Neuro Event Labs Oy

Validation of Basic Form Submission

Pavithra Subramaniyam

12-27-2025

Contents

1. Objective

The objective of this assignment is to design and implement automated UI tests based on the provided user stories using Playwright with Typescript. The focus is on validating expected user-interface behavior and demonstrating test-case validation rather than validating the demo application.

2. Interpretation of User Stories

User stories were treated as the single source of truth. No additional validation rules were assumed beyond these statements. Few requirements like name and email validations were deliberately excluded from tests.

2.1. General rule (all scenarios)

- Validation is checked after submission is attempted
- “Form should not be submitted” means:
  - no success confirmation
  - no navigation/progression

2.2. Scenario 1: Successful submission with all required fields filled

- User provides valid input for all mandatory fields
- User clicks Submit
- System accepts the input

Validation

- Submission success message is visible (confirmation)
- No validation errors are shown

2.3. Scenario 2: Submission fails when required fields are empty

- User does not enter any values
- User clicks Submit
- System must prevent successful submission
- Validation feedback is shown for required fields
- Submission does not succeed

Validation

- Validation feedback is shown for required fields (no explicit mention of the ‘feedback message’)
- Submission does not succeed (no explicit mention of the ‘feedback message’)

2.4. Scenario 3: Invalid email address is rejected

- User enters valid First Name and Last Name
- User enters an invalid email
- User tries to click Submit

Validation

- Email-specific validation error is shown
- Submission does not succeed (no explicit mention of the 'feedback message')

Some feedback messages already implemented in the application are directly used in the test case expected outputs. For unknown messages *basic feedback messages* are assumed in the expected output, as focus is on meeting User story scenarios as validation features are absent. Still for the clearly visible implementation gaps, few more scenarios were added in the appendix, on my own interest.

3. Scope and rationale

Each test validates a single behavior to ensure clear failure reasons. Both positive and negative paths were covered as required by the user stories.

- Included:
  - Successful form submission with valid data
  - Required field validation behavior
  - Invalid email format handling
- Excluded:
  - Name field character or length validation
  - Email domain existence validation
  - UI styling or visual checks

These exclusions were intentional due to lack of explicit requirements from the user stories. Functional helper methods were used instead of Page Object Model to reduce complexity while eliminating repetitive code.

3.1. Locators

- Role-based locators (getByRole()) for interactive elements such as buttons and links
- Label-based locators (getByLabel()) for form controls where labels are semantically associated with inputs
- Text-based locators (getByText()) for locating elements that contain given text

In certain cases, semantic locators were not applicable due to the structure of the demo application. In such cases, CSS-based locators were used (e.g. survey root container).

3.2. Assertions

Playwright includes test assertions in the form of expect function. To make an assertion, call expect(value) and choose a matcher that reflects the expectation. The following Playwright assertions are used in this assignment:

- toBeVisible()
- not.toBeVisible()
- toBeDisabled()
- toBeChecked()
- not.toBeChecked()

- toHaveCount()

3.3. Determinism

The tests are deterministic because they rely on Playwright's auto-waiting and UI state assertions rather than fixed delays or timing assumptions.

Negative assertions in Playwright don't wait for async UI updates, so they can pass even when the UI changes later. Without an explicit failure signal, the only deterministic approach is to assert expected validation and let the test fail.

3.4. Consistency

All tests in this assignment follow a consistent structure and style. The same test flow, naming conventions, helper and assertion usage are used throughout the suite to keep the code easy to read and maintain.

3.5. Browser specification

Added navigation to simple form instead of visiting the URL directly to increase reusability (for other forms). There is no explicit mention of browser specification, so tests are made to run on all three browsers by default.

4. Results of test scenarios

Some tests are expected to fail because the demo application have not fully implemented the behaviors described in the user stories. These failures highlights identification of missing functionality using the automated testing implementation rather than issues with the application. Application under test: https://formio.github.io/angular-demo/#/

