyboard shortcuts & multi-select
  - Be ready to explain advanced actions in interviews
- 

### Next step:

Type Day 20 and we'll cover Handling Alerts, Prompts & Confirm Boxes with Assertions



Day 20

Awesome! 

DAY 20 – Handling Alerts, Prompts & Confirm Boxes with Assertions

(🔥 Very common interview topic and real-time scenario handling)

---

## DAY 20 – Alerts, Prompts & Confirm Boxes

### Goal

- Handle JavaScript dialogs: `alert`, `confirm`, `prompt`
  - Combine handling with assertions
  - Be confident in explaining scenarios
- 

## 1 Types of Dialogs

Dialog Type	Description
alert	Simple message with OK

Dialog Type	Description
confirm	Message with OK / Cancel
prompt	Input box to accept user input

📌 Interview question:

**What are different JS dialogs and how do you handle them in Playwright?**

---

## 2 Handling Alert

ts

```
page.on("dialog", async (dialog) => {
  console.log(dialog.message()); // print message
  await dialog.accept();        // click OK
});

await page.click("#alertButton");
```

 Copy code

Always listen for dialog before triggering it.

---

## 3 Handling Confirm Box

ts

```
page.on("dialog", async (dialog) => {
  console.log(dialog.message());
  await dialog.dismiss(); // click Cancel
};

await page.click("#confirmButton");
```

 Copy code

📌 Use accept() for OK, dismiss() for Cancel.

---

## 4 Handling Prompt Box

ts

```
page.on("dialog", async (dialog) => {
  console.log(dialog.message());
```

```
    await dialog.accept("Playwright"); // enter text and click OK
});
```

```
await page.click("#promptButton");
```

 Copy code



Always combine with assertion to verify the result.

---

## 5 Assertion After Dialog

ts

```
await page.on("dialog", async (dialog) => {
  expect(dialog.message()).toContain("Are you sure");
  await dialog.accept();
});
```

```
await page.click("#confirmButton");
await expect(page.locator("#result")).toHaveText("Confirmed");
```

 Copy code



## 6 Real Interview Scenarios

1.  ? Accept alert and verify message on page → solution?  
 Use `page.on("dialog") + expect` on page locator.
  2.  ? Cancel confirm box and verify result → solution?  
 `dialog.dismiss() + assertion.`
  3.  ? Enter text in prompt box → solution?  
 `dialog.accept("text") + verify page changes.`
- 

## 🔥 Common Interview Questions (Day 20)

1. How do you handle `alert` in Playwright?
  2. Difference between `alert`, `confirm`, and `prompt`?
  3. How do you dismiss a confirm box?
  4. How do you enter text in a prompt box?
  5. How do you verify the result after handling dialog?
-

## Coding Practice (IMPORTANT)

Q 1 Accept alert and print message

---

Q 2 Dismiss confirm box and assert result

---

Q 3 Enter text in prompt box and verify

---

Q 4 Combine dialog handling with assertion for verification

---

### Day 20 Task

- ✓ Practice handling alert, confirm & prompt
  - ✓ Always combine with assertion
  - ✓ Be ready to explain steps clearly in interview
- 

### Next step:

Type Day 21 and we'll cover Playwright CLI, Config & Environment Variables  



Day 21

Awesome! 

### DAY 21 – Playwright CLI, Config & Environment Variables

( Very important for real-time projects and interviews)

---

## DAY 21 – Playwright CLI, Config & Environment Variables

### Goal

- Learn Playwright CLI commands

- Configure `playwright.config.ts`
  - Use **environment variables** for flexible tests
- 

## 1 Playwright CLI Basics

### Run all tests

bash

```
npx playwright test
```

 Copy code

### Run a single test file

bash

```
npx playwright test tests/login.spec.ts
```

 Copy code

### Run a specific test in a file

bash

```
npx playwright test tests/login.spec.ts -g "Valid Login"
```

 Copy code

### Run in headed mode

bash

```
npx playwright test --headed
```

 Copy code

### Run with browser selection

bash

```
npx playwright test --project=firefox
```

 Copy code

 Interview question:

**How do you run a single test or browser using CLI?**

 Use `-g` for test name and `--project` for browser.

---

## 2 Playwright Config (`playwright.config.ts`)

### Basic config example

ts

```
import { defineConfig, devices } from "@playwright/test";

export default defineConfig({
  testDir: "./tests",
  timeout: 30000,
  retries: 1,
  reporter: [["list"], ["html", { open: "always" }]],
  use: {
    headless: true,
    viewport: { width: 1280, height: 720 },
    ignoreHTTPSErrors: true,
    video: "on-first-retry",
  },
  projects: [
    { name: "chromium", use: { ...devices["Desktop Chrome"] } },
    { name: "firefox", use: { ...devices["Desktop Firefox"] } },
    { name: "webkit", use: { ...devices["Desktop Safari"] } },
  ],
});

```

 Copy code

👉 Interview question:

Why use playwright.config.ts?

➡ Centralized configuration for all tests, browsers, retries, and reporting.

---

### 3 Environment Variables

- Useful for **base URLs, credentials, or API keys**
- Do not hardcode sensitive info in tests

Example

bash

```
# Linux / Mac
export BASE_URL="https://example.com"
export USERNAME="admin"
```

 Copy code

ts

```
// playwright.config.ts
use: {
```

```
        baseURL: process.env.BASE_URL,  
    }  
  
ts
```

 Copy code

```
// In test  
await page.goto(process.env.BASE_URL + "/login");
```

 Copy code

📌 Interview question:

**How do you handle different environments (dev, staging, prod)?**

➡ Use environment variables + config file.

---

## 4 Multiple Environment Configs

ts

```
const devConfig = { use: { baseURL: "https://dev.example.com" } };  
const stagingConfig = { use: { baseURL: "https://staging.example.com" } };  
  
export default defineConfig({  
    projects: [  
        { name: "dev", ...devConfig },  
        { name: "staging", ...stagingConfig },  
    ],  
});
```

 Copy code

---

## 5 Real Interview Scenarios

1. ? Run tests on Firefox in staging → solution?  
✓ Use CLI with --project=staging or env variable.
  2. ? Base URL changes between environments → solution?  
✓ Use process.env.BASE\_URL in config or test.
  3. ? Need HTML report after CI → solution?  
✓ Configure reporter: [["html", { open: "never" }]] .
- 

## 🔥 Common Interview Questions (Day 21)

1. How do you run a specific test using CLI?
2. What is playwright.config.ts used for?

- 
3. How do you handle multiple environments?
  4. How to pass credentials without hardcoding?
  5. How to generate HTML report from CLI?
- 

## Coding Practice (IMPORTANT)

Q **1** Run a single test using CLI

---

Q **2** Set `headless=false` in config

---

Q **3** Use env variable for base URL in test

---

Q **4** Add retry for flaky tests in config

---

## Day 21 Task

- ✓ Master Playwright CLI commands
  - ✓ Configure browsers, retries, and reporters
  - ✓ Use environment variables for flexible tests
- 

## Next step:

Type Day 22 and we'll cover CI/CD Integration with Playwright (GitHub Actions, Jenkins)



Day 22

Awesome! 

