QE Automation Revision Guide

> A structured revision reference covering every automation concept present in this repository — from JavaScript fundamentals through Playwright and Cypress to CI pipelines and AI-assisted testing agents.

Table of Contents

Part 1 — Language Fundamentals (JavaScript)

> 🗀 All files in `JSSamples/`

1.1 Syntax, Variables & Data Types

File: `JSSamples/0JSSyntax.js`, `JSSamples/1VariablesAndDatatypes.js`

JavaScript is case-sensitive. Statements end with `;` (optional but recommended).

```
"use strict"; // Enforces variable declaration, catches unsafe actions

// Three ways to declare variables
let name = "John";      // block-scoped, reassignable
const age = 30;         // block-scoped, NOT reassignable
var legacy = "avoid";   // function-scoped (legacy — avoid)

// Primitive types
let str = "Hello";             // string
let num = 42;                  // number
let bool = true;              // boolean
let undef;                    // undefined
let nul = null;               // null
let sym = Symbol("unique");   // symbol
let big = 9007199254740991n; // BigInt (note the 'n' suffix)

// Non-primitive types
let arr = [1, 2, 3];                              // array
let obj = { firstName: "John", lastName: "Doe" }; // object
```

Type coercion — JS auto-converts types in certain operations:

```
"5" + 10;   // "510"  (string concat wins with +)
"5" - 2;    // 3      (- forces numeric conversion)
"5" * "2";  // 10     (* forces numeric conversion)
```

Explicit conversions: `Number("123")`, `String(456)`, `Boolean("")` → `false`

Template Literals:

```
let greeting = `Hello, ${name}! Age: ${age}`;    // backtick strings
let multiLine = `Line 1
Line 2`;
```

1.2 Operators

File: `JSSamples/2Operators.js`

```
// Arithmetic
+  -  *  /  %  **          // ** = exponentiation (e.g. 2**3 = 8)

// Comparison — KNOW THE DIFFERENCE
x == y;    // loose equality  (compares VALUE only, with coercion)
x === y;   // strict equality (compares VALUE + TYPE, no coercion)
x !== y;   // strict inequality

// Logical
&&  ||  !                  // and, or, not

// Ternary
let canVote = (age >= 18) ? "Yes" : "No";

// Nullish coalescing
let val = null ?? "default";  // "default" (only null/undefined trigger fallback)

// typeof
typeof "hello";  // "string"
typeof 42;        // "number"
typeof undefined; // "undefined"
typeof null;      // "object"  ← known JS quirk
```

1.3 Conditionals & Switch

File: `JSSamples/3Concitions.js`

```
if (num1 >= num2 && num1 >= num3) {
    console.log("num1 is largest");
} else if (num2 >= num1 && num2 >= num3) {
    console.log("num2 is largest");
} else {
    console.log("num3 is largest");
}

// Switch uses STRICT (===) comparison
switch (dayNumber) {
    case 1: dayName = "Monday"; break;
    case 2: dayName = "Tuesday"; break;
    default: dayName = "Invalid";
}
```

1.4 Loops

File: `JSSamples/4Loops.js`, `JSSamples/6ForLoops.js`

```
// Standard for
for (let i = 0; i < 5; i++) { }

// for...of — iterates over VALUES of iterables (arrays, strings)
for (let fruit of ["Apple", "Banana"]) { }

// for...in — iterates over KEYS / indices (use for objects)
for (let key in { name: "John", age: 30 }) { }

// forEach — array method (no break/continue support)
fruits.forEach((fruit, index) => { });

// while / do...while
while (condition) { }     // may never execute
do { } while (condition); // always executes at least once
```

Loop control: `break` exits the loop; `continue` skips to next iteration.

1.5 Arrays

File: `JSSamples/7Arrays.js`

```
let fruits = ["Apple", "Banana", "Cherry"];

// Mutating methods
fruits.push("Date");      // add to end        → returns new length
fruits.pop();             // remove from end    → returns removed item
fruits.unshift("Mango");  // add to beginning   → returns new length
fruits.shift();           // remove from beginning → returns removed item
fruits.splice(1, 1);      // remove 1 element at index 1
fruits.splice(1, 0, "Kiwi"); // insert "Kiwi" at index 1

// Non-mutating / functional (return NEW arrays)
let squared = [1,2,3].map(n => n * n);           // [1, 4, 9]
let evens   = [1,2,3,4].filter(n => n % 2===0);  // [2, 4]
let sum     = [1,2,3].reduce((acc, n) => acc+n, 0); // 6
let found   = [1,2,3].find(n => n > 1);           // 2 (first match)
let idx     = [1,2,3].findIndex(n => n > 1);      // 1

// Searching
fruits.indexOf("Cherry");   // index or -1
fruits.includes("Apple");   // true/false

// Sorting
let nums = [3,1,4,1,5];
nums.sort((a, b) => a - b);  // ascending: [1,1,3,4,5]

// Destructuring
let [a, b, ...rest] = fruits;  // a="Apple", b="Banana", rest=["Cherry"]

// Spread
let combined = [...fruits, ...moreFruits];
let max = Math.max(...numbers);
```

1.6 Functions

File: `JSSamples/8Functions.js`

```
// Named function (hoisted — can call before declaration)
function add(a, b) { return a + b; }

// Anonymous / function expression (NOT hoisted)
let multiply = function(a, b) { return a * b; };

// Arrow function (concise; does NOT have its own 'this')
let subtract = (a, b) => a - b;
let square = n => n * n;          // single param: no parens needed

// Default parameters
function greet(name = "Guest") { return "Hello, " + name; }

// Rest parameters (...collects into array)
```

```
function sumAll(...numbers) {
    return numbers.reduce((a, b) => a + b, 0);
}

// Spread operator (in function calls)
sumAll(...[1, 2, 3]);

// Callback — passing a function as an argument
function calculate(a, b, operation) { return operation(a, b); }
calculate(10, 5, add);             // 15
calculate(10, 5, (a, b) => a * b); // 50

// IIFE — Immediately Invoked Function Expression
(function() { console.log("Runs once"); })();

// Recursive function
function factorial(n) { return n <= 1 ? 1 : n * factorial(n - 1); }
```

1.7 Objects & JSON

File: `JSSamples/9Objects.js`

```
let person = { name: "Alice", age: 25 };

// Access / modify
person.name;          // dot notation
person["age"];        // bracket notation (useful with variables)
person.country = "US"; // add property
delete person.city;    // remove property

// Destructuring
let { name, age } = person;

// Spread
let copy = { ...person, city: "NYC" };

// Iteration
Object.keys(person);    // ["name", "age"]
Object.values(person);  // ["Alice", 25]
Object.entries(person);  // [["name","Alice"],["age",25]]

// JSON conversions
let jsonStr = JSON.stringify(person); // object → JSON string
let parsed  = JSON.parse(jsonStr);    // JSON string → object

// Object methods & 'this'
let rect = {
    width: 10, height: 5,
    area() { return this.width * this.height; }
};
```

```
// Constructor function + prototype
function Car(make, model) { this.make = make; this.model = model; }
Car.prototype.getDetails = function() { return `${this.make} ${this.model}`; };
let car1 = new Car("Toyota", "Camry");
```

1.8 Asynchronous JavaScript

Files: `JSSamples/10Asynchronous.js`, `JSSamples/11Callback.js`, `JSSamples/12Promise.js`, `JSSamples/13AsyncAwait.js`

This is **critical for test automation** — every Playwright command is asynchronous.

```
// ──────────────────────────────────────────────
// 1. Callbacks — the original pattern
// ──────────────────────────────────────────────
function fetchData(callback) {
    setTimeout(() => { callback({ id: 1, name: "John" }); }, 3000);
}
fetchData((data) => { console.log(data); });

// Callback hell — nested callbacks become unreadable
fetchUser((user) => {
    fetchOrders(user.id, (orders) => {
        fetchDetails(orders[0].id, (details) => {
            // deeply nested...
        });
    });
});

// ──────────────────────────────────────────────
// 2. Promises — chainable, cleaner than callbacks
// ──────────────────────────────────────────────
function fetchUser() {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve({ id: 1 }), 3000);
        // call reject(new Error("...")) on failure
    });
}
fetchUser()
    .then(user => updateUser(user))     // returns next promise
    .then(updated => console.log(updated))
    .catch(err => console.error(err))   // catches ANY error in chain
    .finally(() => console.log("done"));

// Promise.all — run in parallel, wait for ALL
await Promise.all([fetchUser(), fetchOrders()]);

// Promise.race — first to settle wins
await Promise.race([fetchUser(), timeout(5000)]);

// ──────────────────────────────────────────────
// 3. Async/Await — reads like synchronous code ⭐
// ──────────────────────────────────────────────
async function processUser() {
    try {
        const user = await fetchUser();       // pauses until resolved
        const updated = await updateUser(user);
        console.log(updated);
    } catch (error) {
        console.error(error);                 // catches rejected promises
    }
}
```

