**File: e2e/todo-flow.spec.js**

javascript

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

test.describe('Todo Flow E2E Test', () => {

test('Logs in, creates 3 todos, deletes one, verifies deletion and logs out', async ({ page }) => {

// --- Login ---

// Navigate to the login page

await page.goto('http://localhost:3000/login');

// Fill in the fake login form

// (Assuming the login form uses name attributes on the input fields)

await page.fill('input[name="username"]', 'testuser');

await page.fill('input[name="password"]', 'password123');

// Click the login button (assumed to be a submit button)

await page.click('button[type="submit"]');

// Wait for navigation and confirm successful login by checking the URL or an element

await page.waitForURL('http://localhost:3000/');

await expect(page.locator('h1')).toHaveText(/todos/i); // Assuming a header exists

// --- Add 3 Todos ---

// Define the list of todos to add

const todos = ['Todo 1', 'Todo 2', 'Todo 3'];

for (const todo of todos) {

// Fill in the todo input field (assumed to have an id "new-todo")

await page.fill('#new-todo', todo);

// Click on the "Add" button (assumed to have an id "add-todo")

await page.click('#add-todo');

// Use Playwright's auto-waiting: verify the todo item appears in the list as an <li> with class "todo-item"

await expect(page.locator('li.todo-item', { hasText: todo })).toBeVisible({ timeout: 5000 });

}

// --- Delete One Todo ---

// For example, delete "Todo 2"

// Assumes that each todo item has a delete button inside with a class "delete-btn"

const todoToDelete = 'Todo 2';

await page.click(`li.todo-item:has-text("${todoToDelete}") button.delete-btn`);

// Verify that "Todo 2" no longer appears in the list (using a retry-able expect)

await expect(page.locator('li.todo-item', { hasText: todoToDelete })).toHaveCount(0, { timeout: 5000 });

// Optionally, verify that any filter or count element reflects that we now only have 2 todos

// Assumes there is an element with the class "todo-count" showing the number of active todos

const countText = await page.locator('.todo-count').innerText();

expect(Number(countText)).toBe(2);

// --- Logout ---

// Click the logout button (assumed to have an id "logout")

await page.click('#logout');

// Verify that the user is taken back to the login page

await expect(page).toHaveURL('http://localhost:3000/login');

});

});

**File: e2e/README.md**

markdown

# E2E Test + Retry Logic: Explanation

### Handling Flakiness

- \*\*Auto-Waiting & Explicit Assertions:\*\*

We rely on Playwright’s built-in auto-waiting mechanisms along with explicit `expect` assertions (e.g., `toBeVisible`, `toHaveCount`). These checks wait for the element states to meet the expected conditions before moving forward.

- \*\*Retries in Configuration:\*\*

In our Playwright configuration (typically in `playwright.config.ts`), we set `retries` (e.g., `retries: 2`) so that transient failures—often due to network delays or flakey UI updates—are automatically retried. This reduces false negatives in CI environments.

### Reporting Test Failures

- \*\*Detailed Reports:\*\*

On failure, Playwright gathers detailed logs, screenshots, and traces. These artifacts are automatically attached to test reports.

- \*\*Notifications:\*\*

In a CI setup, failures would be integrated with our reporting tools—sending notifications via Slack or email. This ensures quick awareness and review of any issues.

### Integration into CI

- \*\*Automated Testing Pipeline:\*\*

The E2E tests run as part of our Continuous Integration pipeline (e.g., via GitHub Actions or another CI tool), ensuring that every pull request and commit triggers a full end-to-end verification. This helps prevent regressions in critical user flows before production deployment.

- \*\*Blocking Deployment on Failure:\*\*

Since these tests simulate real user interactions (login, todo creation, deletion, logout), a failure in any of these paths will halt the deployment process, ensuring that only stable features reach production.

These examples showcase how our E2E test validates crucial user flows while incorporating retry logic to minimize flaky results. The README further explains our strategy on flakiness, failure reporting, and CI integration, ensuring robust automation for our demo app.