DAY 22 – CI/CD Integration with Playwright (GitHub Actions, Jenkins)

( Very common in real-time projects and often asked in interviews)

---



# DAY 22 – CI/CD Integration with Playwright

## 🎯 Goal

- Learn how to run Playwright tests in CI/CD pipelines
  - Integrate with GitHub Actions and Jenkins
  - Understand best practices for reports & artifacts
- 

## 1 Why CI/CD Integration?

- Ensures automated test execution on every push/PR
- Detects breakages early
- Generates reports & artifacts for debugging

### ✍ Interview answer:

CI/CD integration helps maintain quality and ensures automated tests run consistently across environments.

---

## 2 GitHub Actions Integration

Example Workflow: `.github/workflows/playwright.yml`

```
yaml

name: Playwright Tests

on:
  push:
    branches: [ main ]
  pull_request:

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 18
```

```
- name: Install dependencies
  run: npm install

- name: Install Playwright Browsers
  run: npx playwright install --with-deps

- name: Run Tests
  run: npx playwright test --reporter=html

- name: Upload Report
  uses: actions/upload-artifact@v3
  with:
    name: playwright-report
    path: playwright-report
```

 Copy code



### How do you run Playwright tests in GitHub Actions?

→ Use `npx playwright test` in a workflow and optionally upload HTML report as artifact.

---

## 3 Jenkins Integration

### Steps to run Playwright tests:

1. Create Jenkins job
2. Pull code from Git repository
3. Install Node.js & dependencies (`npm install`)
4. Install Playwright browsers (`npx playwright install`)
5. Run tests (`npx playwright test`)
6. Publish **HTML report** via Jenkins plugin or archive artifacts



Always include browser installation and artifacts for debugging.

---

## 4 Generating Reports & Artifacts in CI/CD

- **HTML Report:**

bash

 Copy code

```
npx playwright test --reporter=html
```

- **Videos / Screenshots:** Configure in `playwright.config.ts`:

ts

```
use: {
  video: "on-first-retry",
  screenshot: "only-on-failure"
}
```

 Copy code

- **Upload artifacts** for debugging failed tests
- 

## 5 Real Interview Scenarios

1.  Tests failing in CI but passing locally → solution?  
 Check **headless mode, environment variables, browser installation, and network issues.**
  2.  Need to run tests on multiple browsers → solution?  
 Use **projects in `playwright.config.ts`.**
  3.  Generate report for team → solution?  
 Use **HTML report + upload artifacts.**
- 

## 🔥 Common Interview Questions (Day 22)

1. How do you integrate Playwright tests with GitHub Actions?
  2. How do you run tests on multiple browsers in CI?
  3. How do you capture screenshots/videos in CI?
  4. How do you debug failing CI tests?
  5. How do you upload reports/artifacts in CI/CD pipelines?
- 

## 💡 Coding / Configuration Practice (IMPORTANT)

### Q 1 Create GitHub Actions workflow to run Playwright tests

---

### Q 2 Configure HTML report & upload as artifact

---

## Q 3 Configure Playwright for multiple browsers in CI

---

## Q 4 Ensure videos/screenshots captured for failed tests

---

### Day 22 Task

- ✓ Run Playwright tests automatically in CI/CD
  - ✓ Capture reports, videos, and screenshots
  - ✓ Be ready to explain CI/CD workflow in interviews
- 

### Next step:

Type Day 23 and we'll cover API Testing with Playwright (REST, GraphQL, Assertions) 🔥



Day 23

Awesome! 

DAY 23 – API Testing with Playwright (REST, GraphQL, Assertions)

(🔥 Very important for interviews and modern automation frameworks)

---

## DAY 23 – API Testing with Playwright

### Goal

- Learn API testing using Playwright
  - Test REST & GraphQL APIs
  - Make assertions on responses
  - Be confident in explaining scenarios in interviews
- 

### 1 Why Use Playwright for API Testing?

- Playwright provides request context to make HTTP requests
- Can combine UI + API testing in one framework

- Useful for **data setup, validation, and end-to-end tests**

### 📌 Interview answer:

Playwright API testing helps validate backend endpoints and ensures end-to-end flow consistency.

---

## 2 Creating Request Context

ts

```
import { test, expect, request } from "@playwright/test";

test("API Test Example", async ({ request }) => {
  const response = await request.get("https://jsonplaceholder.typicode.com/posts/");
  expect(response.status()).toBe(200);

  const data = await response.json();
  expect(data.userId).toBe(1);
  expect(data).toHaveProperty("title");
});


```

 Copy code

## 3 POST Request

ts

```
const response = await request.post("https://jsonplaceholder.typicode.com/posts",
  data: {
    title: "Playwright Test",
    body: "Testing API",
    userId: 1,
  },
);

expect(response.status()).toBe(201);
const responseData = await response.json();
expect(responseData.title).toBe("Playwright Test");
```

 Copy code

## 4 PUT / PATCH Request

ts

```
const response = await request.put("https://jsonplaceholder.typicode.com/posts/1"
  data: { title: "Updated Title" },
);
expect(response.status()).toBe(200);
```

 Copy code

## 5 DELETE Request

ts

```
const response = await request.delete("https://jsonplaceholder.typicode.com/posts/1");
expect(response.status()).toBe(200);
```

 Copy code

## 6 GraphQL Request

ts

```
const query = {
  query: `query GetUser($id: ID!) {
    user(id: $id) {
      id
      name
      email
    }
  }`,
  variables: { id: 1 },
};

const response = await request.post("https://your-graphql-api.com/graphql", { data: query });
expect(response.status()).toBe(200);
```

```
const result = await response.json();
expect(result.data.user.name).toBeDefined();
```

 Copy code

## 7 Assertions on Response

- **Status Code:** `expect(response.status()).toBe(200)`

- **Body Data:** expect(data.property).toBe(value)
- **Headers:** expect(response.headers()["content-type"]).toContain("json")

👉 Interview tip: Always validate **status + data + headers**.

---

## 8 Real Interview Scenarios

1. **?** Verify GET API returns correct user → solution?  
✓ request.get() + expect() on status & JSON data
  2. **?** Create a user via POST → solution?  
✓ request.post() + check status() === 201 + JSON properties
  3. **?** Validate GraphQL query → solution?  
✓ Send POST request with query + validate response data
- 

## 🔥 Common Interview Questions (Day 23)

1. How do you test APIs with Playwright?
  2. Difference between REST and GraphQL in Playwright testing?
  3. How do you assert status code & response body?
  4. Can Playwright combine UI + API tests?
  5. How do you handle dynamic data returned by APIs?
- 

## 💡 Coding Practice (IMPORTANT)

Q 1 GET request → verify status 200 & userId

---

Q 2 POST request → create a new resource & assert title

---

Q 3 PUT request → update data & verify

---

Q 4 GraphQL query → fetch user data & assert name/email

---

✓ Day 23 Task

- ✓ Practice GET, POST, PUT, DELETE APIs
  - ✓ Practice GraphQL queries
  - ✓ Make proper assertions on status, headers, and body
- 

### 👉 Next step:

Type Day 24 and we'll cover **Handling Authentication (Basic, OAuth, Token)** in Playwright



Day 24

Awesome! 👍

## DAY 24 – Handling Authentication (Basic, OAuth, Token) in Playwright

(🔥 Crucial for real-world automation and interviews)