> **Key insight from the repo:** "Playwright has already implemented async/await to handle asynchronous operations like page loading, element interaction, etc. We just need to use `async/await` keywords in our test scripts." — `JSSamples/12Promise.js`

1.9 Scopes & Closures

File: `JSSamples/14Scopes.js`

```
// Global scope — accessible everywhere
let globalVar = "global";

// Function (local) scope
function example() {
    let localVar = "local";    // only accessible inside this function
}

// Block scope — let/const only exist inside { }
if (true) {
    let blockVar = "block";    // NOT accessible outside this if-block
    var hoisted = "leaks";     // var IGNORES block scope — accessible outside
}

// Closure — inner function retains access to outer scope
function initCounter() {
    let count = 0;                             // private variable
    return function() { return ++count; };    // closure over 'count'
}
let counter = initCounter();
counter(); // 1
counter(); // 2 — remembers state between calls
```

1.10 Classes & OOP

File: `JSSamples/15Classes.js`

```
class Person {
    #age;  // private field (prefix with #)
    static count = 0; // shared across all instances

    constructor(name, age) {
        this.name = name;
        this.#age = age;
        Person.count++;
    }

    greet() { console.log(`Hello, I'm ${this.name}`); }

    get info() { return `${this.name}, ${this.#age}`; }  // getter
    set info(value) { /* setter logic */ }
}

// Inheritance
class Employee extends Person {
    constructor(name, age, position) {
        super(name, age);   // MUST call parent constructor first
        this.position = position;
    }
    greet() {  // method override
        super.greet(); // optionally call parent
        console.log(`Position: ${this.position}`);
    }
}

// Static methods — called on the CLASS, not instances
class MathUtil {
    static add(a, b) { return a + b; }
}
MathUtil.add(5, 10); // 15 (not instantiated)

// instanceof check
let emp = new Employee("John", 30, "QA");
emp instanceof Employee;  // true
emp instanceof Person;    // true (inheritance chain)
```

1.11 Debugging

File: `JSSamples/16Debugging.js`

```
// console methods
console.log("message");
console.error("error");
console.warn("warning");
console.table([{a:1},{a:2}]);  // tabular display
console.time("label"); /* code */ console.timeEnd("label"); // measure time

// Breakpoints
debugger;  // pauses execution when DevTools is open
```

```
// Playwright: use --debug flag
// npx playwright test --debug
```

1.12 Exception Handling

File: `JSSamples/17ExceptionHandling.js`

```
try {
    if (b === 0) throw new Error("Division by zero");
    let result = a / b;
} catch (error) {
    console.error(`Error: ${error.message}`);
    console.error(`Stack: ${error.stack}`);
} finally {
    console.log("Always runs — cleanup code");
}

// Custom error class
class ValidationError extends Error {
    constructor(message) {
        super(message);
        this.name = "ValidationError";
    }
}

// Throw custom error
throw new ValidationError("Invalid email format");
```

1.13 Modules (ES6)

Files: `JSSamples/Modules/Module1.js`, `JSSamples/Modules/ReuseModule.js`

```
// ── Module1.js — Exporting ──
export function greet(name) { return `Hello, ${name}!`; }
export const PI = 3.14159;
export default function farewell(name) { return `Goodbye, ${name}!`; }

// ── ReuseModule.js — Importing ──
import farewell from './Module1.js';                // default export (any
name)
import { greet, PI } from './Module1.js';           // named exports (exact
names)
import * as Module1 from './Module1.js';           // namespace import
```

> Requires `"type": "module"` in `package.json` (as this repo uses).

Part 2 — TypeScript for Test Automation

🗀 `TSSamples/demo.ts`

TypeScript is a **superset of JavaScript** that adds static typing. All JS code is valid TS. The compiler (`tsc`) transpiles `.ts` → `.js`.

2.1 Basic Types

```
let num: number = 10;
let str: string = "Hello";
let bool: boolean = true;
let arr: number[] = [1, 2, 3];
let tuple: [string, number] = ["Hello", 10]; // fixed types per position
let anything: any = "no type checking";       // escape hatch — avoid
let nothing: void = undefined;                // function returns nothing
let nope: never;   // function never returns (throws or infinite loop)

// Enum — named constants
enum Color { Red, Green, Blue }
let c: Color = Color.Green;  // 1 (auto-numbered from 0)
```

2.2 Interfaces & Type Aliases

```
// Interface — defines the shape of an object
interface Person {
    name: string;
    age: number;
    greet(): void;
}

// Optional properties
interface Config {
    host: string;
    port?: number;   // optional
}

// Union type
let id: string | number;

// Type alias
type ID = string | number;
```

2.3 Generics

```
function identity<T>(arg: T): T { return arg; }
let output = identity<string>("Hello");
let inferred = identity(42);  // T inferred as number
```

2.4 Classes with Types

```
class Animal {
    name: string;
    constructor(name: string) { this.name = name; }
    speak(): void { console.log(this.name + " makes a noise."); }
}
class Dog extends Animal {
    speak(): void { console.log(this.name + " barks."); }
}
```

2.5 tsconfig.json

File: `tsconfig.json`

```
{
  "compilerOptions": {
    "module": "nodenext",       // enables ES modules
    "target": "esnext",         // compile target
    "strict": true,             // enable all strict checks
    "sourceMap": true,          // .map files for debugging
    "declaration": true,        // generate .d.ts files
    "skipLibCheck": true,       // skip checking node_modules types
    "isolatedModules": true     // per-file compilation safety
  }
}
```

Part 3 — Node.js Project Structure & Configuration

3.1 package.json

File: `package.json`

```
{
  "type": "module",
  "scripts": {
    "start:qe-api": "node qe-local-api/server.js"
  },
  "devDependencies": {
    "@playwright/test": "^1.57.0",
    "cypress": "^15.9.0",
    "typescript": "^5.9.3"
  },
  "dependencies": {
    "exceljs": "^4.4.0",
    "express": "^4.21.2",
    "mysql": "^2.18.1",
    "mysql2": "^3.16.2",
    "cors": "^2.8.5"
  }
}
```

Key	Purpose
`type: "module"`	Enables ES module `import/export` syntax (not CommonJS `require`)
`dependencies`	Required at runtime (express, exceljs, mysql)
`devDependencies`	Only for dev/test (playwright, cypress, typescript)
`scripts`	Runnable via `npm run <name>`

3.2 Playwright Config

File: `playwright.config.ts`

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

export default defineConfig({
  testDir: './tests',
  fullyParallel: true,
  forbidOnly: !!process.env.CI,          // fail if test.only in CI
  retries: process.env.CI ? 2 : 0,       // retry on CI only
  workers: process.env.CI ? 1 : undefined,// single worker in CI
  reporter: 'html',
  use: {
    trace: 'on',      // records trace for Trace Viewer debugging
    video: 'on',      // records video of every test
  },
  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'], headless: false },
    },
  ],
});
```

Setting	What it does
`fullyParallel`	Tests in different files run in parallel
`retries`	Auto-retry failed tests (CI gets 2 retries)
`trace: 'on'`	Records trace for post-mortem debugging (Trace Viewer)
`video: 'on'`	Records video of every test run
`projects`	Define browser targets (chromium, firefox, webkit)

Setting	What it does
`forbidOnly`	Prevents `test.only` from accidentally running in CI

3.3 Cypress Config

File: `cypress.config.ts`

```ts
import { defineConfig } from "cypress";

export default defineConfig({
  e2e: {
    setupNodeEvents(on, config) { /* Node event listeners */ },
    defaultCommandTimeout: 20000,  // wait 20s for commands
    pageLoadTimeout: 60000         // wait 60s for page load
  },
});
```

Part 4 — Element Locator Strategies

4.1 CSS Selectors

File: `TSSamples/CSSforTestAutomation.ts`

```
Basic:
  tag                   → input, div, button
  #id                   → input#username
  .class                → input.form-control
  [attr='value']        → input[type='text']
  combined              → input#username.form-control[type='text']

