**This is the response of the exercise found here :** <https://github.com/igorstojakovic/qa_assessment_app?tab=readme-ov-file>

**Selenium framework:**

**Link to my framework : https://github.com/hmir992/cucumber\_frameworkV3/tree/main/POM-cucumber-selenium-framework**

**Playwright tests :**

[**https://github.com/hmir992/cucumber\_frameworkV3/tree/main/playright%20tests/tests**](https://github.com/hmir992/cucumber_frameworkV3/tree/main/playright%20tests/tests)

**Original Application, with updated unit tests in it.**

**https://github.com/hmir992/cucumber\_frameworkV3/tree/main/qa\_assessment\_app-main%20(1)**

**Question**

**Feature**

As a product owner I would like to have an easy and lightweight app that calculates amount of an additional charge per customers car return.  
Cars should return with the full tank.  
Discrepancy allowed is 5%.

e.g. Car returns with "4/8" of the tank, telematics report 44% - we need to charge customer X amount.

e.g. Car returns with "8/8" of the tank, telematics report 95% - we will not charge the customer.

**1.1 Read the app documentation (this readme), and based on "Feature" section check:**

**Is the story & feature well-defined?**

The "Feature" part is clear however the description is basic, all it tells us is that the application calculates an extra charge if the customer returns a car with a fuel not at full capacity (100% full tank) based on the telematics report.

Discrepancy allowed is 5% of the fuel of the returned car

e.g. Car returns with "8/8" of the tank, telematics report 95% - we will not charge the customer.

e.g. Car returns with "4/8" of the tank, telematics report 44% - we need to charge customer X amount.

**What questions would you ask to make it clearer, if any?**

Based on just reading the feature alone, there are still many questions I would ask if this was a story sizing session.

Given we have to build this feature from scratch.

Some questions I would be asking in sizing.

**Pricing**

* + What is the exact way we are doing the calculation of the telematics, what’s the formula?
  + Does the system account for the charge only based on missing fuel (discrepancy)?
  + Does the system account for fuel price per litre or location-based charges?
  + Is there a flat fee in addition to the calculated cost of discrepancy eg taxes added to the cost.
  + What is the currency of the system and how do we handle different currencies? Euro vs Dollar

**Data handling:**

* How will the system react if the telematics data is wrong? For example Telematics data shows less than 100% tank but the user actually returned a full tank.
* How will the system handle fractions of a number like 4.85% will we round up or down?
* What happens if the system returns inconsistent data? So different prices for different users
* What if the users of the system are overcharged, how will they get a refund?
* Should we be able override the telematics reading manually? In the case of when the system is not working or incorrect data or the fuel size is unknown.

**Legal**

* Do we need to store a record of discrepancies and charges for auditing purposes or disputes?

**UI and Security:**

* + the UI should show the breakdown of the cost of the charge
  + The UI should to send notifications receipt via sms or email
  + There should be a admin section and users section in the system, this is to allow for admin tasks like overriding the telematics reading in case of error
  + Is multiple users supported in the system with different fuel rates?

**1.2 Use the app in exploratory fashion and check:**

**Does the app work as designed?**

In Core Functionality:

* Yes the system allows a user to login and input Fuel on Return (in eighths) and Fuel Price per Litre, and press calculate to do the calculation
* However, I do see a few improvements that can be made:
  + Looking at the calculation logic this needs to redefined to 5% discrepancy (currently, set to 3% discrepancy, this a bug in the current system)
  + There is currently no restriction on how the values are inputted, for example I could input a really large number eg 1000 for both Fuel on Return (in eighths) and Fuel Price per Litre and press calculate and it would over charge the customer, there should be a restriction around this.
  + Similarly, I can input negative values for both Fuel on Return (in eighths) and Fuel Price per Litre and press calculate, this would result in an incorrect reading being outputted
  + There is no error handling at present, I can input no values for both Fuel on Return (in eighths) and Fuel Price per Litre and press calculate and it doesn’t stop me from doing so, there should be a good error message to inform the user

**Are there any bugs?**

**If so write a bug report on them**

Bug 1: Incorrect Discrepancy Calculation

Description: Fuel discrepancy threshold is incorrectly set to 3% instead of 5%

How did I find this bug? I played around with the system, checked the source code and ran the unit tests to determine this was an issue with the system.

Steps to Reproduce:

1. Input car data into the system with a fuel that has a discrepancy between 3% and 5%
2. Click calculate and see that the system incorrectly applies a charge when it should allow up to 5% discrepancy without charging.