---

## 📅 DAY 24 – Authentication in Playwright

### 🎯 Goal

- Learn how to handle different types of authentication
  - Combine with UI/API tests
  - Be confident explaining authentication scenarios
- 

### 1 Basic Authentication (Browser Login Prompt)

ts

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

test("Basic Auth Test", async ({ page }) => {
  await page.goto("https://username:password@example.com");
  await page.waitForLoadState("networkidle");
});
```

Copy code

Alternate via `httpCredentials` in config:

ts

```
use: {  
  httpCredentials: { username: "admin", password: "admin123" },  
}
```

 Copy code



How do you handle basic authentication in Playwright?

- Use URL credentials or `httpCredentials` in config.
- 

## 2 Token-Based Authentication (Bearer Token)

### API Request

ts

```
const response = await request.get("https://api.example.com/data", {  
  headers: { Authorization: `Bearer ${process.env.TOKEN}` },  
});  
expect(response.status()).toBe(200);
```

 Copy code

### UI Authentication via Local Storage

ts

```
await page.addInitScript(value => {  
  localStorage.setItem("token", value);  
}, process.env.TOKEN);  
  
await page.goto("https://example.com/dashboard");
```

 Copy code



How do you handle token-based authentication in UI tests?

- Set token in `localStorage/sessionStorage` before navigating to the page.
- 

## 3 OAuth / SSO Authentication

- Usually involves redirect to auth provider
- Two common approaches:

### 1 Bypass login with token

ts

```
await page.goto("https://example.com");
await page.evaluate(token => {
  localStorage.setItem("authToken", token);
}, process.env.OAUTH_TOKEN);
```

 Copy code

## 2 Automate login flow (if necessary)

ts

```
await page.goto("https://auth.example.com/login");
await page.fill("#username", process.env.USERNAME);
await page.fill("#password", process.env.PASSWORD);
await page.click("#login");
```

 Copy code

📌 Interview question:

Which approach is better for CI?

- Using token/localStorage is faster and avoids flaky external login.
- 

## 4 Session Storage / Login State (Optimizing Login)

ts

```
// Save Login state
await page.context().storageState({ path: "auth.json" });

// Reuse Login state
const context = await browser.newContext({ storageState: "auth.json" });
const page = await context.newPage();
await page.goto("https://example.com/dashboard");
```

 Copy code

📌 Helps avoid repeated login in multiple tests.

---

## 5 Real Interview Scenarios

1. ? How to skip login for CI tests?
  - ✓ Use `storageState` or `token` in `localStorage`
2. ? API requires Bearer token → solution?
  - ✓ Pass token in `Authorization header`
3. ? How to handle OAuth login flow reliably?
  - ✓ Prefer token/localStorage over automating third-party login.

---

## 🔥 Common Interview Questions (Day 24)

1. How do you handle basic authentication?
  2. How do you set Bearer token for API or UI tests?
  3. How do you reuse login session in Playwright?
  4. How do you handle OAuth or SSO in tests?
  5. Why is storageState useful for login?
- 

## 🧪 Coding Practice (IMPORTANT)

Q 1 Test basic auth using `httpCredentials`

---

Q 2 Access a page using Bearer token in `localStorage`

---

Q 3 Save login state in `auth.json` and reuse

---

Q 4 Automate OAuth login (if no token available)

---

## ✓ Day 24 Task

- ✓ Handle basic, token-based, and OAuth authentication
  - ✓ Use `storageState` to optimize tests
  - ✓ Be ready to explain **why each method is preferred in CI/CD**
- 

## 👉 Next step:

Type Day 25 and we'll cover Playwright Test Retry, Flaky Test Handling & Debugging Techniques 🐛 🔥



Awesome! 👍

## DAY 25 – Playwright Test Retry, Flaky Test Handling & Debugging Techniques

(🔥 Very important for real-world projects and interviews)

---



## DAY 25 – Retry, Flaky Tests & Debugging

### 🎯 Goal

- Learn how to **handle flaky tests**
  - Configure **retry logic**
  - Use **debugging techniques** effectively
  - Be confident in explaining strategies in interviews
- 

### 1 Handling Flaky Tests with Retry

- Flaky tests may fail due to **network, timing, or UI instability**
- Playwright provides `retries` in config or CLI

**Set retries in config ( `playwright.config.ts` )**

ts

```
export default defineConfig({  
  retries: 2, // Retry failed tests 2 times  
});
```

Copy code

**Override retries via CLI**

bash

```
npx playwright test --retries=3
```

Copy code

📌 Interview question:

**Why do you use retries?**

→ To reduce false negatives caused by flaky tests.

---

### 2 Capturing Flaky Test Artifacts

- Use screenshots, videos, and logs to debug flaky tests:

ts

```
use: {  
  screenshot: "only-on-failure",  
  video: "on-first-retry",  
  trace: "on-first-retry",  
}
```

 Copy code

- **Trace Viewer** is very useful:

bash

```
npx playwright show-trace trace.zip
```

 Copy code

 Helps identify **why a test failed** in CI or locally.

---

## 3 Debugging Techniques in Playwright

### 1 Debugging in Headed Mode

bash

```
npx playwright test --headed
```

 Copy code

### 2 Slow Motion

ts

```
use: {  
  launchOptions: { slowMo: 100 },  
}
```

 Copy code

## 3 Pausing Test Execution

ts

```
await page.pause(); // Opens Playwright Inspector
```

 Copy code

## 4 Playwright Inspector (Visual Debugger)

bash

```
npx playwright test --debug
```

 Copy code

 Interview tip:

Knowing debugging tools is as important as writing tests.

---

## 4 Flaky Test Best Practices

1. Use stable locators – avoid text that changes dynamically
  2. Explicit waits – await page.waitForSelector() or expect(locator).toBeVisible()
  3. Retry only unstable tests – avoid masking real bugs
  4. Capture screenshots/videos/traces for failed tests
- 

## 5 Real Interview Scenarios

1. ? Test fails sometimes on CI → solution?
    - ✓ Enable retries, capture screenshots/videos, use trace viewer
  2. ? How to debug slow or failing tests locally?
    - ✓ Use --headed, slowMo, or page.pause()
  3. ? How to avoid flaky locators?
    - ✓ Use role-based, stable, or unique selectors
- 

## 🔥 Common Interview Questions (Day 25)

1. How do you handle flaky tests in Playwright?
  2. How do you configure retries?
  3. How do you debug failed tests?
  4. What is trace viewer and when do you use it?
  5. How do you capture screenshots/videos for failures?
- 

## 💡 Coding / Practice

### Q 1 Configure retries for a flaky test

---

### Q 2 Capture screenshot/video only on failure

---

### Q 3 Use Playwright Inspector to debug a test

---

## Q 4 Implement explicit wait to fix flaky UI test

---

### Day 25 Task

- Configure retries for flaky tests
  - Capture screenshots, videos, and trace files
  - Use headed mode, slow motion, and `page.pause()` to debug
  - Be ready to explain **why retries are important** in CI/CD
- 

### Next step:

Type Day 26 and we'll cover **Advanced Assertions & Custom Matchers** in Playwright 



Day 26

Awesome! 

## DAY 26 – Advanced Assertions & Custom Matchers in Playwright

( Very important for interviews and writing robust tests)



## DAY 26 – Advanced Assertions & Custom Matchers

### Goal

- Learn **advanced assertions** using Playwright's `expect`
  - Create **custom matchers** for reusable validations
  - Handle **dynamic UI content** confidently in interviews
- 

### 1 Advanced Assertions

Playwright `expect` supports **status**, **visibility**, **text**, **CSS**, **attributes**, and more.

### 1 Visibility & State

ts