Attribute selectors:
  [attr='val']          → exact match
  [attr*='val']         → contains
  [attr^='val']         → starts with
  [attr$='val']         → ends with

Hierarchy:
  space                 → descendant (any depth)    div form input
  >                     → direct child              div > form > input
  +                     → immediate next sibling    label + input
  ~                     → all following siblings    label ~ input

Pseudo-classes:
  :first-child          → first child of parent
  :last-child           → last child of parent
  :nth-child(n)         → nth child (1-indexed)
  :nth-of-type(n)       → nth of specific tag type
  :not(selector)        → negation
```

Playwright-specific pseudo-classes:

```
page.locator('button:visible');                        // only visible elements
page.locator('article:has-text("Playwright")');        // contains text
page.locator('input:right-of(:text("Username"))');    // layout-based
page.locator('button:has(span.icon)');                 // has matching child
```

4.2 XPath

File: `TSSamples/XpathForTestAutomation.ts`

```
Absolute:  /html/body/div/form/input         (fragile — avoid)
Relative:  //input[@attribute='value']        (preferred)

Attribute matching:
  //input[@id='user']
  //input[@type='text' and @name='email']    (multiple conditions)
  //input[@type='text' or @name='email']

Text functions:
  text()        → //button[text()='Submit']          (exact match)
  contains()    → //div[contains(text(),'Welcome')]   (partial match)
  starts-with() → //input[starts-with(@id,'user')]
  normalize-space() → //span[normalize-space()='Dashboard']

Positional:
  (//input[@type='text'])[2]     → 2nd match (1-indexed)
  (//div[@class='item'])[last()] → last match
  (//div)[position()>2]          → 3rd onward

Axes (navigation from current node):
  child::       → //div/child::input             (direct children)
  parent::      → //input/parent::div            (shortcut: //input/..)
  ancestor::    → //input/ancestor::form          (any ancestor)
  descendant::  → //form/descendant::input        (shortcut: //form//input)
  following-sibling:: → //label/following-sibling::input
  preceding-sibling:: → //input/preceding-sibling::label
```

> In Playwright, locators starting with `//` or `xpath=` are auto-detected as XPath.

4.3 Playwright Built-in Locators (Recommended)

Files: `tests/orangehrm.spec.ts`, `tests/tsrtc.spec.ts`

```
page.getByRole('button', { name: 'Submit' })    // accessible role
page.getByRole('textbox', { name: 'Username' }) // input by role
page.getByLabel('Username')                     // associated <label>
page.getByPlaceholder('Enter your email')       // placeholder attr
page.getByText('Welcome to the site')           // visible text
page.getByTestId('submit-button')               // data-testid attr
page.getByAltText('Company Logo')               // img alt text
page.getByTitle('More information')             // title attr
```

> These are **user-facing** locators — preferred over CSS/XPath for resilience to DOM changes.

Part 5 — Playwright Fundamentals

> 🗀 `tests/`

5.1 Test Structure

File: `tests/example.spec.ts`

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

test('has title', async ({ page }) => {
    await page.goto('https://playwright.dev/');
    await expect(page).toHaveTitle(/Playwright/);
});

test('get started link', async ({ page }) => {
    await page.goto('https://playwright.dev/');
    await page.getByRole('link', { name: 'Get started' }).click();
    await expect(page.getByRole('heading', { name: 'Installation'
})).toBeVisible();
});
```

Every test receives **fixtures** as destructured params: `{ page }`, `{ browser }`, `{ context }`, `{ request }`.

5.2 Test Suite with describe

```
test.describe('OrangeHRM Tests', () => {
    test('login test', async ({ page }) => { /* ... */ });
    test('add employee', async ({ page }) => { /* ... */ });
});
```

5.3 Hooks (Setup / Teardown)

File: `tests/pwTestExamples.spec.ts`

```
test.describe('OrangeHRM Tests', () => {
    test.beforeAll(async ({ browser }) => {
        // Runs ONCE before all tests — e.g., create shared context
    });

    test.beforeEach(async ({ page }) => {
        // Runs before EACH test — e.g., login
        await page.goto('https://opensource-demo.orangehrmlive.com');
        await page.getByPlaceholder('Username').fill('Admin');
        await page.getByPlaceholder('Password').fill('admin123');
        await page.getByRole('button', { name: 'Login' }).click();
    });

    test.afterEach(async ({ page }) => {
        // Runs after EACH test — e.g., logout / cleanup
    });

    test.afterAll(async ({ browser }) => {
        // Runs ONCE after all tests
    });

    test('Add Employee', async ({ page }) => {
        // Already logged in from beforeEach
    });
});
```

5.4 Assertions

```
// —— Page-level assertions ——
await expect(page).toHaveTitle(/Playwright/);
await expect(page).toHaveURL(/.*dashboard/);

// —— Element-level assertions ——
await expect(page.getByRole('heading', { name: 'Dashboard' })).toBeVisible();
await expect(page.locator('.error')).toHaveText('Invalid credentials');
await expect(employeeIdField).toHaveValue(/^\d+$/);
await expect(page.locator('li')).toHaveCount(3);
await expect(page.locator('.btn')).toBeEnabled();
await expect(page.locator('.btn')).toBeDisabled();

// —— Negation ——
await expect(page.locator('.error')).not.toBeVisible();

// —— Soft assertions (don't stop test on failure) ——
await expect.soft(page.locator('.warning')).toBeVisible();

// —— Custom timeout per assertion ——
await expect(heading).toBeVisible({ timeout: 10000 });
```

5.5 Navigation & Waits

```
await page.goto('https://example.com');
await page.goBack();
await page.goForward();
await page.reload();

// Wait strategies
await page.waitForURL(/.*\/pim\/viewPersonalDetails/);
await page.waitForSelector('div:has-text("loaded")');
await page.waitForLoadState('networkidle');
await page.waitForTimeout(5000); // hard wait — use ONLY as last resort
```

5.6 Element Actions

```
await page.getByPlaceholder('Username').fill('Admin');  // clear + type
await page.getByRole('button', { name: 'Login' }).click();
await page.locator('#draggable').dragTo(page.locator('#droppable'));
await page.selectOption('select#country', 'US');
await page.check('input[type="checkbox"]');
await page.uncheck('input[type="checkbox"]');
await page.setInputFiles('input[type="file"]', 'path/to/file.pdf');
```

5.7 Working with Locator Lists

File: `tests/apsrtc.spec.ts`

```
// Get all matching elements
const busList = await page.locator('div.srvceNO:visible').all();
console.log('Count:', busList.length);

for (const bus of busList) {
    const busName = await bus.textContent();
    console.log('Bus:', busName?.trim());
}

// Count assertion
await expect(page.locator('div.srvceNO:visible')).toHaveCount(5);
```

5.8 Filtering & Chaining Locators

File: `tests/apsrtc.spec.ts`

```
// Filter parent by child text, then find a button within
await page.locator(`div.row:has-text("${serviceNumber}")`)
    .getByRole('button', { name: 'Select Seats' }).click();

// Filter using locator method
```

```
await page.locator('span').filter({ hasText: 'user name' }).click();

// Chain locators
await page.locator('.sidebar').locator('.menu-item').first().click();
```

5.9 Serial Test Execution

File: `tests/qe-api.spec.ts`

```
test.describe.serial("QE Local API", () => {
    // Tests run one after another, in declared order
    // If test A fails, tests B and C are skipped
    test("A: setup", async ({ request }) => { /* ... */ });
    test("B: depends on A", async ({ request }) => { /* ... */ });
    test("C: depends on B", async ({ request }) => { /* ... */ });
});
```

5.10 Custom Timeouts

File: `tests/cookiestest.spec.ts`

```
// File-level timeout
test.setTimeout(100000);

// Per-test timeout
test('slow test', async ({ page }) => {
    test.setTimeout(60000);
    // ...
});
```

5.11 Seed Tests (Reusable Setup)

File: `tests/seed.spec.ts`

```
// Seed file — validates prerequisites before main test suite
test('Login works', async ({ page }) => {
    await page.goto('https://...');
    await page.getByPlaceholder('Username').fill('Admin');
    await page.getByPlaceholder('Password').fill('admin123');
    await page.getByRole('button', { name: 'Login' }).click();
    await expect(page.getByRole('heading', { name: 'Dashboard' })).toBeVisible();
});
```

Part 6 — Cypress Fundamentals

> 🗀 `cypress/e2e/`

6.1 Test Structure (Mocha + Chai)

File: `cypress/e2e/1-getting-started/todo.cy.js`

```
/// <reference types="cypress" />

describe('example to-do app', () => {
    beforeEach(() => {
        cy.visit('https://example.cypress.io/todo');
    });

    it('displays two todo items by default', () => {
        cy.get('.todo-list li').should('have.length', 2);
        cy.get('.todo-list li').first().should('have.text', 'Pay electric bill');
        cy.get('.todo-list li').last().should('have.text', 'Walk the dog');
    });

    it('can add new todo items', () => {
        const newItem = 'Feed the cat';
        cy.get('[data-test=new-todo]').type(`${newItem}{enter}`);
        cy.get('.todo-list li').should('have.length',
3).last().should('have.text', newItem);
    });
});
```

Cypress uses **Mocha** for test structure (`describe`, `it`, `before`, `beforeEach`, `after`, `afterEach`) and **Chai** for assertions (`.should()`, `expect()`).

