End-to-End Testing a Node App with Playwright

Same registration/login app as before, rebuilt as a frontend-only app with Vite and plain JavaScript. You'll run it, check it by hand, then automate two checks with Playwright.

---

1. The project

```
node-auth-app/
├── index.html            # the single page
├── src/
│   ├── auth.js           # logic: validation, hashing, user store
│   ├── main.js           # UI: renders pages, handles forms
│   └── style.css
├── tests/
│   └── auth.spec.js      # 2 Playwright tests
├── playwright.config.js
└── package.json
```

- No backend, no database. Users live in a JavaScript `Map` inside the browser tab.
- Pages are switched with the URL hash (`#/login`, `#/register`, `#/welcome`). This never reloads the page, so the data survives navigation.
- Refreshing the page wipes all users. That's expected.

---

2. Install Node.js

Install the LTS version (v26 at the time of writing; anything from v22.12 works). `npm` comes with it.

OS	Option
Any	Installer from https://nodejs.org → *LTS*
macOS	`brew install node`
Windows	`winget install OpenJS.NodeJS.LTS`
Linux	`nvm` (below), or the packages listed at https://nodejs.org/en/download

Check it, in a new terminal:

```
node -v      # v22.12 or higher
npm -v
```

---

3. Run the app

```
cd node-auth-app
npm install          # installs Vite and Playwright into node_modules/
npm run dev
```

Open http://localhost:5173. Stop the server with `Ctrl+C`.

---

4. Confirm it works

Go through this checklist by hand. Each row is something we could automate.

#	Do this	Expected
1	*Create an account* with `alice` / `abc`	Red: *Password must be at least 8 characters*
2	Register `alice` / `secret123`	Green: *Account created for alice*, now on login
3	Register `alice` again	Red: *Username already taken*
4	Log in with `alice` / `wrong-pass1`	Red: *Invalid username or password*
5	Log in with `alice` / `secret123`	*Welcome, alice!*
6	Click *Log out*, then go to `http://localhost:5173/#/welcome`	Sent back to login
7	Refresh the page, log in as `alice`	Fails: data was only in memory

Checking all this by hand after every change is slow. Playwright does it for us.

---

5. Install Playwright

Playwright drives a real browser from code: it clicks, types and checks what's on screen. These are end-to-end (E2E) tests: they test the app the way a user sees it.

`npm install` already installed the Playwright library (`@playwright/test` in `package.json`). You only need the browser:

```
npx playwright install chromium
```

> Starting a project from scratch? `npm init playwright@latest` sets everything up interactively.

The config: `playwright.config.js`

```
export default defineConfig({
  testDir: "./tests",
```

```
  reporter: [["list"], ["html", { open: "never" }]],
  use: {
    baseURL: "http://localhost:5173",
    trace: "retain-on-failure",
  },
  projects: [{ name: "chromium", use: { ...devices["Desktop Chrome"] } }],
  webServer: {
    command: "npm run dev",
    url: "http://localhost:5173",
    reuseExistingServer: true,
  },
});
```

- `reporter`: print results in the terminal and save an HTML report.
- `baseURL`: tests can write `page.goto("/#/login")` instead of the full address.
- `webServer`: Playwright starts the app for you before testing, and stops it after. No second terminal needed.
- `trace`: when a test fails, Playwright keeps a step-by-step recording to debug it.

---

6. The tests: `tests/auth.spec.js`

A helper

```
async function submitForm(page, username, password, button) {
  await page.getByLabel("Username").fill(username);
  await page.getByLabel("Password").fill(password);
  await page.getByRole("button", { name: button }).click();
}
```

- `page` is a browser tab, given to every test by Playwright.
- Locators find elements the way a user would: by label text (`getByLabel`) or by role and visible name (`getByRole`). No CSS classes or IDs, so the tests survive styling changes.
- Every browser action is `async`, so always `await` it.

Shared setup

```
test.beforeEach(async ({ page }) => {
  await page.goto("/#/register");
  await submitForm(page, "alice", "secret123", "Register");
  await expect(page.getByRole("alert")).toHaveText("Account created for
alice. Please log in.");
});
```

- Each test gets a fresh tab, so the user store starts empty. Tests can't affect each other.
- That's why each test needs `alice` registered first. `beforeEach` does it once for both tests.

- `expect(...).toHaveText(...)` waits (up to 5 s) until it's true. No `sleep` needed.

Test 1: happy path

```
test("registered user can log in", async ({ page }) => {
  await submitForm(page, "alice", "secret123", "Log in");

  await expect(page).toHaveURL(/#\/welcome$/);
  await expect(page.getByRole("heading", { name: "Welcome, alice!"
})).toBeVisible();
});
```

Checks both *where* we are (URL) and *what* the user sees (heading).

Test 2: sad path

```
test("login with invalid credentials shows an error", async ({ page }) => {
  await submitForm(page, "alice", "wrong-pass1", "Log in");

  await expect(page.getByRole("alert")).toHaveText("Invalid username or
password");
  await expect(page).toHaveURL(/#\/login$/);
  await expect(page.getByRole("heading", { name: "Welcome"
})).toBeHidden();
});
```

A sad-path test checks **two things**: the error is shown, *and* nothing it shouldn't do happened (no welcome page).

> `getByRole("alert")` works because the message box in `main.js` has `role="alert"`. Accessible markup makes apps easier to test, too.

---

7. Run the tests

```
npx playwright test               # headless, prints results
npx playwright test --headed      # watch the browser do it
npx playwright test --ui          # interactive mode: step through, time-
travel
npx playwright show-report        # open the HTML report
```

Expected:

```
  ✓  1 [chromium] › tests/auth.spec.js › registered user can log in
  ✓  2 [chromium] › tests/auth.spec.js › login with invalid credentials
```

```
shows an error

  2 passed
```

---

8. Exercises

1. Break it on purpose. In `src/auth.js`, make `login()` return `stored !== undefined` (skip the password check). Which test fails? Why only that one? Revert.
2. Unknown user. Add a sad-path test: log in as `bob`, who never registered.
3. Validation. Add a test for row 1 of the checklist (password too short).
4. Logout. Add a test for row 6: after logging out, `/#/welcome` redirects to login.