```
const button = page.locator("#submit");
await expect(button).toBeVisible(); // visible in DOM
await expect(button).toBeEnabled(); // enabled
await expect(button).toBeEditable(); // editable input
```

 Copy code

## 2 Text Content

ts

```
await expect(page.locator("h1")).toHaveText("Welcome");
await expect(page.locator("p")).toContainText("Hello World");
```

 Copy code

## 3 Attribute & CSS

ts

```
await expect(page.locator("#logo")).toHaveAttribute("src", "/images/logo.png");
await expect(page.locator("#button")).toHaveCSS("color", "rgb(255, 0 ^``");
```

 Copy code

## 4 Count & Length

ts

```
await expect(page.locator(".list-item")).toHaveLength(5);
```

 Copy code

## 2 Assertions for API Testing

ts

```
const response = await request.get("/api/users/1");
await expect(response.status()).toBe(200);
```

```
const data = await response.json();
await expect(data).toHaveProperty("id");
await expect(data.name).toBe("John Doe");
```

 Copy code

 Combines UI + API assertions in one framework.

## 3 Assertions for Dynamic Content

- Wait for dynamic elements:

ts

```
await expect(page.locator(".notification")).toHaveText(/Success/i, { timeout: 5000 })
```

 Copy code

- Assert element becomes visible after some action:

ts

```
await page.click("#loadButton");
```

```
await expect(page.locator("#dynamic")).toBeVisible({ timeout: 7000 })
```

 Copy code

## 4 Custom Matchers (Reusable Assertions)

Sometimes default `expect` isn't enough, you can **wrap assertions in functions**:

ts

```
export async function expectUserLoggedIn(page) {
  const profile = page.locator("#profile");
  await expect(profile).toBeVisible();
  await expect(profile.locator(".username")).toHaveText(/John/i);
}
```

 Copy code

Usage:

ts

```
await expectUserLoggedIn(page);
```

 Copy code



**Why use custom matchers?**

→ Reusable, cleaner code, consistent validations across tests.

## 5 Real Interview Scenarios

1.  Assert element becomes visible dynamically → solution?
  - Use `await expect(locator).toBeVisible({ timeout: 5000 })`
2.  Verify text partially matches → solution?
  - Use regex: `/partial text/i`
3.  Check API response has a property → solution?
  - `expect(data).toHaveProperty("id")`

4. Reuse complex assertion across multiple tests → solution?

Create custom matcher function

---

## Common Interview Questions (Day 26)

1. How do you assert element visibility, state, or text?
  2. How do you assert attributes, CSS, or count?
  3. How do you handle dynamic elements in assertions?
  4. What are custom matchers and why are they useful?
  5. How do you assert API responses with Playwright?
- 

## Coding / Practice

Q **1** Assert an input is visible, enabled, and editable

---

Q **2** Assert element has specific text partially matching a pattern

---

Q **3** Assert API response has status 200 and required property

---

Q **4** Create a reusable custom matcher for user login validation

---

## Day 26 Task

- Master advanced assertions for UI & API
  - Handle dynamic elements with timeout
  - Build reusable custom matcher functions
  - Be ready to explain **why custom matchers improve test maintenance**
- 

## Next step:

Type Day 27 and we'll cover File Upload, Download & Handling PDFs / Images in Playwright



Awesome! 👍

## DAY 27 – File Upload, Download & Handling PDFs / Images in Playwright

(🔥 Very practical and commonly asked in automation interviews)

---



## DAY 27 – File Upload, Download & Media Handling

### 🎯 Goal

- Learn file upload & download in Playwright
  - Handle PDFs, images, and other media validations
  - Be confident explaining solutions in interviews
- 

### 1 File Upload

#### Standard File Input

ts

```
await page.setInputFiles("#fileInput", "path/to/file.txt");
await page.click("#uploadButton");
```

Copy code

#### Multiple Files

ts

```
await page.setInputFiles("#fileInput", [
  "path/to/file1.txt",
  "path/to/file2.txt"
]);
```

Copy code

📌 Interview question:

**How do you upload a file in Playwright?**

→ Use `page.setInputFiles()` with file path(s).

---

### 2 File Download

## Download a file and save locally

ts

```
const [download] = await Promise.all([
  page.waitForEvent("download"),
  page.click("#downloadButton"),
]);

// Save to custom path
await download.saveAs("downloads/report.pdf");
console.log("Downloaded file path:", download.path());
```

 Copy code



Validate a downloaded file exists and check its content or size.

---

## 3 Handling PDFs

- Playwright cannot read PDF directly; use **Node libraries** like `pdf-parse`:

ts

```
import fs from "fs";
import pdf from "pdf-parse";

const dataBuffer = fs.readFileSync("downloads/report.pdf");
const pdfData = await pdf(dataBuffer);
console.log(pdfData.text);
```

 Copy code



---

## 4 Handling Images

### Check image visibility

ts

```
await expect(page.locator("img#logo")).toBeVisible();
```

 Copy code

### Validate image source

ts

```
await expect(page.locator("img#logo")).toHaveAttribute("src", "/imag")
```

 Copy code

## Optional: Download image via API

ts

```
const response = await page.request.get("https://example.com/logo.png");
const buffer = await response.body();
fs.writeFileSync("downloads/logo.png", buffer);
```

 Copy code

## 5 Real Interview Scenarios

1.  Upload multiple files → solution?
  - Use `page.setInputFiles()` with array of file paths
2.  Download a PDF and verify content → solution?
  - `page.waitForEvent("download")` + Node library like `pdf-parse`
3.  Validate an image is displayed → solution?
  - `expect(locator).toBeVisible()` and check `src` attribute

## 🔥 Common Interview Questions (Day 27)

1. How do you upload a file in Playwright?
2. How do you download a file and save locally?
3. How do you validate PDF content in tests?
4. How do you check image visibility and attributes?
5. How do you handle multiple file uploads or downloads?

## 💡 Coding / Practice

Q 1 Upload `sample.txt` to file input and submit

Q 2 Download a PDF and save as `invoice.pdf`

Q 3 Parse downloaded PDF and assert it contains text "Invoice"

## Q 4 Validate an image is visible and its source URL

---

### Day 27 Task

- ✓ Master file upload & download
  - ✓ Handle PDFs and image validations
  - ✓ Be ready to explain how to verify file content in tests
- 

### Next step:

Type Day 28 and we'll cover Playwright Best Practices & Test Optimization for Interviews



Day 28

Awesome! 

## DAY 28 – Playwright Best Practices & Test Optimization for Interviews

( Crucial for writing maintainable tests and impressing interviewers)

---



## DAY 28 – Best Practices & Test Optimization

### Goal

- Learn best practices for writing robust Playwright tests
  - Optimize test performance & reliability
  - Be ready to explain your approach in interviews
- 

### 1 Test Structure Best Practices

#### 1. Organize tests in folders

- Example:

pgsql