Expected Result: The system should only charge if the discrepancy is greater than 5%  
Actual Result: The system without the bug fix currently charges the customer if the discrepancy is greater than 3%.

Bug 2: System returns Negative Fuel Cost Calculations for Discrepancy

How did I find this bug? I played around with the system and ran the unit tests to determine this was an issue with the system.

Description: System allows for negative fuel cost calculations

Steps to Reproduce

1. Enter a fuel return value eg 8
2. Enter a Fuel Price per Litre eg 1 (for 1 euro I assume)
3. Click Calculate and see the cost of the fuel

Expected Result: The system should return a value of 0 for this calculation  
Actual Result: The system returns a negative calculation.

Bug 3: calculatedCost is Null

How did I find this bug? It was not an obvious bug, I had a look at the unit tests and this made me aware of this issue with the system

Description: calculatedCost sometimes remains null due to async state update issues

Steps to Reproduce:

1. Enter fuel details and click Calculate.
2. Check the console logs (console.log('Calculation Data:', { telematicsFuel, calculatedCost })).
3. Note that calculatedCost sometimes logs as null despite calculations being performed.

Expected Result: calculatedCost should always contain a valid number after calculation, check the logs.  
Actual Result: calculatedCost is null, check the logs

**1.3 Check unit test execution and coverage**

**Do you find these unit tests done right?**

I find the unit tests to be done correctly, it covers the basic aspects of the tests.

1. **renders calculate form**
2. **calculates the fuel cost**
3. **disables OK button until Calculate is clicked**
4. **generates new car data when OK is clicked**

**How would you improve the existing tests? And Some tests are failing; can you identify why?**

**(Note I also added this to the page.test.tsx in the project as well if you want to run it.)**

1. Explanation : I would improve the existing test, 'generates new car data when OK is clicked' to something like the below

this is a better way of doing the assertion, It asserts that the input fields are empty by checking .value === ''

it('generates new car data when OK is clicked, v2', async () => {  
 render(<CalculatePage />);  
  
 fireEvent.change(screen.getByLabelText(/Fuel on Return \(in eighths\)/i), { target: { value: '3' } });  
 fireEvent.change(screen.getByLabelText(/Fuel Price per Litre/i), { target: { value: '1.5' } });  
  
 fireEvent.click(screen.getByRole('button', { name: /Calculate/i }));  
 const okButton = screen.getByRole('button', { name: /OK/i });  
 expect(okButton).not.toBeDisabled();  
 fireEvent.click(okButton);  
  
 // Wait for the form inputs to be cleared  
 await waitFor(() => {  
 // Explicitly cast to HTMLInputElement to avoid TypeScript error

// when accessing the value property of an input field.  
 const fuelOnReturnInput = screen.getByLabelText(/Fuel on Return \(in eighths\)/i) as HTMLInputElement;  
 const fuelPriceInput = screen.getByLabelText(/Fuel Price per Litre/i) as HTMLInputElement;  
  
 // this is a better way of doing the assertion, It asserts that the input fields are empty by checking .value === ''  
 // Check that both the "Fuel on Return" and "Fuel Price" fields are empty  
 expect(fuelOnReturnInput.value).toBe('');  
 expect(fuelPriceInput.value).toBe('');

1. Explanation: Another way we can have fix the existing tests is by instead of checking input values, check if new car data (VIN) is generated.

it('generates new car data when OK is clicked v3', async () => {  
 render(<CalculatePage />);  
  
 fireEvent.change(screen.getByLabelText(/Fuel on Return \(in eighths\)/i), { target: { value: '3' } });  
 fireEvent.change(screen.getByLabelText(/Fuel Price per Litre/i), { target: { value: '1.5' } });  
 fireEvent.click(screen.getByRole('button', { name: /Calculate/i }));  
  
 const okButton = screen.getByRole('button', { name: /OK/i });  
  
 // Instead of checking input values, check if new car data (VIN) is generated.  
 const previousVin = screen.getByText(/Return for vehicle \(VIN\):/i).textContent;  
 fireEvent.click(okButton);  
 await waitFor(() => {  
 const newVin = screen.getByText(/Return for vehicle \(VIN\):/i).textContent;  
 expect(newVin).not.toEqual(previousVin);  
 });  
});

Another issue I noticed with the tests are the following:

1. The console.log('Calculation Data:', { telematicsFuel, calculatedCost }) runs before setCalculatedCost(cost), causing logs to show calculatedCost: null

