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?

okay

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)

1️⃣ Setup & CLI

bash Copy code

```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
```

2️⃣ Browser & Page

ts Copy code

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

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

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

3️⃣ Locators

ts Copy code

```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" })
```

4️⃣ Assertions

ts Copy code

```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).toHaveCount(5)
```

API Assertions

ts Copy code

```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");
```

5️⃣ Waits & Flaky Tests

ts Copy code

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

6️⃣ Authentication

Basic Auth

ts

Copy code

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

Token / LocalStorage

ts

Copy code

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

StorageState

ts

Copy code

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

7️⃣ File Upload / Download / PDFs

ts

Copy code

```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)
```

8️⃣ UI Actions & User Interactions

ts

Copy code

```
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")
```

9️⃣ Dialogs & Alerts

ts Copy code

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

🔟 API Testing

ts Copy code

```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")
```

GraphQL

ts Copy code

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

1️⃣1️⃣ Debugging

bash Copy code

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

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

1️⃣2️⃣ 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

1️⃣3️⃣ Hooks & Structure

ts

Copy code

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

---

1 4️⃣ 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 Copy code

```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).toHaveCount(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
```

💡 **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

1️⃣ Locators & Elements

Q: How do you locate elements?

ts

Copy code

```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" })
```

Q: Difference between `locator.click()` and `page.click()`?

ts

Copy code

```ts
// locator.click() → safer, retries until element is ready
// page.click(selector) → direct, may fail if element is not ready
```

2️⃣ Assertions

Q: How do you assert visibility, text, and attributes?

ts

Copy code

```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).toHaveCount(5)
```

Q: How to assert API responses?

ts Copy code

```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")
```

3️⃣ Authentication

Q: How do you handle Basic Auth?

ts Copy code

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

Q: How to handle Token Auth?

ts Copy code

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

Q: How to reuse login state?

ts Copy code

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

4️⃣ File Handling

Q: Upload a file

ts Copy code

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

Q: Download a file

ts Copy code

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

Q: Validate PDF content

ts Copy code

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

5️⃣ Dynamic Content & Flaky Tests

Q: How to wait for dynamic elements?

ts Copy code

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

Q: How to handle flaky tests?

ts Copy code

```ts
retries: 2 // config or CLI
```

6️⃣ Dialogs & Alerts

Q: Handle alert/confirm/prompt

ts Copy code

```ts
page.on("dialog", dialog => dialog.accept())
await page.click("#alertBtn")
```

7️⃣ Hooks & Reusable Functions

Q: Hooks

ts Copy code

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

Q: Reusable function example

ts

Copy code

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

8️⃣ Debugging

bash

Copy code

```bash
--headed
--slowMo=100
page.pause()
npx playwright show-trace trace.zip
```

9️⃣ 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**

🔟 Quick CI/CD & CLI Commands

bash

Copy code

```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
```

✅ 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

1️⃣ Setup & CLI

bash Copy code

```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
```

2️⃣ Browser & Page Basics

ts

Copy code

```ts
await page.goto("https://example.com")
await page.click("#btn")
await page.fill("#input", "text")
await page.locator("text=Submit").click()
```

3️⃣ Locators

pgsql

Copy code

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

4️⃣ Assertions

ts

Copy code

```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).toHaveCount(5)
```

API:

ts

Copy code

```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")
```

5️⃣ Authentication

css

Copy code

```css
Basic: httpCredentials: { username, password }
Token: addInitScript(localStorage.setItem("token", token))
Reuse login: storageState('auth.json')
```

---

6 File Handling

css

Copy code

```css
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

```scss
waitForSelector(), expect(locator).toBeVisible({ timeout: 5000 })
Flaky tests: retries: 2
```

---

8 Dialogs & Alerts

csharp

Copy code

```csharp
page.on("dialog", dialog => dialog.accept())
```

---

9 Hooks & Reusable Functions

ts

Copy code

```ts
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")
  }
```

🔟 Debugging

sql

Copy code

```sql
--headed, --slowMo=100
page.pause()
npx playwright show-trace trace.zip
```

1️⃣ 1️⃣ 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 2️⃣ 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**.