```
tests/  
  login/  
  dashboard/  
  api/
```

 Copy code

## 2. One assertion per test principle (where possible)

- Makes tests **easy to debug & maintain**

## 3. Use `beforeAll`, `beforeEach`, `afterAll`, `afterEach` hooks

ts

```
test.beforeEach(async ({ page }) => {  
  await page.goto("/");  
});
```

 Copy code

## 4. Reuse login and common functions

ts

```
async function login(page, username, password) {  
  await page.fill("#username", username);  
  await page.fill("#password", password);  
  await page.click("#login");  
}
```

 Copy code

---

## 2 Locators & Selectors Best Practices

- Prefer `role`, `data-testid`, or unique IDs

ts

```
await page.locator('[data-testid="submit"]').click();
```

 Copy code

- Avoid **dynamic text or CSS selectors** that may break easily
- Use `locator.filter()` for complex elements

ts

```
await page.locator(".list-item").filter({ hasText: "Option1" }).click();
```

 Copy code

---

## 3 Test Optimization Tips

### 1. Run tests in parallel (default in Playwright)

ts

```
projects: [
  { name: "chromium", use: { ...devices["Desktop Chrome"] } },
]
```

 Copy code

## 2. Use storageState to avoid repeated login

ts

```
await context.storageState({ path: "auth.json" });
```

 Copy code

## 3. Explicit waits only when necessary

ts

```
await expect(page.locator("#dynamic")).toBeVisible({ timeout: 5000 })`
```

 Copy code

## 4. Use retries for flaky tests (configurable)

ts

```
retries: 2
```

 Copy code

## 5. Capture artifacts (screenshots/videos/traces) for CI debugging

---

## 4 Assertions & Readability

- Combine **custom matchers** for reusable validations
- Use **clear test names**:

ts

```
test("Login with valid credentials should navigate to dashboard", async () => {
```

 Copy code

- 
- 
- 
- Avoid hardcoded URLs; use **environment variables**

ts

```
await page.goto(process.env.BASE_URL + "/login");
```

 Copy code

---

## 5 Real Interview Scenarios

1.  How do you make tests maintainable?
  - Use hooks, reusable functions, stable locators, and clear test names

2.  How do you optimize test execution?
    - Run in parallel, reuse login state, avoid unnecessary waits
  3.  How do you debug flaky tests in CI?
    - Enable retries, capture screenshots/videos, use trace viewer
  4.  How do you write readable and reusable assertions?
    - Custom matchers and clear assertion logic
- 

## Common Interview Questions (Day 28)

1. What are your best practices for writing Playwright tests?
  2. How do you handle locators to avoid flaky tests?
  3. How do you optimize Playwright test execution?
  4. How do you reuse login or common steps across tests?
  5. How do you handle environment-specific configurations?
- 

## Coding / Practice

Q 1 **Organize tests with `beforeEach` and reusable login function**

---

Q 2 **Use `data-testid` locators for stable element interaction**

---

Q 3 **Run tests in parallel with different browsers**

---

Q 4 **Add retry and capture screenshot for CI failures**

---

## Day 28 Task

- Follow best practices for locators, hooks, and reusable functions
  - Optimize tests for speed and reliability
  - Be ready to explain how your test design reduces flakiness in interviews
- 

 Tip for Interviews:

Explaining how you structure tests, reuse code, and optimize execution often impresses interviewers even more than individual test scripts.

---

If you want, I can create a full 30-day Playwright roadmap summary with all important topics + interview questions, so you can revise in 1-2 days before interviews.

Do you want me to do that?



okay

Perfect! 🚀

Here's a complete 30-day Playwright roadmap summary for 3-year experience level (TypeScript/JavaScript). I've included all major topics, key points, and common interview questions for quick revision.

---

## 🎯 30-Day Playwright Roadmap Summary

---

### Day 1–5: Core Playwright Basics

#### Topics Covered:

- Playwright setup & installation (`npm init playwright@latest`)
- Writing first test, `page.goto()`, basic locators (`text=`, `id=`, `css=`)
- `click()`, `fill()`, `type()`, `selectOption()`, `check()`, `uncheck()`
- Assertions: `expect(locator).toHaveText()`, `toBeVisible()`, `toBeEnabled()`
- Test hooks: `beforeAll`, `beforeEach`, `afterAll`, `afterEach`

#### Common Interview Questions:

1. How do you launch browser and page in Playwright?
  2. How to locate elements? Difference between CSS, text, and role selectors?
  3. How do you assert element visibility, text, or state?
-

## Day 6–10: Advanced Locators & User Actions

### Topics Covered:

- Complex locators: `nth`, `filter`, `has`, `locator.locator()` chaining
- Mouse/keyboard actions: `hover()`, `dragAndDrop()`, `keyboard.press()`, `mouse.click()`
- Handling dropdowns & checkboxes
- Frames/Iframes: `frameLocator()`
- Multiple windows: `page.context().waitForEvent('page')`

### Common Interview Questions:

1. How to interact with elements inside iframe?
  2. How do you handle multiple browser tabs/windows?
  3. Difference between `locator.click()` and `page.click()` ?
- 

## Day 11–15: Dialogs, Alerts & Conditional Flows

### Topics Covered:

- Handling `alert`, `confirm`, `prompt` dialogs
- Conditional testing & dynamic element handling
- Using `if/else`, `try/catch`, `locator.count()` for conditional flows
- Waits: `waitForSelector()`, `waitForLoadState()`, `expect(locator).toBeVisible()`

### Common Interview Questions:

1. How do you handle alert, confirm, and prompt boxes?
  2. How to handle elements that appear dynamically?
  3. How to handle stale element or timing issues?
- 

## Day 16–20: CLI, Config, Environment & Artifacts

### Topics Covered:

- Playwright CLI commands: `npx playwright test`, `--project`, `-g`, `--headed`
- `playwright.config.ts` setup: `timeout`, `retries`, `reporter`, `projects`
- Environment variables & multiple environments (`process.env.BASE_URL`)
- File artifacts: screenshots, videos, traces for debugging
- Alert handling + assertions in CI/CD

### Common Interview Questions:

1. How do you run a single test or browser via CLI?

- 
2. What is `playwright.config.ts` used for?
  3. How to manage environment-specific URLs or credentials?
- 

## Day 21–23: CI/CD & API Testing

### Topics Covered:

- CI/CD Integration: GitHub Actions, Jenkins pipelines
- HTML reports, uploading artifacts
- API Testing with Playwright: GET, POST, PUT, DELETE, GraphQL
- Assertions on API responses: status, headers, body

### Common Interview Questions:

1. How do you integrate Playwright tests with GitHub Actions?
  2. How do you assert API responses?
  3. How to combine UI + API tests?
- 

## Day 24: Authentication

### Topics Covered:

- Basic auth (`httpCredentials`)
- Token-based auth (Bearer tokens in API and UI via `localStorage`)
- OAuth/SSO: bypass login using token or automate login
- Using `storageState` to reuse login session

### Common Interview Questions:

1. How do you handle basic or token-based authentication?
  2. How do you reuse login session across multiple tests?
  3. How to handle OAuth/SSO login reliably?
- 

## Day 25: Retry, Flaky Tests & Debugging

### Topics Covered:

- `retries` in config or CLI for flaky tests
- Debugging: `--headed`, `slowMo`, `page.pause()`, Playwright Inspector
- Capture screenshots, videos, traces for failed tests
- Best practices to avoid flaky tests (stable locators, explicit waits)

### Common Interview Questions:

- 
1. How to debug failing tests in Playwright?
  2. How to handle flaky tests?
  3. What are trace files and how are they useful?
- 

## Day 26: Advanced Assertions & Custom Matchers

### Topics Covered:

- Advanced assertions: visibility, text, CSS, count, attributes
- Dynamic element assertions with timeout
- Custom matchers for reusable validations
- API response assertions

### Common Interview Questions:

1. How to assert dynamic elements?
  2. How to create reusable custom matchers?
  3. Difference between partial text match and exact match?
- 

## Day 27: File Upload, Download & Media

### Topics Covered:

- File upload: `setInputFiles()`
- File download: `page.waitForEvent("download")`, `download.saveAs()`
- PDF content validation with `pdf-parse`
- Image validation: visibility, src attribute, download via API

### Common Interview Questions:

1. How do you upload/download files in Playwright?
  2. How to validate PDF content?
  3. How to check image visibility or source?
- 

## Day 28: Best Practices & Test Optimization

### Topics Covered:

- Test structure, hooks, reusable functions
- Stable locators: `data-testid`, roles
- Parallel execution, retries, storageState reuse
- Explicit waits only where necessary
- Clear test names & environment variables

## Common Interview Questions:

1. How do you make tests maintainable?
  2. How do you optimize test execution?
  3. How do you reduce flakiness in CI/CD?
- 

## Day 29–30: Interview Prep & Real Scenarios

### Topics Covered:

- Combine everything into **end-to-end test scenarios**
- UI + API + authentication + file handling + assertions
- Mocking APIs using `page.route()`
- Debugging CI/CD failed tests
- Revision of all common interview questions

### Key Tips for Interviews:

1. Explain your **test strategy first** (structure, hooks, reusable functions)
  2. Highlight **CI/CD, retries, and artifact usage**
  3. Showcase **API + UI integration and stable locators**
  4. Use **real-life examples** from projects for questions
- 

### Key Takeaways for 3-Year Playwright Interview

- Master locators, assertions, waits, hooks
  - Handle alerts, dialogs, files, authentication
  - Optimize test speed, reliability, CI/CD integration
  - Be comfortable debugging flaky tests
  - Understand API testing & custom matchers
  - Know Playwright CLI, config, environment variables
  - Explain best practices for maintainable test automation
- 

If you want, I can also create a **compact 1-page “Playwright Interview Cheat Sheet”** with **all commands, locators, assertions, and code snippets** ready to revise in a single day.

Do you want me to make that cheat sheet?



Perfect! 🚀

Here's your 1-page Playwright Interview Cheat Sheet – concise, ready for last-minute revision. I've included all important commands, locators, assertions, and code snippets.

---

## 🎯 Playwright Interview Cheat Sheet (1 Page)

---

### 1 Setup & CLI

bash