6.2 Querying Elements

File: `cypress/e2e/2-advanced-examples/querying.cy.js`

```
cy.get('#id');                             // by ID
cy.get('.class');                          // by class
cy.get('[data-test-id="example"]');        // by attribute
cy.contains('bananas');                    // by text content
cy.get('.query-form').find('input');       // scoped find

// Scoped querying with within()
cy.get('.query-form').within(() => {
    cy.get('input:first').should('have.attr', 'placeholder', 'Email');
    cy.get('input:last').should('have.attr', 'placeholder', 'Password');
});

// Root — escape from within() scope
cy.get('.query-ul').within(() => {
    cy.root().should('have.class', 'query-ul');
});
```

6.3 Implicit vs Explicit Assertions

File: `cypress/e2e/2-advanced-examples/assertions.cy.js`

```
// ── Implicit — chained on commands (retries automatically) ──
cy.get('.element')
    .should('have.class', 'active')
    .and('have.attr', 'href')
    .and('include', 'cypress.io');

cy.get('tbody tr:first').should('have.class', 'success');

// ── Explicit — BDD style with expect() ──
expect(true).to.be.true;
expect({ foo: 'bar' }).to.deep.equal({ foo: 'bar' });
expect('FooBar').to.match(/bar$/i);
expect([1, 2]).to.have.length(2);

// ── Callback assertion with should() ──
cy.get('.assertions-link')
    .should('have.class', 'active')
    .and('have.attr', 'href')
    .and(($a) => {
        expect($a).to.have.length(1);
        const className = $a[0].className;
        expect(className).to.match(/active/);
    });
```

6.4 Actions

File: `cypress/e2e/2-advanced-examples/actions.cy.js`

```
cy.get('.action-email').type('user@test.com');              // type
cy.get('.action-email').type('{selectall}{backspace}');      // special keys
cy.get('.action-btn').click();                              // click
cy.get('.action-btn').click({ position: 'topLeft' });       // click position
cy.get('#action-canvas').click(80, 75);                     // click coords
cy.get('.action-btn').dblclick();                           // double click
cy.get('.action-btn').rightclick();                         // right click
cy.get('.action-checkboxes [type="checkbox"]').check();     // check
cy.get('.action-checkboxes [type="checkbox"]').uncheck();   // uncheck
cy.get('.action-select').select('bananas');                  // dropdown
cy.get('.action-inputrange').invoke('val', 25).trigger('change'); // slider
cy.get('.action-email').clear();                            // clear input
cy.get('.action-form').submit();                            // submit form
cy.get('.disabled').type('text', { force: true });          // bypass checks
cy.get('#scroll-horizontal').scrollTo('right');             // scroll
cy.get('#scroll-both').scrollTo(300, 200);                  // scroll by coords
```

6.5 Aliasing

File: `cypress/e2e/2-advanced-examples/aliasing.cy.js`

```
// Alias a DOM element
cy.get('.as-table').find('tbody>tr').first().find('td').first()
    .find('button').as('firstBtn');
cy.get('@firstBtn').click();

// Alias a network route
cy.intercept('GET', '**/comments/*').as('getComment');
cy.get('.network-btn').click();
cy.wait('@getComment').its('response.statusCode').should('eq', 200);

// Alias a fixture
cy.fixture('example.json').as('testData');
```

6.6 Network Requests & Intercepts

File: `cypress/e2e/2-advanced-examples/network_requests.cy.js`

```
// cy.request — bypass the browser (no CORS, no UI)
cy.request('https://jsonplaceholder.cypress.io/comments')
    .should((response) => {
        expect(response.status).to.eq(200);
        expect(response.body).to.have.length(500);
    });

// cy.intercept — intercept & modify network traffic
cy.intercept('GET', '**/comments/*').as('getComment');
cy.get('.btn').click();
cy.wait('@getComment').its('response.statusCode').should('eq', 200);

// Stub response
cy.intercept('GET', '**/comments/*', { body: { id: 1 } }).as('stubbed');
```

6.7 Waiting

File: `cypress/e2e/2-advanced-examples/waiting.cy.js`

```
// Hard wait (avoid when possible)
cy.wait(1000);

// Wait for aliased network request (preferred pattern)
cy.intercept('GET', '**/comments/*').as('getComment');
cy.get('.network-btn').click();
cy.wait('@getComment').its('response.statusCode').should('be.oneOf', [200, 304]);
```

6.8 `within()` vs `then()`

File: `cypress/e2e/practice/test5WithInVsThen.cy.js`

```
// within() — scopes ALL cy commands to the matched element
cy.get('.oxd-form').within(() => {
    cy.get('input[name="username"]').type('admin');  // scoped to form
    cy.get('input[name="password"]').type('admin123');
});

// then() — gives you the jQuery element; cy.get() still searches from root
cy.get('.oxd-form').then(($form) => {
    // cy.get() here would search the FULL DOM, not just $form
    cy.wrap($form).find('input[name="username"]').type('admin');
    // Use cy.wrap($el).find() to stay scoped
});
```

> **Key difference:** `within()` limits ALL child `cy.get()` calls to the element's subtree. `then()` gives you the element but doesn't change `cy.get()` scope.

6.9 Fixtures

Files: `cypress/e2e/practice/test6Fixture.cy.js`, `test7FixtureAlias.cy.js`

```
// Direct usage — inside .then()
cy.fixture('userdata').then((user) => {
    cy.wrap($form).find('input[name="username"]').type(user.username);
    cy.wrap($form).find('input[name="password"]').type(user.password);
});

// Aliased usage — load once in beforeEach, use everywhere
describe('Login', () => {
    beforeEach(() => {
        cy.fixture('userdata.json').as('user');
    });

    it('should login', function() {     // NOTE: must use function(), not arrow
        cy.get('@user').then((user) => {
            // user.username, user.password
        });
    });
});
```

> Fixtures live in `cypress/fixtures/`. No import needed — `cy.fixture()` reads them.

6.10 Cookies

File: `cypress/e2e/2-advanced-examples/cookies.cy.js`

```
cy.getCookie('token').should('have.property', 'value', '123ABC');
cy.getCookies().should('have.length', 1);
cy.setCookie('key', 'value');
cy.clearCookie('token');
cy.clearCookies();
cy.clearAllCookies();
```

6.11 Spies, Stubs & Clock

File: `cypress/e2e/2-advanced-examples/spies_stubs_clocks.cy.js`

```
// Spy — observe function calls without modifying behavior
const obj = { foo() { console.log('real'); } };
cy.spy(obj, 'foo').as('fooSpy');
obj.foo();
cy.get('@fooSpy').should('have.been.calledOnce');

// Stub — replace a function's implementation
cy.stub(obj, 'foo').as('fooStub');
obj.foo('a', 'b');
cy.get('@fooStub').should('have.been.calledWith', 'a', 'b');

// Clock — control JavaScript time
cy.clock(Date.UTC(2017, 2, 14));  // freeze time
cy.get('#clock-div').click().should('have.text', '1489449600');
cy.tick(10000);                    // advance time by 10 seconds
```

6.12 Custom Commands

File: `cypress/support/commands.ts`

```
// Definition
Cypress.Commands.add('login', (username, password) => {
    // reusable login logic
});

// Usage in tests
cy.login('admin', 'admin123');
```

6.13 Support File

File: `cypress/support/e2e.ts`

```
import './commands'; // auto-loaded before every spec file
```

Part 7 — Page Object Model (POM)

> 🗀 `src/pages/`

POM encapsulates **locators** and **actions** for each page in a dedicated class, keeping tests clean and maintainable.