4.1. Scenario – 1

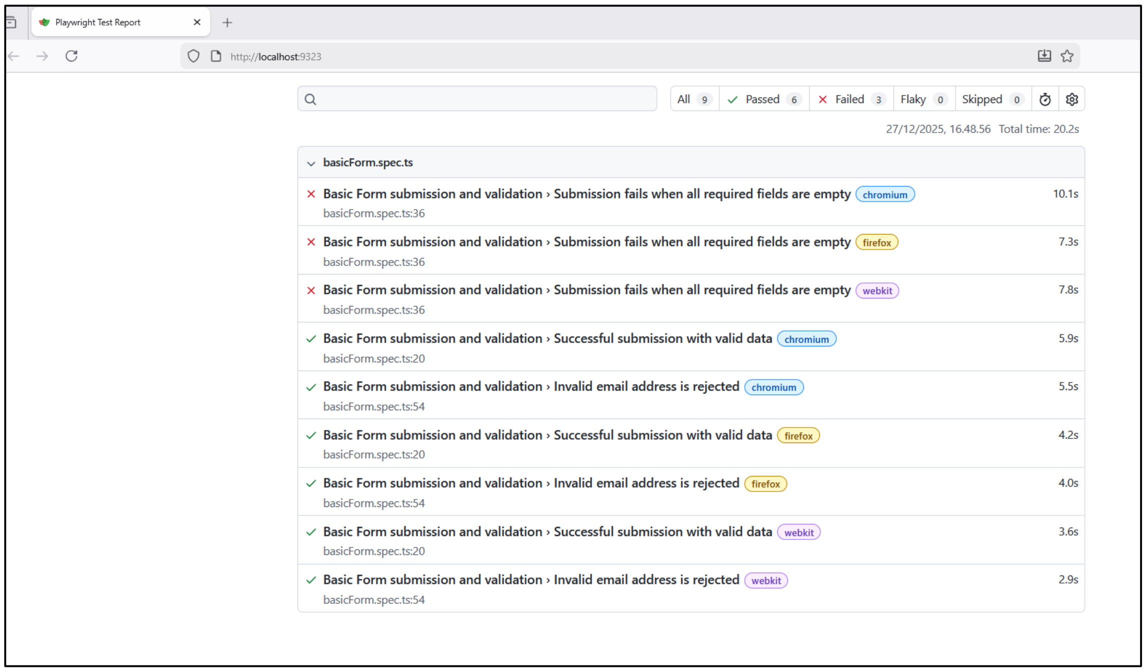
SCENARIO	Successful submission with all required fields filled	
1	Script ID	TID-01
2	Objective	When all the required fields are filled, the form should be submitted successfully without any validation error messages
3	Preconditions	
	1	User has access to a supported browser
4	Test Data	
	1	Username: Pavithra
	2	Password: Subramaniyam
	3	Email: example@example.com
5	Script Steps	
	1	Open the Form.io demo application
	2	Navigate to the **Simple Form**
	3	Enter a valid value in the **First Name** field
	4	Enter a valid value in the **Last Name** field
	5	Enter a valid email address in the **Email** field
	6	Click the **Submit** button
6	Expected Results	
	1	The form should be submitted successfully
	2	Success message should be displayed (confirmation)
	3	No validation error messages should be visible
7	Status	Passed
8	Comments	Submission complete message displayed

4.2. Scenario – 2

SCENARIO	Submission fails when required fields are empty		
1	Script ID	TID-02	
2	Objective	When the user submits the form without filling any required fields, validation error messages should be displayed and the form should not be submitted.	
3	Preconditions		
	1	User has access to a supported browser	
4	Test Data		
	1	No data required to be filled	
5	Script Steps		
	1	Open the Form.io demo application	
	2	Navigate to the **Simple Form**	
	3	Click the **Submit** button without entering any values	
6	Expected Results		
	1	Validation error messages should be displayed for required fields	
	2	The form should not be submitted (no success message at least)	
7	Status	Failed	
8	Comments	Validation not enforced in demo application, so basic error message of “invalid or required” is expected and fails.	

4.3. Scenario - 3

SCENARIO	Invalid email address is rejected	
1	Script ID	TID-03
2	Objective	When an invalid email address is entered, a validation error message should be displayed and the form should not be submitted.
3	Preconditions	
	1	User has access to a supported browser
4	Test Data	
	1	Username: Pavithra
	2	Password: Subramaniyam
	3	Email: invalid-email
5	Script Steps	
	1	Open the Form.io demo application
	2	Navigate to the **Simple Form**
	3	Enter a valid value in the **First Name** field
	4	Enter a valid value in the **Last Name** field
	5	Enter an invalid email address in the **Email** field
	6	Click the **Submit** button
6	Expected Results	
	1	Email validation error message should be displayed
	2	The form should not be submitted
7	Status	Passed
8	Comments	Submit button disabled, no proceeding for validation of form submission


Playwright Test Report

http://localhost:9323

All 9 ✓ Passed 6 ✗ Failed 3 Flaky 0 Skipped 0

27/12/2025, 16.48.56 Total time: 20.2s

```
basicForm.spec.ts
✗ Basic Form submission and validation › Submission fails when all required fields are empty  chromium   10.1s
  basicForm.spec.ts:36
✗ Basic Form submission and validation › Submission fails when all required fields are empty  firefox    7.3s
  basicForm.spec.ts:36
✗ Basic Form submission and validation › Submission fails when all required fields are empty  webkit     7.8s
  basicForm.spec.ts:36
✓ Basic Form submission and validation › Successful submission with valid data  chromium   5.9s
  basicForm.spec.ts:20
✓ Basic Form submission and validation › Invalid email address is rejected  chromium   5.5s
  basicForm.spec.ts:54
✓ Basic Form submission and validation › Successful submission with valid data  firefox   4.2s
  basicForm.spec.ts:20
✓ Basic Form submission and validation › Invalid email address is rejected  firefox   4.0s
  basicForm.spec.ts:54
✓ Basic Form submission and validation › Successful submission with valid data  webkit   3.6s
  basicForm.spec.ts:20
✓ Basic Form submission and validation › Invalid email address is rejected  webkit   2.9s
  basicForm.spec.ts:54
```