```
npm init playwright@latest      # Install Playwright
npx playwright test            # Run all tests
npx playwright test filename   # Run single test
npx playwright test --headed    # Run in headed mode
npx playwright test --project=chromium
npx playwright test --retries=2
npx playwright show-trace trace.zip
```

 Copy code

### 2 Browser & Page

ts

```
import { test, expect } from "@playwright/test";

test("basic test", async ({ page }) => {
  await page.goto("https://example.com");
});
```

 Copy code

- page.goto(url) → navigate
  - page.click(selector), page.fill(selector, value)
  - page.locator(selector) → stable locator
- 

### 3 Locators

ts

```
page.locator("text=Submit")
page.locator("#id")
page.locator(".class")
page.locator('[data-testid="input"]')
page.locator("role=button[name='Login']")
page.locator(".list-item").nth(0)
page.locator(".item").filter({ hasText: "Option1" })
```

 Copy code

---

## 4 Assertions

ts

```
await expect(locator).toBeVisible()
await expect(locator).toBeEnabled()
await expect(locator).toHaveText("Welcome")
await expect(locator).toContainText(/Hello/i)
await expect(locator).toHaveAttribute("src", "/logo.png")
await expect(locator).toHaveLength(5)
```

 Copy code

### API Assertions

ts

```
const response = await request.get("/api/users/1");
await expect(response.status()).toBe(200);
const data = await response.json();
await expect(data).toHaveProperty("id");
```

 Copy code

---

## 5 Waits & Flaky Tests

ts

```
await page.waitForSelector("#dynamic")
await expect(locator).toBeVisible({ timeout: 5000 })
test.beforeEach(async ({ page }) => {...})
test.afterEach(async ({ page }) => {...})
retries: 2 // config or CLI
```

 Copy code

## 6 Authentication

### Basic Auth

ts

```
use: { httpCredentials: { username: "admin", password: "admin123" } }  Copy code
```

### Token / LocalStorage

ts

```
await page.addInitScript(token => localStorage.setItem("token", token), process.e  
await page.goto("/dashboard")  Copy code
```

### StorageState

ts

```
await page.context().storageState({ path: "auth.json" })  
const context = await browser.newContext({ storageState: "auth.json" })  Copy code
```

## 7 File Upload / Download / PDFs

ts

```
// Upload  
await page.setInputFiles("#fileInput", "path/to/file.txt")
```

```
// Download  
const [download] = await Promise.all([ page.waitForEvent("download"), page.click()  
await download.saveAs("downloads/file.pdf") ])
```

```
// PDF validation (Node)  
import fs from "fs"; import pdf from "pdf-parse";  
const data = await pdf(fs.readFileSync("downloads/file.pdf"));  
console.log(data.text)
```

 Copy code

## 8 UI Actions & User Interactions

ts

```
await page.hover("#menu")
await page.dragAndDrop("#source", "#target")
await page.keyboard.press("Enter")
await page.mouse.click(x, y)
await page.check("#checkbox")
await page.uncheck("#checkbox")
await page.selectOption("#dropdown", "value")
```

 Copy code

---

## 9 Dialogs & Alerts

ts

```
page.on("dialog", dialog => dialog.accept())
await page.click("#trigger-alert")
```

 Copy code

---

## 10 API Testing

ts

```
const response = await request.post("/api", { data: { name: "John" } })
await expect(response.status()).toBe(201)
const result = await response.json()
await expect(result.name).toBe("John")
```

 Copy code

### GraphQL

ts

```
const query = { query: `query { user(id:1){id,name} }` }
const response = await request.post("/graphql", { data: query })
await expect(response.status()).toBe(200)
```

 Copy code

---

## 1 1 Debugging

bash

```
--headed      # Run in browser
--slowMo=100   # Slow motion
```

```
page.pause()      # Pause test  
npx playwright show-trace trace.zip
```

 Copy code

---

## 1 2 Best Practices

- Use **data-testid / role locators** for stability
  - Reuse **functions & login state** ( `storageState` )
  - Use **explicit waits** only when necessary
  - Capture **screenshots/videos/traces** in CI
  - Clear, descriptive **test names**
  - Run **tests in parallel** for speed
- 

## 1 3 Hooks & Structure

ts

