***Suggested Test Case Listing*:**

1. Verify the form allows record creation when all inputs are valid.
2. To Verify an error is shown and submission is blocked if required fields are missing.
3. Check whether an invalid email format is rejected by the form.
4. Verify duplicate entries for unique fields result in a failed record creation.
5. Verify the reset button clears all input fields.
6. To Verify an existing record can be modified successfully.
7. Check whether an existing record can be removed from the system.

------------------------------------------------------------------------------------------------------------- **Version 1: Straightforward Setup**

**Install Node.js & Project Setup**

Ensure Node.js (v14+) is installed.

Create or open your existing Node.js project folder (this will be the root of your web application).

**Install Playwright**

In your project directory, run:

npm install --save-dev @playwright/test

This adds Playwright’s testing framework and browsers to your dev dependencies.

**Initialize Playwright Config**

(Optional) Run:

npx playwright init

This command generates a basic playwright.config.js and example tests folder.

**Integrate with “Testing Engine”**

In your web app’s Testing Engine module, create a new function or API endpoint (e.g., /run-tests).

That function should execute:

npx playwright test

You can trigger this either manually from the UI (a “Run Tests” button) or automatically on a schedule or when new code is pushed.

**View Test Results & Reports**

By default, Playwright can generate HTML reports.

Expose or embed the HTML report within your Testing Engine UI so team members can see pass/fail summaries and details of each test run.

**Version 2: Emphasis on Configuration & CI/CD**

**Project Environment**

Install Node.js if not already available.

Go to your web application’s root folder and ensure you have a package.json file.

**Install & Configure Playwright**

Run:

npm install --save-dev @playwright/test

Create or modify playwright.config.js:

module.exports = {

testDir: 'tests',

use: { headless: true },

reporter: [ ['list'], ['html', { outputFolder: 'playwright-report' }] ]

};

**Create Test Suites**

Inside tests/, add your Playwright test files (.spec.js or .test.js).

These tests can reference the routes/pages in your web application.

**Tie into Your Testing Engine**

In your Testing Engine module, add logic to run the Playwright CLI:

npx playwright test --config=playwright.config.js

This can happen either via an internal node script or a route that the Testing Engine calls.

**Set Up CI/CD Integration**

In GitHub Actions, GitLab CI, Jenkins, etc., add steps to:

Install dependencies (npm ci or npm install).

Run tests using npx playwright test.

Publish or store the generated report so the Testing Engine can link to it or display it.

**Version 3: Detailed Technical Breakdown**

**System Prerequisites**

Node.js: Install at least v14.

NPM/Yarn: Ensure your package manager is functional.

Initialize the Web App Project

If you haven’t already, run:

npm init -y

This creates a package.json to track dependencies.

**Add Playwright**

Execute:

npm install --save-dev @playwright/test

Optionally install browsers (usually automated by Playwright).

**Create the Playwright Configuration**

Manually create playwright.config.js or use:

npx playwright init

Configure default browser, test directory, base URL, etc.

**Integrate into Testing Engine**

Inside your Testing Engine code, create a function (e.g., runPlaywrightTests()) that does:

const { exec } = require('child\_process');

exec('npx playwright test', (error, stdout, stderr) => {

// Handle output

});

Provide a UI button or endpoint in your Testing Engine that calls runPlaywrightTests().

Present test results: capture the HTML report or parse the JSON output to display within your web app.

---------------------------------------------------------------------------------------------------------------------------------