7.1 Page Class Structure

File: `src/pages/loginpage.ts`

```
import { expect, type Locator, type Page } from '@playwright/test';

export class LoginPage {
    page: Page;
    #userName: Locator;   // private locators (TypeScript # syntax)
    #password: Locator;
    #loginButton: Locator;

    constructor(page: Page) {
        this.page = page;
        this.#userName = this.page.getByPlaceholder('Username');
        this.#password = this.page.getByPlaceholder('Password');
        this.#loginButton = this.page.getByRole('button', { name: 'Login' });
    }

    async navigate() {
        await this.page.goto(
            'https://opensource-demo.orangehrmlive.com/web/index.php/auth/login'
        );
    }
    async enterUserName(username: string) { await this.#userName.fill(username); }
    async enterPassword(password: string) { await this.#password.fill(password); }
    async clickLogin() { await this.#loginButton.click(); }
    async verifyLogin() {
        await expect(
            this.page.getByRole('heading', { name: 'Dashboard' })
        ).toBeVisible();
    }
}
```

File: `src/pages/dashboardpage.ts`

```
export class DashboardPage {
    page: Page;
    #pim: Locator;
    constructor(page: Page) {
        this.page = page;
        this.#pim = this.page.locator('span:has-text("PIM")');
    }

    async clickPIM() { await this.#pim.click(); }
    async verifyDashboard() {
        await expect(
            this.page.getByRole('heading', { name: 'Dashboard' })
        ).toBeVisible();
    }
}
```

File: `src/pages/addEmployeePage.ts`

```
export class AddEmployeePage {
    page: Page;
    #firstName: Locator;
    #lastName: Locator;
    #saveButton: Locator;

    constructor(page: Page) {
        this.page = page;
        this.#firstName = this.page.getByPlaceholder('First Name');
        this.#lastName = this.page.getByPlaceholder('Last Name');
        this.#saveButton = this.page.getByRole('button', { name: 'Save' });
    }

    async enterFirstName(fn: string) { await this.#firstName.fill(fn); }
    async enterLastName(ln: string) { await this.#lastName.fill(ln); }
    async clickSave() { await this.#saveButton.click(); }
    async verifyEmployeeAdded() {
        await this.page.waitForURL(/.*\/pim\/viewPersonalDetails/);
    }
}
```

Pattern:

1. Constructor takes `Page`, initializes locators
2. Private `#` locators — not accessible outside the class
3. Public `async` methods for each user action
4. Assertion methods (verify*) stay in the page object

7.2 Using POM (Direct Instantiation)

File: `tests/OhrmPOMtest.spec.ts`

```
import { LoginPage } from '../src/pages/loginpage';
import { DashboardPage } from '../src/pages/dashboardpage';
import { EmployeeListPage } from '../src/pages/employeeListPage';
import { AddEmployeePage } from '../src/pages/addEmployeePage';

test.describe('OrangeHRM POM Tests', () => {
    let loginPage: LoginPage;
    let dashboardPage: DashboardPage;
    let employeeListPage: EmployeeListPage;
    let addEmployeePage: AddEmployeePage;

    test.beforeEach(async ({ page }) => {
        loginPage = new LoginPage(page);
        dashboardPage = new DashboardPage(page);
        employeeListPage = new EmployeeListPage(page);
        addEmployeePage = new AddEmployeePage(page);
    });

    test('Add Employee', async ({ page }) => {
        await loginPage.navigate();
        await loginPage.enterUserName('Admin');
        await loginPage.enterPassword('admin123');
        await loginPage.clickLogin();
        await loginPage.verifyLogin();
        await dashboardPage.clickPIM();
        await employeeListPage.clickAddEmployee();
        await addEmployeePage.enterFirstName('John');
        await addEmployeePage.enterLastName('Doe');
        await addEmployeePage.clickSave();
        await addEmployeePage.verifyEmployeeAdded();
    });
});
```

Part 8 — Custom Fixtures & Base Test Extension

> 🗀 `src/basetest.ts`, `src/baseRequest.ts`

8.1 Extending `test` with Custom Fixtures

Playwright's `test.extend()` injects page objects as fixtures — eliminating manual instantiation in every `beforeEach`.

File: `src/basetest.ts`

```
import { test as base } from '@playwright/test';
import { LoginPage } from '../src/pages/loginpage';
import { DashboardPage } from '../src/pages/dashboardpage';
import { EmployeeListPage } from '../src/pages/employeeListPage';
import { AddEmployeePage } from '../src/pages/addEmployeePage';

export const BaseTest = base.extend<{
    loginPage: LoginPage;
    dashboardPage: DashboardPage;
    employeeListPage: EmployeeListPage;
    addEmployeePage: AddEmployeePage;
}>({
    loginPage: async ({ page }, use) => { await use(new LoginPage(page)); },
    dashboardPage: async ({ page }, use) => { await use(new DashboardPage(page));
},
    employeeListPage: async ({ page }, use) => { await use(new
EmployeeListPage(page)); },
    addEmployeePage: async ({ page }, use) => { await use(new
AddEmployeePage(page)); },
});
```

How it works:

1. `base.extend<{ ... }>()` declares new fixture names and their types
2. Each fixture function receives existing fixtures (`{ page }`) and a `use` callback
3. Create the object, pass it to `use()`, and it's available in tests
4. Teardown logic goes after `await use(...)` if needed

8.2 Using Custom Fixtures in Tests

File: `tests/OhrmPOMtest-custom.spec.ts`

```
import { BaseTest as test } from '../src/basetest';

test.describe('OrangeHRM POM Tests', () => {
    test('Add Employee', async ({
        loginPage,         // injected automatically
        dashboardPage,
        employeeListPage,
        addEmployeePage
    }) => {
        await loginPage.navigate();
        await loginPage.enterUserName('Admin');
        await loginPage.enterPassword('admin123');
        await loginPage.clickLogin();
        // ... clean, no setup code
    });
});
```

> **Benefit:** No `beforeEach` boilerplate. Each test declares exactly what it needs as parameters.

8.3 API Request Fixture

File: `src/baseRequest.ts`

```
import { test as base, APIRequestContext } from '@playwright/test';
```

```
export const BaseRequestTest = base.extend<{
    apiRequest: APIRequestContext
}>({
    apiRequest: async ({ playwright }, use) => {
        const apiRequestContext = await playwright.request.newContext({
            baseURL: 'https://api.example.com',
            extraHTTPHeaders: {
                'X-API-Key': process.env.API_KEY || 'your-api-key',
            },
        });
        await use(apiRequestContext);
        await apiRequestContext.dispose(); // cleanup after test
    },
});
```

Part 9 — Data-Driven Testing

9.1 JSON Import (Playwright)

File: `tests/orangehrmdt.spec.ts`

```
import * as ohrmdata from '../src/data/ohrmdata.json' with { type: 'json' };

test('login to orangehrm', async ({ page }) => {
    await page.getByPlaceholder('Username').fill(ohrmdata.adminuser.username);
    await page.getByPlaceholder('Password').fill(ohrmdata.adminuser.password);
});
```

Data file (`src/data/ohrmdata.json`):

```
{
    "adminuser": { "username": "admin", "password": "admin123" },
    "url": "https://opensource-demo.orangehrmlive.com/web/index.php/auth/login"
}
```

> Note: `with { type: 'json' }` is the ES module JSON import assertion syntax.