```
test.beforeAll(async () => {...})  
test.beforeEach(async ({ page }) => {...})  
test.afterEach(async ({ page }) => {...})  
test.afterAll(async () => {...})
```

 Copy code

---

## 1 4 Common Interview Questions

1. How do you handle dynamic locators?
  2. How do you debug flaky tests?
  3. How do you upload/download files?
  4. How to handle authentication (Basic/Token/OAuth)?
  5. How do you combine API + UI testing?
  6. How do you optimize CI/CD execution?
  7. How do you reuse login state across tests?
  8. How do you assert PDF, image, or dynamic content?
- 

This cheat sheet covers **all essentials for a 3-year Playwright interview** and can be revised in **under 1 hour** before the interview.

---

If you want, I can also make a **super condensed version with just code snippets & commands only**, literally **one-page** with zero explanations, perfect for **quick last-minute glance**.

Do you want me to make that version too?



okay

Perfect! 😎

Here's your **Super Condensed Playwright Cheat Sheet** – literally all commands & code snippets on one page, zero explanations, ideal for quick last-minute glance.

---



## Playwright Quick Reference (1-Page)

ts

```
// ♦ Import
import { test, expect } from "@playwright/test";

// ♦ Browser & Page
await page.goto("https://example.com");
await page.click("#btn");
await page.fill("#input", "text");
await page.locator("text=Submit").click();

// ♦ Locators
page.locator("#id")
page.locator(".class")
page.locator('[data-testid="input"]')
page.locator("role=button[name='Login']")
page.locator(".item").nth(0)
page.locator(".item").filter({ hasText: "Option1" })

// ♦ Assertions
await expect(locator).toBeVisible()
await expect(locator).toBeEnabled()
await expect(locator).toBeEditable()
await expect(locator).toHaveText("Welcome")
```

```
await expect(locator).toContainText(/Hello/i)
await expect(locator).toHaveAttribute("src", "/logo.png")
await expect(locator).toHaveLength(5)

// ♦ Waits
await page.waitForSelector("#dynamic")
await expect(locator).toBeVisible({ timeout: 5000 })

// ♦ Hooks
test.beforeAll(async () => {})
test.beforeEach(async ({ page }) => {})
test.afterEach(async ({ page }) => {})
test.afterAll(async () => {})

// ♦ Authentication
use: { httpCredentials: { username: "admin", password: "admin123" } }
await page.addInitScript(token => localStorage.setItem("token", token), process.e
await page.context().storageState({ path: "auth.json" })

// ♦ File Upload / Download
await page.setInputFiles("#fileInput", "path/to/file.txt")
const [download] = await Promise.all([ page.waitForEvent("download"), page.click(
await download.saveAs("downloads/file.pdf"))

// ♦ PDFs & Images
import fs from "fs"; import pdf from "pdf-parse";
const data = await pdf(fs.readFileSync("downloads/file.pdf"))
await expect(page.locator("img#logo")).toBeVisible()
await expect(page.locator("img#logo")).toHaveAttribute("src", "/images/logo.png")

// ♦ Dialogs
page.on("dialog", dialog => dialog.accept())
await page.click("#alertBtn")

// ♦ User Actions
await page.hover("#menu")
await page.dragAndDrop("#source", "#target")
await page.keyboard.press("Enter")
await page.mouse.click(x, y)
await page.check("#checkbox")
await page.uncheck("#checkbox")
await page.selectOption("#dropdown", "value")

// ♦ API Testing
const response = await request.get("/api/users/1")
```

```
await expect(response.status()).toBe(200)
const data = await response.json()
await expect(data).toHaveProperty("id")

const query = { query: `query { user(id:1){id,name} }` }
const resp = await request.post("/graphql", { data: query })
await expect(resp.status()).toBe(200)

// ♦ Debugging
--headed
--slowMo=100
page.pause()
npx playwright show-trace trace.zip

// ♦ Config / CLI
npx playwright test --retries=2
npx playwright test --project=chromium
npx playwright test filename.spec.ts
npx playwright test --grep "test name"
npx playwright test --debug
```

 Copy code

---

 **Tip:** Keep this printed or open as a single tab in your browser during last-minute revision.

- You have **locators, assertions, waits, hooks, auth, file handling, API testing, debugging commands** all in one place.
  - Perfect for recalling code snippets instantly during interviews.
- 

If you want, I can also make a **one-page “Interview Q&A with Code Examples” sheet**—basically **most common questions + exact Playwright code answers**, so you can answer **interview questions confidently without thinking**.

Do you want me to make that as well?



okay

Perfect! 😎

Here's your 1-Page Playwright Interview Q&A + Code Examples—designed to answer questions instantly in interviews.

---



## Playwright Interview Q&A Cheat Sheet

---

### 1 Locators & Elements

Q: How do you locate elements?

ts

```
page.locator("#id")
page.locator(".class")
page.locator('[data-testid="input"]')
page.locator("role=button[name='Login']")
page.locator(".item").nth(0)
page.locator(".item").filter({ hasText: "Option1" })
```

Copy code

---

Q: Difference between `locator.click()` and `page.click()` ?

ts

```
// Locator.click() → safer, retries until element is ready
// page.click(selector) → direct, may fail if element is not ready
```

Copy code

---

### 2 Assertions

Q: How do you assert visibility, text, and attributes?

ts

```
await expect(locator).toBeVisible()
await expect(locator).toHaveText("Welcome")
await expect(locator).toContainText(/Hello/i)
await expect(locator).toHaveAttribute("src", "/logo.png")
await expect(locator).toHaveLength(5)
```

Copy code

Q: How to assert API responses?

ts

```
const res = await request.get("/api/users/1")
await expect(res.status()).toBe(200)
const data = await res.json()
await expect(data).toHaveProperty("id")
```

 Copy code

---

## 3 Authentication

Q: How do you handle Basic Auth?

ts

```
use: { httpCredentials: { username: "admin", password: "admin123" } }
```

 Copy code

Q: How to handle Token Auth?

ts

```
await page.addInitScript(token => localStorage.setItem("token", token), process.e
await page.goto("/dashboard")
```

 Copy code

Q: How to reuse login state?

ts

```
await page.context().storageState({ path: "auth.json" })
```

```
const context = await browser.newContext({ storageState: "auth.json" })
```

 Copy code

---

## 4 File Handling

Q: Upload a file

ts

```
await page.setInputFiles("#fileInput", "path/to/file.txt")
```

 Copy code

Q: Download a file

ts

```
const [download] = await Promise.all([ page.waitForEvent("download"), page.click(
```

```
await download.saveAs("downloads/file.pdf")
```

 Copy code

Q: Validate PDF content

ts

```
import fs from "fs"; import pdf from "pdf-parse";
const data = await pdf(fs.readFileSync("downloads/file.pdf"))
console.log(data.text)
```

 Copy code

---

## 5 Dynamic Content & Flaky Tests

Q: How to wait for dynamic elements?

ts