Figure 1. Test report of user story based test cases

5. How to Run

Click on : https://github.com/spavythra/Playwright_practice

I have two folders

- BasicForm_strict is the user requirement story
- BasicForm_extended is my addition

```
cd .\BasicForm_strict
npm install
npx playwright test
```

To open UI mode, run the following command in your terminal:

```
npx playwright test --ui
```

6. Summary

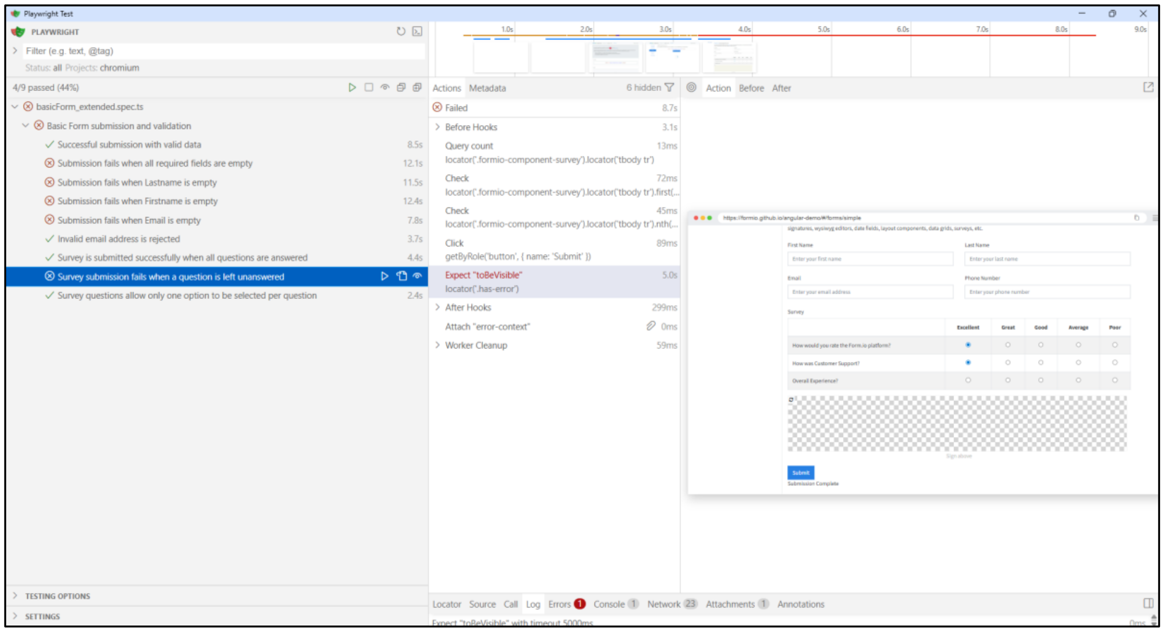
This assignment demonstrates a requirement-driven testing approach with the given test application. The emphasis was on implementing all user stories, using sensible locators, deterministic test, meaningful assertion and consistent code, furthermore extra stories are added in Appendix based on my interest.

Appendix I - Exploratory scenarios

In addition to the assignment required tests, a small set of extended tests are included to express my interest and product understanding. These tests are clearly separated from user-story-driven tests (assignment needs) and derived based on reasonable UI observations (e.g., who needs a form without survey answers? May be anonymous forms with survey answers are okay). Code is available in GitHub and README also explains the details.

Derived test suite

- Submission is unsuccessful when First Name is missing
- Submission is unsuccessful when Last Name missing
- Submission is unsuccessful when Email is missing
- Submission is successful when all the survey questions are answered
- Survey questions allow only one option to be selected per question
- Submission is unsuccessful when one or more survey questions are left unanswered



```
4/9 passed (44%)
basicForm_extended.spec.ts
  Basic Form submission and validation
    ✓ Successful submission with valid data                              8.5s
    ⊗ Submission fails when all required fields are empty               12.1s
    ⊗ Submission fails when Lastname is empty                           11.5s
    ⊗ Submission fails when Firstname is empty                          12.4s
    ⊗ Submission fails when Email is empty                               7.8s
    ✓ Invalid email address is rejected                                  3.7s
    ✓ Survey is submitted successfully when all questions are answered   4.4s
    ⊗ Survey submission fails when a question is left unanswered
    ✓ Survey questions allow only one option to be selected per question 2.4s
```

```
Failed                                                              8.7s
> Before Hooks                                                      3.1s
  Query count                                                       13ms
  locator('.formio-component-survey').locator('tbody tr')
  Check                                                             72ms
  locator('.formio-component-survey').locator('tbody tr').first(...
  Check                                                             45ms
  locator('.formio-component-survey').locator('tbody tr').nth(...
  Click                                                             89ms
  getByRole('button', { name: 'Submit' })
  Expect "toBeVisible"                                              5.0s
  locator('.has-error')
> After Hooks                                                      299ms
  Attach "error-context"                                             0ms
> Worker Cleanup                                                    59ms
```

```
Expect "toBeVisible" with timeout 5000ms
```

Figure 2. Test execution UI for extended test case showing the results