9.2 Fixture Files (Cypress)

File: `cypress/e2e/practice/test6Fixture.cy.js`

```
cy.fixture('userdata').then((user) => {
    cy.wrap($form).find('input[name="username"]').type(user.username);
    cy.wrap($form).find('input[name="password"]').type(user.password);
});
```

9.3 Fixture with Alias (Cypress)

File: `cypress/e2e/practice/test7FixtureAlias.cy.js`

```
describe('Login Tests', () => {
    beforeEach(() => {
        cy.fixture('userdata.json').as('user'); // alias for later
    });

    it('login test', function() {   // must use function(), not () =>
        cy.get('@user').then((user) => {
            // user.username, user.password available
        });
    });
});
```

9.4 Parameterized Tests (forEach Pattern)

File: `src/data/HandlingData.js`

```
const testData = [
    { username: 'user1', password: 'pass1', expected: 'Dashboard' },
    { username: 'user2', password: 'pass2', expected: 'Error' },
];

testData.forEach(({ username, password, expected }) => {
    test(`Login test for ${username}`, async ({ page }) => {
        await page.fill('#username', username);
        await page.fill('#password', password);
        // assert expected outcome
    });
});
```

9.5 Reading Data from Files

Reading JSON — `src/data/ReadJson.ts`:

```
import fs from 'fs';
import path from 'path';

const filePath = path.resolve(".", fileName);
const data = fs.readFileSync(filePath, 'utf-8');
const parsed = JSON.parse(data);
```

Reading/Writing Text — `src/data/ReadTxtData.ts`, `WriteTextData.ts`:

```
import fs from 'fs';

// Read
const content = fs.readFileSync(fileName, 'utf-8');

// Write (overwrite)
fs.writeFileSync(fileName, data, 'utf-8');

// Append
fs.appendFileSync(fileName, data, 'utf-8');
```

Reading/Writing Excel — `src/data/ReadExcel.ts`, `WriteExcelData.ts`, `UpdateExcelData.ts`:

```
import ExcelJS from 'exceljs';

let workbook = new ExcelJS.Workbook();
await workbook.xlsx.readFile("src/data/Ohrm.xlsx");
let worksheet = workbook.getWorksheet('AddEmp');

// Read cell by cell
for (let r = 1; r <= worksheet.rowCount; r++) {
    for (let c = 1; c <= worksheet.columnCount; c++) {
        console.log(worksheet.getRow(r).getCell(c).value);
    }
}

// Write new row
worksheet.addRow({ id: 7, name: 'John', role: "HR" });
await workbook.xlsx.writeFile("src/data/output.xlsx");

// Update existing cell
worksheet.getRow(2).getCell(3).value = "Updated Value";
await workbook.xlsx.writeFile("src/data/Ohrm.xlsx");
```

9.6 Database Operations

Files: `src/data/ReadDataFromDB.ts`, `ReadDataFromDBNew.ts`, `WriteDataToDB.ts`

```
import mysql from 'mysql2';

const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '***',
    database: 'company',
    port: 3306
});
```

```
connection.connect();

// Read
connection.query("SELECT * FROM emp", (error, results) => {
    if (error) throw error;
    console.log('Data:', results);
});

// Write
connection.query(
    "INSERT INTO emp (name, role, salary) VALUES (?, ?, ?)",
    ['John', 'QA', 50000],
    (error, results) => { if (error) throw error; }
);

connection.end();
```

Promise-based alternative (`ReadDataFromDBNew.ts`):

```
import mysql from 'mysql2/promise';

const connection = await mysql.createConnection({ /* config */ });
const [rows] = await connection.execute('SELECT * FROM emp');
console.log(rows);
await connection.end();
```

Part 10 — API Testing

10.1 Playwright API Testing (Standalone Script)

File: `TSSamples/PlaywrightApiTests.ts`

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

const apiContext = await request.newContext({
    baseURL: 'https://jsonplaceholder.typicode.com',
    extraHTTPHeaders: { 'Content-Type': 'application/json' },
});

// POST
const postResponse = await apiContext.post('/posts', {
    data: { title: 'foo', body: 'bar', userId: 1 },
});
console.log('Status:', postResponse.status());      // 201
const postData = await postResponse.json();

// GET
const getResponse = await apiContext.get('/posts/1');
const getData = await getResponse.json();

// PUT
const putResponse = await apiContext.put('/posts/1', {
    data: { id: 1, title: 'updated', body: 'updated body', userId: 1 },
});

// PATCH
const patchResponse = await apiContext.patch('/posts/1', {
    data: { title: 'patched title' },
});

// DELETE
const deleteResponse = await apiContext.delete('/posts/1');

await apiContext.dispose(); // always clean up
```

10.2 API Testing with Token Auth

File: `TSSamples/RestfulBookerApiTests.ts`

```
// Step 1: Authenticate and get token
let authResponse = await apiContext.post('/auth', {
    data: { username: 'admin', password: 'password123' }
});
let token = (await authResponse.json()).token;

// Step 2: Create authenticated context with token cookie
const authApiContext = await request.newContext({
    baseURL: 'https://restful-booker.herokuapp.com',
    extraHTTPHeaders: {
        'Content-Type': 'application/json',
        'Cookie': `token=${token}`     // inject token
    }
});

// Step 3: Protected operations
const updateResponse = await authApiContext.put(`/booking/${id}`, {
    data: { firstname: "Updated", lastname: "User", /* ... */ }
});
const deleteResponse = await authApiContext.delete(`/booking/${id}`);
```

10.3 Full CRUD Test Suite (Playwright Test Runner)

File: `tests/qe-api.spec.ts`

```
import { test, expect } from "@playwright/test";
const baseURL = process.env.QE_API_URL || "http://localhost:3000";
```

```
test.describe.serial("QE Local API", () => {
    test("health check", async ({ request }) => {
        const response = await request.get(`${baseURL}/health`);
        expect(response.status()).toBe(200);
        const body = await response.json();
        expect(body.data.status).toBe("ok");
    });

    test("users CRUD flow", async ({ request }) => {
        // CREATE
        const create = await request.post(`${baseURL}/users`, {
            data: { name: "QA Student", email: "qa@example.com" },
        });
        expect(create.status()).toBe(201);
        const userId = (await create.json()).data.user.id;

        // READ
        const get = await request.get(`${baseURL}/users/${userId}`);
        expect(get.status()).toBe(200);

        // UPDATE
        const update = await request.put(`${baseURL}/users/${userId}`, {
            data: { name: "Updated Student" },
        });
        expect(update.status()).toBe(200);

        // DELETE
        const del = await request.delete(`${baseURL}/users/${userId}`);
        expect(del.status()).toBe(200);

        // VERIFY DELETION
        const after = await request.get(`${baseURL}/users/${userId}`);
        expect(after.status()).toBe(404);
    });

    test("orders require auth", async ({ request }) => {
        // Without token → 401
        const unauth = await request.get(`${baseURL}/orders`);
        expect(unauth.status()).toBe(401);

        // Login → get token
        const login = await request.post(`${baseURL}/auth/login`, {
            data: { username: "admin", password: "password123" },
        });
        const { data } = await login.json();

        // Use token for protected endpoint
        const orders = await request.get(`${baseURL}/orders`, {
            headers: { Authorization: data.token },
        });
        expect(orders.status()).toBe(200);
    });

    test("schema validation", async ({ request }) => {
        const response = await request.get(`${baseURL}/products`);
        const body = await response.json();

        const productSchema = {
            id: expect.any(Number),
            name: expect.any(String),
            price: expect.any(Number),
        };
        for (let product of body.data.products) {
            expect(product).toMatchObject(productSchema);
        }
    });

    test("error responses", async ({ request }) => {
        const r1 = await request.get(`${baseURL}/errors/not-found`);
        expect(r1.status()).toBe(404);

        const r2 = await request.get(`${baseURL}/errors/unauthorized`);
        expect(r2.status()).toBe(401);

        const r3 = await request.get(`${baseURL}/errors/forbidden`);
        expect(r3.status()).toBe(403);

        const r4 = await request.get(`${baseURL}/errors/server-error`);
        expect(r4.status()).toBe(500);
    });
});
```

10.4 Cypress API Testing

File: `cypress/e2e/apitests/test1PostSample.cy.js`

```
before(function () {
    Cypress.config("baseUrl", "http://localhost:5002");
});

it("Get Members", () => {
    cy.request({
        url: "/api/members",
        method: "GET",
        auth: { username: "admin", password: "admin" },
    }).its("body").then((body) => {
        expect(body).to.have.length(26);
    });
});

it("Post Member", () => {
    cy.request({
        url: "/api/members",
        method: "POST",
        auth: { username: "admin", password: "admin" },
        body: { name: "John Doe", gender: "Male" },
    }).its("body").then((body) => {
        expect(body).to.have.property("id");
    });
});
```

10.5 Local API Server

File: `qe-local-api/server.js`

The repo includes a full Express REST API for local testing practice:

Endpoint	Methods	Auth	Notes
`GET /health`	GET	No	Returns `{ status: "ok" }`
`POST /auth/login`	POST	No	Returns fake token
`GET/POST /users`	GET, POST	No	CRUD operations
`GET/PUT/DELETE /users/:id`	Various	No	By user ID
`GET/POST /products`	GET, POST	No	Product catalog
`GET/POST /orders`	GET, POST	**Yes**	Bearer token required
`GET/PUT/DELETE /orders/:id`	Various	**Yes**	By order ID
`GET /errors/unauthorized`	GET	No	Returns 401
`GET /errors/forbidden`	GET	No	Returns 403
`GET /errors/server-error`	GET	No	Returns 500

Start with: `npm run start:qe-api` (runs on port 3000)

Auth middleware (`qe-local-api/middleware/auth.middleware.js`):