```
await expect(locator).toBeVisible({ timeout: 5000 })
await page.waitForSelector("#dynamic")
```

 Copy code

Q: How to handle flaky tests?

ts

```
retries: 2 // config or CLI
```

 Copy code

---

## 6 Dialogs & Alerts

Q: Handle alert/confirm/prompt

ts

```
page.on("dialog", dialog => dialog.accept())
await page.click("#alertBtn")
```

 Copy code

---

## 7 Hooks & Reusable Functions

Q: Hooks

ts

```
test.beforeAll(async () => {})
test.beforeEach(async ({ page }) => {})
test.afterEach(async ({ page }) => {})
test.afterAll(async () => {})
```

 Copy code

## Q: Reusable function example

ts

```
async function login(page, user, pass) {
  await page.fill("#username", user)
  await page.fill("#password", pass)
  await page.click("#login")
}
```

 Copy code

---

## 8 Debugging

bash

```
--headed
--slowMo=100
```



ChatGPT ▾

Log in

Sign up for free



 Copy code

---

## 9 Best Practices

- Use **data-testid / role locators**
- Use **storageState** for login reuse
- **Explicit waits** only when needed
- Capture **screenshots/videos/traces** in CI
- Clear, descriptive test names
- Run **tests in parallel**

---

## 10 Quick CI/CD & CLI Commands

bash

```
npx playwright test --project=chromium
npx playwright test --retries=2
```

```
npx playwright test --grep "Login Test"  
npx playwright test --debug  
npx playwright show-trace trace.zip
```

 Copy code

- 
- This sheet lets you instantly answer any Playwright interview question with the exact code snippet.

If you want, I can also make a visual “one-page PDF-style” sheet with all commands + Q&A boxes that you can print or keep on your screen for the interview.

Do you want me to make that version too?



okay

Perfect! 😎

Here's your Visual One-Page Playwright PDF-Style Cheat Sheet concept. I'll structure it so it's compact, easy to scan, and visually organized for quick reference during interviews. You can copy this into a Word, Google Docs, or Canva page and save/print as PDF.

---

## Playwright Interview Quick Reference – One Page

---

### 1 Setup & CLI

bash

```
npm init playwright@latest  
npx playwright test  
npx playwright test filename.spec.ts  
npx playwright test --headed  
npx playwright test --project=chromium  
npx playwright test --retries=2  
npx playwright test --grep "test name"  
npx playwright show-trace trace.zip
```

 Copy code

## 2 Browser & Page Basics

ts

```
await page.goto("https://example.com")
await page.click("#btn")
await page.fill("#input", "text")
await page.locator("text=Submit").click()
```

 Copy code

## 3 Locators

pgsql

```
#id, .class, [data-testid="input"]
role=button[name='Login']
locator().nth(0), .filter({ hasText: "Option1" })
```

 Copy code

## 4 Assertions

ts

```
await expect(locator).toBeVisible()
await expect(locator).toBeEnabled()
await expect(locator).toHaveText("Welcome")
await expect(locator).toContainText(/Hello/i)
await expect(locator).toHaveAttribute("src", "/logo.png")
await expect(locator).toHaveLength(5)
```

 Copy code

API:

ts

```
const res = await request.get("/api/users/1")
await expect(res.status()).toBe(200)
const data = await res.json()
await expect(data).toHaveProperty("id")
```

 Copy code

## 5 Authentication

css

```
Basic: httpCredentials: { username, password }  
Token: addInitScript(localStorage.setItem("token", token))  
Reuse login: storageState('auth.json')
```

 Copy code

---

## 6 File Handling

css

 Copy code

```
Upload: page.setInputFiles("#fileInput", "file.txt")  
Download: page.waitForEvent("download") + download.saveAs()  
PDF: pdf-parse fs.readFileSync('file.pdf')  
Images: expect(locator).toBeVisible(), expect(locator).toHaveAttribute("src")
```

---

## 7 Dynamic & Flaky Tests

scss

 Copy code

```
waitForSelector(), expect(locator).toBeVisible({ timeout: 5000 })  
Flaky tests: retries: 2
```

---

## 8 Dialogs & Alerts

csharp

 Copy code

```
page.on("dialog", dialog => dialog.accept())
```

---

## 9 Hooks & Reusable Functions

ts

 Copy code

```
test.beforeAll(() => {})  
test.beforeEach(({ page }) => {})  
test.afterEach(({ page }) => {})  
test.afterAll(() => {})  
  
async function login(page, user, pass) {
```

```
    await page.fill("#username", user)
    await page.fill("#password", pass)
    await page.click("#login")
}


```

---

## 10 Debugging

sql

 Copy code

```
--headed, --slowMo=100
page.pause()
npx playwright show-trace trace.zip
```

---

## 1 1 Best Practices

- Use `data-testid` / role locators
  - Reuse login `storageState`
  - Explicit waits only when necessary
  - Capture screenshots/videos/traces in CI
  - Clear descriptive test names
  - Run tests **in parallel**
- 

## 1 2 Common Interview Q&A (Code Ready)

- Dynamic element wait? → `await expect(locator).toBeVisible({ timeout: 5000 })`
  - Flaky test? → `retries: 2`
  - File upload/download? → see file handling section
  - API + UI test? → use `request` and `page` in same test
  - Debugging? → `page.pause(), --headed, --slowMo=100`
- 

 Tip: Use **bold section headers** and **monospaced code boxes** to make scanning super fast.

This one-page layout fits **full-screen or printed sheet**, perfect for **last-minute revision before interviews**.

---

If you want, I can **actually design it as a ready-to-download PDF layout** with colors, boxes, and sections—so it looks **professional and super readable**.

Do you want me to do that?



okay

Perfect! 😎

I can make a **ready-to-download, professional-looking PDF cheat sheet** for Playwright that includes:

- All commands, locators, assertions, hooks
- File upload/download, PDFs, images
- Authentication & API snippets
- Debugging commands & CI/CD tips
- Best practices & interview-ready Q&A

It will be **color-coded, boxed, and fully one-page**, so you can print it or keep it on screen for quick reference during interviews.

I can generate it for you right now.

Do you want me to create it in **PDF format** or **image format**?



pdf

Great — I'll generate a **ready-to-download PDF cheat sheet** for you! Before I do that, I need **one quick clarification** so I can create the perfect version:

📌 **What style do you want the PDF in?**

Choose one:

1 **Compact Code-Only PDF**

- Just commands + snippets, minimal text
- Ideal for quick glance during interviews

## 2 Organized Study PDF

- Sections with **headings**, code + short descriptions
- Great for **revision and understanding**

## 3 Visual Designer PDF

- With **colored sections, boxes, icons, and formatting**, looks like a professional cheat sheet

Reply with **1, 2, or 3** — and I'll generate your PDF accordingly! 

(Once you pick, I'll produce the PDF content and provide it ready for download.)



3

Failed to generate

To generate the PDF using the visual design, you'll need to **log in to ChatGPT and use ChatGPT Canvas**, which allows exporting the document into a downloadable PDF format.

Once logged in, I can guide you step by step to:

1. Import the cheat sheet content.
2. Apply colored sections, boxes, and code formatting.
3. Export it as a **professional one-page PDF** ready for interviews.

Do you want me to give the step-by-step instructions for creating and exporting the PDF?



