Home Assignment: Test Developer – Playwright

Test the “Create Transport Request” flow with Playwright E2E tests (stage environment)

Create a small Playwright test suite that validates the end-to-end flow of creating a Transport Request in our stage app. Show how you think about happy paths, edge cases, and negative scenarios. The goal of testing is to find bugs; there is no known trap in this flow.

Scope

1. Register and work only in stage:
   https://stage.4shipper.transportly.eu/register
2. Automate the Create Transport Request flow:
   - Happy path: create a valid request end-to-end and verify meaningful assertions (UI state, success messages, data presence).
   - Negative cases: demonstrate thoughtful validations (e.g., required fields, invalid formats, out-of-range values, impossible dates, missing pickup/delivery, etc.).
   - Usability/UX guards (choose 1–2): disable/enable of buttons, error message consistency, prevention of duplicate submit, navigation back/refresh resilience.
3. Keep tests isolated and repeatable:
   - Use your own test account(s).
   - Clean up any entities you create or use unique identifiers to avoid collisions.

Technical constraints

- Use Playwright Test with Node.js and NPM only. No extra services should be required to run.
- Configure baseURL to the stage site. Do not target production.
- Prefer TypeScript and standard Playwright project structure.
- Do not hardcode secrets in code; if you must use env variables, include .env.example and reference via process.env.

Deliverables

- A runnable Playwright project with tests that cover:

1. Happy path for creating a Transport Request
2. A negative scenario you consider important
3. A resilience/usability check (see Scope, item 2)
   - README.md with short run instructions (keep it brief).
   - Optional: Playwright trace on failure enabled; meaningful screenshots on failure.

Submission

- Zip file with the project, or a public GitHub repository (earns bonus points).
- Do not include test result artifacts in your submission (node_modules excluded, of course).
- Be mindful of the server: do not flood stage with parallel requests or excessive loops.

Evaluation criteria

- Correctness and reliability of tests (stable, deterministic, repeatable)
- Coverage depth: meaningful assertions, negative thinking, and edge-case awareness
- Code quality: structure, naming, locators, reuse, readability
- Developer experience: ease of setup and running, clarity of README
- Thoughtfulness: small touches like unique test data, cleanup, trace usage

Prerequisities

- Node.js, NPM

Useful commands

npx playwright test Runs all tests. Headless by default.
npx playwright test --headed Runs all tests in headded mode to see what is going on live during the tests execution.
npx playwright test landing-page.spec.ts Runs the specified tests.
npx playwright show-trace test-results/failed-setup-trace.zip Opens the specified trace log.

Notes and hints

- You can use data-testid/test-ids if available; otherwise pick robust locators (role, label, text) over brittle CSS.
- Prefer explicit assertions that prove business value (not just “element visible”).
- If you find a bug, document it briefly in README (step to reproduce + expected vs. actual).
- If Node.js/NPM setup blocks you, ping us; we can provide a blank scaffold. That’s a minus, but great tests can outweigh it.