```
export const authenticate = (req, res, next) => {
    const token = req.headers.authorization;
    if (!token) return res.status(401).json({ error: 'No token provided' });
    if (!token.startsWith('Bearer ')) return res.status(401).json({ error:
'Invalid format' });
    next(); // token accepted (simplified for practice)
};
```

Part 11 — Advanced Browser Interactions

11.1 Dialog / Alert Handling (Playwright)

File: `tests/alerts.spec.ts`

```
// Register handler BEFORE triggering the dialog
page.once('dialog', async (dialog) => {
    console.log(`Dialog type: ${dialog.type()}`);     // alert, confirm, prompt
    console.log(`Dialog text: ${dialog.message()}`);
    expect(dialog.message()).toContain('Please select start place.');
    await dialog.accept();     // or dialog.dismiss()
    // For prompts: await dialog.accept('input text');
});
await page.click('input#searchBtn'); // triggers the dialog
```

> **Cypress:** Alerts are auto-accepted. No explicit handler needed.

11.2 Frames / iFrames

Playwright — `tests/FramesDragAndDrop.spec.ts`:

```
const frame = page.frameLocator('iframe.demo-frame');
await frame.locator('#draggable').dragTo(frame.locator('#droppable'));
// frameLocator returns a scoped locator — interact just like regular page
```

Cypress — `cypress/e2e/practice/test3DragAndDrop.cy.js`:

```
// Cypress doesn't natively enter iframes — workaround:
cy.get('iframe.demo-frame').then($iframe => {
    const $body = $iframe.contents().find('body');
    cy.wrap($body).as('iframeBody');
});
cy.get('@iframeBody').find('#draggable')
    .trigger('mousedown', { which: 1 })
    .trigger('mousemove', { pageX: 300, pageY: 200 })
    .trigger('mouseup', { force: true });
```

11.3 Multiple Tabs / Pages (Playwright)

File: `tests/ContextAndPages.spec.ts`

```
// Click opens new tab
await page.click('text=Book Now');

// Wait for the new page event
await page.context().waitForEvent('page');

// Access all open pages in context
const pages = page.context().pages();
const newPage = pages[pages.length - 1];
await newPage.waitForLoadState();
```

```
console.log('New page title:', await newPage.title());
console.log('New page URL:', newPage.url());

// Interact with the new page
await newPage.getByRole('button', { name: 'Submit' }).click();
```

11.4 Cookies & Session Management (Playwright)

File: `tests/cookiestest.spec.ts`

```
// Extract cookies after login
const cookies = await page.context().cookies();
console.log('All cookies:', cookies);

// Find a specific cookie
const sessionCookie = cookies.find(c => c.name === 'orangehrm');

// Inject cookies into a NEW context (session reuse / parallel tabs)
const anotherContext = await browser.newContext();
await anotherContext.addCookies(cookies);
const anotherPage = await anotherContext.newPage();
await anotherPage.goto('https://...');  // already logged in!
```

11.5 Drag and Drop (Playwright)

File: `tests/FramesDragAndDrop.spec.ts`

```
await page.locator('#draggable').dragTo(page.locator('#droppable'));
```

11.6 Environment Variables

File: `tests/qe-api.spec.ts`

```
const baseURL = process.env.QE_API_URL || "http://localhost:3000";
```

```
# Set env var when running tests
QE_API_URL="http://remote-server:3000" npx playwright test

# Or use .env file
npx playwright test --env-file=.env
```

11.7 Idempotent Test Pattern (Cypress)

File: `cypress/e2e/practice/test20hrmAddEmp.cy.js`

```
// After creating an employee, delete them to leave the system clean
it('Add and cleanup employee', () => {
    // ... create employee ...
    // Navigate to employee list
    // Search for the employee
    // Delete the employee
    // Verify deletion
});
```

> **Idempotent tests** leave the system in the same state as before — critical for reliable CI.

Part 12 — Test Planning & Specifications

> 🗀 `specs/`

12.1 Test Plan Structure

File: `specs/orangehrm-employee-management.plan.md`

The repo uses structured Markdown plans that map directly to test files:

```
### 1. Authentication Tests
**Seed:** `tests/seed.spec.ts`

#### 1.1. Valid Login Flow
**File:** `tests/ohrm/authentication/valid-login.spec.ts`

**Steps:**
  1. Navigate to login page
  2. Verify login page elements (username, password, login button, logo)
  3. Enter valid credentials (Admin/admin123)
  4. Click Login button
  5. Wait for dashboard to load

**Expected Results:**
  - User authenticated and redirected to dashboard
  - Dashboard displays modules: PIM, Admin, Leave, Time, Recruitment, etc.

#### 1.2. Invalid Login Flow
**File:** `tests/ohrm/authentication/invalid-login.spec.ts`

**Steps:**
  1. Navigate to login page
  2. Enter invalid credentials
  3. Click Login

**Expected Results:**
```

```
   - Error message: "Invalid credentials"
   - User remains on login page
```

12.2 Plan-Driven Test Files

Tests reference their plan and seed:

File: `tests/ohrm/authentication/valid-login.spec.ts`

```
// spec: specs/orangehrm-employee-management.plan.md
// seed: tests/seed.spec.ts

import { test, expect } from '@playwright/test';

test.describe('Authentication Tests', () => {
    test('Valid Login Flow', async ({ page }) => {
        // Step 1: Navigate to login page
        await page.goto('https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login');

        // Step 2: Verify login page elements
        await expect(page.getByRole('textbox', { name: 'username'
})).toBeVisible();
        await expect(page.getByRole('textbox', { name: 'password'
})).toBeVisible();
        await expect(page.getByRole('button', { name: 'Login' })).toBeVisible();

        // Step 3-4: Enter credentials & login
        await page.getByPlaceholder('Username').fill('Admin');
        await page.getByPlaceholder('Password').fill('admin123');
        await page.getByRole('button', { name: 'Login' }).click();

        // Step 5: Verify dashboard
        await expect(page.getByRole('heading', { name: 'Dashboard'
})).toBeVisible();
    });
});
```

Part 13 — CI/CD Integration

13.1 GitHub Actions for Playwright

File: `.github/workflows/playwright.yml`

```
name: Playwright Tests
on:
  push:
    branches: [ main, master ]
  pull_request:
    branches: [ main, master ]

jobs:
  test:
    timeout-minutes: 60
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-node@v4
      with:
        node-version: lts/*
    - name: Install dependencies
      run: npm ci
    - name: Install Playwright Browsers
      run: npx playwright install --with-deps
    - name: Run Playwright tests
      run: npx playwright test
    - uses: actions/upload-artifact@v4
      if: ${{ !cancelled() }}
      with:
        name: playwright-report
        path: playwright-report/
        retention-days: 30
```

Key points:

- Triggers on push/PR to main/master
- `npm ci` (not `npm install`) for deterministic, clean installs
- `npx playwright install --with-deps` installs browsers AND OS dependencies
- Report artifacts uploaded even if tests fail (`!cancelled()`)
- 60-minute timeout prevents hung jobs

13.2 CI-Aware Config

```
// playwright.config.ts
forbidOnly: !!process.env.CI,       // prevent test.only from reaching CI
retries: process.env.CI ? 2 : 0,    // retry failures in CI only
workers: process.env.CI ? 1 : undefined, // sequential in CI for stability
```

13.3 Copilot Setup for CI

File: `.github/workflows/copilot-setup-steps.yml`