To fix this we need to fix this in page.tsx

**Original Application, with updated unit tests in it.**

**https://github.com/hmir992/cucumber\_frameworkV3/tree/main/qa\_assessment\_app-main%20(1)**

**Would you suggest to your team to add more, or write them in a different way?**

The tests I would add more are:

1. Test that system does not return a negative fuel cost for Discrepancy range of 5% – currently my unit tests fail because there is a bug in the system where is returning negative values for Discrepancy within 5% range
2. Test that the system returns an info message if no input for Fuel on Return (in eighths) field and Fuel Price per Litre field
3. Test that the system returns an error message if input for Fuel on Return (in eighths) field is out of range
4. Test that the system returns an error message if Fuel Price per Litre field is a large value eg 100,000
5. Test that ok button is disabled if before calculate button is pressed with valid inputs

I wrote some of these unit tests in the project but not all in the interest of time.

**What other tools or processes would you suggest?**

I would suggest that we use selenium or playwright for writing automation tests to work through the different flows of the system. Its good for functional testing of the system.

**Selenium framework**

**I have implemented a sample selenium framework to go through 1 test flow**

**Here is example cucumber File : (see full implementation if you download my framework)**

Feature: Testing Fuel On Return  
*#I expect the below the test to fail see bug 2 in my documentation, System returns Negative Fuel Cost Calculations for Discrepancy* Scenario: Testing Fuel on return 8/8 and Fuel Price per Litre cost to be 1 euro  
 Given User logins into the system  
 When User Enters "8" into the Fuel on return field  
 And User Enters "1" into the Fuel Price per Litre field  
 And User clicks the calculate button  
 Then Assert that is Estimated Fuel Cost is "0"

**How to use selenium cucumber framework:**

**Link to my framework : https://github.com/hmir992/cucumber\_frameworkV3/tree/main/POM-cucumber-selenium-framework**

Just simply download my cucumber framework and import to intelji and run the testrunner class

If you experience any issues, please check the chrome driver version and I used java 17 to create this framework

**Remember to run mvn clean install and then run the testrunner class from edit configurations**

If you experience any issues, kindly let me know, thanks

Any improvements to the framework?

Yes, the cucumber reporting could be added.

Screenshot if the tests fail was a feature I wanted to add as well.

**Playwright**

**Download Playwright file from here : https://github.com/hmir992/cucumber\_frameworkV3/tree/main/POM-cucumber-selenium-framework**

I also created the exact same test in **playwright**

**Playwright tests :**

[**https://github.com/hmir992/cucumber\_frameworkV3/tree/main/playright%20tests/tests**](https://github.com/hmir992/cucumber_frameworkV3/tree/main/playright%20tests/tests)

All I did here was to install playwright :

* 1. Open windows cmd
  2. npm init playwright@latest
  3. Follow the onscreen instructions on the cmd
  4. npx playwright test –ui
  5. Opened the test folder and opened a example test
  6. Re-did the exact same test flow as my cucumber framework
  7. Used playwright ui to debug the tests

Here is the that implementation of that test:

import { test, expect } from '@playwright/test';  
  
test('Testing Fuel on return 8/8 and Fuel Price per Litre cost to be 1 euro', async ({ page }) => {  
 await page.goto('http://localhost:3000');  
  
 // Assert we on the login page  
 const loginHeading = page.locator('h1', { hasText: 'Login' });  
 await expect(loginHeading).toBeVisible()  
  
 await page.fill('#username', 'guest');  
 await page.fill('#password', 'guestpassword');  
  
 await page.click('.button[type="submit"]');  
  
 // Assert we are on the file Fuel Return page  
 const fuelCheckerHeader = page.locator('h1', { hasText: 'Fuel Return Checker' });  
 await expect(fuelCheckerHeader).toBeVisible()  
  
  
 await page.fill('#fuelOnReturn', '8');  
 await page.fill('#fuelPrice', '1');  
  
  
 // wait for elements  
 const calculateButton = page.locator('button.button.half-button', { hasText: 'Calculate' });  
  
 await calculateButton.click();  
  
 // Wait for the elements  
 const fuelCostElement = page.locator('.form-group', { hasText: /Estimated Fuel Cost:/ });  
   
 await fuelCostElement.click();  
  
 const resultText = await fuelCostElement.innerText();  
  
 // Assert that the fuel cost is 0 as we expect  
 // the test to fail see bug 2 in my documentation, System returns Negative Fuel Cost Calculations for Discrepancy  
 expect(resultText).toContain('Estimated Fuel Cost: 0 units');  
});