```
steps:
  - uses: actions/checkout@v4
  - uses: actions/setup-node@v4
    with:
        node-version: lts/*
    - run: npm ci
```

This ensures GitHub Copilot agents have the right environment in CI.

Part 14 — AI Agents for Test Automation

> 🗀 `.github/agents/`

The repo defines three GitHub Copilot agents that use the **Playwright MCP server** to interact with live browsers.

14.1 Test Planner Agent

File: `.github/agents/playwright-test-planner.agent.md`

Purpose: Explores a live web app and generates comprehensive test plans.

Workflow:

1. Uses MCP tools (`browser_navigate`, `browser_snapshot`, `browser_click`) to explore the app
2. Identifies all pages, forms, buttons, flows
3. Generates a Markdown test plan in `specs/` folder
4. Includes: scenarios, steps, expected results, and target file paths

14.2 Test Generator Agent

File: `.github/agents/playwright-test-generator.agent.md`

Purpose: Takes a plan item and generates a working Playwright test.

Workflow:

1. Reads the plan item
2. Sets up a page via MCP tools
3. Executes each step (navigate, fill, click, assert) in the real browser
4. Reads the generator log for actual locators/selectors
5. Writes the `.spec.ts` file with plan/seed references as comments

14.3 Test Healer Agent

File: `.github/agents/playwright-test-healer.agent.md`

Purpose: Debugs and fixes failing Playwright tests automatically.

Workflow:

1. `test_run` — execute failing test
2. Identify failure reason from output
3. `test_debug` — run with step-by-step inspection
4. Inspect browser snapshots at each step

5. Fix the code (locator change, timing, etc.)
6. Re-run to verify fix
7. If unfixable → mark test `test.fixme()` with explanation comment

14.4 MCP Server Configuration

File: `.vscode/mcp.json`

```
{
  "servers": {
    "playwright-test": {
      "type": "stdio",
      "command": "npx",
      "args": ["playwright", "run-test-mcp-server"]
    }
  }
}
```

The MCP server provides tools like `browser_navigate`, `browser_click`, `browser_snapshot`, `test_run`, `test_debug` that agents use to interact with real browsers.

Part 15 — Cypress vs Playwright Comparison

> Source: `cypress/e2e/practice/test1.cy.js` (extensive inline comments)

Architecture

Aspect	Cypress	Playwright
Execution	Runs **inside** the browser	Controls browser **externally** (CDP/WebDriver)
Languages	JavaScript only	JS, TS, Python, Java, C#
Browsers	Chromium, Firefox, WebKit	Chromium, Firefox, WebKit
Multi-tab	Limited (workarounds)	Native (`context.waitForEvent('page')`)
Multi-origin	Limited (`cy.origin()`)	Full support
Bundled	Mocha + Chai + Sinon + jQuery	Everything built-in

Command Style

Aspect	Cypress	Playwright
Commands	Queued, chainable, synchronous-looking	Awaited `async/await`
Retry	Built-in on `.should()`	Built-in auto-wait on actions
iFrames	Workaround / plugin	Native `page.frameLocator()`

Aspect	Cypress	Playwright
File upload	Plugin	Native `page.setInputFiles()`
Alerts	Auto-accepted	Explicit `page.on('dialog')` handler

Action Mapping

Cypress	Playwright
`cy.visit(url)`	`await page.goto(url)`
`cy.get(sel)`	`page.locator(sel)`
`cy.get(sel).click()`	`await page.locator(sel).click()`
`cy.get(sel).type(text)`	`await page.locator(sel).fill(text)`
`cy.get(sel).clear()`	`await page.locator(sel).clear()`
`cy.url().should('include', x)`	`await expect(page).toHaveURL(/x/)`
`cy.title().should('eq', x)`	`await expect(page).toHaveTitle(x)`
`cy.get(sel).should('be.visible')`	`await expect(page.locator(sel)).toBeVisible()`
`cy.get(sel).should('have.text', x)`	`await expect(page.locator(sel)).toHaveText(x)`
`cy.fixture('file')`	`import data from './file.json'`
`cy.request({...})`	`await request.get/post(...)`
`cy.intercept()`	`await page.route()`
`cy.wait('@alias')`	`await page.waitForResponse()`
`cy.wrap($el).find(sel)`	`page.locator(parent).locator(child)`
`describe() / it()`	`test.describe() / test()`
`beforeEach()`	`test.beforeEach()`

Quick Reference — Commonly Used Commands

Playwright CLI

```
npx playwright test                          # run all tests
npx playwright test tests/example.spec.ts # run specific file
npx playwright test -g "login"              # run tests matching pattern
npx playwright test --headed                # visible browser
npx playwright test --debug                 # step-through debugger
npx playwright test --ui                    # interactive UI mode
npx playwright show-report                  # open HTML report
npx playwright codegen https://example.com # record actions → code
npx playwright install                   # install browsers
npx playwright install --with-deps       # browsers + OS dependencies
```

Cypress CLI

```
npx cypress open                         # interactive GUI runner
npx cypress run                          # headless run (all specs)
npx cypress run --spec "path/to/spec"    # specific spec
npx cypress run --headed                 # visible browser
npx cypress run --browser chrome         # specific browser
npx cypress run --env key=value          # pass env variables
```

npm Scripts (This Repo)

```
npm run start:qe-api    # start local API server on port 3000
```

Environment Variables

```
# Playwright CI detection
CI=true npx playwright test

# Custom API URL
QE_API_URL="http://host:3000" npx playwright test

# With .env file
npx playwright test --env-file=.env
```

Folder Map

```
EverNorth-QE/
├── JSSamples/               → JavaScript fundamentals (18 files, 0–17)
│   └── Modules/             → ES6 module import/export
├── TSSamples/               → TypeScript basics, CSS/XPath selectors, API scripts
├── src/
│   ├── pages/               → Page Object Model classes (Playwright)
│   │   ├── loginpage.ts
│   │   ├── dashboardpage.ts
│   │   ├── employeeListPage.ts
│   │   └── addEmployeePage.ts
│   ├── data/                → Data utilities (JSON, Excel, text, DB)
│   │   ├── ReadJson.ts / ReadTxtData.ts / WriteTextData.ts
│   │   ├── ReadExcel.ts / WriteExcelData.ts / UpdateExcelData.ts
│   │   ├── ReadDataFromDB.ts / WriteDataToDB.ts / ReadDataFromDBNew.ts
│   │   ├── HandlingData.js  → parameterized test pattern
│   │   └── ohrmdata.json / userdata.json
│   ├── basetest.ts          → Custom test fixtures (POM injection via extend)
│   └── baseRequest.ts       → API request fixture with auth headers
├── tests/                   → Playwright test specs
│   ├── example.spec.ts      → Basic test structure
│   ├── orangehrm.spec.ts    → Full OrangeHRM flow (no POM)
│   ├── orangehrmdt.spec.ts  → Data-driven test with JSON import
│   ├── OhrmPOMtest.spec.ts  → POM with direct instantiation
│   ├── OhrmPOMtest-custom.spec.ts → POM with custom fixtures
│   ├── alerts.spec.ts       → Dialog/alert handling
│   ├── apsrtc.spec.ts       → Complex locators, filtering, lists
│   ├── tsrtc.spec.ts        → Built-in locators (getByRole, etc.)
│   ├── ContextAndPages.spec.ts → Multi-tab handling
│   ├── FramesDragAndDrop.spec.ts → iFrames, drag and drop
│   ├── cookiestest.spec.ts  → Cookie extraction/injection
│   ├── pwTestExamples.spec.ts → Hooks (before/after each/all)
│   ├── qe-api.spec.ts       → Full API test suite (CRUD, auth, schema)
│   ├── seed.spec.ts         → Prerequisite validation
│   └── ohrm/                → Plan-driven generated tests
│       ├── authentication/  → Login tests (valid, invalid, empty)
│       ├── employee-management/ → Create employee happy path
│       └── session-management/  → Logout test
├── cypress/
│   ├── e2e/
│   │   ├── 1-getting-started/todo.cy.js → Basic Cypress test
│   │   ├── practice/        → OrangeHRM tests, fixtures, within vs then
│   │   ├── apitests/        → Cypress API testing
│   │   └── 2-advanced-examples/ → Actions, assertions, aliasing,
│   │                            cookies, network, spies, stubs, etc.
│   ├── fixtures/            → Test data JSON files
│   └── support/             → Custom commands, e2e setup
├── qe-local-api/            → Express REST API server for testing
│   ├── server.js            → Main server (port 3000)
│   ├── routes/              → auth, users, products, orders
│   ├── middleware/          → Bearer token auth
│   └── data/               → In-memory data stores
├── specs/                   → Test plans in Markdown
│   └── orangehrm-employee-management.plan.md
├── .github/
│   ├── agents/              → Copilot agents (planner, generator, healer)
│   └── workflows/           → GitHub Actions CI pipelines
├── .vscode/mcp.json         → MCP server config for Playwright
├── playwright.config.ts     → Playwright configuration
├── cypress.config.ts        → Cypress configuration
├── tsconfig.json            → TypeScript configuration
├── package.json             → Dependencies and scripts
└── explore-orangehrm.ts     → Locator exploration script
```

Generated from full recursive scan of the EverNorth-QE repository.